PingDataSync Server™ Administration Guide

Version 7.1

PingDataSync Server™ Product Documentation

Trademarks

Disclaimer

Support

https://support.pingidentity.com/

Table of Contents

Chapter 1: Introduction

The PingDataSync Server is a high-capacity, high-reliability data synchronization and transfer pipe between source and destination topologies.

This chapter presents a general overview of the PingDataSync Server process and examples for use.

Topics include:

Overview of the PingDataSync Server

The PingDataSync Server is an efficient, Java-based server that provides high throughput, low-latency, and bidirectional real-time synchronization between two endpoint topologies consisting of PingDirectory Servers, PingDirectoryProxy Servers, PingOne, and/or Relational Database Management Systems (RDBMS) systems. The PingDataSync Server uses a dataless approach that synchronizes changes directly from the data sources in the background, so that applications can continue to update their data sources directly. The PingDataSync Server does not store any data from the endpoints themselves, thereby reducing hardware and administration costs. The server's high-availability mechanisms also makes it easy to fail over from the main PingDataSync Server to redundant instances.

Designed to run with little administrative maintenance, the PingDataSync Server includes the following features:

- High performance and availability with built-in redundancy.
- Dataless virtual architecture for a small-memory footprint and easy maintenance.
- Hassle-free setup that enables mapping attribute names, values, and DNs between endpoints. For directory server endpoints, this enables making schema and Directory Information Tree (DIT) changes without custom coding and scripting.
- Multi-vendor directory server support including the PingDirectory Server, PingDirectoryProxy Server, Nokia 8661 Directory Server, Nokia 8661 Directory Proxy Server, Oracle/Sun Directory Server Enterprise Edition, Oracle/Sun Directory Server, Oracle Unified Directory, OpenDJ, and Microsoft Active Directory, and generic LDAP directories.
- RDBMS support including Oracle Database, and Microsoft SQL Server systems.
- Proxy Server support including the PingDirectoryProxy Server and the Nokia 8661 Directory Proxy Server.
- Notification support that allows real-time change notifications to be pushed to client applications or services as they occur.

Data synchronization process

The PingDataSync Server performs point-to-point synchronization between a source endpoint and a destination endpoint. An endpoint is defined as any source or destination topology of directory or database servers.

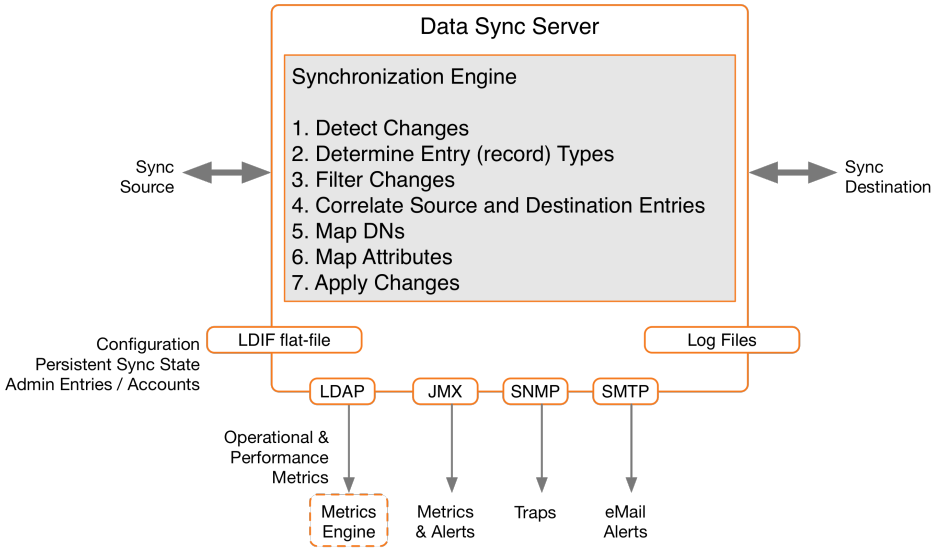
The PingDataSync Server synchronizes data in one direction or bidirectionally between endpoints. For example, in a migration phase from Sun Directory Server to a PingData PingDirectory Server, synchronization can occur in one direction from the source server to a staging server. With one-way synchronization, the source server is the authoritative endpoint for changes in the system. Bidirectional synchronization allows for parallel active installations

between the source and the destination endpoints. With bidirectional synchronization, both endpoints are authoritative for the same set of attributes or for different sets of data.

ThePingDataSync Server also contains no single point of failure, either for detecting changes or for applying changes. PingDataSync Server instances themselves are redundant. There can be multiple instances running at a time, but only the server with the highest priority is actively synchronizing changes. The stand-by servers are constantly polling the active server instance to update their persistent state. This state contains the minimum amount of information needed to begin synchronization where the primary server left off, which logically is the last processed change number for the source server. In the case of a network partition, multiple servers can synchronize simultaneously without causing problems as they each verify the full entry before making changes.

Synchronization architecture

The PingDataSync Server uses a virtualized, dataless approach that does not store directory data locally. The log files, administrator entries, configuration, sync state information are stored as flat files (LDIF format) within the system. No additional database is required.



Synchronization Architecture

Change tracking, monitoring, and logging

The PingDataSync Server tracks and manages processes and server health with the following tools:

- **Change Tracking** – Each directory instance stores a separate entry under cn=changelog for every modification made to the directory. The PingDataSync Server provides full control over the synchronization process by determining which entries are

synchronized, how they are correlated to the entries at the destination endpoint, and how they are transformed into the destination schema.
    - For the PingData PingDirectory Server or Nokia 8661 Directory Server topologies, the PingDataSync Server uses the servers's LDAP Change Log for modification detection.
    - For Oracle/Sun Directory Server, OpenDJ, Oracle Unified Directory, and generic LDAP directory topologies, the PingDataSync Server uses the server's Retro Change Log, which provides a detailed summary of each change.
    - For Active Directory, the PingDataSync Server uses the DirSync control, which polls for object attribute changes.
    - For RDBMS systems, the PingDataSync Server uses an Ping Identity Server SDK plug-in to interface with a customized RDBMS change log table. Database triggers on each table record all INSERT, UPDATE, and DELETE operations to the change log table.
- **Monitoring, Alerts, and Alarms** – The PingDataSync Server supports several industry-standard, administrative protocols for monitoring, alarms, and alerts. System alarms and gauges can be configured to determine healthy performance thresholds and the server actions taken when performance values are outside the threshold. All administrative alarms are exposed over LDAP as entries under base DN `cn=alarms`. An administrative alert framework sends warnings, errors, or other server events through log messages, email, or JMX notifications. Administrative alerts are also exposed over LDAP as entries below base DN `cn=alerts`. Typical alert events are startup or shutdown, applied configuration changes, or synchronized resources unavailable.
- **Logging** – The PingDataSync Server provides standard logs (`sync`, `access`, `error`, `failed-operations`, `config-audit.log`, `debug`). The server can also be configured for multiple active sync logs. For example, each detected change, each dropped change, each applied change, or each failed change can be logged.

Synchronization Modes

The PingDataSync Server runs as a standalone Java process with two synchronization modes: standard and notification.

Standard Synchronization

In standard synchronization mode, thePingDataSync Server polls the directory server change log for create, modify, and delete operations on any entry. The server fetches the full entries from both the source and destination endpoints, and compares them to produce the minimal set of changes required to synchronize the destination with the source.

The following shows the standard synchronization change flow between two servers. The changes are processed in parallel, which increases throughput and offsets network latency.

Notification Synchronization

In notification synchronization mode, the PingDataSync Server skips the fetch and compare phases of processing and simply notifies the destination that a change has happened and provides the details of the change. Notification mode is currently available for the PingData and Alcatel-Lucent 8661 directory and proxy servers only.

PingDataSync Server Operations

The PingDataSync Server provides seamless integration between disparate systems to transform data using attribute and DN mappings. A bulk resynchronization operation can be run verify mappings and test synchronization settings.

Real-Time Synchronization

Real-time synchronization is performed with the `realtime-sync` utility. The `realtime-sync` utility polls the source server for changes and synchronizes the destination entries immediately. Once the server determines that a change should be synchronized, it fetches the full entry from the source. It then searches for the corresponding entry in the destination endpoint using correlation rules and applies the minimum set of changes to synchronize the attributes. The server fetches and compares the full entries to make sure it does not synchronize any old data from the change log.

After a synchronization topology is configured, run `resync` to synchronize the endpoints, and then run `realtime-sync` to start global synchronization.

The `realtime-sync` tool is used for the following tasks:

- Start or stop synchronization globally or for specific sync pipes only.
- Set a start point at which synchronization should begin such as the beginning or end of the change log, at a specified change number, at a specified change sequence number, or at a specified time frame in the change log.

Data Transformations

Data transformations alter the contents of synchronized entries between the source and destination directory server to handle variances in attribute names, attribute values, or DN structures. When entries are synchronized between a source and a destination server, the contents of these entries can be changed using attribute and DN mappings, so that neither server needs be aware of the transformations.

- **Attribute Mapping** – Any attribute in the entry can be renamed to fit the schema definitions from the source endpoint to the destination endpoint. This mapping makes it possible to synchronize information stored in one directory's attribute to an attribute

with a different name in another directory server, or to construct an attribute using portions of the source attribute values.

- **DN Mapping** – Any DNs referenced in the entries can be transparently altered. This mapping makes it possible to synchronize data from a topology that uses one DIT structure to a system that uses a different DIT structure.

Bulk Resync

The `resync` tool performs a bulk comparison of entries on source topologies and destination topologies. The PingDataSync Server streams entries from the source, and either updates the corresponding destination entries or reports those that are different. The `resync` utility resides in the `/bin` folder (UNIX or LINUX) or `\bat` folder (Windows), and can be used for the following tasks:

- Verify that the two endpoints are synchronized after an initial configuration.
- Initially populate a newly configured target endpoint.
- Validate that the server is behaving as expected. The `resync` tool has a `--dry-run` option that validates that synchronization is operating properly, without updating any entries. This option also can be used to check attribute or DN mappings.
- Perform scheduled synchronization.
- Recover from a failover by resynchronizing entries that were modified since the last backup was taken.

The `resync` tool also enables control over what can be synchronized, such as:

- Include or exclude any source and destination attributes.
- Apply an LDAP filter to only sync entries created since that last time the tool ran.
- Synchronize only creations or only modifications.
- Change the logging verbosity.
- Set a limit on `resync` operations (such as 2000 operations per second) to reduce impact on endpoint servers.

The Sync Retry Mechanism

The PingDataSync Server is designed to quickly synchronize data and attempt a retry should an operation fail for any reason. The retry mechanism involves two possible retry levels, which are configurable on the Sync Pipe configuration using advanced Sync Pipe properties. For detailed information, see the *PingDataSync Server Reference Guide* for the Sync Pipe configuration parameters.

Retry involves two possible levels:

First Level Retry – If an operation fails to synchronize, the server will attempt a configurable number of retries. The total number of retry attempts is set in the `max-operation-attempts`

property on the Sync Pipe. The property indicates how many times a worker thread should retry the operation before putting the operation into the second level of retry, or failing the operation altogether.

Second Level Retry – Once the `max-operation-attempts` property has been exceeded, the retry is sent to the second level, called the delayed-retry queue. The delayed-retry queue uses two advanced Sync Pipe properties to determine the number of times an operation should be retried in the background after a specified delay.

Operations that make it to this level will be retried after the `failed-op-background-retry-delay` property (default: 1 minute). Next, the PingDataSync Server checks the `max-failed-op-background-retries` property to determine the number of times a failed operation should be retried in the background. By default, this property is set to `0`, which indicates that no background retry should be attempted, and that the operation should be logged as failed.

> **Note**
> Background operations can hold up processing other changes, since the PingDataSync Server will only process up to the next 5000 changes while waiting for a retried operation to complete.

Retry can be controlled by the custom endpoint based on the type of error exception. When throwing an exception, the endpoint code can signal that a change should be aborted, retried a limited number of times, or retried an unlimited number of times. Some errors, such as endpoint server down, should be retried indefinitely.

If the `max-failed-op-background-retries` property has been exceeded, the retry is logged as a failure and appears in the `sync` and the `sync-failed-ops` logs.

Configuration Components

The PingDataSync Server supports the following configuration parameters that determine how synchronization takes place between directories or databases:

Sync Pipe – Defines a single synchronization path between the source and destination topologies. Every Sync Pipe has one or more Sync Classes that control how and what is synchronized. Multiple Sync Pipes can run in a single server instance.

Sync Source – Defines the directory topology that is the source of the data to be synchronized. A Sync Source can reference one or more supported external servers.

Sync Destination – Defines the topology of directory servers where changes detected at the Sync Source are applied. A Sync Destination can reference one or more external servers.

External Server – Defines a single server in a topology of identical, replicated servers to be synchronized. A single external server configuration object can be referenced by multiple Sync Sources and Sync Destinations.

Sync Class – Defines the operation types and attributes that are synchronized, how attributes and DNs are mapped, and how source and destination entries are correlated. A source entry is in one Sync Class and is determined by a base DN and LDAP filters. A Sync Class can reference zero or more Attribute Maps and DN Maps, respectively. Within a Sync Pipe, a Sync Class is defined for each type of entry that needs to be treated differently. For example, entries that

define attribute mappings, or entries that should not be synchronized at all. A Sync Pipe must have at least one Sync Class but can refer to multiple Sync Class objects.

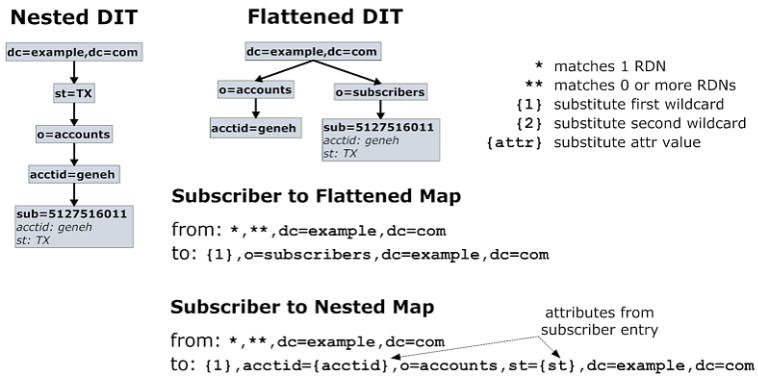
DN Map – Defines mappings for use when destination DNs differ from source DNs. These mappings allow the use of wild cards for DN transformations. A single wild card ("*") matches a single RDN component and can be used any number of times. The double wild card ("**") matches zero or more RDN components and can be used only once. The wild card values can be used in the `to-dn-pattern` attribute using `{1}` and their original index position in the pattern, or `{attr}` to match an attribute value. For example:

```
**,dc=myexample,dc=com->{1},o=example
```

Regular expressions and attributes from the user entry can also be used. For example, the following mapping constructs a value for the `uid` attribute, which is the RDN, out of the initials (first letter of `givenname` and `sn`) and the employee ID (`eid` attribute).

```
uid={givenname:/^(.)(.*)/$1/s}{sn:/^(.)(.*)/$1/s}{eid},{2},o=example
```

The following illustrates a how a nested DIT can be mapped to a flattened structure.



Nested DIT Mapping

Attribute Map and Attribute Mappings – Defines a mapping for use when destination attributes differ from source attributes. A Sync Class can reference multiple attribute maps. Multiple Sync Classes can share the same attribute map. There are three types of attribute mappings:

- Direct Mapping – Attributes are directly mapped to another attribute: For example:

  ```
  employeenumber->employeeid
  ```

- Constructed Mapping – Destination attribute values are derived from source attribute values and static text. For example:

  ```
  {givenname}.{sn}@example.com->mail
  ```

- DN Mapping – Attributes are mapped for DN attributes. The same DN mappings that map entry DNs can be referenced. For example:

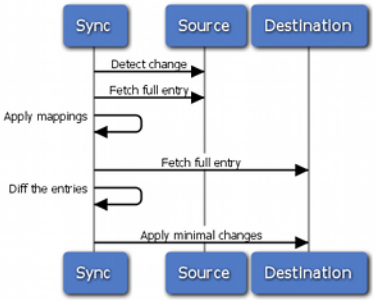
  ```
  uid=jdoe,ou=People,dc=example,dc=com.
  ```

Sync Flow Examples

The PingDataSync Server processes changes by fetching the most up-to-date, full entries from both sides and then compares them. This process flow is called standard synchronization mode. The processing flow differs depending on the type of PingDataSync Serverchange (`ADD`, `MODIFY`, `DELETE`, `MODDN`) that is requested. The following examples show the control flow diagrams for the sync operations, especially for those cases when a `MODIFY` or a `DELETE` operation is dropped. The sync log records all completed and failed operations.

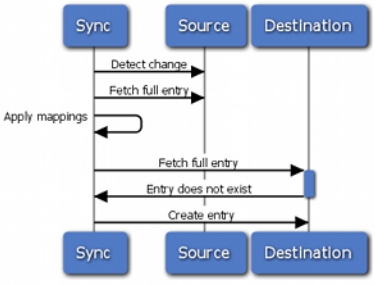
Modify Operation Example

1. Detect change from the change log table on the source.
2. Fetch the entry or table rows from affected tables on the source.
3. Perform any mappings and compute the equivalent destination entry by constructing an equivalent LDAP entry or equivalent table row.
4. Fetch the entry or table rows from affected tables on the destination.
5. Diff the computed destination entry and actual destination entry.
6. Apply the changes to the destination.



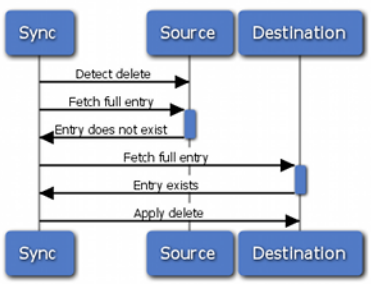
Add Operation Example

1. Detect change from the change log table on the source.
2. Fetch the entry or table rows from affected tables on the source.
3. Perform any mappings and compute the equivalent destination entry by constructing an equivalent LDAP entry or equivalent table row.
4. Fetch the entry or table rows from affected tables on the destination.
5. The entry or table row does not exist on the destination.
6. Create the entry or table row.



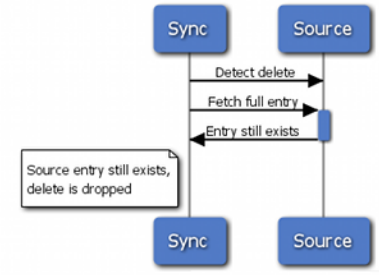
Delete Operation Example

1. Detect delete from the change log table on the source.
2. Fetch the entry or table rows from affected tables on the source.
3. Perform any mappings and compute the equivalent destination entry by constructing an equivalent LDAP entry or equivalent table row.
4. Fetch the entry or table rows from affected tables on the destination.
5. The entry or table row exists on the destination.
6. Apply the delete on the destination.



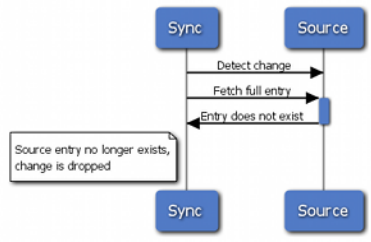
Delete After Source Entry is Re-Added

1. Detect delete from the change log table on the source.
2. Fetch the entry or table rows from affected tables on the source.
3. The entry or table row exists on the source.
4. Delete request is dropped.



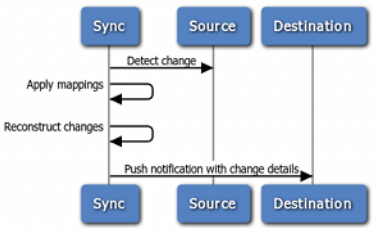
Standard Modify After Source Entry is Deleted

1. Detect change from the change log table on the source.
2. Fetch the entry or table rows from affected tables on the source.
3. The entry does not exist.
4. Change request is dropped because the source entry no longer exists.



Notification Add, Modify, ModifyDN, and Delete

1. Detect change from the change log table on the source.
2. Perform any mappings and compute the equivalent destination entry by constructing an equivalent LDAP entry or equivalent table row.
3. Reconstruct changed entries.
4. Push notification with change details to the destination.



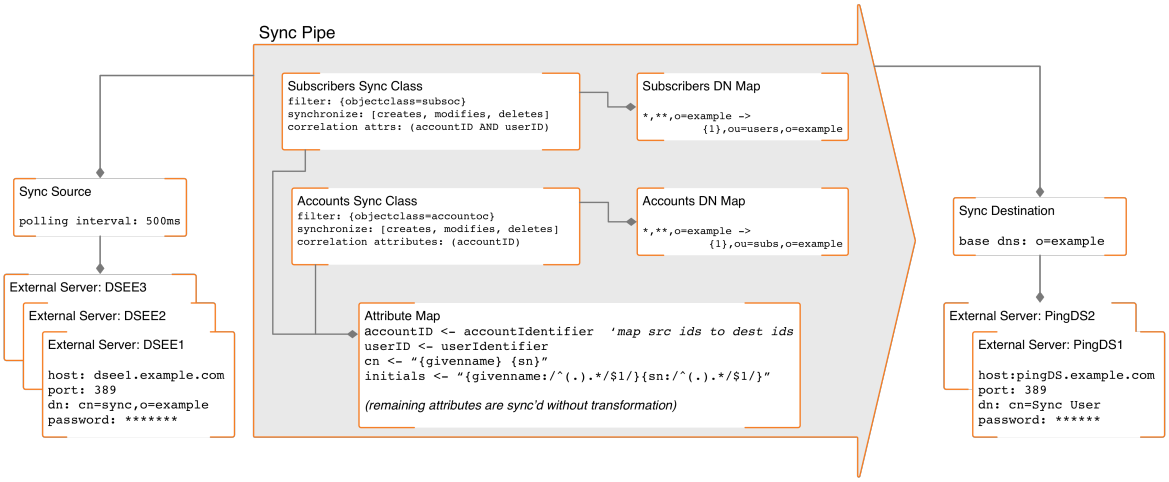
Sample Synchronization

The following is a synchronization migration example from a Sun Directory Server Enterprise Edition (DSEE) topology to the PingData PingDirectory Server topology, including a change in the DIT structure to a flattened directory structure. The Sync Pipe connects the Sun Directory Server topology as the Sync Source and the PingDirectory Server topology as the Sync Destination. Each endpoint is defined with three external servers in their respective topology. The Sync Pipe destination has its base DN set to `o=example`, which is used when performing LDAP searches for entries.

Two Sync Classes are defined: one for Subscribers, and one for Accounts. Each Sync Class uses a single "Sun DS to PingData Attribute Map" that has four attribute mappings defined. Each Sync Class also defines its own DN maps. For example, the Accounts Sync Class uses a DN map, called PingData Account Map, that is used to flatten a hierarchical DIT, so that the Account entries appear directly under `ou=accounts` as follows:

```
*,**,o=example -> {1},ou=accounts,o=example
```

With this mapping, if an entry DN has `uid=jsmith,ou=people,o=example`, then "`*`" matches `uid=jsmith`, "`**`" matches `ou=people`, and `{1}` matches `uid=jsmith`. Therefore, `uid=jsmith,ou=people,o=example` gets mapped to `uid=jsmith,ou=accounts,o=example`. A similar map is configured for the Subscribers Sync Class.



A Sample Synchronization Topology Configuration

Chapter 2: Installing the PingDataSync Server

This section describes how to install and run the PingDataSync Server. It includes pre-installation requirements and considerations.

Topics include:

Supported Platforms

The PingDataSync Server is a pure Java application. It is intended to run within the Java Virtual Machine on any Java Standard Edition (SE) or Enterprise Edition (EE) certified platform. For the list of supported platforms and Java versions, access the Ping Identity Customer Support Center portal or contact a PingData authorized support provider.

Note
It is highly recommended that a Network Time Protocol (NTP) system be in place so that multi-server environments are synchronized and timestamps are accurate.

Install the JDK

The Java 64-bit JDK is required on the server. Even if Java is already installed, create a separate Java installation for use by the server to ensure that updates to the system- wide Java installation do not inadvertently impact the installation.

Optimize the Linux Operating System

Configure the Linux filesystem by making the following changes.

Note
The server explicitly overrides environment variables like `PATH`, `LD_LIBRARY_PATH`, and `LD_PRELOAD` to ensure that settings used to start the server do not inadvertently impact its behavior. If these variables must be edited, set values by editing the `set_environment_vars` function of the `lib/_script-util.sh` script. Stop and restart the server for the change to take effect.

Set the file descriptor limit

The server allows for an unlimited number of connections by default, but is restricted by the file descriptor limit on the operating system. If needed, increase the file descriptor limit on the operating system with the following procedure.

Note
If the operating system relies on `systemd`, refer to the Linux operating system documentation for instructions on setting the file descriptor limit.

1. Display the current hard limit of the system. The hard limit is the maximum server limit that can be set without tuning the kernel parameters in the `proc` filesystem.

   ```
   ulimit -aH
   ```

2. Edit the `/etc/sysctl.conf` file. If the `fs.file-max` property is defined in the file, make sure its value is set to at least 65535. If the line does not exist, add the following to the end of the file:

   ```
   fs.file-max = 65535
   ```

3. Edit the `/etc/security/limits.conf` file. If the file has lines that set the soft and hard limits for the number of file descriptors, make sure the values are set to `65535`. If the lines are not present, add the following lines to the end of the file (before `#End of file`). Insert a tab between the columns.

```
* soft nofile 65535
* hard nofile 65535
```

4. Reboot the system, and then use the `ulimit` command to verify that the file descriptor limit is set to `65535` with the following command:

```
ulimit -n
```

Once the operating system limit is set, the number of file descriptors that the server will use can be configured by either using a `NUM_FILE_DESCRIPTORS` environment variable, or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_DESCRIPTORS=12345`. If these are not set, the default of 65535 is used. This is strictly optional if wanting to ensure that the server shuts down safely prior to reaching the file descriptor limit.

Note
For RedHat 7 or later, modify `the 20-nproc.conf` file to set both the open files and max user processes limits:

```
/etc/security/limits.d/20-nproc.conf

Add or edit the following lines if they do not already exist:
*     soft    nproc     65536
*     soft    nofile    65536
*     hard    nproc     65536
*     hard    nofile    65536
root  soft    nproc     unlimited
```

Set the filesystem flushes

Linux systems running the ext3 filesystem only flush data to disk every five seconds. If the server is on a Linux system, edit the mount options to include the following:

```
commit=1
```

This variable changes the flush frequency from five seconds to one. Also, set the flush frequency in the `/etc/fstab` file to make sure the configuration remains after reboot.

Install sysstat and pstack on Red Hat

The server troubleshooting tool `collect-support-data` relies on the `iostat`, `mpstat`, and `pstack` utilities to collect monitoring, performance statistics, and stack trace information on the server's processes. For Red Hat systems, make sure that these packages are installed, for example:

```
$ sudo yum install sysstat gdb dstat -y
```

Install the dstat utility

The `dstat` utility is used by the `collect-support-data` tool.

Disable filesystem swapping

Any performance tuning services, like `tuned`, should be be disabled. If performance tuning is required, `vm.swappiness` can be set by cloning the existing performance profile then adding `vm.swappiness = 0` to the new profile's `tuned.conf` file in `/usr/lib/tuned/profile-name/tuned.conf`. The updated profile is then selected by running `tuned-adm profile customized_profile`.

Manage system entropy

Entropy is used to calculate random data that is used by the system in cryptographic operations. Some environments with low entropy may have intermittent performance issues with SSL-based communication. This is more typical on virtual machines, but can occur in physical instances as well. Monitor the `kernel.random.entropy_avail` in `sysctl` value for best results.

If necessary, update `$JAVA_HOME/jre/lib/security/java.security` to use `file:/dev/./urandom` for the `securerandom.source` property.

Enable the server to listen on privileged ports

Linux provides 'capabilities' used to grant specific commands the ability to do things that are normally only allowed for a root account. Instead of granting the ability to a specific user, capabilities are granted to a specific command. It may be convenient to enable the server to listen on privileged ports while running as a non-root user.

The `setcap` command is used to assign capabilities to an application. The `cap_net_bind_service` capability enables a service to bind a socket to privileged ports (port numbers less than 1024). If Java is installed in `/ds/java` (and the Java command to run the server is `/ds/java/bin/java`), the Java binary can be granted the `cap_net_bind_service` capability with the following command:

```
$ sudo setcap cap_net_bind_service=+eip /ds/java/bin/java
```

The java binary needs an additional shared library (`libjli.so`) as part of the Java installation. More strict limitations are imposed on where the operating system will look for shared libraries to load for commands that have capabilities assigned. So it is also necessary to tell the operating system where to look for this library. This can be done by creating the file `/etc/ld.so.conf.d/libjli.conf` with the path to the directory that contains the `libjli.so` file. For example, if the Java installation is in `/ds/java`, the contents of that file should be:

```
/ds/java/lib/amd64/jli
```

Run the following command for the change to take effect:

```
$ sudo ldconfig -v
```

Ping license keys

License keys are required to install all Ping products. Obtain licenses through Salesforce or from https://www.pingidentity.com/en/account/request-license-key.html.

- A license is always required for setting up a new single server instance and can be used site-wide for all servers in an environment. When cloning a server instance with a valid license, no new license is needed.
- A new license must be obtained when updating a server to a new major version, for example from 6.2 to 7.0. Licenses with no expiration date are valid until the server is upgraded to the next major version. A prompt for a new license is displayed during the update process.
- A license may expire on particular date. If a license does expire, obtain a new license and install it using `dsconfig` or the Administrative Console. The server will provide a notification as the expiration date approaches. License details are available using the server's `status` tool.

When installing the server, specify the license key file in one of the following ways:

- Copy the license key file to the server root directory before running setup. The interactive `setup` tool will discover the file and not require input. If the file is not in the server root, the setup tool will prompt for its location.
- If the license key is not in the server root directory, specify the `--licenseKeyFile` option for non-interactive setup, and the path to the file.

Install the PingDataSync Server

Use the `setup` tool to install the server. The server needs to be started and stopped by the user who installed it.

Note
A Windows installation requires that the Visual Studio 2010 runtime patch be installed prior to running the `setup` command.

1. Log in as a user, other than root.
2. Obtain the latest zip release bundle from Ping Identity and unpack it in a directory owned by this user.

```
$ unzip PingDataSync-<version>.zip
```

3. Change to the server root directory.

```
$ cd PingDataSync
```

4. Run the `setup` command.

```
$ ./setup
```

5. Type **yes** to accept the End-User License Agreement and press **Enter** to continue.
6. If adding this server to an existing PingDataSync Server topology, type **yes**, or press **Enter** to accept the default (no).
7. Enter the fully qualified host name or IP address of the local host.
8. Create the initial root user DN for the PingDataSync Server, or press **Enter** to accept the default (cn=Directory Manager).
9. Enter and confirm a password for this account.
10. Press **Enter** to enable server services and the Administrative Console.
11. Enter the port on which the PingDataSync Server will accept connections from HTTPS clients, or press **Enter** to accept the default.
12. Enter the port on which the PingDataSync Server will accept connections from LDAP clients, or press **Enter** to accept the default.
13. Press **Enter** to enable LDAPS, or enter **no**.
14. Press **Enter** to enable StartTLS, or enter **no**.
15. Select the certificate option for this server.
16. Choose the desired encryption for the directory data, backups, and log files from the choices provided:
    - Encrypt data with a key generated from an interactively provided passphrase. Using a passphrase (obtained interactively or read from a file) is the recommended approach for new deployments, and you should use the same encryption passphrase when setting up each server in the topology.
    - Encrypt data with a key generated from a passphrase read from a file.
    - Encrypt data with a randomly generated key. This option is primarily intended for testing purposes, especially when only testing with a single instance, or if you intend to import the resulting encryption settings definition into other instances in the topology.
    - Encrypt data with an imported encryption settings definition. This option is recommended if you are adding a new instance to an existing topology that has older server instances with data encryption enabled.
    - Do not encrypt server data.
17. Choose the option for the amount of memory that should be allocated to the server.
18. To start the server when the configuration is complete, press **Enter** for (yes).
19. A Setup Summary is displayed. choose the option to setup the server with the listed parameters, change the parameters, or cancel the setup.

After the server configuration is complete, the `create-sync-pipe-config` tool can be run to configure the synchronization environment.

The PingDataSync Server Administrative Console enables browser-based server management, the `dsconfig` tool enables command line management, and the Configuration API enables management by third-party interfaces.

Log into the Administrative Console

After the server is installed, access the Administrative Console, `https://<host>/console/login`, to verify the configuration and manage the server. The root user DN or the common name of a root user DN is required to log into the Administrative Console. For example, if the DN created when the server was installed is `cn=Directory Manager`, `directory manager` can be used to log into the Administrative Console.

If the Administrative Console needs to run in an external container, such as Tomcat, a separate package can be installed according to that container's documentation. Contact Ping Identity Customer Support for the package location and instructions.

Server folders and files

After the distribution file is unzipped, the following folders and command-line utilities are available:

Directories/Files/Tools	Description
ldif	Stores any LDIF files that you may have created or imported.
import-tmp	Stores temporary imported items.
classes	Stores any external classes for server extensions.
bak	Stores the physical backup files used with the backup command-line tool.
velocity	Stores Velocity templates that define the server's application pages.
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems.
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
ping_logo.png	The image file for the Ping Identity logo.
setup.bat, and setup	The setup tool for UNIX/Linux systems and Windows systems.
revert-update.bat, and revert-update	The revert-update tool for UNIX/Linux systems and Windows systems.
README	README file that describes the steps to set up and start the server.
License.txt	Licensing agreement for the product.
legal-notices	Stores any legal notices for dependent software used with the product.
docs	Provides the release notes, Configuration Reference (HTML), API Reference, and all other product documentation.
metrics	Stores the metrics that can be gathered for this server and surfaced in the PingDataMetrics Server.

Directories/Files/Tools	Description
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
lib	Stores any scripts, jar files, and library files needed for the server and its extensions.
collector	Used by the server to make monitored statistics available to the PingDataMetrics Server.
locks	Stores any lock files in the backends.
tmp	Stores temporary files.
resource	Stores the MIB files for SNMP and can include ldif files, make-ldif templates, schema files, dsconfig batch files, and other items for configuring or managing the server.
config	Stores the configuration files for the backends (admin, config) as well as the directories for messages, schema, tools, and updates.
logs	Stores log files.
AD-Password-Sync-Agent.zip	The Active Directory Sync Agent package.

Start and stop the server

To start the PingDataSync Server, run the `bin/start-sync-server` command on UNIX or Linux systems (the `bat` folder on Microsoft Windows systems).

Start the Server as a Background Process

Navigate to the server root directory, and run the following command:

```
$ bin/start-server
```

For Windows systems:

```
$ bat/start-server
```

Start the server at boot time

By default, the server does not start automatically when the system is booted. To configure the server to start automatically, use the `create-rc-script` tool to create a run control script as follows:

1. Create the startup script. In this example `ds` is the user.

   ```
   $ bin/create-rc-script \
     --outputFile Ping-Identity-Sync.sh \
     --userName ds
   ```

2. Log in as root, move the generated `Ping-Identity-Sync.sh` script into the `/etc/init.d` directory, and create symlinks to it from the `/etc/rc3.d` (starting with an "S" to start

the server) and `/etc/rc0.d` directory (starting with a "K" to stop the server).

```
# mv Ping-Identity-Sync.sh /etc/init.d/
# ln -s /etc/init.d/Ping-Identity-Sync.sh /etc/rc3.d/S50-Ping-Identity-Sync.sh
# ln -s /etc/init.d/Ping-Identity-Sync.sh /etc/rc0.d/K50-Ping-Identity-Sync.sh
```

Stop the Server

If the PingDataSync Server has been configured to use a large amount of memory, it can take several seconds for the operating system to fully release the memory. Trying to start the server too quickly after shut down can fail because the system does not yet have enough free memory. On UNIX systems, run the `vmstat` command and watch the values in the "free" column increase until all memory held by the PingDataSync Server is released back to the system.

A configuration option can also be set that specifies the maximum shutdown time a process can take.

To stop the server, navigate to the server root directory and run the following command:

```
$ bin/stop-server
```

Restart the server

Restart the server using the `bin/stop-server` command with the `--restart` or `-R` option. Running the command is equivalent to shutting down the server, exiting the JVM session, and then starting up again.

```
$ bin/stop-server --restart
```

Run the server as a Microsoft Windows service

The server can run as a Windows service on Windows Server 2012 R2 and Windows Server 2016. This enables log out of a machine without the server being stopped.

Register the service

Perform the following steps to register the server as a service:

1. Stop the server with `bin/stop-server`. A server cannot be registered while it is running.
2. Register the server as a service. From a Windows command prompt, run `bat/register-windows-service.bat`.
3. After a server is registered, start the server from the Windows Services Control Panel or with the `bat/start-server.bat` command.

Note
Command-line arguments for the `start-server.bat` and `stop-server.bat` scripts are not supported while the server is registered to run as a Windows service. Using a task to stop the server is also not supported.

Run multiple service instances

Only one instance of a particular service can run at one time. Services are distinguished by the `wrapper.name` property in the `<server-root>/config/wrapper-product.conf` file. To run additional service instances, change the `wrapper.name` property on each additional instance. Descriptions of the services can also be added or changed in the `wrapper-product.conf` file.

Deregister and uninstall

While a server is registered as a service, it cannot run as a non-service process or be uninstalled. Use the `bat/deregister-windows-service.bat` file to remove the service from the Windows registry. The server can then be uninstalled with the `uninstall.bat` script.

Log files

The log files are stored in `<server-root>/logs`, and filenames start with `windows-service-wrapper`. They are configured to rotate each time the wrapper starts or due to file size. Only the last three log files are retained. These configurations can be changed in the `<server-root>/config/wrapper.conf` file.

Uninstall the server

Use the `uninstall` command-line utility to uninstall the server using either interactive or non-interactive modes. Interactive mode provides options, progress, and a list of the files and directories that must be manually deleted if necessary.

Non-interactive mode, invoked with the `--no-prompt` option, suppresses progress information, except for fatal errors. All options for the `uninstall` command are listed with the `--help` option.

The `uninstall` command must be run as either the root user or the user account that installed the server.

Perform the following steps to uninstall in interactive mode:

1. Navigate to the server root directory.

   ```
   $ cd PingData<server>
   ```

2. Start the uninstall command:

   ```
   $ ./uninstall
   ```

3. Select the components to be removed, or press **Enter** to remove all components.
4. If the server is running, press **Enter** to shutdown the server before continuing.

5. Manually remove any remaining files or directories, if required.

Update servers in a topology

An update to the current release includes significant changes, and the introduction of a topology registry, which will store information previously stored in the admin backend (server instances, instance and secret keys, server groups, and administrator user accounts). For the admin backend to be migrated, the `update` tool must be provided LDAP authentication options to the peer servers of the server being updated.

The LDAP connection security options requested (either plain, TLS, StartTLS, or SASL) must be configured on every server in the topology. The LDAP credentials must be present on every server in the topology, and must have permissions to read from the admin backend and the config backend of every server in thetopology. For example, a root DN user with the `inherit-default-privileges` set to true (such as the `cn=Directory Manager` user) that exists on every server can be used.

After enabling or fixing the configuration of the LDAP connection handler(s) to support the desired connection security mechanism on each server, run the following `dsframework` command on the server being updated so that its admin backend has the most up-to-date information:

```
$ bin/dsframework set-server-properties \
  --serverID serverID \
  --set ldapport:port \
  --set ldapsport:port \
  --set startTLSEnabled:true
```

The `update` tool will verify that the following conditions are satisfied on every server in the topology before allowing the update:

- When the first server is being updated, all other servers in the topology must be online. When updating additional servers, all topology information will be obtained from one of the servers that has already been updated. The `update` tool will connect to the peer servers of the server being updated to obtain the necessary information to populate the topology registry. The provided LDAP credentials must have read permissions to the config and admin backends of the peer servers.
- The instance name is set on every server, and is unique across all servers in the topology. The instance name is a server's identifier in the topology. After all servers in the topology have been updated, each server will be uniquely identified by its instance name. Once set, the name cannot be changed. If needed, the following command can be used to set the instance name of a server prior to the update:

  ```
  $ bin/dsconfig set-global-configuration-prop \
    --set instance-name:uniqueName
  ```

- The cluster-wide configuration is synchronized on all servers in the topology. Older versions have some topology configuration under `cn=cluster,cn=config` (JSON

attribute and field constraints). These items did not support mirrored cluster-wide configuration data. An update should avoid custom configuration changes on a server being overwritten with the configuration on the mirrored subtree master. To synchronize the cluster-wide configuration data across all servers in the topology, run the `config-diff` tool on each pair of servers to determine the differences, and use `dsconfig` to update each instance using the `config-diff` output. For example:

```
$ bin/config-diff --sourceHost hostName \
  --sourcePort port \
  --sourceBindDN bindDN \
  --sourceBindPassword password \
  --targetHost hostName \
  --targetPort port \
  --targetBindDN bindDN \
  --targetBindPassword password
```

If any of these conditions are not satisfied, the `update` tool will list all of the errors encountered for each server, and provide instructions on how to fix them.

Update the server

This procedure assumes that an existing version of the server is stored at `PingData-server-old`. Make sure a complete, readable backup of the existing system is available before upgrading the server. Also, make sure there is a clear backout plan and schedule.

1. Download the latest version of the server software and unzip the file. For this example, the new server is located in the `PingData-server-new` directory.
2. Use the `update` tool of the newly unzipped build to update the server. Make sure to specify the server instance that is being upgrading with the `--serverRoot` option. The server must be stopped for the update to be applied.

Reverting an Update

If necessary, a server can be reverted to the previous version using the `revert-update` tool. The tool accesses a log of file actions taken by the `update` tool to put the filesystem back to its prior state. If multiple updates have been run, the `revert-update` tool can be used multiple times to revert to each prior update sequentially. For example, the `revert-update` command can be run to revert to the server's previous state, then run again to return to its original state. The server is stopped during the revert-update process.

Note
Reverting an update is not supported for upgrades to version 7.0, due to the topology backend changes.

Use the `revert-update` tool in the server root directory revert back to the most recent version of the server:

```
$ PingData-server-old/revert-update
```

Install a failover server

Ping Identity supports redundant failover servers that automatically become active when the primary server is not available. Multiple servers can be present in the topology in a configurable prioritized order.

Before installing a failover server, have a primary server already installed and configured. When installing the redundant server, the installer will copy the first server's configuration.

The primary and secondary server configuration remain identical. Both servers should be registered to the `all servers` group and all `dsconfig` changes need to be applied to the server group `all servers`.

Note

If the primary server has extensions defined, they should also be installed on any cloned or redundant servers. If extensions are missing from a secondary server, the following message is displayed during the installation:

```
Extension class <com.server.directory.sync.MissingSyncExtension> was not
found. Run manage-extension --install to install your extensions. Re-run
setup to continue.
```

To remove a failover server, use the `uninstall` command.

1. Unpack the Ping Identity server zip build. Name the unpacked directory something other than the first server instance directory.

   ```
   $ unzip PingData<server>-<version>.zip -d <server2>
   ```

2. Navigate to the server root directory.
3. Use the `setup` tool in interactive mode in Install the Server, or in non-interactive mode as follows:

   ```
   $ ./setup --localHostName <server2>.example.com --ldapPort 7389 \
     --masterHostName <server1>.example.com --masterPort 8389 \
     --masterUseNoSecurity \
     --acceptLicense \
     --rootUserPassword password \
     --no-prompt
   ```

   The secondary server is now ready to take over as a primary server in the event of a failover. No `realtime-sync` invocations are needed for this server.

4. Verify the configuration by using the `bin/status` tool. Each server instance is given a priority index. The server with the lowest priority index number has the highest priority.

   ```
   $ bin/status --bindPassword secret

   ...(status output)...
                                --- Sync Topology ---
   Host:Port                                :Status       :Priority
   -----------------------------------------:-------------:---------------
   <server>.example.com:389 (this server) : Active      : 1
   <server>.example.com:389               : Unavailable : 2
   ```

5. Obtain the name of a particular server, run the `dsconfig` tool with the `list-external-servers` option.

```
$ bin/dsconfig list-external-servers
```

6. To change the priority index of the server, use the `bin/dsconfig` tool:

```
$ bin/dsconfig set-external-server-prop \
  --server-name <server2>.example.com:389 \
  --set <server>-priority-index:1
```

Administrative accounts

Users that authenticate to the Configuration API or the Administrative Console are stored in `cn=Root DNs,cn=config`. The `setup` tool automatically creates one administrative account when performing an installation. Accounts can be added or changed with the `dsconfig` tool.

Change the administrative password

Root users are governed by the Root Password Policy and by default, their passwords never expire. However, if a root user's password must be changed, use the `ldappasswordmodify` tool.

1. Open a text editor and create a text file containing the new password. In this example, name the file rootuser.txt.

```
$ echo password > rootuser.txt
```

2. Use ldappasswordmodify to change the root user’s password.

```
$ bin/ldappasswordmodify --port 1389 --bindDN "cn=Directory Manager" \
--bindPassword secret --newPasswordFile rootuser.txt
```

3. Remove the text file.

```
$ rm rootuser.txt
```

Chapter 3: Configuring the PingDataSync Server

The PingDataSync Server provides a suite of tools to configure a single server instance or server groups. All configuration changes to the server are recorded in the `config-audit.log`, and configuration is stored as a flat file (LDIF format) in the `cn=config` branch. Before configuring the PingDataSync Server, review Sync Configuration Components.

Topics include:

Configuration checklist

Prior to any deployment, determine the configuration parameters necessary for the Synchronization topology. Gather the following:

External servers

External Server Type – Determine the type of external servers included in the synchronization topology. See Overview of the PingDataSync Server for a list of supported servers.

LDAP Connection Settings – Determine the host, port, bind DN, and bind password for each external server instance(s) included in the synchronization topology.

Security and Authentication Settings – Determine the secure connection types for each external server (SSL or StartTLS). Determine authentication methods for external servers such as simple, or external (SASL mechanisms). If synchronizing passwords, encoded or especially for clear-text, the connection should be secure. If synchronizing to or from a Microsoft Active Directory system, establish an SSL or StartTLS connection to the PingDataSync Server. Password encryption should also be enabled for synchronization from Active Directory, or when synchronizing clear-text passwords.

Sync Pipes

A Sync Pipe defines a single synchronization path between the source and destination targets. One Sync Pipe is needed for each point-to-point synchronization path defined for a topology.

Sync Source – Determine which external server is the Sync Source for the synchronization topology. A prioritized list of external servers can be defined for failover purposes.

Sync Destination – Determine which external server is the Sync Destination for the synchronization topology. A prioritized list of external servers can be defined for failover purposes.

Sync Classes

A Sync Class defines how attributes and DNs are mapped and how Source and Destination entries are correlated. For each Sync Pipe defined, define one or more Sync Classes with the following information:

Evaluation Order – If defining more than one Sync Class, the evaluation order of each Sync Class must be determined with the `evaluation-order-index` property. If there is an overlap between criteria used to identify a Sync Class, the Sync Class with the most specific criteria is used first.

Base DNs – Determine which base DNs contain entries needed in the Sync Class.

Include Filters – Determine the filters to be used to search for entries in the Sync Source.

Synchronized Entry Operations – Determine the types of operations that should be synchronized: creates, modifications, and/or deletes.

DNs – Determine the differences between the DNs from the Sync Source topology to the Sync Destination topology. Are there structural differences in each Directory Information Tree (DIT)? For example, does the Sync Source use a nested DIT and the Sync Destination use a flattened DIT?

Destination Correlation Attributes – Determine the correlation attributes that are used to associate a source entry to a destination entry during the synchronization process. During the configuration setup, one or more comma-separated lists of destination correlation attributes are defined and used to search for corresponding source entries. The PingDataSync Server maps all attributes in a detected change from source to destination attributes using the attribute maps defined in the Sync Class.

The correlation attributes are flexible enough so that several destination searches with different combinations of attributes can be performed until an entry matches. For LDAP server endpoints, use the distinguished name (DN) to correlate entries. For example, specify the attribute lists `dn,uid`, `uid,employeeNumber` and `cn,employeeNumber` to correlate entries in LDAP deployments. The PingDataSync Server will search for a corresponding entry that has the same `dn` and `uid` values. If the search fails, it then searches for `uid` and `employeeNumber`. Again if the search fails, it searches for `cn` and `employeeNumber`. If none of these searches are successful, the synchronization change would be aborted and a message logged.

To prevent incorrect matches, the most restrictive attribute lists (those that will never match the wrong entry) should be first in the list, followed by less restrictive attribute lists, which will be used when the earlier lists fail. For LDAP-to-LDAP deployments, use the DN with a combination of other unique identifiers in the entry to guarantee correlation. For other non-LDAP deployments, determine the attributes that can be synchronized across the network.

Attributes – Determine the differences between the attributes from the Sync Source to the Sync Destination, including the following:

- **Attribute Mappings** – How are attributes mapped from the Sync Source to the Sync Destination? Are they mapped directly, mapped based on attribute values, or mapped based on attributes that store DN values?
- **Automatically Mapped Source Attributes** – Are there attributes that can be automatically synchronized with the same name at the Sync Source to Sync Destination? For example, can direct mappings be set for `cn`, `uid`, `telephoneNumber`, or for all attributes?
- **Non-Auto Mapped Source Attributes** – Are there attributes that should not be automatically mapped? For example, the Sync Source may have an attribute, `employee`, while the Sync Destination may have a corresponding attribute, `employeeNumber`. If an attribute is not automatically mapped, a map must be provided if it is to be synchronized.
- **Conditional Value Mapping** – Should some mappings only be applied some of the time as a function of the source attributes? Conditional value mappings can be used with the

`conditional-value-pattern` property, which conditionalizes the attribute mapping based on the subtype of the entry, or on the value of the attribute. For example, this might apply if the `adminName` attribute on the destination should be a copy of the `name` attribute on the source, but only if the `isAdmin` attribute on the source is set to `true`. Conditional mappings are multi-valued. Each value is evaluated until one is matched (the condition is `true`). If none of the conditional mappings are matched, the ordinary mappings is used. If there is not an ordinary mapping, the attribute will not be created on the destination.

The Sync User account

The PingDataSync Server creates a Sync User account DN on each external server. The account (by default, `cn=Sync User`) is used exclusively by the PingDataSync Server to communicate with external servers. The entry is important in that it contains the credentials (DN and password) used by the PingDataSync Server to access the source and target servers. The Sync User account resides in different entries depending on the targeted system:

- For the Ping Identity PingDirectory Server, Ping Identity PingDirectoryProxy Server, Nokia 8661 Directory Server, Nokia 8661 Directory Proxy Server, the Sync User account resides in the configuration entry (`cn=Sync User, cn=Root DNs,cn=config`).
- For Sun Directory Server, Sun DSEE, OpenDJ, Oracle Unified Directory, and generic LDAP directory topologies, the Sync User account resides under the base DN in the `userRoot` backend (`cn=Sync User,dc=example,dc=com`). The Sync User account should not reside in the `cn=config` branch for Sun Directory Server and DSEE machines.
- For Microsoft Active Directory servers, the Sync User account resides in the Users container (`cn=Sync User,cn=Users,DC=adsync,DC=unboundid,DC=com`).
- For Oracle and Microsoft SQL Servers, the Sync User account is a login account (`SyncUser`) with the sufficient privileges to access the tables to be synchronized.

In most cases, modifications to this account are rare. Make sure that the entry is not synchronized by setting up an optional Sync Class if the account resides in the `userRoot` backend (Sun Directory Server or Sun DSEE) or Users container (Microsoft Active Directory). For example, a Sync Class can be configured to have all CREATE, MODIFY, and DELETE operations set to `false`.

Configure the PingDataSync Server in Standard mode

The `create-sync-pipe-config` tool is used to configure Sync Pipes and Sync Classes. For bidirectional deployments, configure two Sync Pipes, one for each directional path.

Using the create-sync-pipe Tool to Configure Synchronization illustrates a bidirectional synchronization deployment in standard mode. The example assumes that two replicated topologies are configured:

- The first endpoint topology consists of two Sun LDAP servers: the main server and one failover. Both servers have Retro change logs enabled and contain the full DIT that will be synchronized to the second endpoint.
- The second endpoint topology consists of two PingDirectory Servers: the main server and one failover. Both servers have change logs enabled and contain entries similar to the first endpoint servers, except that they use a `mail` attribute, instead of an `email` attribute.

A specific `mail` to `email` mapping must also be created to exclude the source attribute on the Sync Pipe going the other direction.

Note
If the source attribute is not excluded, the PingDataSync Server will attempt to create an `email` attribute on the second endpoint, which could fail if the attribute is not present in the destination server's schema.

Then, two Sync Classes are defined:

- One to handle the customized `email` to `mail` attribute mapping.
- Another to handle all other cases (the default Sync Class).

The `dsconfig` command is used to create the specific attribute mappings. The `resync` command is used to test the mappings. Synchronization can start using the `realtime-sync` command.

Use the create-sync-pipe tool to configure synchronization

Use the `create-sync-pipe-config` utility to configure a Sync Pipe. Once the configuration is completed, settings can be adjusted using the `dsconfig` tool.

Note
If servers have no base entries or data, the `cn=Sync User`, `cn=Root DNs`, `cn=config` account needed to communicate cannot be created. Make sure that base entries are created on the destination servers.

If synchronizing pre-encoded passwords to a Ping PingDirectory Server destination, allow pre-encoded passwords in the default password policy. Password encryption must also be configured on the destination. Be sure that the password encryption algorithm is supported by both source and destination servers with the following command:

```
$ bin/dsconfig set-password-policy-prop \
  --policy-name "Default Password Policy" \
  --set allow-pre-encoded-passwords:true
```

Encrypted and clear-text passwords can be synchronized by configuring the Sync Destination `password-synchronization-format`, and `require-secure-connection-for-clear-text-passwords` properties.

Note
The `require-secure-connection-for-clear-text-passwords` property can be set to `false` when working in a test environment. If the `password-synchronization-format` property is set to `clear-text`, and `require-secure-connection-for-clear-text-passwords` property is set

to `true`, the connection must be secure. If a secure connection is not available, an error is generated and the password is not synchronized.

Perform the following steps to configure the PingDataSync Server using `create-sync-pipe-config`:

1. Start the PingDataSync Server.

   ```
   $ <server-root>/bin/start-server
   ```

2. From the `bin` directory, run the `create-sync-pipe-config` tool.

   ```
   $ bin/create-sync-pipe-config
   ```

3. On the Initial Synchronization Configuration Tool menu, press **Enter** (yes) to continue the configuration.
4. On the Synchronization Mode menu, press **Enter** to select Standard Mode.
5. On the Synchronization Directory menu, select **oneway (1)** or **bidirectional (2)** for the synchronization topology. This example assumes bidirectional synchronization.
6. On the Source Endpoint Type menu, select the directory or database server for the first endpoint.
7. On the Source Endpoint Name menu, type a name for the endpoint server, or press **Enter** accept the default.
8. On the Base DNs menu, type the base DN on the first endpoint topology where the entries will be searched. In this example, (`dc=example,dc=com`) is used.
9. Select an option for the server security.
10. Type the host name and listener port number for the source server, or accept the default. Make sure that the endpoint servers are online and running.
11. Enter another server host and port, or press **Enter** to continue.
12. Enter the Sync User account DN for the endpoint servers, or press **Enter** to accept the default (`cn=Sync User,cn=Root DNs,cn=config`).
13. Enter and confirm a password for this account.
14. The servers in the destination endpoint topology can now be configured. Repeat steps 6–13 to configure the second server.
15. Define the maximum age of changelog log entries, or press **Enter** to accept the default.
16. After the source and destination topologies are configured, the PingDataSync Server will "prepare" each external server by testing the connection to each server. This step determines if each account has the necessary privileges (root privileges are required) to communicate with and transfer data to each endpoint during synchronization.
17. Create a name for the Sync Pipe on the Sync Pipe Name menu, or press **Enter** to accept the default. Because this configuration is bidirectional, the following step is setting up a

Sync Pipe path from the source endpoint to the destination endpoint. A later step will define another Sync Pipe from the PingDirectory Server to another server.

18. On the Sync Class Definitions menu, type **Yes** to create a custom Sync Class. A Sync Class defines the operation types (creates, modifies, or deletes), attributes that are synchronized, how attributes and DNs are mapped, and how source and destination entries are correlated.
19. Enter a name for the new Sync Class, such as "server1_to_server2."
20. On the Base DNs for Sync Class menu, enter one or more base DNs to synchronize specific subtrees of a DIT. Entries outside of the specified base DNs are excluded from synchronization. Make sure the base DNs do not overlap.
21. On the Filters for Sync Class menu, define one or more LDAP search filters to restrict specific entries for synchronization, or press **Enter** to accept the default (no). Entries that do not match the filters will be excluded from synchronization.
22. On the Synchronized Attributes for Sync Class menu, specify which attributes will be automatically mapped from one system to another. This example will exclude the source attribute (`email`) from being auto-mapped to the target servers.
23. On the Operations for Sync Class menu, select the operations that will be synchronized for the Sync Class, or press **Enter** to accept the default (`1, 2, 3`).
24. Define a default Sync Class that specifies how the other entries are processed, or press **Enter** to create a Sync Class called "Default Sync Class."
25. On the Default Sync Class Operations menu, specify the operations that the default Sync Class will handle during synchronization, or press **Enter** to accept the default.
26. Define a Sync Pipe going from the PingDirectory Server to the Sun Directory Server and exclude the `mail` attribute from being synchronized to the other endpoint servers.
27. Review the Sync Pipe Configuration Summary, and press **Enter** to accept the default (write configuration), which records the commands in a batch file (`<server-root>/sync-pipe-cfg.txt`). The batch file can be re-used to set up other topologies.

Apply the configuration changes to the local PingDataSync Server instance using a `dsconfig` batch file. Any Server SDK extensions, should be saved to the `<server-root>/lib/extensions` directory.

The next step will be to configure the attribute mappings using the `dsconfig` command.

Configuring attribute mapping

The following procedure defines an attribute map from the `email` attribute in the source servers to a `mail` attribute in the target servers. Both attributes must be valid in the target servers and must be present in their respective schemas.

Note
The following can also be done with `dsconfig` in interactive mode. Attribute mapping options are available from the PingDataSync Server main menu.

1. On the PingDataSync Server, run the `dsconfig` command to create an attribute map for the "SunDS>DS" Sync Class for the "Sun DS to Ping Identity DS" Sync Pipe, and then run the second `dsconfig` command to apply the new attribute map to the Sync Pipe and Sync Class.

```
$ bin/dsconfig --no-prompt create-attribute-map \
  --map-name "SunDS>DS Attr Map" \
  --set "description:Attribute Map for SunDS>Ping Identity Sync Class" \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

```
$ bin/dsconfig --no-prompt set-sync-class-prop \
  --pipe-name "Sun DS to DS" \
  --class-name "SunDS>DS" \
  --set "attribute-map:SunDS>DS Attr Map" \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

2. Create an attribute mapping (from `email` to `mail`) for the new attribute map.

```
$ bin/dsconfig --no-prompt create-attribute-mapping \
  --map-name "SunDS>DS Attr Map" \
  --mapping-name mail --type direct \
  --set "description:Email>Mail Mapping" \
  --set from-attribute:email \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

3. For a bidirectional deployment, repeat steps 1–2 to create an attribute map for the DS>SunDS Sync Class for the Ping Identity DS to Sun DS Sync Pipe, and create an attribute mapping that maps `mail` to `email`.

```
$ bin/dsconfig --no-prompt create-attribute-map \
  --map-name "DS>SunDS Attr Map" \
  --set "description:Attribute Map for DS>SunDS Sync Class" \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

```
$ bin/dsconfig --no-prompt set-sync-class-prop \
  --pipe-name "Ping Identity DS to Sun DS" \
  --class-name "DS>SunDS" \
  --set "attribute-map:DS>SunDS Attr Map" \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

```
$ bin/dsconfig --no-prompt create-attribute-mapping \
  --map-name "DS>SunDS Attr Map" \
  --mapping-name email \
  --type direct \
  --set "description:Mail>Email Mapping" \
  --set from-attribute:mail \
  --port 7389 \
  --bindDN "cn=admin,dc=example,dc=com" \
  --bindPassword secret
```

Configure server locations

The PingDataSync Server supports endpoint failover, which is configurable using the `location` property on the external servers. By default, the server prefers to connect to, and failover to, endpoints in the same location as itself. If there are no location settings configured, the PingDataSync Server will iterate through the configured list of external servers on the Sync Source and Sync Destination when failing over.

Note
Location-based failover is only applicable for LDAP endpoint servers.

1. On the PingDataSync Server, run the `dsconfig` command to set the location for each external server in the Sync Source and Sync Destination. For example, the following command sets the location for six servers in two data centers, `austin` and `dallas`.

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:1389 \
  --set location:austin
```

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:2389 \
  --set location:austin
```

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:3389 \
  --set location:austin
```

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:4389 \
  --set location:dallas
```

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:5389 \
  --set location:dallas
```

```
$ bin/dsconfig set-external-server-prop \
  --server-name example.com:6389 \
  --set location:dallas
```

2. Run `dsconfig` to set the location on the Global Configuration. This is the location of the PingDataSync Server itself. In this example, set the location to "austin."

```
$ bin/dsconfig set-global-configuration-prop \
  --set location:austin
```

Use the Configuration API

PingData servers provide a Configuration API, which may be useful in situations where using LDAP to update the server configuration is not possible. The API is consistent with the System for Cross-domain Identity Management (SCIM) 2.0 protocol and uses JSON as a text exchange format, so all request headers should allow the `application/json` content type.

The server includes a servlet extension that provides read and write access to the server's configuration over HTTP. The extension is enabled by default for new installations, and can be enabled for existing deployments by simply adding the extension to one of the server's HTTP Connection Handlers, as follows:

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --add http-servlet-extension:Configuration
```

The API is made available on the HTTPS Connection handler's `host:port` in the `/config` context. Due to the potentially sensitive nature of the server's configuration, the HTTPS Connection Handler should be used, for hosting the Configuration extension.

Authentication and authorization

Clients must use HTTP Basic authentication to authenticate to the Configuration API. If the username value is not a DN, then it will be resolved to a DN value using the identity mapper associated with the Configuration servlet. By default, the Configuration API uses an identity mapper that allows an entry's UID value to be used as a username. To customize this behavior, either customize the default identity mapper, or specify a different identity mapper using the Configuration servlet's `identity-mapper` property. For example:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Configuration \
  --set "identity-mapper:Alternative Identity Mapper”
```

To access configuration information, users must have the appropriate privileges:

- To access the `cn=config` backend, users must have the `bypass-acl` privilege or be allowed access to the configuration using an ACI.
- To read configuration information, users must have the `config-read` privilege.
- To update the configuration, users must have the `config-write` privilege.

Relationship between the Configuration API and the dsconfig tool

The Configuration API is designed to mirror the `dsconfig` tool, using the same names for properties and object types. Property names are presented as hyphen case in `dsconfig` and as camel-case attributes in the API. In API requests that specify property names, case is not important. Therefore, `baseDN` is the same as `baseDn`. Object types are represented in hyphen case. API paths mirror what is in dsconfig. For example, the `dsconfig list-connection-handlers` command is analogous to the API's `/config/connection-handlers` path. Object types that appear in the schema URNs adhere to a `type:subtype` syntax. For example, a Local

DB Backend's schema URN is `urn:unboundid:schemas:configuration:2.0:backend:local-db`. Like the `dsconfig` tool, all configuration updates made through the API are recorded in `logs/config-audit.log`.

The API includes the filter, sort, and pagination query parameters described by the SCIM specification. Specific attributes may be requested using the `attributes` query parameter, whose value must be a comma-delimited list of properties to be returned, for example `attributes=baseDN,description`. Likewise, attributes may be excluded from responses by specifying the `excludedAttributes` parameter. See Sorting and Filtering with the Configuration API for more information on query parameters.

Operations supported by the API are those typically found in REST APIs:

HTTP Method	Description	Related dsconfig Example

GET	Lists the attributes of an object when used with a path representing an object, such as `/config/global-configuration` or `/config/backends/userRoot`. Can also list objects when used with a path representing a parent relation, such as `/config/backends`.	`get-backend-prop`
		`list-backends`
		`get-global-configuration-prop`

POST	Creates a new instance of an object when used with a relation parent path, such as `config/backends`.	`create-backend`
PUT	Replaces the existing attributes of an object. A PUT operation is similar to a PATCH operation, except that the PATCH is determined by determining the difference between an existing target object and a supplied source object. Only those attributes in the source object are modified in the target object. The target object is specified using a path, such as `/config/backends/userRoot`.	`set-backend-prop` `set-global-configuration-prop`
PATCH	Updates the attributes of an existing object when used with a path representing an object, such as `/config/backends/userRoot`. See PATCH Example.	`set-backend-prop` `set-global-configuration-prop`
DELETE	Deletes an existing object when used with a path representing an object, such as `/config/backends/userRoot`.	`delete-backend`

The OPTIONS method can also be used to determine the operations permitted for a particular path.

Object names, such as `userRoot` in the Description column, must be URL-encoded in the `path` segment of a URL. For example, `%20` must be used in place of spaces, and `%25` is used in place of the percent (%) character. So the URL for accessing the HTTP Connection Handler object is:

```
/config/connection-handlers/http%20connection%20handler
```

GET Example

The following is a sample GET request for information about the `userRoot` backend:

```
GET /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json
```

The response:


```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://localhost:5033/config/backends/userRoot"
  },
  "backendID": "userRoot2",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example2,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "10",
  "dbCacheSize": "0 b",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "1 m",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "0",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
  "dbNumCleanerThreads": "0",
  "dbNumLockTables": "0",
  "dbRunCleaner": "true",
  "dbTxnNoSync": "false",
  "dbTxnWriteNoSync": "true",
  "dbUseThreadLocalHandles": "true",
  "deadlockRetryLimit": "10",
  "defaultCacheMode": "cache-keys-and-values",
  "defaultTxnMaxLockTimeout": "10 s",
  "defaultTxnMinLockTimeout": "10 s",
  "enabled": "false",
  "explodedIndexEntryThreshold": "4000",
  "exportThreadCount": "0",
```

```
  "externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
  "externalTxnDefaultMaxLockTimeout": "100 ms",
  "externalTxnDefaultMinLockTimeout": "100 ms",
  "externalTxnDefaultRetryAttempts": "2",
  "hashEntries": "false",
  "id2childrenIndexEntryLimit": "66",
  "importTempDirectory": "import-tmp",
  "importThreadCount": "16",
  "indexEntryLimit": "4000",
  "isPrivateBackend": "false",
  "javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
  "jeProperty": [
    "je.cleaner.adjustUtilization=false",
    "je.nodeMaxEntries=32"
  ],
  "numRecentChanges": "50000",
  "offlineProcessDatabaseOpenTimeout": "1 h",
  "primeAllIndexes": "true",
  "primeMethod": [
    "none"
  ],
  "primeThreadCount": "2",
  "primeTimeLimit": "0 ms",
  "processFiltersWithUndefinedAttributeTypes": "false",
  "returnUnavailableForUntrustedIndex": "true",
  "returnUnavailableWhenDisabled": "true",
  "setDegradedAlertForUntrustedIndex": "true",
  "setDegradedAlertWhenDisabled": "true",
  "subtreeDeleteBatchSize": "5000",
  "subtreeDeleteSizeLimit": "5000",
  "uncachedId2entryCacheMode": "cache-keys-only",
  "writabilityMode": "enabled"
}
```


GET list example

The following is a sample GET request for all local backends:

```
GET /config/backends
Host: example.com:5033
Accept: application/scim+json
```

The response (which has been shortened):


```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 24,
  "Resources": [
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:ldif"
      ],
      "id": "adminRoot",
      "meta": {
```

```
        "resourceType": "LDIF Backend",
        "location": "http://localhost:5033/config/backends/adminRoot"
      },
      "backendID": "adminRoot",
      "backupFilePermissions": "700",
      "baseDN": [
        "cn=admin data"
      ],
      "enabled": "true",
      "isPrivateBackend": "true",
      "javaClass": "com.unboundid.directory.server.backends.LDIFBackend",
      "ldifFile": "config/admin-backend.ldif",
      "returnUnavailableWhenDisabled": "true",
      "setDegradedAlertWhenDisabled": "false",
      "writabilityMode": "enabled"
    },
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:trust-store"
      ],
      "id": "ads-truststore",
      "meta": {
        "resourceType": "Trust Store Backend",
        "location": "http://localhost:5033/config/backends/ads-truststore"
      },
      "backendID": "ads-truststore",
      "backupFilePermissions": "700",
      "baseDN": [
        "cn=ads-truststore"
      ],
      "enabled": "true",
      "javaClass":
"com.unboundid.directory.server.backends.TrustStoreBackend",
      "returnUnavailableWhenDisabled": "true",
      "setDegradedAlertWhenDisabled": "true",
      "trustStoreFile": "config/server.keystore",
      "trustStorePin": "********",
      "trustStoreType": "JKS",
      "writabilityMode": "enabled"
    },
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:alarm"
      ],
      "id": "alarms",
      "meta": {
        "resourceType": "Alarm Backend",
        "location": "http://localhost:5033/config/backends/alarms"
      },
 ...
```

PATCH example

Configuration can be modified using the HTTP PATCH method. The PATCH request body is a JSON object formatted according to the SCIM patch request. The Configuration API, supports a subset of possible values for the `path` attribute, used to indicate the configuration attribute to modify.

The configuration object's attributes can be modified in the following ways. These operations are analogous to the `dsconfig modify-[object]` options.

- An operation to set the single-valued `description` attribute to a new value:

```
{
  "op" : "replace",
  "path" : "description",
  "value" : "A new backend."
}
```

  is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --set "description:A new backend"
```

- An operation to add a new value to the multi-valued `jeProperty` attribute:

```
{
  "op" : "add",
  "path" : "jeProperty",
  "value" : "je.env.backgroundReadLimit=0"
}
```

  is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --add je-property:je.env.backgroundReadLimit=0
```

- An operation to remove a value from a multi-valued property. In this case, `path` specifies a SCIM filter identifying the value to remove:

```
{
  "op" : "remove",
  "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
}
```

  is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.cleaner.adjustUtilization=false
```

- A second operation to remove a value from a multi-valued property, where the `path` specifies both an attribute to modify, and a SCIM filter whose attribute is `value`:

```
{
  "op" : "remove",
  "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.nodeMaxEntries=32
```

- An option to remove one or more values of a multi-valued attribute. This has the effect of restoring the attribute's value to its default value:

```
{
  "op" : "remove",
  "path" : "id2childrenIndexEntryLimit"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --reset id2childrenIndexEntryLimit
```

The following is the full example request. The API responds with the entire modified configuration object, which may include a SCIM extension attribute `urn:unboundid:schemas:configuration:messages` containing additional instructions:

Example request:

```
PATCH /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json

{
  "schemas" : [ "urn:ietf:params:scim:api:messages:2.0:PatchOp" ],
  "Operations" : [ {
    "op" : "replace",
    "path" : "description",
    "value" : "A new backend."
  }, {
    "op" : "add",
    "path" : "jeProperty",
    "value" : "je.env.backgroundReadLimit=0"
  }, {
    "op" : "remove",
    "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
  }, {
    "op" : "remove",
    "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
  }, {
    "op" : "remove",
    "path" : "id2childrenIndexEntryLimit"
  } ]
}
```

Example response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
```

```
  ],
  "id": "userRoot2",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://example.com:5033/config/backends/userRoot2"
  },
  "backendID": "userRoot2",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example2,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "10",
  "dbCacheSize": "0 b",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "1 m",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "0",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
  "dbNumCleanerThreads": "0",
  "dbNumLockTables": "0",
  "dbRunCleaner": "true",
  "dbTxnNoSync": "false",
  "dbTxnWriteNoSync": "true",
  "dbUseThreadLocalHandles": "true",
  "deadlockRetryLimit": "10",
  "defaultCacheMode": "cache-keys-and-values",
  "defaultTxnMaxLockTimeout": "10 s",
  "defaultTxnMinLockTimeout": "10 s",
  "description": "123",
  "enabled": "false",
  "explodedIndexEntryThreshold": "4000",
  "exportThreadCount": "0",
  "externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
```

```
  "externalTxnDefaultMaxLockTimeout": "100 ms",
  "externalTxnDefaultMinLockTimeout": "100 ms",
  "externalTxnDefaultRetryAttempts": "2",
  "hashEntries": "false",
  "importTempDirectory": "import-tmp",
  "importThreadCount": "16",
  "indexEntryLimit": "4000",
  "isPrivateBackend": "false",
  "javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
  "jeProperty": [
  "\"je.env.backgroundReadLimit=0\""
  ],
  "numRecentChanges": "50000",
  "offlineProcessDatabaseOpenTimeout": "1 h",
  "primeAllIndexes": "true",
  "primeMethod": [
    "none"
  ],
  "primeThreadCount": "2",
  "primeTimeLimit": "0 ms",
  "processFiltersWithUndefinedAttributeTypes": "false",
  "returnUnavailableForUntrustedIndex": "true",
  "returnUnavailableWhenDisabled": "true",
  "setDegradedAlertForUntrustedIndex": "true",
  "setDegradedAlertWhenDisabled": "true",
  "subtreeDeleteBatchSize": "5000",
  "subtreeDeleteSizeLimit": "5000",
  "uncachedId2entryCacheMode": "cache-keys-only",
  "writabilityMode": "enabled",
  "urn:unboundid:schemas:configuration:messages:2.0": {
    "requiredActions": [
      {
        "property": "jeProperty",
        "type": "componentRestart",
        "synopsis": "In order for this modification to take effect,
         the component must be restarted, either by disabling and
         re-enabling it, or by restarting the server"
      },
      {
        "property": "id2childrenIndexEntryLimit",
        "type": "other",
        "synopsis": "If this limit is increased, then the contents
         of the backend must be exported to LDIF and re-imported to
         allow the new limit to be used for any id2children keys
         that had already hit the previous limit."
      }
    ]
  }
}
```

API paths

The Configuration API is available under the `/config` path. A full listing of root sub-paths can be obtained from the `/config/ResourceTypes` endpoint:

```
GET /config/ResourceTypes
Host: example.com:5033
Accept: application/scim+json
```

Sample response (abbreviated):

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 520,
  "Resources": [
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "dsee-compat-access-control-handler",
      "name": "DSEE Compat Access Control Handler",
      "description": "The DSEE Compat Access Control
         Handler provides an implementation that uses syntax
         compatible with the Sun Java System Directory Server
         Enterprise Edition access control handler.",
      "endpoint": "/access-control-handler",
      "meta": {
        "resourceType": "ResourceType",
        "location": "http://example.com:5033/config/ResourceTypes/dsee-compat-
access-control-handler"
      }
    },
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "access-control-handler",
      "name": "Access Control Handler",
      "description": "Access Control Handlers manage the
         application-wide access control. The server's access
         control handler is defined through an extensible
         interface, so that alternate implementations can be created.
         Only one access control handler may be active in the server
         at any given time.",
      "endpoint": "/access-control-handler",
      "meta": {
        "resourceType": "ResourceType",
        "location": "http://example.com:5033/config/ResourceTypes/access-
control-handler"
```

```
      }
    },
    {
...
```

The response's `endpoint` elements enumerate all available sub-paths. The path `/config/access-control-handler` in the example can be used to get a list of existing access control handlers, and create new ones. A path containing an object name like `/config/backends/{backendName}`, where `{backendName}` corresponds to an existing backend (such as `userRoot`) can be used to obtain an object's properties, update the properties, or delete the object.

Some paths reflect hierarchical relationships between objects. For example, properties of a local DB VLV index for the `userRoot` backend are available using a path like `/config/backends/userRoot/local-db-indexes/uid`. Some paths represent singleton objects, which have properties but cannot be deleted nor created. These paths can be differentiated from others by their singular, rather than plural, relation name (for example `global-configuration`).

Sorting and filtering configuration objects

The Configuration API supports SCIM parameters for filter, sorting, and pagination. Search operations can specify a SCIM filter used to narrow the number of elements returned. See the SCIM specification for the full set of operations for SCIM filters. Clients may also specify sort parameters, or paging parameters. As previously mentioned, clients may specify attributes to include or exclude in both get and list operations.

GET Parameters for Sorting and Filtering

GET Parameter	Description
filter	Values can be simple SCIM filters such as `id eq "userRoot"` or compound filters like `meta.resourceType eq "Local DB Backend"` and `baseDn co "dc=exmple,dc=com"`.
sortBy	Specifies a property value by which to sort.
sortOrder	Specifies either `ascending` or `descending` alphabetical order.
startIndex	1-based index of the first result to return.
count	Indicates the number of results per page.

Update properties

The Configuration API supports the HTTP PUT method as an alternative to modifying objects with HTTP PATCH. With PUT, the server computes the differences between the object in the request with the current version in the server, and performs modifications where necessary. The server will never remove attributes that are not specified in the request. The API responds with the entire modified object.

Request:

```
PUT /config/backends/userRoot
Host: example.com:5033
```

```
Accept: application/scim+json
{
  "description" : "A new description."
}
```

Response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://example.com:5033/config/backends/userRoot"
  },
  "backendID": "userRoot",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "25",
  "dbCacheSize": "0 b",
  "dbCheckpointerBytesInterval": "20 mb",
  "dbCheckpointerHighPriority": "false",
  "dbCheckpointerWakeupInterval": "30 s",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "5",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
  "dbNumCleanerThreads": "1",
  "dbNumLockTables": "0",
  "dbRunCleaner": "true",
  "dbTxnNoSync": "false",
  "dbTxnWriteNoSync": "true",
  "dbUseThreadLocalHandles": "true",
```

```
  "deadlockRetryLimit": "10",
  "defaultCacheMode": "cache-keys-and-values",
  "defaultTxnMaxLockTimeout": "10 s",
  "defaultTxnMinLockTimeout": "10 s",
  "description": "abc",
  "enabled": "true",
  "explodedIndexEntryThreshold": "4000",
  "exportThreadCount": "0",
  "externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
  "externalTxnDefaultMaxLockTimeout": "100 ms",
  "externalTxnDefaultMinLockTimeout": "100 ms",
  "externalTxnDefaultRetryAttempts": "2",
  "hashEntries": "true",
  "importTempDirectory": "import-tmp",
  "importThreadCount": "16",
  "indexEntryLimit": "4000",
  "isPrivateBackend": "false",
  "javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
  "numRecentChanges": "50000",
  "offlineProcessDatabaseOpenTimeout": "1 h",
  "primeAllIndexes": "true",
  "primeMethod": [
    "none"
  ],
  "primeThreadCount": "2",
  "primeTimeLimit": "0 ms",
  "processFiltersWithUndefinedAttributeTypes": "false",
  "returnUnavailableForUntrustedIndex": "true",
  "returnUnavailableWhenDisabled": "true",
  "setDegradedAlertForUntrustedIndex": "true",
  "setDegradedAlertWhenDisabled": "true",
  "subtreeDeleteBatchSize": "5000",
  "subtreeDeleteSizeLimit": "100000",
  "uncachedId2entryCacheMode": "cache-keys-only",
  "writabilityMode": "enabled"
}
```


Administrative actions

Updating a property may require an administrative action before the change can take effect. If so, the server will return `200 Success`, and any actions are returned in the `urn:unboundid:schemas:configuration:messages:2.0` section of the JSON response that represents the entire object that was created or modified.

For example, changing the `jeProperty` of a backend will result in the following:


```
"urn:unboundid:schemas:configuration:messages:2.0": {
  "requiredActions": [
    {
      "property": "baseContextPath",
      "type": "componentRestart",
```

```
      "synopsis": "In order for this modification to
         take effect, the component must be restarted,
         either by disabling and re-enabling it, or by
         restarting the server"
    },
    {
      "property": "id2childrenIndexEntryLimit",
      "type": "other",
      "synopsis": "If this limit is increased, then the
         contents of the backend must be exported to LDIF
         and re-imported to allow the new limit to be used
         for any id2children keys that had already hit the
         previous limit."
    }
  ]
}
...
```

Update servers and server groups

Servers can be configured as part of a server group, so that configuration changes that are applied to a single server, are then applied to all servers in a group. When managing a server that is a member of a server group, creating or updating objects using the Configuration API requires the `applyChangeTo` query parameter. The behavior and acceptable values for this parameter are identical to the `dsconfig` parameter of the same name. A value of `singleServer` or `serverGroup` can be specified. For example:

```
https://example.com:5033/config/Backends/userRoot?applyChangeTo=singleServer
```

Note
This does not apply to mirrored subtree objects, which include Topology and Cluster level objects. Changes made to mirrored objects are applied to all objects in the subtree.

Configuration API Responses

Clients of the API should examine the HTTP response code in order to determine the success or failure of a request. The following are response codes and their meanings:

Response Code	Description	Response Body
200 Success	The requested operation succeeded, with the response body being the configuration object that was created or modified. If further actions are required, they are included in the `urn:unboundid:schemas:configuration:messages:2.0` object.	List of objects, or object properties, administrative actions.
204 No Content	The requested operation succeeded and no further information has been provided, such as in the case of a DELETE operation.	None.
400 Bad Request	The request contents are incorrectly formatted or a request is made for an invalid API version.	Error summary and optional

Response Code	Description	Response Body
		message.
401 Unauthorized	User authentication is required. Some user agents such as browsers may respond by prompting for credentials. If the request had specified credentials in an Authorization header, they are invalid.	None.
403 Forbidden	The requested operation is forbidden either because the user does not have sufficient privileges or some other constraint such as an object is edit-only and cannot be deleted.	None.
404 Not Found	The requested path does not refer to an existing object or object relation.	Error summary and optional message.
409 Conflict	The requested operation could not be performed due to the current state of the configuration. For example, an attempt was made to create an object that already exists or an attempt was made to delete an object that is referred to by another object.	Error summary and optional message.
415 Unsupported Media Type	The request is such that the Accept header does not indicate that JSON is an acceptable format for a response.	None.
500 Server Error	The server encountered an unexpected error. Please report server errors to customer support.	Error summary and optional message.

An application that uses the Configuration API should limit dependencies on particular text appearing in error message content. These messages may change, and their presence may depend on server configuration. Use the HTTP return code and the context of the request to create a client error message. The following is an example encoded error message:

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:Error"
  ],
  "status": 404,
  "scimType": null,
  "detail": "The Local DB Index does not exist."
}
```

Configuration with the dsconfig tool

The Ping Identity servers provide several command-line tools for management and administration. The command-line tools are available in the `bin` directory for UNIX or Linux systems and `bat` directory for Microsoft Windows systems.

The `dsconfig` tool is the text-based management tool used to configure the underlying server configuration. The tool has three operational modes:

- Interactive mode
- Non-interactive mode

- Batch mode

The `dsconfig` tool also offers an offline mode using the `--offline` option, in which the server does not have to be running to interact with the configuration. In most cases, the configuration should be accessed with the server running in order for the server to give the user feedback about the validity of the configuration.

Each command-line utility provides a description of the subcommands, arguments, and usage examples needed to run the tool. View detailed argument options and examples by typing `--help` with the command.

```
$ bin/dsconfig --help
```

To list the subcommands for each command:

```
$ bin/dsconfig --help-subcommands
```

To list more detailed subcommand information:

```
$ bin/dsconfig list-log-publishers --help
```

Use dsconfig in interactive mode

Running `dsconfig` in interactive command-line mode provides a menu-driven interface for accessing and configuring the PingData server. To start `dsconfig` in interactive mode, run the tool without any arguments:

```
$ bin/dsconfig
```

Running the tool requires server connection and authentication information. After connection information is confirmed, a menu of the available operation types is displayed.

Use dsconfig in non-interactive mode

Non-interactive command-line mode provides a simple way to make arbitrary changes to the server, and to use administrative scripts to automate configuration changes. To make changes to multiple configuration objects at the same time, use batch mode.

The general format for the non-interactive command line is:

```
$ bin/dsconfig --no-prompt {globalArgs} {subcommand} {subcommandArgs}
```

The `--no-prompt` argument specifies non-interactive mode. The `{globalArgs}` argument provides a set of arguments that specify how to connect and authenticate to the server. Global arguments can be standard LDAP connection parameters or SASL connection parameters depending on the implementation. The `{subcommand}` specifies which general action to perform. The following uses standard LDAP connections:

```
$ bin/dsconfig --no-prompt list-backends \
  --hostname server.example.com \
  --port 389 \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword password
```

The following uses SASL GSSAPI (Kerberos) parameters:

```
$ bin/dsconfig --no-prompt list-backends \
  --saslOption mech=GSSAPI \
  --saslOption authid=admin@example.com \
  --saslOption ticketcache=/tmp/krb5cc_1313 \
  --saslOption useticketcache=true
```

The `{subcommandArgs}` argument contains a set of arguments specific to the particular task. To always display the advanced properties, use the `--advanced` command-line option.

Note
Global arguments can appear anywhere on the command line. The subcommand-specific arguments can appear anywhere after the subcommand.

Use dsconfig batch mode

The `dsconfig` tool provides a batching mechanism that reads multiple invocations from a file and executes them sequentially. The batch file provides advantages over standard scripting by minimizing LDAP connections and JVM invocations that normally occur with each `dsconfig` call. Batch mode is the best method to use with setup scripts when moving from a development environment to test environment, or from a test environment to a production environment. The `--no-prompt` option is required with `dsconfig` in batch mode.

```
$ bin/dsconfig --no-prompt \
  --hostname host1 \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --batch-file /path/to/sync-pipe-config.txt
```

If a `dsconfig` command has a missing or incorrect argument, the command will fail and stop the batch process without applying any changes to the server. A `--batch-continue-on-error` option is available, which instructs `dsconfig` to apply all changes and skip any errors.

View the `logs/config-audit.log` file to review the configuration changes made to the server, and use them in the batch file. The batch file can have blank lines for spacing, and lines starting with a pound sign (#) for comments. The batch file also supports a "\" line continuation character for long commands that require multiple lines.

The PingDataSync Server also provides a `docs/sun-ds-compatibility.dsconfig` file for migrations from Sun/Oracle to Ping Identity PingDataSync Server machines.

Topology configuration

Topology configuration enables grouping servers and mirroring configuration changes automatically. It uses a master/slave architecture for mirroring shared data across the topology. All writes and updates are forwarded to the master, which forwards them to all other servers. Reads can be served by any server in the group.

Note
To remove a server from the topology, it must be uninstalled with the `uninstall` tool.

Topology master requirements and selection

A topology master server receives any configuration change from other servers in the topology, verifies the change, then makes the change available to all connected servers when they poll the master. The master always sends a digest of its subtree contents on each update. If the node has a different digest than the master, it knows it's not synchronized. The servers will pull the entire subtree from the master if they detect that they are not synchronized. A server may detect it is not synchronized with the master under the following conditions:

- At the end of its periodic polling interval, if a server's subtree digest differs from that of its master, then it knows it's not synchronized.
- If one or more servers have been added to or removed from the topology, the servers will not synchronized.

The master of the topology is selected by prioritizing servers by minimum supported product version, most available, newest server version, earliest start time, and startup UUID (a smaller UUID is preferred).

After determining a master, the topology data is reviewed from all available servers (every five seconds by default) to determine if any new information makes a server better suited to being the master. If a new server can be the master, it will communicate that to the other servers, if no other server has advertised that it should be the master. This ensures that all servers accept the same master at approximately the same time (within a few milliseconds of each other). If there is no better master, the initial master maintains the role.

After the best master has been selected for the given interval, the following conditions are confirmed:

- A majority of servers is reachable from that master. (The master server itself is considered while determining this majority.)
- There is only a single master in the entire topology.

If either of these conditions is not met, the topology is without a master and the peer polling frequency is reduced to 100 milliseconds to find a new master as quickly as possible. If there is no master in the topology for more than one minute, a `mirrored-subtree-manager-no-master-found` alarm is raised. If one of the servers in the topology is forced as master with the `force-as-master-for-mirrored-data` option in the Global Configuration configuration object, a `mirrored-subtree-manager-forced-as-master-warning` warning alarm is raised. If multiple servers have been forced as masters, then a `mirrored-subtree-manager-forced-as-master-error` critical alarm will be raised.

Topology components

When a server is installed, it can be added to an existing topology, which will clone the server's . Topology settings are designed to operate without additional configuration. If required, some settings can be adjusted to fit the needs of the environment.

Server configuration settings

Configuration settings for the topology are configured in the Global Configuration and in the Config File Handler Backend. Though they are topology settings, they are unique to each server and are not mirrored. Settings must be kept the same on all servers.

The Global Configuration object contains a single topology setting, `force-as-master-for-mirrored-data`. This should be set to `true` on only one of the servers in the topology, and is used only if a situation occurs where the topology cannot determine a master because a majority of servers is not available. A server with this setting enabled will be assigned the role of master, if no suitable master can be determined. See Topology master requirements and selection for details about how a master is selected for a topology.

The Config File Handler Backend defines three topology (`mirrored-subtree`) settings:

- `mirrored-subtree-peer-polling-interval` – Specifies the frequency at which the server polls its topology peers to determine if there are any changes that may warrant a new master selection. A lower value will ensure a faster failover, but it will also cause more traffic among the peers. The default value is five seconds. If no suitable master is found, the polling frequency is adjusted to 100 milliseconds until a new master is selected.
- `mirrored-subtree-entry-update-timeout` – Specifies the maximum length of time to wait for an update operation (add, delete, modify or modify-dn) on an entry to be applied by the master on all of the servers in the topology. The default is 10 seconds. In reality, updates can take up to twice as much time as this timeout value if master selection is in progress at the time the update operation was received.
- `mirrored-subtree-search-timeout` – Specifies the maximum length of time in milliseconds to wait for search operations to complete. The default is 10 seconds.

Topology settings

Topology meta-data is stored under the `cn=topology,cn=config` subtree and cluster data is stored under the `cn=cluster,cn=config` subtree. The only setting that can be changed is the cluster name.

Monitor data for the topology

Each server has a monitor that exposes that server's view of the topology in its monitor backend, so that peer servers can periodically read this information to determine if there are changes in the topology. Topology data includes the following:

- The server ID of the current master, if the master is not known.
- The instance name of the current master, or if a master is not set, a description stating why a master is not set.
- A flag indicating if this server thinks that it should be the master.

- A flag indicating if this server is the current master.
- A flag indicating if this server was forced as master.
- The total number of configured peers in the topology group.
- The peers connected to this server.
- The current availability of this server
- A flag indicating whether or not this server is not synchronized with its master, or another node in the topology if the master is unknown.
- The amount of time in milliseconds where multiple masters were detected by this server.
- The amount of time in milliseconds where no suitable server is found to act as master.
- A SHA-256 digest encoded as a base-64 string for the current subtree contents.

The following metrics are included if this server has processed any operations as master:

- The number of operations processed by this server as master.
- The number of operations processed by this server as master that were successful.
- The number of operations processed by this server as master that failed to validate.
- The number of operations processed by this server as master that failed to apply.
- The average amount of time taken (in milliseconds) by this server to process operations as the master.
- The maximum amount of time taken (in milliseconds) by this server to process an operation as the master.

Updating the server instance listener certificate

To change the SSL certificate for the server, update the keystore and truststore files with the new certificate. The certificate file must have the new certificate in PEM-encoded format, such as:

```
-----BEGIN CERTIFICATE-----

MIIDKTCCAhGgAwIBAgIEacgGrDANBgkqhkiG9w0BAQsFADBFMR4wHAYDVQQKExVVbmJvdW5kSUQgQ2
VydGlmaWNhdGUxIzAhBgNVBAMTGnZtLW1lZGl1bS03My51bmJvdW5kaWQubGFiMB4XDTE1MTAxMjE1
MzU0OFoXDTM1MTAwNzE1MzU0OFowRTEeMBwGA1UEChMVVW5ib3VuZElEIENlcnRpZmljYXRlMSMwIQ
YDVQQDExp2bS1tZWRpdW0tNzMudW5ib3VuZGlkLmxhYjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBAKN4tAN3o9Yw6Cr9hivwVDxJqF6+aEi9Ir3WGFYLSrggRNXsiAOfWkSMWdIC5vyF5OJ9Dl
IgvHL4OuqP/YNEGzKDkgr6MwtUeVSK14+dCixygJGC0nY7k+f0WSCjtIHzrmc4WWdrZXmgb+qv9Lup
S30JG0FXtcbGkYpjaKXIEqMg4ekz3B5cAvE0SQUFyXEdN4rWOn96nVFkb2CstbiPzAgne2tu7paJ6S
GFOW0UF7v018XY1m2WHBIoD0WC8nOVLTG9zFUavaOxtlt1TlhClkI4HRMNg8n2EtSTdQRizKuw9DdT
XJBb6Kfvnp/nI73VHRyt47wUVueehEDfLtDP8pMCAwEAAaMhMB8wHQYDVR0OBBYEFMrwjWxl2K+yd9
+Y65oKn0g5jITgMA0GCSqGSIb3DQEBCwUAA4IBAQBpsBYodblUGew+HewqtO2i8Wt+vAbt31zM5/kR
vo6/+iPEASTvZdCzIBcgletxKGKeCQ0GPeHr42+erakiwmGDlUTYrU3LU5pTGTDLuR2IllTT5xlEhC
```

```
WJGWipW4q3Pl3cX/9m2ffY/JLYDfTJaoJvnXrh7Sg719skkHjWZQgOHXlkPLx5TxFGhAovE1D4qLVR
WGohdpWDrIgFh0DVfoyAn1Ws9ICCXdRayajFI4Lc6K1m6SA5+25Y9nno8BhVPf4q5OW6+UDc8MsLbB
sxpwvR6RJ5cv3ypfOriTehJsG+9ZDo7YeqVsTVGwAlW3PiSd9bYP/8yu9Cy+0MfcWcSeAE
-----END CERTIFICATE-----
```

If clients that already have a secure connection established with this server need to be maintained, information about both certificates can reside in the same file (each with their own begin and end headers and footers).

After the keystore and truststore files are updated, run the following `dsconfig` command to update the server's certificate in the topology registry:

```
$ bin/dsconfig set-server-instance-listener-prop \
  --instance-name <server-instance-name> \
  --listener-name ldap-listener-mirrored-config \
  --set listener-certificate <path-to-new-certificate-file>
```

The `listener-certificate` in the topology registry is like a trust store. The public certificates that it has are automatically trusted by the local server. When the local server attempts a secure LDAP connection to a peer, and the peer presents it with its certificate, the local server will check the `listener-certificate` property for that server in the topology registry. If the property contains the peer server's certificate, the local server will trust the peer.

Remove the self-signed certificate

The server is installed with a self-signed certificate and key (`ads-certificate`), which are used for internal purposes such as replication authentication, inter-server authentication in the topology registry, reversible password encryption, and encrypted backup/LDIF export. The ads-certificate lives in the keystore file called `ads-truststore` under the server's `/config` directory. If your deployment requires removing the self-signed certificate, it can be replaced.

The certificate is stored in the topology registry, which enables replacing it on one server and having it mirrored to all other servers in the topology. Any change is automatically mirrored on other servers in the topology. It is stored in human-readable PEM-encoded format and can be updated with `dsconfig`. The following general steps are required to replace the self-signed certificate:

1. Prepare a new keystore with the replacement key-pair.
2. Update the server configuration to use the new certificate by adding it to the server's list of certificates in the topology registry so that it is trusted by other servers.
3. Update the server's `ads-truststore` file to use the new key-pair.
4. Retire the old certificate by removing it from the topology registry.

Note

Replacing the entire key-pair instead of just the certificate associated with the original private key can make existing backups and LDIF exports invalid. This should be performed immediately after setup or before the key-pair is used. After the first time, only the certificate associated with the private key should have to be changed, for example, to extend its validity period or replace it with a certificate signed by a different CA.

Prepare a new keystore with the replacement key-pair

The self-signed certificate can be replaced with an existing key-pair, or the certificate associated with the original key-pair can be used.

Use an existing key-pair

If a private key and certificate(s) in PEM-encoded format already exist, both the original private key and self-signed certificate can be replaced in ads-truststore with the `manage-certificates` tool. The following command imports existing certificates into a new keystore file, `ads-truststore.new`:

```
$ bin/manage-certificates import-certificate \
  --keystore ads-truststore.new \
  --keystore-type JKS \
  --keystore-password-file ads-truststore.pin \
  --alias ads-certificate \
  --private-key-file existing.key \
  --certificate-file existing.crt \
  --certificate-file intermediate.crt \
  --certificate-file root-ca.crt
```

The certificates listed using the `--certificate-file` options must be ordered so that each subsequent certificate is the issuer for the previous one. So the server certificate comes first, the intermediate certificates next (if any), and the root CA certificate last.

Use the certificate associated with the original key-pair

The certificate associated with the original server-generated private key can be replaced with the following commands:

1. Create a CSR for the `ads-certificate`:

   ```
   $ bin/manage-certificates generate-certificate-signing-request \
     --keystore ads-truststore \
     --keystore-type JKS \
     --keystore-password-file ads-truststore.pin \
     --alias ads-certificate \
     --use-existing-key-pair \
     --subject-dn "CN=ldap.example.com,O=Example Corporation,C=US" \
     --output-file ads.csr
   ```

2. Submit `ads.csr` to a CA for signing.
3. Export the server's private key into `ads.key`:

   ```
   $ bin/manage-certificates export-private-key \
     --keystore ads-truststore \
     --keystore-password-file ads-truststore.pin \
     --alias ads-certificate \
     --output-file ads.key
   ```

4. Import the certificates obtained from the CA (the CA-signed server certificate, any intermediate certificates, and root CA certificate) into `ads-truststore.new`:

```
$ bin/manage-certificates import-certificate \
  --keystore ads-truststore.new \
  --keystore-type JKS \
  --keystore-password-file ads-truststore.pin \
  --alias ads-certificate \
  --private-key-file ads.key \
  --certificate-file new-ads.crt \
  --certificate-file intermediate.crt \
  --certificate-file root-ca.crt
```

Update the server configuration to use the new certificate

To update the server to use the desired key-pair, the `inter-server-certificate` property for the server instance must first be updated in the topology registry. The old and the new certificates may appear within their own begin and end headers in the `inter-server-certificate` property to support transitioning from the old certificate to the new one.

1. Export the server's old ads-certificate into `old-ads.crt`:

```
$ bin/manage-certificates export-certificate \
  --keystore ads-truststore \
  --keystore-password-file ads-truststore.pin \
  --alias ads-certificate \
  --export-certificate-chain \
  --output-file old-ads.crt
```

2. Concatenate the old, new certificate, and issuer certificates into one file. On Windows, an editor like notepad can be used. On Unix platforms, use the following command:

```
$ cat old-ads.crt new-ads.crt intermediate.crt root-ca.crt > chain.crt
```

3. Update the `inter-server-certificate` property for the server instance in the topology registry using `dsconfig`:

```
$ bin/dsconfig -n set-server-instance-prop \
  --instance-name <instance-name> \
  --set "inter-server-certificate<chain.crt"
```

Update the ads-truststore file to use the new key-pair

The server will still use the old `ads-certificate`. When the new `ads-certificate` needs to go into effect, the old `ads-truststore` file must be replaced with `ads-truststore.new` in the server's `config` directory.

```
$ mv ads-truststore.new ads-truststore
```

Retire the old certificate

The old certificate is retired by removing it from the topology registry when it has expired. All existing encrypted backups and LDIF exports are not affected because the public key in the old and new server certificates are the same, and the private key will be able to decrypt them.

```
$ cat new-ads.crt intermediate.crt root-ca.crt > chain.crt
```

```
$ bin/dsconfig -n set-server-instance-prop \
  --instance-name <instance-name> \
  --set "inter-server-certificate<chain.crt"
```

Domain Name Service (DNS) caching

If needed, two global configuration properties can be used to control the caching of hostname-to-numeric IP address (DNS lookup) results returned from the name resolution services of the underlying operating system. Use the `dsconfig` tool to configure these properties.

network-address-cache-ttl– Sets the Java system property `networkaddress.cache.ttl`, and controls the length of time in seconds that a hostname-to-IP address mapping can be cached. The default behavior is to keep resolution results for one hour (3600 seconds). This setting applies to the server and all extensions loaded by the server.

network-address-outage-cache-enabled – Caches hostname-to-IP address results in the event of a DNS outage. This is set to `true` by default, meaning name resolution results are cached. Unexpected service interruptions may occur during planned or unplanned maintenance, network outages or an infrastructure attack. This cache may allow the server to function during a DNS outage with minimal impact. This cache is not available to server extensions.

IP address reverse name lookups

Ping Identity servers do not explicitly perform numeric IP address-to-hostname lookups. However, address masks configured in Access Control Lists (ACIs), Connection Handlers, Connection Criteria, and Certificate handshake processing may trigger implicit reverse name lookups. For more information about how address masks are configured in the server, review the following information for each server:

- ACI dns: bind rules under *Managing Access Control* (PingDirectory Server and PingDirectoryProxy Servers)
- `ds-auth-allowed-address`: *Adding Operational Attributes that Restrict Authentication* (PingDirectory Server)
- Connection Criteria: *Restricting Server Access Based on Client IP Address* (PingDirectory Server and PingDirectoryProxy Servers)
- Connection Handlers: restrict server access using Connection Handlers (Configuration Reference Guide for all PingData servers)

Configure the synchronization environment with dsconfig

The `dsconfig` tool can be used to configure any part of the PingDataSync Server, but will likely be used for more fine-grained adjustments. If configuring a Sync Pipe for the first time, use the `bin/create-sync-pipe-config` tool to guide through the necessary Sync Pipes creation steps.

Configure server groups with dsconfig interactive

In a typical deployment, onePingDataSync Server and one or more redundant failover servers are configured. Primary and secondary servers must have the same configuration settings to ensure proper operation. To enable this, assign all servers to a server group using the `dsconfig` tool. Any change to one server will automatically be applied to the other servers in the group.

Run the `dsconfig` command and set the global configuration property for server groups to `all-servers`. On the primary PingDataSync Server, do the following:

```
$ bin/dsconfig set-global-configuration-prop \
  --set configuration-server-group:all-servers
```

Updates to servers in the group are made using the `--applyChangeTo servergroup` option of the `dsconfig` command. To apply the change to one server in the group, use the `--applyChangeTo single-server` option. If additional servers are added to the topology, the `setup` tool will copy the configuration from the primary server to the new server(s).

Start the Global Sync Configuration with dsconfig interactive

After the Synchronization topology is configured, perform the following steps to start the Global Sync Configuration, which will use only those Sync Pipes that have been started:

1. On the `dsconfig` main menu, type the number corresponding to the Global Sync Configuration.

2. On the Global Sync Configuration menu, type the number corresponding to view and edit the configuration.

3. On the Global Sync Configuration Properties menu, type the number corresponding to setting the started property, and then follow the prompts to set the value to `TRUE`.

4. On the Global Sync Configuration Properties menu, type **f** to save and apply the changes.

Prepare external server communication

The `prepare-endpoint-server` tool sets up any communication variances that may occur between the PingDataSync Server and the external servers. Typically, directory servers can have different security settings, privileges, and passwords configured on the Sync Source that might reject the import of entries in the Sync Destination.

The `prepare-endpoint-server` tool also creates a Sync User Account and its privileges on all of the external servers (see About the Sync User Account for more detailed information). The `prepare-endpoint-server` tool verifies that the account has the proper privileges to access the `firstChangeNumber` and `lastChangeNumber` attributes in the root DSE entry so that it can access the latest changes. If the Sync User does not have the proper privileges, the PingDataSync Server displays a warning message, which is saved in the `logs/prepare-endpoint-server.log` file.

Note
If the synchronization topology was created using the `create-sync-pipe-config` tool, this command does not need to be run. It is already part of the `create-sync-pipe-config` process.

Perform the following steps to prepare thePingDataSync Server for external server communication:

1. Use the `prepare-endpoint-server` tool to prepare the directory server instances on the remote host for synchronization as a data source for the subtree, `dc=example,dc=com`. If the user account is not present on the external server, it will be created if a parent entry exists.

```
$ bin/prepare-endpoint-server \
  --hostname sun-ds1.example.com \
  --port 21389 \
  --syncServerBindDN "cn=Sync User,dc=example,dc=com" \
  --syncServerBindPassword secret \
  --baseDN "dc=example,dc=com" \
  --isSource
```

2. When prompted, enter the bind DN and password to create the user account. This step enables the change log database and sets the `changelog-maximum-age` property.
3. Repeat steps 1–2 for any other external source servers.
4. For the destination servers, repeat steps 2–3 and include the `--isDestination` option. If destination servers do not have any entries, a "Denied" message will display when creating the `cn=Sync User` entry.

```
$ bin/prepare-endpoint-server \
  --hostname PingIdentity-ds1.example.com \
  --port 33389 \
  --syncServerBindDN "cn=Sync User,cn=Root DNs,cn=config" \
  --syncServerBindPassword sync \
  --baseDN "dc=example,dc=com" \
  --isDestination
```

5. Repeat step 4 for any other destination servers.

Configuration with the dsconfig tool

The Ping Identity servers provide several command-line tools for management and administration. The command-line tools are available in the `bin` directory for UNIX or Linux

systems and `bat` directory for Microsoft Windows systems.

The `dsconfig` tool is the text-based management tool used to configure the underlying server configuration. The tool has three operational modes:

- Interactive mode
- Non-interactive mode
- Batch mode

The `dsconfig` tool also offers an offline mode using the `--offline` option, in which the server does not have to be running to interact with the configuration. In most cases, the configuration should be accessed with the server running in order for the server to give the user feedback about the validity of the configuration.

Each command-line utility provides a description of the subcommands, arguments, and usage examples needed to run the tool. View detailed argument options and examples by typing `--help` with the command.

```
$ bin/dsconfig --help
```

To list the subcommands for each command:

```
$ bin/dsconfig --help-subcommands
```

To list more detailed subcommand information:

```
$ bin/dsconfig list-log-publishers --help
```

Use dsconfig in interactive mode

Running `dsconfig` in interactive command-line mode provides a menu-driven interface for accessing and configuring the PingData server. To start `dsconfig` in interactive mode, run the tool without any arguments:

```
$ bin/dsconfig
```

Running the tool requires server connection and authentication information. After connection information is confirmed, a menu of the available operation types is displayed.

Use dsconfig in non-interactive mode

Non-interactive command-line mode provides a simple way to make arbitrary changes to the server, and to use administrative scripts to automate configuration changes. To make changes to multiple configuration objects at the same time, use batch mode.

The general format for the non-interactive command line is:

```
$ bin/dsconfig --no-prompt {globalArgs} {subcommand} {subcommandArgs}
```

The `--no-prompt` argument specifies non-interactive mode. The `{globalArgs}` argument provides a set of arguments that specify how to connect and authenticate to the server. Global arguments can be standard LDAP connection parameters or SASL connection parameters

depending on the implementation. The `{subcommand}` specifies which general action to perform. The following uses standard LDAP connections:

```
$ bin/dsconfig --no-prompt list-backends \
  --hostname server.example.com \
  --port 389 \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword password
```

The following uses SASL GSSAPI (Kerberos) parameters:

```
$ bin/dsconfig --no-prompt list-backends \
  --saslOption mech=GSSAPI \
  --saslOption authid=admin@example.com \
  --saslOption ticketcache=/tmp/krb5cc_1313 \
  --saslOption useticketcache=true
```

The `{subcommandArgs}` argument contains a set of arguments specific to the particular task. To always display the advanced properties, use the `--advanced` command-line option.

Note
Global arguments can appear anywhere on the command line. The subcommand-specific arguments can appear anywhere after the subcommand.

Use dsconfig batch mode

The `dsconfig` tool provides a batching mechanism that reads multiple invocations from a file and executes them sequentially. The batch file provides advantages over standard scripting by minimizing LDAP connections and JVM invocations that normally occur with each `dsconfig` call. Batch mode is the best method to use with setup scripts when moving from a development environment to test environment, or from a test environment to a production environment. The `--no-prompt` option is required with `dsconfig` in batch mode.

```
$ bin/dsconfig --no-prompt \
  --hostname host1 \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --batch-file /path/to/sync-pipe-config.txt
```

If a `dsconfig` command has a missing or incorrect argument, the command will fail and stop the batch process without applying any changes to the server. A `--batch-continue-on-error` option is available, which instructs `dsconfig` to apply all changes and skip any errors.

View the `logs/config-audit.log` file to review the configuration changes made to the server, and use them in the batch file. The batch file can have blank lines for spacing, and lines starting with a pound sign (#) for comments. The batch file also supports a "\" line continuation character for long commands that require multiple lines.

The PingDataSync Server also provides a `docs/sun-ds-compatibility.dsconfig` file for migrations from Sun/Oracle to Ping Identity PingDataSync Server machines.

Using the resync Tool

The `resync` tool provides bulk synchronization that can be used to verify the synchronization setup. The tool operates on a single Sync Pipe at a time, retrieves entries from the Sync Source in bulk, and compares the source entries with the corresponding destination entries. If destination entries are missing or attributes are changed, they are updated.

The command provides a `--dry-run` option that can be used to test the matches between the Sync Source and Destination, without committing any changes to the target topology. The `resync` tool also provides options to write debugging output to a log.

> **Note**
> The `resync` tool should be used for relatively small datasets. For large deployments, export entries from the Sync Source into an LDIF file, run the `bin/translate-ldif` tool to translate and filter the entries into the destination format, and then import the result LDIF file into the Sync Destination.

Use the `resync --help` command for more information and examples. Logging is located in `logs/tools/resync.log` and `logs/tools/resync-errors.log`. If necessary, the logging location can be changed with the `--logFilePath` option.

Testing Attribute and DN Maps

The `resync` tool can be used to test how attribute maps and DN maps are configured by synchronizing a single entry. If the `--logFilePath` and `--logLevel` options are specified, the `resync` tool generates a log file with details.

Use the`--dry-run` option and specify a single entry, such as `uid=user.0`. Any logging performed during the operation appears in `logs/tools/resync.log`.

```
$ bin/resync --pipe-name sun-to-ds-sync-pipe \
  --sourceSearchFilter "(uid=user.0)" \
  --dry-run \
  --logLevel debug
```

Verifying the Synchronization Configuration

The most common use for the `resync` tool is to test that the Sync Pipe configuration has been set up correctly. For example, the following procedure assumes that the configuration was set up with the Sync Source topology (two replicated Sun Directory servers) with 2003 entries; the Sync Destination topology (two replicated PingData PingDirectory Server) has only the base entry and the `cn=Sync User` entry. Both source and destination topologies have their LDAP Change Logs enabled. Because both topologies are not actively being updated, the `resync` tool can be run with one pass through the entries.

Use `resync` with the `--dry-run` option to check the synchronization configuration. The output displays a timestamp that can be tracked in the logs.

```
$ bin/resync --pipe-name sun-to-ds-sync-pipe \
  --numPasses 1 \
  --dry-run
```

```
Starting Pass 1

Status after completing all passes[20/Mar/2010:10:20:07 -0500]
---------------------------------
Source entries retrieved 2003
Entries missing 2002
Entries out-of-sync 1
Duration (seconds) 4

Resync completed in 4 s.

0 entries were in-sync, 0 entries were modified, 0 entries were created,
1 entries are still out-of-sync, 2002 entries are still missing, and
0 entries could not be processed due to an error
```

Populating an Empty Sync Destination Topology

The following procedure uses the `resync` tool to populate an empty Sync Destination topology for small datasets. For large deployments, use the `bin/translate-ldif`.

In this example, assume that the Sync Destination topology has only the base entry (`dc=example,dc=com`) and the `cn=Sync User` entry. Perform the following steps to populate an empty Sync Destination:

1. Run the `resync` command with the log file path and with the log level `debug`. Logging is located in `logs/tools/resync.log` and `logs/tools/resync-errors.log`.

   ```
   $ bin/resync --pipe-name sun-to-ds-sync-pipe \
     --numPasses 1 \
     --logLevel debug
   ```

2. Open the `logs/resync-failed-DNs.log` file in a text editor to locate the error and fix it. An entry cannot be created because the parent entry does not exist.

   ```
   # Entry '(see below)' was dropped because there was a failure at the
   resource:
   Failed to create entry uid=mlott,ou=People,dc=example,dc=com. Cause:
   LDAPException(resultCode=no such object, errorMessage='Entry
   uid=user.38,ou=People,dc=example,dc=com cannot be added because its parent
   entry ou=People,dc=example,dc=com does not exist in the server',
   matchedDN='dc=example,dc=com')
   (id=1893859385ResourceOperationFailedException.java:126 Build
   revision=4881)
   dn: uid=user.38,ou=People,dc=example,dc=com
   ```

3. Rerun the `resync` command. The command creates the entries in the Sync Destination topology that are present in the Sync Source topology.

   ```
   $ bin/resync --pipe-name sun-to-ds-sync-pipe
   ```

   ```
   ...(output from each pass)...

   Status after completing all passes[20/Mar/2016:14:23:33 -0500]
   ```

```
----------------------------
Source entries retrieved 160
Entries in-sync 156
Entries created 4
Duration (seconds) 11

Resync completed in 12 s.

156 entries were in-sync, 0 entries were modified, 4 entries were created,
0 entries are still out-of-sync, 0 entries are still missing, and 0
entries could not be processed due to an error
```

Note
If importing a large amount of data into an PingData PingDirectory Server, run `export-ldif` and `import-ldif` on the newly populated backend for most efficient disk space use. If needed, run `dsreplication initialize` to propagate the efficient layout to additional replicas.

Setting the Synchronization Rate

The `resync` command has a `--ratePerSecondFile` option that enables a specific synchronization rate. The option can be used to adjust the rate during off-peak hours, or adjust the rate based on measured loads for very long operations.

Note
The `resync` command also has a `--ratePerSecond` option. If this option is not provided, the tool operates at the maximum rate.

Run the `resync` tool first at 100 operations per second, measure the impact on the source servers, then adjust as desired. The file must contain a single positive integer number surrounded by white space. If the file is updated with an invalid number, the rate is not updated.

1. Create a text file containing the rate. The number must be a positive integer surrounded by white space.

   ```
   $ echo '100 ' > rate.txt
   ```

2. Run the `resync` command with the `--ratePerSecondFile` option.

   ```
   $ bin/resync --pipe-name "sun-to-ds-sync-pipe" \
     --ratePerSecondPath rate.txt
   ```

Synchronizing a Specific List of DNs

The `resync` command enables synchronizing a specific set of DNs that are read from a file using the `--sourceInputFile` option. This option is useful for large datasets that require faster processing by targeting individual base-level searches for each source DN in the file. If any DN fails (parsing, search, or process errors), the command creates an output file of the skipped entries (`resync-failed-DNs.log`), which can be run again.

The file must contain only a list of DNs in LDIF format with `dn:` or `dn::`. The file can include comment lines. All DNs can be wrapped and are assumed to be wrapped on any lines that begin with a space followed by text. Empty lines are ignored.

Small files can be created manually. For large files, use `ldapsearch` to create an LDIF file, as follows:

1. Run an `ldapsearch` command using the special OID "1.1" extension, which only returns the DNs in the DIT. For example, on the Sync Source directory server, run the following command:

```
$ bin/ldapsearch --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com \
  --baseDN dc=example,dc=com \
  --searchScope sub "(objectclass=*)" "1.1" > dn.ldif
```

2. Run the `resync` command with the file.

```
$ bin/resync --pipe-name "sun-to-ds-pipe" \
  --sourceInputFile dn.ldif
```

```
Starting pass 1
[20/Mar/2016:10:32:11 -0500]

-------------------------------------
Resync pass 1
Source entries retrieved 1999
Entries created 981
Current pass, entries processed 981
Duration (seconds) 10
Average ops/second 98
Status after completing all passes[20/Mar/2016:10:32:18 -0500]

---------------------------------
Source entries retrieved 2003
Entries created 2003
Duration (seconds) 16
Average ops/second 98
Resync completed in 16 s.
0 entries were in-sync, 0 entries were modified, 2003 entries were
created, 0 entries are still out-of-sync, 0 entries are still missing, and
0 entries could not be processed due to an error
```

3. View the `logs/tools/resync-failed-DNs.log` to determine skipped DNs. Correct the source DNs file, and rerun the `resync` command.

Using the realtime-sync Tool

The `bin/realtime-sync` tool controls starting and stopping synchronization globally or for individual Sync Pipes. The tool also provides features to set a specific starting point for real-time synchronization.

To see the current status of real-time synchronization, view the monitor properties: `num-sync-ops-in-flight`, `num-ops-in-queue`, and `source-unretrieved-changes`. For example, use `ldapsearch` to view a specific Sync Pipe's monitor information:

```
$ bin/ldapsearch --baseDN cn=monitor \
--searchScope sub "(cn=Sync PipeMonitor: PIPE_NAME)"
```

The Stats Logger can also be used to view status. See the *Ping IdentityPingDirectory Server Administration Guide* for details.

Starting Real Time Synchronization Globally

The `realtime-sync` tool assumes that the synchronization topology is configured correctly.

Perform the following steps to start real time synchronization globally:

1. Run the tool from the `<server-root>/bin` directory. This example assumes that a single Sync Pipe called "dsee-to-ds-sync-pipe" exists.

   ```
   $ bin/realtime-sync start --pipe-name "dsee-to-ds-sync-pipe" \
     --port 389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret
   ```

2. If more than one Sync Pipe is configured, specify each using the `--pipe-name` option. The following example starts synchronization for a bidirectional synchronization topology.

   ```
   $ bin/realtime-sync start --pipe-name "Sun DS to DS" \
     --pipe-name "DS to Sun DS" \
     --port 389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret
   ```

Starting or Pausing Synchronization

Pause or start synchronization by using the `start` and `stop` subcommands. If synchronization is stopped and then restarted, changes made at the Sync Source while synchronization was stopped will still be detected and applied. Synchronization for individual Sync Pipes can be started or stopped using the `--pipe-name` argument. If the `--pipe-name` argument is omitted, then synchronization is started or stopped globally.

The following command stops all Sync Pipes:

```
$ bin/realtime-sync stop --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --no-prompt
```

If a topology has two Sync Pipes, Sync Pipe1 and Sync Pipe2, the following command stops Sync Pipe1.

```
$ bin/realtime-sync stop --pipe-name "Sync Pipe1" \
  --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret --no-prompt
```

Setting Startpoints

Startpoints instruct the Sync Pipe to ignore all changes made prior to the current time. Once synchronization is started, only changes made after this command is run will be detected at the Sync Source and applied at the Sync Destination.

The `set-startpoint` subcommand is often run during the initial setup prior to starting realtime synchronization. It should be run prior to initializing the data in the Sync Destination.

The `set-startpoint` subcommand can start synchronization at a specific change log number, or back at a state that occurred at a specific time. For example, synchronization can start 10 minutes prior to the current time.

Perform the following steps to set a synchronization startpoint:

1. If started, stop the synchronization topology globally with the following command:

   ```
   $ bin/realtime-sync stop --pipe-name "Sync Pipe1" \
     --port 389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret \
     --no-prompt
   ```

2. Set the startpoint for the synchronization topology. Any changes made before setting this command will be ignored.

   ```
   $ bin/realtime-sync set-startpoint --pipe-name "Sync Pipe1" \
     --port 389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret \
     --no-prompt \
     --beginning-of-changelog
   ```

   ```
   Set StartPoint task 2011072109564107 scheduled to start immediately
   [21/Jul/2016:09:56:41 -0500] severity="INFORMATION" msgCount=0
   msgID=1889535170
   message="The startpoint has been set for Sync Pipe 'Sync Pipe1'.
   Synchronization will resume from the last change number in the Sync
   Source"
   Set StartPoint task 2011072109564107 has been successfully completed
   ```

Restarting Synchronization at a Specific Change Log Event

Perform the following steps to restart synchronization at a specific event:

1. Search for a specific change log event from which to restart the synchronization state. On one of the endpoint servers, run `ldapsearch` to search the change log.

```
$ bin/ldapsearch -p 1389
--bindDN "uid=admin,dc=example,dc=com" \
--bindPassword secret \
--baseDN cn=changelog \
--dontWrap
```

```
"(objectclass=*)"
dn: cn=changelog
objectClass: top
objectClass: untypedObject
cn: changelog

dn: changeNumber=1,cn=changelog
objectClass: changeLogEntry
objectClass: top
targetDN: uid=user.13,ou=People,dc=example,dc=com
changeType: modify
changes::
cmVwbGFjZTogcm9vbU51bWJlcgpyb29tTnVtYmVyOiAwMTM4Ci0KcmVwbGFjZTogbW9kaW
ZpZXJzTmFtZQptb2RpZmllcnNOYW1lOiBjbj1EaXJlY3RvcnkgTWFuYWdlcixjbj1Sb290
IEROcyxjbj1jb25maWcKLQpyZXBsYWNlOiBkcy11cGRhdGUtdGltZQpkcy11cGRhdGUtdG
ltZTo6IEFBQUJKZ25OWlUwPQotCgA=
changenumber: 1
          ... (more output)
dn: changeNumber=2329,cn=changelog
objectClass: changeLogEntry
objectClass: top
targetDN: uid=user.49,ou=People,dc=example,dc=com
changeType: modify
changes::
cmVwbGFjZTogcm9vbU51bWJlcgpyb29tTnVtYmVyOiAwNDMzCi0KcmVwbGFjZTogbW9kaW
ZpZXJzTmFtZQptb2RpZmllcnNOYW1lOiBjbj1EaXJlY3RvcnkgTWFuYWdlcixjbj1Sb290
IEROcyxjbj1jb25maWcKLQpyZXBsYWNlOiBkcy11cGRhdGUtdGltZQpkcy11cGRhdGUtdG
ltZTo6IEFBQUJKZ25PMC84PQotCgA=
changenumber: 2329
```

2. Restart synchronization from change number 2329 using the `realtime-sync` tool. Any event before this change number will not be synchronized to the target endpoint.

```
$ bin/realtime-sync set-startpoint \
  --change-number 2329 \
  --pipe-name "Sync Pipe 1" \
  --bindPassword secret \
  --no-prompt
```

Changing the Synchronization State by a Specific Time Duration

The following command will start synchronizing data at the state that occurred 2 hours and 30 minutes prior to the current time on External Server 1 for Sync Pipe 1. Any changes made

before this time will not be synchronized. Specify days (d), hours (h), minutes (m), seconds (s), or milliseconds (ms).

Use `realtime-sync` with the `--startpoint-rewind` option to set the synchronization state and begin synchronizing at the specified time.

```
$ bin/realtime-sync set-startpoint \
  --startpoint-rewind 2h30m \
  --pipe-name "Sync Pipe 1" \
  --bindPassword secret \
  --no-prompt
```

Scheduling a Realtime Sync as a Task

The `realtime-sync` tool features both an offline mode of operation as well as the ability to schedule an operation to run within the PingDataSync Server's process. To schedule an operation, supply LDAP connection options that allow this tool to communicate with the server through its task interface. Tasks can be scheduled to run immediately or at a later time. Once scheduled, tasks can be managed using the `manage-tasks` tool.

Perform the following steps to schedule a synchronization task:

1. Use the `--start` option with the `realtime-sync` command to schedule a start for the synchronization topology. The following command will set the start time at July 21, 2016 at 12:01:00 AM. The scheduled task can be stopped with the `--stop` subcommand.

   ```
   $ bin/realtime-sync set-startpoint \
     --pipe-name "sun-to-ds-sync-pipe" \
     --port 389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret \
     --start 20150721000100 \
     --no-prompt
   ```

   ```
   Set StartPoint task 2009072016103807 scheduled to start Jul 21, 2016
   12:01:00 AM CDT
   ```

2. Run the `manage-tasks` tool to manage or cancel the task.

   ```
   $ bin/manage-tasks --port 7389 \
     --bindDN "uid=admin,dc=example,dc=com" \
     --bindPassword secret
   ```

Configuring the PingDirectory Server Backend for Synchronizing Deletes

The PingDirectory Server’s change log backend's `changelog-deleted-entry-include-attribute` property specifies which attributes should be recorded in the change log entry during a DELETE operation. Normally, the PingDataSync Server cannot correlate a deleted entry to the entry on the destination. If a Sync Class is configured with a filter, such as

"`include-filter: objectClass=person`," the `objectClass` attribute must be recorded in the change log entry. Special correlation attributes (other than DN), will also need to be recorded on the change log entry to be properly synchronized at the endpoint server.

On each PingDirectory Server backend, use the `dsconfig` command to set the property.

```
$ bin/dsconfig set-backend-prop --backend-name changelog \
  --set changelog-deleted-entry-include-attribute:objectClass
```

If the destination endpoint is an Oracle/Sun DSEE (or Sun DS) server, the Sun DSEE server does not store the value of the user deleting the entry, specified in the modifiers name attribute. It only stores the value of the user who last modified the entry while it still existed.

To set up a Sun DSEE destination endpoint to record the user who deleted the entry, use the Ping Identity Server SDK to create a plug-in as follows:

1. Update the Sun DSEE schema to include a `deleted-by-sync auxiliary` objectclass. It will only be used as a marker objectclass, and not require or allow additional attributes to be present on an entry.
2. Update the Sun DSEE Retro Change Log Plug-in to include the `deleted-by-sync auxiliary` object class as a value for the `deletedEntryAttrs` attribute.
3. Write an `LDAPSyncDestinationPlugin` script that in the `preDelete()` method modifies the entry that is being deleted to include the `deleted-by-sync` objectclass.
4. Update the Sync Class filter that is excluding changes by the Sync User to also include `(!(objectclass=deleted-by-sync))`.

Configure DN maps

Similar to attribute maps, DN maps define mappings when destination DNs differ from source DNs. These differences must be resolved using DN maps in order for synchronization to successfully take place. For example, the Sync Source could have a DN in the following format:

```
uid=jdoe,ou=People,dc=example,dc=com
```

While the Sync Destination could have the standard X.500 DN format.

DN mappings allow the use of wild cards for DN transformations. A single wild card (`*`) matches a single RDN component and can be used any number of times. The double wild card (`**`) matches zero or more RDN components and can be used only once.

> **Note**
> If a literal '*' is required in a DN then it must be escaped as `'\2A'`.

The wild card values can be used in the `to-dn-pattern` attribute using `{1}` to replace their original index position in the pattern, or `{attr}` to match an attribute value. For example:

```
*,**,dc=com->{1},ou=012,o=example,c=us
```

For example, using the DN, `uid=johndoe,ou=People,dc=example,dc=com`, and mapping to the target DN, `uid=johndoe,ou=012,o=example,c=us`:

- "*" matches one RDN component, uid=johndoe
- "**" matches zero or more RDN components, ou=People,dc=example
- "dc=com" matches dc=com in the DN.

The DN is mapped to the {1},ou=012,o=example,c=us. "{1}" substitutes the first wildcard element "uid=johndoe", so that the DN is successfully mapped to:

```
uid=johndoe,ou=012,o=example,c=us
```

Regular expressions and attributes from the user entry can also be used in the to-dn-pattern attribute. For example, the following expression constructs a value for the uid attribute, which is the RDN, out of the initials (first letter of givenname and sn) and the employee ID (the eid attribute) of a user.

```
uid={givenname:/^(.)(.*)/$1/s}{sn:/^(.)(.*)/$1/s}{eid},{2},o=example
```

Note
The PingDataSync Server automatically validates any DN mapping prior to applying the configuration.

Configuring a DN Map Using dsconfig

A DN map can be configured using dsconfig, either with the interactive DN Map menu, or from the command line.

Perform the following to configure a DN map:

1. Use dsconfig to create a DN map for the PingDataSync Server.

```
$ bin/dsconfig --no-prompt create-dn-map \
  --map-name nested-to-flattened \
  --set "from-dn-pattern:*,*,dc=example,dc=com" \
  --set "to-dn-pattern:uid={givenname:/^(.)(.*)/\$1/s}{sn:/^(.)(.*)/\$1/s}
(eid},{2},o=example" \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret
```

2. After DN mappings are configured, add the new DN map to a new Sync Class or modify an existing Sync Class.

```
$ bin/dsconfig --no-prompt set-sync-class-prop \
  --pipe-name test-sync-pipe \
  --class-name test-sync-class \
  --set dn-map:test-dn-map \
  --port 389 --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret
```

Configure synchronization with JSON attribute values

The PingDataSync Server supports synchronization of attributes that hold JSON objects. The following scenarios are supported:

- **Synchronizing a JSON attribute to another JSON attribute** - A subset of fields can be synchronized, optionally retaining fields that appear at the destination but not at the source.
- **Synchronizing a JSON attribute to a non-JSON attribute** - A single field of the JSON value can be extracted with a constructed attribute mapping.
- **Synchronizing a non-JSON attribute to a JSON attribute** - The source value can be escaped so that it ensures the JSON value is properly formatted.
- **Attribute correlation** - A JSON field can be used when correlating a destination entry with a source entry.

The following examples show configuration scenarios based on the LDAP `ubidEmailJSON` attribute, which has fields of `value`, `type`, `primary`, and `verified`:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home",
                "primary" : true,
                "verified" : true}
```

Synchronize ubidEmailJSON fully

If a source JSON attribute value should be synchronized fully to a destination JSON attribute value, no special configuration is required.

Synchronize a subset of fields from the source attribute

For example, the following configuration can be used to synchronize the `value` and `type` fields of `ubidEmailJSON` from the source to a destination. To synchronize this source value:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home",
                "primary" : true}
```

to this value at the destination:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home"}
```

A JSON Attribute configuration object must be created and associated with the Sync Class. This can be done by either explicitly including the fields to synchronize:

```
$ bin/dsconfig create-json-attribute --pipe-name "A to B" \
  --class-name Users \
  --attribute-name ubidEmailJSON \
  --set include-field:type \
  --set include-field:value
```

Or by excluding the fields that should not be synchronized:

```
$ bin/dsconfig create-json-attribute \
  --pipe-name "A to B" \
```

```
  --class-name Users \
  --attribute-name ubidEmailJSON \
  --set exclude-field:preferred \
  --set exclude-field:verified
```

If the destination is prepared to only handle a specific subset of fields, then list the fields to include. However, if only a small, known subset of fields from the source should be excluded, then `exclude-field` could be used. In this example, the destination data for the `ubidEmailJSON` attribute will always be a subset of the full data.

Note

A Sync Class can be configured to exclude certain attributes from synchronization. Creating a regular attribute mapping will override this setting, and the attribute will be synchronized. Creating a JSON attribute mapping does not override this setting, and the JSON attribute will not be synchronized. A JSON attribute is not a traditional attribute mapping. It only includes information on the destination attribute name. To work around this, the attribute either needs to be mapped from a source attribute, or have its value constructed.

The following scenario illustrates how the destination can include additional fields that are not present at the source.

Retain destination-only fields

To synchronize changes to the source fields while preserving the value of the the `verified` field of the `ubidEmailJSON` attribute at the destination, configure the JSON Attribute as follows:

```
$ bin/dsconfig create-json-attribute \
  --pipe-name "A to B" \
  --class-name Users \
  --attribute-name ubidEmailJSON \
  --set id-field:value \
  --set exclude-field:verified
```

The `verified` field is excluded and `value` is chosen to correlate destination values with source values. For example, given that the source and destination `value` fields match, if the source initially contained:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home"}
```

and the destination contained:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home",
                "verified" : true},
```

if the source changed to:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "other"}
```

then the destination would change to:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "other",
                "verified" : true}
```

However, if the source changed to:

```
ubidEmailJSON: {"value" : "john.smith@example.com",
                "type" : "home"}
```

then the destination would be updated to:

```
ubidEmailJSON: {"value" : "john.smith@example.com",
                "type" : "home"}
```

The `verified` field has been dropped because this logically represents a new JSON object rather than an update of an existing one.

Synchronize a field of a JSON attribute into a non-JSON attribute

If the source stores:

```
ubidEmailJSON: {"value" : "jsmith@example.com",
                "type" : "home"}
```

but the destination stores:

```
mail: jsmith@example.com
```

To synchronize changes between these systems, a constructed attribute mapping must be configured:

```
$ bin/dsconfig create-attribute-mapping \
  --map-name "Attribute Map" \
  --mapping-name mail \
  --type constructed \
  --set "value-pattern:{ubidEmailJSON.value}"
```

The `value-pattern` syntax allows attributes to be referenced by placing them in `{}`. JSON fields within the attribute can be referenced by using the syntax `{attribute.field}`. See this property in the Configuration Refernce guide, or `dsconfig` tool command help for more information.

After the "Attribute Map" is created, it can be referenced from the Sync Class:

```
$ bin/dsconfig set-sync-class-prop
  --pipe-name "A to B" \
  --class-name Users \
  --set "attribute-map:Attribute Map"
```

Note

While LDAP attribute names are not case sensitive, the JSON field names are. By default, errors related to attribute mapping are not logged. To enable error logging, configure the Debug Logger with the following:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Debug Logger" \
  --set enabled:true
```

```
$ bin/dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name com.unboundid.directory.sync.mapping \
  --set debug-level:warning
```

Synchronize a non-JSON attribute into a field of a JSON attribute

This is the reverse of the previous example. Suppose the source stores:

```
mail: jsmith@example.com
```

but the destination stores:

```
ubidEmailJSON: {"value" : "jsmith@example.com"}
```

A constructed attribute mapping can be used in this case as well:

```
$ bin/dsconfig create-attribute-mapping \
  --map-name "Attr Map" \
  --mapping-name ubidEmailJSON \
  --type constructed \
  --set 'value-pattern:{{"value" : "{mail:jsonEscape}"}}'
```

When constructing the value, the following are important:

- Double curly brackets (`{{ }}`) are necessary to represent a single curly bracket (`{ }`) in the output. These brackets are typically used to reference attribute values.
- Attribute values that appear within a JSON attribute should be escaped using the `:jsonEscape` modifier. This prevents values that include quotes like `'"John Smith" <jsmith@example.com>'` from producing invalid JSON.

In this example, a JSON Attribute object should be created since the destination value is likely to be augmented with additional information:

```
$ bin/dsconfig create-json-attribute \
  --pipe-name "A to B" \
  --class-name Users \
  --attribute-name ubidEmailJSON \
  --set id-field:value \
  --set include-field:value
```

Correlating attributes based on JSON fields

When the destination of a Sync Pipe is a Ping Directory Server or PingDirectoryProxy Server, source and destination entries can be correlated by referencing a field within a JSON attribute. In the following example, source entries will be matched with destination entries that have the same `value` field within the `ubidEmailJSON` value.

```
$ bin/dsconfig set-sync-class-prop \
  --pipe-name "A to B" \
  --class-name Users \
  --set destination-correlation-attributes:ubidEmailJSON.value
```

This could also be used with the previous example, which does not store `ubidEmailJSON.value` at the source but maps into it before correlating at the destination.

Configure fractional replication

The PingDataSync Server supports fractional replication to any server type. For example, if a replica only performs user authentications, the PingDataSync Server can be configured to propagate only the `uid` and `userpassword` password policy attributes, reducing the database size at the replica and the network traffic needed to keep this servers synchronized.

The following example configures a fractional replication, where the `uid` and `userPassword` attributes of all entries in the source topology are synchronized to the destination topology. Because the `uid` and `userPassword` attributes are present, the `objectclass` attribute must also be synchronized. The example assumes that a PingDataSync Server and external servers are configured and a Sync Pipe and Sync Class are defined, but realtime synchronization or bulk resync have not been performed.

Perform the following steps to configure fractional replication from the `dsconfig` interactive menu:

1. On the main menu, type the number corresponding to Sync Classes.
2. On the Sync Class menu, type the number corresponding to viewing and editing an existing Sync Class. Assume that only one Sync Class has been defined.
3. Verify that the Sync Pipe and Sync Class exist.
4. On the Sync Class Properties menu, type the number specifying the source LDAP filter (`include-filter` property) that defines which source entries are to be included in the Sync Class.
5. On the Include-Filter Property menu, type the number corresponding to adding a filter value. For this example, type (`objectclass=person`). When prompted, enter another filter. Press **Enter** to continue. On the menu, enter 1 to use the value when specifying it.
6. On the Sync Class Properties menu, type the number corresponding to the `auto-mapped-source-attribute` property. Change the value from "`-all-`" to a specific attribute, so that only the specified attribute is automatically mapped from the source topology to the destination topology.
7. On the Auto-Mapped-Source-Attribute Property menu, type the number corresponding to adding the source attributes that will be automatically mapped to the destination attributes of the same name. When prompted, enter each attribute, and then press **Enter**.

```
Enter another value for the 'auto-mapped-source-attribute' property
[continue]: uid
Enter another value for the 'auto-mapped-source-attribute' property
[continue]: userPassword
Enter another value for the 'auto-mapped-source-attribute' property
[continue]: objectclass
```

```
Enter another value for the 'auto-mapped-source-attribute' property
[continue]:
```

8. On the Auto-Mapped-Source-Attribute Property menu, type the number corresponding to removing one or more values. In this example, remove the "`-all-`" value, so that only the `objectclass`, `uid`, and `userPassword` attributes are only synchronized.
9. On the Auto-Mapped-Source-Attribute Property menu, press **Enter** to accept the values.
10. On the Sync Class Properties menu, type the number corresponding to excluding some attributes from the synchronization process. When using the `objectclass=person` filter, the `cn`, `givenName`, and `sn` attributes must be excluded. Enter the option to add one or more attributes, and then add each attribute to exclude on the `excluded-auto-mapped-source-attributes` Property menu. For this example, exclude the `cn`, and `sn` attributes, which are required attributes of the `Person` objectclass. Also exclude the `givenName` attribute, which is an optional attribute of the `inetOrgPerson` objectclass.

```
Enter another value for the 'excluded-auto-mapped-source-attributes'
property
[continue]: givenName
Enter another value for the 'excluded-auto-mapped-source-attributes'
property
[continue]: sn
Enter another value for the 'excluded-auto-mapped-source-attributes'
property
[continue]:
```

11. On the Excluded-Auto-Mapped-Source-Attributes Property menu, press **Enter** to accept the changes.

Note
If using `entryUUID` as a correlation attribute, some attribute uniqueness errors may occur while using the `resync` tool. Either set the `excluded-auto-mapped-source-attributes` property value to `entryUUID` on the Sync Class configuration menu, or run `resync` with the `--excludeDestinationAttr` entryUUID argument.

12. On the Sync Class Properties menu, review the configuration and accept the changes.
13. On the server instances in the destination topology, turn off schema checking to avoid a schema error that occurs when the required attributes in the `Person` object class are not present. Make sure that the global configuration property for the `server-group` is set to `all-servers`. Use the following command to turn off schema checking on all of the servers in the group.

```
$ bin/dsconfig --no-prompt set-global-configuration-prop \
  --set check-schema:false \
  --applyChangeTo server-group \
  --port 3389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret
```

14. Run `bin/resync` to load the filtered data from the source endpoint to the target endpoint.

```
$ bin/resync --pipe-name "test-sync-pipe" \
  --numPasses 3
```

15. Run `bin/realtime-sync` to start synchronization.

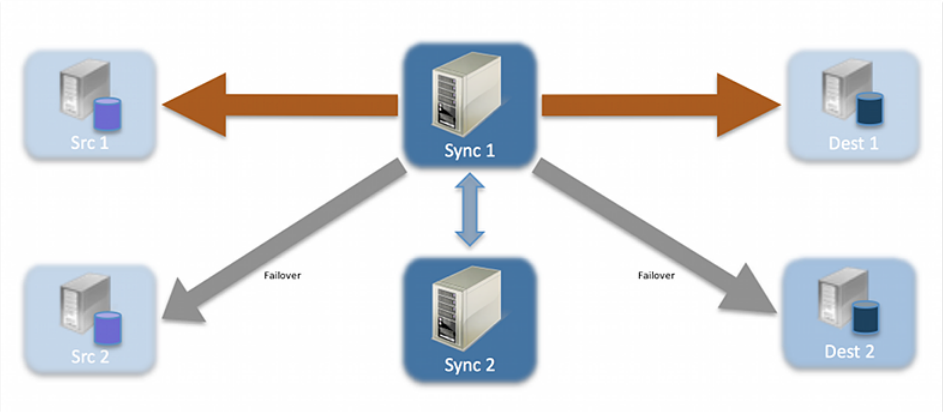
```
$ bin/realtime-sync start --pipe-name "test-sync-pipe" \
  --port 7389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --no-prompt
```

Configure failover behavior

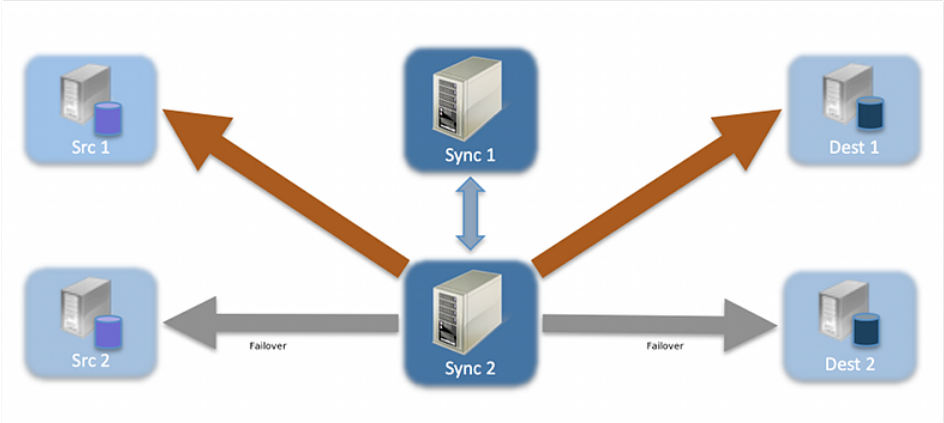
The following illustrates a simplified synchronization topology with a single failover server on the source, destination, and PingDataSync Server, respectively. The gray lines represent possible failover connections in the event the server is down. The external servers are prioritized so that src1 has higher priority than src2; dest1 has higher priority than dest2.

The main PingDataSync Server and its redundant failover instance communicate with each other over LDAP and bind using `cn=IntraSync User,cn=Root DNs,cn=config`. The servers run periodic health checks on each other and share information on all changes that have been processed. Whenever the failover server loses connection to the main server, it assumes that the main server is down and begins processing changes from the last known change. Control reverts back to the main server once it is back online.

Unlike the PingDataSync Server servers, the external servers and their corresponding failover server(s) do not run periodic health checks. If an external server goes offline, the failover server will receive transactions and remain connected to the PingDataSync Server until the Sync Pipe is restarted, regardless of if the main external server comes back online.



The PingDataSync Server in a Simplified Setup



The PingDataSync Server Sample Failover

Conditions that trigger immediate failover

Immediate failover occurs when the PingDataSync Server receives one of the following error codes from an external server:

- BUSY (51)
- UNAVAILABLE (52)
- SERVER CONNECTION CLOSED (81)
- CONNECT ERROR (91)

If the PingDataSync Server attempts a write operation to a target server that returns one of these error codes, the PingDataSync Server will automatically fail over to the next highest prioritized server instance in the target topology, issue an alert, and then reissue the retry attempt. If the operation is unsuccessful for any reason, the server logs an error.

Failover server preference

The PingDataSync Server supports endpoint failover, which is configurable using the `location` property on the external servers. By default, the PingDataSync Server prefers to connect to and failover to endpoint servers in the same location as itself. If no location settings are configured, the PingDataSync Server will iterate through the configured list of external servers on the Sync Source and Sync Destination when failing over.

The PingDataSync Server does not do periodic health checks and will not failover to a preferred server automatically. Because of the cost of sync failover, it will always stay connected to a given server until that server stops responding or until the Sync Pipe is restarted. When a failover does happen, it will always go back to the most preferred server (optionally using location settings to determine this) and work its way down the list. The following provides an example configuration of external servers:

```
austin1.server.com:1389
london1.server.com:2389
boston1.server.com:3389
```

```
austin2.server.com:4389
boston2.server.com:5389
london2.server.com:6389
```

If the austin1 server were to become unavailable, the PingDataSync Server will automatically pick up changes on the next server on the list, london1. If london1 is also down, then the next server, boston1 will be picked up. Once the PingDataSync Server iterates through the list, it returns to the top of the list. So, if the PingDataSync Server is connected to london2 and it goes down, it will fail over to austin1.

To minimize WAN traffic, configure the `location` property for each external server using the `dsconfig` command on the PingDataSync Server. Assume that PingDataSync Server has its own `location` property (set in the Global Configuration) set to "austin."

```
austin1.server.com:1389 location=austin
london1.server.com:2389 location=london
boston1.server.com:3389 location=boston
austin2.server.com:4389 location=austin
boston2.server.com:5389 location=boston
london2.server.com:6389 location=london
```

With the `location` property set for each server, the PingDataSync Server gets its changes from server austin1. If austin1 goes down, the PingDataSync Server will pick up changes from austin2. If austin2 goes down, the server will iterate through the rest of the list in the order it is configured.

The `location` property has another sub-property, `preferred-failover-location` that specifies a set of alternate locations if no servers in this location are available. If multiple values are provided, servers are tried in the order in which the locations are listed. The `preferred-failover-location` property provides more control over the failover process and allows the failover process to jump to a specified location. Care must be used so that circular failover reference does not take place. Here is an example configuration:

```
austin1.server.com:1389 location=austin preferred-failover-location=boston
london1.server.com:2389 location=london preferred-failover-location=austin
boston1.server.com:3389 location=boston preferred-failover-location=london
austin2.server.com:4389 location=austin preferred-failover-location=boston
boston2.server.com:5389 location=boston preferred-failover-location=austin
london2.server.com:6389 location=london preferred-failover-location=london
```

The PingDataSync Server will respect the `preferred-failover-location` if it cannot find any external servers in the same location as itself, it will look for any external servers in its own `preferred-failover-location`. In this example, when austin1 becomes unavailable, it will fail over to austin2 because they are in the same location. If austin2 is unavailable, it will fail over to boston1, which is in the `preferred-failover-location` of the PingDataSync Server. If boston1 in unavailable, the PingDataSync Server will fail over to boston2, and finally, it will try the london1 and london2 servers.

Configuration properties that control failover behavior

There are four important advanced properties to fine tune the failover mechanism:

- `max-operation-attempts` (Sync Pipe)
- `response-timeout` (source and destination endpoints)
- `max-failover-error-code-frequency` (source and destination endpoints)
- `max-backtrack-replication-latency` (source endpoints only)

These properties apply to the following LDAP error codes:

LDAP Error Codes

Error Code	Description
ADMIN_LIMIT_EXCEEDED (11)	Indicates that processing on the requested operation could not continue, because an administrative limit was exceeded.
ALIAS_DEREFERENCING_ PROBLEM (36)	Indicates that a problem was encountered while attempting to dereference an alias for a search operation.
CANCELED (118)	Indicates that a cancel request was successful, or that the specified operation was canceled.
CLIENT_SIDE_LOCAL_ERROR (82)	Indicates that a local (client-side) error occurred.
CLIENT_SIDE_ENCODING_ERROR (83)	Indicates that an error occurred while encoding a request.
CLIENT_SIDE_DECODING_ERROR (84)	Indicates that an error occurred while decoding a request.
CLIENT_SIDE_TIMEOUT (85)	Indicates that a client-side timeout occurred.
CLIENT_SIDE_USER_CANCELLED (88)	Indicates that a user canceled a client-side operation.
CLIENT_SIDE_NO_MEMORY (90)	Indicates that the client could not obtain enough memory to perform the requested operation.
CLIENT_SIDE_CLIENT_LOOP (96)	Indicates that a referral loop is detected.
CLIENT_SIDE_REFERRAL_LIMIT_ EXCEEDED (97)	Indicates that the referral hop limit was exceeded.
DECODING_ERROR (84)	Indicates that an error occurred while decoding a response.
ENCODING_ERROR (83)	Indicates that an error occurred while encoding a response.
INTERACTIVE_TRANSACTION_ ABORTED (30221001)	Indicates that an interactive transaction was aborted.
LOCAL_ERROR (82)	Indicates that a local error occurred.
LOOP_DETECT (54)	Indicates that a referral or chaining loop was detected while processing a request.
NO_MEMORY (90)	Indicates that not enough memory could be obtained to perform the requested operation.
OPERATIONS_ERROR (1)	Indicates that an internal error prevented the operation from being processed properly.
OTHER (80)	Indicates that an error occurred that does not fall into any of the other categories.
PROTOCOL_ERROR (2)	Indicates that the client sent a malformed or illegal

LDAP Error Codes

Error Code	Description
	request to the server.
TIME_LIMIT_EXCEEDED (3)	Indicates that a time limit was exceeded while attempting to process the request.
TIMEOUT (85)	Indicates that a timeout occurred.
UNWILLING_TO_PERFORM (53)	Indicates that the server is unwilling to perform the requested operation.

The max-operation-attempts property

The `max-operation-attempts` property (part of the Sync Pipe configuration) specifies the maximum number of times to retry a synchronization operation that fails for reasons other than the Sync Destination being busy, unavailable, server connection closed, or connect error.

To change the default number of retries, use `dsconfig` in non-interactive mode to change the `max-operation-attempts` value on the Sync Pipe object. The following command changes the number of maximum attempts from five (default) to four.

```
$ bin/dsconfig set-sync-pipe-prop \
  --pipe-name "Test Sync Pipe" \
  --set max-operation-attempts:4
```

The response-timeout property

The `response-timeout` property specifies how long the PingDataSync Server should wait for a response from a search request to a source server before failing with LDAP result code 85 (client-side timeout). When a client-side timeout occurs, the Sync Source will retry the request according to the `max-failover-error-code-frequency` property before failing over to a different source server and performing the retry. The total number of retries will not exceed the `max-operation-attempts` property defined in the Sync Pipe configuration. A value of zero indicates that there should be no client-side timeout. The default value is one minute.

Assuming a bidirectional topology, the property can be set with `dsconfig` on the Sync Source and Sync Destination, respectively.

```
$ bin/dsconfig set-sync-source-prop \
  --source-name src \
  --set "response-timeout:8 s"
```

```
$ bin/dsconfig set-sync-destination-prop \
  --destination-name U4389 \
  --set "responsetimeout:9 s"
```

The max-failover-error-code-frequency property

The `max-failover-error-code-frequency` property (part of the Sync Source configuration) specifies the maximum time period that an error code can re-appear until it fails over to

another server instance. This property allows the retry logic to be tuned, so that retries can be performed once on the same server before giving up and trying another server. The value can be set to zero if there is no acceptable error code frequency and failover should happen immediately. It can also be set to a very small value (such as 10 ms) if a high frequency of error codes is tolerable. The default value is three minutes.

To change the `max-failover-error-code-frequency` property, use `dsconfig` in non-interactive mode to change the property on the Sync Source object. The following command changes the frequency from three minutes to two minutes.

```
$ bin/dsconfig set-sync-source-prop \
  --source-name source1 \
  --set "max-failover-error-code-frequency:2 m"
```

The max-backtrack-replication-latency property

The `max-backtrack-replication-latency` property (part of the Sync Source configuration) sets the time period that an PingDataSync Server will look for missed changes in the change log due to replication delays. The property should be set to a conservative upper-bound of the maximum replication delay between two servers in the topology. A value of zero implies that there is no limit on the replication latency. The default value is two hours. ThePingDataSync Server stops looking in the change log once it finds a change that is older than the maximum replication latency than the last change it processed on the failed server.

For example, after failing over to another server, the PingDataSync Server must look through the new server's change log to find the equivalent place to begin synchronizing changes. Normally, the PingDataSync Server can successfully backtrack with only a few queries of the directory, but in some situations, it might have to look further back through the change log to make sure that no changes were missed. Because the changes can come from a variety of sources (replication, synchronization, and over LDAP), the replicated changes between directory servers are interleaved in each change log. When the PingDataSync Server fails over between servers, it has to backtrack to figure out where synchronization can safely pick up the latest changes.

Backtracking occurs until the following:

- The server determines that there is no previous change log state available for any source servers, so it must start at the beginning of the change log.
- The server finds the last processed replication change sequence number (CSN) from the last time it was connected to that replica, if at all. This process is similar to the `set-startpoint` functionality on the `realtime-sync` tool.
- The server finds the last processed replication CSN from every replica that has produced a change so far, and it determines that each change entry in the next oldest batch of changes has already been processed.
- The server finds a change that is separated by more than a certain duration (specified by the `max-backtrack-replication-latency` property) from the most recently processed change.

The following command changes the maximum backtracking from two hours to three hours.

```
$ bin/dsconfig set-sync-source-prop \
  --source-name source1 \
  --set "max-backtrack-replication-latency:3 h"
```

Configure traffic through a load balancer

If a PingData server is sitting behind an intermediate HTTP server, such as a load balancer, a reverse proxy, or a cache, it will log incoming requests as originating with the intermediate HTTP server instead of the client that actually sent the request. If the actual client's IP address should be recorded to the trace log, enable `X-Forwarded-*` handling in both the intermediate HTTP server and the PingData server. See the product documentation for the device type. For PingData servers:

- Edit the appropriate Connection Handler object (HTTPS or HTTP) and set `use-forwarded-headers` to `true`.
- When `use-forwarded-headers` is set to `true`, the server will use the client IP address and port information in the `X-Forwarded-*` headers instead of the address and port of the entity that's actually sending the request, the load balancer. This client address information will show up in logs where one would normally expect it to show up, such as in the `from` field of the HTTP REQUEST and HTTP RESPONSE messages.

Configure authentication with a SASL external certificate

By default, the PingDataSync Server authenticates to the PingDirectory Server using LDAP simple authentication (with a bind DN and a password). However, the PingDataSync Server can be configured to use SASL EXTERNAL to authenticate to the PingDirectory Server with a client certificate.

Note
This procedure assumes that PingDataSync Server instances are installed and configured to communicate with the backend PingDirectory Server instances using either SSL or StartTLS.

After the servers are configured, perform the following steps to configure SASL EXTERNAL authentication:

1. Create a JKS keystore that includes a public and private key pair for a certificate that the PingDataSync Server instance(s) will use to authenticate to the PingDirectory Server instance(s). Run the following command in the instance root of one of the PingDataSync Server instances. When prompted for a keystore password, enter a strong password to protect the certificate. When prompted for the key password, press **ENTER** to use the keystore password to protect the private key:

   ```
   $ keytool -genkeypair \
     -keystore config/sync-user-keystore \
   ```

```
  -storetype JKS \
  -keyalg RSA \
  -keysize 2048 \
  -alias sync-user-cert \
  -dname "cn=Sync User,cn=Root DNs,cn=config" \
  -validity 7300
```

2. Create a `config/sync-user-keystore.pin` file that contains a single line that is the keystore password provided in the previous step.
3. If there are other PingDataSync Server instances in the topology, copy the `sync-user-keystore` and `sync-user-keystore.pin` files into the config directory for all instances.
4. Use the following command to export the public component of the user certificate to a text file:

```
$ keytool -export \
  -keystore config/sync-user-keystore \
  -alias sync-user-cert \
  -file config/sync-user-cert.txt
```

5. Copy the `sync-user-cert.txt` file into the `config` directory of all PingDirectory Server instances. Import that certificate into each server's primary trust store by running the following command from the server root. When prompted for the keystore password, enter the password contained in the `config/truststore.pin` file. When prompted to trust the certificate, enter **yes**.

```
$ keytool -import \
  -keystore config/truststore \
  -alias sync-user-cert \
  -file config/sync-user-cert.txt
```

6. Update the configuration for each PingDataSync Server instance to create a new key manager provider that will obtain its certificate from the `config/sync-user-keystore` file. Run the following `dsconfig` command from the server root:

```
$ dsconfig create-key-manager-provider \
  --provider-name "Sync User Certificate" \
  --type file-based \
  --set enabled:true \
  --set key-store-file:config/sync-user-keystore \
  --set key-store-type:JKS \
  --set key-store-pin-file:config/sync-user-keystore.pin
```

7. Update the configuration for each LDAP external server in each PingDataSync Server instance to use the newly-created key manager provider, and also to use SASL EXTERNAL authentication instead of LDAP simple authentication. Run the following `dsconfig` command:

```
$ dsconfig set-external-server-prop \
  --server-name ds1.example.com:636 \
```

```
  --set authentication-method:external \
  --set "key-manager-provider:Sync User Certificate"
```

After these changes, the PingDataSync Server should re-establish connections to the LDAP external server and authenticate with SASL EXTERNAL. Verify that the PingDataSync Server is still able to communicate with all backend servers by running the `bin/status` command. All of the servers listed in the "--- LDAP External Servers ---" section should have a status of `Available`. Review the PingDirectory Server access log can to make sure that the BIND RESULT log messages used to authenticate the connections from the PingDataSync Server include `authType="SASL", saslMechanism="EXTERNAL", resultCode=0`, and `authDN="cn=Sync User,cn=Root DNs,cn=config"`.

Server SDK extensions

Custom server extensions can be created with the Server SDK. Extension bundles are installed from a .zip archive or a file system directory. Use the `manage-extension` tool to install or update any extension that is packaged using the extension bundle format. It opens and loads the extension bundle, confirms the correct extension to install, stops the server if necessary, copies the bundle to the server install root, and then restarts the server.

> **Note**
> The `manage-extension` tool must be used with Java extensions packaged using the extension bundle format. For more information, see the "Building and Deploying Java-Based Extensions" section of the Server SDK documentation.

The Server SDK enables creating extensions for all PingData servers. Cross-product extensions include:

- Access Loggers
- Alert Handlers
- Error Loggers
- Key Manager Providers
- Monitor Providers
- Trust Manager Providers
- OAuth Token Handlers
- Manage Extension Plugins

Chapter 4: Synchronizing with PingOne

The PingDataSync Server supports PingOne as a Sync Destination for newly created or modified accounts with native password changes between directory servers, relational databases, or other PingOne systems.

This chapter presents configuration procedures for synchronization between PingDirectory Server, Nokia 8661 Directory Server, or other LDAP source servers or targets with PingOne.

Topics include:

Configuration on PingOne

PingOne can be configured as a Sync Destination. You must have a PingOne account. The following need to be configured in the PingOne administrative console:

1. Log into PingOne.
2. On the **Connections** tab, create an application with the following settings:
    a. Make the application non-interactive.
    b. Provide a name, such as SyncApp.
    c. Grant the following scopes:
    - "p1:read:env:user"
    - "p1:create:env:user"
    - "p1:update:env:user"
    - "p1:delete:env:user"
    - "p1:read:env:population"
    - "p1:create:env:population"
    - "p1:update:env:population"
    - "p1:delete:env:population"
    - "p1:set:env:userPassword"
    - "p1:validate:env:userPassword"
3. Enable the application.

After the application is enabled, collect the following information to configure the PingDataSync Server:

- Client ID.
- Client secret.
- Token endpoint.
- PingOne environment ID.
- Organization ID.
- Default population ID.

Overview of configuration tasks

The PingDataSync Server provides a `dsconfig` batch file that can be configured and run to enable synchronization with PingOne. Included in the file, the following steps are used to configure synchronization with PingOne systems:

- **Configure the external server** – PingOne must be configured as an external server.
- **Define the Sync Source and Destination** – The PingOne environment name, OAuth client ID, token URL, and OAuth client secret are required to configure synchronization with a PingDataSync Server.
- **Create a Sync Pipe** – The `create-sync-pipe-config` tool is used to configure the Sync Pipes to communicate with the PingOne source or target.
- **Define attribute mappings** – PingOne has a fixed schema. In order to successfully synchronize users from a Sync Source, it is necessary to provide values for the following attributes:
  - `name`: A JSON object in the form `{"name": {"given": "John", "family": "Doe", "middle": "M"}}`.
  - `email`: A JSON string that must be a valid email address AND be UNIQUE in a PingOne environment. This value is required on creates.
  - `username`: A JSON string that must be UNIQUE in a PingOne environment. This value is required on creates.
  - `population`: A JSON object in the form `{"id": "population-id"}`. This value is required on creates.
  - `phone`: A JSON string in the form `"+1.5555555555"`.
- **Create a Sync Class** – After defining the attribute map, create a Sync Class and associate the map.
- **Define JSON attributes** – The name and population attributes must have the associated Sync Pipe and Sync Class.

The `reference-pingone-sync-destination-configuration.dsconfig` file can be found in the `<server-root>/resource` directory. Add the relevant configuration details to this file before running it.

Note
If values are provided for fields not listed here it may result in errors during synchronization. The `auto-mapped-source-attribute` must be set to `none`, because only the listed attributes can be synchronized. Passwords can be synchronized only when created. They cannot be synchronized when modified.

Configure synchronization

Use the `reference-pingone-sync-destination-configuration.dsconfig` file to configure PingOne synchronization. This file shows an example configuration for a one-way sync relationship between PingDirectory Server and PingOne. Configuration includes the following tasks.

Create the external server for source

The following sample configures a PingDirectory Server.

```
$ bin/dsconfig create-external-server \
  --server-name [SOURCE_SERVER_NAME] \
  --type ping-identity-ds \
  --set server-host-name:[SOURCE_HOST_NAME] \
  --set server-port:[SOURCE_PORT] \
  --set authentication-method:simple \
  --set bind-dn:[SOURCE_BIND_DN] \
  --set password:[SOURCE_BIND_PASSWORD]
```

Define Sync Source and Destination

The following commands create the Sync Source and destinations.

```
$ bin/dsconfig create-sync-source \
  --source-name DS  \
  --type ping-identity  \
  --set base-dn:dc=example,dc=com  \
  --set server:[SOURCE_SERVER_NAME]
```

```
$ bin/dsconfig create-sync-destination \
  --destination-name PingOne  \
  --type ping-one-customer  \
  --set ping-one-api-url:https://api.pingone.com/v1 \
  --set ping-one-auth-url:https://auth.pingone.com/[PING_ONE_ENV_ID]/as/token
\
  --set ping-one-environment-id:[PING_ONE_ENV_ID]  \
  --set ping-one-oauth-client-id:[PING_ONE_OAUTH_CLIENT_ID]  \
  --set ping-one-oauth-client-secret:[PING_ONE_OAUTH_CLIENT_SECRET]
```

Create a Sync Pipe

The following sample command creates a Sync Pipe.

```
$ bin/dsconfig create-sync-pipe \
  --pipe-name DS_to_PingOne  \
  --set 'retry-backoff-increase-by:100 ms'  \
  --set sync-destination:PingOne  \
  --set sync-source:DS
```

Define attribute mappings

PingOne has a fixed schema. The `name` and `population` attributes are JSON objects in PingOne, therefore it is important to define them as JSON attributes in the Sync Class so that modifications are handled properly.

To successfully synchronize users from a Sync Source, values for the following attributes must be provided:

- `"name"`: A JSON object in the form `{"name": {"given": "John", "family": "Doe", "middle": "M"}}`
- `"email"`: A JSON string that must be a valid email address AND be UNIQUE in a PingOne Environment. This value is required on creates.
- `"username"`: A JSON string that must be UNIQUE in a PingOne Environment. This value is required on creates.
- `"population"`: A JSON object in the form `{"id": "population-id"}`. This value is required on creates.
- `"phone"`: A JSON string in the form `"+1.5555555555"`.

If values are provided for fields not listed here, errors may occur during synchronization. The following samples create the map:

```
$ bin/dsconfig create-attribute-map \
  --map-name PingOneUserMap
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name username \
  --type constructed \
  --set value-pattern:{givenname}.{sn}
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name password \
  --type direct \
  --set from-attribute:userPassword
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name email \
  --type constructed \
  --set value-pattern:{givenname}.{sn}@example.com
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name population \
  --type constructed  \
  --set 'value-pattern:{{"id":"[DEFAULT_POPULATION_ID]"}}'
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name phone \
  --type direct \
  --set from-attribute:telephoneNumber
```

```
$ bin/dsconfig create-attribute-mapping \
  --map-name PingOneUserMap \
  --mapping-name name \
  --type constructed \
```

```
  --set 'value-pattern:{{"givenName":"{givenname:jsonEscape}","familyName":"
{sn:jsonEscape}"}}'
```

Create a Sync Class and associate the defined Attribute Map

```
$ bin/dsconfig create-sync-class \
  --pipe-name [SYNC_PIPE_NAME] \
  --class-name [SYNC_CLASS_NAME] \
  --set attribute-map:PingOneUserMap \
  --set destination-correlation-attributes:username \
  --set destination-correlation-attributes:email \
  --set creates-as-modifies:true \
  --set include-filter:(objectClass=person)
```

Define JSON attributes

```
$ bin/dsconfig create-json-attribute \
  --pipe-name [SYNC_PIPE_NAME] \
  --class-name [SYNC_CLASS_NAME] \
  --attribute-name name
```

```
$ bin/dsconfig create-json-attribute \
  --pipe-name [SYNC_PIPE_NAME] \
  --class-name [SYNC_CLASS_NAME] \
  --attribute-name population \
  --set include-field:id
```

Run the resync command

If `resync` is run against an instance that has already been loaded with users, the `password` attribute should be added to the list of ignored destination attributes.

$ resync --pipe-name [SYNC_PIPE_NAME] \
  --excludeDestinationAttr password

Setup debug targets for added logging

The following will log all of the raw HTTP traffic between PingOne and PingDataSync Server, which can include sensitive information, such as passwords and authentication tokens. Do not enable or run this on in a production environment.

```
$ bin/dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name
com.unboundid.directory.sync.pingone.PingOneCustomerSyncDestination \
  --set debug-level:verbose
```

```
$ bin/dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name com.unboundid.directory.sync.pingone.HTTPTrafficLogger \
  --set debug-level:verbose
```

Chapter 5: Synchronizing with Active Directory systems

The PingDataSync Server supports full synchronization for newly created or modified accounts with native password changes between directory server, relational databases, and Microsoft Active Directory systems.

This chapter presents configuration procedures for synchronization between PingDirectory Server, Nokia 8661 Directory Server, or other LDAP source servers or targets with Microsoft Active Directory systems.

Topics include:

Overview of configuration tasks

To configure synchronization with Active Directory systems, the following tasks are performed:

- **Enable SSL connections** – If synchronizing passwords between systems, synchronization with Microsoft Active Directory systems requires that SSL be enabled on the Active Directory domain controller, so that the PingDataSync Server can securely propagate the `cn=Sync User` account password and other user passwords to the target.
- **Run the create-sync-pipe-config tool** – On the PingDataSync Server, use the `create-sync-pipe-config` tool to configure the Sync Pipes to communicate with the Active Directory source or target.
- **Configure outbound password synchronization on an PingDirectory Server Sync Source** – After running the `create-sync-pipe-config` tool, determine if outbound password synchronization from an PingDirectory Server Sync Source is required. If so, enable the Password Encryption component on all PingDirectory Server sources that receive password modifications. The PingDirectory Server uses the Password Encryption component, analogous to the Password Sync Agent component, to intercept password modifications and add an encrypted attribute, `ds-changelog-encrypted-password`, to the changelog entry. The component enables passwords to be synchronized securely to the Active Directory system, which uses a different password storage scheme. The encrypted attribute appears in the change log and is synchronized to the other servers, but does not appear in the entries.
- **Configure outbound password synchronization on an Active Directory Sync Source** – After running the `create-sync-pipe-config` tool, determine if outbound password synchronization from an Active Directory Sync Source is required. If so, install the Password Sync Agent (PSA) after configuring the PingDataSync Server.
- **Run the realtime-sync set-startpoint tool** – The `realtime-sync set-startpoint` command may take several minutes to run, because it must issue repeated searches of the Active Directory domain controller until it has paged through all the changes and receives a cookie that is up-to-date.

Configuring synchronization with Active Directory

The following procedure configures a one-way Sync Pipe with the Active Directory topology as the Sync Source and an PingDirectory Server topology as the Sync Destination.

1. From the server-root directory, start the PingDataSync Server.

```
$ <server-root>/bin/start-server
```

2. Run the `create-sync-pipe-config` tool to set up the initial synchronization topology.

```
$ bin/create-sync-pipe-config
```

3. On the Initial Synchronization Configuration Tool menu, press **Enter** to continue the configuration.
4. On the Synchronization Mode menu, press **Enter** to select Standard mode.
5. On the Synchronization Directory menu, select the option for one-way (1) or bidirectional (2) for the synchronization topology.
6. On the Source Endpoint Type menu, enter the option for Microsoft Active Directory.
7. On the Source Endpoint Name menu, type a name for the source server, or accept the default.
8. On the Server Security menu, select the security connection type for the source server.
9. On the Servers menu, enter the host name and listener port for the Source Server, or press **Enter** to accept the default (port 636). The server will attempt a connection to the server. After adding the first server, add additional servers for the source endpoints, which will be prioritized below the first server.
10. On the Synchronization User Account DN menu, enter the User Account DN for the source servers. The account will be used exclusively by the PingDataSync Server to communicate with the source external servers. Enter a User Account DN and password. The User Account DN password must meet the minimum password requirements for Active Directory domains.
11. Set up the Destination Endpoint servers.

The Active Directory Sync User account

The Sync User created for Active Directory is added to the `cn=Administrators` branch and is given most of a root user's permissions. If this account cannot be secured and there is a need to configure the permissions required by the Sync User, the following are required to perform synchronization tasks:

As a Sync Source, these permissions are needed:

- List contents
- Read all properties
- Read permissions

Deleted items are a special case. For the Sync Server to see deleted entries, the user account must have sufficient access to `cn=Deleted Objects,<domain name>`. Giving access to that DN requires using the `dsacls` tool, such as:

```
# Take ownership may be required to make the needed changes.
dsacls "CN=Deleted Objects,DC=example,DC=com" /takeOwnership
```

```
# Give the Sync User generic read permission to the domain.
dsacls "CN=Deleted Objects,DC=example,DC=com" /G "example\SyncUser":GR
```

```
# List the permisison for the domain.
dsacls "CN=Deleted Objects,DC=example,DC=com"
```

To revoke all permissions from the Sync User, run the following `dsacls` command:

```
dsacls "CN=Deleted Objects,DC=example,DC=com" /R "example\SyncUser"
```

If Active Directory is used as a destination for synchronization, the Sync User account should not be changed.

Prepare external servers

Perform the following steps to prepare external servers:

1. After configuring the source and destination endpoints, the PingDataSync Server prompts to "prepare" each external server. The process requires trusting the certificate presented to the server, and then testing the connection. If this step is not performed, the process can be completed after configuring the Sync Pipes using the `prepare-endpoint-server` tool.
2. Configuring this server for synchronization requires manager access. Enter the DN and password of an account capable of managing the external directory server.
3. Enter the maximum age of changelog entries. The value is formatted as `[number][time-unit]`, where the time unit format resembles ("8h" for eight hours, "3d" for three days, "1w" for one week). Setting this value caps how long the PingDataSync Server can be offline. A smaller value limits how many changes are stored and is necessary to limit excessive changelog growth in high-modification environments.
4. To prepare another server in the topology, follow the prompts. The previously entered manager credentials can be reused to access additional servers. Repeat the process for each server configured in the system.

Configure Sync Pipes and Sync Classes

Perform the following steps to configure Sync Pipes and Sync Classes:

1. On the Sync Pipe Name menu, type a unique name to identify the Sync Pipe, or accept the default.
2. On the Pre-Configured Sync Class Configuration for Active Directory Sync Source menu, enter **yes** to synchronize user CREATE operations, and enter the object class for the user entries at the destination server, or accept the default (`user`). To synchronize user MODIFY and DELETE operations from Active Directory, enter **yes**.

3. To synchronize passwords from Active Directory, press **Enter** to accept the default (yes). If synchronizing passwords from Active Directory, install the Ping Identity Password Sync Agent component on each domain controller.
4. To create a DN map for the user entries in the Sync Pipe, enter the base DN for the user entries at the Microsoft Active Directory Sync Source, then enter the base DN for the user entries at the PingDataSync Server Destination.
5. A list of basic attribute mappings from the Microsoft Active Directory Source to the PingDirectory Server destination is displayed. More complex attribute mappings involving constructed or DN attribute mappings must be configured with the `dsconfig` tool. The following is a sample mapping.

```
Below is a list of the basic mappings that have been set up for user
entries synchronized from Microsoft Active Directory -> PingDirectory
Server. You can add to or modify this list with any direct attribute
mappings. To set up more complex mappings (such as constructed or DN
attribute mappings), use the 'dsconfig' tool.

1) cn -> cn
2) sn -> sn
3) givenName -> givenName
4) description -> description
5) sAMAccountName -> uid
6) unicodePwd -> userPassword
```

6. Enter the option to add a new attribute mapping. Enter the source attribute, and then enter the destination attribute. The following example maps the `telephoneNumber` attribute (Active Directory) to the `otherTelephone` attribute (PingDirectory Server).

```
Select an attribute mapping to remove, or choose 'n' to add a new one
[Press ENTER to continue]: n

Enter the name of the source attribute: telephoneNumber
Enter the name of the destination attribute: otherTelephone
```

7. If synchronizing group CREATE, MODIFY, and DELETE operations from Active Directory, enter **yes**.
8. Review the basic user group mappings.
9. On the Sync Pipe Sync Class Definitions menu, enter another name for a new Sync Class if required. Repeat steps 2–7 to define this new Sync Class. If no additional Sync Class definitions are required, press **Enter** to continue.
10. Review the Sync Pipe Configuration Summary, and accept the default ("write configuration"), which records the commands in a batch file (`sync-pipe-cfg.txt`). The batch file can be used to set up other topologies. The following summary shows two Sync Pipes and its associated Sync Classes.

```
>>>> Configuration Summary

  Sync Pipe: AD to PingDirectory Server
    Source: Microsoft Active Directory
      Type: Microsoft Active Directory
      Access Account: cn=Sync
User,cn=Users,DC=adsync,DC=PingIdentity,DC=com
      Base DN: DC=adsync,DC=PingIdentity,DC=com
      Servers: 10.5.1.149:636

    Destination: PingDirectory Server
      Type: PingDirectory Server
      Access Account: cn=Sync User,cn=Root DNs,cn=config
      Base DN: dc=example,dc=com
      Servers: localhost:389

    Sync Classes:
      Microsoft Active Directory Users Sync Class
      Base DN: DC=adsync,DC=PingIdentity,DC=com
      Filters: (objectClass=user)
      DN Map: **,CN=Users,DC=adsync,DC=PingIdentity,DC=com ->{1},ou=users,
      dc=example,dc=com
      Synchronized Attributes: Custom set of mappings are defined
      Operations: Creates,Deletes,Modifies

  Sync Pipe: PingDirectory Server to AD
    Source: PingDirectory Server
      Type: PingDirectory Server
      Access Account: cn=Sync User,cn=Root DNs,cn=config
      Base DN: dc=example,dc=com
      Servers: localhost:389

    Destination: Microsoft Active Directory
      Type: Microsoft Active Directory
      Access Account: cn=Sync
User,cn=Users,DC=adsync,DC=PingIdentity,DC=com
      Base DN: DC=adsync,DC=PingIdentity,DC=com
      Servers: 10.5.1.149:636

    Sync Classes:
       PingDirectory Server Users Sync Class
      Base DN: dc=example,dc=com
      Filters: (objectClass=inetOrgPerson)
      DN Map: **,ou=users,dc=example,dc=com ->{1},CN=Users,DC=adsync,
       DC=PingIdentity,DC=com
      Synchronized Attributes: Custom set of mappings are defined
      Operations: Creates,Deletes,Modifies
```

11. To apply the configuration to the local PingDataSync Server instance, type **yes**. The configuration is recorded at `<server-root>/logs/tools/createsync-pipe-`

`config.log`.

Configure password encryption

This procedure is required if synchronizing passwords from an PingDirectory Server to Active Directory, or if synchronizing clear text passwords. These steps are not required if synchronizing from Active Directory to an PingData PingDirectory Server, or if not synchronizing passwords.

1. On the PingDirectory Server that will receive the password modifications, enable the Change Log Password Encryption component. The component intercepts password modifications, encrypts the password and adds an encrypted attribute, `ds-changelog-encrypted-password`, to the change log entry. The encryption key can be copied from the output if displayed, or accessed from the `<serverroot>/bin/sync-pipe-cfg.txt` file.

   ```
   $ bin/dsconfig set-plugin-prop --plugin-name "Changelog Password
   Encryption" \
     --set enabled:true \
     --set changelog-password-encryption-key:<key>
   ```

2. On the PingDataSync Server, set the decryption key used to decrypt the user password value in the change log entries. The key allows the user password to be synchronized to other servers that do not use the same password storage scheme.

   ```
   $ bin/dsconfig set-global-sync-configuration-prop \
     --set changelog-password-decryption-key:ej5u9e39pqo68
   ```

Test the configuration or populate data in the destination servers using bulk resync mode. See Using the resync Tool on the Identity Sync Server. Then, use `realtime-sync` to start synchronizing the data. See Using the realtime-sync Tool for more information. If synchronizing passwords, install the Password Sync Agent (PSA) on all of the domain controllers in the topology.

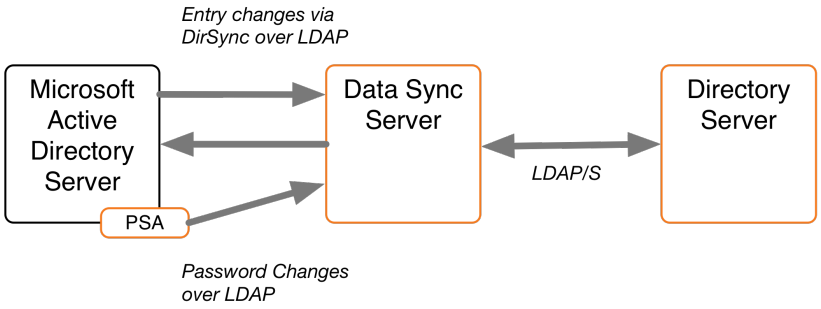
The Password Sync Agent

When synchronizing passwords with Active Directory systems, the PingDataSync Server requires that the Ping Identity Password Sync Agent (PSA) be installed on all domain controllers in the synchronization topology. This component provides real-time outbound password synchronization from Microsoft Active Directory to any supported Sync Destinations.

The PSA component provides password synchronization between directories that support differing password storage schemes. The PSA immediately hashes the password with a 160-bit salted secure hash algorithm and erases the memory where the clear-text password was stored. The component only transmits data over a secure (SSL) connection, and follows Microsoft's security guidelines when handling clear-text passwords. The PSA also uses Microsoft Windows password filters, which are part of the local security authority (LSA)

process. The password filters enable implementing password policy validation and change notification mechanisms. For more information, see Microsoft's product documentation.

Note
For outbound password synchronization from an PingDirectory Server to Active Directory, enable the Password Encryption component. See Configuring the Password Encryption Component for more information.



Password Synchronization with Microsoft Active Directory

The PSA supports failover between servers. It caches the hashed password changes in a local database until it can be guaranteed that all PingDataSync Servers in the topology have received them. The failover features enable any or all of the PingDataSync Servers to be taken offline without losing any password changes from Active Directory.

The PSA is safe to leave running on a domain controller indefinitely. To stop synchronizing passwords, remove the `userPassword` attribute mapping in the PingDataSync Server, or stop the server. The PSA will not allow its local cache of password changes to grow out of control; it automatically cleans out records from its local database as soon as they have been acknowledged. It also purges changes that have been in the database for more than a week.

Before installing the PSA, consider the following:

- Make sure that the Active Directory domain controller has SSL enabled and running.
- Make sure the PingDataSync Servers are configured to accept SSL connections when communicating with the Active Directory host.
- At least one Active Directory Sync Source (ADSyncSource) needs to be configured on the PingDataSync Server and should point to the domain controller(s) on which the PSA will reside.
- At the time of installation, all PingDataSync Servers in the sync topology must be online and available.
- The PSA component is for outbound-only password synchronization from the Active Directory Systems. It is not necessary if performing a one-way password synchronization from the PingDirectory Server to the Active Directory server.

Install the Password Sync Agent

The PingDirectory Server distributes the PSA in zip file format with each PingDataSync Server package. The initial installation of the PSA requires a system restart.

Perform the following steps to install the PSA

1. On the domain controller, double-click the `setup.exe` file to start the installation.
2. Select a folder for the PSA binaries, local database, and log files.
3. Enter the host names (or IP addresses) and SSL ports of the PingDataSync Servers, such as `sync.host.com:636`. Do not add any prefixes to the hostnames.
4. Enter the Directory Manager DN and password. This creates an ADSync user on the PingDataSync Server.
5. Enter a secret key (password) that the sync agent uses when the ADSync user connects to the PingDataSync Server instances.
6. Click **Next** to begin the installation. All of the specified PingDataSync Servers are contacted, and any failures will roll back the installation. If everything succeeds, a message displays indicating that a restart is required. The PSA will start when the computer restarts, and the LSA process is loaded into memory. The LSA process cannot be restarted at runtime.
7. If synchronizing pre-encoded passwords from Active Directory to a Ping Identity system, allow pre-encoded passwords in the default password policy.

```
$ bin/dsconfig set-password-policy-prop \
  --policy-name "Default Password Policy" \
  --set allow-pre-encoded-passwords:true
```

Upgrade or Uninstall the Password Agent

The PingDataSync Server provides the `update` tool for upgrades to the server code, including the PSA. The upgrade does not require a restart, because the core password filter is already running under LSA. The upgrade replaces the implementation binaries, which are encapsulated from the password filter DLL.

To uninstall the PSA on the Active Directory system, use **Add/Remove Programs** on the Windows Control Panel. The implementation DLL will be unloaded, and the database and log files are deleted. Only the binaries remain.

The core password filter will still be running under the LSA process. It imposes zero overhead on the domain controller, because the implementation DLL has been unloaded. To remove the password filter itself (located at `C:\WINDOWS\System32\ubidPassFilt.dll`), restart the computer. On restart, the password filter and implementation binaries (in the install folder) can be deleted.

Note
The PSA cannot be reinstalled without another reboot.

Manually Configure the Password Sync Agent

Configuration settings for the Password Sync Service are stored in the Windows registry in `HKLM\SOFTWARE\UnboundID\PasswordSync`. Configuration values under this registry key can be modified during runtime. The agent automatically reloads and refresh its settings from the registry. Verify that the agent is working by checking the current log file, located in `<server-root>\logs\password-sync-current.log`.

Chapter 6: Synchronize with relational databases

The PingDataSync Server supports high-scale, highly-available data synchronization between the directory servers and relational database management systems (RDBMS). Any database with a JDBC 3.0 or later driver can be used.

Topics include:

Use the Server SDK

Synchronizing LDAP data to or from a relational database requires creating a JDBC Sync Source or Destination extension to act as an interface between the PingDataSync Server and the relational database. The Server SDK provides APIs to develop plug-ins and third-party extensions to the server using Java or Groovy. The Server SDK's documentation is delivered with the Server SDK build in zip format.

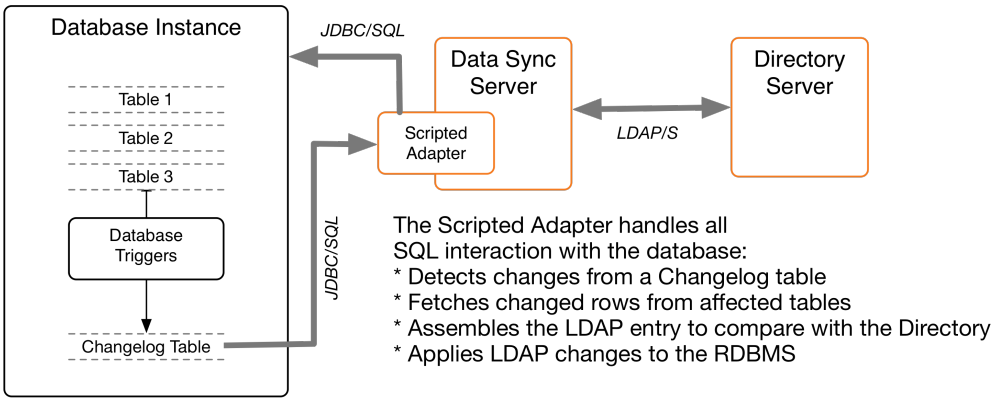
> **Note**
> Server SDK support is provided with Premium Support for the each product. Ping Identity does not provide support for the third party extensions developed using the Server SDK. Contact a Ping Identity support representative for assistance.

The Server SDK contains two abstract classes that correspond to how the database is used:

- `com.unboundid.directory.sdk.sync.api.JDBCSyncSource`
- `com.unboundid.directory.sdk.sync.api.JDBCSyncDestination`

The remainder of the SDK contains helper classes and utility functions to facilitate the script implementation. The SDK can use any change tracking mechanism to detect changes in the database. Examples are provided in the `<server-root>/config/jdbc/samples` directory for Oracle Database and Microsoft SQL Server.

The PingDataSync Server uses a scripted adapter layer to convert any database change to an equivalent LDAP entry. The Sync Pipe then processes the data through inclusive (or exclusive) filtering using attribute and DN maps defined in the Sync Classes to update the endpoint servers. For example, a script using Java can be configured by setting the `extension-class` property on a `ThirdPartyJDBCSyncSource` or `ThirdPartyJDBCSyncDestination` configuration object within the PingDataSync Server. The following is a sample architecture.



Synchronizing with RDBMS Overview

The RDBMS synchronization process

The PingDataSync Server synchronizes data between a directory server and an RDBMS system with a Server SDK extension. The PingDataSync Server provides multiple configuration options, such as advanced filtering (fractional and subtree), attribute and DN mappings, transformations, correlations, and configurable logging.

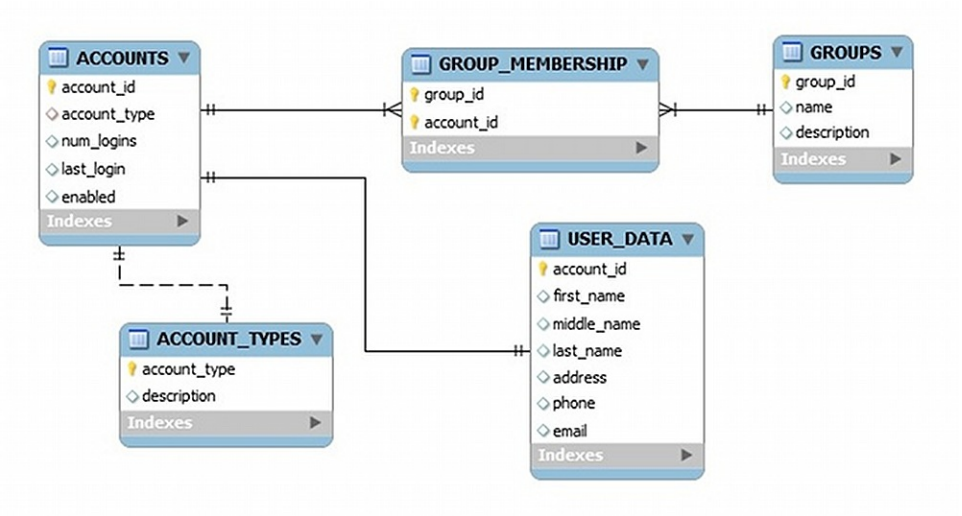
To support synchronizing changes, the database must be configured with a change tracking mechanism. An approach involving triggers, (one trigger per table) to record all changes to a change log table, is recommended. The database change log table Ping Identityshould record the type of change (INSERT, UPDATE, DELETE), the specific table name, the unique identifier for the changed row, the database entry type, the changed columns (from the source table), the modifier's name, and the change timestamp.

The PingDataSync Server delegates the physical interaction with the database to a user-defined extension, which has full control of the SQL queries. The extension layer provides flexibility in how the mapping semantics between the LDAP environment and the relational database environment are defined. The connection management, pooling, retry logic, and other boilerplate code are handled internally by the PingDataSync Server.

The RDBMS Synchronization (DBSync) implementation does not support failover between different physical database servers. Most enterprise databases have a built-in failover layer from which the PingDataSync Server can point to a single virtual address and port and still be highly available. A single RDBMS node can scale to multiple directory server endpoints.

DBSync example

ThePingDataSync Server provides a DBSync example between two endpoints consisting of an Ping Identity PingDirectory Server source and a RDBMS system, which will be used in this chapter. The entity-relational diagram for the normalized database schema is available in `<server-root>/config/jdbc/samples/oracle-db/ComplexSchema.jpg`, and is illustrated here:



Entry Relation Diagrams for the Schema Tables

Five tables are represented with their primary keys in bold. The entity relations and foreign keys are marked by the relationship lines. The illustration shows mapping to a custom object class on the directory server, while the "groups" table maps to a standard LDAP group entry with `objectclass:groupOfUniqueNames`.

Example directory server entries

The following example assumes that the directory server's schema is configured to handle the mapped attributes. If configuring a database-to-directory Sync Pipe with a newly installed directory server, make sure that the schema has the correct `attributeType` and `objectClass` definitions in place. The definitions can be added in a custom `99-user.ldif` file in the `config/schema` folder of the directory server, if necessary. The following are the LDAP entries that are used in the synchronization example:

```
dn: accountid=0,ou=People,dc=example,dc=com
objectClass: site-user
firstName: John
lastName: Smith
accountID: 1234
email: jsmith@example.com
phone: +1 556 805 4454
address: 17121 Green Street
numLogins: 4
lastLogin: 20070408182813.196Z
enabled: true

dn: cn=Group 1,ou=Groups,dc=example,dc=com
objectClass: groupOfUniqueNames
description: This is Group 1
uniqueMember: accountID=0,ou=People,dc=example,dc=com
uniqueMember: accountID=1,ou=People,dc=example,dc=com
```

Configure DBSync

Configuring a DBSync system includes extra tasks to create the extensions and to configure the database. The overall configuration process is as follows:

1. Download the appropriate JDBC driver to the PingDataSync Server's `/PingDataSync/lib` directory, and restart the server for the driver to load into the runtime.
2. Open the `java.properties` file with a text editor and add the `jdbc.drivers` argument. Save the file.
3. Run the `dsjavaproperties` command to apply the change. For example, enter the following for `start-sync-server`:

   ```
   start-sync-server.java-args=-d64 -server -Xmx256m -Xms256m -
   XX:+UseConcMarkSweepGC -
   ```

```
Djdbc.drivers=foo.bah.Driver:wombat.sql.Driver:com.example.OurDriver ...
etc.
```

4. Create one or more JDBC extensions based on the Server SDK. If configuring for bidirectional synchronization, two scripts are needed: one for the JDBC Sync Source; the other for the JDBC Sync Destination. Place the compiled extensions in the `/lib/extensions` directory.
5. Configure the database change log table and triggers (presented later). The vendor's native change tracking mechanism can be used, but a change log table should also be configured. Each table requires one database trigger to detect the changes and loads them into the change log table.
6. Configure the Sync Pipes, Sync Classes, external servers, DN and attribute maps for one direction.
7. Run the `resync --dry-run` command to test the configuration settings.
8. Run `realtime-sync set-startpoint` to initialize the starting point for synchronization.
9. Run the `resync` command to populate data on the destination endpoint.
10. Start the Sync Pipes using the `realtime-sync start` command.
11. Monitor the PingDataSync Server using the `status` commands and logs.
12. For bidirectional synchronization, configure another Sync Pipe, and repeat steps 4–8 to test the system.

Create the JDBC extension

The JDBC extension implementation must be written in Java, or the Groovy scripting language. Consult the Server SDK documentation for details on how to build and deploy extensions. The examples in this guide use Java. Java extensions are more strict and will catch programming errors during compile time rather than at runtime. Groovy is more flexible and can accomplish more with less lines of code.

Groovy scripts must reside in the `/lib/groovy-scripted-extensions` directory (Java implementations reside in `/lib/extensions`), which may also contain other plug-ins built using the Server SDK. If a script declares a package name, it must live under the corresponding folder hierarchy, just like a Java class. For example, to use a script class called `ComplexJDBCSyncSource` whose package is `com.unboundid.examples.oracle`, place it in `/lib/groovy-scripted-extensions/com/unboundid/examples/oracle` and set the `script-class` property on the Sync Source to `com.unboundid.examples.oracle.ComplexJDBCSyncSource`. There are a few reference implementations provided in the `config/jdbc/samples` directory. Use the `manage-extension` tool in the `bin` directory, or `bat` (Windows) to install or update the extension. See the Server SDK Extensions section for more information.

Note
Any changes to an existing script require a manual Sync Pipe restart. Any configuration change automatically restarts the affected Sync Pipe.

The default libraries available on the classpath to the script implementation include:

- Groovy
- LDAP SDK for Java
- JRE

Logging from within a script can be done with the Server SDK's `ServerContext` abstract class. Some of `ServerContext` methods are not available when the `resync` tool runs, because it runs outside of the PingDataSync Server Server process. Any logging during a `resync` operation is saved to the `logs/tools/resync.log` file.

Implement a JDBC Sync Source

The `JDBCSyncSource` abstract class must be implemented to synchronize data from a relational database. Since the PingDataSync Server is LDAP-centric, this class enables database content to be converted into LDAP entries. For more detailed information on the class, see the Server SDK Javadoc.

The extension imports classes from the Java API, LDAP SDK for Java API, and the Server SDK. Depending on the data, implement the following methods:

- `initializeJDBCSyncSource` – Called when a Sync Pipe first starts, or when the `resync` process starts. Any initialization should be performed here, such as creating internal data structures and setting up variables.
- `finalizeJDBCSyncSource` – Called when a Sync Pipe stops, or when the `resync` process stops. Any clean up should be performed here, and all internal resources should be freed.
- `setStartpoint` – Sets the starting point for synchronization by identifying the starting point in the change log. This method should cause all changes previous to the specified start point to be disregarded and only changes after that point to be returned by the `getNextBatchOfChanges` method. There are several different startpoint types (see SetStartpointOptions in the Server SDK), and this implementation is not required to support them all. If the specified startpoint type is unsupported, this method throws an exception (`IllegalArgumentException`). This method can be called from two different contexts: when the `realtime-sync set-startpoint` command is used (the Sync Pipe is required to be stopped) or immediately after a connection is established to the source server.

Note
The `RESUME_AT_SERIALIZABLE` startpoint type must be supported. This method is used when a Sync Pipe first starts and loads its state from disk.

- `getStartpoint` – Gets the current value of the startpoint for change detection.
- `fetchEntry` – Returns a full source entry (in LDAP form) from the database, corresponding to the `DatabaseChangeRecord` object that is passed. The `resync` command also uses this class to retrieve entries.
- `acknowledgeCompletedOps` – Provides a means for the PingDataSync Server to acknowledge to the database which operations have completed processing.

Note
The internal value for the startpoint should only be updated after a synchronization operation is acknowledged in to this script (through this method). If not, changes could be missed when the PingDataSync Server is restarted.

- `getNextBatchOfChanges` – Retrieves the next set of changes for processing. The method also provides a generic means to limit the size of the result set.
- `listAllEntries` – Used by the `resync` command to get a listing of all entries.
- `cleanupChangelog` – In general, we recommend implementing a `cleanupChangelog` method, so that the PingDataSync Server can purge old records from the change log table, based on a configurable age.

See the `config/jdbc/samples` directory for example script implementations and the Server SDK Javadoc for more detailed information on each method.

Implement a JDBC Sync Destination

The `JDBCSyncDestination` abstract class must be implemented to synchronize data into a relational database. The class enables converting LDAP content to database content. The extension imports classes from the Java API, LDAP SDK for Java API, and the Server SDK, depending on the database configuration. Implement the following methods in the script:

- `initializeJDBCSyncDestination` – Called when a Sync Pipe starts, or when the `resync` process starts. Any initialization should be performed here, such as creating internal data structures and setting up variables.
- `finalizeJDBCSyncDestination` – Called when a Sync Pipe stops, or when the `resync` process stops. Any clean up should be performed here, and all internal resources should be freed.
- `createEntry` – Creates a full database entry (or row), corresponding to the LDAP entry that is passed in.
- `modifyEntry` – Modifies a database entry, corresponding to the LDAP entry that is passed in.
- `fetchEntry` – Returns a full destination database entry (in LDAP form), corresponding to the source entry that is passed in.

- `deleteEntry` – Deletes a full entry from the database, corresponding to the LDAP entry that is passed in.

For more detailed information on the abstract class, consult the Server SDK Javadoc.

Configure the database for synchronization

Configuring the database for synchronization includes defining:

- a database SyncUser account
- the change tracking mechanism
- the database triggers (one per table) for the application

The following procedure uses the example setup script in `/config/jdbc/samples/oracle-db/OracleSyncSetup.sql`. Items in brackets are user-named labels.

Note
Database change tracking necessary if synchronizing FROM the database. If synchronizing TO a database, configure the Sync User account and the correct privileges.

1. Create an Oracle login (`SyncUser`) for the PingDataSync Server, so that it can access the database server. Grant sufficient privileges to the `SyncUser` for any tables to be synchronized, and change the default password.

```
CREATE USER SyncUser IDENTIFIED BY password
DEFAULT TABLESPACE users TEMPORARY TABLESPACE temp;
GRANT "RESOURCE" TO SyncUser;
GRANT "CONNECT" TO SyncUser;
```

2. Create change log tables on the database as follows:

```
CREATE TABLE ubid_changelog (
  --This is the unique number for the change change_number Number NOT NULL
PRIMARY KEY,
  --This is the type of change (insert, update, delete). NOTE: This should
represent
  --the actual type of change that needs to happen on the destination(for
example a
  --database delete might translate to a LDAPmodify, etc.)
    change_type VARCHAR2(10) NOT NULL,

  --This is the name of the table that was changed table_name VARCHAR(50)
NOT NULL,
  --This is the unique identifier for the row that was changed. It is up
to
  --the trigger code to construct this, but it should follow a DN-like
format
  --(e.g. accountID={accountID}) where at least the primary key(s) are
  --present. If multiple primary keys are required, they should be
delimited
  --with a unique string, such as '%%' (e.g. accountID={accountID}%%
```

```
  --groupID={groupID})
    identifier VARCHAR2(100) NOT NULL,

  --This is the database entry type. The allowable values for this must be
  --set on the JDBC Sync Source configuration within the Synchronization
  --Server.
    entry_type VARCHAR2(50) NOT NULL,

  --This is a comma-separated list of columns from the source table that
were updated as part of
  --this change.
    changed_columns VARCHAR2(1000) NULL,

  --This is the name of the database user who made the change
    modifiers_name VARCHAR2(50) NOT NULL,

  --This is the timestamp of the change
    change_time TIMESTAMP(3) NOT NULL, CONSTRAINT chk_change_type
    CHECK (change_type IN ('insert','update','delete'))) ORGANIZATION
INDEX;
```

3. Create an Oracle function to get the `SyncUser` name. This is a convenience function for the triggers.

```
CREATE OR REPLACE FUNCTION get_sync_user RETURN VARCHAR2
IS
BEGIN
  RETURN 'SyncUser';
END get_sync_user;
```

4. Create an Oracle sequence object for the change-number column in the change log table.

```
CREATE SEQUENCE ubid_changelog_seq MINVALUE 1 START WITH 1
NOMAXVALUE INCREMENT BY 1 CACHE 100 NOCYCLE;
```

5. Create a database trigger for each table that will participate in synchronization. An example, located in `/config/jdbc/samples/oracle-db/OracleSyncSetup.sql`, shows a trigger for the Accounts table that tracks all changed columns after any INSERT, UPDATE, and DELETE operation. The code generates a list of changed items and then inserts them into the change log table.

Considerations for synchronizing to database destination

When configuring a directory-to-database Sync Pipe, the following are recommended:

- **Identify the Object Classes** – Create a Sync Class per object class, so that they can easily be distinguished and have different mappings and synchronization rules.

- For each Sync Class, set the following items in the configuration menus using the `dsconfig` tool.
  - **Set the Include-Filter Property** – Make sure the `include-filter` property is set on the Sync Class configuration menu to something that will uniquely identify the source entries, such as `objectClass=customer`.
  - **Create Specific Attribute Mappings** – Create an attribute mapping for every LDAP attribute to be synchronized to a database column, add these to a single attribute map, and set it on the Sync Class. With this configured, the script does not need to know about the schema on the directory side. A constructed attribute mapping that maps a literal value to the `objectClass` attribute can be added to the script to determine the database entry type. For example, `"account" -> objectClass` can be added, which would result in the constructed destination LDAP entry always containing an `objectClass` of `"account."` If needed, a mulit-valued `conditional-value-pattern` property can be used to conditionalize the attribute mapping based on the subtype of the entry or on the value of the attribute. See Conditional Value Mapping for additional information.
  - **Create Specific DN Maps (optional)** – If necessary, create a DN map that recognizes the DN's of the source entries and maps them to a desired destination DN. In most cases, the script will use the attributes rather than the DN to figure out which database entry needs to be changed.
  - **Set auto-mapped-source-attribute to "-none-"** – Remove the default value of "-all-" from the `auto-mapped-source-attribute` on the Sync Class configuration menu, and replace it with "-none-."
- **Configure Create-Only Attributes** – Any attributes that should be included when created, but never modified (such as `objectclass`) should be specified on the Sync Pipe as a `create-only` attribute. If the PingDataSync Server ever computes a difference in that attribute between the source and destination, it will not try to modify it at the destination. To avoid bidirectional loop-back, set the `ignore-changes-by-[user|dn]` property on both Sync Sources when configuring for bidirectional synchronization.
- **Synchronizing DELETE Operations** – On PingDirectory Server and Nokia 8661 Directory Server systems, configure the `changelog-deleted-entry-include-attribute` property on the changelog backend menu using the `dsconfig` tool. This property allows for the proper synchronization of DELETE operations. For more information, see Configuring the Directory Server Backend for Synchronizing Deletes.
- **Attribute-Synchronization-Mode for DBSync** – For MODIFY operations, the PingDataSync Server detects any change on the source change log, fetches the source entry, applies mappings, computes the equivalent destination entry, fetches the actual destination entry, and then runs a diff between the two entries to determine the minimal set of changes to synchronize. By default, changes on the destination entry are made

only for those attributes that were detected in the original change log entry. This is configurable using the `attribute-synchronization-mode` property, which sets the type of diff operation that is performed between the source and destination entries.

If the source endpoint is a database server, set the `attribute-synchronization-mode` property to `all-attributes` on the Sync Class configuration menu. The diff operation will consider all source attributes. Any that have changed will be updated on the destination, even if the change was not originally detected in the change log. This mode is useful when a list of changed columns in the database may not be available. If both endpoints are directory servers, use the default configuration of `modified-attributes-only` to avoid possible replication conflicts.

- **Handling MODDN Operations** – The concept of renaming an entry (modifyDN) does not have a direct equivalent for relational databases. The `JDBCSyncDestination` API does not handle changes of this type. Instead, the `modifyEntry()` method is called as if it is a normal change. The extension can verify if the entry was renamed by looking at the `SyncOperation` that is passed in (`syncOperation.isModifyDN()`). If true, the `fetchedDestEntry` parameter will have the old DN. The new DN can be obtained by calling `syncOperation.getDestinationEntryAfterChange()`.

Configure a directory-to-database Sync Pipe

The following configures a one-way Sync Pipe with an PingDirectory Server as the Sync Source and an RDBMS (Oracle) system as the Sync Destination with the `create-sync-pipe-config` tool. Sync Pipes can be configured later using `dsconfig`.

Create the Sync Pipe

The following procedures configure the Sync Pipe, external servers, and Sync Classes. The examples are based on the Complex JDBC sample in the `config/jdbc/samples/oracle-db` directory.

The `create-sync-pipe-config` tool can be run with the server offline and the configuration can later be imported.

1. Run the `create-sync-pipe-config` tool.

   ```
   $ bin/create-sync-pipe-config
   ```

2. At the Initial Synchronization Configuration Tool prompt, press **Enter** to continue.
3. On the Synchronization Mode menu, select Standard Mode or Notification Mode.
4. On the Synchronization Directory menu, choose one-way or bidirectional synchronization.

Configure the Sync Source

1. On the Source Endpoint Type menu, enter the number for the sync source corresponding to the type of source external server.
2. Enter a name for the Source Endpoint.
3. Enter the base DN for the directory server, which is used as the base for LDAP searches. For example, enter `dc=example,dc=com`, and then press **Enter** again to return to the menu. If entering more than one base DN, make sure the DNs do not overlap.
4. On the Server Security menu, select the type of communication that the PingDataSync Server will use with the endpoint servers.
5. Enter the host and port of the source endpoint server. The Sync Source can specify a single server or multiple servers in a replicated topology. The server tests that a connection can be established.
6. Enter the DN of the Sync User account and create a password for this account. The Sync User account enables the PingDataSync Server to access the source endpoint server. By default, the Sync User account is stored as `cn=SyncUser,cn=Root DNs,cn=config`.

Configure the destination endpoint server

1. On the Destination Endpoint Type menu, select the type of data store on the endpoint server. This example is configuring an Oracle Database.
2. Enter a name for the Destination Endpoint.
3. On the JDBC Endpoint Connection Parameters menu, enter the fully-qualified host name or IP address for the Oracle database server.
4. Enter the listener port for the database server, or press **Enter** to accept the default (1521).
5. Enter a database name such as `dbsync-test`.
6. The server attempts to locate the JDBC driver in the `lib` directory. If the file is found, a success message is displayed.

   ```
   Successfully found and loaded JDBC driver for:
   jdbc:oracle:thin:@//dbsync-w2k8-vm-2:1521/dbsync-test
   ```

   If the server cannot find the JDBC driver, add it later, or quit the `create-sync-pipe-config` tool and add the file to the `lib` directory.
7. Add any additional JDBC connection properties for the database server, or press **Enter** to accept the default (no). Consult the JDBC driver's vendor documentation for supported properties.

8. Enter a name for the database user account with which the PingDataSync Server will communicate, or press **Enter** to accept the default (SyncUser). Enter the password for the account.
9. On the Standard Setup menu, enter the number for the language (Java or Groovy) that was used to write the server extension.
10. Enter the fully qualified name of the Server SDK extension class that implements the JDBCSyncDestination API.

    ```
    Enter the fully qualified name of the Java class that will implement
    com.unboundid.directory.sdk.sync.api.JDBCSyncDestination:
    com.unboundid.examples.oracle.ComplexJDBCSyncDestination
    ```

11. Configure any user-defined arguments needed by the server extension. These are defined in the extension itself and the values are specified in the server configuration. If there are user-defined arguments, enter **yes**.
12. To prepare the Source Endpoint server, which tests the connection to the server with the Sync User account, press **Enter** to accept the default (yes). For the Sync User account, it will return "Denied" as the account has not been written yet to the Directory Server at this time.

    ```
    Testing connection to server1.example.com:1389 ..... Done
    Testing 'cn=Sync User,cn=Root DNs,cn=config' access ..... Denied
    ```

13. To configure the Sync User account on the directory server, press **Enter** to accept the default (yes). Enter the bind DN (`cn=Directory Manager`) and the bind DN password of the directory server so that you can configure the `cn=Sync User` account. The PingDataSync Server creates the Sync User account, tests the base DN, and enables the change log.

    ```
    Created 'cn=Sync User,cn=Root DNs,cn=config'
    Verifying base DN 'dc=example,dc=com' ..... Done
    Enabling cn=changelog .....
    ```

14. Enter the maximum age of the change log entries, or press **Enter** to accept the default.

Configure the Sync Pipe and Sync Classes

The following procedures define a Sync Pipe and two Sync Classes. The first Sync Class is used to match the `accounts` objects. The second Sync Class matches the `group` objects.

1. Continuing from the previous session, enter a name for the Sync Pipe.
2. When prompted to define one or more Sync Classes, enter **yes**.

Configure the accounts Sync Class

1. Enter a name for the Sync Class. For example, type `accounts_sync_class`.
2. If restricting entries to specific subtrees, enter one or more base DNs. If not, press **Enter** to accept the default (no).
3. To set an LDAP search filter, type **yes** and enter the filter `"(accountid=*)"`. Press **Enter** again to continue. This property sets the LDAP filters and returns all entries that match the search criteria to be included in the Sync Class. In this example, specify that any entry with an `accountID` attribute be included in the Sync Class. If the entry does not contain any of these values, it will not be synchronized to the target server.
4. Choose to synchronize all attributes, specific attributes, or exclude specific attributes from synchronization, or press **Enter** to accept the default (all).
5. Specify the operations that will be synchronized for the Sync Class, or press **Enter** to accept the default.

Configure the groups Sync Class

For this example, configure another Sync Class to handle the `groups` objectclass. The procedures are similar to that of the configuration steps for the `account_sync_class` Sync Class.

1. On the Sync Class menu, enter a name for a new sync class, such as `groups_sync_class`.
2. To restrict entries to specific subtrees, enter one or more base DNs.
3. Set an LDAP search filter. Type **yes** to set up a filter and enter the filter `"(objectClass=groupOfUniqueNames)."` This property sets the LDAP filters and returns all entries that match the `groupOfUniqueNames` attribute to be included in the Sync Class. If the entry does not contain any of these values, it will not be synchronized to the target server.
4. Choose to synchronize all attributes, specific attributes, or exclude specific attributes from synchronization, or press **Enter** to accept the default (all).
5. Specify the operations that will be synchronized for the Sync Class, or press **Enter** to accept the default.
6. At the prompt to enter the name of another Sync Class, press **Enter** to continue.
7. On the Default Sync Class Operations menu, press **Enter** to accept the default. The Default Sync Class determines how all entries that do not match any other Sync Class are handled.
8. Review the configuration, and press **Enter** to write the configuration to the server.

Use the `dsconfig` tool to make changes to this configuration. See Configuring the PingDataSync Server for configuration options and details.

Considerations for synchronizing from a database source

When synchronizing from a database to a directory or RDBMS server, the following are recommended:

- **Identify Database Entry Types** – Identify the database entry types that will be synchronized, and:
  - Set the `database-entry-type` property on the JDBC Sync Source (this is required), and make sure the entry types are what the triggers are inserting into the change tracking mechanism.
  - Create a Sync Class per entry type, and set different mappings and rules for each one.
- For each Sync Class, do the following:
  - Make sure the `include-filter` property is set to match the entry type.
  - Create a specific attribute mapping for every database column to be synchronized to an LDAP attribute and set it on the Sync Class. If this is done, the script will not have to know about the schema on the directory side.
  - Create a DN map that recognizes the DNs generated by the script and maps them to the correct location at the destination.
  - Remove the default value of "-all-" from the `auto-mapped-source-attribute` property on the Sync Class, and replace it with the value `objectClass`. The object class for the fetched source entry is determined by the scripted layer. Values from the database should not be automatically mapped to an attribute with the same name, except the `objectclass` attribute, which should map directly for CREATE operations. If this is not done, an error is generated.
  - Change the `destination-correlation-attributes` property to contain the attributes that uniquely represent the database entries on the directory server destination.
- **Avoid Bidirectional Loopback** – Set the `ignore-changes-by-[user|dn]` property on both Sync Sources when configuring for bidirectional synchronization, to make sure that changes are not looped back by the PingDataSync Server.

See Use the create-sync-pipe tool to configure synchronization for details about creating the Sync Pipe.

Synchronize a specific list of database elements

The `resync` command enables synchronizing a specific set of database keys that are read from a JDBC Sync Source file using the `--sourceInputFile` option. The contents of the file are

passed line-by-line into the `listAllEntries()` method of the `JDBCSyncSource` extension, which is used for the Sync Pipe. The method processes the input and returns `DatabaseChangeRecord` instances based on the input from the file.

Perform the following steps to synchronize a specific list of database elements using the `resync` tool:

1. Create a file of JDBC Sync Source elements. There is no set format for the file, but it typically contains a list of primary keys or SQL queries. For example, create a file containing a list of primary keys and save it as `sourceSQL.txt`.

   ```
   user.0
   user.1
   user.2
   user.3
   ```

2. Run the `resync` command with the `--sourceInputFile` option to run on individual primary keys in the file.

   ```
   $ bin/resync --pipe-name "dbsync-pipe" \
     --sourceInputFile sourceSQL.txt
   ```

3. If searching for a specific type of database entry, use the `--entryType` option that matches one of the configured entry types in the JDBCSyncSource.

   ```
   $ bin/resync --pipe-name "dbsync-pipe" \
     --entryType account \
     --sourceInputFile sourceSQL.txt
   ```

Chapter 7: Synchronize through PingDirectoryProxy Servers

Because most data centers deploy directory servers in a proxied environment, the PingDataSync Server can also synchronize data through a proxy server in both load-balanced and entry-balancing deployments. The following proxy endpoints are supported:

- Ping Identity PingDirectoryProxy Servers
- Nokia 8661 Directory Proxy Servers

This chapter details a Sync-through-Proxy deployment and provides background information on how it works. Before setting up the PingDataSync Server, review the *Ping Identity PingDirectoryProxy Server Administration Guide* for information about the PingDirectoryProxy Server.

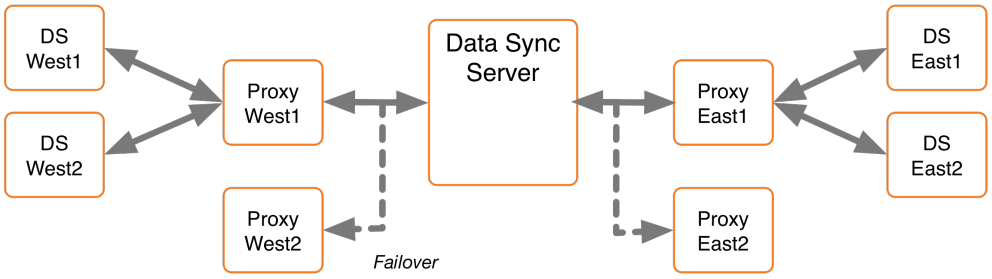
Topics include:

Synchronization through a Proxy Server overview

To handle data synchronization through a proxy server, PingData servers have a `cn=changelog` state management system that supports a token-based API. In a standard, non-proxied configuration, the PingDataSync Server polls the source server for changes, determines if a change is necessary, and fetches the full entry from the source. Then, it finds the corresponding entry in the destination endpoint using correlation rules and applies the minimal set of changes. The server fetches and compares the full entries to make sure it does not synchronize any stale data from the change log.

In a proxied environment, the PingDataSync Server passes the request through a proxy server to the backend set of directory servers. The PingDataSync Server uses the highest priority proxy server designated in its endpoint server configuration and can use others in the event of a failover.

The following illustrates a deployment with two endpoints consisting of a proxy server deployment in front of the backend set of directory servers.



Synchronization Through Proxy Example

Change log operations

When the PingDataSync Server runs a poll for any changes, it sends a get change log batch extended request to the `cn=changelog` backend. The batch request looks for entries in the change log and asks for the server ID, change number, and replica state for each change. The PingDirectoryProxy Server routes the request to a directory server instance, which then returns a changed entry plus a token identifying the server ID, change number and replica state for each change. The PingDirectoryProxy Server then sends a get change log batch response back to the PingDataSync Server with this information. For entry-balancing deployments, the PingDirectoryProxy Server must "re-package" the directory server tokens into its own proxy token to identify the specific data set.

The first time that the PingDataSync Server issues the batch request, it also issues a get server ID request to identify the specific server ID that is processing the extended request. The PingDirectoryProxy Server routes the request to the directory server instance, and then returns a server ID in the response. With the next request, the PingDataSync Server sends a 'route to server' request that specifies the server instance to access again in this batch session. It also issues a server ID request in the event that the particular server is down. This method avoids round-robin server selection and provides more efficient overall change processing.

PingDirectory Server and PingDirectoryProxy Server tokens

The PingDirectory Server maintains a change log database index to determine when to resume sending changes (for ADD, MODIFY, or DELETE operations) in its change log. While a simple stand-alone directory server can track its resume point by the last change number sent, replicated servers or servers deployed in entry balancing environments have a different change number ordering in its change log because updates can come from a variety of sources.

The following illustrates two change logs in two replicated directory servers, server A and B. "A" represents the replica identifier for a replicated subtree in Server A, and "B" represents the replica identifier for the same replicated subtree in server B. The replica identifiers with a hyphen ("-") mark any local, non-replicated but different changes. While the two replicas record all of the changes, the two change logs have two different change number orderings because updates come in at different times.

Server A			Server B		
ChangeNumber	ReplicaIdentifier	ReplicationCSN	ChangeNumber	ReplicaIdentifier	ReplicationCSN
1001	A_{ri}	10	2001	B_{ri}	11
1002	-	-	2002	A_{ri}	10
1003	A_{ri}	15	2003	-	-
1004	B_{ri}	11	2004	B_{ri}	12
1005	B_{ri}	12	2005	A_{ri}	15

Different Change Number Order in Two Replicated Change Logs

To track the change log resume position, the PingDirectory Server uses a change log database index to identify the latest change number position corresponding to the highest replication CSN number for a given replica. This information is encapsulated in a directory server token and returned in the get change log batch response to the PingDirectoryProxy Server. The token has the following format:

```
Directory Server Token: server ID, changeNumber, replicaState
```

For example, if the PingDirectoryProxy Server sends a request for any changed entries, and the directory servers return the change number 1003 from server A and change number 2005 from server B, then each directory server token would contain the following information:

```
Directory Server Token A:
  serverID A, changeNumber 1003, replicaState {15(A)}

Directory Server Token B:
  serverID B, changeNumber 2005, replicaState {12(B), 15(A)}
```

Change log tracking in entry balancing deployments

Change log tracking can become more complex in that a shared area of data can exist above the entry-balancing base DN in addition to each backend set having its own set of changes and tokens. In the following figure, Server A belongs to an entry-balancing set 1, and server B belonging to an entry-balancing set 2. Shared areas that exist above the entry-balancing base DN are assumed to be replicated to all servers. "SA" represents the replica identifier for that shared area on Server A and "SB" represents the replica identifier for the same area on Server B.

Set 1 - Server A			Set 2 - Server B		
ChangeNumber	ReplicaIdentifier	ReplicationCSN	ChangeNumber	ReplicaIdentifier	ReplicationCSN
1001	SA_{ri}	5	2001	SB_{ri}	10
1002	A_{ri}	10	2002	B_{ri}	20
1003	SB_{ri}	15	2003	SA_{ri}	5

Different Change Number Order in Two Replicated Change Logs

The PingDirectoryProxy Server cannot pass a directory server token from the client to the directory server and back again. In an entry-balancing deployment, the PingDirectoryProxy Server must maintain its own token mechanism that associates a directory server token (`changeNumber`, `replicaIdentifier`, `replicaState`) to a particular backend set.

```
Proxy Token:
backendSetID 1: ds-token 1 (changeNumber, replicaIdentifier, replicaState)
backendSetID 2: ds-token 2 (changeNumber, replicaIdentifier, replicaState)
```

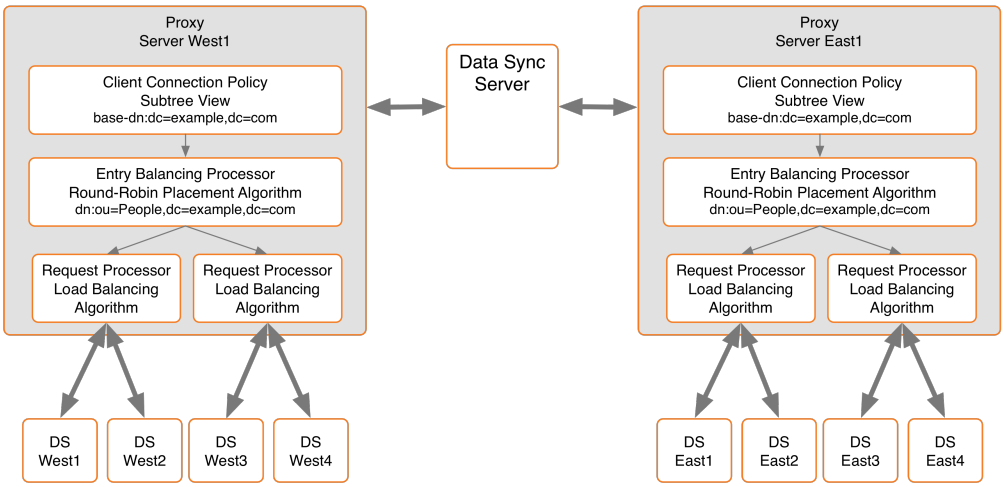
For example, if the PingDirectoryProxy Server returned change 1002 from Server A and change 2002 from Server B, then the Proxy token would contain the following:

```
Proxy Token:
backendSetID 1: ds-token-1 {serverID A, changeNumber 1002, replicaState (5
(SA), 15(A)}
backendSetID 2: ds-token-2 {serverID B, changeNumber 2002, replicaState (10
(SB), 20(B)}
```

For each change entry returned by a backend, the PingDirectoryProxy Server must also decide whether it is a duplicate of a change made to the backend set above the entry-balancing base. If the change is a duplicate, then it is discarded. Otherwise, any new change is returned with a new value of the proxy token.

Example configuration

This following configures synchronization through a proxy and use two endpoints consisting of a PingDirectoryProxy Server with a backend set of PingDirectory Servers: both sets are replicated. The PingDirectoryProxy Server uses an entry-balancing environment for the DN `ou=People,dc=example,dc=com` and provides a subtree view for `dc=example,dc=com` in its client connection policy. For this example, communication is over standard LDAP and failover servers are not installed or designated in the PingDataSync Server.



Example Synchronization Through Proxy Configuration

Configure the source PingDirectory Server

The following procedures configure a backend set of directory servers. The procedure is the same for the source servers and the destination servers in a synchronization topology. For directory server installation and configuration details, see the *Ping Identity PingDirectory Server Administration Guide*.

1. On each backend Directory Server that will participate in synchronization, enable the change log database, either from the command line or by using a `dsconfig` batch file.

```
$ dsconfig --no-prompt set-backend-prop \
  --backend-name changelog \
  --set enabled:true
```

2. Stop the server if it is running, and import the dataset for the first backend set into the first server in the backend set prior to the import.

```
$ bin/stop-server
```

```
$ bin/import-ldif --backendID userRoot --ldifFile ../dataset.ldif
```

```
$ bin/start-server
```

3. On the first server instance in the first backend set, configure replication between this server and the second server in the same backend set.

```
$ bin/dsreplication enable --host1 ldap-west-01.example.com \
  --port1 389 \
  --bindDN1 "cn=Directory Manager" \
  --bindPassword1 password \
  --replicationPort1 8989 \
  --host2 ldap-west-02.example.com \
  --port2 389 \
  --bindDN2 "cn=Directory Manager" \
  --bindPassword2 password \
  --replicationPort2 9989 \
  --adminUID admin \
  --adminPassword admin \
  --baseDN dc=example,dc=com \
  --no-prompt
```

4. Initialize the second server in the backend set with data from the first server in the backend set. This command can be run from either instance.

```
$ bin/dsreplication initialize \
  --hostSource ldap-west-01.example.com \
  --portSource 389 \
  --hostDestination ldap-west-02.example.com \
  --portDestination 389 \
  --baseDN "dc=example,dc=com" \
  --adminUID admin \
  --adminPassword admin \
  --no-prompt
```

5. Run the following command to check replica status.

```
$ bin/dsreplication status \
  --hostname ldap-west-01.example.com \
  --port 389 \
  --adminPassword admin \
  --no-prompt
```

6. Repeat steps 3 through 6 (import, enable replication, initialize replication, check status) for the second backend set.

Configure a Proxy Server

The following procedures configure a proxy server, including defining the external servers and configuring the client-connection policy. The procedure is the same for the source servers and the destination servers in a synchronization topology. For additional changes, use the `dsconfig` tool. For proxy installation and configuration details, see the *Ping Identity PingDirectoryProxy Server Administration Guide*.

1. From the PingDirectoryProxy Server root directory, run the `prepare-external-server` command to set up the `cn=Proxy User` account for access to the backend directory servers. The server tests the connection and creates the `cn=Proxy User` account.

```
$ bin/prepare-external-server --no-prompt \
  --hostname ldap-west-01.example.com \
  --port 389 --bindDN "cn=Directory Manager" \
  --bindPassword password \
  --proxyBindDN "cn=Proxy User,cn=Root DNs,cn=config" \
  --proxyBindPassword pass \
  --baseDN "dc=example,dc=com"
```

2. Repeat step 1 for any other directory server instances.
3. Run the `dsconfig` command to define the external servers and their types. For this example, round-robin load balancing algorithms are defined, which do not require health checks or locations to be specified.

```
$ bin/dsconfig --no-prompt create-external-server \
  --server-name ldap-west-01 \
  --type "ping-identity-ds" \
  --set "server-host-name:ldap-west-01.example.com" \
  --set "server-port:389" \
  --set "bind-dn:cn=Proxy User" \
  --set "password:password" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

```
$ bin/dsconfig --no-prompt create-external-server \
  --server-name ldap-west-02 \
  --type "ping-identity-ds" \
  --set "server-host-name:ldap-west-02.example.com" \
  --set "server-port:389" \
  --set "bind-dn:cn=Proxy User" \
  --set "password:password" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

```
$ bin/dsconfig --no-prompt create-external-server \
  --server-name ldap-west-03 \
  --type "ping-identity-ds" \
  --set "server-host-name:ldap-west-03.example.com" \
  --set "server-port:389" \
  --set "bind-dn:cn=Proxy User" \
  --set "password:password" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

```
$ bin/dsconfig --no-prompt create-external-server
  --server-name ldap-west-04 \
  --type "ping-identity-ds" \
  --set "server-host-name:ldap-west-04.example.com" \
  --set "server-port:389" \
```

```
  --set "bind-dn:cn=Proxy User" \
  --set "password:password" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

4. Create a load-balancing algorithm for each backend set.

```
$ bin/dsconfig --no-prompt create-load-balancing-algorithm \
  --algorithm-name "test-lba-1" \
  --type "round-robin" --set "enabled:true" \
  --set "backend-server:ldap-west-01" \
  --set "backend-server:ldap-west-02" \
  --set "use-location:false" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

```
$ bin/dsconfig --no-prompt create-load-balancing-algorithm \
  --algorithm-name "test-lba-2" \
  --type "round-robin" --set "enabled:true" \
  --set "backend-server:ldap-west-03"
  --set "backend-server:ldap-west-04"
  --set "use-location:false" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

5. Configure the proxying request processors, one for each load-balanced directory server set. A request processor provides the logic to either process the operation directly, forward the request to another server, or hand off the request to another request processor.

```
$ bin/dsconfig --no-prompt create-request-processor \
  --processor-name "proxying-processor-1" --type "proxying" \
  --set "load-balancing-algorithm:test-lba-1" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

```
$ bin/dsconfig --no-prompt create-request-processor \
  --processor-name "proxying-processor-2" --type "proxying" \
  --set "load-balancing-algorithm:test-lba-2" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

6. Define an entry-balancing request processor. This request processor is used to distribute entries under a common parent entry among multiple backend sets. A backend set is a collection of replicated directory servers that contain identical portions of the data. Multiple proxying request processors are used to process operations.

   Next, define the placement algorithm, which selects the server set to use for new add operations to create new entries. In this example, a round-robin placement algorithm forwards LDAP add requests to backends sets.

```
$ bin/dsconfig --no-prompt create-placement-algorithm \
  --processor-name "entry-balancing-processor" \
```

```
  --algorithm-name "round-robin-placement" \
  --set "enabled:true" \
  --type "round-robin" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

7. Define the subtree view that specifies the base DN for the entire deployment.

```
$ bin/dsconfig --no-prompt create-subtree-view \
  --view-name "test-view" \
  --set "base-dn:dc=example,dc=com" \
  --set "request-processor: entry-balancing-processor" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

8. Finally, define a client connection policy that specifies how the client connects to the proxy server.

```
$ bin/dsconfig --no-prompt set-client-connection-policy-prop \
  --policy-name "default" \
  --add "subtree-view:test-view" \
  --bindDN "cn=Directory Manager" \
  --bindPassword pxy-pwd
```

Configuring the PingDataSync Server

Configure the PingDataSync Server once the PingDirectoryProxy Server and its backend set of PingDirectory Server instances are configured and fully functional for each endpoint, which are labeled as `ldap-west` and `ldap-east` in this example. For information on installing and configuring the PingDataSync Server, see Installing the PingDataSync Server.

1. From the PingDataSync Server root directory, run the `create-sync-pipe-config` tool.

```
$ bin/create-sync-pipe-config
```

2. At the Initial Synchronization Configuration Tool prompt, press **Enter** to continue.
3. On the Synchronization Mode menu, press **Enter** to select Standard mode.
4. On the Synchronization Directory menu, choose the option for one-way or bidirectional synchronization.
5. On the First Endpoint Type menu, enter the number for the type of backend data store for the first endpoint. In this example, type the number corresponding to the PingDirectoryProxy Server.

```
>>>> First Endpoint Type
Enter the type of data store for the first endpoint:
1) Ping Identity Directory Server
2) Ping Identity Directory Proxy Server
3) Alcatel-Lucent Directory Server
4) Alcatel-Lucent Proxy Server
5) Sun Directory Server
```

```
6) Microsoft Active Directory
7) Microsoft SQL Server
8) Oracle Database
9) Custom JDBC

b) back
q) quit

Enter choice [1]: 2
```

6. Enter a descriptive name for the first endpoint.
7. Enter the base DN where the PingDataSync Server can search for the entries on the first endpoint server.
8. Specify the type of security when communicating with the endpoint server.
9. Enter the hostname and port of the endpoint server. The PingDataSync Server tests the connection. Repeat this step if configuring another server for failover.
10. Enter the Sync User account that will be used to access the endpoint server, or press **Enter** to accept the default `cn=Sync User,cn=Root DNs,cn=config`. Enter a password for the account.
11. The first endpoint deployment is defined using the PingDirectoryProxy Server (`ldap-west`). Repeat steps 5-10 to define the second proxy deployment (`ldap-east`) on the PingDataSync Server.
12. Prepare the endpoint servers in the topology. This step confirms that the Sync User account is present on each server and can communicate between the PingDataSync Server and the PingDirectoryProxy Servers. In addition to preparing the PingDirectoryProxy Server, the PingDataSync Server prepares the backend set of directory servers as the proxy server passes through the authorization to access these servers.
13. Repeat the previous step to prepare the second endpoint server. If other servers have not been prepared, make sure that they are prior to synchronization.
14. Define the Sync Pipe from proxy 1 to proxy 2. In this example, accept the default "Ping Identity Proxy 1 to Ping Identity Proxy 2."
15. To customize on a per-entry basis how attributes get synchronized, define one or more Sync Classes. Create a Sync Class for the special cases, and use the default Sync Class for all other mappings.
16. For the default Sync Class Operations, specify the operations that will be synchronized.
17. Review the configuration settings, and write the configuration to the PingDataSync Server in the `sync-pipe-cfg.txt` file.

Test the configuration

If the `create-sync-pipe-config` tool was not used to create the synchronization configuration, two properties must be verified on each endpoint: `proxy-server` and `use-changelog-batch-request`. The `proxy-server` property should specify the name of the proxy server. The `use-changelog-batch-request` should be set to `true` on the Sync Source only. The `use-changelog-batch-request` is not available on the destination endpoint.

The PingDataSync Server connection parameters (hostname, port, bind DN, and bind password) are required.

1. The following commands check the properties on a Sync Source.

   On the Sync Source:

   ```
   $ bin/dsconfig --no-prompt \
     get-sync-source-prop \
     --source-name "Ping Identity Proxy 1" \
     --property "proxy-server" \
     --property "use-changelog-batch-request"
   ```

   On the Sync Destination:

   ```
   $ bin/dsconfig --no-prompt \
     get-sync-source-prop \
     --source-name "Ping Identity Proxy 2" \
     --property "proxy-server"
   ```

2. From the server root directory, run the `dsconfig` command to set a flag indicating that the endpoints are PingDirectoryProxy Servers:

   ```
   $ bin/dsconfig --no-prompt \
     set-sync-source-prop \
     --source-name "Ping Identity Proxy 1" \
     --set proxy-server:ldap-west-01 \
     --set use-changelog-batch-request:true
   ```

   ```
   $ bin/dsconfig --no-prompt \
     set-sync-source-prop \
     --source-name "Ping Identity Proxy 2" \
     --set proxy-server:ldap-east-01
   ```

3. Run the `resync --dry-run` command to test the configuration settings for each Sync Pipe and debug any issues.

   ```
   $ bin/resync --pipe-name "Ping Identity Proxy 1 to Ping Identity Proxy 2"
   --dry-run
   ```

4. Run `realtime-sync set-startpoint` to initialize the starting point for synchronization.

   ```
   $ realtime-sync set-startpoint --end-of-changelog \
     --pipe-name "Ping Identity Proxy 1 to Ping Identity Proxy 2" \
     --port 389 \
     --bindDN "cn=Directory Manager" \
     --bindPassword password
   ```

Note
For synchronization through proxy deployments, the `--change-number` option cannot be used with the `realtime-sync set-startpoint` command, because the PingDataSync Server cannot retrieve specific change numbers from the backend directory servers. Use the `--change-sequence-number`, `--end-of-changelog`, or other options available for the tool.

5. Run the `resync` command to populate data on the endpoint destination server if necessary.

```
$ bin/resync --pipe-name "Ping Identity Proxy 1 to Ping Identity Proxy 2"
\
  --numPasses 3
```

6. Start the Sync Pipe using the `realtime-sync start` command.

```
$ bin/realtime-sync start \
  --pipe-name "Ping Identity Proxy 1 to Ping Identity Proxy 2"
```

7. Monitor the PingDataSync Server using the status commands and logs.

Index the LDAP changelog

The PingData PingDirectory Server and the Nokia 8661 Directory Server (3.0 or later) both support attribute indexing in the changelog backend to enable get changelog batch requests to filter results that include only changes of specific attributes. For example, in an entry balanced proxy deployment, the PingDataSync Server sends a get changelog batch request to the Proxy Server, which will send out individual requests to each backend server.

Each directory server that receives a request must iterate over the whole range of changelog entries, and then match entries based on search criteria for inclusion in the batch. The majority of this processing involves determining whether a changelog entry includes changes to a particular attribute or set of attributes, or not. Changelog indexing can dramatically speed up throughput when targeting specific attributes.

Attribute indexing is configured using the `index-include-attribute` and `index-exclude-attribute` properties on the changelog backend. The properties can accept the specific attribute name or special LDAP values "*" to specify all user attributes or "+" to specify all operational attributes.

Perform the following steps to configure changelog indexing:

1. On all source directory servers, enable changelog indexing for the attributes that will be synchronized. Use the `index-include-attribute` and `index-exclude-attribute` properties. The following example specifies that all user attributes (`index-include-attribute:*`) be indexed in the changelog, except the description and location attributes (`index-exclude-attribute:description` and `index-exclude-attribute:location`).

```
$ bin/dsconfig set-backend-prop --backend-name changelog \
  --set "index-include-attribute:*" \
  --set "index-exclude-attribute:description \
  --set "index-exclude-attribute:location
```

Note
There is little performance and disk consumption penalty when using `index-include-attribute:*` with a combination of `index-exclude-attribute` properties, instead of explicitly defining each attribute using `index-include-attribute`. The only cautionary note about using `index-include-attribute:*` is to be careful that unnecessary attributes get indexed.

2. On the PingDataSync Server, configure the `auto-map-source-attributes` property to specify the mappings for the attributes that need to be synchronized.

The PingDataSync Server will write a NOTICE message to the error log when the Sync Pipe first starts, indicating whether the server is using changelog indexing or not.

```
[30/Mar/2016:13:21:36.781 -0500] category=SYNC severity=NOTICE
msgID=1894187256 msg="Sync Pipe 'TestPipe' is not using changelog indexing on
the source server"
```

Changelog synchronization considerations

If the Sync Source is configured with `use-changelog-batch-request=true`, the PingDataSync Server will use the get changelog batch request to retrieve changes from the LDAP changelog. This extended request can contain an optional set of selection criteria, which specifies changelog entries for a specific set of attributes.

The PingDataSync Server takes the union of the source attributes from DN mappings, attribute mappings, and the `auto-mapped-source-attributes` property on the Sync Class to create the selection criteria. However, if it encounters the value "-all-" in the `auto-mapped-source-attributes` property, it cannot make use of selection criteria because the Sync Pipe is interested in all possible source attributes.

When the PingDirectory Server receives a get changelog request that contains selection criteria, it returns changelog entries for one or more of the attributes that meet the criteria.

- For ADD and MODIFY changelog entries, the changes must include at least one attribute from the selection criteria.
- For MODDN changelog entries, one of the RDN attributes must match the selection criteria.
- For DELETE changelog entries, one of the `deletedEntryAttrs` much match the selection criteria.

If `auto-mapped` is not set to `all` source attributes, at least one should be configured to show up in the `deletedEntryAttrs` (with the `changelog-deleted-entry-include-attribute` property on the changelog backend).

Another way to do this is to set `use-reversible-form` to `true` on the changelog backend. This includes all attributes in the `deletedEntryAttrs`.

Chapter 8: Synchronize in Notification Mode

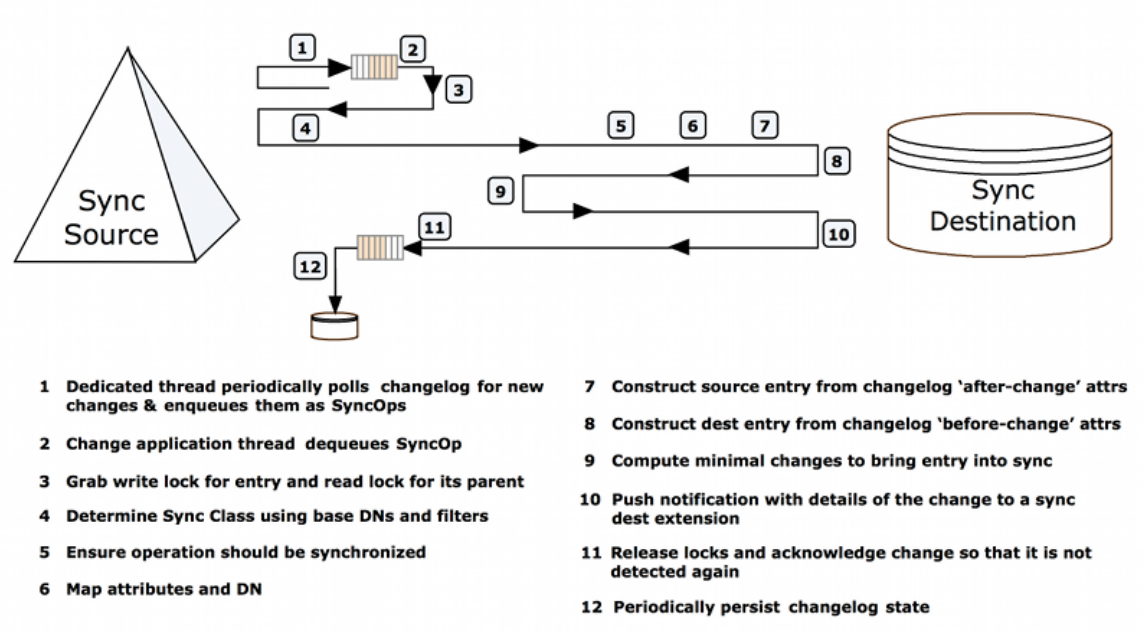
The PingDataSync Server supports a notification synchronization mode that transmits change notifications on a source endpoint to third-party destination applications. As with standard mode, notifications can be filtered based on the type of entry that was changed, the specific attributes that were changed, and the type of change (ADD, MODIFY, DELETE). The PingDataSync Server can send a notification to arbitrary endpoints by using a custom server extension.

Topics include:

Notification mode overview

The PingDataSync Server supports standard and notification synchronization modes. Notification Mode polls the directory server's LDAP change log for changes on any entry but skips the fetch and compare phases of processing of Standard Mode. Instead, the Sync Destination is notified of the change regardless of the current state of that entry at the source or destination. The PingDataSync Server accesses state information on the change log to reconstruct the before-and-after values of any modified attribute (for example, for MODIFY change operation types). It passes in the change information to a custom server extension based on the Server SDK.

Third-party libraries can be employed to customize the notification message to an output format required by the client application or service. For example, the server extension can use a third-party XML parsing library to convert the change notifications to a SOAP XML format. Notification mode can only be used with an PingDirectory Server, Nokia 8661 Directory Server, PingDirectoryProxy Server, or Nokia 8661 Directory Proxy Server as the source endpoint.



Notification Mode Synchronization Change Flow

The PingDataSync Server can use notification mode with any type of endpoint; therefore, it is not an absolute requirement to have a custom server extension in your system. For example, it is possible to set up a notification Sync Pipe between two LDAP server endpoints.

Implementation Considerations

Before implementing and configuring a Sync Pipe in notification mode, answer the following questions:

- What is the interface to client applications?
- What type of connection logic is required?
- How will the custom server extension handle timeouts and connection failures?
- What are the failover scenarios?
- What data needs to be included in the change log?
- How long do the change log entries need to be available?
- What are the scalability requirements for the system?
- What attributes should be used for correlation?
- What should happen with each type of change?
- What mappings must be implemented?

Use the Server SDK and LDAP SDK

To support notification mode, the Server SDK provides a `SyncDestination` extension to synchronize with any client application. The PingDataSync Server engine processes the notification and makes it available to the extension, which can be written in Java or Groovy. This generic extension type can also be used for standard synchronization mode.

Similar to database synchronization, the custom server extension is stored in the `<server-root>/lib/groovy-scripted-extensions` folder (for Groovy-based extensions) or the jar file in the `<server-root>/lib/extensions` folder (for Java-based extensions) prior to configuring the PingDataSync Server for notification mode. Groovy scripts are compiled and loaded at runtime.

The Server SDK's `SyncOperation` interface represents a single synchronized change from the Sync Source to the Sync Destination. The same `SyncOperation` object exists from the time a change is detected, through when the change is applied at the destination.

The LDAP SDK's `UnboundIDChangelogEntry` class (in the `com.unboundid.ldap.sdk.unboundidds` package) has high level methods to work with the `ds-changelog-before-value`, `ds-changelogafter-values`, and `ds-changelog-entry-key-attr-values` attributes. The class is part of the commercial edition of the LDAP SDK for Java and is installed automatically with the PingDataSync Server. For detailed information and examples, see the LDAP SDK Javadoc.

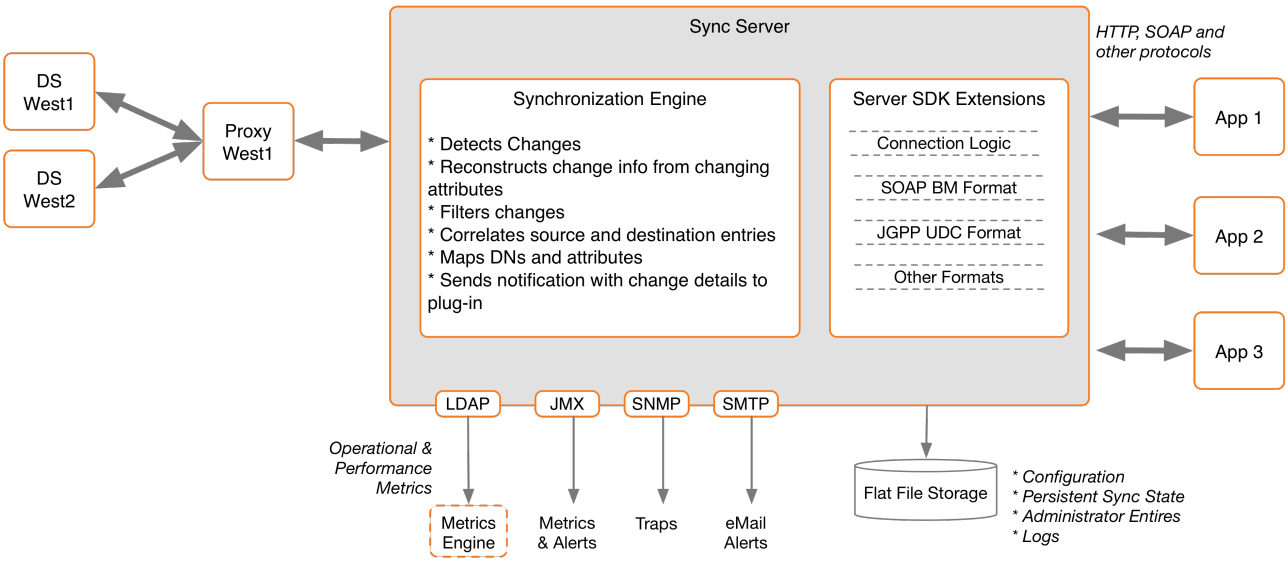
Notification mode architecture

Notification mode, a configuration setting on the Sync Pipe, requires a one-way directional Sync Pipe from a source endpoint topology to a target client application. The PingDataSync Server detects the changes in the PingDirectory Server's LDAP change log, filters the results specified in the Sync Classes, applies any DN and attribute mappings, then reconstructs the change information from the change log attributes. A server extension picks up the notification arguments from the `SyncOperation` interface (part of the Server SDK) and converts the data to the desired output format. The server extension establishes the connections and protocol

logic to push the notification information to the client applications or services. All of the operations, administration, and management functions available in standard mode, such as monitoring, (LDAP, JMX, SNMP), alerts (JMX, SNMP, SMTP), and logging features are the same for notification mode.

Note
The Server SDK includes documentation and examples on how to create a directory server extension to support notification mode.

For a given entry, the PingDataSync Server sends notifications in the order that the changes occurred in the change log even if a modified attribute has been overwritten by a later change. For example, if an entry's `telephoneNumber` attribute is changed three times, three notifications will be sent in the order they appeared in the change log.



Notification Mode Architecture

Sync Source requirements

A separate Sync Pipe is required for each client application that should receive a notification. The Sync Sources must consist of one or more instances of the following directory or proxy servers with the PingDataSync Server:

- Ping IdentityPingDirectory Server and PingDirectoryProxy Server (version 3.0.5 or later)
- Nokia 8661 Directory Server
- The Sync Destination can be of any type

Note While the PingDirectoryProxy Server and Nokia 8661 Directory Proxy Server can front other vendor's directory servers, such as Active Directory and Sun DSEE, for processing LDAP operations, the PingDataSync Server cannot synchronize changes from these sources through a proxy server.

Synchronizing changes directly from Active Directory and Sun DSEE cannot be done with notification mode.

Failover Capabilities

For sync source failovers, configure replication between the Directory Servers to ensure data consistency between the servers. A PingDirectoryProxy Server can also front the backend PingDirectory Server set to redirect traffic, if connection to the primary server fails. A PingDirectoryProxy Server must be used for synchronizing changes in an entry-balancing environment. Once the primary PingDirectory Server is online, it assumes control with no information loss as its state information is kept across the backend PingDirectory Servers.

For sync destination failovers, connection retry logic must be implemented in the server extension, which will use the Sync Pipe's advanced property settings to retry failed operations. There is a difference between a connection retry and an operation retry. An extension should not retry operations because the PingDataSync Server does this automatically. However, the server extension is responsible for re-establishing connections to a destination that has gone down, or failing over to an alternate server. The server extension can also be designed to trigger its own error- handling code during a failed operation.

For PingDataSync Server failovers, the secondary PingDataSync Servers will be at or slightly behind the state where the primary server initiated a failover. Both primary and secondary PingDataSync Servers track the last failed acknowledgement, so once the primary server fails over to a secondary server, the secondary server will not miss a change.

Note

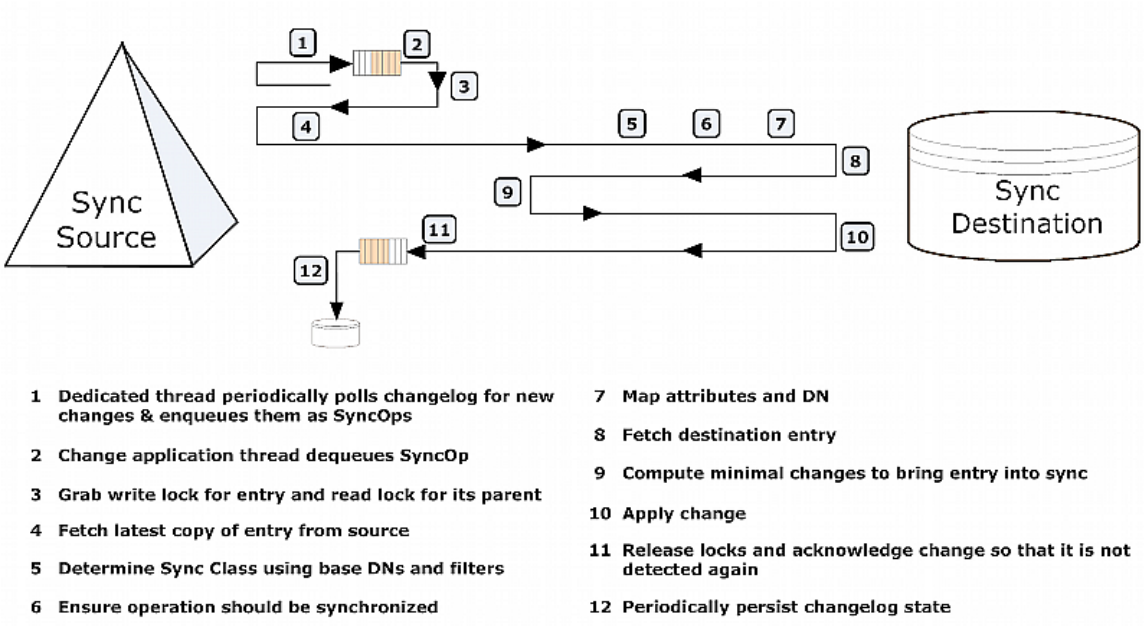
If failover is a concern between PingDataSync Servers, change the `sync-failover-polling-interval` property from 7500 ms to a smaller value. This will result in a quicker failover, but will marginly increase traffic between the two PingDataSync Servers. The `sync-failover-connection-timeout` and `sync-failover-response-timeout` properties may also be updated to use different failover timeout durations. Use `dsconfig` to access the property on the Global Sync Configuration menu.

Notification Sync Pipe change flow

Multi-threaded Sync Pipes allow the PingDataSync Server to process multiple notifications in parallel in the same manner as synchronizing changes in standard mode, which increases throughput and offsets network latency. A single change-detection thread pulls in batches of change log entries and queues them internally. To guarantee consistency, the PingDataSync Server's internal locking mechanisms ensure the following properties:

- Changes to the same entry will be processed in the same order that they appear in the change log.
- Changes to parent entries will be processed before changes to its children.
- Changes to entries with the same RDN value are handled sequentially.

The number of concurrent threads is configurable on the Sync Pipe using the `num-worker-threads` property. In general, single-threading should be avoided.



Notification Sync Pipe Change Flow

Configure Notification mode

The PingDataSync Server supports notification mode with the following components:

Use the create-sync-pipe-config tool

The `create-sync-pipe-config` tool supports the configuration of notification mode. Any pre-existing Sync Sources can be read from the local configuration (in the `config.ldif` file).

No resync command functionality

The `resync` function is disabled on a Sync Pipe in notification mode as its functionality is not supported in this implementation. Notification mode views the directory server's change log as a rolling set of data that pushes out change notifications to its target application.

LDAP change log features required for notifications

The PingDirectory Server and the Nokia 8661 Directory Server require the following advanced global change log properties: `changelog-max-before-after-values` and `changelog-include-key-attribute`.

These properties are enabled and configured during the `create-sync-pipe-config` configuration process on the PingDataSync Server. The properties can also be enabled on the directory servers using the `dsconfig` advanced properties setting on the Backend Changelog menu.

changelog-include-key-attribute

The `changelog-include-key-attribute` property specifies one or more attributes that should always be included in the change log entry. These are attributes needed to correlate entries between the source and destination, such as `uid`, `employeeNumber`, or `mail`. These properties are also needed for evaluating any filters in the Sync Class. For example, if notifications are only sent for user entries, and the Sync Class included the filter `(objectclass=people)`, the `objectclass` attribute must be configured as a `changelog-include-key-attribute` so that the Sync Pipe can evaluate the inclusion criteria when processing the change. In standard mode, values needed in the filter are read from the entry itself after it is fetched instead of from the changelog entry. These attributes are always included in a change log entry, also called a change record, regardless if they have changed or not.

The `changelog-include-key-attribute` property causes the current (after-change) value of the specified attributes to be recorded in the `ds-changelog-entry-key-attr-values` attribute on the change log entry. This applies for all change types. During a delete operation, the values are from the entry before it was deleted. The key values are recorded on every change and override any settings configured in the `changelog-include-attribute`, `changelog-exclude-attribute`, `changelog-deleted-entry-include-attribute`, or `changelog-deleted-entry-exclude-attribute` properties in the directory server changelog (see the *Ping Identity PingDirectory Server Configuration Reference* for more information).

Normal LDAP to LDAP synchronization topologies typically use `dn` as a correlation attribute. If `dn` is used as a correlation attribute only, the `changelog-include-key-attribute` property does not need to be set. However, if another attribute is used for correlation, this property must be set during the Sync Pipe configuration.

The LDAP change log attribute, `ds-changelog-entry-key-attr-values`, stores the attribute that is always included in a change log entry on every change for correlation purposes. In addition to regular attributes, virtual and operational attributes can be specified as entry keys.

To view an example, see the *Ping Identity PingDirectory Server Administration Guide*.

changelog-max-before-after-values

The `changelog-max-before-after-values` property specifies the maximum number of "before and after" values (default 200) that should be stored for any changed attribute in the change log. Also, when enabled, it will add the `ds-changelog-before-values` and `ds-changelog-after-values` attributes to any change record that contains changes (for Modify and ModifyDN).

The main purpose of the `changelog-max-before-after-values` property is to ensure that an excessively large number of changes is not stored for multi-valued attributes. In most cases, the directory server's schema defines a multi-valued attribute to be unlimited in an entry. For example, if a group entry whose member attribute references 10000 entries, the desire may be to not have all of the attributes if a new member added.

If either the `ds-changelog-before-values` or the `ds-changelog-after-values` attributes exceed the count set in the `changelog-max-before-after-values` property, the attribute values are no longer stored in a change record but its attribute name and number is stored in

the `ds-changelog-attr-exceeded-max-values-count` attribute, which appears in the change record.

In addition to this property, set the `use-reversible-form` property to `TRUE`. This guarantees that sufficient information is stored in the change log for all operation types to be able to replay the operations at the destination. The `create-sync-pipe-config` tool configures these properties when it prepares the servers.

The `changelog-max-before-after-values` property configures the following change log attributes:

- `ds-changelog-before-values` – Captures all "before" values of a changed attribute. It will store up to the specified value in the `changelog-max-before-after-values` property (default 200).
- `ds-changelog-after-values` – Captures all "after" values of a changed attribute. It will store up to the specified value in the `changelog-max-before-after-values` property (default 200).
- `ds-changelog-attr-exceeded-max-values-count` – Stores the attribute names and number of before and after values on the change log entry after the maximum number of values (set by the `changelog-max-before-after-values` property) has been exceeded. This is a multi-valued attribute with the following format:

  ```
  attr=attributeName,beforeCount=200,afterCount=201
  ```

  where `attributeName` is the name of the attribute and the `beforeCount` and `afterCount` are the total number of values for that attribute before and after the change, respectively. In either case (before or after the change), if the number of values is exceeding the maximum, those values will not be stored.

LDAP change log for Notification and Standard Mode

Both Notification and Standard mode Sync Pipes can consume the same LDAP Change Log without affecting the other. Standard mode polls the change record in the change log for any modifications, fetches the full entries on the source and the destination, and then compares them for the specific changes. Notification mode gets the before and after values of a changed attribute to reconstruct an entry, and bypasses the fetch-and-compare phase. Both can consume the same LDAP Change Log with no performance loss or conflicts.

Note
If the configuration obtains the change log through a PingDirectoryProxy Server, the contents of the change log will not change as it is being read from the change logs on the directory server backend.

Implementing the Server Extension

Notification mode relies heavily on the server extension code to process and transmit the change using the required protocol and data formats needed for the client applications. Create the extension using the Server SDK, which provides the APIs to develop code for any destination endpoint type. The Server SDK's documentation (Javadoc and examples) is

delivered with the Server SDK built-in zip format. The SDK provides all of the necessary classes to extend the functionality of the PingDataSync Server without code changes to the core product. Once the server extension is in place, use other third-party libraries to transform the notification to any desired output format.

Consider the following when implementing the extension:

- **Use the manage-extension Tool** – Use the `manage-extension` tool in the `bin` directory or `bat` directory (Windows) to install or update the extension. See Managing Extensions for more information.
- **Review the Server SDK Package** – Review Server SDK documentation and examples before building and deploy a Java or Groovy extension.
- **Connection and Protocol Logic** – The Server SDK extension must manage the notification connection and protocol logic to the client applications.
- **Implementing Extensions** – Test the create methods, the delete methods, and the modify methods for each entry type. Update the configuration as needed. Finally, package the extensions for deployment. Logging levels can be increased to include more details.
- **Use the SyncOperation Type** – The `SyncOperation` class encapsulates everything to do with a given change. Objects of this type are used in all of the synchronization SDK extensions. See the Server SDK Javadoc for the `SyncOperation` class for information on the full set of methods.
- **Use the EndpointException Type** – The Sync Destination type offers an exception type called `EndpointException` to extend a standard Java exception and provide custom exceptions. There is also logic to handle LDAP exceptions, using the LDAP SDK.
- **About the PostStep result codes** – The `EndpointException` class uses `PostStep` result codes that are returned in the server extension:
  - `retry_operation_limited` – Retry a failed attempt up to the limit set by `max_operation_attempts`.
  - `retry_operation_unlimited` – Retry the operation an unlimited number of times until a success, abort, or `retried_operation_limited`. This should only be used when the destination endpoint is unavailable.
  - `abort_operation` – Abort the current operation without any additional processing.
- **Use the ServerContext class for logging** – The `ServerContext` class provides several logging methods which can be used to generate log messages and/or alerts from the scripted layer: `logMessage()`, `sendAlert()`, `debugCaught()`, `debugError()`, `debugInfo()`, `debugThrown()`, `debugVerbose()`, and `debugWarning()`. These are described in the Server SDK API Javadocs. Logging related to an individual `SyncOperation` should be done with the `SyncOperation#logInfo` and `SyncOperation#logError` methods.

- **Diagnosing Script Errors** – When a Groovy extension does not behave as expected, first look in the error log for stack traces. If `classLoader` errors are present, the script could be in the wrong location or may not have the correct package. Groovy checks for errors at runtime. Business logic errors must be systematically found by testing each operation. Make sure logger levels are set high enough to debug.

Configuring the Notification Sync Pipe

The following procedure configures a one-way Sync Pipe with a PingDirectory Server as the Sync Source and a generic sync destination. The procedure uses the `create-sync-pipe-config` tool in interactive command-line mode and highlights the differences for configuring a Sync Pipe in notification mode.

Considerations for Configuring Sync Classes

When configuring a Sync Class for a Sync Pipe in notification mode, consider the following:

- Exclude any operational attributes from synchronizing to the destination so that its before and after values are not recorded in the change log. For example, the following attributes can be excluded: `creatorsName`, `createTimeStamp`, `ds-entry-unique-id`, `modifiersName`, and `modifyTimeStamp`. Filter the changes at the change log level instead of making the changes in the Sync Class to avoid extra configuration settings with the following:
  - Use the directory server's `changelog-exclude-attribute` property with (`+`) to exclude all operational attributes (`change-log-exclude-attribute:+`).
  - Configure a Sync Class that sets the `excluded-auto-mapped-source-attributes` property to each operational attribute to exclude from the synchronization process.
  - Use the directory server's `changelog-exclude-attribute` property to specify each operational attribute to exclude in the synchronization process. Set the configuration using the `dsconfig` tool on the directory server Change Log Backend menu. For example, `set changelog-exclude-attribute:modifiersName`.
- Use the `destination-create-only-attribute` advanced property on the Sync Class. This property sets the attributes to include on CREATE operations only.
- Use the `replace-all-attr-values` advanced property on the Sync Class. This property specifies whether to use the ADD and DELETE modification types (reversible), or the REPLACE modification type (non-reversible) for modifications to destination entries. If set to `true`, REPLACE is used.
- If targeting specific attributes that require higher performance throughput, consider implementing change log indexing. See Synchronizing Through Proxy Servers for more information.

Creating the Sync Pipe

The initial configuration steps show how to set up a single Sync Pipe from a directory server instance to a generic Sync Destination.

Before starting:

- Place any third-party libraries in the `<server-root>/lib/extensions` folder.
- Implement a server extension for any custom endpoints and place it in the appropriate directory.

1. If necessary, start the PingDataSync Server:

   ```
   $ bin/start-server
   ```

2. Run the `create-sync-pipe-config` tool.

   ```
   $ bin/create-sync-pipe-config
   ```

3. At the Initial Synchronization Configuration Tool prompt, press **Enter** to continue.
4. On the Synchronization Mode menu, select the option for notification mode.
5. On the Synchronization Directory menu, enter the option to create a one-way Sync Pipe in notification mode from directory to a generic client application.

Configuring the Sync Source

1. On the Source Endpoint Type menu, enter the option for the Sync Source type.
2. Choose a pre-existing Sync Source, or create a new sync source.
3. Enter a name for the Source Endpoint and a name for the Sync Source.
4. Enter the base DN for the directory server used for LDAP searches, such as `dc=example,dc=com`, and press **Enter** to return to the menu. If entering more than one base DN, make sure they do not overlap.
5. On the Server Security menu, select the type of communication that the PingDataSync Server will use with endpoint servers.
6. Enter the host and port of the first Source Endpoint server. The Sync Source can specify a single server or multiple servers in a replicated topology. The PingDataSync Server contacts this first server if it is available, then contacts the next highest priority server if the first server is unavailable. The server tests the connection.
7. On the Sync User Account menu, enter the DN of the sync user account and password, or press **Enter** to accept the default, `cn=Sync User,cn=Root DNs,cn=config`. This account allows the PingDataSync Server to access the source endpoint server.

Configure the Destination Endpoint Server

1. On the Destination Endpoint Type menu, select the type of data store on the endpoint server. In this example, select the option for Custom.
2. Enter a name for the Destination Endpoint and a name for the Sync Destination.
3. On the Notifications Setup menu, select the language (Java or Groovy) used to write the server extension.
4. Enter the fully qualified name of the Server SDK extension that implements the abstract class. A Java, extension should reside in the `/lib/extensions` directory. A Groovy script should reside in the `/lib/groovy-scripted-extensions` directory.
5. Configure any user-defined arguments needed by the server extension. Typically, these are connection arguments, which are defined by the extension itself. The values are then entered here and stored in the server configuration.
6. Configure the maximum number of before and after values for all changed attributes. Notification mode requires this. Set the cap to something well above the maximum number of values that any synchronized attribute will have. If this cap is exceeded, the PingDataSync Server will issue an alert. For this example, we accept the default value of 200.

   ```
   Enter a value for the max changelog before/after values,
   or -1 for no limit [200]:
   ```

7. Configure any key attributes in the change log that should be included in every notification. These attributes can be used to find the destination entry corresponding to the source entry, and will be present whether or not the attributes changed. Later, any attributes used in a Sync Class include-filter should also be configured as key attributes in the Sync Class.
8. In both standard and notification modes, the Sync Pipe processes the changes concurrently with multiple threads. If changes must be applied strictly in order, the number of Sync Pipe worker threads will be reduced to 1. This will limit the maximum throughput of the Sync Pipe.

The rest of the configuration steps follow the same process as a standard synchronization mode Sync Pipe. See About the Sync User Account for more information.

Access control filtering on the Sync Pipe

The PingDataSync Server provides an advanced Sync Pipe configuration property, `filter-changes-by-user`, that performs access control filtering on a changelog entry for a specific user.

Since the changelog entry contains data from the target entry, the access controls filter out attributes that the user does not have the privileges to see before it is returned. For example, values in the changes, `ds-changelog-before-values`, `ds-changelog-after-values`, `ds-changelog-entry-key-attr-values`, and `deletedEntryAttrs` are filtered out through access control instructions.

Note
This property is only available for notification mode and can be configured using the `create-sync-pipe-config` or the `dsconfig` tool.

The source server must be an Ping Identity PingDirectory Server or Nokia 8661 Directory Server, or an Ping Identity PingDirectoryProxy Server or Nokia 8661 Directory Proxy Server that points to an Ping Identity PingDirectory Server or Nokia 8661 Directory Server.

Considerations for access control filtering

- The directory server will not return the changelog entry if the user is not allowed to see the target entry.
- The directory server strips out any attributes that the user is not allowed to see.
- If no changes are left in the entry, no changelog entry will be returned.
- If only some attributes are stripped out, the changelog entry will be returned.
- Access control filtering on a specific attribute value is not supported. Either all attribute values are returned or none.
- If a sensitive attribute policy is used to filter attributes when a client normally accesses the directory server, this policy will not be taken into consideration during notifications since the Sync User is always connecting using the same method. Configure access controls to filter out attributes, not based on the type of connection made to the server, but based on who is accessing the data. The `filter-changes-by-user` property will be able to evaluate if that person should have access to these attributes.

Configure the Sync Pipe to filter changes by access control instructions

1. Set the `filter-changes-by-user` property to filter changes based on access controls for a specific user.

```
$ bin/dsconfig set-sync-pipe-prop \
  --pipe-name "Notifications Sync Pipe" \
  --set "filter-changes-by-user:uid=admin,dc=example,dc=com"
```

2. On the source directory server, set the `report-excluded-changelog-attributes` property to include the names of users that have been removed through access control filtering. This will allow the PingDataSync Server to warn about attributes that were supposed to be synchronized but were filtered out. This step is recommended but not required.

```
$ bin/dsconfig set-backend-prop \
--backend-name "changelog" \
  --set "report-excluded-changelog-attributes:attribute-names"
```

Note

The PingDataSync Server only uses the `attribute-names` setting for the PingDirectory Server's `report-excluded-changelog-attributes` property. It does not use the `attribute-counts` setting for the property.

Chapter 9: Configure synchronization with SCIM

The PingDataSync Server provides data synchronization between directory servers or proxy servers and System for Cross-domain Identity Management (SCIM) applications over HTTP. Synchronization can be done with custom SCIM applications, or with the PingData PingDirectory Server and PingDirectoryProxy Server configured as SCIM servers using the SCIM extension.

Topics include:

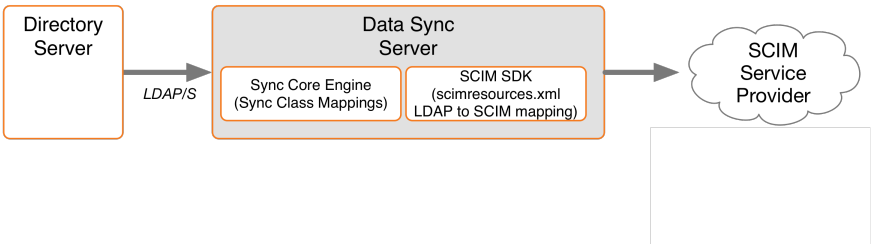
Synchronize with a SCIM Sync Destination overview

The SCIM protocol is designed to make managing user identity in cloud-based applications and services easier. SCIM enables provisioning identities, groups, and passwords to, from, and between clouds. The PingDataSync Server can be configured to synchronize with SCIM service providers.

Note
Both the Ping Identity PingDirectory Server and PingDirectoryProxy Server can be configured to be SCIM servers using the SCIM HTTP Servlet Extension.

The PingDataSync Server is LDAP-centric and operates on LDAP attributes. The SCIM Sync Destination server component acts as a translation layer between a SCIM service provider's schema and an LDAP representation of the entries. While the PingDataSync Server is LDAP-centric and typically at least one endpoint is an LDAP Directory Server, this is not a strict requirement. For example, a JDBC to SCIM sync pipe can be configured.

The PingDataSync Server contains sync classes that define how source and destination entries are correlated. The SCIM Sync Destination contains its own mapping layer, based on `scimresources.xml` that maps LDAP schema to and from SCIM.



Synchronizing with a SCIM Sync Destination

Note
The PingDataSync Server can only use SCIM as a Sync Destination. There is no mechanism in the SCIM protocol for detecting changes, so it cannot be used as a Sync Source.

SCIM destination configuration objects

The `SCIMSyncDestination` object defines a SCIM service provider Sync Pipe destination that is accessible over HTTP through the SCIM protocol. It is configured with the following properties:

- `server` – Specifies the names of the SCIM External Servers that are used as the destination of synchronization.

- `resource-mapping-file` – Specifies the path to the `scim-resources.xml` file, a configuration file that defines the SCIM schema and maps it to the LDAP schema. This file is located in `<server root>/config/scim-resources.xml` by default. This file can be customized to define and expose deployment-specific resources.
- `rename-policy` – Specifies how to handle the rename of a SCIM resource.

The SCIM Sync Destination object is based on the SCIM SDK. Before configuring a SCIM destination, review the following documents on the Simple Cloud web site:

- SCIM Core Schema
- SCIM REST API

Considerations for synchronizing to a SCIM destination

When configuring an LDAP to SCIM Sync Pipe, consider the following:

- **Use `scim-resources.xml` for Attribute and DN Mappings** – There are two layers of mapping: once at the Sync Class level and again at the SCIM Sync Destination level in the `scim-resources.xml` file. To reduce complexity, do all possible mappings in the `scim-resources.xml` file.
- **Avoid Groups Unless the SCIM ID is DN Based** – Group synchronization is supported if the SCIM ID is based on the DN. If the SCIM ID is not the DN itself, it must be one of the components of the RDN, meaning that the DNs of group members must contain the necessary attribute.
- **SCIM Modifies Entries Using PUT** – The SCIM Sync Destination modifies entries using the full HTTP PUT method. For every modify, SCIM replaces the entire resource with the updated resource. For information about the implications of this on password updates, see Password Considerations with SCIM.

Renaming a SCIM resource

The SCIM protocol does not support changes that require the SCIM resource to be renamed, such as a MODDN operation. Instead, when a change is detected to an attribute value that is used as part of the SCIM ID attribute, the PingDataSync Server handles it in one of the following ways:

- Deletes the specified SCIM resource and then adds the new resource with the new SCIM ID.
- Adds the new resource with the new SCIM ID and then deletes the old resource.
- Skips the rename portion of the change. If renames are expected on the source endpoint, a careful set of destination-correlation attributes should be chosen so that the destination can still be found after it is renamed on the source.

Configure this by setting the `rename-policy` property of the SCIM Sync Destination.

Password considerations with SCIM

Because the SCIM sync destination modifies entries using a full PUT method, special considerations need to be made for password attributes. An Ping Identity SCIM Server allows password attributes to be omitted from a change when they have not been modified by an operation. This prevents passwords from inadvertently being overwritten during the PUT operation, which does not include the password attribute. Ideally, other SCIM service providers will not wipe a password because a PUT request does not contain it. Check with the SCIM vendor to confirm this behavior before starting a SCIM sync pipe.

Configure synchronization with SCIM

Configure synchronization with SCIM using the `create-sync-pipe-config` utility and the `dsconfig` command. Configuring synchronization between an LDAP server and a SCIM service provider includes the following:

- Configure one external server for every physical endpoint.
- Configure the Sync Source server and designate the external servers that correspond to the source server.
- Configure the Sync Destination server and designate the external servers that correspond to the SCIM sync destination.
- Configure the LDAP to SCIM Sync Pipe.
- Configure the Sync Classes. Each Sync Class represents a type of entry that needs to be synchronized. When specifying a Sync Class for synchronization with a SCIM service provider, avoid including attribute and DN mappings. Instead use the Sync Class to specify the operations to synchronize and which correlation attributes to use.
- Set the evaluation order for the Sync Classes to define the processing precedence for each class.
- Configure the `scim-resources.xml` file. If possible, change the `<resourceIDMapping>` element(s) to use whatever the SCIM Service Provider uses as the SCIM ID.
- Set Up Communication for each External Server. Run `prepare-endpoint-server` once for every LDAP external server that is part of the Sync Source.
- Use `realtime-sync` to start the Sync Pipe.

Configure the external servers

Perform the following to configure an external server for each host in the deployment:

1. Configure an PingDirectory Server as an external server, which will later be configured as a Sync Source. On the PingDataSync Server, run the following `dsconfig` command:

```
$ bin/dsconfig create-external-server \
  --server-name source-ds \
  --type ping-identity-ds \
  --set server-host-name:ds1.example.com \
  --set server-port:636 \
  --set "bind-dn:cn=Directory Manager" \
  --set password:secret \
  --set connection-security:ssl \
  --set key-manager-provider:Null \
  --set trust-manager-provider:JKS
```

2. Configure the SCIM server as an external server, which will later be configured as a Sync Destination. The `scim-service-url` property specifies the complete URL used to access the SCIM service provider. The `user-name` property specifies the account used to connect to the SCIM service provider. By default, the value is `cn=Sync User,cn=Root DNs,cn=config`. Some SCIM service providers may not have the user name in DN format.

```
$ bin/dsconfig create-external-server \
  --server-name scim \
  --type scim \
  --set scim-service-url:https://scim1.example.com:8443 \
  --set "user-name:cn=Sync User,cn=Root DNs,cn=config" \
  --set password:secret \
  --set connection-security:ssl \
  --set hostname-verification-method:strict \
  --set trust-manager-provider:JKS
```

Configure the PingDirectory Server Sync Source

Configure the Sync Source for the synchronization network. More than one external server can be configured to act as the Sync Source for failover purposes. If the source is an PingDirectory Server, also configure the following items:

- Enable the changelog password encryption plug-in on any directory server that will receive password modifications. This plugin intercepts password modifications, encrypts the password, and adds an encrypted attribute to the change log entry.
- Configure the `changelog-deleted-entry-include-attribute` property on the changelog backend, so that the PingDataSync Server can record which attributes were removed during a DELETE operation.

Perform the following steps to configure the Sync Source:

1. Run `dsconfig` to configure the external server as the Sync Source. Based on the previous example where the PingDirectory Server was configured as `source-ds`, run the following command:

```
$ bin/dsconfig create-sync-source --source-name source \
  --type ping-identity \
```

```
  --set base-dn:dc=example,dc=com \
  --set server:source-ds \
  --set use-changelog-batch-request:true
```

2. Enable the change log password encryption plug-in on any server that will receive password modifications. The encryption key can be copied from the output, if displayed, or accessed from the `<server-root>/bin/sync-pipe-cfg.txt` file, if the `create-sync-pipe-config` tool was used to create the sync pipe.

```
$ bin/dsconfig set-plugin-prop \
  --plugin-name "Changelog Password Encryption" \
  --set enabled:true \
  --set changelog-password-encryption-key:<key>
```

3. On the PingDataSync Server, set the decryption key used to decrypt the user password value in the change log entries. The key allows the user password to be synchronized to other servers that do not use the same password storage scheme.

```
$ bin/dsconfig set-global-sync-configuration-prop \
  --set changelog-password-decryption-key:ej5u9e39pq-68
```

4. Configure the `changelog-deleted-entry-include-attribute` property on the changelog backend.

```
$ bin/dsconfig set-backend-prop --backend-name changelog \
  --set changelog-deleted-entry-include-attribute:objectClass
```

Configure the SCIM Sync Destination

Configure the SCIM Sync Destination to synchronize data with a SCIM service provider. Run the `dsconfig` command:

```
$ bin/dsconfig create-sync-destination \
  --destination-name scim \
  --type scim \
  --set server:scim
```

Configure the Sync Pipe, Sync Classes, and evaluation order

Configure a Sync Pipe for LDAP to SCIM synchronization, create Sync Classes for the Sync Pipe, and set the evaluation order index for the Sync Classes.

Note
The Synchronization mode must be set to Standard. Notification Mode cannot be used with SCIM.

1. Once the source and destination endpoints are configured, configure the Sync Pipe for LDAP to SCIM synchronization. Run the `dsconfig` command to configure an LDAP-to-SCIM Sync Pipe:

```
$ bin/dsconfig create-sync-pipe \
  --pipe-name ldap-to-scim \
```

```
  --set sync-source:source \
  --set sync-destination:scim
```

2. The next set of steps define three Sync Classes. The first Sync Class is used to match user entries in the Sync Source. The second class is used to match group entries. The third class is a DEFAULT class that is used to match all other entries.

   Run the `dsconfig` command to create the first Sync Class and set the Sync Pipe Name and Sync Class name:

```
$ bin/dsconfig create-sync-class \
  --pipe-name ldap-to-scim \
  --class-name user
```

3. Use `dsconfig` to set the base DN and filter for this Sync Class. The `include-base-dn` property specifies the base DN in the source, which is `ou=people,dc=example,dc=com` by default. This Sync Class is invoked only for changes at the `ou=people` level. The `include-filter` property specifies an LDAP filter that tells the PingDataSync Server to include `inetOrgPerson` entries as user entries. The `destination-correlation-attributes` specifies LDAP attributes that allow the PingDataSync Server to find the destination resource on the SCIM server. The value of this property will vary. See Identifying a SCIM Resource at the Destination Server for details.

```
$ bin/dsconfig set-sync-class-prop \
  --pipe-name ldap-to-scim \
  --class-name user \
  --add include-base-dn:ou=people,dc=example,dc=com \
  --add "include-filter:(objectClass=inetOrgPerson)" \
  --set destination-correlation-attributes:externalId
```

4. Create the second Sync Class, which is used to match group entries.

```
$ bin/dsconfig create-sync-class \
  --pipe-name ldap-to-scim \
  --class-name group
```

5. For the second Sync Class, set the base DN and the filters to match the group entries.

```
$ bin/dsconfig set-sync-class-prop \
  --pipe-name ldap-to-scim \
  --class-name group \
  --add include-base-dn:ou=groups,dc=example,dc=com \
  --add "include-filter:(|(objectClass=groupOfEntries)\
    (objectClass=groupOfNames)(objectClass=groupOfUniqueNames)\
    (objectClass=groupOfURLs))"
```

6. For the third Sync Class, create a DEFAULT Sync Class that is used to match all other entries. To synchronize changes from only user and group entries, set `synchronize-creates`, `synchronize-modifies`, and `synchronize-delete` to `false`.

```
$ bin/dsconfig create-sync-class \
  --pipe-name ldap-to-scim \
```

```
  --class-name DEFAULT \
  --set evaluation-order-index:99999 \
  --set synchronize-creates:false \
  --set synchronize-modifies:false \
  --set synchronize-deletes:false
```

7. After all of the Sync Classes needed by the Sync Pipe are configured, set the evaluation order index for each Sync Class. Classes with a lower number are evaluated first. Run `dsconfig` to set the evaluation order index for the Sync Class. The actual number depends on the deployment.

```
$ bin/dsconfig set-sync-class-prop \
  --pipe-name ldap-to-scim \
  --class-name user \
  --set evaluation-order-index:100
```

Configure communication with the source server(s)

Configure communication between the PingDataSync Server and the LDAP source servers with the `prepare-endpoint-server` tool. If user accounts do not exist, this tool creates the appropriate user account and its privileges. Also, because the source is an PingDirectory Server, this tool enables the change log.

Note
The `prepare-endpoint-server` tool can only be used on LDAP directory servers. For the SCIM Server, manually create a sync user entry.

Run the `prepare-endpoint-server` command to setup communication between the PingDataSync Server and the source server(s). The tool will prompt for the bind DN and password to create the user account and enable the change log.

```
$ bin/prepare-endpoint-server \
  --hostname ds1.example.com \
  --port 636 \
  --useSSL \
  --trustAll \
  --syncServerBindDN "cn=Sync User,cn=Root DNs,cn=config" \
  --syncServerBindPassword "password" \
  --baseDN "dc=example,dc=com" \
  --isSource
```

Start the Sync Pipe

The `realtime-sync` tool sets a specific starting point for real-time synchronization, so that changes made before the current time are ignored.

1. Run the `realtime-sync` tool to set the startpoint for the Sync Source.

```
$ bin/realtime-sync set-startpoint \
  --end-of-changelog \
  --pipe-name ldap-to-scim
```

2. When ready to start synchronization, run the following command:

```
$ bin/realtime-sync start \
  --pipe-name ldap-to-scim \
  --no-prompt
```

Map LDAP schema to SCIM resource schema

The resources configuration file is used to define the SCIM resource schema and its mapping to LDAP schema. The default configuration of the `scim-resources.xml` file provides definitions for standard SCIM Users and Groups resources, and mappings to standard LDAP `inetOrgPerson` and `groupOfUniqueNames` object classes. It is installed with the PingDirectory Server. This file can be customized by adding extension attributes to the Users and Groups resources, or by adding new extension resources. The resources file is composed of a single `<resources>` element, containing one or more `<resource>` elements.

The default configuration maps the SCIM resource ID to the LDAP `entryUUID` attribute. In all cases, this must be changed to match the attribute that the destination SCIM service provider is using for its SCIM resource ID. For example, if the destination uses the value of the `uid` attribute, modify the `scim-resources.xml` file to change the `resourceIDMapping` as follows:

```
<resourceIDMapping ldapAttribute="uid" />
```

Ideally, this would be an attribute that exists on the source LDAP entry. If not, the PingDataSync Server can construct it using a Constructed Attribute Mapping. For example, the SCIM service provider used the first and last initials of the user, concatenated with the employee ID (given by the `eid` attribute) as the SCIM resource ID. In this case, an attribute mapping would be constructed as follows:

```
$ dsconfig create-attribute-mapping \
  --map-name MyAttrMap \
  --mapping-name scimID \
  --type constructed \
  --set 'value-pattern:{givenname:/^(.)(.*)/$1/s}{sn:/^(.)(.*)/$1/s}{eid}'
```

This creates an attribute called `scimID` on the mapped entry when processed by the Sync engine. For example, if the user's name was John Smith, with employee ID 12345, then the `scimID` would be `js12345`. Once this is done,configure the `scim-resources.xml` file as follows:

```
<resourceIDMapping ldapAttribute="scimID" />
```

This will cause it to pull out the constructed `scimID` value from the entry and use that at the SCIM resource ID when making requests to the service provider.

Note

Constructed attribute mappings support multivalued source attributes for conditional (using the `conditional-value-pattern` property) and non-conditional (using the `value-pattern` property) value patterns. Only one of the source attributes that contribute to a given value pattern can be multivalued.

For any given SCIM resource endpoint, only one `<LDAPAdd>` template can be defined, and only one `<LDAPSearch>` element can be referenced. If entries of the same object class can be located under different subtrees or base DNs of the PingDirectory Server, then a distinct SCIM resource must be defined for each unique entry location in the Directory Information Tree. If using the SCIM HTTP Servlet Extension for the PingDirectory Server, this can be implemented in many ways, such as:

- Create multiple SCIM servlets, each with a unique `resources.xml` configuration, and each running under a unique HTTP connection handler.
- Create multiple SCIM servlets, each with a unique `resources.xml` configuration, each running under a single, shared HTTP connection handler, but each with a unique context path.

LDAP attributes are allowed to contain characters that are invalid in XML (because not all valid UTF-8 characters are valid XML characters). Make sure that any attributes that contain binary data are declared using `dataType=binary` in the `scim-resources.xml` file. When using the Identity Access API, make sure that the underlying LDAP schema uses the Binary or Octet String attribute syntax for attributes that contain binary data. This instructs the server to base64-encode the data before returning it to clients.

If attributes that are not declared as binary in the schema and contain binary data (or just data that is invalid in XML), the server will check for this before returning them to the client. If the client has set the content-type to XML, then the server may choose to base64-encode any values that include invalid XML characters. When this is done, a special attribute is added to the XML element to alert the client that the value is base64-encoded. For example:

```
<scim:value base64Encoded="true">AAABPB0EBZc=</scim:value>
```

The remainder of this section describes the mapping elements available in the `scimresources.xml` file.

The <resource> element

A `resource` element has the following XML attributes:

- **schema**: a required attribute specifying the SCIM schema URN for the resource. Standard SCIM resources already have URNs assigned for them, such as `urn:scim:schemas:core:1.0`. A new URN must be obtained for custom resources using any of the standard URN assignment methods.
- **name**: a required attribute specifying the name of the resource used to access it through the SCIM REST API.
- **mapping**: a custom Java class that provides the logic for the resource mapper. This class must extend the `com.unboundid.scim.ldap.ResourceMapper` class.

A `resource` element contains the following XML elements in sequence:

- **description**: a required element describing the resource.
- **endpoint**: a required element specifying the endpoint to access the resource using the SCIM REST API.
- **LDAPSearchRef**: a mandatory element that points to an `LDAPSearch` element. The `LDAPSearch` element allows a SCIM query for the resource to be handled by an LDAP service and also specifies how the SCIM resource ID is mapped to the LDAP server.
- **LDAPAdd**: an optional element specifying information to allow a new SCIM resource to be added through an LDAP service. If the element is not provided then new resources cannot be created through the SCIM service.
- **attribute**: one or more elements specifying the SCIM attributes for the resource.

The <attribute> element

An `attribute` element has the following XML attributes:

- **schema**: a required attribute specifying the schema URN for the SCIM attribute. If omitted, the schema URN is assumed to be the same as that of the enclosing resource, so this only needs to be provided for SCIM extension attributes. Standard SCIM attributes already have URNs assigned for them, such as `urn:scim:schemas:core:1.0`. A new URN must be obtained for custom SCIM attributes using any of the standard URN assignment methods.
- **name**: a required attribute specifying the name of the SCIM attribute.
- **readOnly**: an optional attribute indicating whether the SCIM sub-attribute is not allowed to be updated by the SCIM service consumer. The default value is `false`.
- **required**: an optional attribute indicating whether the SCIM attribute is required to be present in the resource. The default value is `false`.

An `attribute` element contains the following XML elements in sequence:

- **description**: a required element describing the attribute. Then just one of the following elements:
- **simple**: specifies a simple, singular SCIM attribute.
- **complex**: specifies a complex, singular SCIM attribute.
- **simpleMultiValued**: specifies a simple, multi-valued SCIM attribute.
- **complexMultiValued**: specifies a complex, multi-valued SCIM attribute.

The <simple> element

A `simple` element has the following XML attributes:

- **dataType**: a required attribute specifying the simple data type for the SCIM attribute. The following values are permitted: `binary`, `boolean`, `dateTime`, `decimal`, `integer`, and `string`.
- **caseExact**: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `simple` element contains the following XML elements in sequence:

- **mapping**: an optional element specifying a mapping between the SCIM attribute and an LDAP attribute. If this element is omitted, the SCIM attribute has no mapping and the SCIM service ignores any values provided for the SCIM attribute.

The <complex> element

The `complex` element does not have any XML attributes. It contains the following XML element:

- **subAttribute**: one or more elements specifying the sub-attributes of the complex SCIM attribute, and an optional mapping to LDAP. The standard type, primary, and display sub-attributes do not need to be specified.

The <simpleMultiValued> element

A `simpleMultiValued` element has the following XML attributes:

- **childName**: a required attribute specifying the name of the tag that is used to encode values of the SCIM attribute in XML in the REST API protocol. For example, the tag for the standard emails SCIM attribute is email.
- **dataType**: a required attribute specifying the simple data type for the plural SCIM attribute (i.e. the data type for the value sub-attribute). The following values are permitted: `binary`, `boolean`, `dateTime`, `integer`, and `string`.
- **caseExact**: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `simpleMultiValued` element contains the following XML elements in sequence:

- **canonicalValue**: specifies the values of the type sub-attribute that is used to label each individual value, and an optional mapping to LDAP.
- **mapping**: an optional element specifying a default mapping between the SCIM attribute and an LDAP attribute.

The <complexMultiValued> element

A `complexMultiValued` element has the following XML attributes:

- **tag**: a required attribute specifying the name of the tag that is used to encode values of the SCIM attribute in XML in the REST API protocol. For example, the tag for the standard `addresses` SCIM attribute is `address`.

A `complexMultiValued` element contains the following XML elements in sequence:

- **subAttribute**: one or more elements specifying the sub-attributes of the complex SCIM attribute. The standard `type`, `primary`, and `display` sub-attributes do not need to be specified.
- **canonicalValue**: specifies the values of the type sub-attribute that is used to label each individual value, and an optional mapping to LDAP.

The <subAttribute> element

A `subAttribute` element has the following XML attributes:

- **name**: a required element specifying the name of the sub-attribute.
- **readOnly**: an optional attribute indicating whether the SCIM sub-attribute is not allowed to be updated by the SCIM service consumer. The default value is `false`.
- **required**: an optional attribute indicating whether the SCIM sub-attribute is required to be present in the SCIM attribute. The default value is `false`.
- **dataType**: a required attribute specifying the simple data type for the SCIM sub-attribute. The following values are permitted: `binary`, `boolean`, `dateTime`, `integer`, and `string`.
- **caseExact**: an optional attribute that is only applicable for string data types. It indicates whether comparisons between two string values use a case-exact match or a case-ignore match. The default value is `false`.

A `subAttribute` element contains the following XML elements in sequence:

- **description**: a required element describing the sub-attribute.
- **mapping**: an optional element specifying a mapping between the SCIM sub-attribute and an LDAP attribute. This element is not applicable within the `complexMultiValued` element.

The <canonicalValue> element

A `canonicalValue` element has the following XML attributes:

- **name**: specifies the value of the type sub-attribute. For example, work is the value for emails, phone numbers and addresses intended for business purposes.

A `canonicalValue` element contains the following XML elements in sequence:

- **subMapping**: an optional element specifying mappings for one or more of the subattributes. Any sub-attributes that have no mappings will be ignored by the mapping service.

The <mapping> element

A `mapping` element has the following XML attributes:

- **ldapAttribute**: a required element specifying the name of the LDAP attribute to which the SCIM attribute or sub-attribute map.
- **transform**: an optional element specifying a transformation to apply when mapping an attribute value from SCIM to LDAP, and LDAP to SCIM. The available transformations are described in Mapping LDAP Schema to SCIM Resource Schema.

The <subMapping> element

A `subMapping` element has the following XML attributes:

- **name**: a required element specifying the name of the sub-attribute that is mapped.
- **ldapAttribute**: a required element specifying the name of the LDAP attribute to which the SCIM sub-attribute maps.
- **transform**: an optional element specifying a transformation to apply when mapping an attribute value from SCIM to LDAP and vice-versa. The available transformations are described later. Available transformations are described in Mapping LDAP Schema to SCIM Resource Schema.

The <LDAPSearch> element

A `LDAPSearch` element has the following XML attributes:

- **baseDN**: a required element specifying the LDAP search base DN to be used when querying for the SCIM resource.
- **filter**: a required element specifying an LDAP filter that matches entries representing the SCIM resource. This filter is typically an equality filter on the LDAP object class.
- **resourceIDMapping**: an optional element specifying a mapping from the SCIM resource ID to an LDAP attribute. When the element is omitted, the resource ID maps to the LDAP entry DN.

Note
The `LDAPSearch` element can be added as a top-level element outside of any `<Resource>` elements, and then referenced within them with an `ID` attribute.

The <resourceIDMapping> element

A `resourceIDMapping` element has the following XML attributes:

- **ldapAttribute**: a required element specifying the name of the LDAP attribute to which the SCIM resource ID maps.
- **createdBy**: a required element specifying the source of the resource ID value when a new resource is created by the SCIM consumer using a POST operation. Allowable values for this element include `<scim-consumer>`, meaning that a value must be present in the initial resource content provided by the SCIM consumer, or directory, (as would be the case if the mapped LDAP attribute is `entryUUID`).

If the LDAP attribute value is not listed as destination correlation attribute, this setting is not used by the PingDataSync Server.

The following example illustrates an `LDAPSearch` element that contains a `resourceIDMapping` element:

```
<LDAPSearch id="userSearchParams">
  <baseDN>ou=people,dc=example,dc=com</baseDN>
  <filter>(objectClass=inetOrgPerson)</filter>
  <resourceIDMapping ldapAttribute="entryUUID" createdBy="directory"/>
</LDAPSearch>
```

The <LDAPAdd> element

A `LDAPAdd` element has the following XML attributes:

- **DNTemplate**: a required element specifying a template that is used to construct the DN of an entry representing a SCIM resource when it is created. The template may reference values of the entry after it has been mapped using `{ldapAttr}`, where `ldapAttr` is the name of an LDAP attribute.
- **fixedAttribute**: zero or more elements specifying fixed LDAP values to be inserted into the entry after it has been mapped from the SCIM resource.

The <fixedAttribute> element

A `fixedAttribute` element has the following XML attributes:

- **ldapAttribute**: a required attribute specifying the name of the LDAP attribute for the fixed values.
- **onConflict**: an optional attribute specifying the behavior when the LDAP entry already contains the specified LDAP attribute. The default value `merge` indicates that the fixed values should be merged with the existing values. The value `overwrite` indicates that the existing values are to be overwritten by the fixed values. The value `preserve` indicates that no changes should be made.

A `fixedAttribute` element contains the following XML element:

- **fixedValue**: one or more elements specifying the fixed LDAP values.

Identify a SCIM resource at the destination

When a SCIM Sync Destination needs to synchronize a change to a SCIM resource on the destination SCIM server, it must first fetch the destination resource. If the destination resource ID is known, the resource will be retrieved by its ID. If not, a search is performed using the mapped destination correlation attributes. Configuring this requires coordination between the Sync Class and the `scim-resources.xml` mapping file.

The `scim-resources.xml` mapping file treats the value of the `<resourceIDMapping>` element's `ldapAttribute` attribute as the SCIM ID of the source entry. If this value is also listed as a value of the Sync Class's `destination-correlation-attributes` property, then the value of this LDAP attribute is used as the SCIM ID of the destination resource.

If no value of `destination-correlation-attributes` matches the `<resourceIDMapping>` element's `ldapAttribute` attribute, the SCIM ID of the destination resource is considered unknown. In this case, the SCIM Sync Destination treats the values of `destination-correlation-attributes` as search terms, using them to construct a filter for finding the destination resource. Each value of `destination-correlation-attributes` will be mapped to a corresponding SCIM attribute name, and equality matches will be used in the resulting filter.

If the `ldapAttribute` value is not listed as a destination correlation attribute, this setting is not used by the PingDataSync Server.

The following table illustrates an `LDAPSearch` element that contains a `resourceIDMapping` element:

Identifying a SCIM Resource

Method for Retrieving SCIM Resource	Condition	Example Condition	Example Request
Retrieve resource directly	Used if a `destination-correlation-attribute` value matches the `<resourceIDMapping> ldapAttribute` value.	`destination-correlation-attribute=mail,uid;<resourceIDMapping ldapAttribute="mail" createdBy="directory"/>`	`GET scim/Users/person@example.com`
Retrieve resource using search	Used if no `destination-correlation-attribute` value matches the `<resourceIDMappin`	`destination-correlation-attribute=mail,uid;<resourceIDMapping ldapAttribute="entryUUID"createdBy="directory"/>`	`GET /scim/Users?filter=emails+eq+"person@example.com" and+userName+eq"person"`

Identifying a SCIM Resource

Method for Retrieving SCIM Resource	Condition	Example Condition	Example Request
	g> ldapAttribute value.		

The unique ID of a destination SCIM resource will most likely be unknown, and the search method will need to be used. However, not all SCIM service providers support the use of filters. Therefore, not all SCIM service providers may be usable as SCIM Sync Destinations.

Chapter 10: Manage logging, alerts, and alarms

Each PingData server supports extensive logging features to track all aspects of the PingData topology.

Topics include:

Logs and log publishers

Synchronization logs and messages

Create a new log publisher

Configure log signing

Configure log retention and rotation policies

Configure log listeners

System alarms, alerts, and gauges

Test alerts and alarms

The status tool

Sync-specific status

Monitor the PingDataSync Server

Logs and Log Publishers

PingData servers support different types of log publishers that can be used to provide the monitoring information for operations, access, debug, and error messages that occur during normal server processing. The server provides a standard set of default log files as well as mechanisms to configure custom log publishers with their own log rotation and retention policies.

Types of Log Publishers

Several types of log publishers can be used to log processing information about the server, including:

- **Audit loggers** – provide information about actions that occur within the server. Specifically, this type of log records all changes applied, detected or failed; dropped operations that were not completed; changes dropped due to being out of scope, or no changes needed for an operation. The log also shows the entries that were involved in a process.
- **Error loggers** – provide information about warnings, errors, or significant events that occur within the server.
- **Debug loggers** – provide detailed information about processing performed by the server, including any exceptions caught during processing, detailed information about data read from or written to clients, and accesses to the underlying database.
- **Access loggers** – provide information about LDAP operations processed within the server. This log only applies to operations performed in the server. This includes configuration changes, searches of monitor data, and bind operations for authenticating administrators using the command-line tools and the Administrative Console.

View the list of log publishers

View the list of log publishers on each server using the `dsconfig` tool:

```
$ bin/dsconfig list-log-publishers

Log Publisher                       : Type              : enabled
------------------------------------:-------------------:--------
Debug ACI Logger                    : debug-access      : false
Expensive Operations Access Logger  : file-based-access : false
Failed Operations Access Logger     : file-based-access : true
File-Based Access Logger            : file-based-access : true
File-Based Audit Logger             : file-based-audit  : false
File-Based Debug Logger             : file-based-debug  : false
File-Based Error Logger             : file-based-error  : true
Replication Repair Logger           : file-based-error  : true
```

Log compression

PingData servers support the ability to compress log files as they are written. Because of the inherent problems with mixing compressed and uncompressed data, compression can only be enabled when the logger is created. Compression cannot be turned on or off once the logger is configured. If the server encounters an existing log file at startup, it will rotate that file and begin a new one rather than attempting to append it to the previous file.

Compression is performed using the standard gzip algorithm. Because it can be useful to have an amount of uncompressed log data for troubleshooting, having a second logger defined that does not use compression may be desired.

Configure compression by setting the `compression-mechanism` property to have the value of `gzip` when creating a new logger. See Creating a New Log Publisher for details.

Configure log file encryption

The server supports the ability to encrypt log files as they are written. The encrypt-log configuration property controls whether encryption will be enabled for the logger. Enabling encryption causes the log file to have an .encrypted extension (and if both encryption and compression are enabled, the extension will be `.gz.encrypted`). Any change that affects the name used for the log file could prevent older files from getting properly cleaned up.

Like compression, encryption can only be enabled when the logger is created. Encryption cannot be turned on or off once the logger is configured. For any log file that is encrypted, enabling compression is also recommended to reduce the amount of data that needs to be encrypted. This will also reduce the overall size of the log file. The `encrypt-file` tool (or custom code, using the LDAP SDK's `com.unboundid.util.PassphraseEncryptedInputStream`) is used to access the encrypted data.

To enable encryption, at least one encryption settings definition must be defined in the server. Use the one created during setup, or create a new one with the `encryption-settings create` command. By default, the encryption will be performed with the server's preferred encryption settings definition. To explicitly specify which definition should be used for the encryption, the `encryption-settings-definition-id` property can be set with the ID of that definition. It is recommended that the encryption settings definition is created from a passphrase so that the file can be decrypted by providing that passphrase, even if the original encryption settings definition is no longer available. A randomly generated encryption settings definition can also be created, but the log file can only be decrypted using a server instance that has that encryption settings definition.

When using encrypted logging, a small amount of data may remain in an in-memory buffer until the log file is closed. The encryption is performed using a block cipher, and it cannot write an incomplete block of data until the file is closed. This is not an issue for any log file that is not being actively written. To examine the contents of a log file that is being actively written, use the `rotate-log` tool to force the file to be rotated before attempting to examine it.

The following commands can be used to set log file encryption:

1. Use `dsconfig` to enable encryption for a Log Publisher. In this example, the File-basedAccess Log Publisher "Encrypted Access" is created, compression is set, and rotation and retention policies are set.

```
$ bin/dsconfig create-log-publisher-prop --publisher-name "Encrypted
Access" \
  --type file-based-access \
  --set enabled:true \
  --set compression-mechanism:gzip \
  --set encryption-settings-definition-
id:332C846EF0DCD1D5187C1592E4C74CAD33FC1E5FC20B726CD301CDD2B3FFBC2B \
  --set encrypt-log:true \
  --set log-file:logs/encrypted-access \
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --set "rotation-policy:Size Limit Rotation Policy" \
   --set "retention-policy:File Count Retention Policy" \
  --set "retention-policy:Free Disk Space Retention Policy" \
   --set "retention-policy:Size Limit Retention Policy"
```

2. To decrypt and decompress the file:

```
$ bin/encrypt-file --decrypt \
  --decompress-input \
  --input-file logs/encrypted-access.20180216040332Z.gz.encrypted \
  --output-file decrypted-access
Initializing the server's encryption framework...DoneWriting decrypted
data to file '/ds/PingDirectory/decrypted-access' using akey generated
from encryption settings definition
'332c846ef0dcd1d5187c1592e4c74cad33fc1e5fc20b726cd301cdd2b3ffbc2b'Success
fully wrote 123,456,789 bytes of decrypted data
```

Synchronization logs and messages

The PingDataSync Server provides a standard set of default log files to monitor the server activity. View this set of logs in the `<server-root>/logs` directory. The following default log files are available.

PingDataSync Server Logs

Log File	Description
access	File-based Access Log that records LDAP operations processed by the PingDataSync Server. Access log records can be used to provide information about problems during operation processing and provide information about the time required to process each operation.
config-audit.log	Records information about changes made to the server configuration in a format that can be replayed using the `dsconfig` tool.
errors	File-based Error Log that provides information about warnings, errors, and significant events that are not errors but occur during server processing.
server.out	Records anything written to standard output or standard error, which includes startup messages. If garbage collection debugging is enabled, then the information will be written to

Log File	Description
	`server.out`.
server.pid	Stores the server's process ID.
server.status	Stores the timestamp, a status code, and an optional message providing additional information on the server status.
setup.log	Records messages that occur during the initial server configuration with the `setup` command.
sync	File-based Sync Log that records synchronization operations processed by the server. Specifically, the log records all changes applied, detected or failed; dropped operations that were not synchronized; changes dropped due to being out of scope, or no changes needed for synchronization.
sync-pipe-cfg.txt	Records the configuration changes used with the `bin/create-sync-pipe-config` tool. The file is placed wherever the tool is run. Typically, this is in `<server-root>` or in the `bin` directory.
tools	Holds logs for long running utilities. Current and previous copies of the log are present in the directory.
update.log	Records messages that occur during a server upgrade.

Sync log message types

The PingDataSync Server logs certain types of log messages with the sync log. Message types can be included or excluded from the logger, or added to a custom log publisher.

Sync Log Message Types

Message Type	Description
change-applied	Default summary message. Logged each time a change is applied successfully.
change-detected	Default summary message. Logged each time a change is detected.
change-failed-detailed	Default detail message. Logged when a change cannot be applied. It includes the reason for the failure and details about the change that can be used to manually repair the failure.
dropped-op-type-not-synchronized	Default summary message. Logged when a change is dropped because the operation type (for example, ADD) is not synchronized for the matching Sync Class.
dropped-out-of-scope	Default summary message. Logged when a change is dropped because it does not match any Sync Class.
no-change-needed	Default summary message. Logged each time a change is dropped because the modified source entry is already synchronized with the destination entry.
change-detected-detailed	Optional detail message. Logged each time a change is detected. It includes attribute values for added and modified entries. This information is useful for diagnosing problems, but it causes log files to grow faster, which impacts performance.
entry-mapping-details	Optional detail message. Logged each time a source entry (attributes and DN) are mapped to a destination entry. This information is useful for diagnosing problems, but it causes log files to grow faster, which impacts performance.
change-applied-detailed	Optional detail message. Logged each time a change is applied. It includes attribute values for added and modified entries. This information is useful for diagnosing

Message Type	Description
	problems, but it causes log files to grow faster, which impacts performance.
change-failed	Optional summary message. Logged when a change cannot be applied. It includes the reason for the failure but not enough information to manually repair the failure.
intermediate-failure	Optional summary message. Logged each time an attempt to apply a change fails. Note that a subsequent retry of applying the change might succeed.

Create a new log publisher

PingData servers provide customization options to create log publishers with the `dsconfig` command.

After creating a new log publisher, configure the log retention and rotation policies. For more information, see Configure log rotation and Configure log retention.

1. Use the `dsconfig` command to create and configure the new log publisher. (If using `dsconfig` in interactive mode, log publishers are created and manged under the Log Publisher menu.) The following example shows how to create a logger that only logs disconnect operations.

   ```
   $ bin/dsconfig create-log-publisher \
     --type file-based-access --publisher-name "Disconnect Logger" \
     --set enabled:true \
     --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
     --set "rotation-policy:Size Limit Rotation Policy" \
     --set "retention-policy:File Count Retention Policy" \
     --set log-connects:false \
     --set log-requests:false --set log-results:false \
     --set log-file:logs/disconnect.log
   ```

   To configure compression on the logger, add the following option to the previous command:

   ```
   --set compression-mechanism: gzip
   ```

   Compression cannot be disabled or turned off once configured for the logger. Determine logging requirements before configuring this option.

2. View log publishers with the following command:

   ```
   $ bin/dsconfig list-log-publishers
   ```

Configuring log signing

PingData servers support the ability to cryptographically sign a log to ensure that it has not been modified. For example, financial institutions require tamper-proof audit logs files to ensure that transactions can be properly validated and ensure that they have not been modified by a third-party entity or internally by an unauthorized person.

When enabling signing for a logger that already exists, the first log file will not be completely verifiable because it still contains unsigned content from before signing was enabled. Only log files whose entire content was written with signing enabled will be considered completely valid. For the same reason, if a log file is still open for writing, then signature validation will not indicate that the log is completely valid because the log will not include the necessary "end signed content" indicator at the end of the file.

To validate log file signatures, use the `validate-file-signature` tool provided in the `bin` directory of the server (or the `bat` directory on Windows systems). Once this property is enabled, disable and then re-enable the log publisher for the changes to take effect. Perform the following steps to configure log signing:

1. Use `dsconfig` to enable log signing for a Log Publisher. In this example, set the `sign-log` property on the File-based Audit Log Publisher.

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Audit Logger" \
  --set sign-log:true
```

2. Disable and then re-enable the Log Publisher for the change to take effect.

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Audit Logger" \
  --set enabled:false
```

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Audit Logger" \
  --set enabled:true
```

3. To validate a signed file, use the `validate-file-signature` tool to check if a signed file has been altered.

```
$ bin/validate-file-signature --file logs/audit
```

```
All signature information in file 'logs/audit' is valid
```

If any validations errors occur, a message displays that is similar to this:

```
One or more signature validation errors were encountered while validating
the contents of file 'logs/audit':
* The end of the input stream was encountered without encountering the end
of an active signature block. The contents of this signed block cannot be
trusted because the signature cannot be verified
```

Configure log retention and log rotation policies

PingData servers enable configuring log rotation and log retention policies.

Log Retention – When any retention limit is reached, the server removes the oldest archived log prior to creating a new log. Log retention is only effective if a log rotation policy is in place. A new log publisher must have at least one log retention policy configured. The following policies are available:

- **File Count Retention Policy** – Sets the number of log files for the server to retain. The default file count is 10 logs. If the file count is set to 1, the log will continue to grow indefinitely without being rotated.
- **Free Disk Space Retention Policy** – Sets the minimum amount of free disk space. The default free disk space is 500 MB.
- **Size Limit Retention Policy** – Sets the maximum size of the combined archived logs. The default size limit is 500 MB.
- **Custom Retention Policy** – Create a new retention policy that meets the server's requirements. This will require developing custom code to implement the desired log retention policy.
- **Never Delete Retention Policy** – Used in a rare event that does not require log deletion.

Log Rotation – When a rotation limit is reached, the server rotates the current log and starts a new log. A new log publisher must have at least one log rotation policy configured. The following policies are available:

- **Time Limit Rotation Policy** – Rotates the log based on the length of time since the last rotation. Default implementations are provided for rotation every 24 hours and every seven days.
- **Fixed Time Rotation Policy** – Rotates the logs every day at a specified time (based on 24-hour). The default time is 2359.
- **Size Limit Rotation Policy** – Rotates the logs when the file reaches the maximum size. The default size limit is 100 MB.
- **Never Rotate Policy** – Used in a rare event that does not require log rotation.

Configure the log rotation policy

Use `dsconfig` to modify the log rotation policy for the access logger:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --remove "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --add "rotation-policy:7 Days Time Limit Rotation Policy"
```

Configure the log retention policy

Use `dsconfig` to modify the log retention policy for the access logger:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

Configure log listeners

The server provides two log file rotation listeners: the copy log file rotation listener and the summarize log file rotation listener, which can be enabled with a log publisher. Log file rotation listeners allow the server to perform a task on a log file as soon as it has been rotated out of service. Custom log file listeners can be created with the Server SDK.

The copy log file rotation listener can be used to compress and copy a recently-rotated log file to an alternate location for long-term storage. The original rotated log file will be subject to deletion by a log file retention policy, but the copy will not be automatically removed.

Use the following command to create a new copy log file rotation listener:

```
$ dsconfig create-log-file-rotation-listener \
  --listener-name "Copy on Rotate" \
  --type copy \
  --set enabled:true \
  --set copy-to-directory:/path/to/archive/directory \
  --set compress-on-copy:true</screen>
```

The path specified by the `copy-to-directory` property must exist, and the filesystem containing that directory must have enough space to hold all of the log files that will be written there. The server will automatically monitor free disk space on the target filesystem and will generate administrative alerts if the amount of free space gets too low.

The summarize log file rotation listener invokes the `summarize-access-log` tool on a recently-rotated log file and writes its output to a file in a specified location.

This provides information about the number and types of operations processed by the server, processing rates and response times, and other useful metrics. Use this with access loggers that log in a format that is compatible with the `summarize-access-log` tool, including the `file-based-access` and `operation-timing-access` logger types. Use the following command to create a new summarize log file rotation listener:

```
$ dsconfig create-log-file-rotation-listener \
  --listener-name "Summarize on Rotate" \
  --type summarize \
  --set enabled:true \
  --set output-directory:/path/to/summary/directory
```

The summary output files have the same name as the rotated log file, with an extension of `.summary`. If the `output-directory` property is specified, the summary files are written to that directory. If not specified, files are placed in the directory in which the log files are written.

As with the copy log file rotation listener, summary files are not automatically be deleted. Though files are generally small in comparison to the log files themselves, make sure that there is enough space available in the specified storage directory. The server automatically monitors free disk space on the filesystem to which the summary files are written.

System alarms, alerts, and gauges

An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server available' or ‘server unavailable’ (Indicator Gauge). Gauges generate alarms, when the gauge’s severity changes due to changes in the monitored value. Like alerts, alarms have severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a `Condition` property, and may have a Specific Problem or Resource property. If surfaced through SNMP, a `Probable Cause` property and `Alarm Type` property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

There are two alert types supported by the server - standard and alarm-specific. The server constantly monitors for conditions that may need attention by administrators, such as low disk space. For this condition, the standard alert is `low-disk-space-warning`, and the alarm-specific alert is `alarm-warning`. The server can be configured to generate alarm-specific alerts instead of, or in addition to, standard alerts. By default, standard alerts are generated for conditions internally monitored by the server. However, gauges can only generate alarm-alerts.

The server installs a set of gauges that are specific to the product and that can be cloned or configured through the `dsconfig` tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

PingData servers are compliant with the International Telecommunication Union CCITT Recommendation X.733 (1992) standard for generating and clearing alarms. If configured, entering or exiting an alarm state can result in one or more alerts. An alarm state is exited when the condition no longer applies. An `alarm_cleared` alert type is generated by the system when an alarm's severity changes from a non-normal severity to any other severity. An `alarm_cleared` alert will correlate to a previous alarm when Condition and Resource property are the same. The Alarm Manager, which governs the actions performed when an alarm state is entered, is configurable through the `dsconfig` tool and Administrative Console.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool. As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

Alert handlers

Alert notifications can be sent to administrators when significant problems or events occur during processing, such as problems during server startup or shutdown. The server provides a number of alert handler implementations configured with the `dsconfig` tool, including:

- **Error Log Alert Handler** – Sends administrative alerts to the configured server error logger(s).
- **JMX Alert Handler** – Sends administrative alerts to clients using the Java Management Extensions (JMX) protocol. The server uses JMX for monitoring entries and requires that the JMX connection handler be enabled.
- **SNMP Alert Handler** – Sends administrative alerts to clients using the Simple Network Monitoring Protocol (SNMP). The server must have an SNMP agent capable of communicating via SNMP 2c.

If needed, the Server SDK can be used to implement additional, third-party alert handlers.

Configure alert handlers

Alert handlers can be configured with the `dsconfig` tool. PingData servers support JMX, SMTP, and SNMP. Use the `--help` option for a list of configuration options. The following is a sample command to create and enable an SMTP Alert handler from the command line:

```
$ bin/dsconfig create-alert-handler \
  --handler-name "SMTP Alert Handler" \
  --type smtp \
  --set enabled:true \
  --set "sender-address:alerts@example.com" \
  --set "recipient-address:administrators@example.com" \
  --set "message-subject:Directory Admin Alert \%\%alert-type\%\%" \
  --set "message-body:Administrative alert:\\n\%\%alert-message\%\%"
```

Test alerts and alarms

After alarms and alert handlers are configured, verify that the server takes the appropriate action when an alarm state changes by manually increasing the severity of a gauge. Alarms and alerts can be verified with the `status` tool.

1. Configure a gauge with `dsconfig` and set the `override-severity` property to critical. The following example uses the CPU Usage (Percent) gauge.

   ```
   $ dsconfig set-gauge-prop \
     --gauge-name "CPU Usage (Percent)" \
     --set override-severity:critical
   ```

2. Run the `status` tool to verify that an alarm was generated with corresponding alerts. The `status` tool provides a summary of the server's current state with key metrics and a list of recent alerts and alarms. The sample output has been shortened to show just the alarms and alerts information.

   ```
   $ bin/status
   ```

   ```
                           --- Administrative Alerts ---
   Severity : Time           : Message
   ---------:---------------:-------------------------------------------
   ```

```
-------
Error    : 11/Aug/2016    : Alarm [CPU Usage (Percent). Gauge CPU Usage
(Percent)
         : 15:41:00 -0500 : for Host System has
         :                : a current value of '18.583333333333332'.
         :                : The severity is currently OVERRIDDEN in the
         :                : Gauge's configuration to 'CRITICAL'.
         :                : The actual severity is: The severity is
         :                : currently 'NORMAL', having assumed this
severity
         :                : Mon Aug 11 15:41:00 CDT 2016. If CPU use is
high,
         :                : check the server's current workload and make
any
         :                : needed adjustments. Reducing the load on the
system
         :                : will lead to better response times.
         :                : Resource='Host System']
         :                : raised with critical severity
Shown are alerts of severity [Info,Warning,Error,Fatal] from the past 48
hours
Use the --maxAlerts and/or --alertSeverity options to filter this list
```

```
                                  --- Alarms ---
Severity : Severity   : Condition : Resource    : Details
         : Start Time :           :              :
---------:------------:-----------:-------------:-------------------------
--
Critical : 11/Aug/2016: CPU Usage : Host System : Gauge CPU Usage
(Percent) for
         : 15:41:00   : (Percent) : : Host System
         :  -0500     :           :              : has a current value of
         :            :           :              : '18.785714285714285'.
         :            :           :              : The severity is
currently
         :            :           :              : 'CRITICAL', having
assumed
         :            :           :              : this severity Mon Aug 11
         :            :           :              : 15:49:00 CDT 2016. If
CPU use
         :            :           :              : is high, check the
server's
         :            :           :              : current workload and
make any
         :            :           :              : needed adjustments.
Reducing
         :            :           :              : the load on the system
will
         :            :           :              : lead to better response
times
```

```
Shown are alarms of severity [Warning,Minor,Major,Critical
Use the --alarmSeverity option to filter this list
```

Use the status tool

PingData servers provides the `status` tool, which outputs the health of the server. The `status` tool polls the current health of the server and displays summary information about the number of operations processed in the network. The tool provides the following information:

Status Tool Sections

Status Section	Description
Server Status	Displays the server start time, operation status, number of connections (open, max, and total).
Server Details	Displays the server details including host name, administrative users, install path, server version, and Java version.
Connection Handlers	Displays the state of the connection handlers including address, port, protocol and current state.
Admin Alerts	Displays the 15 administrative alerts that were generated over the last 48-hour period. Limit the number of displayed alerts using the `--maxAlerts` option. For example, status `--maxAlerts 0` suppresses all alerts.

Synchronization-specific status

The `status` tool displays the following information for the PingDataSync Server.

PingDataSync Server Status Information

Status Section	Description
Sync Topology	Displays information about the connected Sync topology and any standby PingDataSync Server instances.
Summary for Sync Pipe	Displays the health status for each Sync Pipe configured on the topology. Status for each Sync Pipe includes the following: • Started – Indicates whether the Sync Pipe has started. • Current Ops Per Second – Lists the current throughput rate in operations per second. • Percent Busy – Lists the number of current operations currently divided by the number of worker threads. • Changes Detected – Lists the total number of changes detected. • Ops Completed Total – Lists the total number of changes detected and completed. • Num Ops In Flight – Lists the number of operations that are in flight. • Num Ops In Queue – Lists the number of operations that are on the input queue waiting to be synchronized. • Source Unretrieved Changes – Lists how many outstanding changes are still in the source change log that have not yet been retrieved by the PingDataSync Server. If this

Status Section	Description
	is greater than zero, it indicates a backlog, because the internal queue is too full to include these changes. • Failed Op Attempts – Lists the number of failed operation attempts. • Poll For Source Changes Count – Lists the number of times that the source has been polled for changes.
Operations Completed for the Sync Pipe	Displays the completed operation statistics for the sync pipe, including the following: • Success – Lists the number of changes that completed successfully. • Out Of Scope – Lists the number of changes that were included in the Sync Pipe, but were dropped because they did not match criteria in a Sync Class. • Op Type Not Synced – Lists the number of changes that completed because the operation type is not synchronized. • No Change Needed – Lists the number of changes that completed because no change was needed. • Entry Already Exists – Lists the number of changes that completed unsuccessfully because the entry already existed for a Create operation. • No Match Found – Lists the number of changes that completed unsuccessfully because no match for an operation (such as Modify) was found. • Multiple Matches Found – Lists the number of changes that completed unsuccessfully because multiple matches for a source entry were found at the destination. • Failed During Mapping – Lists the number of changes that completed unsuccessfully because there was a failure during attribute or DN mapping. • Failed At Resource – Lists the number of changes that completed unsuccessfully because they failed at the source. • Unexpected Exception – Lists the number of changes that completed unsuccessfully because there was an unexpected exception during processing. • Total – Lists the number of operations completed.
Sync Pipe Source Stats	Displays the source statistics for the external server, including the following: • Is Connected – Indicates whether the Sync Source is connected or not. • Connected Server – Indicates the hostname and port number of the connected server. • Successful Connect Attempts – Indicates the number of successful connection attempts. • Failed Connect Attempts – Indicates the number of failed connection attempts. • Forced Disconnects – Indicates the number of forced disconnects. • Root DSE Polls – Indicates the number of polling attempts of the root DSE. • Unretrieved Changes – Indicates the number of unretrieved changes.

Status Section	Description
	• Entries Fetched – Indicates the number of entries fetched from the source. • Failed To Decode Changelog Entry – Indicates the operations that failed to decode changelog entries. • Ops Excluded By Modifiers Name – Indicates the number of operations excluded by modifier's name. • Num Backtrack Batches Retrieved – Indicates the number of backtrack batches retrieved.
Sync Pipe Destination Stats	Displays the destination statistics for the external server, including the following: • Is Connected – Indicates whether the Sync Source is connected or not. • Connected Server – Indicates the connection URL of the connected server. • Successful Connect Attempts – Indicates the number of successful connection attempts. • Failed Connect Attempts – Indicates the number of failed connection attempts. • Forced Disconnects – Indicates the number of forced disconnects. • Entries Fetched – Indicates the number of entries fetched. • Entries Created – Indicates the number of entries created. • Entries Modified – Indicates the number of entries modified. • Entries Deleted – Indicates the number of entries deleted.

Monitor the PingDataSync Server

The PingDataSync Server exposes its monitoring information under the `cn=monitor` entry. Various tools can be used to surface the server's information including the PingDataMetrics Server, the Administrative Console, JConsole, LDAP command-line tools, or SNMP. The following information is collected for the PingDataSync Server. To configure the PingData PingDataMetrics Server to display PingDataSync Server data, see the *Ping IdentityPingDataMetrics Server Administration Guide*.

PingDataSync Server Monitoring Component

Component	Description
Active Operations	Provides information about the operations currently being processed by the server including the number of operations, information about the operation, and the number of active persistent searches.
Backend	Provides general information about the state of the server backend, including the backend ID, base DN(s), entry counts, entry count for the `cn=admin` data, writability mode, and whether it is a private backend. The following backend monitors are provided: • adminRoot

Component	Description
	• ads-truststore • alerts • backup • config • monitor • schema • tasks • userRoot
Berkeley DB JE Environment	Provides information about the state of the Oracle Berkeley DB Java Edition database used by the PingDataSync Server backend.
Client Connections	Provides information about all client connections to the server.
Disk Space Usage	Provides information about the disk space available to various components of the server. The disk space usage monitor evaluates the free space at locations registered through the `DiskSpaceConsumer` interface. Disk space monitoring excludes disk locations that do not have server components registered. However, other disk locations may still impact server performance, such as the operating system disk, if it becomes full. When relevant to the server, these locations include the server root, the location of the `config` directory, the location of every log file, all JE backend directories, the location of the changelog, the location of the replication environment database, and the location of any server extension that registers itself with the `DiskSpaceConsumer` interface.
Connection Handler	Provides information about the available connection handlers on the server, which includes the LDAP and LDIF connection handlers. These handlers are used to accept client connections.
General	Provides general information about the server, including product name and server version.
JVM Stack Trace	Provides a stack trace of all threads processing within the JVM.
LDAP Connection Handler Statistics	Provides statistics about the interaction that the associated LDAP connection handler has had with its clients, including the number of connections established and closed, bytes read and written, LDAP messages, and operations handled.
Processing Time Histogram	Categorizes operation processing times into a number of user-defined buckets of information, including the total number of operations processed, overall average response time, and number of processing times between `0ms` and `1ms`.
System Information	Provides general information about the system and the JVM on which the server is running, including host name, operating system, JVM architecture, Java home, and Java version.
Version	Provides information about the product version, including build ID and revision number.
Work Queue	Provides information about the state of the PingDataSync Server work queue, which holds requests until they can be processed by a worker thread. The work queue configuration has a `monitor-queue-time` property set to `true` by default. This logs messages for new operations with a `qtime` attribute included in

Component	Description
	the log messages. Its value is expressed in milliseconds and represents the length of time that operations are held in the work queue. A dedicated thread pool can be used for processing administrative operations. This thread pool enables diagnosis and corrective action if all other worker threads are processing operations. To request that operations be processed using the administrative thread pool, the requester must have the `use-admin-session` privilege (included for root users). By default, eight threads are available for this purpose. This can be changed with the `num-administrative-session-worker-threads` property in the work queue configuration.

Chapter 11: Troubleshooting

There are several ways to troubleshoot issues with the PingDataSync Server.

Topics include:

Synchronization troubleshooting

The majority of synchronization problems involve the connection state of the external servers and the synchronization of the data between the two endpoints. Make sure the PingDataSync Server can properly fail over to another endpoint or server instance if the connection fails on the highest priority external server.

Another factor in troubleshooting synchronization is determining if the DN and attribute mappings were properly configured and if the information is properly being synchronized across the network. Typical scenarios include checking for any entry failures and mapping issues.

> **Note**
> Use the `resync` tool to validate Sync Classes and data mappings from one endpoint to another. The tool provides a `dry-run` option that verifies data operations without actually affecting the data.

The following log files are specific to the PingDataSync Server, and contain details about the synchronization processes:

- **Sync Log** – provides information about the synchronization operations that occur within the server. Specifically, the Sync Log records all changes applied, detected or failed; dropped operations that were not synchronized; changes dropped due to being out of scope, or no changes needed for synchronization. The log also shows the entries that were involved in the synchronization process.
- **Sync Failed Operations Log** – provides a list of synchronization operations that have failed.
- **Resync Log** – provides summaries or details of synchronized entries and any missing entries in the Sync Destination.
- **Resync Error Log** – provides error information for `resync` operations.

Management tools

Each PingData server provides command-line tools to manage, monitor, and diagnose server operations. Each tool provides a description of the subcommands, arguments, and usage examples needed to run the tool.

> **Note**
> For detailed information and examples of the command-line tools, see the Configuration Reference Guide in the `<server-root>/docs` directory, or linked from the Administrative Console.

To view detailed argument options and examples, use `--help` with the each tool:

```
$ bin/dsconfig --help
```

For those utilities that support additional subcommands (such as `dsconfig`), list the subcommands with the following:

```
$ bin/dsconfig --help-subcommands
```

View more detailed subcommand information by using `--help` with the specific subcommand:

```
$ bin/dsconfig list-log-publishers --help
```

Troubleshooting tools

PingData provides utilities to troubleshoot the state of each server and to determine the cause of any issues. The following tools are available for diagnosing any problems and are located in the `<server-root>/bin` directory, or the `<server-root>/bat` directory on Windows systems:

Troubleshooting Tools

Tool	Description
`status`	Provides a high-level view of the current operational state of the server and displays any recent alerts that have occurred in past 24 hours.
`ldap-diff`	Used to compare one or more entries across two server endpoints to determine data issues.
`ldapsearch`	Retrieves the full entries from two different servers to determine the exact content of an entry from each server.
`logs`	The logs directory provides important logs that should be used to troubleshoot or monitor any issue with the server. Logs include server-specific operations and the following general logs: • Error Log – Provides information about warnings, errors, or significant events that occur within the server. • Debug Log – Provides detailed information, if enabled, about processing performed by the server, including any exceptions caught during processing, detailed information about data read from or written to clients, and accesses to the underlying database. • Access loggers – Provide information about LDAP operations processed within the server. This log only applies to operations performed in the server. This includes configuration changes, searches of monitor data, and bind operations for authenticating administrators using the command-line tools and the Administrative Console.
`collect-support-data`	Used to aggregate the results of various support tools data for the Ping IdentitySupport team to diagnose. For more information, see Using the collect-support-data Tool.
`config-diff`	Generate a summary of the configuration changes in a local or remote server instance. The tool can be used to compare configuration settings when troubleshooting issues, or when verifying configuration settings on new servers.

Use the status tool

PingData servers provides the `status` tool, which outputs the health of the server. The `status` tool polls the current health of the server and displays summary information about the number of operations processed in the network. The tool provides the following information:

Status Tool Sections

Status Section	Description
Server Status	Displays the server start time, operation status, number of connections (open, max, and total).
Server Details	Displays the server details including host name, administrative users, install path, server version, and Java version.
Connection Handlers	Displays the state of the connection handlers including address, port, protocol and current state.
Admin Alerts	Displays the 15 administrative alerts that were generated over the last 48-hour period. Limit the number of displayed alerts using the `--maxAlerts` option. For example, status `--maxAlerts 0` suppresses all alerts.

Use the collect-support-data tool

PingData servers provide information about their current state and any problems encountered. If a problem occurs, run the `collect-support-data` tool in the `/bin` directory. The tool aggregates all relevant support files into a zip file that can be sent to a support provider for analysis. The tool also runs data collector utilities, such as `jps`, `jstack`, and `jstat` plus other diagnostic tools for the operating system.

The tool may only archive portions of certain log files to conserve space, so that the resulting support archive does not exceed the typical size limits associated with e-mail attachments.

The data collected by the `collect-support-data` tool may vary between systems. The data collected includes the configuration directory, summaries and snippets from the `logs` directory, an LDIF of the monitor and RootDSE entries, and a list of all files in the server root.

Perform the following steps to run this tool:

1. Navigate to the server root directory.
2. Run the `collect-support-data` tool. Include the host, port number, bind DN, and bind password.

```
$ bin/collect-support-data \
  --hostname 100.0.0.1 --port 389 \
  --bindDN "cn=Directory Manager"
  --bindPassword secret \
  --serverRoot /opt/PingData<server> \
  --pid 1234
```

3. Email the zip file to a support provider.

Use the Sync log

The Sync log, located in the logs directory (`<server-root>/logs/sync`), provides useful troubleshooting information on the type of operation that was processed or completed. Most log entries provide the following common elements in their messages:

Sync Log Elements

Sync Log Element	Description
category	Indicates the type of operation, which will always by SYNC.
severity	Indicates the severity type of the message: INFORMATION, MILD_WARNING, SEVERE_WARNING, MILD_ERROR, SEVERE_ERROR, FATAL_ERROR, DEBUG, or NOTICE.
msgID	Specifies the unique ID number assigned to the message.
op	Specifies the operation number specific to the PingDataSync Server.
changeNumber	Specifies the change number from the source server assigned to the modification.
replicationCSN	Specifies the replication change sequence number from the source server.
replicaID	Specifies the replica ID from the source server if there are multiple backend databases.
pipe	Specifies the Sync Pipe that was used for this operation.
msg	Displays the result of this operation.

Sync log example 1

The following example displays an informational message that a modification to an entry was detected on the source server.

```
$ tail -f logs/sync

[17/May/2015:15:46:19 -0500] category=SYNC severity=INFORMATION
msgID=1893728293
op=14 changeNumber=15 replicationCSN=00000128A7E3C7D31E960000000F
replicaID=7830
pipe="DS1 to DS2" msg="Detected MODIFY of uid=user.993,ou=People,dc=example,
dc=com at ldap://server1.example.com:1389"
```

Sync log example 2

The next example shows a successful synchronization operation that resulted from a MODIFY operation on the source server and synchronized to the destination server.

```
[18/May/2015:13:54:04 -0500] category=SYNC severity=INFORMATION
msgID=1893728306 op=701
changeNumber=514663 replicationCSN=00000128ACC249A31E960007DA67 replicaID=7830
pipe="DS1 to DS2" class="DEFAULT" msg="Synchronized MODIFY of
uid=user.698,ou=People,
dc=example,dc=com at ldap://server1.example.com:1389 by modifying entry
uid=user.698,
ou=People,dc=example,dc=com at ldap://server3.example.com:3389"
```

Sync log example 3

The next example shows a failed synchronization operation on a MODIFY operation from the source server that could not be synchronized on the destination server. The log displays the LDIF-formatted modification that failed, which came from a schema violation that resulted from an incorrect attribute mapping (`telephoneNumber -> telephone`) from the source to destination server.

```
[18/May/2015:11:29:49 -0500] category=SYNC severity=SEVERE_WARNING
msgID=1893859389
op=71831 changeNumber=485590 replicationCSN=00000128AC3DE8D51E96000768D6
replicaID=7830 pipe="DS1 to DS2" class="DEFAULT" msg="Detected MODIFY of
uid=user.941,ou=People,dc=example,dc=com at ldap://server1.example.com:1389,
but
failed to apply this change because: Failed to modify entry uid=user.941,
ou=People,dc=example,dc=com on destination 'server3.example.com:3389'.
Cause: LDAPException(resultCode=65(object class violation), errorMessage='
Entry uid=user.941,ou=People,dc=example,dc=com cannot be modified because the
resulting entry would have violated the server schema: Entry
uid=user.941,ou=People,
dc=example,dc=com violates the Directory Server schema configuration because
it
includes attribute telephone which is not allowed by any of theobjectclasses
defined in that entry') (id=1893859386
ResourceOperationFailedException.java:125
Build revision=6226). Details: Source change detail:

dn: uid=user.941,ou=People,dc=example,dc=com
changetype: modify
replace: telephoneNumber
telephoneNumber: 027167170433915
-
replace: modifiersName
modifiersName: cn=Directory Manager,cn=Root DNs,cn=config
-
replace: modifyTimestamp
modifyTimestamp: 20131010020345.546Z
Equivalent destination changes:
dn: uid=user.941,ou=People,dc=example,dc=com
changetype: modify
replace: telephone
telephone: 818002279103216
Full source entry:
dn: uid=user.941,ou=People,dc=example,dc=com
objectClass: person
... (more output)
Mapped destination entry:
dn: uid=user.941,ou=People,dc=example,dc=com
telephone: 818002279103216
objectClass: person
```

```
objectClass: inetOrgPerson
... (more output) ...
```

Troubleshoot synchronization failures

While many PingDataSync Server issues are deployment-related and are directly affected by the hardware, software, and network structure used in the synchronization topology, most failures usually fall into one of the following categories:

- **Entry Already Exists** – When an add operation was attempted on the destination server, an entry with the same DN already exists.
- **No Match Found** – A match was not found at the destination based on the current Sync Classes and correlation rules (DN and attribute mapping). When this value has a high count, correlation rule problems are likely.
- **Failure at Resource** – Indicates that some other error happened during the synchronization process that does not fall into the first two categories. Typically, these errors are communication problems with a source or destination server.

Statistics for these and other types of errors are kept in the `cn=monitor` branch and can be viewed directly using the `status` command.

Troubleshoot "Entry Already Exists" failures

If there is a count for the `Entry Already Exists` statistic using the `status` tool, verify the problem in the sync log. For example, the `status` tool displays the following information:

```
           --- Ops Completed for 'DS1 to DS2' Sync Pipe ---
Op Result              : Count
-----------------------:------
Success                : 0
Out Of Scope           : 0
Op Type Not Synced     : 0
No Change Needed       : 0
Entry Already Exists   : 1
No Match Found         : 1
Multiple Matches Found : 0
Failed During Mapping  : 0
Failed At Resource     : 0
Unexpected Exception   : 0
Total                  : 2
```

Verify the change by viewing the `<server-root>/logs/sync` file to see the specific operation, which could be due to someone manually adding the entry on the target server:

```
op=2 changeNumber=529277 replicationCSN=00000128AD0D9BA01E960008137D
replicaID=7830
pipe="DS1 to DS2" class="DEFAULT" msg="Detected ADD of
uid=user.1001,ou=People,
dc=example,dc=com at ldap://server1.example.com:1389, but cannot create this
```

```
entry
at the destination because an equivalent entry already exists at
ldap://server3.
example.com:3389. Details: Search using [search-criteria dn:
uid=user.1001,ou=People,
dc=example,dc=com attrsToGet: [*, dn]] returned results;
[uid=user.1001,ou=People,
dc=example,dc=com]. "
```

Perform the following steps to troubleshoot this type of problem:

1. Assuming that a possible DN mapping is ill-formed, first run the `ldap-diff` utility to compare the entries on the source and destination servers. Then look at the `ldap-diff` results with the mapping rules to determine why the original search did not find a match.

   ```
   $ bin/ldap-diff \
     --outputLDIF config-difference.ldif \
     --baseDN "dc=example,dc=com" \
     --sourceHost server1.example.com \
     --targetHost server2.example.com \
     --sourcePort 1389 \
     --targetPort 3389 \
     --sourceBindDN "cn=Directory Manager" \
     --sourceBindPassword password \
     --searchFilter "(uid=1234)"
   ```

2. Review the destination server access logs to verify the search and filters used to find the entry. Typically, the key correlation attributes are not synchronized.
3. If the mapping rule attributes are not synchronized, review the Sync Classes and mapping rules, and use the information from the `ldap-diff` results to determine why a specific attribute may not be getting updated. Some questions to answer are as follows:
   - Is there more than one Sync Class that the operation could match?
   - If using an `include-base-dn` or `include-filter` in the mapping rules, does this exclude this operation by mistake?
   - If using an attribute map, are the mappings correct? Usually, the cause of this error is in the destination mapping attribute settings. For example, if a set of correlation attributes is defined as: `dn`, `mobile`, `accountNumber`, and the `accountNumber` changes for some reason, future operations on this entry will fail. To resolve this, you either remove `accountNumber` from the rule, or add a second rule as: `dn`, `mobile`. The second rule is used only if the search using the first set of attributes fails. In this case, the entry is found and the `accountNumber` information is updated.
4. If deletes are being synchronized, check to see if there was a previous delete of this entry that was not synchronized properly. In some cases, simpler logic should be used for deletes due to the available attributes in the change logs. This scenario could cause

an entry to not be deleted for some reason, which would cause an issue when a new entry with the same DN is added later. Use this information for mapping rules to see why the original search did not find a match.

5. Look at the destination directory server access logs to verify the search and filters it used to find the entry. Typically, the key attribute mappings are not synchronized.

Troubleshoot "No Match Found" failures

If there is a count for the No Match Found statistic using the `status` tool, verify the problem in the sync log. For example, if the `status` tool displays the following:

```
              --- Ops Completed for 'DS1 to DS2' Sync Pipe ---
Op Result              : Count
-----------------------:------
Success                : 0
Out Of Scope           : 0
Op Type Not Synced     : 0
No Change Needed       : 0
Entry Already Exists   : 1
No Match Found         : 1
Multiple Matches Found : 0
Failed During Mapping  : 0
Failed At Resource     : 0
Unexpected Exception   : 0
Total                  : 2
```

Verify the change in the `<server-root>/logs/sync` file to see the specific operation:

```
[12/May/2016:10:30:45 -0500] category=SYNC severity=MILD_WARNING
msgID=1893793952
op=4159648 changeNumber=6648922 replicationCSN=4beadaf4002f21150000
replicaID=8469-
ou=test,dc=example,dc=com pipe="DS1 to DS2" class="Others" msg="Detected
DELETE of
'uid=1234,ou=test,dc=example,dc=com' at ldap://server1.example.com:389, but
cannot
DELETE this entry at the destination because no matching entries were found at
ldap://
server2.example.com:389. Details: Search using [search-criteria dn:
uid=1234,ou=test,dc=alu,dc=com filter: (nsUniqueId=3a324c60-5ddb11df-80ffe681-
717b93af) attrsToGet: [*, accountNumber, dn, entryuuid, mobile, nsUniqueId,
object-
Class]] returned no results."
```

Perform the following steps to fix the issue:

1. Test the search using the filter in the error message, if displayed. For example, if the sync log specifies `filter: (nsUniqueId=3a324c60-5ddb11df-80ffe681-717b93af)`, use the `ldapsearch` tool to test the filter. If it is successful, is there anything in the

attribute mappings that would exclude this from working properly?

2. Test the search using the full DN as the base. For example, use `ldapsearch` with the full DN `(uid=1234,ou=People,dc=example,dc=com)`. If it is successful, does the entry contain the attribute used in the mapping rule?
3. If the attribute is not in the entry, determine if there is a reason why this value was not synchronized. Look at the attribute mappings and the filters used in the Sync Classes.

Troubleshoot "Failed at Resource" failures

If there is a count for the "Failed at Resource" statistic using the `status` tool, verify the problem in the sync log. For example, if the `status` tool displays the following information:

```
             --- Ops Completed for 'DS1 to DS2' Sync Pipe ---
Op Result              : Count
-----------------------:------
Success                : 0
Out Of Scope           : 0
Op Type Not Synced     : 0
No Change Needed       : 0
Entry Already Exists   : 0
No Match Found         : 0
Multiple Matches Found : 0
Failed During Mapping  : 0
Failed At Resource     : 1
Unexpected Exception   : 0
Total                  : 1
```

This will register after a change is detected at the source in any of the following cases:

- If the fetch of the full source entry fails. The entry exists but there is a connection failure, server down, timeout, or something similar.
- If the fetch of the destination entry fails or if the modification to the destination fails for an exceptional reason (but not for "Entry Already Exists," "Multiple Matches Found," or "No Match Found" issues).

Verify the change by viewing the `<server-root>/logs/sync` file to see the specific operation. If any of the following result codes are listed , the server is experiencing timeout errors:

- `resultCode=timeout: errorMessage=A client-side timeout was encountered while waiting 60000ms for a search response from server server1.example.com:1389`
- `resultCode=timeout: errorMessage=An I/O error occurred while trying to read the response from the server`
- `resultCode=server down: errorMessage=An I/O error occurred while trying to read the response from the server`

- `resultCode=server down: errorMessage=The connection to server server1.example.com:1389 was closed while waiting for a response to search request SearchRequest`
- `resultCode=object class violation: errorMessage='Entry device=1234,dc=example,dc=com violates the Directory Server schema configuration because it contains undefined object class`

With these "Failure at Destination" timeout errors, look at the following settings in the PingDirectory Server to determine if adjustments are needed:

1. For External Server Properties, check the `connect-timeout` property. This property specifies the maximum length of time to wait for a connection to be established before giving up and considering the server unavailable.
2. For the Sync Destination/Sync Source Properties, check the `response-timeout` property. This property specifies the maximum length of time that an operation should be allowed to be blocked while waiting for a response from the server. A value of zero indicates that there should be no client-side timeout. In this case, the server's default will be used.

```
$ bin/dsconfig --no-prompt --port 389 \
  --bindDN "cn=Directory Manager" \
  --bindPassword password list-external-servers \
  --property connect-timeout
```

```
External Server         : Type             : connect-timeout : response-
                                                               timeout
------------------------:------------------:-----------------:-----------
server1.example.com:389 : sundsee-ds       : 10 s            : -
server2.example.com:389 : sundsee-ds       : 10 s            : -
server3.example.com:389 : ping-identity-ds : 10 s            : -
server4.example.com:389 : ping-identity-ds : 10 s            : -
```

3. For Sync Pipe Properties, check the `max-operation-attempts`, `retry-backoff-initialwait`, `retry-backoff-max-wait`, `retry-backoff-increase-by`, `retry-backoff-percentage-increase`. These Sync Pipe properties provide tuning parameters that are used in conjunction with the timeout settings. When a Sync Pipe experiences an error, it will use these settings to determine how often and quickly it will retry the operation.

```
$ bin/dsconfig --no-prompt list-sync-pipes \
  --property max-operation-attempts --property retry-backoff-initial-wait
\
  --property retry-backoff-max-wait --property retry-backoff-increase-by \
  --property retry-backoff-percentage-increase \
  --port 389 --bindDN "cn=Directory Manager" \
  --bindPassword password
```

Installation and maintenance issues

The following are common installation and maintenance issues and possible solutions.

The setup program will not run

If the `setup` tool does not run properly, some of the most common reasons include the following:

A Java Environment Is Not Available – The server requires that Java be installed on the system prior to running the `setup` tool.

If there are multiple instances of Java on the server, run the `setup` tool with an explicitly-defined value for the `JAVA_HOME` environment variable that specifies the path to the Java installation. For example:

```
$ env JAVA_HOME=/ds/java ./setup
```

Another issue may be that the value specified in the provided `JAVA_HOME` environment variable can be overridden by another environment variable. If that occurs, use the following command to override any other environment variables:

```
$ env UNBOUNDID_JAVA_HOME="/ds/java" UNBOUNDID_JAVA_BIN="" ./setup
```

Unexpected Arguments Provided to the JVM – If the `setup` tool attempts to launch the java command with an invalid set of arguments, it may prevent the JVM from starting. By default, no special options are provided to the JVM when running `setup`, but this might not be the case if either the `JAVA_ARGS` or `UNBOUNDID_JAVA_ARGS` environment variable is set. If the `setup` tool displays an error message that indicates that the Java environment could not be started with the provided set of arguments, run the following command:

```
$ unset JAVA_ARGS UNBOUNDID_JAVA_ARGS
```

The Server Has Already Been Configured or Started – The `setup` tool is only intended to provide the initial configuration for the server. It will not run if it detects that it has already been run.

A previous installation should be removed before installing a new one. However, if there is nothing of value in the existing installation, the following steps can be used to run the `setup` program:

- Remove the `config/config.ldif` file and replace it with the `config/update/config.ldif.{revision}` file containing the initial configuration.
- If there are any files or subdirectories in the `db` directory, then remove them.
- If a `config/java.properties` file exists, then remove it.
- If a `lib/setup-java-home` script (or `lib\set-java-home.bat` file on Microsoft Windows) exists, then remove it.

The server will not start

If the server does not start, then there are a number of potential causes.

The Server or Other Administrative Tool Is Already Running – Only a single instance of the server can run at any time from the same installation root. Other administrative operations can prevent the server from being started. In such cases, the attempt to start the server should fail with a message like:

```
The <server> could not acquire an exclusive lock on file
/ds/PingData<server>/locks/server.lock:
The exclusive lock requested for file
/ds/PingData<server>/locks/ server.lock
was not granted, which indicates that another
process already holds a shared or exclusive lock on
that file. This generally means that another instance
of this server is already running.
```

If the server is not running (and is not in the process of starting up or shutting down), and there are no other tools running that could prevent the server from being started, it is possible that a previously-held lock was not properly released. Try removing all of the files in the locks directory before attempting to start the server.

There Is Not Enough Memory Available – When the server is started, the JVM attempts to allocate all memory that it has been configured to use. If there is not enough free memory available on the system, the server generates an error message indicating that it could not be started.

There are a number of potential causes for this:

- If the amount of memory in the underlying system has changed, the server might need to be re-configured to use a smaller amount of memory.
- Another process on the system is consuming memory and there is not enough memory to start the server. Either terminate the other process, or reconfigure the server to use a smaller amount of memory.
- The server just shut down and an attempt was made to immediately restart it. If the server is configured to use a significant amount of memory, it can take a few seconds for all of the memory to be released back to the operating system. Run the `vmstat` command and wait until the amount of free memory stops growing before restarting the server.
- If the system is configured with one or more memory-backed filesystems (such as `/tmp`), determine if any large files are consuming a significant amount of memory. If so, remove them or relocate them to a disk-based filesystem.

An Invalid Java Environment or JVM Option Was Used – If an attempt to start the server fails with 'no valid Java environment could be found,' or 'the Java environment could not be started,' and memory is not the cause, other causes may include the following:

- The Java installation that was previously used to run the server no longer exists. Update the `config/java.properties` file to reference the new Java installation and run the `bin/dsjavaproperties` command to apply that change.
- The Java installation has been updated, and one or more of the options that had worked with the previous Java version no longer work. Re-configure the server to use the previous Java version, and investigate which options should be used with the new installation.
- If an `UNBOUNDID_JAVA_HOME` or `UNBOUNDID_JAVA_BIN` environment variable is set, its value may override the path to the Java installation used to run the server (defined in the `config/java.properties` file). Similarly, if an `UNBOUNDID_JAVA_ARGS` environment variable is set, then its value might override the arguments provided to the JVM. If this is the case, explicitly unset the `UNBOUNDID_JAVA_HOME`, `UNBOUNDID_JAVA_BIN`, and `UNBOUNDID_JAVA_ARGS` environment variables before starting the server.

Any time the `config/java.properties` file is updated, the `bin/dsjavaproperties` tool must be run to apply the new configuration. If a problem with the previous Java configuration prevents the `bin/dsjavaproperties` tool from running properly, remove the `lib/set-java-home` script (or `lib\set-java-home.bat` file on Microsoft Windows) and invoke the `bin/dsjavaproperties` tool with an explicitly-defined path to the Java environment, such as:

```
$ env UNBOUNDID_JAVA_HOME=/ds/java bin/dsjavaproperties
```

An Invalid Command-Line Option was Used – There are a small number of arguments that can be provided when running the `bin/start-server` command. If arguments were provided and are not valid, the server displays an error message. Correct or remove the invalid argument and try to start the server again.

The Server Has an Invalid Configuration – If a change is made to the server configuration using `dsconfig` or the Administrative Console, the server will validate the change before applying it. However, it is possible that a configuration change can appear to be valid, but does not work as expected when the server is restarted.

In most cases, the server displays (and writes to the error log) a message that explains the problem. If the message does not provide enough information to identify the problem, the `logs/config-audit.log` file provides recent configuration changes, or the `config/archived-configs` directory contains configuration changes not made through a supported configuration interface. The server can be started with the last valid configuration using the `--useLastKnownGoodConfig` option:

```
$ bin/start-server --useLastKnownGoodConfig
```

To determine the set of configuration changes made to the server since the installation, use the `config-diff` tool with the arguments `--sourceLocal --targetLocal --sourceBaseline`. The `dsconfig --offline` command can be used to make configuration changes.

Proper Permissions are Missing – The server should only be started by the user or role used to initially install the server. However, if the server was initially installed as a non-root

user and then started by the root account, the server can no longer be started as a non-root user. Any new files that are created are owned by root.

If the user account used to run the server needs to change, change ownership of all files in the installation to that new user. For example, if the server should be run as the "ds" user in the "other" group, run the following command as root:

```
$ chown -R ds:other /ds/PingData<server>
```

The server has shutdown

Check the current server state by using the `bin/server-state` command. If the server was previously running but is no longer active, potential reasons may include:

- Shut down by an administrator – Unless the server was forcefully terminated, then messages are written to the error and server logs stating the reason.
- Shut down when the underlying system crashed or was rebooted – Run the `uptime` command on the underlying system to determine what was recently started or stopped.
- Process terminated by the underlying operating system – If this happens, a message is written to the system error log.
- Shut down in response to a serious problem – This can occur if the server has detected that the amount of usable disk space is critically low, or if errors have been encountered during processing that left the server without worker threads. Messages are written to the error and server logs (if disk space is available).
- JVM has crashed – If this happens, then the JVM should provide a fatal error log (a `hs_err_pid<processID>.log` file), and potentially a core file.

The server will not accept client connections

Check the current server state by using the `bin/server-state` command. If the server does not appear to be accepting connections from clients, reasons can include the following:

- The server is not running.
- The underlying system on which the server is installed is not running.
- The server is running, but is not reachable as a result of a network or firewall configuration problem. If that is the case, connection attempts should time out rather than be rejected.
- If the server is configured to allow secure communication through SSL or StartTLS, a problem with the key manager or trust manager configuration can cause connection rejections. Messages are written to the server access log for each failed connection attempt.
- The server may have reached its maximum number of allowed connections. Messages should be written to the server access log for each rejected connection attempt.

- If the server is configured to restrict access based on the address of the client, messages should be written to the server access log for each rejected connection attempt.
- If a connection handler encounters a significant error, it can stop listening for new requests. A message should be written to the server error log with information about the problem. Restarting the server can also solve the issue. Another option is to create an LDIF file that disables and then re-enables the connection handler, create the `config/auto-process-ldif` directory if it does not already exist, and then copy the LDIF file into it.

The server is unresponsive

Check the current server state by using the `bin/server-state` command. If the server process is running and appears to be accepting connections but does not respond to requests received on those connections, potential reasons for this include:

- If all worker threads are busy processing other client requests, new requests are forced to wait until a worker thread becomes available. A stack trace can be obtained using the `jstack` command to show the state of the worker threads and the waiting requests.

  If all worker threads are processing the same requests for a long time, the server sends an alert that it might be deadlocked. All threads might be tied up processing unindexed searches.
- If a request handler is busy with a client connection, other requests sent through that request handler are forced to wait until it is able to read data. If there is only one request handler, all connections are impacted. Stack traces obtained using the `jstack` command will show that a request handler thread is continuously blocked.
- If the JVM in which the server is running is not properly configured, it can spend too much time performing garbage collection. The effect on the server is similar to that of a network or firewall configuration problem. A stack trace obtained with the `pstack` utility will show that most threads are idle except the one performing garbage collection. It is also likely that a small number of CPUs is 100% busy while all other CPUs are idle. The server will also issue an alert after detecting a long JVM pause that will include details.
- If the JVM in which the server is running has hung, the `pstack` utility should show that one or more threads are blocked and unable to make progress. In such cases, the system CPUs should be mostly idle.
- If a there is a network or firewall configuration problem, communication attempts with the server will fail. A network sniffer will show that packets sent to the system are not receiving TCP acknowledgment.
- If the host system is hung or lost power with a graceful shutdown, the server will be unresponsive.

If it appears that the problem is with the server software or the JVM, work with a support provider to diagnose the problem and potential solutions.

Problems with the Administrative Console

If a problem occurs when trying to use the Administrate Console, reasons may include one of the following:

- The web application container that hosts the console is not running. If an error occurs while trying to start it, consult the logs for the web application container.
- If a problem occurs while trying to authenticate, make sure that the target server is online. If it is, the access log may provide information about the authentication failure.
- If a problem occurs while interacting with the server instance using the Administrative Console, the access and error logs for that instance may provide additional information.

Problems with SSL communication

Enable TLS debugging in the server to troubleshoot SSL communication issues:

```
$ dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name
com.unboundid.directory.server.extensions.TLSConnectionSecurityProvider \
  --set debug-level:verbose \
  --set include-throwable-cause:true
```

```
$ dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Debug Logger" \
  --set enabled:true \
  --set default-debug-level:disabled
```

In the `java.properties` file, add `-Djavax.net.debug=ssl` to the `start-ds` line, and run `bin/dsjavaproperties` to make the option take effect on a scheduled server restart.

Conditions for automatic server shutdown

All PingData servers will shutdown in an out of memory condition, a low disk space error state, or for running out of file descriptors. The PingDirectory Server will enter lockdown mode on unrecoverable database environment errors, but can be configured to shutdown instead with this setting:

```
$ dsconfig set-global-configuration-prop \
  --set unrecoverable-database-error-mode:initiate-server-shutdown
```

Insufficient memory errors

If the server shuts down due to insufficient memory errors, it is possible that the allocated heap size is not enough for the amount of data being returned. Consider increasing the heap size, or reducing the number of request handler threads using the following `dsconfig` command:

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTP Connection Handler" \
  --set num-request-handlers:<num-of-threads>
```

Enable JVM debugging

Enable the JVM debugging options to track garbage collection data for the system. These options can impact JVM performance, but provide valuable data to tune the server. While the `jstat` utility with the `-gc` option can be used to obtain some information about garbage collection activity, there are additional arguments that can be added to provide additional detail, such as:

```
-XX:+PrintGCDetails
-XX:+PrintTenuringDistribution
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintGCDateStamps
```

Perform the following steps to enable these options for the server:

1. On the server, navigate to the `config/java.properties` file.
2. Edit the `config/java.properties` file. Add any additional arguments to the end of the line that begins with `start-<server>.java-args`.
3. Save the file.
4. Run the following command for the new arguments to take effect the next time the server is started:

   ```
   $ bin/dsjavaproperties
   ```

Index