PingDataGovernance™

Release 7

Guides

PingDataGovernance 7.0.1 Server Administration Guide 3

PingDataGovernance 7.0 Server Administration Guide 123

Release 7.0.x Platform Suppport Matrix 245

PingDataGovernance™

Release 7.0.1

Server Administration Guide

PingDataGovernance Server™ Product Documentation

Support

https://support.pingidentity.com/

Table of Contents

Chapter 1: Introduction **1**

PingDataGovernance Server overview 2

PingDataGovernance Server features 2

PingDataGovernance Server deployment considerations 3

Configuration overview 3

SCIM 4

Data Sources 4

Scopes and Policies 4

System 5

Web Services and Applications 6

LDAP Administration and Monitoring 6

Logging, Monitoring, and Notifications 6

Chapter 2: Installation **8**

Installation prerequisites 9

Supported platforms 9

Set the file descriptor limit 9

Set the maximum user processes 10

Disable filesystem swapping 10

Install the dstat utility on SuSE Linux 11

Managing system entropy 11

Enable the server to listen on privileged ports 11

Install the JDK 12

Encryption keys 12

User store overview 12

Ping license keys 12

Install the PingDirectory Server 13

PingDataGovernance Server Installation Tools 15

Install the PingDataGovernance Server 15

Configure the PingDataGovernance Server 17

Log into the Administrative Console 19

Install an additional PingDataGovernance Server in a topology 19

Server folders and files 20

Plan a scripted installation 21

Start the PingDataGovernance Server 23
Stop the PingDataGovernance Server 23
Schedule a server shutdown 23
Run an in-core restart 24
Run the server as a Microsoft Windows service 24
Register the service 24
Run multiple service instances 24
Deregister and uninstall 25
Log files 25
Update servers in a topology 25
Update the server 26
Reverting an Update 26
Uninstall the PingDataGovernance Server 27
Chapter 3: Data access and mapping 28
Data components 29
Store adapter mappings 29
Directory Servers 29
Primary and secondary store adapters 29
Defining correlation attributes 30
Sample configuration 30
SCIM schemas 32
Store adapter mappings 32
SCIM attribute search considerations 32
Maintain username uniqueness 33
Define SCIM Resource Types 33
Pass-through SCIM Resource Type 33
Mapping SCIM Resource Type 34
Create a SCIM Resource Type 34
Create a Mapping SCIM Resource Type 34
Create a Pass Through SCIM Resource Type 35
Edit attribute and sub-attribute properties 35
Edit store adapter mappings 37
Complex attribute mapping 38

Client-specific SCIM attributes 38
Access data 40
Chapter 4: Token access 41
PingDataGovernance Server endpoint for OAuth2 clients 42
Access token validation 42
PingFederate Access Token Validator 42
JWT token validation 43
Chapter 5: Configure scopes and policies 44
OAuth2 scopes 45
Authenticated Identity Scope 45
Resource Scope 46
Create scopes 47
Create an Authenticated Identity OAuth2 scope 47
Create a Resource OAuth2 scope 47
Policy overview 47
Requesting operations through SCIM 48
Policy structure 48
JEXL use in policy structure 49
JEXL identifiers and variables 49
References containing invalid JEXL characters 49
Extended data type support 50
JEXL extension functions 50
Use obligations and advice 51
SCIM resource requests 51
Filter obligation 51
Policies and request processing 52
SCIM resource type policy evaluation 52
SCIM search request 52
SCIM GET request 53
SCIM POST request 53
SCIM PATCH and PUT requests 54
SCIM DELETE request 54
Policy Decision Point (PDP) endpoint 55
Policy engine request context 55
XACML attribute categories 55

Standard attribute use 56
Custom XACML function 57
SCIM resource properties 57
Scope properties 57
SCIM request properties 58
Applicable scopes 59
Access token properties 59
Configure the Policy Service 60
Policy Information Providers 60
PIP Evaluation Order 60
Create policies 61
Creating a policy set 62
Troubleshoot policies with traces 62
Troubleshoot denied access 63
Chapter 6: Advanced configuration 64
General server configuration 65
Available configuration tools 65
Use the dsconfig tool 66
Administrative accounts 67
Change the administrative password 67
Use the Configuration API 67
Authentication and authorization 68
Relationship between the Configuration API and the dsconfig tool 68
GET example 69
GET list example 71
PATCH example 72
API paths 76
Sorting and filtering configuration objects 77
Update Properties 77
Administrative actions 79
Updating servers and server groups 80
Configuration API responses 80
Configuring HTTP connection handlers 81

Domain Name Service (DNS) caching 82
IP address reverse name lookups 83
Problems with SSL communication 83
Conditions for automatic server shutdown 83
Configuring traffic through a load balancer 84
System alarms, alerts, and gauges 84
Alert handlers 85
Test alarms and alerts 86
ConsPing Identityume admin alerts health check for PingDirectoryProxy Server 87
Logs and log publishers 87
Types of log publishers 87
View and configure log publishers 88
Create a new log publisher 88
Configure log compression 88
Configure log file encryption 89
Configure log signing 90
Configure log retention and log rotation policies 91
Configure the Log Rotation Policy 92
Configure the Log Retention Policy 92
Server monitoring 92
Backend monitor entries 92
View system and consent data through the PingDataMetrics Server 94
Using the status tool 94
Server SDK extensions 94
PingDataGovernance Server Advanced Server Configuration 95
Configuring Third-Party Store Adapters 95
Example Third-Party Store Adapter 96
Cross-Origin Resource Sharing Support 96
CORS Implementation 96
HTTP servlet services 97
HTTP servlet cross origin policies 97
Assigning a CORS policy to an HTTP servlet extension 98
Public and private key store configuration 99
Managing server encryption settings 100
Rotating the encryption key 100

Address a compromised encryption key 101
Topology configuration 102
Topology master requirements and selection 102
Topology components 103
Server configuration settings 103
Topology settings 104
Monitor data for the topology 104
Updating the server instance listener certificate 105
Remove the self-signed certificate 106
Prepare a new keystore with the replacement key-pair 106
Use an existing key-pair 106
Use the certificate associated with the original key-pair 107
Update the server configuration to use the new certificate 107
Update the ads-truststore file to use the new key-pair 108
Retire the old certificate 108
Index 109

Chapter 1: Introduction

The PingDataGovernance Server provides solutions to manage and monitor user data and access to account resources.

Topics include:

PingDataGovernance Server overview

PingDataGovernance Server features

PingDataGovernance Server deployment considerations

Configuration overview

PingDataGovernance Server overview

Most organizations today are working toward creating a unified customer profile. An essential part of creating that common profile is to centralize multiple, overlapping accounts and define the logic and security criteria for determining which applications should access data in a profile. The PingDataGovernance Server enables managing large amounts of customer data while ensuring end-user privacy.

The PingDataGovernance Server provides a common identity and single view of the customer by mapping account resources from multiple backend PingDirectory Servers to SCIM Resource Types defined in the PingDataGovernance Server. Restricted access to end users' information is maintained through policy rules .

PingDataGovernance Server features

The PingDataGovernance Server provides the following features to securely manage account resources:

- **Support for multiple backend Servers**. The PingDataGovernance Server supports multiple directory servers, with native support for the Ping Identity PingDirectory Server and extension points for others. PingDirectory Servers serve as user stores to provide the resources that can be requested by clients. Clients can be written one time for access to the PingDataGovernance Server and receive data from any type of infrastructure backend.
- **Support for Open Banking**. The PingDataGovernance Server supports a subset of Open Banking APIs. A working Open Banking solution requires a PingDataGovernance Server, a PingDirectory Server, and a PingFederate instance. These products working together provide an implementation for an Open Banking Account Servicing Payment Service Provider (ASPSP), which is the role played by a bank or other entity providing payment accounts.
- **Support for General Data Protection Regulation (GDPR)**. GDPR 2016/679 is a European Union (EU) regulation on data protection and privacy for all individuals within the EU, and businesses outside of the EU that handle EU individual's data. The GDPR aims primarily to give control to citizens and residents over their personal data through the use and management of consents. PingDataGovernance Server can be used with the PingDirectoryProxy Server's Consent Service to meet the security, tracking, and auditing requirements of GDPR. See the *PingDirectory Server Consent Guide* for details about configuring the Consent Service.
- **SCIM Resource Types**. SCIM Resource Types determine what attributes can be accessed by a client through the PingDataGovernance Server. The SCIM resource type

defines the resource name, endpoint URL, schemas, and other metadata that indicate where a resource is managed and how it is composed.

- **Access to resources based on policy**. The PingDataGovernance Server ensures that data is provided to authorized clients through the use of defined OAuth2 Scopes and policies. JEXL is used to define access control policies, and the processing model that determines how to evaluate requests based on rules defined in the policies. Policies can be based on industry rules, corporate policy, or consent granted by customers.
- **Application Developer Resources**. The enables client application developers to work with the PingDataGovernance Server APIs to design applications that can access PingDataGovernance Server resources. See the Ping Identity API site for configuration examples (`https://www.pingidentity.com/content/developer/en/explore.html`).

PingDataGovernance Server deployment considerations

The PingDataGovernance Server accepts client requests to access user data. Clients are granted authorization through an identity provider and receive access through the PingDataGovernance Server SCIM endpoint. The PingDataGovernance Server validates an OAuth2 access token request, where the scopes requested represent resources in backend servers.

Planning a PingDataGovernance Server deployment should start with defining what data can be accessed and updated from backend PingDirectory Servers, which are configured as User Stores. User Stores have a schema defined to surface attributes. SCIM Resource Types are then defined to enable access to attributes, and provide a unified view of identity data found in multiple PingDirectory Servers through Store Adapter Mappings. OAuth2 scopes are created to define the resources that can be requested by a client and the actions that can be performed on those resources.

Policies determine if a client can access requested scopes, based on the information provided with the request. Obligations within the policy can define conditions for access. Policies then determine the operations that can be performed on attributes within the requested scopes.

Configuration overview

PingDataGovernance Server configuration defines all server services, policies, applications, resources, and the mapping of data from one or more backend PingDirectory Servers. Configuration can be done from the command line with the `dsconfig` tool or through the Administrative Console interface. All settings have associated help text in the interface and in the linked Configuration Guide. The Configuration Guide contains details and relationship specifics for all configuration objects and is available from the Administrative Console interface or from the `<server-root>/docs/index.html` page.

SCIM

The SCIM protocol is an application-level, REST protocol for provisioning and managing identity data. The SCIM Schema provides a schema and extension for representing users and groups. Only those attributes defined in the SCIM Resource Type can be accessed through the PingDataGovernance Server. Any changes to these settings are saved to all PingDataGovernance Servers in a topology.

- **SCIM Resource Types** – Defines attribute mapping from a SCIM schema to native attributes found in PingDirectory Server entries. A pass-through SCIM Resource Type can also be created to allow the addition of new attributes that are not mapped to any in a PingDirectory Server. The SCIM schema defines the attributes that comprise a SCIM Resource Type. The SCIM Resource Type determines the attributes that can be accessed by a client application.
- **SCIM Schemas** – Specifies the SCIM 2.0 schemas for data that can be accessed from backend PingDirectory Servers. Schemas provide the basis for creating SCIM Resource Types.

Data Sources

Data sources are the servers that house the resources governed by the PingDataGovernance Server.

- **External Servers** – Lists the LDAP PingDirectory Server instances that are configured with the PingDataGovernance Server.
- **LDAP Health Checks** – Checks the status of external LDAP servers on a regular basis, and examines failures to determine if the server has become unavailable. This is an advanced setting.
- **Load Balancing Algorithms** – Used to determine the appropriate LDAP external server to use to process a request. They may be used to provide improved availability and performance by distributing the workload across multiple backend servers. This is an advanced setting.
- **Store Adapters** – Provides a PingDirectory Server interface to the PingDataGovernance Server. Changes or additions to Store Adapters are saved to all PingDataGovernance Servers in a topology. Third-party store adapters can be created with the Server SDK.

Scopes and Policies

These settings define the rules for accessing resources through the PingDataGovernance Server. Any changes to these settings are saved to all PingDataGovernance Servers in a topology.

- **Access Token Validators** – Validates an access token used to access protected resources (OAuth2 scopes). Validators are used to decode tokens and return token metadata. The PingDataGovernance Server's local access token validator can be used, or a third-party token validator can be defined using the Server SDK.
- **OAuth Scopes** – Specifies the data being requested from a client.
- **Policy Information Providers** – Retrieves XACML attributes from a Policy Information Point (PIP) for policy evaluation. This is an advanced setting.
- **Policies** – Specifies the rules for how requested resources can be shared with clients, written with JEXL (Java Expression Language) and based on the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0*. The PingDataGovernance Server provides default policies that can be used or modified.
- **Policy Service** – Contains the properties that affect the overall operation of the PingDataGovernance Server Policy Decision Point (PDP).

System

System settings define communication, connection, and the criteria for triggering alarms regarding the server's resources. Changes to these setting can be saved to the local server or saved to a group of servers. Most are not mirrored across a topology, unless otherwise stated. See General Server Configuration for more information.

- **Connection Handlers** – Defines the settings for handling all interaction with the clients, including accepting connections, reading requests, and sending responses.
- **Global Configuration** – Specifies the SMTP server, password policies, and LDAP request criteria configured for this server.
- **Key Manager Providers** – Manages the key material used to authenticate to another server. This is an advanced setting.
- **Key Pairs** – Defines the key pair that can be used to provide credentials for digital signatures. An existing key pair can be imported or a new one can be generated by the server. This configuration object is mirrored across a topology.
- **Locations** – Lists the locations in which servers that are accessed by the PingDataGovernance Server reside.
- **Trust Manager Providers** – Determine whether to trust certificates presented to the server. This is an advanced setting.
- **Trusted Certificates** – Specifies a trusted public key that can be used to verify credentials for digital signatures and public-key encryption. This configuration object is mirrored across a topology.

Web Services and Applications

These settings define the HTTP connection criteria for application access to the PingDataGovernance Server. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

- **HTTP Configuration** – Defines configuration for the PingDataGovernance Server HTTP Service. This is an advance setting and cannot be changed other than to include stack traces in error pages.
- **HTTP Servlet Cross Origin Policies** – Defines the configuration for handling Cross-Origin HTTP requests using the Cross Origin Resource Sharing (CORS) protocol. An instance of HTTP Servlet Cross Origin Policy can be associated with multiple HTTP Servlet Extensions.
- **HTTP Servlet Extensions** – Defines classes and initialization parameters used by a servlet invoked by an HTTP connection handler.
- **Web Application Extensions** – Specifies the configuration settings for the Administrative Console and any other web applications that are configured to work with the PingDataGovernance Server.

LDAP Administration and Monitoring

These are all advanced settings to manage the local server's accounts, account requirements and security settings, and backend configuration. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

Logging, Monitoring, and Notifications

These settings define the notification criteria for system alerts, and the logging criteria for actions within the PingDataGovernance Server. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

- **Alarm Manager** – Defines the severity of alarms to be raised.
- **Alert Handlers** – Specifies the Alert Handlers used to notify administrators of problems or events that occur in the PingDataGovernance Server.
- **Gauges** – Specifies server performance thresholds and circumstances that merit the raising of an alarm.
- **Gauge Data Sources** – Defines the source of gauge data obtained from the server, including available memory and disk space.

- **LDAP SDK Debug Logger** – Records debug messages generated by the LDAP SDK for Java. This is an advanced setting.
- **Log File Rotation Listeners** – Defines an action for the server to take before a log file is rotated out of service, such as copying the file to a new location. This is an advanced setting.
- **Log Publishers** – Defines the distribution of log messages from different loggers to a destination.
- **Log Retention Policies** – Defines how long logs should be kept.
- **Log Rotation Policies** – Specifies when log files should be rotated.
- **Monitor Providers** – Provides information about the state of the server or server components.

Chapter 2: Installation

The PingDataGovernance Server installation requires few prerequisites, and can be deployed on virtualized and/or commodity hardware.

Topics include:

Installation prerequisites

Encryption keys

User store overview

Ping license keys

Install the PingDirectory Server

PingDataGovernance Server installation tools

Install the PingDataGovernance Server

Configure the PingDataGovernance Server

Log into the Administrative Console

Install an additional PingDataGovernance Server in a topology

Server folders and files

Plan a scripted installation

Start the PingDataGovernance Server

Stop the PingDataGovernance Server

Run the server as a Microsoft Windows Service

Update servers in a topology

Uninstall the PingDataGovernance Server

Installation prerequisites

The following are required before installing the PingDataGovernance Server:

- Java 8
- Minimum of 2 GB RAM
- Ping PingDirectory Server 6.0, or later

Supported platforms

The PingDataGovernance 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 an 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.

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. The file descriptor limit on the operating system can be increased 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 server, 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 maximum user processes

Redhat Enterprise Linux Server/CentOS 6.x sets the default maximum number of user processes to 1024, which is lower than the setting on older distributions. This may cause JVM memory errors when running multiple servers on a machine because each Linux thread is counted as a user process.

At startup, the PingDataGovernance Server attempts to raise this limit to 16,383 if the value reported by `ulimit` is less. If the value cannot be set, an error message is displayed. Explicitly set the limit in `/etc/security/ limit.conf`. For example:

```
* soft nproc 100000
* hard nproc 100000
```

The 16,383 value can also be set in the `NUM_USER_PROCESSES` environment variable, or by setting the same variable in `config/num-user-processes`.

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`.

Install the dstat utility on SuSE Linux

The `dstat` utility is used by the `collect-support-data` tool to gather support data. It can be obtained from the OpenSuSE project website. Perform the following steps to install the `dstat` utility:

1. Log into the server as root.
2. Add the appropriate repository using the `zypper` tool.
3. Install the `dstat` utility:

   ```
   $ zypper install dstat
   ```

Managing 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 systems provide '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
```

Install the JDK

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

Encryption keys

Encryption setting definitions are used to validate encrypted access tokens. All PingDataGovernance Server instances must use the same set of definitions. Encryption setting definitions are managed using the `encryption-settings` tool.

If new encryption settings must be defined, the new definition must be exported using the `encryption-settings` tool and imported on all PingDataGovernance Server instances. Only after the new definition is imported on all servers can the new definition be used for subsequent encryption operations.

See Managing the Server Encryption Settings for more information.

User store overview

During the PingDataGovernance Server installation, at least one Ping PingDirectory Server is defined to serve as a user store. The user store is a repository of user data, such as names, email addresses, and preferences, as well as user-specific metadata needed by the PingDataGovernance Server. For example, some user data may be stored in an LDAP directory server while other attributes may be stored in a relational database. SCIM Resource Types are defined to enable access to a user store's resources, and provide a consistent view of a user's profile that may cross multiple PingDirectory Servers.

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 Admin 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 PingDirectory Server

The PingDataGovernance Server requires at least one installed Ping Identity PingDirectory Server. The PingDataGovernance Server can be configured with multiple user stores.

Note

All sensitive data in the user store can be encrypted. When using the Ping Identity PingDirectory Server as the user store, server-level encryption can be enabled as described in the "Encrypting Sensitive Data" section in the *Ping PingDirectory Server Administration Guide*.

The following information is needed during the installation:

- Server hostname
- LDAPS port
- Root DN and password
- Base DN
- Location of user entries

All configuration settings can be later modified through the `dsconfig` tool.

Perform the following steps to install the PingDirectory Server:

1. Download the PingDirectory Server zip distribution, `PingDirectory-<version>.zip`.
2. Unzip the file in any location.

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

3. Change to the top level PingDirectory folder.

   ```
   $ cd PingDirectory
   ```

4. Run the `setup` command.

   ```
   $ ./setup
   ```

5. Enter **yes** to agree to the license terms.

6. Enter the fully qualified host name or IP address of the local host, or press **Enter** to accept the default.
7. Create the initial root user DN for the PingDirectory Server, or accept the default, (`cn=Directory Manager`). This account has full access privileges.
8. Enter a password for this account, and confirm it.
9. To enable the Platform APIs over HTTPS, enter **yes**. These are the product's configuration APIs.
10. Enter the port to accept connections from HTTPS clients, or press **Enter** to accept the default. The default may be different depending on the account privileges of the user installing.
11. Enter the port to accept connections from LDAP clients, or press **Enter** to accept the default.
12. Type **yes** to enable LDAPS, or press **Enter** to accept the default (no). If enabling LDAPS, enter the port to accept connections, or press **Enter** to accept the default LDAPS port.
13. Type **yes** to enable StartTLS for encrypted communication, or press **Enter** to accept the default (no).
14. Select the certificate option for the server and provide the certificate location.
15. 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.
16. Specify the base DN for the PingDirectory Server repository, for example `dc=company,dc=com`.
17. Select an option to populate the database.

18. If this machine is dedicated to the PingDirectory Server, tune the JVM memory allocation to use the maximum amount of memory the **Aggressive** option). This ensures that communication with the PingDirectory Server is given the maximum amount of memory. Choose the best memory option for the system and press **Enter**.
19. Enter **yes** to configure the server on startup and load the backend into memory before accepting connections, or press **Enter** to accept the default (no).
20. To start the server after the configuration, press **Enter** for (yes).
21. Review the Setup Summary, and enter an option to accept the configuration, redo it, or cancel.

PingDataGovernance Server Installation Tools

The PingDataGovernance Server provides a number of tools to install and configure the system.

- The `setup` tool performs the initial tasks needed to start the PingDataGovernance Server server, including configuring JVM runtime settings and assigning listener ports for the PingDataGovernance Server's REST services and applications.
- The `create-initial-config` tool continues after `setup` and configures connectivity between the user store and the PingDataGovernance Server. During the process, the `prepare-external-store` tool prepares each Ping PingDirectory Server to serve as a user store by the PingDataGovernance Server. Configuration can be written to a file to use for additional installations.
- Once the configuration is done, the `dsconfig` tool and the Administrative Console enable more granular configuration.

Install the PingDataGovernance Server

To expedite the setup process, be prepared to enter the following information:

- An administrative account for the PingDataGovernance Server.
- An available port for the PingDataGovernance Server to accept HTTPS connections from REST API clients.
- An available port for the Administrative Console's communication.
- An available port to accept LDAP client connections.
- Information related to the server's connection security, including the location of a keystore containing the server certificate, the nickname of that server certificate, and the location of a truststore.

Perform the following steps for an interactive installation of the PingDataGovernance Server:

1. Download the latest zip distribution of the PingDataGovernance Server software.
2. Unzip the file in any location.

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

3. Change to the top level PingDataGovernance folder.
4. Run the `setup` command.

```
$ ./setup
```

5. Type **yes** to accept the terms of this license agreement.
6. The `setup` tool enables cloning a configuration by adding to an existing PingDataGovernance Server topology. For an initial installation, press **Enter** to accept the default (no). When adding additional PingDataGovernance Server instances, an existing peer can be chosen to mirror configuration.
7. Enter the fully qualified host name or IP address of the machine that hosts the PingDataGovernance Server, or press **Enter** to accept the default (local hostname).
8. Create the initial root user DN for the PingDataGovernance Server. This account has full access privileges. To accept the default (`cn=Directory Manager`), press **Enter**.
9. Enter and confirm a password for this account.
10. Define a port for PingDataGovernance Server REST APIs and the Administrative Console to accept HTTPS connections, or press **Enter** to accept the default.
11. Enter the port to accept LDAP client connections, or press **Enter** to accept the default.
12. To enable LDAPS connections press **Enter** and enter a port, or type **no**.
13. To enable StartTLS connections over regular LDAP connection press **Enter**, or type **no**.
14. Enter the certificate option for this server. If needed, the server will generate self-signed certificates that should be replaced before the server is put into production.
15. 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.

16. If this machine is dedicated to the PingDataGovernance Server, tune the JVM memory to use the maximum amount of memory (the **Aggressive** option). If this system supports other applications, choose an appropriate option.
17. Enter a location name for this server. The location is used for failover purposes if this server belongs to a server group.
18. Enter an instance name for this PingDataGovernance Server, or press **Enter** to accept the default (<location> Server 1). The name must be unique in a topology and cannot be changed once configured.
19. Press **Enter** (yes) to start the server when the configuration is applied.
20. Review the configuration options and press **Enter** to accept the default (set up the server).

The installation will continue with the `create-initial-config` tool.

Configure the PingDataGovernance Server

The next set of steps in the setup process rely on the `create-initial-config` tool. The `setup` tool will continue with the `create-initial-config` tool to configure the PingDataGovernance Server. Having the following in place will expedite the configuration:

- At least one Ping Identity PingDirectory Server is installed. Have the host name and communication port available.
- Any additional PingDirectory Servers that act as user stores. Only Ping PingDirectory Servers can be configured with this tool. Other user stores must be configured outside of this process. Have the host names and communication ports available.
- Locations for this and any other PingDataGovernance Servers for failover.

Note

The `create-initial-config` tool can install starter schemas that enable having a base schema for the product to use after installation and/or using reference applications. The schemas are `user` and `user-and-ref-apps`.If neither schema is installed, a custom schema and mapping SCIM resource type or a Pass-through SCIM resource type can be configured later.

After the initial setup and configuration, run the `dsconfig` tool later to make configuration adjustments. Perform the following steps to configure the PingDataGovernance Server:

1. Press **Enter** (yes) to continue with `create-initial-config`. If for some reason the initial configuration cannot be completed in one session, manually restart `create-initial-config` later.
2. Define the account used by the PingDataGovernance Server to communicate with an external User Store, or press **Enter** to accept the default (`cn=Governance User,cn=Root DNs,cn=config`).
3. Enter and confirm the account password.
4. Specify the type of security that the PingDataGovernance Server uses when communicating with all external store instances, or press **Enter** to accept the default (SSL).
5. Enter the `host:port` configured for the first PingDirectory Server. The connection is verified.
6. Select the location name for the PingDirectory Server (or user store server), or enter another location if not listed in the menu.
7. Confirm that the identified host should be prepared. If additional servers will be added as backups, select the **Yes, and all subsequent servers** option. This enables the identification of another server later in the configuration. The `prepare-external-store` tool can also be used to perform these tasks at a later time.
8. Enter the account and password needed to create the root user `cn=Governance User,cn=Root DNs,cn=config` account on the PingDirectory Server. This is the root account created in the initial setup, such as the default (`cn=Directory Manager`. The PingDataGovernance Server sets up the DN and tests that it can access the account. This is also the account used to log into the Administrative Console.
9. To configure the initial user store, press **Enter** for (yes). The user store will be configured with a default Store Adapter and SCIM Resource Type, which will enable mapping of resources in the user store.
10. If there are additional PingDirectory Server locations, enter their `host:port`. If there are no additional servers to add, press **Enter** to continue.
11. Choose one of the predefined schemas (the standard user schema and optionally the reference application schema), or no schema. The instructions for configuration in this guide use the standard user schema.
12. Specify the base DN for locating user entries, such as `ou=people,dc=example,dc=com` and press **Enter**.
13. Review the configuration summary, and press **Enter** to accept the default (w) to write the configuration to a `dsconfig` batch file. The configuration is written to `<server-root>/resource/governance-cfg.dsconfig` . Certificate files are written to `external-server-certs.zip`.

14. Press **Enter** (yes) to confirm that the configuration should be applied to this PingDataGovernance Server and written to the `governance-cfg.dsconfig` file.

This completes the initial configuration for the PingDataGovernance Server. Run the `bin/status` tool to see that the PingDataGovernance Server server is up and running.

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. To log into the Administrative Console, use the initial root user DN specified during setup (by default `cn=Directory Manager`).

The `dsconfig` command or the Administrative Console can be used to create additional root DN users in `cn=Root DNs,cn=config`. These new users require the fully qualified DN as the login name, such as `cn=new-admin,cn=Root DNs,cn=config`. To use a simple user name (with out the `cn=` prefix) for logging into the Administrative Console, the root DN user must have the `alternate-bind-dn` attribute configured with an alternate name, such as "admin."

If the Administrative Console needs to run in an external container, such as Tomcat, a separate package `(<server-root>/resource/admin-console.zip`) can be installed according to that container's documentation.

Install an additional PingDataGovernance Server in a topology

A PingDataGovernance Server instance can be cloned to serve as an additional server in a topology. Adding a server to an existing topology copies the original PingDataGovernance Server's local configuration and links the two configurations. The configuration of PingDataGovernance Server's cluster items and the topology settings are automatically mirrored across servers in a topology. See Topology Overview for details.

Note

When setting up a new PingDataGovernance Server from an existing peer, the existing HTTP (S) connection handlers are not cloned. These connection handlers are created from scratch using default values of the new server and any specified port values.

1. Unpack the zip distribution in a folder different from the peer PingDataGovernance Server.
2. Run the `./setup` command in the `<server-root>` directory of the new server.
3. Accept the licensing agreement.
4. Enter **yes** to add this server to an existing PingDataGovernance Server topology.
5. Enter the host name of the peer PingDataGovernance Server server from which the configuration will be copied.

6. Enter the port of the peer PingDataGovernance Server.
7. Choose the security communication to use to connect to the peer PingDataGovernance Server.
8. Enter the manager account DN and password for the peer PingDataGovernance Server. The connection is verified.
9. Enter the fully-qualified host name or IP address of the local host (the new server).
10. Enter the HTTPS client connection port for the PingDataGovernance Server, or press **Enter** to accept the default.
11. Select the option to install the Administrative Console application, if desired.
12. Enter the HTTPS connection port for the Administrative Console application, or press **Enter** to accept the default.
13. Enter the port on which the new PingDataGovernance Server will accept connections from LDAP clients, or press **Enter** to accept the default.
14. Choose a certificate option for this PingDataGovernance Server.
15. Choose the amount of memory to allocate to the JVM.
16. Choose the location for this server. The location of the peer is listed as an option, or a new location can be defined. Regardless, the new server will have its topology and cluster settings mirrored with its peer.
17. Enter a name for this server. The name cannot be changed after installation.
18. Press **Enter** to start the server after configuration.
19. Review the information for the configuration, and press **Enter** to set up the server with these parameters.
20. To write this configuration to a file, press **Enter** to accept the default (yes).

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 have been 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.
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems. (Update is not supported for version 6.0)

Directories/Files/Tools	Description
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
vendor_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	Legal notices for dependent software used with the product.
docs	Provides the release notes, Configuration Reference Guide (HTML), API Reference, and all other product documentation.
metrics	Stores the metrics that can be gathered for this server and surfaced in the Ping PingDataMetrics Server.
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
webapps	Stores the Administrative Console .war file, the Authentication interface reference application's war file and source, and third-party licenses.
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.

Plan a scripted installation

An interactive installation of an PingDataGovernance Server uses the `setup` and `create-initial-config` tools. This is the recommended installation method and should be used when possible. A scripted installation can be performed, for scenarios that require a custom configuration or automated deployment. The resulting `governance-cfg.dsconfig` batch file, created with the `create-initial-config` tool, can then be used as a basis for scripted installations.

The following is performed by the `create-initial-config` tool during an interactive installation:

External store preparation:

- For each Ping PingDirectory Server, the `prepare-external-store` tool is run. This updates the PingDirectory Server's schema, creates a privileged service account for use by the PingDataGovernance Server with the DN `cn=Governance User,cn=Root DNs,cn=config`, and creates an administrative account.
- If the user store is comprised of LDAP directory servers, the `prepare-external-store` tool is run for every server that comprises the user store. This updates the server's schema, and creates a privileged service account for use by the PingDataGovernance Server with the DN `cn=Governance User,cn=Root DNs,cn=config`.

Server configuration with `dsconfig`:

The `create-initial-config` command has a `--dry-run` option that can be used to generate the `governance-cfg.dsconfig` file in non-interactive, or interactive mode, without applying the configuration to the local server.

Note

The PingDirectory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which PingDataGovernance Server services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

Installation Process

The following is a sample of the commands that should be included in a scripted installation:

1. Set up and configure one or more PingDirectory Servers. See Installing the PingDirectory Server.
2. Run the PingDataGovernance Server `setup` command on the server that will host the PingDataGovernance Server.

```
$ ./setup --cli --no-prompt --acceptLicense --maxHeapSize 2g \
  --ldapPort 2389 --ldapsPort 2636 --httpsPort 8443 \
  --location Austin --instanceName server1 \
  --rootUserPassword <password> \
  --useJavaTrustStore <path>/keystores/truststore.jks \
  --useJavaKeystore <path>/keystores/server1keystore.jks \
  --trustStorePasswordFile<path>/keystores/truststore.txt \
  --keystorePasswordFile <path>/keystores/keystore.txt \
  --certNickname server-cert
```

   The `--trustStorePasswordFile` option is only required if this server is expected to update the truststore with certificates of other servers in the topology.

   The password for the private key associated with the certificate (`server-cert`) should be the same as the keystore password.

3. Run `prepare-external-store` for each user store.

```
$ ./prepare-external-store --no-prompt \
  --hostname ds1.example.com \
  --port 1636 --useSSL --trustStorePath <path>/keystores/truststore.jks
\
  --userStoreBaseDN "ou=people,dc=example,dc=com" \
  --governanceBindPassword <password> \
  --bindDN "cn=directory manager" \
  --bindPassword <password>
```

4. Run the `create-initial-config` tool.

```
$ ./create-initial-config --no-prompt \
  --port 2636 --useSSL --trustStorePath <path>/keystores/truststore.jks
\
  --bindDN "cn=Directory Manager" \
  --bindPassword <password> \
  --governanceBindPassword <password> \
  --externalServerConnectionSecurity useSSL \
  --userStoreBaseDN "o=people,dc=example,dc=com" \
  --userStore ds1.example.com:1636:Austin
```

Start the PingDataGovernance Server

To start the PingDataGovernance Server, run the `bin/start-server` tool on UNIX/Linux systems (the `bat` command is in the same folder for Windows systems).

To start the PingDataGovernance Server in the foreground:

1. Enter the `bin/start-server` with the `--nodetach` option to launch the PingDataGovernance Server as a foreground process.

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

2. Stop the PingDataGovernance Server by pressing CTRL-C in the terminal window where the server is running or run the `bin/stop-server` command from another window.

Stop the PingDataGovernance Server

The PingDataGovernance Server provides a shutdown script, `bin/stop-server`, to stop the server.

Schedule a server shutdown

The PingDataGovernance Server enables scheduling a shutdown and sending a notification to the `server.out` log file. The server uses the UTC time format if the provided timestamp includes a trailing "Z," for example, 201304032300Z. The following example includes a `--stopReason` option that writes the reason for the shutdown to the logs:

```
$ bin/stop-server --task \
  --hostname <server1.example.com> \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword <password> \
  --stopReason "Scheduled offline maintenance" \
  --start 201504032300Z
```

Run an in-core restart

Re-start the PingDataGovernance 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. Shutting down and restarting the JVM requires a re-priming of the JVM cache. To avoid destroying and re-creating the JVM, use an in-core restart, which can be issued over LDAP. The in-core restart will keep the same Java process and avoid any changes to the JVM options.

```
$ bin/stop-server \
  --task \
  --restart \
  --hostname <server1.example.com> \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword <password>
```

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.

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 `Ping-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 `Ping-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:

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

Uninstall the PingDataGovernance Server

The PingDataGovernance Server provides an `uninstall` tool to remove the components from the system. If this instance is a member of a topology of PingDataGovernance Server servers, the `uninstall` tool will remove it from the topology.

Note

If a PingDataGovernance Server is a member of a topology, and the `uninstall` tool is not used to remove it (it was shutdown and deleted manually), it will not be removed from the topology registry. In this scenario, use the `bin/remove-defunct-server` tool to remove the instance from the topology.

Perform the following to uninstall the PingDataGovernance Server:

1. From the server root directory, run the `uninstall` command.

   ```
   $ ./uninstall
   ```

1. Select the option to remove all components or select the components to be removed.
2. To selected components, enter **yes** when prompted.

   ```
   Remove Server Libraries and Administrative Tools? (yes / no) [yes]: yes
   Remove Log Files? (yes / no) [yes]: no
   Remove Configuration and Schema Files? (yes / no) [yes]: yes
   Remove Backup Files Contained in bak Directory? (yes / no) [yes]: no
   Remove LDIF Export Files Contained in ldif Directory? (yes / no) [yes]: no
   The files will be permanently deleted, are you sure you want to continue? (yes / no)
   [yes]:
   ```

3. Manually delete any remaining files or directories.

Chapter 3: Data access and mapping

PingDirectory Servers provide the resources that can be accessed by clients. Attributes can be mapped from multiple PingDirectory Servers to create a unified identity in a SCIM Resource Type. The SCIM Resource Type is the component that makes resources available to clients.

Topics include:

Data components

Primary and secondary store adapters

SCIM schemas

Store adapter mappings

SCIM attribute search considerations

Maintain username uniqueness

Define SCIM Resource Types

Complex attribute mapping

Client-specific SCIM attributes

Access data

Data components

When a PingDirectory Server is configured, a store adapter is installed to read and return native SCIM objects. Custom store adapters can be created for non-LDAP PingDirectory Servers with the Ping Identity Server SDK. See Server Extensions for information.

The attributes surfaced for each backend store are mapped in SCIM Resource Types to enable a unified view of a user profile, and to make them available to clients. The PingDataGovernance Server provides full read/write access through the SCIM Resource Type (`/scim/v2/Me`). The access to these resources is subject to policy rules and restrictions.

Store adapter mappings

A SCIM Resource Type enables attribute mappings between the native store adapter schema and the SCIM schema. The store adapter mapping can contain additional information as to whether the native attribute is readable, writable, searchable, or authoritative. One must be authoritative. A SCIM Resource Type can map attributes from multiple PingDirectory Servers and determine which attributes are the authoritative resource for a user profile. See Using SCIM Resource Type Attributes in Policy for details about policy evaluation.

Directory Servers

The user stores provide data resources. One or more Ping PingDirectory Servers, PingDirectoryProxy Servers, or third-party directory servers can serve as a user store. SCIM Resource Type mappings can be used to aggregate attributes from multiple PingDirectory Servers into a unified view.

When a store adapter is added to the PingDataGovernance Server's server configuration, a correlation attribute must be defined for SCIM Resource Types that are backed by multiple store adapters. The correlation attribute defines an attribute for each store adapter that is used to uniquely identify the same end user data across different store adapters. For example, if every PingDirectory Server stores a user's email address, and an email address can always be considered a primary key (that is, it is always unique per use), then each store adapter's email address attribute can be set as its correlation attribute.

Note

The PingDirectory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which PingDataGovernance Server services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

Primary and secondary store adapters

Store adapters contain the configuration that the PingDataGovernance Server uses to interact directly with external PingDirectory Servers. Every PingDirectory Server providing a distinct set of user data must have a store adapter entry in the configuration.

If the PingDataGovernance Server is used to aggregate user attributes from multiple PingDirectory Servers, secondary store adapters can be configured. "Primary store adapter" and "Secondary store adapter" designate how a SCIM Resource Type prioritizes user data lookups to multiple store adapters. The primary store adapter is always checked first when processing a request for a user resource, and then any secondary store adapters are checked. A user account effectively does not exist if a record does not exist for it on the primary store adapter. The primary store adapter should be used to store a user's core attributes, while a secondary store adapter can store additional attributes.

Defining correlation attributes

When handling a request for a particular user, the PingDataGovernance Server needs a way to correlate an entry in the primary store adapter with any related entries in secondary store adapters. This is done by correlating the value of an attribute shared across the store adapters using the secondary store adapter's `primary-correlation-attribute` and `secondary-correlation-attribute` properties. The correlation attribute should have a value that is unique for each user.

Note

When creating SCIM resources backed by secondary store adapters, the server automatically sets the secondary correlation attribute value if it does not already have a value from the resource create request.

For example, user entries can be correlated across store adapters by email address:

```
$ dsconfig create-secondary-store-adapter \
  --type-name Users \
  --adapter-name MarketingData \
  --set store-adapter:DemographicsStoreAdapter \
  --set primary-correlation-attribute:mail \
  --set secondary-correlation-attribute:emailAddress
```

Sample configuration

An environment may have two LDAP PingDirectory Servers with distinct sets of data. Set A may have user credentials and profile attributes, and is configured with the primary store adapter. Set B may have demographic data about these users, and is configured with the secondary store adapter. The following can be configured for this scenario:

1. Configure each server in Set A.

   ```
   $ bin/dsconfig create-external-server \
     --server-name profile-server \
     --type ping-identity-ds \
     ...
   ```

2. Configure each server in Set B.

   ```
   $ dsconfig create-external-server \
     --server-name demographics-server \
     --type ping-identity-ds \
     ...
   ```

3. Create LDAP load balancing algorithms.

```
$ dsconfig create-load-balancing-algorithm \
  --algorithm-name "Profile Store LBA" \
  --type failover \
  --set enabled:true \
  --set backend-server:profile-server
```

```
$ dsconfig create-load-balancing-algorithm \
  --algorithm-name "Demographics Store LBA" \
  --type failover \
  --set enabled:true \
  --set backend-server:demographics-server
```

4. Create store adapters.

```
$ dsconfig --adapter-name ProfileStoreAdapter \
  --type ldap \
  --set enabled:true \
  --set "load-balancing-algorithm:Profile Store LBA"
  ...
```

```
$ dsconfig --adapter-name ProfileStoreAdapter \
  --type ldap \
  --set enabled:true \
  --set "load-balancing-algorithm:Demographics Store LBA"
  ...
```

5. Designate the primary store adapter.

```
$ dsconfig create-scim-resource-type \
  --type-name Users \
  --type mapping \
  --set enabled:true \
  --set endpoint:Users \
  --set primary-store-adapter:ProfileStoreAdapter \
  --set core-schema:urn:example:schemas:Profile:1.0 \
  --set optional-schema-extension:urn:example:schemas:Demographics:1.0
```

6. Designate the secondary store adapter and correlation attributes.

```
$ dsconfig create-secondary-store-adapter \
  --type-name Users \
  --adapter-name MarketingData \
  --set store-adapter:DemographicsStoreAdapter \
  --set primary-correlation-attribute:mail \
  --set secondary-correlation-attribute:emailAddress
```

SCIM schemas

Each SCIM Resource Type maps to one core SCIM schema and optional extension schemas. SCIM schemas are used to define the resources that can be retrieved from a backend PingDirectory Server. Each SCIM Resource Type represents one type of resource, such as "user" or "account," and the schema defines the attributes of that resource.

Store adapter mappings

The PingDataGovernance Server uses store adapter mappings to determine which store adapter handles which attribute from the SCIM schema. For cases in which an attribute can be found on multiple store adapters, one store adapter mapping should be created for each combination of attribute and store adapter. One of these mappings must have the shared attribute set as `authoritative`. This designates the store adapter that will be the authoritative source when multiple possible values are found across a set of store adapters.

In the following example, the SCIM attribute `urn:pingidentity:schemas:sample:profile:1.0:topicPreferences` is mapped to the LDAP attribute `ubidXTopicPreferenceJSON` from the Marketing PingDirectory Server adapter:

```
$ bin/dsconfig create-store-adapter-mapping \
  --type-name Users \
  --mapping-name topicPreferences \
  --set secondary-store-adapter:DemographicsStoreAdapter \
  --set scim-resource-type-attribute:urn:example:schemas:Demographics:1.0:topicPreferences
\
  --set store-adapter-attribute:ubidXTopicPreferenceJSON \
  --set authoritative:true
```

SCIM attribute search considerations

In order to provide paging and sorting, the PingDataGovernance Server holds an entire search result set in memory while it processes a SCIM search request. This is true for searches that do not request paging or sorting. The SCIM Resource Type `lookthrough-limit` property sets an upper bound for searches, so that clients do not exhaust the server resources. If the number of search results for a given request exceeds this value, an error is returned to the client indicating that the search matched too many results. A request that causes an unindexed search is also restricted to the size limit of the `lookthrough-limit` setting.

The PingDataGovernance Server attempts to find a single store adapter that can process the provided search filter. The primary store adapter is checked first to see if it can process the search filter. If it cannot, the secondary store adapters are consulted in no particular order. The first store adapter capable of processing the search filter is chosen. The store adapter must be able to return a superset of possible matches for the filter. The attributes in the search filter must correspond to at least one searchable native attribute in the store adapter. If the SCIM Resource Type is a Mapping SCIM Resource Type, the store adapter mapping for the search filter attribute must be marked as `searchable`.

If no store adapters can process the search, the PingDataGovernance Server returns an error. For each candidate search result from a store adapter, the PingDataGovernance Server assembles a complete SCIM resource by retrieving the native resource for every other store adapter using the store adapter correlation attributes (set when secondary store adapters are defined) and merging them together. Each resulting candidate SCIM resource is checked to see if it matches the provided search filter and is discarded if it does not match.

Maintain username uniqueness

The PingDataGovernance Server's default schema configuration uses "`uid`" as the RDN attribute of user DNs, which ensures that all `uid` values are unique for that branch of the DIT. In the default configuration, `uid` is recognized as a user's username. The PingDataGovernance Serverstore adapter mapping for the `userName` attribute of the default starter schema relies on this.

It may be the case that the attribute used for the username is also an RDN attribute in the PingDirectory Server. If every entry resides on the same branch, these attribute values will always be unique. Any configuration changes that do not maintain this structure must ensure that usernames are unique. The Ping PingDirectory Server provides the attribute uniqueness plugin that can be used if configuration changes are required. See the *Ping PingDirectory Server Administration Guide*.

Define SCIM Resource Types

SCIM Resource Types provide a unified view of resources between the PingDataGovernance Server and one or more underlying PingDirectory Servers, and correspond to the SCIM 2.0 SCIM Resource Type. SCIM Resource Types determine what resources can be accessed from a PingDirectory Server. Each SCIM Resource Type represents one resource, such as "user" or "account" and the schema defines the attributes of that resource.

Note

When mapping attributes, PingDirectory Server attributes and SCIM Resource Type attributes must be of compatible types. For example, an attribute with an integer value must be mapped to another attribute with an integer value. An attribute with a string value can only be mapped to attributes with boolean, integer, or date-time if it can be parsed.

There are two types of SCIM Resource Types: Pass-through SCIM Resource Type and Mapping SCIM Resource Type. A Mapping SCIM Resource Type relies on a SCIM Schema, which is installed with the configuration of a user store on a PingDirectory Server.

Pass-through SCIM Resource Type

This type of SCIM Resource Type simply exposes the primary store adapter’s data as core attributes, while secondary store adapter’s data are exposed as schema extensions. No schema needs to be defined at the SCIM Resource Type and all schema enforcement is at the responsibility of the store adapters. Since no schema is defined at the SCIM Resource Type, attribute mappings are not defined. If the configured store adapter exposes a schema, it will be enforced as the core or extension schemas for the SCIM Resource Type.

Mapping SCIM Resource Type

Attributes associated with a SCIM Resource Type are configured by specifying at least one core schema and one or more schema extensions. The core schema defines attributes that can appear at the root level of the SCIM resource exposed by the SCIM Resource Type. Schema extensions define attributes that are namespaced by the schema's URI. Schema extensions can be optional or required. When processing client requests, the SCIM resource from the client is first checked against the schemas defined for the SCIM Resource Type (core or extension). The request is then mapped to a store adapter object, using the store adapter mappings, and then processed.

Create a SCIM Resource Type

After user stores and store adapters are in place, SCIM Resource Types can be defined to provide a unified view of identity data found in multiple PingDirectory Servers. The SCIM Resource Type determines the attributes that can be accessed by a client.

The following is a sample command for creating a mapping SCIM Resource Type:

```
$ bin/dsconfig create-scim-resource-type \
  --type-name Users \
  --type mapping \
  --set "description:Users Resource Type" \
  --set enabled:true \
  --set endpoint:/Users \
  --set primary-store-adapter:UserStoreAdapter \
  --set core-schema:urn:pingidentity:schemas:User:1.0 \
  --set required-schema-extension:urn:pingidentity:schemas:sample:profile:1.0
```

SCIM Resource Types can also be configured in the Administrative Console through **SCIM -> SCIM Resource Types**.

Create a Mapping SCIM Resource Type

The following information is used to configure a Mapping SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP store adapter.

- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.
- The core schema for the primary store adapter and any extension schemas.

Create a Pass Through SCIM Resource Type

The following information is used to configure a Pass Through SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP store adapter.
- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.

Edit attribute and sub-attribute properties

Attribute properties in the schema can be configured to change the actions that can be performed, and when an attribute is returned to a requesting client. If the attribute contains sub-attributes, those can be configured as well.

```
$ bin/dsconfig set-scim-attribute-prop \
  --schema-name urn:pingidentity:schemas:User:1.0 \
  --attribute-name displayName \
  --set "description:User's name."
  --set required:true \
  --set case-exact:true \
  --set mutability:read-only
```

This can be configured in the Administrative Console by editing a schema in **SCIM -> SCIM Schemas**. Select a schema and edit any of the attributes listed. The following can be configured for an attribute or sub-attribute:

- An optional description of the attribute.
- The attribute type, which can be:
  - **string** - A sequence of zero or more Unicode characters encoded using UTF-8.
  - **boolean** - The literal `true` or `false`.

   - **datetime** - A date and time encoded as a valid `xsd:dateTime` (for example, 2008-01-23T04:56:22Z).
   - **decimal** - A real number with at least one digit to the left and right of the period.
   - **integer** - A decimal number with no fractional digits.
   - **binary** - Arbitrary binary data.
   - **reference** - A URI for a resource. A resource can be a SCIM resource, an external link to a resource (such as a photo), or an identifier such as a URN. The `reference-type` property must be specified for these attributes.
   - **complex** - A singular or multi-valued attribute whose value is a composition of one or more sub-attributes.
- Specify if the attribute is required.
- Specify if the attribute is case-sensitive.
- Specify if the attribute can have multiple values.
- Specify suggested canonical values that can be used (such as work and home).
- The circumstances under which the values of the attribute can be written (mutability). Values include:
   - **read-only** - The attribute cannot be modified.
   - **read-write** - The attribute can be updated and read.
   - **immutable** - The attribute may have its initial value set, but cannot be modified after.
   - **write-only** - The attribute can be updated but cannot be read.
- The circumstances under which the values of the attribute are returned in response to a request. Values include:
   - **by-default** - The attribute is returned by default in all SCIM responses where attribute values are returned.
   - **upon-request** - The attribute is returned in response to any PUT, POST, or PATCH operations if the attribute was specified by the client (for example, the attribute was modified).
   - **always** - The attribute is always returned.
   - **never** - The attribute is never returned.
- The SCIM Resource Types that can be referenced. This property is only applicable for attributes that are of type `reference`. Valid values are a defined SCIM Resource Type, `external` indicating the resource is an external resource (such as a photo), or `uri` indicating that the reference is to a service endpoint or an identifier (such as a schema urn).

- If the attribute is complex and has sub-attributes, they can be edited as well with these values.

Edit store adapter mappings

Store adapters are designed to surface the schema of a backend Directory Server. Store adapter mappings map SCIM Resource Type attributes and store adapter attributes. When the PingDataGovernance Server is installed with a Ping PingDirectory Server, the schema attributes are automatically mapped to a User SCIM Schema Resource Type.

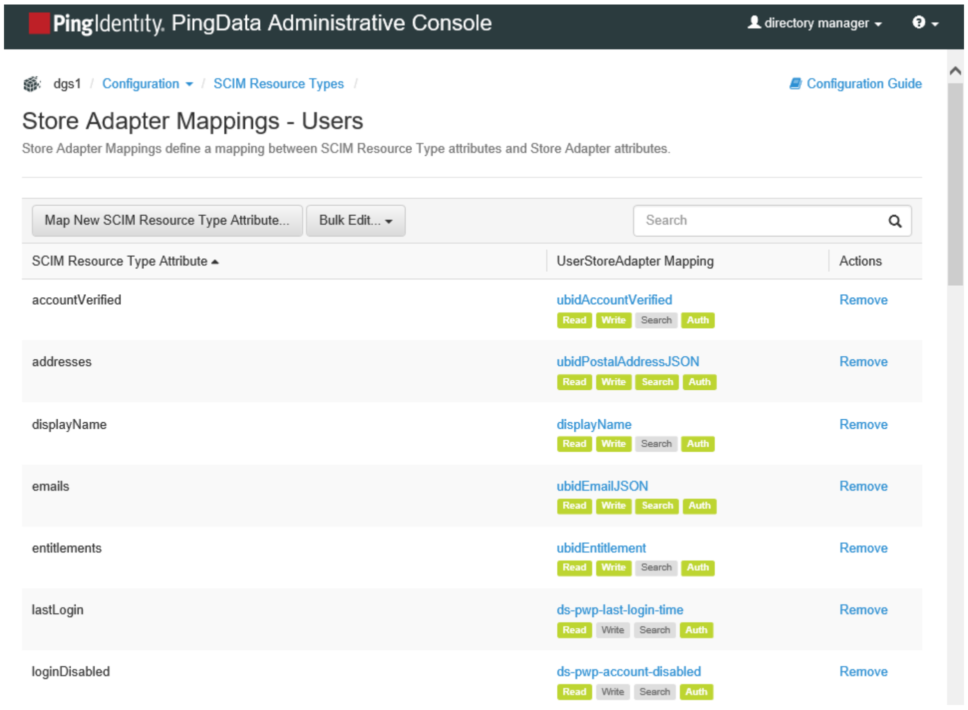
Note

If the SCIM Resource Type attribute name changes, make sure that scopes and OpenID Connect Claims are updated to reflect the change.

The following is a sample command for editing a store adapter attribute mapping:

```
$ bin/dsconfig set-store-adapter-mapping-prop \
  --type-name Users \
  --mapping-name communicationOpts \
  --set store-adapter-attribute:ubidXCommunicationOptJSON \
  --set writable:false \
  --set searchable:true \
  --reset authoritative
```

Store adapter mappings can also be configured in the Administrative Console through **SCIM -> SCIM Resource Types**. Click **Actions -> Edit Store Adapter Mappings** for a SCIM Resource Type. The following is displayed:


PingIdentity. PingData Administrative Console
directory manager
dgs1 / Configuration / SCIM Resource Types /
Configuration Guide
Map New SCIM Resource Type Attribute...
Bulk Edit...
Search

Store Adapter Mappings - Users

Store Adapter Mappings define a mapping between SCIM Resource Type attributes and Store Adapter attributes.

SCIM Resource Type Attribute	UserStoreAdapter Mapping	Actions
accountVerified	ubidAccountVerified Read Write Search Auth	Remove
addresses	ubidPostalAddressJSON Read Write Search Auth	Remove
displayName	displayName Read Write Search Auth	Remove
emails	ubidEmailJSON Read Write Search Auth	Remove
entitlements	ubidEntitlement Read Write Search Auth	Remove
lastLogin	ds-pwp-last-login-time Read Write Search Auth	Remove
loginDisabled	ds-pwp-account-disabled Read Write Search Auth	Remove

Individual attributes can be changed, or all can be edited by clicking **Bulk Edit**. For each attribute, the following can be configured:

- The store adapter attribute that is mapped to the SCIM Resource Type attribute.
- **Readable** – The SCIM Resource Type can read this attribute.
- **Writable** – The SCIM Resource Type can write to this attribute.
- **Searchable** – This specifies whether the attribute is efficiently searchable in the underlying PingDirectory Server. Indexed PingDirectory Server attributes determine what attributes (from the SCIM Resource Type Schema) can be used in a SCIM filter when performing a query. If an attribute is not indexed in the PingDirectory Server, it should not be marked as Searchable here.
- **Authoritative** – If there are multiple mappings for this attribute (from multiple PingDirectory Servers), one must be marked Authoritative.

Complex attribute mapping

For searches involving sub-attributes of SCIM attributes that are mapped to LDAP JSON attributes in the PingDirectory Server, the sub-attribute field names in the search filter are treated case-sensitively because the PingDirectory Server treats them this way. This is a departure from the SCIM 2.0 specification, where attribute names in search filters are case-insensitive.

For example, the SCIM attribute `name` has the sub-attribute `familyName`. The SCIM attribute `name` is mapped to the LDAP JSON attribute `scimName`. The search filter `NAME.FAMILYNAME eq "Zweig"` will not return a search result for an entry containing the specified value `Zweig` in the `familyName` sub-attribute. A search result for this entry is returned if the filter is specified instead as `NAME.familyName`. This is because the top-level attribute can be matched case-insensitively but the sub-attribute can only be matched case-sensitively.

Client-specific SCIM attributes

Some environments may find it useful to designate a namespaced, schema-less portion of a SCIM user resource, in which a client can store its data. For example, a resource type could be configured such that an application may write any previously undefined attributes that are prefixed with `urn:customApp1`.

To enable this, the data store schema must first have a single-valued JSON attribute defined to hold application-specific attributes. For example, for an LDAP attribute called `customApp`:

```
customApp: { "urn:customApp1":{ "wine":["Napa Cabs","French Burgundy","Lodi Zinfandel"],
"age":"2000-2010" } }
```

This value should appear in the SCIM resource as follows:

```
  'urn:customApp1' : {
      'wine' : [ 'Napa Cabs', 'French Burgundy', 'Lodi Zinfandel' ],
      'age' : '2000-2010'
  }
```

The following is a command line sample of the steps needed to configure this type of functionality in the PingDataGovernance Server, or this process can be done in the Administrative Console.

1. Create a store adapter mapping from "`*`" (SCIM) to "`customApp`" (LDAP). Using a wildcard SCIM attribute, client-specific SCIM attributes do not need to be defined in advance. To map only attributes from a single SCIM schema to an LDAP attribute, use a schema-specific SCIM wildcard such as `urn:myExtensionSchema:*`.

```
$ bin/dsconfig create-store-adapter-mapping \
  --type-name "Users" \
  --mapping-name "customAppWildcard" \
  --set "scim-resource-type-attribute:*" \
  --set store-adapter-attribute:customApp
```

2. Set the SCIM Resource Type's `schema-checking-option` property to `allow-undefined-attributes`.

```
$ bin/dsconfig set-scim-resource-type-prop \
  --type-name "Users" \
  --add schema-checking-option:allow-undefined-attributes
```

3. Define a wildcard scope that uses the client-specific namespace `urn:customApp1` as a prefix. Since the mapping is a wildcard, this prevents the client from reading or writing any user attribute, and client-specific attributes do not need to be defined in advance.

```
$ bin/dsconfig create-oauth2-scope \
  --scope-name Wildcard-Scope \
  --type authenticated-identity \
  --set "consent-prompt-text:Save application data to your account!" \
  --set "resource-attribute:urn:customApp1:*" \
  --set resource-operation:modify \
  --set resource-operation:retrieve
```

4. Create the client and assign the wildcard scope to it.

```
$ bin/dsconfig create-oauth2-client \
  --client-name "App1" \
  --set client-id:<App-ID> \
  --set client-secret:<secret> \
  --set grant-type:authorization-code \
  --set grant-type:implicit \
  --set scope:openid \
  --set scope:email \
  --set scope:Wildcard-Scope \
  --set redirect-url:https://company.com:<port>/client/
```

Access data

The SCIM endpoint provides full operations on user profile data through the SCIM protocol. The endpoint's URL context path is `/scim/v2/{name}`. Each SCIM resource, specified in the SCIM Schema, is exposed as an endpoint. For example, the URL path `/scim/v2/Users` would be used to access the `Users` SCIM resource. Access to resources is determined by the XACML policies that are configured for the PingDataGovernance Server. If a request to the PingDataGovernance Server is delivering partial results, it may be due to policy settings. See How Policy affects access to scopes.

The PingDataGovernance Server SCIM endpoint enables applications to perform actions on an end user's resources, if XACML policies permit. The following are important to consider when using the SCIM endpoint:

/Me. SCIM supports a special endpoint to retrieve attributes of the currently authenticated user without knowing the SCIM ID. Retrieve attributes of the currently authenticated user with the following:

```
/scim/v2/Me
```

Chapter 4: Token access

Each client request is processed by policies, which determine whether requested scopes can be granted. The PingDataGovernance Server validates the access tokens included with the client request to ensure that only authorized resources are accessed.

Topics include:

PingDataGovernance Server endpoint for OAuth2 clients

Access token validation

PingFederate Access Token Validator

JWT token validation

PingDataGovernance Server endpoint for OAuth2 clients

The Data Governance Server provides a SCIM REST endpoint for client access. The following list presents a summary of the endpoints that may be called by a client application requesting user profile data. See `<server-root>/docs/restapi/index.html` for details.

Data Governance Server endpoint for clients

Endpoint	Description
/scim	
/scim//v2/<name>	This is the SCIM 2.0 protocol endpoint used to retrieve a specified SCIM Resource Type, where `<name>` is the SCIM Resource Type being accessed. This endpoint supports all SCIM operations and implements its access control through the XACML policies. A request to this endpoint requires a scope that includes a `resourceOperations` value that represents the desired action.

Access token validation

Access Token Validators validate tokens submitted by client applications requesting access to protected resources. Any number of validators can be configured for the PingDataGovernance Server. Two types of access token validators are available. One for PingFederate tokens and another for JWT. Server SDK extensions can be installed to enable the PingDataGovernance Server to accept access tokens issued by other identity providers.

A third-party Access Token Validator is responsible for decoding an incoming access token and returning token metadata that is similar in content to that specified by RFC 7662. Metadata includes whether the token is valid and what scopes are granted to the token. This information is passed to the PingDataGovernance Server policy engine, which is responsible for determining whether the token should be accepted and if it is sufficient to allow the incoming request to be processed.

When an access token is presented with a resource request, the PingDataGovernance Server cycles through each configured Access Token Validator until it finds one that can decode the token.

PingFederate Access Token Validator

Before configuring a PingFederate Access Token Validator on the PingDataGovernance Server, complete the following tasks on the PingFederate instance:

1. Create a client on the PingFederate instance that represents the PingDataGovernance Server as a resource server.
2. Set the Allowed Grant Types for this client to include the Access Token Validation grant type.

3. Record the PingDataGovernance Server client ID and client secret for use when configuring the access token validator on the PingDataGovernance Server.

To configure an instance of the PingFederate Access Token Validator on the PingDataGovernance Server, have the following information:

- The base URL of the PingFederate instance (such as https://myPingFedInstance.example.com:9031).
- The client ID and client secret for the validator to use when validating the token with PingFederate.

Trust and key managers may also need to be configured for communication with the PingFederate server over SSL. See Public and Private Keystore Configuration.

If the access token validator will support tokens obtained through OAuth2 grant types other than Client Credentials, the token owner must be mapped to a user defined within the PingDataGovernance Server’s user store. This is done by creating one or more Token Resource Lookup Methods for the access token validator. Each Token Resource Lookup Method references a SCIM Resource Type and SCIM filter that are used by the PingDataGovernance Server to look up the local user that owns the access token. The SCIM filter is used to associate a token property (such as the `externalId` or `sub` claim) to a SCIM user with matching attribute(s). See the online *Configuration Reference Guide*, or `dsconfig` tool help.

JWT token validation

The JWT Access Token Handler enables self introspection of an access token. The JWT token can be encrypted or plain text.

The PingDataGovernance Server supports creating a key pair using RSA algorithms (Key Pairs). The public key needs to be exported from the PingDataGovernance Server and added to an instance of PingFederate or another client's Access Token Manager and configured to use this public key to encrypt the access token.

To verify signatures, the validator needs the public key(s) of the entity that is issuing access tokens. Those public keys can either be imported (as Trusted Certificates) or obtained from the JWKS endpoint of the access token issuer. Multiple keys are supported.

A mapping mechanism for `sub` and `client_id` is also available, if PingFederate or other client's Access Token Manager is configured to provide this information in non-standard claims. All JWT or custom claims should be made available in the request context for tokens.

Chapter 5: Configure scopes and policies

Scopes define the attributes that a client can request, the name that is displayed to end users, the claims that can be accessed, and the actions that can be performed on each attribute. Scopes must be defined in the PingDataGovernance Server before a client can include them in requests.

Policies are the rules that determine what resources may be accessed by client applications. Policies include the criteria by which access decisions are made using targets, rules, conditions, obligations, and a rule combining algorithm. Default policies are available, or custom policies can be written.

Topics include:

OAuth2 scopes

Create scopes

Policy overview

Policy structure

Policy and request processing

Policy engine request context

Configure the policy service

Policy Information Providers

Create policies

Create a policy set

Troubleshoot policies with traces

OAuth2 scopes

When a client makes a request for resources, it specifies the level of access that it requires using scopes. Based on the application’s configuration, and the policies that process the request, the PingDataGovernance Server decides whether the resource request should be permitted.

There are three scope types:

- Generic OAuth2 scope (used for external Resource servers).
- Authenticated Identity scope.
- Resource scope.

A Generic OAuth2 scope includes the following properties, which are the base properties for the Authenticated Identity and Resource scopes.

Generic OAth2 Scope Properties

Property	Description
tokenName	The scope name as presented in an OAuth2 request.
type	The scope type, which is `oauth2` for generic scopes.
description	A description of the scope for administrative use.
tags	A list of Tags associated with this scope. Tags are arbitrary additional properties that can be examined by policies.

Authenticated Identity Scope

This scope is granted for an authenticated end user. Once granted, the scope can be used to access the attributes of that authenticated identity. The attributes can be obtained through SCIM endpoints using the `/Me` authenticated subject alias as well as the URI of the SCIM resource.

Properties in this scope include those in the generic OAuth2 scope and the following properties. At least one of the operation properties must have a value.

Authenticated Identity Scope Properties

Property	Description
type	The scope type, which is `authenticatedIdentity` for authenticated identity scopes.
resourceOperations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>`, or `/Me` • `replace` (PUT) to endpoint `/scim/v2/<id>`

Authenticated Identity Scope Properties

Property	Description
	• `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
resourceAttributes	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace.

Resource Scope

An OAuth2 scope that allows a client bearing a granted token to access resources of a specified SCIM Resource Type. It defines the SCIM operations (search, create, retrieve, update, and delete) that can be performed by the client, and the attributes that can be retrieved or updated. A Resource scope potentially allows access (subject to policy) to all resources of a specified SCIM Resource Type.

Resource Scope Properties

Property	Description
type	The scope type, which is `resource` for resource scopes.
scimResourceType	The SCIM Resource Type that can be accessed with this scope.
resourceOperations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
resourceAttributes	A list of one or more SCIM attributes of the SCIM Resource Type for which this scope allows access. The type of access is determined by the operation properties `create`, `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace.

For granting access to PingDataGovernance Server resources, the values of the `resourceAttributes` property are attribute notation strings as defined in the SCIM 2.0, with the addition of being able to specify wildcards for all attributes.

Create scopes

An OAuth2 scope indicates which data are being requested with a resource request. Typically, one or more scopes are submitted with each request. Scopes are created based on the access and authentication requirements of the data requested. A standard set of OpenID Connect scopes is installed with the PingDataGovernance Server, and additional scopes can be created.

The following is a sample command for creating a scope:

```
$ bin/dsconfig create-oauth2-scope \
  --scope-name workPhone \
  --type authenticated-identity \
  --set resource-attribute:work-phone \
  --set resource-operation:modify
```

Scopes can also be created in the Administrative Console through **Authorization and Policies -> OAuth2 Scopes**.

Create an Authenticated Identity OAuth2 scope

The following information is used to configure an Authenticated Identity scope. See Authenticated Identity Scope for details about the values allowed for resource operations.

- An OAuth2 access token name that is compliant with the OAuth 2.0 Specification (RFC 6749). The following characters are not permitted: space, `'"'`, `'\'`, `'+'` and `','`.
- An optional description.
- Any optional tags associated with this scope. Tags are arbitrary additional properties that can be examined by XACML policies for authorization decisions, such as `HIPAA` or `billing`.
- Specify the resource operations allowed by this scope.
- Specify the resource attributes for which this scope allows access. The type of access is determined by the Resource Operation property. A value of "`*`" indicates that all attributes are accessible.

Create a Resource OAuth2 scope

All of the Authenticated Identity values are available for the Resource scope, with the addition of the SCIM Resource Type from which the scope can access resources.

Policy overview

Policies determine the scopes that can be accessed by requesting clients through the use of an access token, and the operations on attributes within the scope that are allowed. Policy creation must balance the privacy requirements of the organization with the resource access requirements of the clients. Policies are based on the the eXtensible access control markup language (XACML) as specified in the *OASIS Committee Specification 01, eXtensible access* *control markup language (XACML) Version 3.0*. The targets, rules, conditions, and rule combining algorithms are expressed using JEXL. The native language features of JEXL duplicate a large subset of functions defined by XACML and provide a more concise mechanism for defining policy conditionals. See Policy structure for details about policy components.

Policies are evaluated by the PingDataGovernance Server in response to the following requests made by clients:

- All SCIM requests:
  - Search request
  - Get request
  - Update request
  - Create request
  - Delete request
- A request to the PDP endpoint.

To create policies that will work as expected, or to create clients that can access data correctly, review the parameters and attributes that will be included in the policy.

Requesting operations through SCIM

The PingDataGovernance Server uses policies to determine whether a request for resources should be granted given the scopes defined in the access token. Obligations can be used to define conditions for limiting access to certain attributes. The requested attributes are returned to the client, and any permitted operation (such as adding or modifying an address) is performed.

Policy structure

Variable elements of policy, such as targets, rule conditions, variable definitions, and obligation/advice expressions, are specified using JEXL. For a policy to be evaluated against a request, the request needs to match the values specified in the policy Target element first. If the target for the request matches the target for the policy, the rules in the policy are evaluated. This occurs for each policy.

Just as there is a target for the policy, there is a target for each rule. For the rule Target element to be evaluated, a value in the request must match, as defined in the Match element. If the request matches a value, the rest of the conditions of the rule are evaluated.

Note

If no target is specified for a policy or a rule, the policy or rule is always evaluated.

If the conditions of a rule are satisfied, the result can be either `permit` or `deny` for that single rule. If there are multiple rules in a policy, the rule combining algorithm for the policy determines how the rule evaluation results are combined into a single policy decision.

If there are multiple policies that apply to the request, a policy-combining algorithm determines how the decisions rendered by multiple policies are combined to form a decision by the PingDataGovernance Server. By default, the combining algorithm for PingDataGovernance Server policies is `deny-overrides`. This can be changed in the Policy Service through the Administrative Console or with the `dsconfig` tool.

JEXL use in policy structure

XACML references to data from the request context are either Attribute Designators or Attribute Selectors. Attribute Designators refer to specific named attributes. Attribute Selectors allow JSON-path access to complex JSON objects. Using JEXL, all references to request context data use the syntax `category.value`, where `category` refers to the XACML category name and `value` can be either an attribute name or a JSON path. At policy evaluation time the PingDataGovernance Server first checks to see if `value` refers to a named attribute in the context, and if not will interpret it as a path.

The following JEXL condition can be used to deny a SCIM create request if the user to be created has a work email address in the ".gov" domain. It references the SCIM action attribute using its attribute ID and the user's email address through a JSON path.

```
action.action_id == "create" and scim_request.emails[type eq "work"].value =$ ".gov"
```

JEXL identifiers and variables

JEXL variable names are limited to alphanumeric characters, '`_`', and '`$`.' The PingDataGovernance Server supports the standard XACML attribute categories and attribute names, as described in Standard attribute use. Since JEXL variable names are limited to alphanumeric characters, the PingDataGovernance Server requires the use of short names rather than full URNs when referencing standard XACML attributes from JEXL.. For standard XACML names such as `urn:oasis:names:tc:xacml:3.0:attribute-category:action`, the short name is the relative portion of the URN after the final "`:`" character. In cases where this portion contains a '`-`' character it must be replaced with an underscore '`_`.' Therefore the standard attribute ID `urn:oasis:names:tc:xacml:1.0:action:action-id` should be referenced as `action_id`.

References containing invalid JEXL characters

Because SCIM schema URNs typically contain characters that are not allowed in JEXL identifiers, paths containing extension schemas must be quoted. For example, the following expression could be used to reference the manager ID of a user record that supports the standard SCIM 2.0 Enterprise User Schema Extension:

```
resource.'urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:manager'.id
```

SCIM also allows attribute names to contain dashes (`-`), where JEXL does not. These attributes must also be quoted in a JEXL expression.

Extended data type support

The PingDataGovernance Server JEXL evaluation can comprehend the XML string formats for `time`, `dateTime`, `date`, `dayTimeDuration`, and `yearMonthDuration`. For example, the following expression checks if the access token's expiration time is greater than the current time:

```
access_token.exp > environment.currentDateTime
```

JEXL extension functions

In addition to the set of JEXL functions provided, the following extensions are supported under the namespace `ext`.

XACML Function Access - An extension function is available to access XACML functions that JEXL does not support. The syntax to invoke a XACML function directly is `ext:xacml (functionName, functionArguments …)`. The `functionName` is the short name for the function. Function arguments can be any legal JEXL expression and may themselves refer to values from the XACML request context.

The following example invokes the PingDataGovernance Server's XACML extension function `scimAttribute-subset`. The function name itself is passed as a string, while function arguments may themselves be JEXL expressions.

```
ext:xacml("scimAttribute-subset", scim_request.impacted_attributes, applicable_
scope.scope.resourceAttributes)
```

Accessing Referenced SCIM Resource Attributes - An extension function allows access to SCIM objects that are indirectly referenced from the XACML request context. This supports, for example, the ability to examine data through the SCIM reference attribute type as described in RFC 7643.

The syntax for accessing a referenced SCIM object is `ext:scimReference(String referencePath, String attributePath)` where `referencePath` specifies where in the XACML request context to find the SCIM reference string, and `attributePath` is a JSON path that selects content from the referenced SCIM object.

Consider a case where a User SCIM resource contains a `wallet` attribute that contains a list of references to Payment Method objects, each of which is itself a SCIM resource. Each payment method object in the list is identified by its canonical type (`credit`, `payPal`, `check`). So the value of `resource.wallet[type == "credit"]` might be `PaymentMethod/987654`, which is the ID of a Payment Method object. The ID itself is not of much use to a policy, however the attributes of the referenced Payment Method object are. For example, the following expression would return the expiration date of the user's credit card:

```
ext:scimReference('resource.wallet[type eq "credit"]', 'expirationDate')
```

Note

Policies are not able to use this method to resolve SCIM reference attributes whose value is an external or absolute URI.

Use obligations and advice

The XACML specification defines an obligation as a specified operation that should be performed by the Policy Enforcement Point (PEP) based on an authorization decision. Advice is additional information provided to the PEP based on a policy decision, and can be used by the requesting client to determine why access to a resource was denied. Each obligation or advice type takes zero or more arguments, and the value of each argument is specified as a JEXL expression. The PingDataGovernance Server provides the following obligation types.

SCIM resource requests

Exclude Obligation – Specifies an argument "`attribute-names`" that lists the attributes to be excluded from the response. Each attribute must be formatted using SCIM Attribute Notation.

Include Obligation – Specifies an argument "`attribute-names`" that lists the attributes to be included in the response. Each attribute must be formatted using SCIM Attribute Notation.

The following example creates an exclude obligation that will prevent the `userName` attribute from being returned with a resource:

```
$ bin/dsconfig create-policy-obligation \
  --type exclude-attributes \
  --policy-name myPolicy \
  --rule-name someRule \
  --obligation-name someName \
  --set "attribute-names:['userName']"
```

Filter obligation

When a SCIM search request is sent, each record returned from the underlying user store is passed through the PingDataGovernance Server’s policy engine to determine whether the client is authorized to retrieve the returned resource. This can be expensive if there are a large number of results returned from the search.

The `add-filter` obligation can be used to give policy writers a more efficient way to restrict the results of a SCIM search operation. Rather than check each result against the server’s retrieve policy, this obligation requires the SCIM implementation to pre-filter the results by appending additional filter elements to the search request before the search is executed.

Policies can return zero or more `add-filter` obligations, each of which must specify a syntactically correct SCIM 2.0 filter expression, as described in the SCIM 2.0 Protocol specification. If the original SCIM request included a filter, then the additional filters are ANDed with the original filter.

Each filter obligation takes a single argument, which is the string-valued filter expression. The Attribute Assignment ID is `filter`, and its value may be either a string or a string bag. If the value is a string-bag, then the SCIM implementation will contenate together each string in the bag, with a space character between each segment, to form a single filter expression.

The following example will only permit records with tenant `Id = 7`:

```
$ dsconfig create-policy-obligation \
  --type add-filter \
  --policy-name myPolicy \
  --rule-name someRule \
  --obligation-name pickAnyName \
  --set "filter:tenantId eq 7"
```

Policies and request processing

Resource requests from a client are evaluated by the policy rules configured for the PingDataGovernance Server. This section describes each type of policy request that may be made by the PingDataGovernance Server's policy enforcement points.

SCIM resource type policy evaluation

Each request to the SCIM endpoint explicitly specifies what action is being requested and on what resources. As a REST interface, SCIM uses the HTTP method, query parameters, method body, and URI path to specify request parameters.

All SCIM requests target a specific SCIM Resource Type. For example, a search targeted to `/scim/v2/Users` is executed against the Users SCIM endpoint. An update targeted to `/scim/v2/ConsumerUsers/9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` is executed against a user with ID `9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` in the Users SCIM Resource Type.

SCIM search request

A SCIM search request consists of a search filter and an optional specification of which attributes to return from each record that satisfies the filter definition. The SCIM Resource Type against which the search is to be conducted is derived from the relative URL path, such as `/scim/v2/Users`.

The policy request generated from a SCIM search request contains the following attributes.

SCIM search request attributes

Attribute ID/Content	Attribute Category	Attribute Value
`subject_id`	access_subject	Name of the requesting client, if it can be retrieved from the OAuth2 access token.
`action_id`	action	`search`.
`resource_id`	resource	Relative URL of the SCIM endpoint, such as `Users`.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

After the search is run against the SCIM Resource Type, it generates policy requests for each record returned in the results to determine whether the requesting client has permission to receive the record’s attributes. Each resource and attribute of each record is evaluated independently through a separate policy request to determine if it can be returned. Any resources or individual resource attributes that are denied by policy are omitted from the response. These subsequent policy requests are identical to a SCIM GET request.

Note

The number of search results that can be returned is limited by the SCIM Resource Type's `lookthroughLimit` property, due to the potential cost of checking each response against policy.

SCIM GET request

The following is contained in the authorization request generated for a SCIM GET request for a known resource.

SCIM GET request attributes

Attribute ID/Content	Attribute Category	Value
`subject_id`	access_subject	Name of the requesting client, if it can be retrieved from the OAuth2 access token.
`action_id`	action	`retrieve`.
`resource_id`	resource	Relative URL of the resource or sub resource to retrieve, such as `Users/12345` or `/Users/12345/consents`.
`<JSON Content>`	resource	SCIM object representation of the requested resource.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

The SCIM endpoint will perform the following actions based on the result of the XACML policy authorization request:

- If the result is `deny` – The resource is not returned to the client and an error is returned.
- If the result is `permit` – The initial attribute set to be returned to the client is determined. Since multiple policies and/or rules may be consulted to make the permit decision, it's possible that multiple obligations will be returned with the result. See Use obligations and advice. Include and exclude obligations are processed as follows:
  - All attributes specified in an exclude obligation are removed from the attribute set.
  - If there are include obligations, all attributes that are not specified by an include obligation are removed from the attribute set.
  - If no attributes remain in the attribute set, a 200 success response code is returned but with an empty `resource` object.

These rules for each result type are used for all resources returned from the SCIM endpoint.

SCIM POST request

The following is contained in the authorization request generated for a SCIM POST request.

SCIM POST request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client name.
`action_id`	action	`create`.
`resource_id`	resource	Relative URL of the SCIM Resource Type to be created, such as `Users` or of the SCIM sub resource to be created, such as `/Users/12345/consents`.
`<JSON Content>`	scim_request	SCIM request body of the resource or sub resource to be created.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

If the POST operation is permitted, the new resource is created and the new object is returned to the client. After the POST is complete, a second policy request is issued to determine which attributes of the updated record the client can receive in the response.

SCIM PATCH and PUT requests

PUT requests are internally converted into a PATCH operation, which is why they are handled the same way by policy. The following is contained in the authorization request generated for a SCIM PATCH or PUT request for a known resource.

SCIM PATCH request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client name.
`action_id`	action	`modify`.
`resource_id`	resource	Relative URL of the resource or sub resource to be modified, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	scim_request	The normalized SCIM PATCH request body.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

If the PATCH or PUT operation is permitted, the resource is updated and returned to the client. The updated resource is then subject to the same read criteria in a GET request.

SCIM DELETE request

The following is contained in the authorization request generated for a SCIM DELETE request for a known resource.

SCIM DELETE request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client application name.
`action_id`	action	`delete`.
`resource_id`	resource	Relative URL of the resource or sub-resource to be deleted, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	access-token	Access token properties.
`<JSON Content>`	applicable-scope	Applicable scope objects.

Policy Decision Point (PDP) endpoint

The PDP endpoint enables an external Policy Enforcement Point (PEP) to generate policy requests and send them directly to the PingDataGovernance Server for evaluation. The request is passed directly to the policy engine. The request can contain any standard attributes, PingDataGovernance Server custom attributes, or other attributes that may be required by custom policies. This endpoint requires that the client authenticate using HTTP basic authentication.

Policy engine request context

The policy request context contains the information that is available to the policy engine to make a decision. A request for authorization (OAuth2) will provide information that helps the policy engine determine whether or not a client should be granted or denied access to a scope. A request for resources will provide information that will help determine if the operations on attributes in the requested scopes can be performed.

The request context contains attributes directly passed by a client when making an authorization request to the policy engine. It is supplemented with additional attributes and JSON objects that are retrieved from the attribute categories. In order to make a policy decision, policies can reference any attribute or JSON object from the request context.

XACML attribute categories

All references from policy to objects that can be obtained from the request context are first identified by their XACML attribute category.

- `resource` – This standard category definition is always used to reference the object to which authorization is being requested. With a SCIM request, this is a SCIM resource whose type is determined by the SCIM request path. See Resource properties for details.
- `access_subject` – This standard category definition contains the client ID, on whose behalf the policy request has been made.

- `access_token` – This custom category provides access to properties of the access token that has been used to make the current request. It exposes the access token as a JSON object. See Processing access tokens for details.
- `http_header` – This custom category provides access to the HTTP headers of the incoming request.
- `http_query_param` - – This custom category provides access to the HTTP query parameters of the incoming request. The following example retrieves the value of the query parameter with name `channel`:

  ```
  http_query_param.channel
  ```
- `scim_request` – This custom category is populated by the PingDataGovernance Server SCIM endpoint and contains the JSON request body of the SCIM request that triggered policy evaluation. The content from this attribute category is in standard SCIM 2.0 format. See SCIM request properties for details.
- `applicable_scope` – This custom category is populated with the scopes from the access token that are applicable to authorize a resource request. See Applicable scopes for details.
- `token_owner` – For requests authorized with an access token, this custom category provides access to the SCIM resource of the owner of the access token. See Access token properties for details.

Other attribute categories can be defined by custom PIPs.

Standard attribute use

The following request attributes are specified by the XACML specification. Unless otherwise specified, these are always available in the PingDataGovernance Server's policy request context.

Standard attributes

Attribute URN	Attribute Category	Data Type	Description
subject_id	access_subject	string	Contains the name of the client that is submitting a policy request.
ip_address	access_subject	ipAddress	Contains the originating IP address of the client's authorization request. The availability and accuracy of this attribute is dependent upon the deployed PingDataGovernance Server's network environment. When available, the value is retrieved from the XFORWARDED_FOR header of the client's HTTP request. If that header is not available, the IP address returned may be that of the last proxy to send the request.
resource_id	resource	anyURI	Contains the ID of the resource being requested.

Attribute URN	Attribute Category	Data Type	Description
action_id	action	string	Contains the name of the action being requested.
current_time	environment	time	The time at which the PingDataGovernance Server began processing the current authorization request.
current_date	environment	date	The date on which the current authorization request is being processed.
current_dateTime	environment	dateTime	The date and time at which the PingDataGovernance Server began processing the current authorization request.

Custom XACML function

There is a single custom function implemented by the PingDataGovernance Server. This can be accessed with the JEXL extension function. See JEXL extension functions.

The `scimAttribute_subset` function is similar to the standard XACML string-subset function, except that the arguments are bags of SCIM attribute names using SCIM attribute notation as described in the SCIM specification. The custom function comprehends wildcard attribute specifications as supported in the `resourceAttributes` property of a PingDataGovernance Server OAuth2 scope.

For example, if the second set passed to this function contains the string `urn:mySchema:*`, and the first set contains `urn:mySchema:myAttribute`, the function may still return TRUE (the first set is considered to be a subset of the second).

SCIM resource properties

SCIM Resource Type resources are exposed as JSON objects that can be accessed from policy. The format of the JSON object is determined by the structure of the underlying resource and the mappings defined for its SCIM Resource Type. When a client makes a SCIM request, the resource category content is a SCIM Resource. For example, the following JEXL expression will retrieve the `region` sub-attribute of a user's home address within the requested User resource.

```
resource.addresses[type eq "home"].region
```

Scope properties

The default scope validation policy allows resource operations as long as one of the scopes granted in the access token allows the operation. Access to attributes allowed per operation is the union of all `resourceAttributes` defined in Authenticated Identity or Resource scopes that allow that operation.

For operations to be allowed on resources, the policies that process the requests must allow the operations requested in the scope. The following scope properties can be evaluated by policies.

Scope properties

Property	Data Type	Description
`tokenName`	String.	The scope name as presented in an OAuth2 request.
`type`	String.	The scope type, which is `authenticated-identity` for authenticated identity scopes, `resource` for resource scopes, or `oauth2` for a generic scope.
`tags`	String. Multivalued.	A list of Tags associated with a scope that can be examined by XACML policies.
`scimResourceType`	Aggregation.	If a `resource` scope, the SCIM Resource Type that can be accessed.
`resourceOperations`	Multivalued list. Optional.	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
`resourceAttributes`	Multivalued string.	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace. Access to attributes allowed per operation is the union of all `resourceAttributes` allowed in the scope.

SCIM request properties

For policy evaluation of SCIM requests, the HTTP message body, if one exists, is available as the content of the `scim_request` attribute category. For SCIM POST requests, this content will be the JSON resource to be created, and for SCIM PATCH or PUT requests, the content will be a normalized SCIM PATCH request. For convenience, the attribute with ID `impacted_attributes` is also available. This attribute is computed by the policy engine and returns a list of attribute names in SCIM attribute notation. It returns only the attributes that can be created, modified, or deleted as a result of a SCIM POST, PUT, or PATCH request. See the SCIM 2.0 specification for more details.

The following JEXL example retrieves all impacted attributes from the current SCIM request:

```
scim_request.impacted_attributes
```

Applicable scopes

An access token presented by a client to the PingDataGovernance Server can contain many scopes, only some of which are applicable to the current request. The PingDataGovernance Server's PIP exposes the applicable scopes under the attribute category `applicable_scope`. This category contains a list of JSON scope objects, described in OAuth2 scopes, for those scopes granted by the access token that meet the following criteria:

- The current request's `action_id` is contained in one of the scope's operations properties.
- The type of resource requested matches the type of resource to which the scope grants access. For Authenticated Identity scopes, they are only applicable to requests in which the resource requested is the access token owner.
- Generic OAuth2 scopes are always included since their meaning is not defined by the PingDataGovernance Server.

The following example retrieves all attributes that are granted access by all applicable scopes of the access token:

```
applicable-scope.scope.resourceAttributes
```

Access token properties

The PingDataGovernance Server's Policy Information Provider (PIP) exposes access tokens as JSON objects under the attribute category `access_token`. What is available in the token is determined by the token provider.

The following properties are common in an access token.

Access token properties

Property	Data Type	Description
`active`	Boolean. Required.	`True` if the token is valid, `false` if token is invalid or has expired.
`client_id`	String.	The ID of the client to which this token is granted.
`iss`	String.	The issuer of the token.
`sub`	String.	The unique identifier for the token owner. For user tokens, this will be the relative SCIM path to the user resource, such as `Users/123456789`. For Client Credentials (`app`) tokens, this property is not present.
`scope`	Multivalued string.	A list of scope names granted by this token.
`app`	String.	The name of the client for which this token was created. For application tokens, this value will be equal to `sub`.
`iat`	DateTime.	The date and time at which the token was created.
`exp`	DateTime.	The date and time at which the token will expire.
`nbf`	DateTime.	The time at which the token becomes valid.
`jti`	String.	The unique token identifier.

Access token properties

Property	Data Type	Description
`token_type`	String.	The type of token.
`username`	String.	The user name of the token owner (not present for application tokens).

Configure the Policy Service

Policies are managed by the Policy Service. The default conditions of the Policy Service can be viewed and changed with the `dsconfig` tool, or through the Managment Console **Authorization and Policies -> Policy Service**.

The **combining-algorithm** determines how decisions are made if multiple policies or policy sets are applied to a request for resources. The default for the Policy Service is `deny-overrides`, which specifies that a "deny" decision from a policy should take priority over a "permit" decision. The PingDataGovernance Server also supports `permit-overrides`, `deny-unless-permit`, and `permit-unless-deny`. See the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0. August 2010* for details about each combining algorithm.

Add any custom logged policy request attributes, which enables additional request attributes to be included in the output of a Trace Log Publisher during policy evaluations. The URN of the XACML category ID and Attribute ID are required, in addition to the logger key.

Policy Information Providers

Policy Information Providers are used to retrieve attribute(s) from the Policy Information Point (PIP) during policy evaluation. This is an Advanced setting. See Standard attribute use and Custom attribute use for information about these attributes. The PingDataGovernance Server provides the following Policy Information Providers:

BuiltIn Policy Information Provider – Resolves policy attributes that are implemented by the PingDataGovernance Server.

SCIM Request Policy Information Provider – Resolves policy attributes whose value can be retrieved from an incoming SCIM request.

SCIM Resource Type Policy Information Provider – Resolves policy attributes whose value can be retrieved from a SCIM Resource Type configured on this PingDataGovernance Server instance.

Token Policy Information Provider – Resolves policy attributes whose value can be retrieved from an access token recieved by this PingDataGovernance Server instance.

PIP Evaluation Order

When multiple PIPs are defined, the evaluation order determines the correct provider to verify a specified policy attribute. Each PIP must have a unique evaluation value defined within a PingDataGovernance Server instance. PIPs with a smaller value are evaluated first.

Create policies

The Administrative Console, **Authorization and Policies -> Policies**, or the `dsconfig` tool can be used to create and manage policies.

SCIM Resource Policy Set – A container for policies that authorize requests for protected resources, including SCIM and UserInfo requests.

Scope Validation – Authorizes SCIM requests based on the scopes granted by the access token provided. The scope must also be configured to enable a requested action. See OAuth2 scopes for details.

Token Validation – Denies all SCIM resource requests that do not contain a valid access token.

The following would be used to create the Token Validation policy, which is one of the PingDataGovernance Server's default policies:

```
$ bin/dsconfig create-policy \
  --policy-name "Token Validation" \
  --set "combining-algorithm:deny-unless-permit"
```

```
$ bin/dsconfig create-policy-rule \
  --policy-name "Token Validation" \
  --rule-name "Validate Token" \
  --set "effect:permit" \
  --set "description:Permit access if token is present and valid." \
  --set "condition:access_token.active"
```

```
$ bin/dsconfig create-policy-rule \
  --policy-name "Token Validation" \
  --rule-name "Error Advice" \
  --set "effect:deny" \
  --set "description:Provide error message for denied token."
```

```
$ bin/dsconfig create-policy-advice \
  --type "denied-reason" \
  --policy-name "Token Validation" \
  --rule-name "Error Advice" \
  --advice-name "token error advice" \
  --set 'error:"invalid_token"' \
  --set 'error-description:"Access token is expired or otherwise invalid."'
```

Policies can also be created in the Administrative Console through **Authorization and Policies -> Policies**.

The following information is used to configure a policy .

- A unique name and optional description.
- If this policy needs a rule combining algorithm that overwrites the one specified in the Policy Service, specify one.
- An optional target request. This JEXL expression will filter the incoming authorization request to determine if the policy should be applied when authorizing the request. If a

target filter is specified here, it is applied in addition to the target applied by the policy set in which this resides.

- Add optional policy variable definitions. These are JEXL expressions including rule conditions, obligation expressions, and advice expressions.
- Add any policy advice expressions and error statements.
- Add any policy obligation expressions.

Note

Policies are not enabled until they are added to an enabled policy set.

Creating a policy set

A policy set is an ordered collection of policies that work together to perform a policy task. The policy set is a XACML-defined entity. The PingDataGovernance Server evaluates policy sets the same way it evaluates policies.

Creation of a policy set is the same as that of a policy. A policy set must be created from individual policies that have been configured in the PingDataGovernance Server.

Note

Policy sets can contain both policies and other policy sets. If the combining algorithm is ordered, a policy set may contain policies or other policy sets but not both. A policy set must not contain a direct or nested reference to itself.

Troubleshoot policies with traces

Policy decisions are frequently the result of a complex series of logical steps. Identifying the reason why a particular request is getting an unexpected result can be difficult. The PingDataGovernance Server can generate a trace of any policy decision, and log traces with in the File Based Trace Log Publisher with `dsconfig` or through the Administrative Console.

Note

Policy traces are logged in the File Based Trace Log Publisher. See Logs and log publishers.

A Policy Decision Trace is an XML document that is formatted like the XACML policies. It demonstrates the sequence of steps taken by the policy engine to come to a decision for a specific request. The elements of the trace parallel the policies, policy targets, and policy rules that are evaluated. The following are included:

- The first line of the log entry identifies the message type as `POLICY-DECISION-TRACE`.
- The parameters of the policy request being traced are listed, including the application, action, and resources.
- Following this is the trace itself, which is included in the `<DecisionTrace>` XML element.

The trace also includes entries for each policy, rule, and target evaluated during the decision process. Each entry contains a `result` XML attribute, which specifies the result of evaluating the corresponding policy element.

Troubleshoot denied access

Policies can issue advice expressions for any policy request that is denied. This passes additional information to the client as to the reason for denying access. The SCIM endpoint will look for error advice returned from the policy engine and include it in the error response generated for the client. If a policy denies a request without advice, the error response is `access_denied`.

The following error advice may be included in policy.

Policy error advice

Advice ID	Attribute ID	Value
	`error`	Error identifier or code. For SCIM responses, this value will be used to populate the `scimType` error parameter.
`request_denied_reason`	`error_description`	The value of the `error_description` parameter of an error response, or the detail parameter of a SCIM error response.

Chapter 6: Advanced configuration

The PingDataGovernance Server's non-user data consists of data in the server configuration. Generally, data in the server configuration define an individual PingDataGovernance Server instance, and can include its place in a server topology. Multiple server instances can be grouped in two ways to share or mirror configuration settings:

- Server Groups – Servers that are added to a server group in the global configuration can share configuration changes across the group, or not.
- Cluster – This is a topology management setting that enables a set of servers to be grouped by a functional purpose, and any change to one is mirrored to all. A master server verifies any configuration change before it is propagated to other servers in the group.

Note

All configuration objects and settings are described in the HTML Configuration Reference, which can be accessed from the Administrative Console or from the `<server-root>/docs/index.html` page. Information in this chapter highlights configuration of interest to a PingDataGovernance Server installation. For complete configuration options and details, see the Configuration Reference.

Topics include:

General server configuration

Data Governance Server advanced configuration

Configuring PingDataGovernance Server login pages

Topology management

General server configuration

There are tools and settings that are common across all Ping servers. These enable monitoring and managing the server, configuring and sending alerts and alarms, and managing the server's communication with clients. These configuration objects can be changed at the local server, with the option to apply changes to servers in a group.

Available configuration tools

There are several tools that can be used for server administration and maintenance in the `/bin` directory. The following is a sample of the command-line configuration tools:

Command-line tools

Tool	Description
backup	Run full or incremental backups on one or more PingDataGovernance Servers. This utility also supports the use of a properties file to pass predefined command-line arguments.
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
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.
create-initial-config	Create an initial PingDataGovernance Server configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the PingDataGovernance Server on Unix-based systems.
dsconfig	View and edit the PingDataGovernance Server configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the PingDataGovernance Server and its associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM arguments and the `JAVA_HOME` environment variable.
encryption-settings	Manage the server encryption settings database.
ldapdelete	Perform an LDAP delete operation.
ldapcompare	Perform an LDAP compare operation.
ldapmodify	Perform LDAP modify, add, and modify DN operations in the PingDataGovernance Server.
ldappasswordmodify	Perform LDAP password modify operations in the PingDataGovernance Server.
ldapsearch	Perform LDAP search operations in the PingDataGovernance Server.

Command-line tools

Tool	Description
ldif-diff	Compare the contents of two LDIF files, the output being an LDIF file needed to bring the source file in sync with the target.
ldifmodify	Apply a set of modify, add, and delete operations against data in an LDIF file.
list-backends	List the backends and base DNs configured in the PingDataGovernance Server.
manage-extension	Install or update extension bundles. An extension bundle is a package of extension (s) that utilize the Server SDK to extend the functionality of the PingDataGovernance Server. Any added extensions require a server re-start.
oauth2-request	Performs OAuth2 requests on the PingDataGovernance Server. This tool can be used to test OAuth2 functions of the PingDataGovernance Server, and to manage OAuth2 tokens on behalf of registered applications.
prepare-external-store	Prepares the external PingDirectory Servers for the PingDataGovernance Server. This is run as part of the `create-initial-config` tool during installation. This tool creates the PingDataGovernance Server user account, sets the correct password, and configures the account with required privileges. It will also install the necessary schema required by the PingDataGovernance Server.
remove-defunct-server	Removes a permanently unavailable PingDataGovernance Server after it has been removed from its topology by the `uninstall` tool.
restore	Restore a backup of the PingDataGovernance Server.
review-license	Review and/or accept the product license.
server-state	View information about the current state of the PingDataGovernance Server processes.
start-server	Start the PingDataGovernance Server.
status	Display basic server information.
stop-server	Stop or restart the PingDataGovernance Server.
sum-file-sizes	Calculate the sum of the sizes for a set of files.

Use the dsconfig tool

The `dsconfig` tool, is used to view or edit the PingDataGovernance Server configuration, and is parallel in functionality with the Administrative Console. This utility can be run in interactive mode, non-interactive mode, and batch mode. Interactive mode provides an intuitive, menu-driven interface for accessing and configuring the server.

To start `dsconfig` in interactive mode, enter the following command:

```
$ bin/dsconfig
```

The `dsconfig` tool provides a batching mechanism that reads multiple `dsconfig` invocations from a file and executes them sequentially. The batch file advantage is that it minimizes LDAP connections and JVM invocations required with scripting each call. To use batch mode to read and execute a series of commands in a batch file, enter the following command:

```
$ dsconfig --bindDN uid=admin,dc=company,dc=com \
  --bindPassword password \
  --no-prompt \
  --batch-file </path/to/config-batch.txt>
```

The `logs/config-audit.log` file can be used to review the configuration changes made to the PingDataGovernance Server and use them in the batch file.

Administrative accounts

Users that authenticate to the Config API or the Administrative Console are stored in `cn=Root DNs,cn=config`. These users must exist on all instances of the PingDataGovernance Server to manage a Topology of servers. The `setup` tool automatically copies one administrative account when performing an installation from a peer, but if changed, the accounts must be synchronized. 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
   ```

Use the Configuration API

Ping 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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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."
    }
  ]
}
...
```

Updating 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:pingidentity: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 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

Response Code	Description	Response Body
		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."
}
```

Configuring HTTP connection handlers

The server relies on the HTTP connection handler, which relies on one or more servlet extensions. Servlet extensions are responsible for obtaining Java servlets and registering them to be invoked using one or more context paths. For custom servlet extensions created using the Server SDK, the process varies based on using a Java-based or Groovy-scripted extension. See the Server SDK documentation for details.

HTTP connection handlers are responsible for managing the communication with HTTP clients and invoking servlets to process requests from those clients. They can also be used to host web applications on the server. Each HTTP connection handler must be configured with one or more HTTP servlet extensions and zero or more HTTP operation log publishers.

If the HTTP Connection Handler cannot be started (for example, if its associated HTTP Servlet Extension fails to initialize), this does not prevent the entire server from starting. The server's start tool posts any errors to the error log.

The configuration properties available for use with an HTTP connection handler include:

- `listen-address` – Specifies the address on which the connection handler will listen for requests from clients. If not specified, then requests will be accepted on all addresses bound to the system.

- `listen-port` – Specifies the port on which the connection handler will listen for requests from clients. Required.
- `use-ssl` – Indicates whether the connection handler will use SSL/TLS to secure communications with clients (whether it uses HTTPS rather than HTTP). If SSL is enabled, then key-manager-provider and trust-manager-provider values must also be specified.
- `http-servlet-extension` – Specifies the set of servlet extensions that will be enabled for use with the connection handler. You can have multiple HTTP connection handlers (listening on different address/port combinations) with identical or different sets of servlet extensions. At least one servlet extension must be configured.
- `http-operation-log-publisher` – Specifies the set of HTTP operation log publishers that should be used with the connection handler. By default, no HTTP operation log publishers will be used.
- `ssl-cert-nickname` – In scenarios where the multiple public-private key pairs are in a JKS keystore, the LDAPConnectionHandler allows choosing a specific certificate alias through the `ssl-cert-nickname` property. The HTTPConnectionHandler for HTTPS connections should have the same option for parity.
- `key-manager-provider` – Specifies the key manager provider that will be used to obtain the certificate presented to clients if SSL is enabled.
- `trust-manager-provider` – Specifies the trust manager provider that will be used to determine whether to accept any client certificates presented to the server.
- `num-request-handlers` – Specifies the number of threads that should be used to process requests from HTTP clients. These threads are separate from the worker threads used to process other kinds of requests. The default value of zero means the number of threads will be automatically selected based on the number of CPUs available to the JVM.
- `web-application-extension`– Specifies the web applications to be hosted by the server.

For information about other connection handlers, see the *PingDataGovernance Server Configuration Reference Guide*.

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 servers)

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 Ping 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
```

Configuring traffic through a load balancer

If an Ping Identity 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 Ping server. For Ping 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.

On the load balancer, configure settings to provide the `X-Forwarded-*` information, such as `X-Forwarded-Host:`. See the product documentation for the device type.

System alarms, alerts, and gauges

Ping servers provide tools to monitor and manage the health of the system. The Data Governance Server provides delivery mechanisms (handlers) for administrative alerts using JMX or SNMP, in addition to standard error logging. All can be configured with the `dsconfig` tool.

Alerts and alarms reflect state changes within the server that may be of interest to a user or monitoring service. 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 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 gauges for CPU, disk, and memory usage 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.

The server is 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.

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 PingDataGovernance Server provides a number of alert handler implementations configured with the `dsconfig` tool or the Administrative Console, 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 through SNMP 2c.

If needed, the Server SDK can be used to implement additional, third-party alert handlers.

Test alarms and alerts

After gauges, 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.

Perform the following steps to test alarms and alerts:

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/2015 : Alarm [CPU Usage (Percent). Gauge CPU Usage (Percent)
         : 15:41:00    : for Host System Recent CPU and Memory has
         :  -0500      : 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 2015. 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/2015: 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 2015. 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
```

ConsPing Identityume admin alerts health check for PingDirectoryProxy Server

If the PingDataGovernance Server relies on a PingDirectoryProxy Server as a backend store, two health check instances are available to determine if the LDAP store adapter is reporting itself as degraded or unavailable, and if so, remove it from rotation:

- **Consume Admin Alerts**. This health check detects administrative alerts from the server, as soon as they are issued, by maintaining an LDAP persistent search for changes within the `cn=alerts` branch. When the PingDataGovernance Server is notified by the PingDirectoryProxy Server of a new alert, it immediately retrieves the base `cn=monitor` entry of the PingDirectoryProxy Server. If this entry has a value for the `unavailable-alert-type` attribute, then the PingDataGovernance Server will consider it unavailable. If this entry has a value for the `degraded-alert-type` attribute, then the PingDataGovernance Server will consider it degraded.
- **Get Root DSE**. This health check detects if the root DSE entry exists on the LDAP external server. As this entry always exists on an PingDirectoryProxy Server, the absence of the entry suggests that the LDAP external server may be degraded or unavailable.

Make the following configuration changes to the PingDirectoryProxy Server, if it is configured as an external store to the PingDataGovernance Server:

```
$ bin/dsconfig set-gauge-prop --gauge-name "Data Set Local Availability" \
  --set server-unavailable-severity-level:critical
```

```
$ bin/dsconfig set-gauge-prop --gauge-name "Data Set Availability" \
  --set server-unavailable-severity-level:critical
```

These changes modify two gauges to report the PingDirectoryProxy Server as unavailable if the gauges determine that the PingDirectoryProxy Server's data set is unavailable.

Logs and log publishers

Ping supports 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 default log files as well as mechanisms to configure custom log publishers with their own log rotation and retention policies.

Types of log publishers

Log publishers can be used to log processing information about the server, including:

- **Error loggers** – provide information about warnings, errors, or significant events that occur within the server.

- **Trace logger** – provides information about each HTTP, OAuth2, XACML policy, and SCIM request and response that is processed by the PingDataGovernance Server.

View and configure log publishers

Log publishers can be created or modified on each server using the `dsconfig` tool or through the Administrative Console, **Logging, monitoring, and notifications -> Log Publishers**.

Create a new log publisher

Ping provides customization options to create log publishers with the `dsconfig` command or through the Administrative Console.

After creating a new log publisher, configure the log retention and rotation policies. For more information, see Configuring Log Rotation and Configuring Log Retention.

The following example shows how to create a trace logger that collects debug information for HTTP, external identity provider, XACML policy, and store adapter operations with the `dsconfig` command:

```
$ bin/dsconfig create-log-publisher \
  --publisher-name NewTraceLogger \
  --type file-based-trace \
  --set enabled:true \
  --set debug-message-type:external-identity-provider-request-and-response \
  --set debug-message-type:http-full-request-and-response \
  --set debug-message-type:policy-decision-trace \
  --set debug-message-type:store-adapter-processing \
  --set http-message-type:request \
  --set http-message-type:response \
  --set xacml-policy-message-type:result \
  --set 'exclude-path-pattern:/**/*.css' \
  --set 'exclude-path-pattern:/**/*.gif' \
  --set 'exclude-path-pattern:/**/*.jpg' \
  --set 'exclude-path-pattern:/**/*.png' \
  --set log-file:myfile \
  --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 compression-mechanism:gzip
```

Compression cannot be disabled or turned off once configured for the logger. Determine logging requirements before configuring this option.

Configure log compression

Ping 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'Succes
sfully wrote 123,456,789 bytes of decrypted data
```

Configure log signing

Ping 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 Trace Log Publisher.

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

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Trace 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/trace
   ```

   ```
   All signature information in file 'logs/trace' 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/trace':
   * 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

Ping 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 you want the sever 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.
- **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 Error 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 Error Logger" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

Server monitoring

While the server is running, it generates a significant amount of information available through monitor entries. This section contains information about the following:

- Backend monitor entries
- View system and consent data through the PingDataMetrics Server
- Use the status tool

Backend monitor entries

Each Ping server exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers through SNMP, LDAP command-line tools, and the Stats Logger.

The Monitor Backend contains an entry per component or activity being monitored. The list of all monitor entries can be seen using the `ldapsearch` command as follows:

```
$ bin/ldapsearch --hostname server1.example.com \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --baseDN "cn=monitor" "(objectclass=*)" cn
```

The following table lists a subset of monitor entries.

Monitoring components

Component	Description
Active Operations	Provides information about the operations currently being processed by the server including the number of operations, information on each operation, and the number of active persistent searches.
Backends	Provides general information about the state of a server backend, including the entry count. If the backend is a local database, there is a corresponding database environment monitor entry with information on cache usage and on-disk size.
Client Connections	Provides information about all client connections to the server including a name followed by an equal sign and a quoted value, such as `connID="15", connectTime="20100308223038Z"`.
Connection Handlers	Provides information about the available connection handlers on the server including the LDAP and LDIF connection handlers.
Disk Space Usage	Provides information about the disk space available to various components of the server.
General	Provides general information about the state of the server, including product name, vendor name, and server version.
Index	Provides information on each index including the number of preloaded keys and counters for read, write, remove, open-cursor, and read-for-search actions. These counters provide insight into how useful an index is for a given workload.
HTTP/HTTPS Connection Handler Statistics	Provides statistics about the interaction that the associated HTTP connection handler has had with its clients, including the number of connections accepted, average requests per connection, average connection duration, total bytes returned, and average processing time by status code.
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 read and written, and operations initiated, completed, and abandoned.
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 (ms), 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 system host name, operation system, JVM architecture, Java home, and Java version.
Version	Provides information about the server version, including build ID, and revision number.
Work Queue	Provides information about the state of the server work queue, which holds requests until they can be processed by a worker thread, including the requests rejected, current work queue size, number of worker threads, and number of busy worker threads. 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 the log messages. Its value is expressed in milliseconds and represents the length of time that operations are held in the work queue.

View system and consent data through the PingDataMetrics Server

The PingDataMetrics Server contains several charts to measure and monitor PingDataGovernance Server system and user consent activity. Charts and data are configured from the PingDataMetrics Server Server. The following categories can be made available through a PingDataMetrics Server dashboard:

Authorization Requests – Displays the number of blocked and permitted token requests from client applications.

Request Volume – Displays authorization activity according to grant or deny.

Grant Types – Displays the number of authorization grants by type.

Consent/Deny by Application – Displays authorization activity based on client application.

Consent/Deny by Data Type – Displays authorization activity based on data type.

Most Requested Data – Displays most requested data.

Most Active Applications – Displays most active client applications.

Most Active Policies – Displays most active policies.

See the *Ping PingDataMetrics Server Administration Guide* for more information.

Using the status tool

Ping servers provide the `status` tool, which lists 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.

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 the PingDirectory Server, PingDirectoryProxy Server, PingDataMetrics Server, PingDataGovernance Server, and PingDataSync Server 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

Extensions for the PingDataGovernance Server include:

- Policy Information Provider
- Store Adapter

PingDataGovernance Server Advanced Server Configuration

When a PingDataGovernance Server is set up from a peer, its server configuration is cloned to the new PingDataGovernance Server, and the two configurations are linked such that changes to the configuration are applied to both PingDataGovernance Server servers by default. See Installing a Clone PingDataGovernance Server. If a server is installed in an existing topology (an installation option), the server configurations are also linked.

The server's configuration is stored in an LDIF-based backend under the `cn=config base DN`. It can be accessed using the LDAP protocol and is managed by the `dsconfig` tool, Configuration API, or the Administrative Console.

Configuring Third-Party Store Adapters

Third-party adapters can be created for directory servers, that are not the Ping PingDirectory Server, with the Server SDK available in the `unboundid-server-sdk-<version>.zip` package.

Configuring a custom store adapter includes the following steps:

1. Create a store adapter.
2. Store it in the `/extensions` directory of the PingDataGovernance Server.

3. Create a SCIM Resource Type schema.
4. Map Store Adapter(s) and SCIM Resource Types using the Administrative Console or `dsconfig` tool.

Example Third-Party Store Adapter

The Server SDK provides an example implementation of a third-party store adapter. View the example and associated Javadocs in the Server SDK `docs/example-html/ExampleStoreAdapter.java.html` directory.

`ExampleStoreAdapter.java` is an implementation of a flat-file JSON store adapter, which stores the SCIM user data in JSON. At startup, all resources are loaded from the `json-file-path` parameter (`resource/user-database.json`). The example uses an in-memory hash map of SCIM resources mapped to their SCIM ID.

The example provides full operations plus filterable search support for add, update, and deletes. The example will perform a full-file rewrite on every change, because the file format is a serialized list of `Resources<BaseResource>`. The code example does not support sorting or resource versioning.

Cross-Origin Resource Sharing Support

Cross-Origin Resource Sharing (CORS) enables client applications to make JavaScript requests to the PingDataGovernance Server (or PingDirectory Server) by specifying the domain from which the request is made.These cross-domain requests are generally not allowed by web browsers without CORS support. CORS defines a way in which the browser and the server can interact to determine whether a request is coming from a trusted domain.

CORS Implementation

CORS is implemented per HTTP servlet extension. Access is governed by HTTP Servlet Cross Origin Policies defined through the `dsconfig` tool. Trusted domains can be added to these policies or defined with registered applications in the Administrative Console or with the `dsconfig` tool.

Note

By default, HTTP servlet extensions do not have CORS defined. Without a CORS policy defined, the configuration of the browser will determine application access.

The following are configuration options in `dsconfig`:

```
>>>> HTTP Servlet Cross Origin Policy menu

What would you like to do?

1)  List existing HTTP Servlet Cross Origin Policies
2)  Create a new HTTP Servlet Cross Origin Policy
3)  View and edit an existing HTTP Servlet Cross Origin Policy
4)  Delete an existing HTTP Servlet Cross Origin Policy

b)  back
```

```
q)  quit

Enter option [b]:
```

HTTP servlet services

Enabling CORS for a particular servlet can impact another service provided by the same servlet. It is important to know which services will be affected when enabling CORS for an PingDataGovernance Server servlet. The following are available servlets and their functions.

Servlet	Functions
API Explorer Servlet	Manages requests to the API Explorer, which enables testing PingDataGovernance Server functions.
Authentication Servlet	Manages requests to the `/authentication` API endpoint (used by the `auth-ui`).
Configuration	Used to enable read and write access to the server's Configuration API.
Documentation	Manages requests for the `/docs` content, which includes the `index.html` page, the generated Configuration Reference Guide, and other product documents.
JWK Servlet	Provides access to the JSON Web Key for token validation.
OAuth2 Servlet	OAuth2 authorization, token, revocation, and validation endpoints.
Policy Decision Point Servlet	XACML PDP endpoint.
SCIM2	Profile access by SCIM Resource Type using SCIM.
UserInfo Servlet	Profile access using OpenID Connect.

Note

Any servlet accepting JavaScript calls from client applications that are hosted at a different location than that of the Data Governance Server APIs, such as the Velocity servlet, must have CORS enabled.

HTTP servlet cross origin policies

Two sample policies are available after installation. They can be associated with a servlet extension, or used as templates for additional policies.

Per-Application Origins – This policy trusts origins that are listed as trusted by applications registered with the PingDataGovernance Server.

Restrictive – This policy rejects all cross-origin requests unless explicitly defined with the `cors-allowed-origins` property. Requests from application origins that are not specified are rejected with a `403 Forbidden` return code.

Each policy accepts values for the following properties.

Property	Description
`cors-enabled`	Specifies if the CORS protocol is allowed by the servlet. The default value is `false`.
`cors-allowed-methods`	Specifies the list of HTTP methods allowed for access to resources. The

Property	Description
	default value is GET.
cors-enable-per-application-origins	Specifies that a per-application list of allowed origins is consulted. The default value is false in the Restrictive policy and true in the Per-Application Origins policy.
cors-allowed-origins	Specifies a global list of allowed origins. If the cors-enable-per-application-origins property is set to true, and there are origins listed here, this list is consulted in addition to the per-application list. A value of "*" specifies that all origins are allowed. The default is an empty list.
cors-exposed-headers	Specifies a list of HTTP headers that browsers are allowed to access. Simple response headers, as defined in the Cross-Origin Resource Sharing Specification, are allowed. The default is an empty list.
cors-allowed-headers	Specifies the list of header field names that are supported for a resource and can be specified in a cross-origin request. The default values are Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-Method, and Access-Control-Request-Headers.
cors-preflight-max-age	Specifies the maximum number of seconds that a preflight request can be cached by the client. The default value is 1800 (30 minutes).
cors-allow-credentials	Specifies whether requests that include credentials are allowed. This value should be false for servlets that use OAuth2 authorization. The default value is false.

Assigning a CORS policy to an HTTP servlet extension

CORS policies are assigned to HTTP servlet extensions through dsconfig.

The following are configuration options for the SCIM servlet extension:

```
>>>> Configure the properties of the SCIM Resource Type SCIM HTTP Servlet Extension
Property                Value(s)
----------------------------------------------------------------------

1)  description          -
2)  cross-origin-policy  No cross-origin policy is defined and no CORS headers are
recognized or returned.
3)  base-context-path    /scim

?)  help
f)  finish - apply any changes to the SCIM Resource Type SCIM HTTP Servlet Extension
a)  show advanced properties of the SCIM Resource Type SCIM HTTP Servlet Extension
d)  display the equivalent dsconfig command lines to either re-create this object or only
to apply pending changes
b)  back
q)  quit

Enter option [b]: 2
```

Choose the cross-origin-policy option. Defined policies are listed.

```
>>>> Configuring the 'cross-origin-policy' property
The cross-origin request policy to use for the HTTP Servlet Extension.

A cross-origin policy is a group of attributes defining the level of cross-origin request
supported by the HTTP Servlet Extension.

Do you want to modify the 'cross-origin-policy' property?

1)  Keep the default behavior: No cross-origin policy is defined and no CORS headers are
recognized or returned.
2)  Change it to the HTTP Servlet Cross Origin Policy: Per-Application Origins
3)  Change it to the HTTP Servlet Cross Origin Policy: Restrictive
4)  Create a new HTTP Servlet Cross Origin Policy

?)  help
q)  quit
```

Choose the CORS policy to assign to this servlet extension.

Public and private key store configuration

The PingDataGovernance Server server can be configured to validate access tokens with a private key and expose a public key to enable client applications to read the content of the tokens. If there are multiple PingDataGovernance Servers in an environment, a key-pair created on one server will automatically be mirrored on all other servers. The PingDataGovernance Server supports RSA key pairs.

A certificate key pair can be created by or imported to the server with the `dsconfig` tool, or through the advanced setting **System -> Key Pairs** in the Administrative Console. For example, the following command can be used to create a new key pair:

```
$ bin/dsconfig -n create-key-pair --pair-name jwt2
```

When a key-pair is created or imported, the private key is encrypted by the preferred encryption settings definition in the encryption settings database and a Certificate Signing Request attribute is created. The private key and Certificate Signing Request are read-only properties, but not the certificate chain. The public key is wrapped in the certificate chain.

The Certificate Signing Request can be taken to a Certificate Signing Authority to obtain a signed, public key certificate. This can then be imported with `dsconfig` to replace the self-signed certificate.

Note

The PingDataGovernance Server does not automatically rotate expired keys. If using self-signed certificates, reset the `certificate-chain` property when needed. This will regenerate a new self-signed certificate with the specified validity (`self-signed-certificate-validity`). If using signed certificates, renew the certificate (extend its validity) from the Certificate Signing Authority and set the `certificate-chain` property in the key-pair.

Long keys may require more CPU for processing and affect performance, if request volume is high.

Managing server encryption settings

The server encryption settings database is managed by the `encryption-settings` command-line tool. The keys stored for the server are used to provide support for validating signed JWTs (JWS) and encrypted JWTs (JWE), and to advertise the server's Java Web Key Set (JWKS) through the `/jwk` endpoint. Help and examples are available with the following command:

```
$ bin/encryption-settings --help
```

Information about the cipher algorithms and transformations available for use is located in the *Java Cryptography Architecture Reference Guide* and *Standard Algorithm Name Documentation* available on the Oracle website.

Rotating the encryption key

Perform the following steps for routine rotation of the encryption key:

1. Create a new encryption settings definition.

   ```
   $ encryption-settings create \
     --cipher-algorithm AES \
     --key-length-bits 128
   ```

   ```
   Successfully created a new encryption settings definition with ID <ID>
   ```

2. Verify the new definition was created.

   ```
   $ encryption-settings list
   Encryption Settings Definition ID: <old-key>
     Preferred for New Encryption: true
     Cipher Transformation: AES
     Key Length (bits): 128

   Encryption Settings Definition ID: <ID>
     Preferred for New Encryption: false
     Cipher Transformation: AES
     Key Length (bits): 128
   ```

3. Create a PIN file that will be used for the exported definition.

   ```
   $ echo "secret" > /tmp/exported-key.pin
   ```

4. Export the encrypt settings, referring to the generated encryption settings ID.

   ```
   $ encryption-settings export \
     --id <ID> \
     --output-file /tmp/exported-key \
     --pin-file /tmp/exported-key.pin

   Successfully exported encryption settings definition <ID> to file
   /tmp/exported-key
   ```

5. For every Data Governance Server instance in the topology, copy the exported definition and PIN file to the PingDataGovernance Server's host. Import the encryption settings,

without setting them as preferred. Delete the exported settings and PIN file when finished.

```
$ encryption-settings import \
  --input-file /tmp/exported-key \
  --pin-file /tmp/exported-key.pin

Successfully imported encryption settings definition <ID> from file
/tmp/exported-key
```

```
$ rm /tmp/exported-key
$ rm /tmp/exported-key.pin
```

6. Perform the previous steps for all existing key pairs, as private keys will still be encrypted with the previous preferred encryption definition. Delete the existing key pairs and re-import them (which will automatically use the new preferred encryption definition for the private key).
7. After importing the encryption settings definition to all PingDataGovernance Servers, including the instance where the definition was originally created, set the new definition as preferred.

```
$ encryption-settings set-preferred \
  --id <ID>

Encryption settings definition <ID> is was successfully set as the
preferred definition for subsequent encryption operations.
```

Address a compromised encryption key

If an encryption settings definition becomes compromised, perform the following to create a new definition and update the PingDataGovernance Server servers. See the command line help for the `encryption-settings` tool for arguments.

Note

If the PingDataGovernance Server's encryption key is compromised, and the PingDataGovernance Server has been collecting access tokens for external identity providers through the relying party feature, make sure those tokens are revoked.

1. Back up the encryption settings backend.
2. Back up the user store.
3. Revoke all authorizations for each client.
4. Stop the HTTPS Connection Handler that is used for the Data Governance Server's REST APIs.

```
$ dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --set enabled:false
```

5. Create a new encryption settings definition and set it as preferred. The following will encrypt data using a 128-bit AES cipher:

```
$ encryption-settings create \
  --cipher-algorithm AES \
  --key-length-bits 128
  --set-preferred
```

6. Restart the HTTPS Connection Handler.

```
$ dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --set enabled:true
```

If the deployment includes multiple PingDataGovernance Servers, all servers should be taken offline, and the encryption settings database must be updated on every server.

Note

Do not delete the compromised encryption definition. It will still be used to decrypt tokens, authorization codes, and links that were encrypted with the previous key.

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.

Servers can be added to an existing topology at installation. See Add an additional PingDataGovernance Server to a topology for details.

Note

To remove a server from the topology, it must be uninstalled with the uninstall tool. See Uninstalling thePingDataGovernance Server for details.

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 be 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 configuration. 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-----

MIIDKTCCAhGgAwIBAgIEacgGrDANBgkqhkiG9w0BAQsFADBFMR4wHAYDVQQKExVVbmJvdW5kSUQgQ2VydGlmaWNhd
GUxIzAhBgNVBAMTGnZtLW1lZGl1bS03My51bmJvdW5kaWQubGFiMB4XDTE1MTAxMjE1MzU0OFoXDTM1MTAwNzE1Mz
U0OFowRTEeMBwGA1UEChMVVW5ib3VuZElEIENlcnRpZmljYXRlMSMwIQYDVQQDExp2bS1tZWRpdW0tNzMudW5ib3V
uZGlkLmxhYjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKN4tAN3o9Yw6Cr9hivwVDxJqF6+aEi9Ir3W
GFYLSrggRNXsiAOfWkSMWdIC5vyF5OJ9DlIgvHL4OuqP/YNEGzKDkgr6MwtUeVSK14+dCixygJGC0nY7k+f0WSCjt
IHzrmc4WWdrZXmgb+qv9LupS30JG0FXtcbGkYpjaKXIEqMg4ekz3B5cAvE0SQUFyXEdN4rWOn96nVFkb2CstbiPzA
gne2tu7paJ6SGFOW0UF7v018XY1m2WHBIoD0WC8nOVLTG9zFUavaOxtlt1TlhClkI4HRMNg8n2EtSTdQRizKuw9Dd
TXJBb6Kfvnp/nI73VHRyt47wUVueehEDfLtDP8pMCAwEAAaMhMB8wHQYDVR0OBBYEFMrwjWxl2K+yd9+Y65oKn0g5
jITgMA0GCSqGSIb3DQEBCwUAA4IBAQBpsBYodblUGew+HewqtO2i8Wt+vAbt31zM5/kRvo6/+iPEASTvZdCzIBcgl
etxKGKeCQ0GPeHr42+erakiwmGDlUTYrU3LU5pTGTDLuR2IllTT5xlEhCWJGWipW4q3Pl3cX/9m2ffY/JLYDfTJao
JvnXrh7Sg719skkHjWZQgOHXlkPLx5TxFGhAovE1D4qLVRWGohdpWDrIgFh0DVfoyAn1Ws9ICCXdRayajFI4Lc6K1
m6SA5+25Y9nno8BhVPf4q5OW6+UDc8MsLbBsxpwvR6RJ5cv3ypfOriTehJsG+9ZDo7YeqVsTVGwAlW3PiSd9bYP/8
yu9Cy+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"
```

Index

A

access token 42
- accepting all tokens 43
- accepting external tokens 42
- accepting PingFederate tokens 42

access token properties 59

administrative account
- adding a root user account 67

Administrative Console
- login account 18
- URL 19

administrative password 67

alerts
- Directory Proxy availability 87

attribute mappings 32
- authoritative attribute 32, 38
- complex attributes 38
- described 29
- indexing 37
- mapping in SCIM Resource Types 37

authoritative attribute 32

B

backend monitors
- entries 92

backup tool 65

base DN
- configure Directory Server 14
- configure user entries 18

base64 tool 65

C

client-specific SCIM attributes 38

collect-support-data tool 65

config-diff tool 65

Config File Handler Backend 103

correlation attribute 30

CORS
- configuration 96

create-initial-config 17

create-rc-script tool 65

D

Data Governance Server
- attribute filtering 2-3
- deployment considerations 3
- described 1
- features 2
- in a topology 19
- installing 16
- pluggable authentication 2
- tools 65

Directory Server
- described 29
- installing 13

DNS caching 82

dsconfig
- changing policy-combining algorithm 48
- CORS configuration 96
- described 66
- tool described 65

dsframework tool 65

dsjavaproperties tool 65

dstat
  installing on SuSE Linux 11

E

encryption-settings tool 12, 65
encryption keys 12, 101
endpoints
  SCIM 40
error logger 87

F

filter obligation 51

G

Global Configuration object 103

H

health check
  consume admin alerts 87
  get root DSE 87
HTTP Servlet Cross Origin Policy 97
HTTP servlet extension 98

I

installing
  prerequisites 9
  scripted install 21
inter-server-certificate property 107
IP address reverse name lookup 83

J

Java
  installing the JDK 12
  supported versions 9
JEXL
  identifiers and variables 49
  policy structure details 48
JSON 32
JVM memory allocation
  Data Governance Server 17
  Directory Server 15
JWT 42
JWT token validation 43

K

key pair configuration 99

L

ldapmodify tool 65
ldappasswordmodify tool 65
LDAPS
  configure Data Governance Server 16
  configure Directory Server 14
ldapsearch tool 65
ldif-diff tool 66
ldifmodify tool 66
license key 12
Linux configuration
  filesystem swapping 10
  set file descriptor limit 9
list-backends tool 66
load balancers 84
logging
  available log publishers 87
  configure log retention and rotation 91
  configure log signing 90
  create log publisher 88
  encrypt log files 89
  log compression 88
login account 18

M

manage-certificates tool 106
manage-extension tool 66

metrics
  viewing 94
monitoring entries 93

O

oauth2-request tool 66
OAuth2 clients
  enable client-specific SCIM attributes 38
Open Banking 2

P

PDP endpoint 55
PingFederate access token validator 42
policy
  authorization scenarios 48
  decision trace 62
  described 44
  managing 61
  obligations and advice 51
  PDP endpoint 55
  policy information providers 60
  policy structure 48
  request processing 52
  troubleshoot denied access 63
  viewing policy metrics 94
  XACML 47
policy set
  creating 62
prepare-external-store tool 18, 66

R

remove-defunct-server tool 27, 66
resource operations
  scope properties 45, 58
Resource Scope 46
REST API
  connection port 16
  endpoints 42
restore tool 66
review-licence tool 66
root user DN 18-19

S

SCIM
  DELETE operations 54
  described 40
  GET operations 53
  PATCH and PUT operations 54
  policy request with obligation 51
  POST operations 53
  search request 52
SCIM endpoint 40
  described 29
  search considerations 32
SCIM request properties 58
SCIM resource properties 57
SCIM Resource Type
  creating 34
  managing 33
  REST API endpoints 42
  store adapter mapping 37
SCIM schema
  overview 32
  username uniqueness 33
scopes
  applicable scopes returned 59
  Authenticated Identity scope 47
  managing 47
  Resource scope 46-47

scope types and properties 45

self-signed certificate

replacing 106

server shutdown 83

SSL certificate 105

start-server 66

running in the foreground 23

status tool 66, 94

stop-server 66

example of 23

in-core restart 24

store adapter

correlation attribute 30

mapping attributes 32

primary and secondary adapters 29

search considerations 32

third-party store adapters 95

store adapters

third-party 96

sum-file-sizes tool 66

supported platforms 9

system entropy 11

T

Third-Party Store Adapter 96

token validation

signature validation 43

topology

force master setting 103

inter-server-certificate property 107

master selection 102

monitor data 104

overview 102

replace self-signed certificate 107

server configuration settings 103

subtree polling interval 102

update servers 25

update SSL certificate 105

topology configuration

update SSL Certificate 105

trace logger 88

trace policy decisions 62

troubleshooting

server shutdown 83

SSL 83

U

uid 33

uninstall tool 27

update tool 25

user processes

configuring on Redhat/CentOS 10

user store 12-13

W

Windows service

configuration 24

deregister and uninstall 25

log files 25

X

X-Forwarded values 84

PingDataGovernance™

Release 7.0

Server Administration Guide

PingDataGovernance Server™ Product Documentation

Table of Contents

Chapter 1: Introduction **1**

PingDataGovernance Server overview 2

PingDataGovernance Server features 2

PingDataGovernance Server deployment considerations 3

Configuration overview 3

SCIM 4

Data Sources 4

Scopes and Policies 4

System 5

Web Services and Applications 6

LDAP Administration and Monitoring 6

Logging, Monitoring, and Notifications 6

Chapter 2: Installation **8**

Installation prerequisites 9

Supported platforms 9

Set the file descriptor limit 9

Set the maximum user processes 10

Disable filesystem swapping 10

Install the dstat utility on SuSE Linux 11

Managing system entropy 11

Enable the server to listen on privileged ports 11

Install the JDK 12

Encryption keys 12

User store overview 12

Ping license keys 12

Install the PingDirectory Server 13

PingDataGovernance Server Installation Tools 15

Install the PingDataGovernance Server 15

Configure the PingDataGovernance Server 17

Log into the Administrative Console 19

Install an additional PingDataGovernance Server in a topology 19

Server folders and files 20

Plan a scripted installation 21

Start the PingDataGovernance Server 23
Stop the PingDataGovernance Server 23
Schedule a server shutdown 23
Run an in-core restart 24
Run the server as a Microsoft Windows service 24
Register the service 24
Run multiple service instances 24
Deregister and uninstall 25
Log files 25
Update servers in a topology 25
Update the server 26
Reverting an Update 26
Uninstall the PingDataGovernance Server 27
Chapter 3: Data access and mapping 28
Data components 29
Store adapter mappings 29
Directory Servers 29
Primary and secondary store adapters 29
Defining correlation attributes 30
Sample configuration 30
SCIM schemas 32
Store adapter mappings 32
SCIM attribute search considerations 32
Maintain username uniqueness 33
Define SCIM Resource Types 33
Pass-through SCIM Resource Type 33
Mapping SCIM Resource Type 34
Create a SCIM Resource Type 34
Create a Mapping SCIM Resource Type 34
Create a Pass Through SCIM Resource Type 35
Edit attribute and sub-attribute properties 35
Edit store adapter mappings 37
Complex attribute mapping 38

Client-specific SCIM attributes 38
Access data 40
Chapter 4: Token access 41
PingDataGovernance Server endpoint for OAuth2 clients 42
Access token validation 42
PingFederate Access Token Validator 42
JWT token validation 43
Chapter 5: Configure scopes and policies 44
OAuth2 scopes 45
Authenticated Identity Scope 45
Resource Scope 46
Create scopes 47
Create an Authenticated Identity OAuth2 scope 47
Create a Resource OAuth2 scope 47
Policy overview 47
Requesting operations through SCIM 48
Policy structure 48
JEXL use in policy structure 49
JEXL identifiers and variables 49
References containing invalid JEXL characters 49
Extended data type support 50
JEXL extension functions 50
Use obligations and advice 51
SCIM resource requests 51
Filter obligation 51
Policies and request processing 52
SCIM resource type policy evaluation 52
SCIM search request 52
SCIM GET request 53
SCIM POST request 53
SCIM PATCH and PUT requests 54
SCIM DELETE request 54
Policy Decision Point (PDP) endpoint 55
Policy engine request context 55
XACML attribute categories 55

Standard attribute use 56
Custom XACML function 57
SCIM resource properties 57
Scope properties 57
SCIM request properties 58
Applicable scopes 59
Access token properties 59
Configure the Policy Service 60
Policy Information Providers 60
PIP Evaluation Order 60
Create policies 61
Creating a policy set 62
Troubleshoot policies with traces 62
Troubleshoot denied access 63
Chapter 6: Advanced configuration **64**
General server configuration 65
Available configuration tools 65
Use the dsconfig tool 66
Administrative accounts 67
Change the administrative password 67
Use the Configuration API 67
Authentication and authorization 68
Relationship between the Configuration API and the dsconfig tool 68
GET example 69
GET list example 71
PATCH example 72
API paths 76
Sorting and filtering configuration objects 77
Update Properties 77
Administrative actions 79
Updating servers and server groups 80
Configuration API responses 80
Configuring HTTP connection handlers 81

Domain Name Service (DNS) caching 82
IP address reverse name lookups 83
Problems with SSL communication 83
Conditions for automatic server shutdown 83
Configuring traffic through a load balancer 84
System alarms, alerts, and gauges 84
Alert handlers 85
Test alarms and alerts 86
ConsPing Identityume admin alerts health check for PingDirectoryProxy Server 87
Logs and log publishers 87
Types of log publishers 87
View and configure log publishers 88
Create a new log publisher 88
Configure log compression 88
Configure log file encryption 89
Configure log signing 90
Configure log retention and log rotation policies 91
Configure the Log Rotation Policy 92
Configure the Log Retention Policy 92
Server monitoring 92
Backend monitor entries 92
View system and consent data through the PingDataMetrics Server 94
Using the status tool 94
Server SDK extensions 94
PingDataGovernance Server Advanced Server Configuration 95
Configuring Third-Party Store Adapters 95
Example Third-Party Store Adapter 96
Cross-Origin Resource Sharing Support 96
CORS Implementation 96
HTTP servlet services 97
HTTP servlet cross origin policies 97
Assigning a CORS policy to an HTTP servlet extension 98
Public and private key store configuration 99
Managing server encryption settings 100
Rotating the encryption key 100

-

Address a compromised encryption key ... 101
Customizing the Authentication User Interface ... 102
Branding ... 103
Schema changes ... 103
Topology configuration ... 104
Topology master requirements and selection ... 104
Topology components ... 105
Server configuration settings ... 105
Topology settings ... 106
Monitor data for the topology ... 106
Updating the server instance listener certificate ... 107
Remove the self-signed certificate ... 108
Prepare a new keystore with the replacement key-pair ... 108
Use an existing key-pair ... 108
Use the certificate associated with the original key-pair ... 109
Update the server configuration to use the new certificate ... 110
Update the ads-truststore file to use the new key-pair ... 110
Retire the old certificate ... 110
Index ... 111

Chapter 1: Introduction

The PingDataGovernance Server provides solutions to manage and monitor user data and access to account resources.

Topics include:

PingDataGovernance Server overview

PingDataGovernance Server features

PingDataGovernance Server deployment considerations

Configuration overview

PingDataGovernance Server overview

Most organizations today are working toward creating a unified customer profile. An essential part of creating that common profile is to centralize multiple, overlapping accounts and define the logic and security criteria for determining which applications should access data in a profile. The PingDataGovernance Server enables managing large amounts of customer data while ensuring end-user privacy.

The PingDataGovernance Server provides a common identity and single view of the customer by mapping account resources from multiple backend PingDirectory Servers to SCIM Resource Types defined in the PingDataGovernance Server. Restricted access to end users' information is maintained through policy rules .

PingDataGovernance Server features

The PingDataGovernance Server provides the following features to securely manage account resources:

- **Support for multiple backend Servers**. The PingDataGovernance Server supports multiple directory servers, with native support for the Ping Identity PingDirectory Server and extension points for others. PingDirectory Servers serve as user stores to provide the resources that can be requested by clients. Clients can be written one time for access to the PingDataGovernance Server and receive data from any type of infrastructure backend.
- **Support for Open Banking**. The PingDataGovernance Server supports a subset of Open Banking APIs. A working Open Banking solution requires a PingDataGovernance Server, a PingDirectory Server, and a PingFederate instance. These products working together provide an implementation for an Open Banking Account Servicing Payment Service Provider (ASPSP), which is the role played by a bank or other entity providing payment accounts.
- **Support for General Data Protection Regulation (GDPR)**. GDPR 2016/679 is a European Union (EU) regulation on data protection and privacy for all individuals within the EU, and businesses outside of the EU that handle EU individual's data. The GDPR aims primarily to give control to citizens and residents over their personal data through the use and management of consents. PingDataGovernance Server can be used with the PingDirectoryProxy Server's Consent Service to meet the security, tracking, and auditing requirements of GDPR. See the *PingDirectory Server Consent Guide* for details about configuring the Consent Service.
- **SCIM Resource Types**. SCIM Resource Types determine what attributes can be accessed by a client through the PingDataGovernance Server. The SCIM resource type

defines the resource name, endpoint URL, schemas, and other metadata that indicate where a resource is managed and how it is composed.

- **Access to resources based on policy**. The PingDataGovernance Server ensures that data is provided to authorized clients through the use of defined OAuth2 Scopes and policies. JEXL is used to define access control policies, and the processing model that determines how to evaluate requests based on rules defined in the policies. Policies can be based on industry rules, corporate policy, or consent granted by customers.
- **Application Developer Resources**. The enables client application developers to work with the PingDataGovernance Server APIs to design applications that can access PingDataGovernance Server resources. See the Ping Identity API site for configuration examples (`https://www.pingidentity.com/content/developer/en/explore.html`).

PingDataGovernance Server deployment considerations

The PingDataGovernance Server accepts client requests to access user data. Clients are granted authorization through an identity provider and receive access through the PingDataGovernance Server SCIM endpoint. The PingDataGovernance Server validates an OAuth2 access token request, where the scopes requested represent resources in backend servers.

Planning a PingDataGovernance Server deployment should start with defining what data can be accessed and updated from backend PingDirectory Servers, which are configured as User Stores. User Stores have a schema defined to surface attributes. SCIM Resource Types are then defined to enable access to attributes, and provide a unified view of identity data found in multiple PingDirectory Servers through Store Adapter Mappings. OAuth2 scopes are created to define the resources that can be requested by a client and the actions that can be performed on those resources.

Policies determine if a client can access requested scopes, based on the information provided with the request. Obligations within the policy can define conditions for access. Policies then determine the operations that can be performed on attributes within the requested scopes.

Configuration overview

PingDataGovernance Server configuration defines all server services, policies, applications, resources, and the mapping of data from one or more backend PingDirectory Servers. Configuration can be done from the command line with the `dsconfig` tool or through the Administrative Console interface. All settings have associated help text in the interface and in the linked Configuration Guide. The Configuration Guide contains details and relationship specifics for all configuration objects and is available from the Administrative Console interface or from the `<server-root>/docs/index.html` page.

SCIM

The SCIM protocol is an application-level, REST protocol for provisioning and managing identity data. The SCIM Schema provides a schema and extension for representing users and groups. Only those attributes defined in the SCIM Resource Type can be accessed through the PingDataGovernance Server. Any changes to these settings are saved to all PingDataGovernance Servers in a topology.

- **SCIM Resource Types** – Defines attribute mapping from a SCIM schema to native attributes found in PingDirectory Server entries. A pass-through SCIM Resource Type can also be created to allow the addition of new attributes that are not mapped to any in a PingDirectory Server. The SCIM schema defines the attributes that comprise a SCIM Resource Type. The SCIM Resource Type determines the attributes that can be accessed by a client application.
- **SCIM Schemas** – Specifies the SCIM 2.0 schemas for data that can be accessed from backend PingDirectory Servers. Schemas provide the basis for creating SCIM Resource Types.

Data Sources

Data sources are the servers that house the resources governed by the PingDataGovernance Server.

- **External Servers** – Lists the LDAP PingDirectory Server instances that are configured with the PingDataGovernance Server.
- **LDAP Health Checks** – Checks the status of external LDAP servers on a regular basis, and examines failures to determine if the server has become unavailable. This is an advanced setting.
- **Load Balancing Algorithms** – Used to determine the appropriate LDAP external server to use to process a request. They may be used to provide improved availability and performance by distributing the workload across multiple backend servers. This is an advanced setting.
- **Store Adapters** – Provides a PingDirectory Server interface to the PingDataGovernance Server. Changes or additions to Store Adapters are saved to all PingDataGovernance Servers in a topology. Third-party store adapters can be created with the Server SDK.

Scopes and Policies

These settings define the rules for accessing resources through the PingDataGovernance Server. Any changes to these settings are saved to all PingDataGovernance Servers in a topology.

- **Access Token Validators** – Validates an access token used to access protected resources (OAuth2 scopes). Validators are used to decode tokens and return token metadata. The PingDataGovernance Server's local access token validator can be used, or a third-party token validator can be defined using the Server SDK.
- **OAuth Scopes** – Specifies the data being requested from a client.
- **Policy Information Providers** – Retrieves XACML attributes from a Policy Information Point (PIP) for policy evaluation. This is an advanced setting.
- **Policies** – Specifies the rules for how requested resources can be shared with clients, written with JEXL (Java Expression Language) and based on the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0*. The PingDataGovernance Server provides default policies that can be used or modified.
- **Policy Service** – Contains the properties that affect the overall operation of the PingDataGovernance Server Policy Decision Point (PDP).

System

System settings define communication, connection, and the criteria for triggering alarms regarding the server's resources. Changes to these setting can be saved to the local server or saved to a group of servers. Most are not mirrored across a topology, unless otherwise stated. See General Server Configuration for more information.

- **Connection Handlers** – Defines the settings for handling all interaction with the clients, including accepting connections, reading requests, and sending responses.
- **Global Configuration** – Specifies the SMTP server, password policies, and LDAP request criteria configured for this server.
- **Key Manager Providers** – Manages the key material used to authenticate to another server. This is an advanced setting.
- **Key Pairs** – Defines the key pair that can be used to provide credentials for digital signatures. An existing key pair can be imported or a new one can be generated by the server. This configuration object is mirrored across a topology.
- **Locations** – Lists the locations in which servers that are accessed by the PingDataGovernance Server reside.
- **Trust Manager Providers** – Determine whether to trust certificates presented to the server. This is an advanced setting.
- **Trusted Certificates** – Specifies a trusted public key that can be used to verify credentials for digital signatures and public-key encryption. This configuration object is mirrored across a topology.

Web Services and Applications

These settings define the HTTP connection criteria for application access to the PingDataGovernance Server. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

- **HTTP Configuration** – Defines configuration for the PingDataGovernance Server HTTP Service. This is an advance setting and cannot be changed other than to include stack traces in error pages.
- **HTTP Servlet Cross Origin Policies** – Defines the configuration for handling Cross-Origin HTTP requests using the Cross Origin Resource Sharing (CORS) protocol. An instance of HTTP Servlet Cross Origin Policy can be associated with multiple HTTP Servlet Extensions.
- **HTTP Servlet Extensions** – Defines classes and initialization parameters used by a servlet invoked by an HTTP connection handler.
- **Web Application Extensions** – Specifies the configuration settings for the Administrative Console and any other web applications that are configured to work with the PingDataGovernance Server.

LDAP Administration and Monitoring

These are all advanced settings to manage the local server's accounts, account requirements and security settings, and backend configuration. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

Logging, Monitoring, and Notifications

These settings define the notification criteria for system alerts, and the logging criteria for actions within the PingDataGovernance Server. Changes to these setting can be saved to the local server or saved to a group of servers. They are not mirrored across a topology. See General Server Configuration for more information.

- **Alarm Manager** – Defines the severity of alarms to be raised.
- **Alert Handlers** – Specifies the Alert Handlers used to notify administrators of problems or events that occur in the PingDataGovernance Server.
- **Gauges** – Specifies server performance thresholds and circumstances that merit the raising of an alarm.
- **Gauge Data Sources** – Defines the source of gauge data obtained from the server, including available memory and disk space.

- **LDAP SDK Debug Logger** – Records debug messages generated by the LDAP SDK for Java. This is an advanced setting.
- **Log File Rotation Listeners** – Defines an action for the server to take before a log file is rotated out of service, such as copying the file to a new location. This is an advanced setting.
- **Log Publishers** – Defines the distribution of log messages from different loggers to a destination.
- **Log Retention Policies** – Defines how long logs should be kept.
- **Log Rotation Policies** – Specifies when log files should be rotated.
- **Monitor Providers** – Provides information about the state of the server or server components.

Chapter 2: Installation

The PingDataGovernance Server installation requires few prerequisites, and can be deployed on virtualized and/or commodity hardware.

Topics include:

Installation prerequisites

Encryption keys

User store overview

Ping license keys

Install the PingDirectory Server

PingDataGovernance Server installation tools

Install the PingDataGovernance Server

Configure the PingDataGovernance Server

Log into the Administrative Console

Install an additional PingDataGovernance Server in a topology

Server folders and files

Plan a scripted installation

Start the PingDataGovernance Server

Stop the PingDataGovernance Server

Run the server as a Microsoft Windows Service

Update servers in a topology

Uninstall the PingDataGovernance Server

Installation prerequisites

The following are required before installing the PingDataGovernance Server:

- Java 8
- Minimum of 2 GB RAM
- Ping PingDirectory Server 6.0, or later

Supported platforms

The PingDataGovernance 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 an 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.

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. The file descriptor limit on the operating system can be increased 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 server, 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 maximum user processes

Redhat Enterprise Linux Server/CentOS 6.x sets the default maximum number of user processes to 1024, which is lower than the setting on older distributions. This may cause JVM memory errors when running multiple servers on a machine because each Linux thread is counted as a user process.

At startup, the PingDataGovernance Server attempts to raise this limit to 16,383 if the value reported by `ulimit` is less. If the value cannot be set, an error message is displayed. Explicitly set the limit in `/etc/security/ limit.conf`. For example:

```
* soft nproc 100000
* hard nproc 100000
```

The 16,383 value can also be set in the `NUM_USER_PROCESSES` environment variable, or by setting the same variable in `config/num-user-processes`.

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`.

Install the dstat utility on SuSE Linux

The `dstat` utility is used by the `collect-support-data` tool to gather support data. It can be obtained from the OpenSuSE project website. Perform the following steps to install the `dstat` utility:

1. Log into the server as root.
2. Add the appropriate repository using the `zypper` tool.
3. Install the `dstat` utility:

   ```
   $ zypper install dstat
   ```

Managing 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 systems provide '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
```

Install the JDK

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

Encryption keys

Encryption setting definitions are used to validate encrypted access tokens. All PingDataGovernance Server instances must use the same set of definitions. Encryption setting definitions are managed using the `encryption-settings` tool.

If new encryption settings must be defined, the new definition must be exported using the `encryption-settings` tool and imported on all PingDataGovernance Server instances. Only after the new definition is imported on all servers can the new definition be used for subsequent encryption operations.

See Managing the Server Encryption Settings for more information.

User store overview

During the PingDataGovernance Server installation, at least one Ping PingDirectory Server is defined to serve as a user store. The user store is a repository of user data, such as names, email addresses, and preferences, as well as user-specific metadata needed by the PingDataGovernance Server. For example, some user data may be stored in an LDAP directory server while other attributes may be stored in a relational database. SCIM Resource Types are defined to enable access to a user store's resources, and provide a consistent view of a user's profile that may cross multiple PingDirectory Servers.

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 Admin 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 PingDirectory Server

The PingDataGovernance Server requires at least one installed Ping Identity PingDirectory Server. The PingDataGovernance Server can be configured with multiple user stores.

Note

All sensitive data in the user store can be encrypted. When using the Ping Identity PingDirectory Server as the user store, server-level encryption can be enabled as described in the "Encrypting Sensitive Data" section in the *Ping PingDirectory Server Administration Guide*.

The following information is needed during the installation:

- Server hostname
- LDAPS port
- Root DN and password
- Base DN
- Location of user entries

All configuration settings can be later modified through the `dsconfig` tool.

Perform the following steps to install the PingDirectory Server:

1. Download the PingDirectory Server zip distribution, `PingDirectory-<version>.zip`.
2. Unzip the file in any location.

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

3. Change to the top level PingDirectory folder.

   ```
   $ cd PingDirectory
   ```

4. Run the `setup` command.

   ```
   $ ./setup
   ```

5. Enter **yes** to agree to the license terms.

6. Enter the fully qualified host name or IP address of the local host, or press **Enter** to accept the default.
7. Create the initial root user DN for the PingDirectory Server, or accept the default, (`cn=Directory Manager`). This account has full access privileges.
8. Enter a password for this account, and confirm it.
9. To enable the Platform APIs over HTTPS, enter **yes**. These are the product's configuration APIs.
10. Enter the port to accept connections from HTTPS clients, or press **Enter** to accept the default. The default may be different depending on the account privileges of the user installing.
11. Enter the port to accept connections from LDAP clients, or press **Enter** to accept the default.
12. Type **yes** to enable LDAPS, or press **Enter** to accept the default (no). If enabling LDAPS, enter the port to accept connections, or press **Enter** to accept the default LDAPS port.
13. Type **yes** to enable StartTLS for encrypted communication, or press **Enter** to accept the default (no).
14. Select the certificate option for the server and provide the certificate location.
15. 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.
16. Specify the base DN for the PingDirectory Server repository, for example `dc=company,dc=com`.
17. Select an option to populate the database.

18. If this machine is dedicated to the PingDirectory Server, tune the JVM memory allocation to use the maximum amount of memory the **Aggressive** option). This ensures that communication with the PingDirectory Server is given the maximum amount of memory. Choose the best memory option for the system and press **Enter**.
19. Enter **yes** to configure the server on startup and load the backend into memory before accepting connections, or press **Enter** to accept the default (no).
20. To start the server after the configuration, press **Enter** for (yes).
21. Review the Setup Summary, and enter an option to accept the configuration, redo it, or cancel.

PingDataGovernance Server Installation Tools

The PingDataGovernance Server provides a number of tools to install and configure the system.

- The `setup` tool performs the initial tasks needed to start the PingDataGovernance Server server, including configuring JVM runtime settings and assigning listener ports for the PingDataGovernance Server's REST services and applications.
- The `create-initial-config` tool continues after `setup` and configures connectivity between the user store and the PingDataGovernance Server. During the process, the `prepare-external-store` tool prepares each Ping PingDirectory Server to serve as a user store by the PingDataGovernance Server. Configuration can be written to a file to use for additional installations.
- Once the configuration is done, the `dsconfig` tool and the Administrative Console enable more granular configuration.

Install the PingDataGovernance Server

To expedite the setup process, be prepared to enter the following information:

- An administrative account for the PingDataGovernance Server.
- An available port for the PingDataGovernance Server to accept HTTPS connections from REST API clients.
- An available port for the Administrative Console's communication.
- An available port to accept LDAP client connections.
- Information related to the server's connection security, including the location of a keystore containing the server certificate, the nickname of that server certificate, and the location of a truststore.

Perform the following steps for an interactive installation of the PingDataGovernance Server:

1. Download the latest zip distribution of the PingDataGovernance Server software.
2. Unzip the file in any location.

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

3. Change to the top level PingDataGovernance folder.
4. Run the `setup` command.

   ```
   $ ./setup
   ```

5. Type **yes** to accept the terms of this license agreement.
6. The `setup` tool enables cloning a configuration by adding to an existing PingDataGovernance Server topology. For an initial installation, press **Enter** to accept the default (no). When adding additional PingDataGovernance Server instances, an existing peer can be chosen to mirror configuration.
7. Enter the fully qualified host name or IP address of the machine that hosts the PingDataGovernance Server, or press **Enter** to accept the default (local hostname).
8. Create the initial root user DN for the PingDataGovernance Server. This account has full access privileges. To accept the default (`cn=Directory Manager`), press **Enter**.
9. Enter and confirm a password for this account.
10. Define a port for PingDataGovernance Server REST APIs and the Administrative Console to accept HTTPS connections, or press **Enter** to accept the default.
11. Enter the port to accept LDAP client connections, or press **Enter** to accept the default.
12. To enable LDAPS connections press **Enter** and enter a port, or type **no**.
13. To enable StartTLS connections over regular LDAP connection press **Enter**, or type **no**.
14. Enter the certificate option for this server. If needed, the server will generate self-signed certificates that should be replaced before the server is put into production.
15. 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.

16. If this machine is dedicated to the PingDataGovernance Server, tune the JVM memory to use the maximum amount of memory (the **Aggressive** option). If this system supports other applications, choose an appropriate option.
17. Enter a location name for this server. The location is used for failover purposes if this server belongs to a server group.
18. Enter an instance name for this PingDataGovernance Server, or press **Enter** to accept the default (<location> Server 1). The name must be unique in a topology and cannot be changed once configured.
19. Press **Enter** (yes) to start the server when the configuration is applied.
20. Review the configuration options and press **Enter** to accept the default (set up the server).

The installation will continue with the `create-initial-config` tool.

Configure the PingDataGovernance Server

The next set of steps in the setup process rely on the `create-initial-config` tool. The `setup` tool will continue with the `create-initial-config` tool to configure the PingDataGovernance Server. Having the following in place will expedite the configuration:

- At least one Ping Identity PingDirectory Server is installed. Have the host name and communication port available.
- Any additional PingDirectory Servers that act as user stores. Only Ping PingDirectory Servers can be configured with this tool. Other user stores must be configured outside of this process. Have the host names and communication ports available.
- Locations for this and any other PingDataGovernance Servers for failover.

Note

The `create-initial-config` tool can install starter schemas that enable having a base schema for the product to use after installation and/or using reference applications. The schemas are `user` and `user-and-ref-apps`.If neither schema is installed, a custom schema and mapping SCIM resource type or a Pass-through SCIM resource type can be configured later.

After the initial setup and configuration, run the `dsconfig` tool later to make configuration adjustments. Perform the following steps to configure the PingDataGovernance Server:

1. Press **Enter** (yes) to continue with `create-initial-config`. If for some reason the initial configuration cannot be completed in one session, manually restart `create-initial-config` later.
2. Define the account used by the PingDataGovernance Server to communicate with an external User Store, or press **Enter** to accept the default (`cn=Governance User,cn=Root DNs,cn=config`).
3. Enter and confirm the account password.
4. Specify the type of security that the PingDataGovernance Server uses when communicating with all external store instances, or press **Enter** to accept the default (SSL).
5. Enter the `host:port` configured for the first PingDirectory Server. The connection is verified.
6. Select the location name for the PingDirectory Server (or user store server), or enter another location if not listed in the menu.
7. Confirm that the identified host should be prepared. If additional servers will be added as backups, select the **Yes, and all subsequent servers** option. This enables the identification of another server later in the configuration. The `prepare-external-store` tool can also be used to perform these tasks at a later time.
8. Enter the account and password needed to create the root user `cn=Governance User,cn=Root DNs,cn=config` account on the PingDirectory Server. This is the root account created in the initial setup, such as the default (`cn=Directory Manager`. The PingDataGovernance Server sets up the DN and tests that it can access the account. This is also the account used to log into the Administrative Console.
9. To configure the initial user store, press **Enter** for (yes). The user store will be configured with a default Store Adapter and SCIM Resource Type, which will enable mapping of resources in the user store.
10. If there are additional PingDirectory Server locations, enter their `host:port`. If there are no additional servers to add, press **Enter** to continue.
11. Choose one of the predefined schemas (the standard user schema and optionally the reference application schema), or no schema. The instructions for configuration in this guide use the standard user schema.
12. Specify the base DN for locating user entries, such as `ou=people,dc=example,dc=com` and press **Enter**.
13. Review the configuration summary, and press **Enter** to accept the default (w) to write the configuration to a `dsconfig` batch file. The configuration is written to `<server-root>/resource/governance-cfg.dsconfig` . Certificate files are written to `external-server-certs.zip`.

14. Press **Enter** (yes) to confirm that the configuration should be applied to this PingDataGovernance Server and written to the `governance-cfg.dsconfig` file.

This completes the initial configuration for the PingDataGovernance Server. Run the `bin/status` tool to see that the PingDataGovernance Server server is up and running.

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. To log into the Administrative Console, use the initial root user DN specified during setup (by default `cn=Directory Manager`).

The `dsconfig` command or the Administrative Console can be used to create additional root DN users in `cn=Root DNs,cn=config`. These new users require the fully qualified DN as the login name, such as `cn=new-admin,cn=Root DNs,cn=config`. To use a simple user name (with out the `cn=` prefix) for logging into the Administrative Console, the root DN user must have the `alternate-bind-dn` attribute configured with an alternate name, such as "admin."

If the Administrative Console needs to run in an external container, such as Tomcat, a separate package (`<server-root>/resource/admin-console.zip`) can be installed according to that container's documentation.

Install an additional PingDataGovernance Server in a topology

A PingDataGovernance Server instance can be cloned to serve as an additional server in a topology. Adding a server to an existing topology copies the original PingDataGovernance Server's local configuration and links the two configurations. The configuration of PingDataGovernance Server's cluster items and the topology settings are automatically mirrored across servers in a topology. See Topology Overview for details.

Note

When setting up a new PingDataGovernance Server from an existing peer, the existing HTTP (S) connection handlers are not cloned. These connection handlers are created from scratch using default values of the new server and any specified port values.

1. Unpack the zip distribution in a folder different from the peer PingDataGovernance Server.
2. Run the `./setup` command in the `<server-root>` directory of the new server.
3. Accept the licensing agreement.
4. Enter **yes** to add this server to an existing PingDataGovernance Server topology.
5. Enter the host name of the peer PingDataGovernance Server server from which the configuration will be copied.

6. Enter the port of the peer PingDataGovernance Server.
7. Choose the security communication to use to connect to the peer PingDataGovernance Server.
8. Enter the manager account DN and password for the peer PingDataGovernance Server. The connection is verified.
9. Enter the fully-qualified host name or IP address of the local host (the new server).
10. Enter the HTTPS client connection port for the PingDataGovernance Server, or press **Enter** to accept the default.
11. Select the option to install the Administrative Console application, if desired.
12. Enter the HTTPS connection port for the Administrative Console application, or press **Enter** to accept the default.
13. Enter the port on which the new PingDataGovernance Server will accept connections from LDAP clients, or press **Enter** to accept the default.
14. Choose a certificate option for this PingDataGovernance Server.
15. Choose the amount of memory to allocate to the JVM.
16. Choose the location for this server. The location of the peer is listed as an option, or a new location can be defined. Regardless, the new server will have its topology and cluster settings mirrored with its peer.
17. Enter a name for this server. The name cannot be changed after installation.
18. Press **Enter** to start the server after configuration.
19. Review the information for the configuration, and press **Enter** to set up the server with these parameters.
20. To write this configuration to a file, press **Enter** to accept the default (yes).

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 have been 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.
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems. (Update is not supported for version 6.0)

Directories/Files/Tools	Description
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
vendor_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	Legal notices for dependent software used with the product.
docs	Provides the release notes, Configuration Reference Guide (HTML), API Reference, and all other product documentation.
metrics	Stores the metrics that can be gathered for this server and surfaced in the Ping PingDataMetrics Server.
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
webapps	Stores the Administrative Console .war file, the Authentication interface reference application's war file and source, and third-party licenses.
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.

Plan a scripted installation

An interactive installation of an PingDataGovernance Server uses the `setup` and `create-initial-config` tools. This is the recommended installation method and should be used when possible. A scripted installation can be performed, for scenarios that require a custom configuration or automated deployment. The resulting `governance-cfg.dsconfig` batch file, created with the `create-initial-config` tool, can then be used as a basis for scripted installations.

The following is performed by the `create-initial-config` tool during an interactive installation:

External store preparation:

- For each Ping PingDirectory Server, the `prepare-external-store` tool is run. This updates the PingDirectory Server's schema, creates a privileged service account for use by the PingDataGovernance Server with the DN `cn=Governance User,cn=Root DNs,cn=config`, and creates an administrative account.
- If the user store is comprised of LDAP directory servers, the `prepare-external-store` tool is run for every server that comprises the user store. This updates the server's schema, and creates a privileged service account for use by the PingDataGovernance Server with the DN `cn=Governance User,cn=Root DNs,cn=config`.

Server configuration with `dsconfig`:

The `create-initial-config` command has a `--dry-run` option that can be used to generate the `governance-cfg.dsconfig` file in non-interactive, or interactive mode, without applying the configuration to the local server.

Note

The PingDirectory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which PingDataGovernance Server services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

Installation Process

The following is a sample of the commands that should be included in a scripted installation:

1. Set up and configure one or more PingDirectory Servers. See Installing the PingDirectory Server.
2. Run the PingDataGovernance Server `setup` command on the server that will host the PingDataGovernance Server.

```
$ ./setup --cli --no-prompt --acceptLicense --maxHeapSize 2g \
  --ldapPort 2389 --ldapsPort 2636 --httpsPort 8443 \
  --location Austin --instanceName server1 \
  --rootUserPassword <password> \
  --useJavaTrustStore <path>/keystores/truststore.jks \
  --useJavaKeystore <path>/keystores/server1keystore.jks \
  --trustStorePasswordFile<path>/keystores/truststore.txt \
  --keystorePasswordFile <path>/keystores/keystore.txt \
  --certNickname server-cert
```

   The `--trustStorePasswordFile` option is only required if this server is expected to update the truststore with certificates of other servers in the topology.

   The password for the private key associated with the certificate (`server-cert`) should be the same as the keystore password.

3. Run `prepare-external-store` for each user store.

```
$ ./prepare-external-store --no-prompt \
  --hostname ds1.example.com \
  --port 1636 --useSSL --trustStorePath <path>/keystores/truststore.jks
\
  --userStoreBaseDN "ou=people,dc=example,dc=com" \
  --governanceBindPassword <password> \
  --bindDN "cn=directory manager" \
  --bindPassword <password>
```

4. Run the `create-initial-config` tool.

```
$ ./create-initial-config --no-prompt \
  --port 2636 --useSSL --trustStorePath <path>/keystores/truststore.jks
\
  --bindDN "cn=Directory Manager" \
  --bindPassword <password> \
  --governanceBindPassword <password> \
  --externalServerConnectionSecurity useSSL \
  --userStoreBaseDN "o=people,dc=example,dc=com" \
  --userStore ds1.example.com:1636:Austin
```

Start the PingDataGovernance Server

To start the PingDataGovernance Server, run the `bin/start-server` tool on UNIX/Linux systems (the `bat` command is in the same folder for Windows systems).

To start the PingDataGovernance Server in the foreground:

1. Enter the `bin/start-server` with the `--nodetach` option to launch the PingDataGovernance Server as a foreground process.

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

2. Stop the PingDataGovernance Server by pressing CTRL-C in the terminal window where the server is running or run the `bin/stop-server` command from another window.

Stop the PingDataGovernance Server

The PingDataGovernance Server provides a shutdown script, `bin/stop-server`, to stop the server.

Schedule a server shutdown

The PingDataGovernance Server enables scheduling a shutdown and sending a notification to the `server.out` log file. The server uses the UTC time format if the provided timestamp includes a trailing "Z," for example, 201304032300Z. The following example includes a `--stopReason` option that writes the reason for the shutdown to the logs:

```
$ bin/stop-server --task \
  --hostname <server1.example.com> \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword <password> \
  --stopReason "Scheduled offline maintenance" \
  --start 201504032300Z
```

Run an in-core restart

Re-start the PingDataGovernance 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. Shutting down and restarting the JVM requires a re-priming of the JVM cache. To avoid destroying and re-creating the JVM, use an in-core restart, which can be issued over LDAP. The in-core restart will keep the same Java process and avoid any changes to the JVM options.

```
$ bin/stop-server \
  --task \
  --restart \
  --hostname <server1.example.com> \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword <password>
```

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.

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 `Ping-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 `Ping-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:

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

Uninstall the PingDataGovernance Server

The PingDataGovernance Server provides an `uninstall` tool to remove the components from the system. If this instance is a member of a topology of PingDataGovernance Server servers, the `uninstall` tool will remove it from the topology.

Note

If a PingDataGovernance Server is a member of a topology, and the `uninstall` tool is not used to remove it (it was shutdown and deleted manually), it will not be removed from the topology registry. In this scenario, use the `bin/remove-defunct-server` tool to remove the instance from the topology.

Perform the following to uninstall the PingDataGovernance Server:

1. From the server root directory, run the `uninstall` command.

   ```
   $ ./uninstall
   ```

1. Select the option to remove all components or select the components to be removed.
2. To selected components, enter **yes** when prompted.

   ```
   Remove Server Libraries and Administrative Tools? (yes / no) [yes]: yes
   Remove Log Files? (yes / no) [yes]: no
   Remove Configuration and Schema Files? (yes / no) [yes]: yes
   Remove Backup Files Contained in bak Directory? (yes / no) [yes]: no
   Remove LDIF Export Files Contained in ldif Directory? (yes / no) [yes]: no
   The files will be permanently deleted, are you sure you want to continue? (yes / no)
   [yes]:
   ```

3. Manually delete any remaining files or directories.

Chapter 3: Data access and mapping

PingDirectory Servers provide the resources that can be accessed by clients. Attributes can be mapped from multiple PingDirectory Servers to create a unified identity in a SCIM Resource Type. The SCIM Resource Type is the component that makes resources available to clients.

Topics include:

Data components

Primary and secondary store adapters

SCIM schemas

Store adapter mappings

SCIM attribute search considerations

Maintain username uniqueness

Define SCIM Resource Types

Complex attribute mapping

Client-specific SCIM attributes

Access data

Data components

When a PingDirectory Server is configured, a store adapter is installed to read and return native SCIM objects. Custom store adapters can be created for non-LDAP PingDirectory Servers with the Ping Identity Server SDK. See Server Extensions for information.

The attributes surfaced for each backend store are mapped in SCIM Resource Types to enable a unified view of a user profile, and to make them available to clients. The PingDataGovernance Server provides full read/write access through the SCIM Resource Type (`/scim/v2/Me`). The access to these resources is subject to policy rules and restrictions.

Store adapter mappings

A SCIM Resource Type enables attribute mappings between the native store adapter schema and the SCIM schema. The store adapter mapping can contain additional information as to whether the native attribute is readable, writable, searchable, or authoritative. One must be authoritative. A SCIM Resource Type can map attributes from multiple PingDirectory Servers and determine which attributes are the authoritative resource for a user profile. See Using SCIM Resource Type Attributes in Policy for details about policy evaluation.

Directory Servers

The user stores provide data resources. One or more Ping PingDirectory Servers, PingDirectoryProxy Servers, or third-party directory servers can serve as a user store. SCIM Resource Type mappings can be used to aggregate attributes from multiple PingDirectory Servers into a unified view.

When a store adapter is added to the PingDataGovernance Server's server configuration, a correlation attribute must be defined for SCIM Resource Types that are backed by multiple store adapters. The correlation attribute defines an attribute for each store adapter that is used to uniquely identify the same end user data across different store adapters. For example, if every PingDirectory Server stores a user's email address, and an email address can always be considered a primary key (that is, it is always unique per use), then each store adapter's email address attribute can be set as its correlation attribute.

Note

> The PingDirectory Server ACIs may need to be configured to grant access to elements of data, or specific LDAP controls using ACIs, depending on which PingDataGovernance Server services are used. See `resource/starter-schemas/README.txt` for sample ACIs.

Primary and secondary store adapters

Store adapters contain the configuration that the PingDataGovernance Server uses to interact directly with external PingDirectory Servers. Every PingDirectory Server providing a distinct set of user data must have a store adapter entry in the configuration.

If the PingDataGovernance Server is used to aggregate user attributes from multiple PingDirectory Servers, secondary store adapters can be configured. "Primary store adapter" and "Secondary store adapter" designate how a SCIM Resource Type prioritizes user data lookups to multiple store adapters. The primary store adapter is always checked first when processing a request for a user resource, and then any secondary store adapters are checked. A user account effectively does not exist if a record does not exist for it on the primary store adapter. The primary store adapter should be used to store a user's core attributes, while a secondary store adapter can store additional attributes.

Defining correlation attributes

When handling a request for a particular user, the PingDataGovernance Server needs a way to correlate an entry in the primary store adapter with any related entries in secondary store adapters. This is done by correlating the value of an attribute shared across the store adapters using the secondary store adapter's `primary-correlation-attribute` and `secondary-correlation-attribute` properties. The correlation attribute should have a value that is unique for each user.

Note

When creating SCIM resources backed by secondary store adapters, the server automatically sets the secondary correlation attribute value if it does not already have a value from the resource create request.

For example, user entries can be correlated across store adapters by email address:

```
$ dsconfig create-secondary-store-adapter \
  --type-name Users \
  --adapter-name MarketingData \
  --set store-adapter:DemographicsStoreAdapter \
  --set primary-correlation-attribute:mail \
  --set secondary-correlation-attribute:emailAddress
```

Sample configuration

An environment may have two LDAP PingDirectory Servers with distinct sets of data. Set A may have user credentials and profile attributes, and is configured with the primary store adapter. Set B may have demographic data about these users, and is configured with the secondary store adapter. The following can be configured for this scenario:

1. Configure each server in Set A.

```
$ bin/dsconfig create-external-server \
  --server-name profile-server \
  --type ping-identity-ds \
  ...
```

2. Configure each server in Set B.

```
$ dsconfig create-external-server \
  --server-name demographics-server \
  --type ping-identity-ds \
  ...
```

3. Create LDAP load balancing algorithms.

```
$ dsconfig create-load-balancing-algorithm \
  --algorithm-name "Profile Store LBA" \
  --type failover \
  --set enabled:true \
  --set backend-server:profile-server
```

```
$ dsconfig create-load-balancing-algorithm \
  --algorithm-name "Demographics Store LBA" \
  --type failover \
  --set enabled:true \
  --set backend-server:demographics-server
```

4. Create store adapters.

```
$ dsconfig --adapter-name ProfileStoreAdapter \
  --type ldap \
  --set enabled:true \
  --set "load-balancing-algorithm:Profile Store LBA"
  ...
```

```
$ dsconfig --adapter-name ProfileStoreAdapter \
  --type ldap \
  --set enabled:true \
  --set "load-balancing-algorithm:Demographics Store LBA"
  ...
```

5. Designate the primary store adapter.

```
$ dsconfig create-scim-resource-type \
  --type-name Users \
  --type mapping \
  --set enabled:true \
  --set endpoint:Users \
  --set primary-store-adapter:ProfileStoreAdapter \
  --set core-schema:urn:example:schemas:Profile:1.0 \
  --set optional-schema-extension:urn:example:schemas:Demographics:1.0
```

6. Designate the secondary store adapter and correlation attributes.

```
$ dsconfig create-secondary-store-adapter \
  --type-name Users \
  --adapter-name MarketingData \
  --set store-adapter:DemographicsStoreAdapter \
  --set primary-correlation-attribute:mail \
  --set secondary-correlation-attribute:emailAddress
```

SCIM schemas

Each SCIM Resource Type maps to one core SCIM schema and optional extension schemas. SCIM schemas are used to define the resources that can be retrieved from a backend PingDirectory Server. Each SCIM Resource Type represents one type of resource, such as "user" or "account," and the schema defines the attributes of that resource.

Store adapter mappings

The PingDataGovernance Server uses store adapter mappings to determine which store adapter handles which attribute from the SCIM schema. For cases in which an attribute can be found on multiple store adapters, one store adapter mapping should be created for each combination of attribute and store adapter. One of these mappings must have the shared attribute set as `authoritative`. This designates the store adapter that will be the authoritative source when multiple possible values are found across a set of store adapters.

In the following example, the SCIM attribute `urn:pingidentity:schemas:sample:profile:1.0:topicPreferences` is mapped to the LDAP attribute `ubidXTopicPreferenceJSON` from the Marketing PingDirectory Server adapter:

```
$ bin/dsconfig create-store-adapter-mapping \
  --type-name Users \
  --mapping-name topicPreferences \
  --set secondary-store-adapter:DemographicsStoreAdapter \
  --set scim-resource-type-attribute:urn:example:schemas:Demographics:1.0:topicPreferences
\
  --set store-adapter-attribute:ubidXTopicPreferenceJSON \
  --set authoritative:true
```

SCIM attribute search considerations

In order to provide paging and sorting, the PingDataGovernance Server holds an entire search result set in memory while it processes a SCIM search request. This is true for searches that do not request paging or sorting. The SCIM Resource Type `lookthrough-limit` property sets an upper bound for searches, so that clients do not exhaust the server resources. If the number of search results for a given request exceeds this value, an error is returned to the client indicating that the search matched too many results. A request that causes an unindexed search is also restricted to the size limit of the `lookthrough-limit` setting.

The PingDataGovernance Server attempts to find a single store adapter that can process the provided search filter. The primary store adapter is checked first to see if it can process the search filter. If it cannot, the secondary store adapters are consulted in no particular order. The first store adapter capable of processing the search filter is chosen. The store adapter must be able to return a superset of possible matches for the filter. The attributes in the search filter must correspond to at least one searchable native attribute in the store adapter. If the SCIM Resource Type is a Mapping SCIM Resource Type, the store adapter mapping for the search filter attribute must be marked as `searchable`.

If no store adapters can process the search, the PingDataGovernance Server returns an error. For each candidate search result from a store adapter, the PingDataGovernance Server assembles a complete SCIM resource by retrieving the native resource for every other store adapter using the store adapter correlation attributes (set when secondary store adapters are defined) and merging them together. Each resulting candidate SCIM resource is checked to see if it matches the provided search filter and is discarded if it does not match.

Maintain username uniqueness

The PingDataGovernance Server's default schema configuration uses "`uid`" as the RDN attribute of user DNs, which ensures that all `uid` values are unique for that branch of the DIT. In the default configuration, `uid` is recognized as a user's username. The PingDataGovernance Serverstore adapter mapping for the `userName` attribute of the default starter schema relies on this.

It may be the case that the attribute used for the username is also an RDN attribute in the PingDirectory Server. If every entry resides on the same branch, these attribute values will always be unique. Any configuration changes that do not maintain this structure must ensure that usernames are unique. The Ping PingDirectory Server provides the attribute uniqueness plugin that can be used if configuration changes are required. See the *Ping PingDirectory Server Administration Guide*.

Define SCIM Resource Types

SCIM Resource Types provide a unified view of resources between the PingDataGovernance Server and one or more underlying PingDirectory Servers, and correspond to the SCIM 2.0 SCIM Resource Type. SCIM Resource Types determine what resources can be accessed from a PingDirectory Server. Each SCIM Resource Type represents one resource, such as "user" or "account" and the schema defines the attributes of that resource.

Note

When mapping attributes, PingDirectory Server attributes and SCIM Resource Type attributes must be of compatible types. For example, an attribute with an integer value must be mapped to another attribute with an integer value. An attribute with a string value can only be mapped to attributes with boolean, integer, or date-time if it can be parsed.

There are two types of SCIM Resource Types: Pass-through SCIM Resource Type and Mapping SCIM Resource Type. A Mapping SCIM Resource Type relies on a SCIM Schema, which is installed with the configuration of a user store on a PingDirectory Server.

Pass-through SCIM Resource Type

This type of SCIM Resource Type simply exposes the primary store adapter’s data as core attributes, while secondary store adapter’s data are exposed as schema extensions. No schema needs to be defined at the SCIM Resource Type and all schema enforcement is at the responsibility of the store adapters. Since no schema is defined at the SCIM Resource Type, attribute mappings are not defined. If the configured store adapter exposes a schema, it will be enforced as the core or extension schemas for the SCIM Resource Type.

Mapping SCIM Resource Type

Attributes associated with a SCIM Resource Type are configured by specifying at least one core schema and one or more schema extensions. The core schema defines attributes that can appear at the root level of the SCIM resource exposed by the SCIM Resource Type. Schema extensions define attributes that are namespaced by the schema's URI. Schema extensions can be optional or required. When processing client requests, the SCIM resource from the client is first checked against the schemas defined for the SCIM Resource Type (core or extension). The request is then mapped to a store adapter object, using the store adapter mappings, and then processed.

Create a SCIM Resource Type

After user stores and store adapters are in place, SCIM Resource Types can be defined to provide a unified view of identity data found in multiple PingDirectory Servers. The SCIM Resource Type determines the attributes that can be accessed by a client.

The following is a sample command for creating a mapping SCIM Resource Type:

```
$ bin/dsconfig create-scim-resource-type \
  --type-name Users \
  --type mapping \
  --set "description:Users Resource Type" \
  --set enabled:true \
  --set endpoint:/Users \
  --set primary-store-adapter:UserStoreAdapter \
  --set core-schema:urn:pingidentity:schemas:User:1.0 \
  --set required-schema-extension:urn:pingidentity:schemas:sample:profile:1.0
```

SCIM Resource Types can also be configured in the Administrative Console through **SCIM -> SCIM Resource Types**.

Create a Mapping SCIM Resource Type

The following information is used to configure a Mapping SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP store adapter.

- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.
- The core schema for the primary store adapter and any extension schemas.

Create a Pass Through SCIM Resource Type

The following information is used to configure a Pass Through SCIM Resource Type:

- A name for this SCIM Resource Type.
- An optional description for the SCIM Resource Type.
- The SCIM Resource Type's endpoint HTTP address, which will be relative to the `/scim/v2` base URL.
- A primary store adapter to persist the data for this SCIM Resource Type.
- The primary store adapter attribute to use as the value for the SCIM object ID. The object ID is a unique, immutable identifier for fetch, update, and delete operations on an object. The `entryUUID` attribute is the default for an LDAP store adapter.
- A look-through limit for the maximum number of resources that the SCIM Resource Type should scan when processing a search request. This prevents a client from taking too many of the server's resources for a single search.

Edit attribute and sub-attribute properties

Attribute properties in the schema can be configured to change the actions that can be performed, and when an attribute is returned to a requesting client. If the attribute contains sub-attributes, those can be configured as well.

```
$ bin/dsconfig set-scim-attribute-prop \
  --schema-name urn:pingidentity:schemas:User:1.0 \
  --attribute-name displayName \
  --set "description:User's name."
  --set required:true \
  --set case-exact:true \
  --set mutability:read-only
```

This can be configured in the Administrative Console by editing a schema in **SCIM -> SCIM Schemas**. Select a schema and edit any of the attributes listed. The following can be configured for an attribute or sub-attribute:

- An optional description of the attribute.
- The attribute type, which can be:
  - **string** - A sequence of zero or more Unicode characters encoded using UTF-8.
  - **boolean** - The literal `true` or `false`.

  - **datetime** - A date and time encoded as a valid `xsd:dateTime` (for example, 2008-01-23T04:56:22Z).
  - **decimal** - A real number with at least one digit to the left and right of the period.
  - **integer** - A decimal number with no fractional digits.
  - **binary** - Arbitrary binary data.
  - **reference** - A URI for a resource. A resource can be a SCIM resource, an external link to a resource (such as a photo), or an identifier such as a URN. The `reference-type` property must be specified for these attributes.
  - **complex** - A singular or multi-valued attribute whose value is a composition of one or more sub-attributes.
- Specify if the attribute is required.
- Specify if the attribute is case-sensitive.
- Specify if the attribute can have multiple values.
- Specify suggested canonical values that can be used (such as work and home).
- The circumstances under which the values of the attribute can be written (mutability). Values include:
  - **read-only** - The attribute cannot be modified.
  - **read-write** - The attribute can be updated and read.
  - **immutable** - The attribute may have its initial value set, but cannot be modified after.
  - **write-only** - The attribute can be updated but cannot be read.
- The circumstances under which the values of the attribute are returned in response to a request. Values include:
  - **by-default** - The attribute is returned by default in all SCIM responses where attribute values are returned.
  - **upon-request** - The attribute is returned in response to any PUT, POST, or PATCH operations if the attribute was specified by the client (for example, the attribute was modified).
  - **always** - The attribute is always returned.
  - **never** - The attribute is never returned.
- The SCIM Resource Types that can be referenced. This property is only applicable for attributes that are of type `reference`. Valid values are a defined SCIM Resource Type, `external` indicating the resource is an external resource (such as a photo), or `uri` indicating that the reference is to a service endpoint or an identifier (such as a schema urn).

- If the attribute is complex and has sub-attributes, they can be edited as well with these values.

Edit store adapter mappings

Store adapters are designed to surface the schema of a backend Directory Server. Store adapter mappings map SCIM Resource Type attributes and store adapter attributes. When the PingDataGovernance Server is installed with a Ping PingDirectory Server, the schema attributes are automatically mapped to a User SCIM Schema Resource Type.

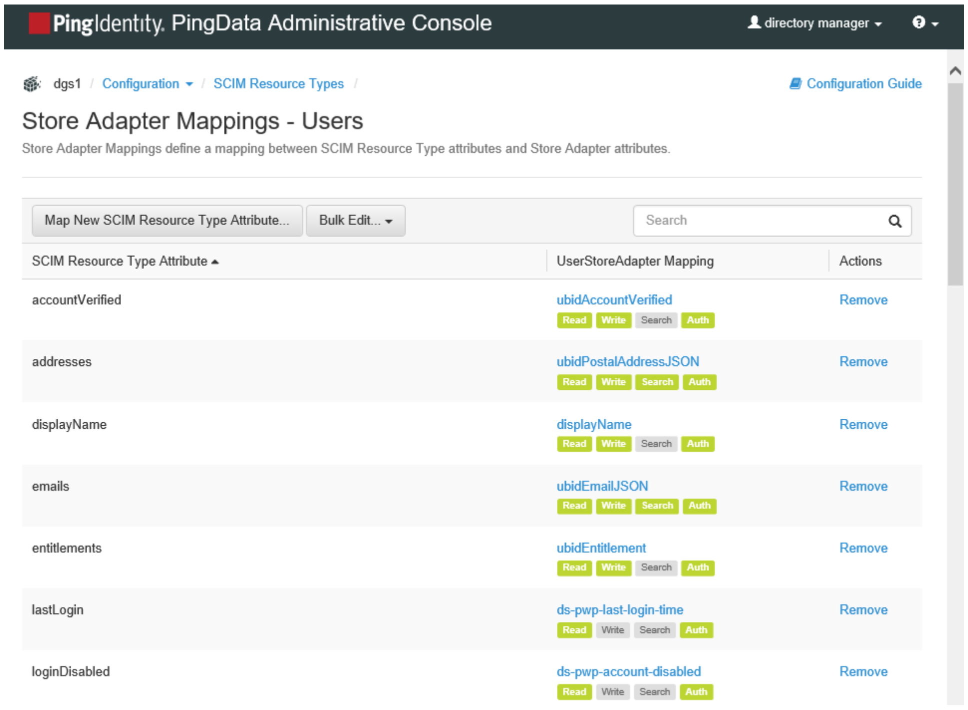
Note

If the SCIM Resource Type attribute name changes, make sure that scopes and OpenID Connect Claims are updated to reflect the change.

The following is a sample command for editing a store adapter attribute mapping:

```
$ bin/dsconfig set-store-adapter-mapping-prop \
  --type-name Users \
  --mapping-name communicationOpts \
  --set store-adapter-attribute:ubidXCommunicationOptJSON \
  --set writable:false \
  --set searchable:true \
  --reset authoritative
```

Store adapter mappings can also be configured in the Administrative Console through **SCIM -> SCIM Resource Types**. Click **Actions -> Edit Store Adapter Mappings** for a SCIM Resource Type. The following is displayed:


PingIdentity. PingData Administrative Console
directory manager
dgs1 / Configuration / SCIM Resource Types /
Configuration Guide
Map New SCIM Resource Type Attribute...
Bulk Edit...
Search

Store Adapter Mappings - Users

Store Adapter Mappings define a mapping between SCIM Resource Type attributes and Store Adapter attributes.

SCIM Resource Type Attribute	UserStoreAdapter Mapping	Actions
accountVerified	ubidAccountVerified Read Write Search Auth	Remove
addresses	ubidPostalAddressJSON Read Write Search Auth	Remove
displayName	displayName Read Write Search Auth	Remove
emails	ubidEmailJSON Read Write Search Auth	Remove
entitlements	ubidEntitlement Read Write Search Auth	Remove
lastLogin	ds-pwp-last-login-time Read Write Search Auth	Remove
loginDisabled	ds-pwp-account-disabled Read Write Search Auth	Remove

Individual attributes can be changed, or all can be edited by clicking **Bulk Edit**. For each attribute, the following can be configured:

- The store adapter attribute that is mapped to the SCIM Resource Type attribute.
- **Readable** – The SCIM Resource Type can read this attribute.
- **Writable** – The SCIM Resource Type can write to this attribute.
- **Searchable** – This specifies whether the attribute is efficiently searchable in the underlying PingDirectory Server. Indexed PingDirectory Server attributes determine what attributes (from the SCIM Resource Type Schema) can be used in a SCIM filter when performing a query. If an attribute is not indexed in the PingDirectory Server, it should not be marked as Searchable here.
- **Authoritative** – If there are multiple mappings for this attribute (from multiple PingDirectory Servers), one must be marked Authoritative.

Complex attribute mapping

For searches involving sub-attributes of SCIM attributes that are mapped to LDAP JSON attributes in the PingDirectory Server, the sub-attribute field names in the search filter are treated case-sensitively because the PingDirectory Server treats them this way. This is a departure from the SCIM 2.0 specification, where attribute names in search filters are case-insensitive.

For example, the SCIM attribute `name` has the sub-attribute `familyName`. The SCIM attribute `name` is mapped to the LDAP JSON attribute `scimName`. The search filter `NAME.FAMILYNAME eq "Zweig"` will not return a search result for an entry containing the specified value `Zweig` in the `familyName` sub-attribute. A search result for this entry is returned if the filter is specified instead as `NAME.familyName`. This is because the top-level attribute can be matched case-insensitively but the sub-attribute can only be matched case-sensitively.

Client-specific SCIM attributes

Some environments may find it useful to designate a namespaced, schema-less portion of a SCIM user resource, in which a client can store its data. For example, a resource type could be configured such that an application may write any previously undefined attributes that are prefixed with `urn:customApp1`.

To enable this, the data store schema must first have a single-valued JSON attribute defined to hold application-specific attributes. For example, for an LDAP attribute called `customApp`:

```
customApp: { "urn:customApp1":{ "wine":["Napa Cabs","French Burgundy","Lodi Zinfandel"],
"age":"2000-2010" } }
```

This value should appear in the SCIM resource as follows:

```
  'urn:customApp1' : {
      'wine' : [ 'Napa Cabs', 'French Burgundy', 'Lodi Zinfandel' ],
      'age' : '2000-2010'
  }
```

The following is a command line sample of the steps needed to configure this type of functionality in the PingDataGovernance Server, or this process can be done in the Administrative Console.

1. Create a store adapter mapping from "`*`" (SCIM) to "`customApp`" (LDAP). Using a wildcard SCIM attribute, client-specific SCIM attributes do not need to be defined in advance. To map only attributes from a single SCIM schema to an LDAP attribute, use a schema-specific SCIM wildcard such as `urn:myExtensionSchema:*`.

```
$ bin/dsconfig create-store-adapter-mapping \
  --type-name "Users" \
  --mapping-name "customAppWildcard" \
  --set "scim-resource-type-attribute:*" \
  --set store-adapter-attribute:customApp
```

2. Set the SCIM Resource Type's `schema-checking-option` property to `allow-undefined-attributes`.

```
$ bin/dsconfig set-scim-resource-type-prop \
  --type-name "Users" \
  --add schema-checking-option:allow-undefined-attributes
```

3. Define a wildcard scope that uses the client-specific namespace `urn:customApp1` as a prefix. Since the mapping is a wildcard, this prevents the client from reading or writing any user attribute, and client-specific attributes do not need to be defined in advance.

```
$ bin/dsconfig create-oauth2-scope \
  --scope-name Wildcard-Scope \
  --type authenticated-identity \
  --set "consent-prompt-text:Save application data to your account!" \
  --set "resource-attribute:urn:customApp1:*" \
  --set resource-operation:modify \
  --set resource-operation:retrieve
```

4. Create the client and assign the wildcard scope to it.

```
$ bin/dsconfig create-oauth2-client \
  --client-name "App1" \
  --set client-id:<App-ID> \
  --set client-secret:<secret> \
  --set grant-type:authorization-code \
  --set grant-type:implicit \
  --set scope:openid \
  --set scope:email \
  --set scope:Wildcard-Scope \
  --set redirect-url:https://company.com:<port>/client/
```

Access data

The SCIM endpoint provides full operations on user profile data through the SCIM protocol. The endpoint's URL context path is `/scim/v2/{name}`. Each SCIM resource, specified in the SCIM Schema, is exposed as an endpoint. For example, the URL path `/scim/v2/Users` would be used to access the `Users` SCIM resource. Access to resources is determined by the XACML policies that are configured for the PingDataGovernance Server. If a request to the PingDataGovernance Server is delivering partial results, it may be due to policy settings. See How Policy affects access to scopes.

The PingDataGovernance Server SCIM endpoint enables applications to perform actions on an end user's resources, if XACML policies permit. The following are important to consider when using the SCIM endpoint:

/Me. SCIM supports a special endpoint to retrieve attributes of the currently authenticated user without knowing the SCIM ID. Retrieve attributes of the currently authenticated user with the following:

```
/scim/v2/Me
```

Chapter 4: Token access

Each client request is processed by policies, which determine whether requested scopes can be granted. The PingDataGovernance Server validates the access tokens included with the client request to ensure that only authorized resources are accessed.

Topics include:

PingDataGovernance Server endpoint for OAuth2 clients

Access token validation

PingFederate Access Token Validator

JWT token validation

PingDataGovernance Server endpoint for OAuth2 clients

The Data Governance Server provides a SCIM REST endpoint for client access. The following list presents a summary of the endpoints that may be called by a client application requesting user profile data. See `<server-root>/docs/restapi/index.html` for details.

Data Governance Server endpoint for clients

Endpoint	Description
/scim	
/scim//v2/<name>	This is the SCIM 2.0 protocol endpoint used to retrieve a specified SCIM Resource Type, where `<name>` is the SCIM Resource Type being accessed. This endpoint supports all SCIM operations and implements its access control through the XACML policies. A request to this endpoint requires a scope that includes a `resourceOperations` value that represents the desired action.

Access token validation

Access Token Validators validate tokens submitted by client applications requesting access to protected resources. Any number of validators can be configured for the PingDataGovernance Server. Two types of access token validators are available. One for PingFederate tokens and another for JWT. Server SDK extensions can be installed to enable the PingDataGovernance Server to accept access tokens issued by other identity providers.

A third-party Access Token Validator is responsible for decoding an incoming access token and returning token metadata that is similar in content to that specified by RFC 7662. Metadata includes whether the token is valid and what scopes are granted to the token. This information is passed to the PingDataGovernance Server policy engine, which is responsible for determining whether the token should be accepted and if it is sufficient to allow the incoming request to be processed.

When an access token is presented with a resource request, the PingDataGovernance Server cycles through each configured Access Token Validator until it finds one that can decode the token.

PingFederate Access Token Validator

Before configuring a PingFederate Access Token Validator on the PingDataGovernance Server, complete the following tasks on the PingFederate instance:

1. Create a client on the PingFederate instance that represents the PingDataGovernance Server as a resource server.
2. Set the Allowed Grant Types for this client to include the Access Token Validation grant type.

3. Record the PingDataGovernance Server client ID and client secret for use when configuring the access token validator on the PingDataGovernance Server.

To configure an instance of the PingFederate Access Token Validator on the PingDataGovernance Server, have the following information:

- The base URL of the PingFederate instance (such as https://myPingFedInstance.example.com:9031).
- The client ID and client secret for the validator to use when validating the token with PingFederate.

Trust and key managers may also need to be configured for communication with the PingFederate server over SSL. See Public and Private Keystore Configuration.

If the access token validator will support tokens obtained through OAuth2 grant types other than Client Credentials, the token owner must be mapped to a user defined within the PingDataGovernance Server’s user store. This is done by creating one or more Token Resource Lookup Methods for the access token validator. Each Token Resource Lookup Method references a SCIM Resource Type and SCIM filter that are used by the PingDataGovernance Server to look up the local user that owns the access token. The SCIM filter is used to associate a token property (such as the `externalId` or `sub` claim) to a SCIM user with matching attribute(s). See the online *Configuration Reference Guide*, or `dsconfig` tool help.

JWT token validation

The JWT Access Token Handler enables self introspection of an access token. The JWT token can be encrypted or plain text.

The PingDataGovernance Server supports creating a key pair using RSA algorithms (Key Pairs). The public key needs to be exported from the PingDataGovernance Server and added to an instance of PingFederate or another client's Access Token Manager and configured to use this public key to encrypt the access token.

To verify signatures, the validator needs the public key(s) of the entity that is issuing access tokens. Those public keys can either be imported (as Trusted Certificates) or obtained from the JWKS endpoint of the access token issuer. Multiple keys are supported.

A mapping mechanism for `sub` and `client_id` is also available, if PingFederate or other client's Access Token Manager is configured to provide this information in non-standard claims. All JWT or custom claims should be made available in the request context for tokens.

Chapter 5: Configure scopes and policies

Scopes define the attributes that a client can request, the name that is displayed to end users, the claims that can be accessed, and the actions that can be performed on each attribute. Scopes must be defined in the PingDataGovernance Server before a client can include them in requests.

Policies are the rules that determine what resources may be accessed by client applications. Policies include the criteria by which access decisions are made using targets, rules, conditions, obligations, and a rule combining algorithm. Default policies are available, or custom policies can be written.

Topics include:

OAuth2 scopes

Create scopes

Policy overview

Policy structure

Policy and request processing

Policy engine request context

Configure the policy service

Policy Information Providers

Create policies

Create a policy set

Troubleshoot policies with traces

OAuth2 scopes

When a client makes a request for resources, it specifies the level of access that it requires using scopes. Based on the application's configuration, and the policies that process the request, the PingDataGovernance Server decides whether the resource request should be permitted.

There are three scope types:

- Generic OAuth2 scope (used for external Resource servers).
- Authenticated Identity scope.
- Resource scope.

A Generic OAuth2 scope includes the following properties, which are the base properties for the Authenticated Identity and Resource scopes.

Generic OAth2 Scope Properties

Property	Description
tokenName	The scope name as presented in an OAuth2 request.
type	The scope type, which is `oauth2` for generic scopes.
description	A description of the scope for administrative use.
tags	A list of Tags associated with this scope. Tags are arbitrary additional properties that can be examined by policies.

Authenticated Identity Scope

This scope is granted for an authenticated end user. Once granted, the scope can be used to access the attributes of that authenticated identity. The attributes can be obtained through SCIM endpoints using the `/Me` authenticated subject alias as well as the URI of the SCIM resource.

Properties in this scope include those in the generic OAuth2 scope and the following properties. At least one of the operation properties must have a value.

Authenticated Identity Scope Properties

Property	Description
type	The scope type, which is `authenticatedIdentity` for authenticated identity scopes.
resourceOperations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>`, or `/Me` • `replace` (PUT) to endpoint `/scim/v2/<id>`

Authenticated Identity Scope Properties

Property	Description
	• `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
resourceAttributes	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace.

Resource Scope

An OAuth2 scope that allows a client bearing a granted token to access resources of a specified SCIM Resource Type. It defines the SCIM operations (search, create, retrieve, update, and delete) that can be performed by the client, and the attributes that can be retrieved or updated. A Resource scope potentially allows access (subject to policy) to all resources of a specified SCIM Resource Type.

Resource Scope Properties

Property	Description
type	The scope type, which is `resource` for resource scopes.
scimResourceType	The SCIM Resource Type that can be accessed with this scope.
resourceOperations	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
resourceAttributes	A list of one or more SCIM attributes of the SCIM Resource Type for which this scope allows access. The type of access is determined by the operation properties `create`, `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace.

For granting access to PingDataGovernance Server resources, the values of the `resourceAttributes` property are attribute notation strings as defined in the SCIM 2.0, with the addition of being able to specify wildcards for all attributes.

Create scopes

An OAuth2 scope indicates which data are being requested with a resource request. Typically, one or more scopes are submitted with each request. Scopes are created based on the access and authentication requirements of the data requested. A standard set of OpenID Connect scopes is installed with the PingDataGovernance Server, and additional scopes can be created.

The following is a sample command for creating a scope:

```
$ bin/dsconfig create-oauth2-scope \
  --scope-name workPhone \
  --type authenticated-identity \
  --set resource-attribute:work-phone \
  --set resource-operation:modify
```

Scopes can also be created in the Administrative Console through **Authorization and Policies -> OAuth2 Scopes**.

Create an Authenticated Identity OAuth2 scope

The following information is used to configure an Authenticated Identity scope. See Authenticated Identity Scope for details about the values allowed for resource operations.

- An OAuth2 access token name that is compliant with the OAuth 2.0 Specification (RFC 6749). The following characters are not permitted: space, `'"'`, `'\'`, `'+'` and `','`.
- An optional description.
- Any optional tags associated with this scope. Tags are arbitrary additional properties that can be examined by XACML policies for authorization decisions, such as `HIPAA` or `billing`.
- Specify the resource operations allowed by this scope.
- Specify the resource attributes for which this scope allows access. The type of access is determined by the Resource Operation property. A value of "*" indicates that all attributes are accessible.

Create a Resource OAuth2 scope

All of the Authenticated Identity values are available for the Resource scope, with the addition of the SCIM Resource Type from which the scope can access resources.

Policy overview

Policies determine the scopes that can be accessed by requesting clients through the use of an access token, and the operations on attributes within the scope that are allowed. Policy creation must balance the privacy requirements of the organization with the resource access requirements of the clients. Policies are based on the the eXtensible access control markup language (XACML) as specified in the *OASIS Committee Specification 01, eXtensible access* *control markup language (XACML) Version 3.0*. The targets, rules, conditions, and rule combining algorithms are expressed using JEXL. The native language features of JEXL duplicate a large subset of functions defined by XACML and provide a more concise mechanism for defining policy conditionals. See Policy structure for details about policy components.

Policies are evaluated by the PingDataGovernance Server in response to the following requests made by clients:

- All SCIM requests:
    - Search request
    - Get request
    - Update request
    - Create request
    - Delete request
- A request to the PDP endpoint.

To create policies that will work as expected, or to create clients that can access data correctly, review the parameters and attributes that will be included in the policy.

Requesting operations through SCIM

The PingDataGovernance Server uses policies to determine whether a request for resources should be granted given the scopes defined in the access token. Obligations can be used to define conditions for limiting access to certain attributes. The requested attributes are returned to the client, and any permitted operation (such as adding or modifying an address) is performed.

Policy structure

Variable elements of policy, such as targets, rule conditions, variable definitions, and obligation/advice expressions, are specified using JEXL. For a policy to be evaluated against a request, the request needs to match the values specified in the policy Target element first. If the target for the request matches the target for the policy, the rules in the policy are evaluated. This occurs for each policy.

Just as there is a target for the policy, there is a target for each rule. For the rule Target element to be evaluated, a value in the request must match, as defined in the Match element. If the request matches a value, the rest of the conditions of the rule are evaluated.

Note

If no target is specified for a policy or a rule, the policy or rule is always evaluated.

If the conditions of a rule are satisfied, the result can be either `permit` or `deny` for that single rule. If there are multiple rules in a policy, the rule combining algorithm for the policy determines how the rule evaluation results are combined into a single policy decision.

If there are multiple policies that apply to the request, a policy-combining algorithm determines how the decisions rendered by multiple policies are combined to form a decision by the PingDataGovernance Server. By default, the combining algorithm for PingDataGovernance Server policies is `deny-overrides`. This can be changed in the Policy Service through the Administrative Console or with the `dsconfig` tool.

JEXL use in policy structure

XACML references to data from the request context are either Attribute Designators or Attribute Selectors. Attribute Designators refer to specific named attributes. Attribute Selectors allow JSON-path access to complex JSON objects. Using JEXL, all references to request context data use the syntax `category.value`, where `category` refers to the XACML category name and `value` can be either an attribute name or a JSON path. At policy evaluation time the PingDataGovernance Server first checks to see if `value` refers to a named attribute in the context, and if not will interpret it as a path.

The following JEXL condition can be used to deny a SCIM create request if the user to be created has a work email address in the ".gov" domain. It references the SCIM action attribute using its attribute ID and the user's email address through a JSON path.

```
action.action_id == "create" and scim_request.emails[type eq "work"].value =$ ".gov"
```

JEXL identifiers and variables

JEXL variable names are limited to alphanumeric characters, '`_`', and '`$`.' The PingDataGovernance Server supports the standard XACML attribute categories and attribute names, as described in Standard attribute use. Since JEXL variable names are limited to alphanumeric characters, the PingDataGovernance Server requires the use of short names rather than full URNs when referencing standard XACML attributes from JEXL.. For standard XACML names such as `urn:oasis:names:tc:xacml:3.0:attribute-category:action`, the short name is the relative portion of the URN after the final "`:`" character. In cases where this portion contains a '`-`' character it must be replaced with an underscore '`_`.' Therefore the standard attribute ID `urn:oasis:names:tc:xacml:1.0:action:action-id` should be referenced as `action_id`.

References containing invalid JEXL characters

Because SCIM schema URNs typically contain characters that are not allowed in JEXL identifiers, paths containing extension schemas must be quoted. For example, the following expression could be used to reference the manager ID of a user record that supports the standard SCIM 2.0 Enterprise User Schema Extension:

```
resource.'urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:manager'.id
```

SCIM also allows attribute names to contain dashes (`-`), where JEXL does not. These attributes must also be quoted in a JEXL expression.

Extended data type support

The PingDataGovernance Server JEXL evaluation can comprehend the XML string formats for `time`, `dateTime`, `date`, `dayTimeDuration`, and `yearMonthDuration`. For example, the following expression checks if the access token's expiration time is greater than the current time:

```
access_token.exp > environment.currentDateTime
```

JEXL extension functions

In addition to the set of JEXL functions provided, the following extensions are supported under the namespace `ext`.

XACML Function Access - An extension function is available to access XACML functions that JEXL does not support. The syntax to invoke a XACML function directly is `ext:xacml (functionName, functionArguments …)`. The `functionName` is the short name for the function. Function arguments can be any legal JEXL expression and may themselves refer to values from the XACML request context.

The following example invokes the PingDataGovernance Server's XACML extension function `scimAttribute-subset`. The function name itself is passed as a string, while function arguments may themselves be JEXL expressions.

```
ext:xacml("scimAttribute-subset", scim_request.impacted_attributes, applicable_
scope.scope.resourceAttributes)
```

Accessing Referenced SCIM Resource Attributes - An extension function allows access to SCIM objects that are indirectly referenced from the XACML request context. This supports, for example, the ability to examine data through the SCIM reference attribute type as described in RFC 7643.

The syntax for accessing a referenced SCIM object is `ext:scimReference(String referencePath, String attributePath)` where `referencePath` specifies where in the XACML request context to find the SCIM reference string, and `attributePath` is a JSON path that selects content from the referenced SCIM object.

Consider a case where a User SCIM resource contains a `wallet` attribute that contains a list of references to Payment Method objects, each of which is itself a SCIM resource. Each payment method object in the list is identified by its canonical type (`credit`, `payPal`, `check`). So the value of `resource.wallet[type == "credit"]` might be `PaymentMethod/987654`, which is the ID of a Payment Method object. The ID itself is not of much use to a policy, however the attributes of the referenced Payment Method object are. For example, the following expression would return the expiration date of the user's credit card:

```
ext:scimReference('resource.wallet[type eq "credit"]', 'expirationDate')
```

Note

Policies are not able to use this method to resolve SCIM reference attributes whose value is an external or absolute URI.

Use obligations and advice

The XACML specification defines an obligation as a specified operation that should be performed by the Policy Enforcement Point (PEP) based on an authorization decision. Advice is additional information provided to the PEP based on a policy decision, and can be used by the requesting client to determine why access to a resource was denied. Each obligation or advice type takes zero or more arguments, and the value of each argument is specified as a JEXL expression. The PingDataGovernance Server provides the following obligation types.

SCIM resource requests

Exclude Obligation – Specifies an argument "`attribute-names`" that lists the attributes to be excluded from the response. Each attribute must be formatted using SCIM Attribute Notation.

Include Obligation – Specifies an argument "`attribute-names`" that lists the attributes to be included in the response. Each attribute must be formatted using SCIM Attribute Notation.

The following example creates an exclude obligation that will prevent the `userName` attribute from being returned with a resource:

```
$ bin/dsconfig create-policy-obligation \
  --type exclude-attributes \
  --policy-name myPolicy \
  --rule-name someRule \
  --obligation-name someName \
  --set "attribute-names:['userName']"
```

Filter obligation

When a SCIM search request is sent, each record returned from the underlying user store is passed through the PingDataGovernance Server's policy engine to determine whether the client is authorized to retrieve the returned resource. This can be expensive if there are a large number of results returned from the search.

The `add-filter` obligation can be used to give policy writers a more efficient way to restrict the results of a SCIM search operation. Rather than check each result against the server's retrieve policy, this obligation requires the SCIM implementation to pre-filter the results by appending additional filter elements to the search request before the search is executed.

Policies can return zero or more `add-filter` obligations, each of which must specify a syntactically correct SCIM 2.0 filter expression, as described in the SCIM 2.0 Protocol specification. If the original SCIM request included a filter, then the additional filters are ANDed with the original filter.

Each filter obligation takes a single argument, which is the string-valued filter expression. The Attribute Assignment ID is `filter`, and its value may be either a string or a string bag. If the value is a string-bag, then the SCIM implementation will contenate together each string in the bag, with a space character between each segment, to form a single filter expression.

The following example will only permit records with tenant `Id = 7`:

```
$ dsconfig create-policy-obligation \
  --type add-filter \
  --policy-name myPolicy \
  --rule-name someRule \
  --obligation-name pickAnyName \
  --set "filter:tenantId eq 7"
```

Policies and request processing

Resource requests from a client are evaluated by the policy rules configured for the PingDataGovernance Server. This section describes each type of policy request that may be made by the PingDataGovernance Server's policy enforcement points.

SCIM resource type policy evaluation

Each request to the SCIM endpoint explicitly specifies what action is being requested and on what resources. As a REST interface, SCIM uses the HTTP method, query parameters, method body, and URI path to specify request parameters.

All SCIM requests target a specific SCIM Resource Type. For example, a search targeted to `/scim/v2/Users` is executed against the Users SCIM endpoint. An update targeted to `/scim/v2/ConsumerUsers/9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` is executed against a user with ID `9f8a23-5f7ec932-55c4-347e-b757-ce74258ea9e6` in the Users SCIM Resource Type.

SCIM search request

A SCIM search request consists of a search filter and an optional specification of which attributes to return from each record that satisfies the filter definition. The SCIM Resource Type against which the search is to be conducted is derived from the relative URL path, such as `/scim/v2/Users`.

The policy request generated from a SCIM search request contains the following attributes.

SCIM search request attributes

Attribute ID/Content	Attribute Category	Attribute Value
`subject_id`	access_subject	Name of the requesting client, if it can be retrieved from the OAuth2 access token.
`action_id`	action	`search`.
`resource_id`	resource	Relative URL of the SCIM endpoint, such as `Users`.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

After the search is run against the SCIM Resource Type, it generates policy requests for each record returned in the results to determine whether the requesting client has permission to receive the record’s attributes. Each resource and attribute of each record is evaluated independently through a separate policy request to determine if it can be returned. Any resources or individual resource attributes that are denied by policy are omitted from the response. These subsequent policy requests are identical to a SCIM GET request.

Note

The number of search results that can be returned is limited by the SCIM Resource Type's `lookthroughLimit` property, due to the potential cost of checking each response against policy.

SCIM GET request

The following is contained in the authorization request generated for a SCIM GET request for a known resource.

SCIM GET request attributes

Attribute ID/Content	Attribute Category	Value
`subject_id`	access_subject	Name of the requesting client, if it can be retrieved from the OAuth2 access token.
`action_id`	action	`retrieve`.
`resource_id`	resource	Relative URL of the resource or sub resource to retrieve, such as `Users/12345` or `/Users/12345/consents`.
`<JSON Content>`	resource	SCIM object representation of the requested resource.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

The SCIM endpoint will perform the following actions based on the result of the XACML policy authorization request:

- If the result is `deny` – The resource is not returned to the client and an error is returned.
- If the result is `permit` – The initial attribute set to be returned to the client is determined. Since multiple policies and/or rules may be consulted to make the permit decision, it's possible that multiple obligations will be returned with the result. See Use obligations and advice. Include and exclude obligations are processed as follows:
  - All attributes specified in an exclude obligation are removed from the attribute set.
  - If there are include obligations, all attributes that are not specified by an include obligation are removed from the attribute set.
  - If no attributes remain in the attribute set, a 200 success response code is returned but with an empty `resource` object.

These rules for each result type are used for all resources returned from the SCIM endpoint.

SCIM POST request

The following is contained in the authorization request generated for a SCIM POST request.

SCIM POST request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client name.
`action_id`	action	`create`.
`resource_id`	resource	Relative URL of the SCIM Resource Type to be created, such as `Users` or of the SCIM sub resource to be created, such as `/Users/12345/consents`.
`<JSON Content>`	scim_request	SCIM request body of the resource or sub resource to be created.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

If the POST operation is permitted, the new resource is created and the new object is returned to the client. After the POST is complete, a second policy request is issued to determine which attributes of the updated record the client can receive in the response.

SCIM PATCH and PUT requests

PUT requests are internally converted into a PATCH operation, which is why they are handled the same way by policy. The following is contained in the authorization request generated for a SCIM PATCH or PUT request for a known resource.

SCIM PATCH request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client name.
`action_id`	action	`modify`.
`resource_id`	resource	Relative URL of the resource or sub resource to be modified, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	scim_request	The normalized SCIM PATCH request body.
`<JSON Content>`	access_token	Access token properties.
`<JSON Content>`	applicable_scope	Applicable scope objects.

If the PATCH or PUT operation is permitted, the resource is updated and returned to the client. The updated resource is then subject to the same read criteria in a GET request.

SCIM DELETE request

The following is contained in the authorization request generated for a SCIM DELETE request for a known resource.

SCIM DELETE request attributes

Attribute ID	Attribute Category	Value
`subject_id`	access_subject	The client application name.
`action_id`	action	`delete`.
`resource_id`	resource	Relative URL of the resource or sub-resource to be deleted, such as `Users/12345` or `/Users/12345/account`.
`<JSON Content>`	access-token	Access token properties.
`<JSON Content>`	applicable-scope	Applicable scope objects.

Policy Decision Point (PDP) endpoint

The PDP endpoint enables an external Policy Enforcement Point (PEP) to generate policy requests and send them directly to the PingDataGovernance Server for evaluation. The request is passed directly to the policy engine. The request can contain any standard attributes, PingDataGovernance Server custom attributes, or other attributes that may be required by custom policies. This endpoint requires that the client authenticate using HTTP basic authentication.

Policy engine request context

The policy request context contains the information that is available to the policy engine to make a decision. A request for authorization (OAuth2) will provide information that helps the policy engine determine whether or not a client should be granted or denied access to a scope. A request for resources will provide information that will help determine if the operations on attributes in the requested scopes can be performed.

The request context contains attributes directly passed by a client when making an authorization request to the policy engine. It is supplemented with additional attributes and JSON objects that are retrieved from the attribute categories. In order to make a policy decision, policies can reference any attribute or JSON object from the request context.

XACML attribute categories

All references from policy to objects that can be obtained from the request context are first identified by their XACML attribute category.

- `resource` – This standard category definition is always used to reference the object to which authorization is being requested. With a SCIM request, this is a SCIM resource whose type is determined by the SCIM request path. See Resource properties for details.
- `access_subject` – This standard category definition contains the client ID, on whose behalf the policy request has been made.

- `access_token` – This custom category provides access to properties of the access token that has been used to make the current request. It exposes the access token as a JSON object. See Processing access tokens for details.
- `http_header` – This custom category provides access to the HTTP headers of the incoming request.
- `http_query_param` - – This custom category provides access to the HTTP query parameters of the incoming request. The following example retrieves the value of the query parameter with name `channel`:

  ```
  http_query_param.channel
  ```
- `scim_request` – This custom category is populated by the PingDataGovernance Server SCIM endpoint and contains the JSON request body of the SCIM request that triggered policy evaluation. The content from this attribute category is in standard SCIM 2.0 format. See SCIM request properties for details.
- `applicable_scope` – This custom category is populated with the scopes from the access token that are applicable to authorize a resource request. See Applicable scopes for details.
- `token_owner` – For requests authorized with an access token, this custom category provides access to the SCIM resource of the owner of the access token. See Access token properties for details.

Other attribute categories can be defined by custom PIPs.

Standard attribute use

The following request attributes are specified by the XACML specification. Unless otherwise specified, these are always available in the PingDataGovernance Server's policy request context.

Standard attributes

Attribute URN	Attribute Category	Data Type	Description
subject_id	access_subject	string	Contains the name of the client that is submitting a policy request.
ip_address	access_subject	ipAddress	Contains the originating IP address of the client's authorization request. The availability and accuracy of this attribute is dependent upon the deployed PingDataGovernance Server's network environment. When available, the value is retrieved from the XFORWARDED_FOR header of the client's HTTP request. If that header is not available, the IP address returned may be that of the last proxy to send the request.
resource_id	resource	anyURI	Contains the ID of the resource being requested.

Attribute URN	Attribute Category	Data Type	Description
action_id	action	string	Contains the name of the action being requested.
current_time	environment	time	The time at which the PingDataGovernance Server began processing the current authorization request.
current_date	environment	date	The date on which the current authorization request is being processed.
current_dateTime	environment	dateTime	The date and time at which the PingDataGovernance Server began processing the current authorization request.

Custom XACML function

There is a single custom function implemented by the PingDataGovernance Server. This can be accessed with the JEXL extension function. See JEXL extension functions.

The `scimAttribute_subset` function is similar to the standard XACML string-subset function, except that the arguments are bags of SCIM attribute names using SCIM attribute notation as described in the SCIM specification. The custom function comprehends wildcard attribute specifications as supported in the `resourceAttributes` property of a PingDataGovernance Server OAuth2 scope.

For example, if the second set passed to this function contains the string `urn:mySchema:*`, and the first set contains `urn:mySchema:myAttribute`, the function may still return TRUE (the first set is considered to be a subset of the second).

SCIM resource properties

SCIM Resource Type resources are exposed as JSON objects that can be accessed from policy. The format of the JSON object is determined by the structure of the underlying resource and the mappings defined for its SCIM Resource Type. When a client makes a SCIM request, the resource category content is a SCIM Resource. For example, the following JEXL expression will retrieve the `region` sub-attribute of a user's home address within the requested User resource.

```
resource.addresses[type eq "home"].region
```

Scope properties

The default scope validation policy allows resource operations as long as one of the scopes granted in the access token allows the operation. Access to attributes allowed per operation is the union of all `resourceAttributes` defined in Authenticated Identity or Resource scopes that allow that operation.

For operations to be allowed on resources, the policies that process the requests must allow the operations requested in the scope. The following scope properties can be evaluated by policies.

Scope properties

Property	Data Type	Description
`tokenName`	String.	The scope name as presented in an OAuth2 request.
`type`	String.	The scope type, which is `authenticated-identity` for authenticated identity scopes, `resource` for resource scopes, or `oauth2` for a generic scope.
`tags`	String. Multivalued.	A list of Tags associated with a scope that can be examined by XACML policies.
`scimResourceType`	Aggregation.	If a `resource` scope, the SCIM Resource Type that can be accessed.
`resourceOperations`	Multivalued list. Optional.	Operations can include: • `create` (POST) to endpoint `/scim/v2` • `search` (GET) from endpoint `/scim/v2` • `retrieve` (GET) from endpoint `/scim/v2/<id>` • `replace` (PUT) to endpoint `/scim/v2/<id>` • `modify` (PATCH) to endpoint `/scim/v2/<id>` • `delete` (DELETE) from endpoint `/scim/v2/<id>`
`resourceAttributes`	Multivalued string.	A list of one or more SCIM attributes of the authenticated identity for which this scope allows access. The type of access is determined by the operation properties `retrieve`, `replace`, and `modify`. A wildcard value of `*` can be used for all attributes. A schema-specific wildcard value of the form `urn:<schemaName>:*` can be used to represent all attributes of a single schema namespace. Access to attributes allowed per operation is the union of all `resourceAttributes` allowed in the scope.

SCIM request properties

For policy evaluation of SCIM requests, the HTTP message body, if one exists, is available as the content of the `scim_request` attribute category. For SCIM POST requests, this content will be the JSON resource to be created, and for SCIM PATCH or PUT requests, the content will be a normalized SCIM PATCH request. For convenience, the attribute with ID `impacted_attributes` is also available. This attribute is computed by the policy engine and returns a list of attribute names in SCIM attribute notation. It returns only the attributes that can be created, modified, or deleted as a result of a SCIM POST, PUT, or PATCH request. See the SCIM 2.0 specification for more details.

The following JEXL example retrieves all impacted attributes from the current SCIM request:

```
scim_request.impacted_attributes
```

Applicable scopes

An access token presented by a client to the PingDataGovernance Server can contain many scopes, only some of which are applicable to the current request. The PingDataGovernance Server’s PIP exposes the applicable scopes under the attribute category `applicable_scope`. This category contains a list of JSON scope objects, described in OAuth2 scopes, for those scopes granted by the access token that meet the following criteria:

- The current request’s `action_id` is contained in one of the scope’s operations properties.
- The type of resource requested matches the type of resource to which the scope grants access. For Authenticated Identity scopes, they are only applicable to requests in which the resource requested is the access token owner.
- Generic OAuth2 scopes are always included since their meaning is not defined by the PingDataGovernance Server.

The following example retrieves all attributes that are granted access by all applicable scopes of the access token:

```
applicable-scope.scope.resourceAttributes
```

Access token properties

The PingDataGovernance Server's Policy Information Provider (PIP) exposes access tokens as JSON objects under the attribute category `access_token`. What is available in the token is determined by the token provider.

The following properties are common in an access token.

Access token properties

Property	Data Type	Description
`active`	Boolean. Required.	`True` if the token is valid, `false` if token is invalid or has expired.
`client_id`	String.	The ID of the client to which this token is granted.
`iss`	String.	The issuer of the token.
`sub`	String.	The unique identifier for the token owner. For user tokens, this will be the relative SCIM path to the user resource, such as `Users/123456789`. For Client Credentials (`app`) tokens, this property is not present.
`scope`	Multivalued string.	A list of scope names granted by this token.
`app`	String.	The name of the client for which this token was created. For application tokens, this value will be equal to `sub`.
`iat`	DateTime.	The date and time at which the token was created.
`exp`	DateTime.	The date and time at which the token will expire.
`nbf`	DateTime.	The time at which the token becomes valid.
`jti`	String.	The unique token identifier.

Access token properties

Property	Data Type	Description
`token_type`	String.	The type of token.
`username`	String.	The user name of the token owner (not present for application tokens).

Configure the Policy Service

Policies are managed by the Policy Service. The default conditions of the Policy Service can be viewed and changed with the `dsconfig` tool, or through the Managment Console **Authorization and Policies -> Policy Service**.

The **combining-algorithm** determines how decisions are made if multiple policies or policy sets are applied to a request for resources. The default for the Policy Service is `deny-overrides`, which specifies that a "deny" decision from a policy should take priority over a "permit" decision. The PingDataGovernance Server also supports `permit-overrides`, `deny-unless-permit`, and `permit-unless-deny`. See the *OASIS Committee Specification 01, eXtensible access control markup language (XACML) Version 3.0. August 2010* for details about each combining algorithm.

Add any custom logged policy request attributes, which enables additional request attributes to be included in the output of a Trace Log Publisher during policy evaluations. The URN of the XACML category ID and Attribute ID are required, in addition to the logger key.

Policy Information Providers

Policy Information Providers are used to retrieve attribute(s) from the Policy Information Point (PIP) during policy evaluation. This is an Advanced setting. See Standard attribute use and Custom attribute use for information about these attributes. The PingDataGovernance Server provides the following Policy Information Providers:

BuiltIn Policy Information Provider – Resolves policy attributes that are implemented by the PingDataGovernance Server.

SCIM Request Policy Information Provider – Resolves policy attributes whose value can be retrieved from an incoming SCIM request.

SCIM Resource Type Policy Information Provider – Resolves policy attributes whose value can be retrieved from a SCIM Resource Type configured on this PingDataGovernance Server instance.

Token Policy Information Provider – Resolves policy attributes whose value can be retrieved from an access token recieved by this PingDataGovernance Server instance.

PIP Evaluation Order

When multiple PIPs are defined, the evaluation order determines the correct provider to verify a specified policy attribute. Each PIP must have a unique evaluation value defined within a PingDataGovernance Server instance. PIPs with a smaller value are evaluated first.

Create policies

The Administrative Console, **Authorization and Policies -> Policies**, or the `dsconfig` tool can be used to create and manage policies.

SCIM Resource Policy Set – A container for policies that authorize requests for protected resources, including SCIM and UserInfo requests.

Scope Validation – Authorizes SCIM requests based on the scopes granted by the access token provided. The scope must also be configured to enable a requested action. See OAuth2 scopes for details.

Token Validation – Denies all SCIM resource requests that do not contain a valid access token.

The following would be used to create the Token Validation policy, which is one of the PingDataGovernance Server's default policies:

```
$ bin/dsconfig create-policy \
  --policy-name "Token Validation" \
  --set "combining-algorithm:deny-unless-permit"
```

```
$ bin/dsconfig create-policy-rule \
  --policy-name "Token Validation" \
  --rule-name "Validate Token" \
  --set "effect:permit" \
  --set "description:Permit access if token is present and valid." \
  --set "condition:access_token.active"
```

```
$ bin/dsconfig create-policy-rule \
  --policy-name "Token Validation" \
  --rule-name "Error Advice" \
  --set "effect:deny" \
  --set "description:Provide error message for denied token."
```

```
$ bin/dsconfig create-policy-advice \
  --type "denied-reason" \
  --policy-name "Token Validation" \
  --rule-name "Error Advice" \
  --advice-name "token error advice" \
  --set 'error:"invalid_token"' \
  --set 'error-description:"Access token is expired or otherwise invalid."'
```

Policies can also be created in the Administrative Console through **Authorization and Policies -> Policies**.

The following information is used to configure a policy .

- A unique name and optional description.
- If this policy needs a rule combining algorithm that overwrites the one specified in the Policy Service, specify one.
- An optional target request. This JEXL expression will filter the incoming authorization request to determine if the policy should be applied when authorizing the request. If a

target filter is specified here, it is applied in addition to the target applied by the policy set in which this resides.

- Add optional policy variable definitions. These are JEXL expressions including rule conditions, obligation expressions, and advice expressions.
- Add any policy advice expressions and error statements.
- Add any policy obligation expressions.

Note

Policies are not enabled until they are added to an enabled policy set.

Creating a policy set

A policy set is an ordered collection of policies that work together to perform a policy task. The policy set is a XACML-defined entity. The PingDataGovernance Server evaluates policy sets the same way it evaluates policies.

Creation of a policy set is the same as that of a policy. A policy set must be created from individual policies that have been configured in the PingDataGovernance Server.

Note

Policy sets can contain both policies and other policy sets. If the combining algorithm is ordered, a policy set may contain policies or other policy sets but not both. A policy set must not contain a direct or nested reference to itself.

Troubleshoot policies with traces

Policy decisions are frequently the result of a complex series of logical steps. Identifying the reason why a particular request is getting an unexpected result can be difficult. The PingDataGovernance Server can generate a trace of any policy decision, and log traces with in the File Based Trace Log Publisher with `dsconfig` or through the Administrative Console.

Note

Policy traces are logged in the File Based Trace Log Publisher. See Logs and log publishers.

A Policy Decision Trace is an XML document that is formatted like the XACML policies. It demonstrates the sequence of steps taken by the policy engine to come to a decision for a specific request. The elements of the trace parallel the policies, policy targets, and policy rules that are evaluated. The following are included:

- The first line of the log entry identifies the message type as `POLICY-DECISION-TRACE`.
- The parameters of the policy request being traced are listed, including the application, action, and resources.
- Following this is the trace itself, which is included in the `<DecisionTrace>` XML element.

The trace also includes entries for each policy, rule, and target evaluated during the decision process. Each entry contains a `result` XML attribute, which specifies the result of evaluating the corresponding policy element.

Troubleshoot denied access

Policies can issue advice expressions for any policy request that is denied. This passes additional information to the client as to the reason for denying access. The SCIM endpoint will look for error advice returned from the policy engine and include it in the error response generated for the client. If a policy denies a request without advice, the error response is `access_denied`.

The following error advice may be included in policy.

Policy error advice

Advice ID	Attribute ID	Value
`request_denied_reason`	`error`	Error identifier or code. For SCIM responses, this value will be used to populate the `scimType` error parameter.
	`error_description`	The value of the `error_description` parameter of an error response, or the detail parameter of a SCIM error response.

Chapter 6: Advanced configuration

The PingDataGovernance Server's non-user data consists of data in the server configuration. Generally, data in the server configuration define an individual PingDataGovernance Server instance, and can include its place in a server topology. Multiple server instances can be grouped in two ways to share or mirror configuration settings:

- Server Groups – Servers that are added to a server group in the global configuration can share configuration changes across the group, or not.
- Cluster – This is a topology management setting that enables a set of servers to be grouped by a functional purpose, and any change to one is mirrored to all. A master server verifies any configuration change before it is propagated to other servers in the group.

Note

All configuration objects and settings are described in the HTML Configuration Reference, which can be accessed from the Administrative Console or from the `<server-root>/docs/index.html` page. Information in this chapter highlights configuration of interest to a PingDataGovernance Server installation. For complete configuration options and details, see the Configuration Reference.

Topics include:

General server configuration

Data Governance Server advanced configuration

Configuring PingDataGovernance Server login pages

Topology management

General server configuration

There are tools and settings that are common across all Ping servers. These enable monitoring and managing the server, configuring and sending alerts and alarms, and managing the server's communication with clients. These configuration objects can be changed at the local server, with the option to apply changes to servers in a group.

Available configuration tools

There are several tools that can be used for server administration and maintenance in the `/bin` directory. The following is a sample of the command-line configuration tools:

Command-line tools

Tool	Description
backup	Run full or incremental backups on one or more PingDataGovernance Servers. This utility also supports the use of a properties file to pass predefined command-line arguments.
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
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.
create-initial-config	Create an initial PingDataGovernance Server configuration.
create-rc-script	Create a Run Control (RC) script that can be used to start, stop, and restart the PingDataGovernance Server on Unix-based systems.
dsconfig	View and edit the PingDataGovernance Server configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the PingDataGovernance Server and its associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM arguments and the `JAVA_HOME` environment variable.
encryption-settings	Manage the server encryption settings database.
ldapdelete	Perform an LDAP delete operation.
ldapcompare	Perform an LDAP compare operation.
ldapmodify	Perform LDAP modify, add, and modify DN operations in the PingDataGovernance Server.
ldappasswordmodify	Perform LDAP password modify operations in the PingDataGovernance Server.
ldapsearch	Perform LDAP search operations in the PingDataGovernance Server.

Command-line tools

Tool	Description
ldif-diff	Compare the contents of two LDIF files, the output being an LDIF file needed to bring the source file in sync with the target.
ldifmodify	Apply a set of modify, add, and delete operations against data in an LDIF file.
list-backends	List the backends and base DNs configured in the PingDataGovernance Server.
manage-extension	Install or update extension bundles. An extension bundle is a package of extension (s) that utilize the Server SDK to extend the functionality of the PingDataGovernance Server. Any added extensions require a server re-start.
oauth2-request	Performs OAuth2 requests on the PingDataGovernance Server. This tool can be used to test OAuth2 functions of the PingDataGovernance Server, and to manage OAuth2 tokens on behalf of registered applications.
prepare-external-store	Prepares the external PingDirectory Servers for the PingDataGovernance Server. This is run as part of the `create-initial-config` tool during installation. This tool creates the PingDataGovernance Server user account, sets the correct password, and configures the account with required privileges. It will also install the necessary schema required by the PingDataGovernance Server.
remove-defunct-server	Removes a permanently unavailable PingDataGovernance Server after it has been removed from its topology by the `uninstall` tool.
restore	Restore a backup of the PingDataGovernance Server.
review-license	Review and/or accept the product license.
server-state	View information about the current state of the PingDataGovernance Server processes.
start-server	Start the PingDataGovernance Server.
status	Display basic server information.
stop-server	Stop or restart the PingDataGovernance Server.
sum-file-sizes	Calculate the sum of the sizes for a set of files.

Use the dsconfig tool

The `dsconfig` tool, is used to view or edit the PingDataGovernance Server configuration, and is parallel in functionality with the Administrative Console. This utility can be run in interactive mode, non-interactive mode, and batch mode. Interactive mode provides an intuitive, menu-driven interface for accessing and configuring the server.

To start `dsconfig` in interactive mode, enter the following command:

```
$ bin/dsconfig
```

The `dsconfig` tool provides a batching mechanism that reads multiple `dsconfig` invocations from a file and executes them sequentially. The batch file advantage is that it minimizes LDAP connections and JVM invocations required with scripting each call. To use batch mode to read and execute a series of commands in a batch file, enter the following command:

```
$ dsconfig --bindDN uid=admin,dc=company,dc=com \
  --bindPassword password \
  --no-prompt \
  --batch-file </path/to/config-batch.txt>
```

The `logs/config-audit.log` file can be used to review the configuration changes made to the PingDataGovernance Server and use them in the batch file.

Administrative accounts

Users that authenticate to the Config API or the Administrative Console are stored in `cn=Root DNs,cn=config`. These users must exist on all instances of the PingDataGovernance Server to manage a Topology of servers. The `setup` tool automatically copies one administrative account when performing an installation from a peer, but if changed, the accounts must be synchronized. 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
   ```

Use the Configuration API

Ping 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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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:pingidentity: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."
    }
  ]
}
...
```

Updating 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:pingidentity: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 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

Response Code	Description	Response Body
		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."
}
```

Configuring HTTP connection handlers

The server relies on the HTTP connection handler, which relies on one or more servlet extensions. Servlet extensions are responsible for obtaining Java servlets and registering them to be invoked using one or more context paths. For custom servlet extensions created using the Server SDK, the process varies based on using a Java-based or Groovy-scripted extension. See the Server SDK documentation for details.

HTTP connection handlers are responsible for managing the communication with HTTP clients and invoking servlets to process requests from those clients. They can also be used to host web applications on the server. Each HTTP connection handler must be configured with one or more HTTP servlet extensions and zero or more HTTP operation log publishers.

If the HTTP Connection Handler cannot be started (for example, if its associated HTTP Servlet Extension fails to initialize), this does not prevent the entire server from starting. The server's start tool posts any errors to the error log.

The configuration properties available for use with an HTTP connection handler include:

- `listen-address` – Specifies the address on which the connection handler will listen for requests from clients. If not specified, then requests will be accepted on all addresses bound to the system.

- `listen-port` – Specifies the port on which the connection handler will listen for requests from clients. Required.
- `use-ssl` – Indicates whether the connection handler will use SSL/TLS to secure communications with clients (whether it uses HTTPS rather than HTTP). If SSL is enabled, then key-manager-provider and trust-manager-provider values must also be specified.
- `http-servlet-extension` – Specifies the set of servlet extensions that will be enabled for use with the connection handler. You can have multiple HTTP connection handlers (listening on different address/port combinations) with identical or different sets of servlet extensions. At least one servlet extension must be configured.
- `http-operation-log-publisher` – Specifies the set of HTTP operation log publishers that should be used with the connection handler. By default, no HTTP operation log publishers will be used.
- `ssl-cert-nickname` – In scenarios where the multiple public-private key pairs are in a JKS keystore, the LDAPConnectionHandler allows choosing a specific certificate alias through the `ssl-cert-nickname` property. The HTTPConnectionHandler for HTTPS connections should have the same option for parity.
- `key-manager-provider` – Specifies the key manager provider that will be used to obtain the certificate presented to clients if SSL is enabled.
- `trust-manager-provider` – Specifies the trust manager provider that will be used to determine whether to accept any client certificates presented to the server.
- `num-request-handlers` – Specifies the number of threads that should be used to process requests from HTTP clients. These threads are separate from the worker threads used to process other kinds of requests. The default value of zero means the number of threads will be automatically selected based on the number of CPUs available to the JVM.
- `web-application-extension`– Specifies the web applications to be hosted by the server.

For information about other connection handlers, see the *PingDataGovernance Server Configuration Reference Guide*.

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 servers)

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 Ping 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
```

Configuring traffic through a load balancer

If an Ping Identity 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 Ping server. For Ping 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.

On the load balancer, configure settings to provide the `X-Forwarded-*` information, such as `X-Forwarded-Host:`. See the product documentation for the device type.

System alarms, alerts, and gauges

Ping servers provide tools to monitor and manage the health of the system. The Data Governance Server provides delivery mechanisms (handlers) for administrative alerts using JMX or SNMP, in addition to standard error logging. All can be configured with the `dsconfig` tool.

Alerts and alarms reflect state changes within the server that may be of interest to a user or monitoring service. 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 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 gauges for CPU, disk, and memory usage 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.

The server is 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.

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 PingDataGovernance Server provides a number of alert handler implementations configured with the `dsconfig` tool or the Administrative Console, 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 through SNMP 2c.

If needed, the Server SDK can be used to implement additional, third-party alert handlers.

Test alarms and alerts

After gauges, 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.

Perform the following steps to test alarms and alerts:

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/2015 : Alarm [CPU Usage (Percent). Gauge CPU Usage (Percent)
         : 15:41:00    : for Host System Recent CPU and Memory has
         :  -0500      : 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 2015. 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/2015: 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 2015. 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
```

ConsPing Identityume admin alerts health check for PingDirectoryProxy Server

If the PingDataGovernance Server relies on a PingDirectoryProxy Server as a backend store, two health check instances are available to determine if the LDAP store adapter is reporting itself as degraded or unavailable, and if so, remove it from rotation:

- **Consume Admin Alerts**. This health check detects administrative alerts from the server, as soon as they are issued, by maintaining an LDAP persistent search for changes within the `cn=alerts` branch. When the PingDataGovernance Server is notified by the PingDirectoryProxy Server of a new alert, it immediately retrieves the base `cn=monitor` entry of the PingDirectoryProxy Server. If this entry has a value for the `unavailable-alert-type` attribute, then the PingDataGovernance Server will consider it unavailable. If this entry has a value for the `degraded-alert-type` attribute, then the PingDataGovernance Server will consider it degraded.
- **Get Root DSE**. This health check detects if the root DSE entry exists on the LDAP external server. As this entry always exists on an PingDirectoryProxy Server, the absence of the entry suggests that the LDAP external server may be degraded or unavailable.

Make the following configuration changes to the PingDirectoryProxy Server, if it is configured as an external store to the PingDataGovernance Server:

```
$ bin/dsconfig set-gauge-prop --gauge-name "Data Set Local Availability" \
  --set server-unavailable-severity-level:critical
```

```
$ bin/dsconfig set-gauge-prop --gauge-name "Data Set Availability" \
  --set server-unavailable-severity-level:critical
```

These changes modify two gauges to report the PingDirectoryProxy Server as unavailable if the gauges determine that the PingDirectoryProxy Server's data set is unavailable.

Logs and log publishers

Ping supports 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 default log files as well as mechanisms to configure custom log publishers with their own log rotation and retention policies.

Types of log publishers

Log publishers can be used to log processing information about the server, including:

- **Error loggers** – provide information about warnings, errors, or significant events that occur within the server.

- **Trace logger** – provides information about each HTTP, OAuth2, XACML policy, and SCIM request and response that is processed by the PingDataGovernance Server.

View and configure log publishers

Log publishers can be created or modified on each server using the `dsconfig` tool or through the Administrative Console, **Logging, monitoring, and notifications -> Log Publishers**.

Create a new log publisher

Ping provides customization options to create log publishers with the `dsconfig` command or through the Administrative Console.

After creating a new log publisher, configure the log retention and rotation policies. For more information, see Configuring Log Rotation and Configuring Log Retention.

The following example shows how to create a trace logger that collects debug information for HTTP, external identity provider, XACML policy, and store adapter operations with the `dsconfig` command:

```
$ bin/dsconfig create-log-publisher \
  --publisher-name NewTraceLogger \
  --type file-based-trace \
  --set enabled:true \
  --set debug-message-type:external-identity-provider-request-and-response \
  --set debug-message-type:http-full-request-and-response \
  --set debug-message-type:policy-decision-trace \
  --set debug-message-type:store-adapter-processing \
  --set http-message-type:request \
  --set http-message-type:response \
  --set xacml-policy-message-type:result \
  --set 'exclude-path-pattern:/**/*.css' \
  --set 'exclude-path-pattern:/**/*.gif' \
  --set 'exclude-path-pattern:/**/*.jpg' \
  --set 'exclude-path-pattern:/**/*.png' \
  --set log-file:myfile \
  --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 compression-mechanism:gzip
```

Compression cannot be disabled or turned off once configured for the logger. Determine logging requirements before configuring this option.

Configure log compression

Ping 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'Succes
sfully wrote 123,456,789 bytes of decrypted data
```

Configure log signing

Ping 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 Trace Log Publisher.

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

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Trace 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/trace
   ```

   ```
   All signature information in file 'logs/trace' 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/trace':
   * 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

Ping 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 you want the sever 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.
- **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 Error 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 Error Logger" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

Server monitoring

While the server is running, it generates a significant amount of information available through monitor entries. This section contains information about the following:

- Backend monitor entries
- View system and consent data through the PingDataMetrics Server
- Use the status tool

Backend monitor entries

Each Ping server exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers through SNMP, LDAP command-line tools, and the Stats Logger.

The Monitor Backend contains an entry per component or activity being monitored. The list of all monitor entries can be seen using the `ldapsearch` command as follows:

```
$ bin/ldapsearch --hostname server1.example.com \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --baseDN "cn=monitor" "(objectclass=*)" cn
```

The following table lists a subset of monitor entries.

Monitoring components

Component	Description
Active Operations	Provides information about the operations currently being processed by the server including the number of operations, information on each operation, and the number of active persistent searches.
Backends	Provides general information about the state of a server backend, including the entry count. If the backend is a local database, there is a corresponding database environment monitor entry with information on cache usage and on-disk size.
Client Connections	Provides information about all client connections to the server including a name followed by an equal sign and a quoted value, such as `connID="15", connectTime="20100308223038Z"`.
Connection Handlers	Provides information about the available connection handlers on the server including the LDAP and LDIF connection handlers.
Disk Space Usage	Provides information about the disk space available to various components of the server.
General	Provides general information about the state of the server, including product name, vendor name, and server version.
Index	Provides information on each index including the number of preloaded keys and counters for read, write, remove, open-cursor, and read-for-search actions. These counters provide insight into how useful an index is for a given workload.
HTTP/HTTPS Connection Handler Statistics	Provides statistics about the interaction that the associated HTTP connection handler has had with its clients, including the number of connections accepted, average requests per connection, average connection duration, total bytes returned, and average processing time by status code.
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 read and written, and operations initiated, completed, and abandoned.
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 (ms), 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 system host name, operation system, JVM architecture, Java home, and Java version.
Version	Provides information about the server version, including build ID, and revision number.
Work Queue	Provides information about the state of the server work queue, which holds requests until they can be processed by a worker thread, including the requests rejected, current work queue size, number of worker threads, and number of busy worker threads. 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 the log messages. Its value is expressed in milliseconds and represents the length of time that operations are held in the work queue.

View system and consent data through the PingDataMetrics Server

The PingDataMetrics Server contains several charts to measure and monitor PingDataGovernance Server system and user consent activity. Charts and data are configured from the PingDataMetrics Server Server. The following categories can be made available through a PingDataMetrics Server dashboard:

Authorization Requests – Displays the number of blocked and permitted token requests from client applications.

Request Volume – Displays authorization activity according to grant or deny.

Grant Types – Displays the number of authorization grants by type.

Consent/Deny by Application – Displays authorization activity based on client application.

Consent/Deny by Data Type – Displays authorization activity based on data type.

Most Requested Data – Displays most requested data.

Most Active Applications – Displays most active client applications.

Most Active Policies – Displays most active policies.

See the *Ping PingDataMetrics Server Administration Guide* for more information.

Using the status tool

Ping servers provide the `status` tool, which lists 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.

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 the PingDirectory Server, PingDirectoryProxy Server, PingDataMetrics Server, PingDataGovernance Server, and PingDataSync Server 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

Extensions for the PingDataGovernance Server include:

- Policy Information Provider
- Store Adapter

PingDataGovernance Server Advanced Server Configuration

When a PingDataGovernance Server is set up from a peer, its server configuration is cloned to the new PingDataGovernance Server, and the two configurations are linked such that changes to the configuration are applied to both PingDataGovernance Server servers by default. See Installing a Clone PingDataGovernance Server. If a server is installed in an existing topology (an installation option), the server configurations are also linked.

The server's configuration is stored in an LDIF-based backend under the `cn=config base DN`. It can be accessed using the LDAP protocol and is managed by the `dsconfig` tool, Configuration API, or the Administrative Console.

Configuring Third-Party Store Adapters

Third-party adapters can be created for directory servers, that are not the Ping PingDirectory Server, with the Server SDK available in the `unboundid-server-sdk-<version>.zip` package.

Configuring a custom store adapter includes the following steps:

1. Create a store adapter.
2. Store it in the `/extensions` directory of the PingDataGovernance Server.

3. Create a SCIM Resource Type schema.
4. Map Store Adapter(s) and SCIM Resource Types using the Administrative Console or `dsconfig` tool.

Example Third-Party Store Adapter

The Server SDK provides an example implementation of a third-party store adapter. View the example and associated Javadocs in the Server SDK `docs/example-html/ExampleStoreAdapter.java.html` directory.

`ExampleStoreAdapter.java` is an implementation of a flat-file JSON store adapter, which stores the SCIM user data in JSON. At startup, all resources are loaded from the `json-file-path` parameter (`resource/user-database.json`). The example uses an in-memory hash map of SCIM resources mapped to their SCIM ID.

The example provides full operations plus filterable search support for add, update, and deletes. The example will perform a full-file rewrite on every change, because the file format is a serialized list of `Resources<BaseResource>`. The code example does not support sorting or resource versioning.

Cross-Origin Resource Sharing Support

Cross-Origin Resource Sharing (CORS) enables client applications to make JavaScript requests to the PingDataGovernance Server (or PingDirectory Server) by specifying the domain from which the request is made.These cross-domain requests are generally not allowed by web browsers without CORS support. CORS defines a way in which the browser and the server can interact to determine whether a request is coming from a trusted domain.

CORS Implementation

CORS is implemented per HTTP servlet extension. Access is governed by HTTP Servlet Cross Origin Policies defined through the `dsconfig` tool. Trusted domains can be added to these policies or defined with registered applications in the Administrative Console or with the `dsconfig` tool.

Note

By default, HTTP servlet extensions do not have CORS defined. Without a CORS policy defined, the configuration of the browser will determine application access.

The following are configuration options in `dsconfig`:

```
>>>> HTTP Servlet Cross Origin Policy menu

What would you like to do?

1)  List existing HTTP Servlet Cross Origin Policies
2)  Create a new HTTP Servlet Cross Origin Policy
3)  View and edit an existing HTTP Servlet Cross Origin Policy
4)  Delete an existing HTTP Servlet Cross Origin Policy

b)  back
```

```
q)  quit

Enter option [b]:
```

HTTP servlet services

Enabling CORS for a particular servlet can impact another service provided by the same servlet. It is important to know which services will be affected when enabling CORS for an PingDataGovernance Server servlet. The following are available servlets and their functions.

Servlet	Functions
API Explorer Servlet	Manages requests to the API Explorer, which enables testing PingDataGovernance Server functions.
Authentication Servlet	Manages requests to the `/authentication` API endpoint (used by the `auth-ui`).
Configuration	Used to enable read and write access to the server's Configuration API.
Documentation	Manages requests for the `/docs` content, which includes the `index.html` page, the generated Configuration Reference Guide, and other product documents.
JWK Servlet	Provides access to the JSON Web Key for token validation.
OAuth2 Servlet	OAuth2 authorization, token, revocation, and validation endpoints.
Policy Decision Point Servlet	XACML PDP endpoint.
SCIM2	Profile access by SCIM Resource Type using SCIM.
UserInfo Servlet	Profile access using OpenID Connect.

Note

Any servlet accepting JavaScript calls from client applications that are hosted at a different location than that of the Data Governance Server APIs, such as the Velocity servlet, must have CORS enabled.

HTTP servlet cross origin policies

Two sample policies are available after installation. They can be associated with a servlet extension, or used as templates for additional policies.

Per-Application Origins – This policy trusts origins that are listed as trusted by applications registered with the PingDataGovernance Server.

Restrictive – This policy rejects all cross-origin requests unless explicitly defined with the `cors-allowed-origins` property. Requests from application origins that are not specified are rejected with a `403 Forbidden` return code.

Each policy accepts values for the following properties.

Property	Description
`cors-enabled`	Specifies if the CORS protocol is allowed by the servlet. The default value is `false`.
`cors-allowed-methods`	Specifies the list of HTTP methods allowed for access to resources. The

Property	Description
	default value is `GET`.
`cors-enable-per-application-origins`	Specifies that a per-application list of allowed origins is consulted. The default value is `false` in the Restrictive policy and `true` in the Per-Application Origins policy.
`cors-allowed-origins`	Specifies a global list of allowed origins. If the `cors-enable-per-application-origins` property is set to `true`, and there are origins listed here, this list is consulted in addition to the per-application list. A value of "*" specifies that all origins are allowed. The default is an empty list.
`cors-exposed-headers`	Specifies a list of HTTP headers that browsers are allowed to access. Simple response headers, as defined in the Cross-Origin Resource Sharing Specification, are allowed. The default is an empty list.
`cors-allowed-headers`	Specifies the list of header field names that are supported for a resource and can be specified in a cross-origin request. The default values are `Origin`, `Accept`, `X-Requested-With`, `Content-Type`, `Access-Control-Request-Method`, and `Access-Control-Request-Headers`.
`cors-preflight-max-age`	Specifies the maximum number of seconds that a preflight request can be cached by the client. The default value is `1800` (30 minutes).
`cors-allow-credentials`	Specifies whether requests that include credentials are allowed. This value should be `false` for servlets that use OAuth2 authorization. The default value is `false`.

Assigning a CORS policy to an HTTP servlet extension

CORS policies are assigned to HTTP servlet extensions through `dsconfig`.

The following are configuration options for the SCIM servlet extension:

```
>>>> Configure the properties of the SCIM Resource Type SCIM HTTP Servlet Extension
Property                Value(s)
----------------------------------------------------------------------

1)  description          -
2)  cross-origin-policy  No cross-origin policy is defined and no CORS headers are
recognized or returned.
3)  base-context-path    /scim

?)  help
f)  finish - apply any changes to the SCIM Resource Type SCIM HTTP Servlet Extension
a)  show advanced properties of the SCIM Resource Type SCIM HTTP Servlet Extension
d)  display the equivalent dsconfig command lines to either re-create this object or only
to apply pending changes
b)  back
q)  quit

Enter option [b]: 2
```

Choose the `cross-origin-policy` option. Defined policies are listed.

```
>>>> Configuring the 'cross-origin-policy' property
The cross-origin request policy to use for the HTTP Servlet Extension.

A cross-origin policy is a group of attributes defining the level of cross-origin request
supported by the HTTP Servlet Extension.

Do you want to modify the 'cross-origin-policy' property?

1)  Keep the default behavior: No cross-origin policy is defined and no CORS headers are
recognized or returned.
2)  Change it to the HTTP Servlet Cross Origin Policy: Per-Application Origins
3)  Change it to the HTTP Servlet Cross Origin Policy: Restrictive
4)  Create a new HTTP Servlet Cross Origin Policy

?)  help
q)  quit
```

Choose the CORS policy to assign to this servlet extension.

Public and private key store configuration

The PingDataGovernance Server server can be configured to validate access tokens with a private key and expose a public key to enable client applications to read the content of the tokens. If there are multiple PingDataGovernance Servers in an environment, a key-pair created on one server will automatically be mirrored on all other servers. The PingDataGovernance Server supports RSA key pairs.

A certificate key pair can be created by or imported to the server with the `dsconfig` tool, or through the advanced setting **System -> Key Pairs** in the Administrative Console. For example, the following command can be used to create a new key pair:

```
$ bin/dsconfig -n create-key-pair --pair-name jwt2
```

When a key-pair is created or imported, the private key is encrypted by the preferred encryption settings definition in the encryption settings database and a Certificate Signing Request attribute is created. The private key and Certificate Signing Request are read-only properties, but not the certificate chain. The public key is wrapped in the certificate chain.

The Certificate Signing Request can be taken to a Certificate Signing Authority to obtain a signed, public key certificate. This can then be imported with `dsconfig` to replace the self-signed certificate.

Note

The PingDataGovernance Server does not automatically rotate expired keys. If using self-signed certificates, reset the `certificate-chain` property when needed. This will regenerate a new self-signed certificate with the specified validity (`self-signed-certificate-validity`). If using signed certificates, renew the certificate (extend its validity) from the Certificate Signing Authority and set the `certificate-chain` property in the key-pair.

Long keys may require more CPU for processing and affect performance, if request volume is high.

Managing server encryption settings

The server encryption settings database is managed by the `encryption-settings` command-line tool. The keys stored for the server are used to encrypt tokens, authorization codes, account linking codes, and external identity provider tokens. Encryption settings definitions can be created, listed, exported and imported. Help and examples are available with the following command:

```
$ bin/encryption-settings --help
```

Information about the cipher algorithms and transformations available for use is located in the *Java Cryptography Architecture Reference Guide* and *Standard Algorithm Name Documentation* available on the Oracle website.

Rotating the encryption key

Perform the following steps for routine rotation of the encryption key:

1. Create a new encryption settings definition.

   ```
   $ encryption-settings create \
     --cipher-algorithm AES \
     --key-length-bits 128
   ```

   ```
   Successfully created a new encryption settings definition with ID <ID>
   ```

2. Verify the new definition was created.

   ```
   $ encryption-settings list
   Encryption Settings Definition ID: <old-key>
     Preferred for New Encryption: true
     Cipher Transformation: AES
     Key Length (bits): 128

   Encryption Settings Definition ID: <ID>
     Preferred for New Encryption: false
     Cipher Transformation: AES
     Key Length (bits): 128
   ```

3. Create a PIN file that will be used for the exported definition.

   ```
   $ echo "secret" > /tmp/exported-key.pin
   ```

4. Export the encrypt settings, referring to the generated encryption settings ID.

   ```
   $ encryption-settings export \
     --id <ID> \
     --output-file /tmp/exported-key \
     --pin-file /tmp/exported-key.pin

   Successfully exported encryption settings definition <ID> to file
   /tmp/exported-key
   ```

5. For every Data Governance Server instance in the topology, copy the exported definition and PIN file to the PingDataGovernance Server's host. Import the encryption settings, without setting them as preferred. Delete the exported settings and PIN file when finished.

   ```
   $ encryption-settings import \
     --input-file /tmp/exported-key \
     --pin-file /tmp/exported-key.pin

   Successfully imported encryption settings definition <ID> from file
   /tmp/exported-key
   ```

   ```
   $ rm /tmp/exported-key
   $ rm /tmp/exported-key.pin
   ```

6. Perform the previous steps for all existing key pairs, as private keys will still be encrypted with the previous preferred encryption definition. Delete the existing key pairs and re-import them (which will automatically use the new preferred encryption definition for the private key).

7. After importing the encryption settings definition to all PingDataGovernance Servers, including the instance where the definition was originally created, set the new definition as preferred.

   ```
   $ encryption-settings set-preferred \
     --id <ID>

   Encryption settings definition <ID> is was successfully set as the
   preferred definition for subsequent encryption operations.
   ```

Address a compromised encryption key

If an encryption settings definition becomes compromised, perform the following to create a new definition and update the PingDataGovernance Server servers. See the command line help for the `encryption-settings` tool for arguments.

Note

If the PingDataGovernance Server's encryption key is compromised, and the PingDataGovernance Server has been collecting access tokens for external identity providers through the relying party feature, make sure those tokens are revoked.

1. Back up the encryption settings backend.

2. Back up the user store.

3. Revoke all authorizations for each client.

4. Stop the HTTPS Connection Handler that is used for the Data Governance Server's REST APIs.

   ```
   $ dsconfig set-connection-handler-prop \
     --handler-name "HTTPS Connection Handler" \
     --set enabled:false
   ```

5. Create a new encryption settings definition and set it as preferred. The following will encrypt data using a 128-bit AES cipher:

```
$ encryption-settings create \
  --cipher-algorithm AES \
  --key-length-bits 128
  --set-preferred
```

6. Restart the HTTPS Connection Handler.

```
$ dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --set enabled:true
```

If the deployment includes multiple PingDataGovernance Servers, all servers should be taken offline, and the encryption settings database must be updated on every server.

Note

Do not delete the compromised encryption definition. It will still be used to decrypt tokens, authorization codes, and links that were encrypted with the previous key.

Customizing the Authentication User Interface

The PingDataGovernance Server interface is implemented as a client-side Angular 2 application without a backend server component. It is written using TypeScript and JavaScript. The project's build process leverages `node` and `npm` (like the Administrative Console), and is packaged as a WAR file. See the Angular 2 documentation for more details about tools and customization.

The PingDataGovernance Server application is deployed as a Web Application Extension with a base-context path of `/auth-ui`. The `auth-ui` source code is shipped with the PingDataGovernance Server in the `auth-ui-source.tar.gz` file in the `/webapps` directory. This can be extracted into a directory on a development machine for customization. There are additional details included in a readme file.

Most of the `npm` scripts defined in the `auth` project's `package.json` file are subcommands used by the top-level scripts `dev`, `test` and `prod`.

Example usage:

```
npm run [dev | test | prod]
```

Note

The PingDataGovernance Server's Authentication API uses a cookie to track user sessions. Cookie management and server domains should be considered when deploying any clients that will use the Authentication API.

The `auth-ui` implementation uses the `/oauth/authorize` and the `/authentication/*` APIs through AJAX to implement the following views and flows:

- Consent prompts
- Error messages

- Login fields and options
- Second-factor authentication
- Recover username or password
- Register new user account
- IDP-callback

Branding

The `auth-ui` interface styling comes from the `assets/css/ubid-account.css` file. To override its styles, either this file can be edited directly, or an additional CSS override file can be added to the project and included in the `copy-assets` script in `package.json`. For example:

1. Add a file called `shopco.css` to the `assets/css` directory.
2. Add the following to the file:

   ```
   .login-div {
     background-color: #222;
   }
   .login-container a,
   .login-container a:hover {
     color: #e15656;
   }
   ```

3. Change the `package.json` file's `copy-assets` script to include the new file in the CSS by replacing this:

   ```
   cleancss -o ../dist/css/ubid-account.min.css css/ubid-account.css
   ```

   with this:

   ```
   cleancss -o ../dist/css/ubid-account.min.css css/ubid-account.css
   css/shopco.css
   ```

Schema changes

The `auth-ui` implementation assumes the sample reference schema is being used. To change the reference schema, surface additional attributes, or use another schema, the `auth-ui` project will need to be modified.

The following example adds the `urn:pingidentity:schemas:sample:profile:1.0:birthDate` attribute from the sample reference schema to the registration form:

1. Edit the `app/register/register.html.ts` file in the `auth-ui` project.
2. Add the following after the “Mobile Number” field’s `form-group` element:

   ```
   <div class="form-group">
     <label for="birthDate" class="control-label">Birth Date</label>
       <input
   [(ngModel)]="resource
   ```

```
['urn:pingidentity:schemas:sample:profile:1.0:birthDate']"
     type="date" class="form-control input-sm" name="birthDate"
     placeholder="Birth Date" tabindex="9">
</div>
```

3. Optionally disable the customization warning message in `app/register/register.component.ts` by replacing this:

   ```
   isExpectedRegistrableAttributes = (registrableAttributes &&
   registrableAttributes.length === 5 &&
   registrableAttributes.indexOf('userName') !== -1 &&
   registrableAttributes.indexOf('name') !== -1 &&
   registrableAttributes.indexOf('password') !== -1 &&
   registrableAttributes.indexOf('emails[type eq "home"].value') !== -1 &&
   registrableAttributes.indexOf('phoneNumbers[type eq "mobile"].value')
   !== -1);
   ```

   with this:

   ```
   isExpectedRegistrableAttributes = true;
   ```

4. On the PingDataGovernance Server development server, add `birthDate` to the `register-resource-attribute` for the Registration Identity Authenticator with the following `dsconfig` command:

   ```
   $ bin/dsconfig set-identity-authenticator-prop \
     --authenticator-name Registration \
     --add register-resource-
   attribute:urn:pingidentity:schemas:sample:profile:1.0:birthDate
   ```

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.

Servers can be added to an existing topology at installation. See Add an additional PingDataGovernance Server to a topology for details.

Note

To remove a server from the topology, it must be uninstalled with the uninstall tool. See Uninstalling thePingDataGovernance Server for details.

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 be 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 configuration. 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-----

MIIDKTCCAhGgAwIBAgIEacgGrDANBgkqhkiG9w0BAQsFADBFMR4wHAYDVQQKExVVbmJvdW5kSUQgQ2VydGlmaWNhd
GUxIzAhBgNVBAMTGnZtLW1lZGl1bS03My51bmJvdW5kaWQubGFiMB4XDTE1MTAxMjE1MzU0OFoXDTM1MTAwNzE1Mz
U0OFowRTEeMBwGA1UEChMVVW5ib3VuZElEIENlcnRpZmljYXRlMSMwIQYDVQQDExp2bS1tZWRpdW0tNzMudW5ib3V
uZGlkLmxhYjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKN4tAN3o9Yw6Cr9hivwVDxJqF6+aEi9Ir3W
GFYLSrggRNXsiAOfWkSMWdIC5vyF5OJ9DlIgvHL4OuqP/YNEGzKDkgr6MwtUeVSK14+dCixygJGC0nY7k+f0WSCjt
IHzrmc4WWdrZXmgb+qv9LupS30JG0FXtcbGkYpjaKXIEqMg4ekz3B5cAvE0SQUFyXEdN4rWOn96nVFkb2CstbiPzA
gne2tu7paJ6SGFOW0UF7v018XY1m2WHBIoD0WC8nOVLTG9zFUavaOxtlt1TlhClkI4HRMNg8n2EtSTdQRizKuw9Dd
TXJBb6Kfvnp/nI73VHRyt47wUVueehEDfLtDP8pMCAwEAAaMhMB8wHQYDVR0OBBYEFMrwjWxl2K+yd9+Y65oKn0g5
jITgMA0GCSqGSIb3DQEBCwUAA4IBAQBpsBYodblUGew+HewqtO2i8Wt+vAbt31zM5/kRvo6/+iPEASTvZdCzIBcgl
etxKGKeCQ0GPeHr42+erakiwmGDlUTYrU3LU5pTGTDLuR2IllTT5xlEhCWJGWipW4q3Pl3cX/9m2ffY/JLYDfTJao
JvnXrh7Sg719skkHjWZQgOHXlkPLx5TxFGhAovE1D4qLVRWGohdpWDrIgFh0DVfoyAn1Ws9ICCXdRayajFI4Lc6K1
m6SA5+25Y9nno8BhVPf4q5OW6+UDc8MsLbBsxpwvR6RJ5cv3ypfOriTehJsG+9ZDo7YeqVsTVGwAlW3PiSd9bYP/8
yu9Cy+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”
```

Index

A

access token 42
- accepting all tokens 43
- accepting external tokens 42
- accepting PingFederate tokens 42

access token properties 59

administrative account
- adding a root user account 67

Administrative Console
- login account 18
- URL 19

administrative password 67

alerts
- Directory Proxy availability 87

attribute mappings 32
- authoritative attribute 32, 38
- complex attributes 38
- described 29
- indexing 37
- mapping in SCIM Resource Types 37

authoritative attribute 32

B

backend monitors
- entries 92

backup tool 65

base DN
- configure Directory Server 14
- configure user entries 18

base64 tool 65

C

client-specific SCIM attributes 38

collect-support-data tool 65

config-diff tool 65

Config File Handler Backend 105

correlation attribute 30

CORS
- configuration 96

create-initial-config 17

create-rc-script tool 65

D

Data Governance Server
- attribute filtering 2-3
- deployment considerations 3
- described 1
- features 2
- in a topology 19
- installing 16
- pluggable authentication 2
- tools 65

Directory Server
- described 29
- installing 13

DNS caching 82

dsconfig
- changing policy-combining algorithm 48
- CORS configuration 96
- described 66
- tool described 65

dsframework tool 65

dsjavaproperties tool 65

dstat
  installing on SuSE Linux 11
E
encryption-settings tool 12, 65
encryption keys 12, 101
endpoints
  SCIM 40
error logger 87
F
filter obligation 51
G
Global Configuration object 106
H
health check
  consume admin alerts 87
  get root DSE 87
HTTP Servlet Cross Origin Policy 97
HTTP servlet extension 98
I
installing
  prerequisites 9
  scripted install 21
inter-server-certificate property 110
IP address reverse name lookup 83
J
Java
  installing the JDK 12
  supported versions 9
JEXL
  identifiers and variables 49
  policy structure details 48
JSON 32
JVM memory allocation
  Data Governance Server 17
  Directory Server 15
JWT 42
JWT token validation 43
K
key pair configuration 99
L
ldapmodify tool 65
ldappasswordmodify tool 65
LDAPS
  configure Data Governance Server 16
  configure Directory Server 14
ldapsearch tool 65
ldif-diff tool 66
ldifmodify tool 66
license key 12
Linux configuration
  filesystem swapping 10
  set file descriptor limit 9
list-backends tool 66
load balancers 84
logging
  available log publishers 87
  configure log retention and rotation 91
  configure log signing 90
  create log publisher 88
  encrypt log files 89
  log compression 88
login account 18
M
manage-certificates tool 108
manage-extension tool 66

metrics
  viewing 94
monitoring entries 93

O

oauth2-request tool 66
OAuth2 clients
  enable client-specific SCIM attributes 38
Open Banking 2

P

PDP endpoint 55
PingFederate access token validator 42
policy
  authorization scenarios 48
  decision trace 62
  described 44
  managing 61
  obligations and advice 51
  PDP endpoint 55
  policy information providers 60
  policy structure 48
  request processing 52
  troubleshoot denied access 63
  viewing policy metrics 94
  XACML 47
policy set
  creating 62
prepare-external-store tool 18, 66

R

remove-defunct-server tool 27, 66
resource operations
  scope properties 45, 58
Resource Scope 46
REST API
  connection port 16
  endpoints 42
restore tool 66
review-licence tool 66
root user DN 18-19

S

SCIM
  DELETE operations 54
  described 40
  GET operations 53
  PATCH and PUT operations 54
  policy request with obligation 51
  POST operations 53
  search request 52
SCIM endpoint 40
  described 29
  search considerations 32
SCIM request properties 58
SCIM resource properties 57
SCIM Resource Type
  creating 34
  managing 33
  REST API endpoints 42
  store adapter mapping 37
SCIM schema
  overview 32
  username uniqueness 33
scopes
  applicable scopes returned 59
  Authenticated Identity scope 47
  managing 47
  Resource scope 46-47

- scope types and properties 45
- self-signed certificate
  - replacing 108
- server shutdown 83
- SSL certificate 107
- start-server 66
  - running in the foreground 23
- status tool 66, 94
- stop-server 66
  - example of 23
  - in-core restart 24
- store adapter
  - correlation attribute 30
  - mapping attributes 32
  - primary and secondary adapters 29
  - search considerations 32
  - third-party store adapters 95
- store adapters
  - third-party 96
- sum-file-sizes tool 66
- supported platforms 9
- system entropy 11

T

- Third-Party Store Adapter 96
- token validation
  - signature validation 43
- topology
  - force master setting 106
  - inter-server-certificate property 110
  - master selection 104
  - monitor data 106
  - overview 104
  - replace self-signed certificate 110
  - server configuration settings 105
  - subtree polling interval 105
  - update servers 25
  - update SSL certificate 107
- topology configuration
  - update SSL Certificate 108
- trace logger 88
- trace policy decisions 62
- troubleshooting
  - server shutdown 83
  - SSL 83

U

- uid 33
- uninstall tool 27
- update tool 25
- user processes
  - configuring on Redhat/CentOS 10
- user store 12-13

W

- Windows service
  - configuration 24
  - deregister and uninstall 25
  - log files 25

X

- X-Forwarded values 84

PingDataGovernance™

Release 7.0.1

Platform Support Matrix

	PingDirectory 7.0 and PingDataGovernance 7.0
Operating Systems	
	RedHat 6.6
	RedHat 6.8
	RedHat 6.9
	RedHat 7.3
	RedHat 7.4
	CentOS 6.8
	CentOS 6.9
	CentOS 7.3
	CentOS 7.4
	SUSE Enterprise 11 SP4
	SUSE Enterprise 12 SP3
	Ubuntu 16.04 LTS
	Amazon Linux
	Windows Server 2012 R2
	Windows Server 2016
Virtual Platforms	
	VMWare vSphere & ESX 6.0
	KVM
	Amazon EC2
	Microsoft Azure (Supported by Professional Services)
	VMs must follow the same minimum system requirements as non-VM deployments. Specifically, the number of CPUs and available RAM must not be constrained. Also note that VM deployments with untuned, SAN-based file systems will not perform well.
	** The AWS Elastic File System is not supported due to its long latency times*
	** Standard support agreements do not cover performance issues related to the virtual platform.*
	** Standard support agreements DO cover performance issues that can be reproduced on a non-virtual platform*
	** Standard support agreements DO cover functionality issues on supported virtual platforms, as long as the OS and JVM/JDK versions are supported*
Java Versions	
	OpenJDK 8.x
	Oracle JDK 8.x 64-bit
App Servers	
	Apache Tomcat 7.x
	Apache Tomcat 8.x
	WildFly 9.x
Admin Browsers	
	Support for versions released in 2013 or later:
	Chrome 24+
	Microsoft Edge 20+
	Safari 4+
	Firefox 25+
	For Microsoft Internet Explorer, only version 11 is supported
End User Browsers	
	Support for versions released in 2013 or later:
	Android 4.1+
	Chrome 24+
	Microsoft Edge 20+
	Safari 4+
	Firefox 25+
	For Microsoft Internet Explorer, only version 11 is supported