

PingDataGovernance™

Release 7.3.0.3

Server Administration Guide



Contents

PingDataGovernance™ Product Documentation.....	5
Introduction to PingDataGovernance Server.....	5
Key components.....	5
What's new.....	6
Explore PingDataGovernance Server.....	6
System entropy.....	7
About the tools.properties file.....	7
System requirements.....	7
Platforms.....	7
Docker.....	8
Java Runtime Environment.....	8
Browsers.....	8
Install and configure PingDataGovernance Server.....	8
Install PingDirectory Server.....	9
Install PingDataGovernance Server.....	9
Configure the PingDataGovernance User Store.....	10
Configure the PingDataGovernance OAuth subject search.....	10
Configure PingDataGovernance logging.....	11
Install and configure the PingDataGovernance Policy Administration GUI.....	11
Import default policies.....	13
Configure PingDataGovernance Server for policy development.....	14
Create the first API policy.....	15
Configure the API security gateway.....	15
Add a policy for programming jokes.....	19
Add a policy for the user city.....	26
Example files.....	29
Create the first SCIM policies.....	29
Create the policy tree.....	30
Create SCIM access token policies.....	31
Create a policy for role-based access control.....	42
Example files.....	44
About the API security gateway.....	44
Request and response flow.....	44
Gateway configuration basics.....	46
API security gateway authentication.....	46
API security gateway policy requests.....	47
Policy request attributes.....	47
Gateway API Endpoint configuration properties that affect policy requests.....	51
Path parameters.....	52
About error templates.....	52
Example.....	53

About the Sideband API.....	54
API gateway integration.....	54
Sideband API configuration basics.....	56
Sideband API authentication.....	56
Authenticating to the Sideband API.....	57
Authenticating API server requests.....	58
Sideband API policy requests.....	59
Policy request attributes.....	59
Sideband API Endpoint configuration properties.....	63
Path parameters.....	63
Error templates.....	64
Error templates: Example.....	65
 About the SCIM service.....	 65
Request and response flow.....	66
SCIM configuration basics.....	68
About the create-initial-config tool.....	68
Example: Mapped SCIM resource type for devices.....	68
SCIM endpoints.....	70
SCIM authentication.....	71
SCIM policy requests.....	72
Policy request attributes.....	72
About SCIM searches.....	76
Lookthrough limit.....	77
Disable the SCIM REST API.....	78
 Policy administration.....	 78
Create policies in a development environment.....	78
Change the active policy branch.....	78
Use policies in a production environment.....	79
Default policies.....	79
Customized policies.....	80
Environment-specific Trust Framework attributes.....	80
Store keys and values in PingDataGovernance Server.....	81
External PDP mode.....	82
Embedded PDP mode.....	88
Advice.....	89
Add Filter.....	90
Allow Attributes.....	90
Combine SCIM Search Authorizations.....	90
Denied Reason.....	91
Exclude Attributes.....	91
Filter Response.....	91
Include Attributes.....	92
Prohibit Attributes.....	93
 Access token validators.....	 93
About access token validator processing.....	93
Access token validator types.....	95
PingFederate access token validator.....	95
JWT access token validator.....	96

Mock access token validator.....	97
Third-party access token validator.....	98
Server configuration.....	98
Administration accounts.....	98
About the dsconfig tool.....	98
PingDataGovernance Administration Console.....	99
About the configuration audit log.....	99
About the config-diff tool.....	100
Certificates.....	100
Inter-server certificate.....	101
Server certificate.....	104
Capture debugging data.....	108
Export policy data.....	108
Enable detailed logging.....	109
Policy Decision logger.....	109
Debug Trace logger.....	109
Debug logger.....	110
Trace a policy-decision response.....	110
Capture debugging data with the collect-support-data tool.....	112
PingDataGovernance Policy Administration GUI single sign-on.....	112
Reconfigure the PingDataGovernance Policy Administration GUI.....	112
PingFederate dependencies.....	113
PingFederate example configuration.....	113
OAuth server settings.....	114
Identity provider settings.....	116
Upgrade PingDataGovernance Server.....	116
Upgrade overview and considerations.....	116
Upgrading PingDataGovernance Server.....	117
Reverting an update.....	117
PingDataGovernance Server 7.3.0.3 Release Notes.....	118
PingDataGovernance Server Release Notes archive.....	118
PingDataGovernance Server 7.3.0.2 Release Notes.....	118
PingDataGovernance Server 7.3.0.1 Release Notes.....	120
PingDataGovernance Server 7.3.0.0 Release Notes.....	121
Index.....	129

PingDataGovernance™ Product Documentation

© Copyright 2004-2019 Ping Identity® Corporation. All rights reserved.

© Copyright 2014-2019 Symphonic Software® Limited. All rights reserved.

Trademarks

Ping Identity, the Ping Identity logo, PingFederate, PingAccess, and PingOne are registered trademarks of Ping Identity Corporation ("Ping Identity"). All other trademarks or registered trademarks are the property of their respective owners.

Disclaimer

The information provided in these documents is provided "as is" without warranty of any kind. Ping Identity disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall Ping Identity or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if Ping Identity or its suppliers have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages so the foregoing limitation may not apply.

Support

<https://support.pingidentity.com/>

Introduction to PingDataGovernance Server

PingDataGovernance Server provides a policy-based security layer for protecting consumer data.

Increasingly, enterprises grant their users more control over their data privacy. While previous use cases were typically simple, like a user opting out of an email newsletter, current use cases are growing more sophisticated. In health care, for example, patients can grant family members and other third parties partial or full access to their health records. Similarly, banking customers frequently control the account data that is shared with different third parties.

The sophistication of modern, user-managed data privacy places increasing demands on security professionals and API developers to ensure that user preferences and other policies are enforced in partner and application APIs. Mistakes often result in costly data breaches as well as a loss of trust.

As an API security gateway to user-related data APIs, PingDataGovernance provides organizations with an additional layer of protection to prevent data breaches. Organizations can add policy to complete the following tasks:

- Inspect the content of API requests and responses
- Verify user preferences and other attributes
- Allow, deny, or sanitize specific API data

Key components

- **PingDataGovernance Policy Administration GUI** – Powered by Symphonic®, the PingDataGovernance Policy Administration GUI gives policy administrators the ability to author and test security and business policies. The GUI is divided into the following sections:
 - In the **Trust Framework**, administrators define the entities and abstractions for the information that a policy uses.

- In **Policies**, administrators define the hierarchies of conditions and rules to evaluate data and make policy decisions.
- **API security gateway** – In PingDataGovernance Server, the API security gateway invokes the policy engine to evaluate API requests, and then enforces the policy decisions. Policy decisions can result in many outcomes, including allowing or denying an API request, and filtering or altering an API response.

What's new

PingDataGovernance 7.3 brings major changes compared to earlier versions.

Fundamentally, PingDataGovernance has always been and continues to be a solution for protecting access to sensitive or regulated consumer data. In earlier versions, PingDataGovernance focused almost entirely on the fine-grained protection of user-profile data. PingDataGovernance 7.3 expands its applicability to provide fine-grained protection to the user-related data that is shared through your partner and application APIs.

This expansion of applicability results in the following additions and changes:

- Earlier versions of PingDataGovernance focused on enforcing policy on SCIM-based APIs. This approach worked for IAM teams that built solutions in need of identity APIs. However, the application and platform APIs through which enterprises share user-related data already exist, and they are incompatible with SCIM. The capability to define new SCIM-based APIs still exists, but now the existing application and platform APIs can also be proxied.
- Earlier versions emphasized the promotion of meaningful OAuth scopes, with each version possessing a fine-grained configuration for the allowed operations on SCIM resource attributes. Although the best practices for meaningful or semantic OAuth scopes remain important to the design of new APIs, existing enterprise APIs and OAuth scopes might already be defined with varying degrees of granularity.

Additionally, the configuration of meaningful OAuth scopes has been removed. Policy now handles the mapping of OAuth scope names to the set of allowed operations on SCIM resource attributes.

- Earlier versions of PingDataGovernance required administrators to develop policy logic in the scripting language Java Expression Language (JEXL). To help business stakeholders and administrators create and manage policies with confidence, administrators require a more user-friendly development environment with a testing interface. As a result, the JEXL-based policy service has been replaced with a new policy service that the PingDataGovernance Policy Administration GUI configures.

Explore PingDataGovernance Server

A complete PingDataGovernance solution includes the following components:

- **PingDataGovernance Server** – Enforces fine-grained data-access policies. It consists of the following major components:
 - API security gateway
 - SCIM service
 - Policy Decision service
- **PingDataGovernance Policy Administration GUI** – Powered by Symphonic, the PingDataGovernance Policy Administration GUI provides an interface that lets business stakeholders and administrators collaborate to develop and test policies. When policies are ready for production, they are exported to PingDataGovernance's Policy Decision Service.
- **User Store** – PingDataGovernance requires a *User Store* from which to obtain attributes about the user who is invoking APIs, or the user about whom a service is invoking APIs, to evaluate the attributes as part of policy. Although PingDataGovernance assumes that PingDirectory Server is the default User Store, other LDAPv3-compliant directories are also supported.

This section explores these components in greater detail.

System entropy

Entropy is used to calculate random data that the system uses in cryptographic operations. Some environments with low entropy might experience intermittent performance issues with SSL-based communication, such as certificate generation. This scenario is more typical on virtual machines but can also occur in physical instances. For best results, monitor the value of `kernel.random.entropy_avail` in the configuration file `/etc/sysctl.conf`.

Note: To increase system entropy on a Windows system, move the mouse pointer in circles or type characters randomly into an empty text document.

On a UNIX or Linux system, ensure that `rng-tools` is installed and run the following command:

```
sudo rngd -r /dev/urandom -o /dev/random
```

To check the level of system entropy on a UNIX or Linux system, run the following command:

```
cat /proc/sys/kernel/random/entropy_avail
```

Values smaller than 3200 are considered too low to generate a certificate and might cause the system to hang indefinitely.

About the tools.properties file

PingDataGovernance Server supports the use of a `tools.properties` file that simplifies command-line invocations by reading in a set of arguments for each tool from a text file. Each property takes the form of a name-value pair that defines predetermined values for a tool's arguments.

Properties files are convenient when quickly testing PingDataGovernance Server in multiple environments. PingDataGovernance Server supports the following types of properties files:

- Default properties files that can be applied to all command-line utilities
- Tool-specific properties files that are specified by the `--propertiesFilePath` option

To override PingDataGovernance Server's command-line utilities, use the properties file `config/tools.properties`. With this approach, you can avoid typing frequently used arguments like `-port` and `-bindDN`.

System requirements

Ping Identity® has qualified the configurations in this section and has certified that they are compatible with the product. Differences in operating system versions, service packs, and other platform variations are supported until the platform or other required software is suspected of causing an issue.

Platforms

- Windows Server 2019
- Windows Server 2016
- Red Hat Enterprise Linux ES 7.6
- Red Hat Enterprise Linux ES 7.5
- Red Hat Enterprise Linux ES 6.10
- Red Hat Enterprise Linux ES 6.9
- CentOS 7.6

- CentOS 7.5
- CentOS 6.10
- CentOS 6.9
- SUSE Linux Enterprise 15
- SUSE Linux Enterprise 12 SP3
- Ubuntu 18.04 LTS
- Ubuntu 16.04 LTS
- Amazon Linux 2
- Amazon Linux

Note: This product has been tested with the default configurations of all operating system components. If your organization has customized implementations or has installed third-party plugins, the deployment of this product might be affected.

Docker

Version: Docker 18.09.0

Host operating system: Ubuntu 18.04 LTS

Kernel: 4.4.0-1052-aws 7.3

Note: Ping Identity accepts no responsibility for the performance of any specific virtualization software and in no way guarantees the performance or interoperability of any virtualization software with its products.

Java Runtime Environment

- Amazon Corretto 8
- OpenJDK 11
- OpenJDK 8
- Oracle Java SE Development Kit 11 LTS
- Oracle Java SE Development Kit 8

Browsers

Administration Console

- Chrome
- Firefox
- Internet Explorer 11 and later

End users

- Chrome
- Edge
- Firefox
- Internet Explorer 11 and later
- Safari

Install and configure PingDataGovernance Server

About this task

This section describes the initial steps of setting up PingDataGovernance Server. For information about updating to a new version of PingDataGovernance Server, see [Upgrade PingDataGovernance Server](#) on page 116.

In this section, you will complete the following tasks:

Steps

1. Install a PingDirectory Server instance and a PingDataGovernance Server instance.
2. Configure PingDataGovernance Server to use PingDirectory Server as the User Store.
3. Configure PingDataGovernance Server to search PingDirectory Server for OAuth token subjects.

Install PingDirectory Server

About this task

PingDataGovernance requires a User Store to evaluate identity attributes as part of policy. The following command sets up PingDirectory Server with 1,000 users:

```
PingDirectory/setup \
  --cli --no-prompt --acceptLicense \
  --licenseKeyFile <path-to-pd-7x-license> \
  --rootUserDN "cn=directory manager" \
  --rootUserPassword <your-ds-password> \
  --ldapPort 1389 \
  --ldapsPort 1636 \
  --httpsPort 1443 \
  --generateSelfSignedCertificate \
  --baseDN "dc=example,dc=com" \
  --maxHeapSize 384m \
  --instanceName dsl \
  --location Austin \
  --sampleData 1000
```

In this example, the server listens for LDAPS requests on port 1636.

Install PingDataGovernance Server

About this task

The following command sets up PingDataGovernance Server:

```
PingDataGovernance/setup \
  --cli --no-prompt --acceptLicense \
  --licenseKeyFile <path-to-dg-7x-license> \
  --rootUserDN "cn=directory manager" \
  --rootUserPassword <your-dg-password> \
  --ldapPort 8389 --ldapsPort 8636 \
  --httpsPort 8443 \
  --generateSelfSignedCertificate \
  --maxHeapSize 1g \
  --instanceName dgl \
  --location Austin
```

In this example, PingDataGovernance Server listens for the following requests:

- LDAPS requests on port 8636
- HTTPS requests on port 8443

Configure the PingDataGovernance User Store

About this task

Configure PingDataGovernance Server to use PingDirectory Server as its User Store.

The first command makes a set of changes to PingDirectory Server that are needed by PingDataGovernance Server, including the creation of a service account:

```
PingDataGovernance/bin/prepare-external-store \
--hostname <your-ds-host> --port 1636 --useSSL --trustAll \
--governanceTrustStorePath PingDataGovernance/config/truststore \
--governanceTrustStorePasswordFile \
PingDataGovernance/config/truststore.pin \
--bindDN "cn=directory manager" \
--bindPassword <your-ds-password> \
--governanceBindDN "cn=Governance User,cn=Root DNs,cn=config" \
--governanceBindPassword <your-dg-service-account-password> \
--userStoreBaseDN "ou=people,dc=example,dc=com" \
--no-prompt
```

The second command configures PingDataGovernance Server with a store adapter that allows it to communicate with PingDirectory Server to retrieve identity attributes. This command also sets up a SCIM resource type that defines a Users type with a SCIM schema that is automatically mapped to an LDAP type (inetOrgPerson) on PingDirectory Server.

```
PingDataGovernance/bin/create-initial-config \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--governanceBindPassword <your-dg-service-account-password> \
--externalServerConnectionSecurity useSSL \
--governanceTrustStorePath PingDataGovernance/config/truststore \
--governanceTrustStorePasswordFile \
PingDataGovernance/config/truststore.pin \
--userStoreBaseDN "ou=people,dc=example,dc=com" \
--userStore "<your-ds-host>:1636:Austin" \
--userObjectClass "inetOrgPerson" \
--initialSchema pass-through
```

Configure the PingDataGovernance OAuth subject search

About this task

Configure PingDataGovernance Server to search the User Store for OAuth token subjects.

The first command configures PingDataGovernance Server to mock OAuth access token validation.

The Mock Access Token Validator accepts tokens without authenticating them, and is used only for demonstration and testing purposes. To use an authorization server like PingFederate, see [Access token validators](#) on page 93.

```
PingDataGovernance/bin/dsconfig create-access-token-validator \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--validator-name "Mock Access Token Validator" \
--type mock --set enabled:true --set subject-claim-name:sub
```

The second command configures PingDataGovernance Server to search the User Store and retrieve the identity attributes of the OAuth token subject, so that the attributes can be evaluated in policy. A token

resource lookup method defines the expression that is used to search SCIM resources by the access token subject or additional claims. In this scenario, the value of the access token subject claim is used to search the `uid` attribute value of the SCIM User resource.

```
PingDataGovernance/bin/dsconfig create-token-resource-lookup-method \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--validator-name "Mock Access Token Validator" \
--method-name "User by uid" \
--set scim-resource-type:Users \
--set 'match-filter:uid eq "%_subject_claim_name%"' \
--set evaluation-order-index:100
```

Configure PingDataGovernance logging

About this task

As you familiarize yourself with developing, testing, and enforcing policies, consider increasing the default logging value to include details that will aid in debugging.

The following command enables more detailed logging to understand how policy decisions are being made, including the comparison values and results of the various expressions that comprise a policy decision tree:

```
PingDataGovernance/bin/dsconfig set-policy-decision-service-prop \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--add decision-response-view:decision-tree \
--add decision-response-view:request \
--add decision-response-view:evaluated-entities
```

Note: `decision-response-view:request` causes the Policy Decision Logger to record potentially sensitive data in API requests and responses.

The following command enables Trace (detailed) logging, including complete HTTP requests and responses:

```
PingDataGovernance/bin/dsconfig set-log-publisher-prop \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--publisher-name "Debug Trace Logger" \
--set enabled:true
```

Note: Logging complete HTTP requests and responses might contain sensitive data.

For information about enabling detailed debug logging for troubleshooting purposes, see [Enable detailed logging](#) on page 109.

Install and configure the PingDataGovernance Policy Administration GUI

About this task

To install an instance of the PingDataGovernance Policy Administration GUI, perform the following steps:

Steps

1. Extract the contents of the compressed PingDataGovernance-PAP distribution file.
2. Change the directory to PingDataGovernance-PAP.
3. To configure the application, run the `./bin/setup` script.
4. Answer the on-screen questions.

We recommend specific answers for the following questions:

- **How should the server be configured for authentication?**

- Basic user name and password
- Connect to PingFederate server for OIDC SSO

Enter option 1, "Basic Authentication (for testing only)," which represents the basic user name and password option.

- **Should the server be configured to use HTTPS?**

Because this example uses self-signed certificates to enable HTTPS, enter option 2, "Generate a self-signed SSL certificate."

- **What URL should be used to access the Symphonic API?**

Unless you are testing on `localhost`, ensure that the provided API URL uses the public DNS name of the PingDataGovernance Policy Administration GUI server. Include the full URL, including the path and trailing slash, as the following example shows:

```
https://<DNS_Name>:<PortFromEarlierQuestion>/api/
```

- **This server provides a Policy Decision Point that is protected by a shared secret. What should be used as the shared secret?**

Enter `my-shared-secret`.

5. To start the PAP, run `bin/start-server`.

The PAP runs in the background, so you can close the terminal window in which it was started without interrupting it.

Results

The following transcript represents an example demo configuration:

```
How should the server be configured for authentication?
 1) Basic Authentication (for testing only)
 2) OpenID Connect

Enter option [1]: 1

Should the server be configured to use HTTPS?
 1) Do not use HTTPS, only allow HTTP (insecure, use for testing)
 2) Generate a self-signed SSL certificate
 3) Use an existing SSL certificate

Enter option [2]:
Enter the path to the PingDataGovernance license key file or copy the file to /home/
centos/PingDataGovernance-PAP/PingDataGovernance.lic and press ENTER:

Symphonic Policy Admin Point installer
=====

Please answer the following questions:

  What port should the application run on? [8080]:

  What URL should be used to access the Symphonic API? (Port must match above.) ["http://
localhost:8080/api/"]: https://pap.example.com:8080/api/

  This server provides a policy decision point that is protected by a shared secret. What
  should be used as the shared secret? [null]: my-shared-secret
```

```

Application connector type? [https]:
KeyStore type? [PKCS12]:
KeyStore path? [keystore]:
KeyStore password? [{}]:
SSL Certificate alias? [my-ssl-cert]:
SSL Certificate password? [{}]:
Validate SSL certificates before starting? [false]:
Validate SSL certificate peers? [false]:

Configuration Summary:

server.applicationConnectors[0].port:          8080
ui.REST_URL:                                   https://pap.example.com:8080/api/
core['Authentication.SharedSecret']:           my-shared-secret
server.applicationConnectors[0].type:          https
server.applicationConnectors[0].keyStoreType:   PKCS12
server.applicationConnectors[0].keyStorePath:   keystore
server.applicationConnectors[0].keyStorePassword: [C@3c87521
server.applicationConnectors[0].certAlias:      my-ssl-cert
server.applicationConnectors[0].keyManagerPassword: [C@2aece37d
server.applicationConnectors[0].validateCerts:  false
server.applicationConnectors[0].validatePeers:  false

>>>> Configuration written to /home/centos/PingDataGovernance-PAP/config/configuration.yml

Generating SSL certificate...

>>>> SSL certificate saved to keystore

To start the server, run the ./bin/start-server script.

```

Next steps

In this example, the PingDataGovernance Policy Administration GUI is now running and listening on port 8080. To log on to the interface, visit `https://<host>:8080`. The default credentials are `admin` and `password123`.

Note: Use the default user name and password logon credentials for demo and testing purposes only, like this initial walk-through. To configure the PingDataGovernance Policy Administration GUI for PingFederate OIDC SSO, see [PingDataGovernance Policy Administration GUI single sign-on](#) on page 112.

Import default policies

About this task

After logging on to PingDataGovernance Server, the following options are displayed:

- **Create a Branch**
- **Import a Branch from Snapshot**

To use the default policies that are distributed with PingDataGovernance Server, perform the following steps:

Steps

1. Under **Import a Branch from a Snapshot**, select **Click here to select a snapshot file**.
The snapshot file is located in the PingDataGovernance installation directory at `resource/policies/defaultPolicies.SNAPSHOT`.
2. Name the branch file `Default Policies`.

3. Click **Import**.

Configure PingDataGovernance Server for policy development

About this task

PingDataGovernance Server can be configured to evaluate policy in Embedded mode or External mode. During policy development, configure PingDataGovernance Server in External mode, where it calls in to the PingDataGovernance Policy Administration GUI for policy evaluation.

Steps

1. From the **Data Sources** section of the PingDataGovernance Administration Console (<https://<your-dg-host>:8443/console>), click **External Servers > New External Server > Policy External Server**.
2. On the **New Policy External Server** page, specify the following information:
 - For the **Name**, specify PingDataGovernance Policy Administration Point.
 - For the **Base URL** property, specify https://<your_PingDataGovernance_host>:8080.
 - For the **Host Name Verification Method**, select **allow-all** for test environments. If you are specifying a host name verification method for a non-test environment, you might need to configure additional mechanisms.
 - For the **X User Id**, specify **admin**.
 - For the **X Branch**, specify the name of the branch that you created while importing the default policy snapshot (Default Policies).
 - For the **Shared Secret**, click **Set Value**, type **my-shared-secret** in the value and confirmation boxes, and click **OK**.
3. To specify a value for the **X Decision Node**, perform the following steps:
 - a) Access the PingDataGovernance Policy Administration interface.
 - b) Click **Policies > Global Decision Point**.
 - c) In the upper-right corner, click
 - d) Click **#**.
 - e) To copy the node ID for the global decision point to the system clipboard, click **Copy ID to clipboard**.
 - f) Paste the node ID (**e51688ff-1dc9-4b6c-bb36-8af64d02e9d1**) into the **X Decision Node** text box.
4. Click **Save**.

Next steps

As an alternative to using the GUI, the following snippet provides an equivalent sample `dsconfig` command to create the external server:

```
PingDataGovernance/bin/dsconfig create-external-server \
--no-prompt --port 8638 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassord <your-dg-password> \
--server-name "PingDataGovernance Policy Administration Point" \
--type policy \
--set base-url:http://<your_PingDataGovernance_host>:8080 \
--set hostname-verification-method:allow-all \
--set user-id:admin \
--set "shared-secret:my-shared-secret" \
--set decision-node:<global-decision-point-id> \
--set "branch:Default Policies"
```

Configure the policy service in External mode

About this task

After the policy external server has been created, perform the following steps:

Steps

1. From the PingDataGovernance Administration Console, click **Authorization and Policies > Policy Decision Service**.
2. For the **PDP Mode**, select `external`.
3. For the **Policy Server**, select the policy external server that you created in [Configure PingDataGovernance Server for policy development](#) on page 14.
4. Keep all the other default values.
5. Click **Save To Data Governance Server Cluster**.



Warning: If you are using automation or DevOps to manage a cluster of PingDataGovernance Servers, do not configure the nodes to share configuration details automatically among the servers.

Next steps

As an alternative to using the GUI, the following snippet provides an equivalent sample `dsconfig` command to configure the policy service in external mode:

```
PingDataGovernance/bin/dsconfig set-policy-decision-service-prop \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--set pdp-mode:external \
--set "policy-server:PingDataGovernance PDP"
```

Create the first API policy

About this task

In this section, you will build and test your first policy for the PingDataGovernance API security gateway.

Suppose that your organization creates an application to provide users with jokes to tell at parties. A joke API generates several jokes in different categories, and users are granted the ability to filter certain types of jokes that they might find offensive or unappealing.

This example uses the public joke API developed by https://github.com/15Dkatz/official_joke_api.

Configure the API security gateway

The API security gateway functions as the intermediary between the API client and the API server. These components are configured in the PingDataGovernance Administration Console.

In this section, you will configure `https://<your-dg-host>:<your-https-dg-port>/jokes/random` to proxy to https://official-joke-api.appspot.com/random_joke.

Create the API external server

About this task

To create the API external server, perform the following steps:

Steps

1. From the PingDataGovernance Administration Console, click **Data Sources > External Servers**.
2. Click **New External Server > API External Server**.
3. For the **Name**, specify Joke API Server.
4. For the **Base URL**, specify `https://official-joke-api.appspot.com`.
5. Click **Save**.

Results

New API External Server

API External Servers are used by API Endpoints to specify connections to external API servers using HTTP or HTTPS.

[View dsconfig](#)
[Save](#)
[Cancel](#)

Name *	<input type="text" value="Joke API Server"/>	?
Description	<div style="border: 1px solid #ccc; height: 30px;"></div>	?
Base URL *	<input type="text" value="https://official-joke-api.appspot.com"/>	?
Hostname Verification Method	<div style="border: 1px solid #ccc; padding: 2px;">strict</div>	?
Key Manager Provider	<div style="border: 1px solid #ccc; padding: 2px;">The Java Runtime Environment's default key manager</div>	?
Trust Manager Provider	<div style="border: 1px solid #ccc; padding: 2px;">The Java Runtime Environment's default trust manager</div>	?
Connect Timeout	<input type="text" value="30 s"/>	?
Response Timeout	<input type="text" value="30 s"/>	?
Allowed Header	<div style="border: 1px solid #ccc; padding: 2px; margin-bottom: 5px;">Enter a value to add</div> <div style="border: 1px solid #ccc; padding: 2px;">If this property is not specified, then all end-to-end request headers will be forwarded</div>	?
User Name	<input type="text"/>	?
Password	Set Value	?

Next steps

As an alternative to using the GUI, the following snippet provides an equivalent sample `dsconfig` command to create the API external server:

```
PingDataGovernance/bin/dsconfig create-external-server \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--server-name "Joke API Server" \
--type api \
--set base-url:https://official-joke-api.appspot.com
```


Create the Gateway API Endpoint

About this task

To create the external Gateway API Endpoint, perform the following steps:

Steps

1. From the PingDataGovernance Administration Console, click **Web Services and Applications > Gateway API Endpoints**.
2. Click **New Gateway API Endpoint**.
3. For the **Name**, specify `Random Joke API`.
4. For the **Inbound Base Path**, specify `/jokes/random`.
5. For the **Outbound Base Path**, specify `/random_joke`.
6. For the **API Server**, specify `Joke API Server`.
7. For **HTTP Auth Evaluation Behavior**, specify `evaluate-and-discard`.
8. For **Access Token Validator**, specify `Mock Access Token Validator`.
9. Click **Save To Data Governance Server Cluster**.



Warning: If you are using automation or DevOps to manage a cluster of PingDataGovernance Servers, do not configure the nodes to share configuration details automatically among the servers.

Results


New Gateway API Endpoint


A Gateway API Endpoint represents an endpoint at an API service that is protected by the Data Governance Server Gateway, which acts as a facade and policy enforcement point (PEP) for the API service.




[View dsconfig](#)


Save To Data Governance Server Cluster
Cancel


General Configuration


Name * 





Description 

Error Template   


Correlation ID Header 


Inbound Base Path * 




Outbound Base Path * 




API Server *    

Authorization and Policies

Service 


Resource Path 





Policy Request Attribute  
 

HTTP Auth Evaluation Behavior   


Access Token Validator

Available





Selected



Mock Access Token Validator

Next steps

As an alternative to using the GUI, the following snippet provides an equivalent sample `dsconfig` command to create the Gateway API Endpoint:

```
PingDataGovernance/bin/dsconfig create-gateway-api-endpoint \
  --no-prompt --port 8636 --useSSL --trustAll \
  --bindDN "cn=directory manger" \
  --bindPassword <your-dg-password> \
```

```
--endpoint-name "Random Joke API" \
--set inbound-base-path:/jokes/random \
--set outbound-base-path:/random_joke \
--set "api-server:Joke API Server" \
--set http-auth-evaluation-behavior:evaluate-and-discard \
--set "access-token-validator:Mock Access Token Validator"
```

Test the gateway with cURL

About this task

Before testing the gateway, make certain that you have configured everything successfully with an HTTP client like Postman or cURL.

Issue the following request:

```
curl --insecure -X GET
https://<your-dg-host>:<your-dg-port>/jokes/random \
-H 'Authorization: Bearer {"active": true}'
```

Note: To provide easy testing, the Mock Access Token Validator allows the use of unencoded and unsigned bearer tokens.

The following response is typical:

```
{
  "id":25,
  "type":"programming",
  "setup":"How many programmers does it take to change a light bulb?",
  "punchline":"None that's a hardware problem"
}
```

Add a policy for programming jokes

Policies are developed and tested in the PingDataGovernance Policy Administration GUI, which is divided into the following sections:

- **Trust Framework** – Defines the attributes for information that the policy rules use.
- **Policies** – Defines the rules that allow or block an API response.

Create attributes for a Joke API response

About this task


To implement user-managed control to filter certain types of jokes that users find offensive or unappealing, the policy requires checking the type attribute of the JSON response body of the Joke API.

The first attribute that you create represents the entire JSON response body of the Joke API:

Steps

1. From the PingDataGovernance Policy Administration GUI, click **Trust Framework > Attributes**.
2. To add a new attribute, click **+**.
3. For the **Name**, specify `Joke`.
4. Verify that **Parent** is not selected.
5. For the **Resolver Type**, select **Attribute** and specify a value of `HttpRequest.ResponseBody`.
6. For the **Value Settings Type**, select **JSON**.
7. Click **Save Changes**.

Results


Joke

Parent
no parent selected

Description

Resolver Settings

Resolver Type
Attribute
HttpRequest.ResponseBody
+ Add Condition

+ Add Resolver

Cache Settings

Cache Strategy
No Caching

Value Settings

Processor
None

Default value
☐

Type
JSON
Secret
☐

Delete Attribute
Save changes

Next steps

The second attribute that you create represents the `type` attribute of the JSON response body of the Joke API:

1. To add a new attribute, click **+**.
2. For the **Name**, specify `type`.
3. For the **Parent**, select `Joke`.
4. For the **Resolver Type**, select `Attribute` and specify a value of `Joke`.
5. For the **Value Settings Processor**, select `JSON Path` and specify a value of `$.type`.
6. For the **Value Settings Type**, select `String`.
7. Click **Save Changes**.

The screenshot shows the configuration interface for a service named "type". The interface includes the following sections:

- Parent:** A dropdown menu with "Joke" selected.
- Description:** A text input field.
- Resolver Settings:**
 - Resolver Type:** A dropdown menu with "Attribute" selected.
 - Condition:** A dropdown menu with "Joke" selected.
 - + Add Condition:** A button to add more conditions.
 - + Add Resolver:** A button to add more resolvers.
- Cache Settings:**
 - Cache Strategy:** A dropdown menu with "No Caching" selected.
- Value Settings:**
 - Processor:** A dropdown menu with "JSON Path" selected.
 - Path:** A text input field with "\$.type" entered.
 - Default value:** A checkbox that is unchecked.
 - Type:** A dropdown menu with "String" selected.
 - Secret:** A checkbox that is unchecked.
- Save changes:** A button at the bottom right with a green download icon.

Create a service for the Random Jokes API

About this task

The name of the Gateway API Endpoint configured in PingDataGovernance Server is passed as the service to the PingDataGovernance PDP. Create a policy that applies only to the requests and responses of the Random Jokes API, and add this service to the Trust Framework.

Steps

1. From the PingDataGovernance Policy Administration GUI, click **Trust Framework > Services**.
2. To add a new service, click **+**.
3. For the **Name**, type Random Joke API.
4. Verify that **Parent** is not selected.
5. Click **Save Changes**.

Results

Random Joke API

Parent: no parent selected

Description

Service Settings

Service Type: None

[Save changes](#)

Create a policy for the Random Jokes API

About this task

To create a policy that targets the outbound response to an HTTP `GET` to the Random Joke API, perform the following steps:

Steps

1. From the PingDataGovernance Policy Administration GUI, navigate to the **Policies** page.
2. Select **Global Decision Point** and click **+**.
3. Click **Add Policy**.
4. For the **Name**, specify `Random jokes API policy`.
5. Click **Show "Applies to"**.
6. In the upper-right corner of the left panel, click **Toolbox**.
7. From the **Actions** list, drag **outbound-GET** to the blue **Targets** box.
8. From the **Services** list, drag **Random Joke API** to the blue **Targets** box.
9. Click **Save Changes**.

Results

Random jokes API policy Disabled ☐

Description

Applies to 2 definitions

outbound-GET Random Joke API

Drag and Drop Targets or Targetable Definitions from Toolbox

Unless one decision is deny, the decision will be permit

[+ Create new Rule](#) or [Drag and Drop Rules from Toolbox](#)

[Hide "Applies to"](#) [Show "Applies when"](#) [Show Advice](#) [Show Properties](#)

[Save changes](#)

Add logic to reject programming jokes

About this task

To add a rule that blocks responses with `programming` as the `joke` type attribute value, perform the following steps:

Steps

1. Click **Create new Rule**.
2. For the **Name**, specify `Block programming jokes`.
3. For the **Effect**, select `Deny`.
4. To specify a **Condition**, perform the following steps:
 - a) From the first **Condition** field, select `Joke.type`.
 - b) From the second field, select `Equals`.
 - c) In the third field, type `programming`.
5. Click **Save Changes**.

Results

Unless one decision is deny, the decision will be permit

The screenshot shows the configuration for a rule named "Block programming jokes". The rule has a red "X" icon and three orange dots in the top right corner. The "Description" field is empty. The "Effect" is set to "Deny". The "Condition" is defined as "A Joke.type Equals C programming". Below the condition, there are buttons for "+ Comparison", "+ Named Condition", and "+ Group", and a "Drag and Drop any Trust Framework item from Toolbox" button. At the bottom, there are links for "Show 'Applies to'", "Show Advice", and "Show Properties". Below the rule configuration, there is a button for "+ Create new Rule" and a "Drag and Drop Rules from Toolbox" button.

Add advice to set the HTTP response code

About this task

Add a command called *advice* that instructs PingDataGovernance Server to set the HTTP response code when rejecting the outbound response. Because this problem is not associated with the HTTP client or its request, set the response code to 502 to indicate a temporary gateway issue.

Steps

1. Expand **Block programming jokes**.
2. Click **Show Advice**.
3. Click **Create new Advice**.
4. For the **Name**, type `Send "Bad gateway" response`.
5. For the **Code**, type `denied-reason`.
6. From the **Applies To** drop-down list, select `Deny`.

7. Click **+Payload**.
8. For a **Payload** value, type `{"status": 502}`.
9. Click **Save Changes**.

Results

Unless one decision is deny, the decision will be permit

Block programming jokes

Description

Effect

Deny

Condition

A

Joke.type

Equals

C

programming

+ Comparison

+ Named Condition

+ Group

Drag and Drop any Trust Framework Item from Toolbox

▼Advice (1 in total)

Name

Send "Bad gateway" response

Obligatory

☐

Description

Code

denied-reason

Applies To

Deny

Payload

{"status": 502}

+ Attributes

+ Services

+ Create new Advice

or

Drag and Drop Advice from Toolbox

Show "Applies to"

Hide Advice

Show Properties

+ Create new Rule

or

Drag and Drop Rules from Toolbox

Test the policy in the GUI

About this task

To test the full policy tree in the PingDataGovernance Policy Administration GUI, perform the following steps:

Steps

1. Navigate to the **Policies** page.
2. Click **Global Decision Point**.
3. Click the **Test** tab.
4. From the **Service** drop-down list, select **Random Joke API**.
5. From the **Action** drop-down list, select **outbound-GET**.
6. Add the following **HttpRequest** attribute:

```
{"AccessToken":{"active":true},
"ResponseBody":{"id":25,
"type":"programming",
```



```
"setup":"How many programmers does it take to change a light bulb?",
"punchline":"None that's a hardware problem"}}}
```

The following image provides an example:

Details History **Test** Analysis

Global Decision Point

Testing Scenario

Request

Domain: Select to add Domain to the testing scenario

Service: Random Joke API

Identity Provider: Select to add Identity Provider to the testing scenario

Action: outbound-GET

Attributes: **HttpRequest**

```
{
  "AccessToken": {"active": true},
  "ResponseBody": {"id": 25,
    "type": "programming",
    "setup": "How many programmers does it take to change a light bulb?",
    "punchline": "None that's a hardware problem"}}
}
```

Select an attribute to add it to the testing scenario

Overrides

Attributes: Select an attribute to add it to the testing scenario

Services: Select a service to add it to the testing scenario

Import Execute

7. Click **Execute**.

The HTTP GET response is rejected because of the "Block programming jokes" rule.

Results



Test the API gateway with cURL

About this task

Test the proxied API again with cURL, as follows:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/jokes/
random \
-H 'Authorization: Bearer {"active": true}'
```

Results

Non-programming jokes are allowed, but programming jokes return an HTTP 502 (bad gateway) response.

Add a policy for the user city

To simulate a user preference that is stored in an online profile, extend the policy to block programming jokes from users in cities that are rich in software developers, like San Francisco, Boston, Austin, or Seattle.

Find a user

About this task

To find a user, perform the following steps:

Steps

1. Verify that your PingDirectory example data contains a user whose profile location is set to one of the developer-dense cities.
2. Issue the following search on your PingDirectory host, taking special note of the location (l, a lowercase L) and user name (uid) of the resulting user:

```
PingDirectory/bin/ldapsearch \
--port 1636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-ds-password> \
--sizeLimit 1 "(&(|(l=Boston)(l=Austin)(l=San Fran*)(l=Seattle)))"
```

Results

The following sample represents the typical output, although your output might differ:

```
dn: uid=user.20,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
mail: user.20@example.com
initials: KFS
homePhone: +1 707 878 3104
pager: +1 188 707 6756
givenName: Katie
employeeNumber: 20
telephoneNumber: +1 024 280 5210
mobile: +1 625 070 5636
sn: Steeves
cn: Katie Steeves
description: This is the description for Katie Steeves.
street: 23279 Seventh Street
st: IN
```

```
postalAddress: Katie Steeves$23279 Seventh Street$Boston, IN 85072
uid: user.20
l: Boston
postalCode: 85072
```

Create an attribute for the user location

About this task

The information of the user whom the OAuth bearer token identified to PingDataGovernance is passed as the `TokenOwner` to the PingDataGovernance PDP. To use the location attribute of the user within the policy, add it to the trust framework:

Steps

1. Click **Trust Framework > Attributes**.
2. To add a new attribute, click **+**.
3. In the **Name** text box, type `city`.
4. From the **Parent** drop-down list, select `TokenOwner`.
5. In the **Resolver Settings** section, perform the following steps:
 - a) Click **Add Resolver**.
 - b) From the **Resolver Type** drop-down list, select `Attribute` and specify a value of `TokenOwner`.
6. In the **Value Settings** section, perform the following steps:
 - a) From the **Processor** drop-down list, select `JSON Path` and type a value of `$.l` (lowercase L).
 - b) Select the **Default Value** check box and type a value of `[]`.
This step prevents a policy rule that uses the `city` attribute from failing if the token owner possesses a null value for the `l` attribute.
 - c) Because the location is a multi-valued attribute, select `Collection` from the **Type** drop-down list.
7. Click **Save changes**.

Results

The screenshot shows the 'city' attribute configuration page in the PingDataGovernance UI. The left sidebar shows a tree view with 'Gateway', 'HttpRequest', 'Joke', 'TokenOwner', and 'city' (selected). The main panel has tabs for 'Details', 'History', and 'Test'. The 'city' attribute is configured with the following settings:

- Parent:** TokenOwner
- Description:** (empty text box)
- Resolver Settings:**
 - Resolver Type:** Attribute
 - TokenOwner:** (value)
 - + Add Condition** (button)
 - + Add Resolver** (button)
- Cache Settings:**
 - Cache Strategy:** No Caching
- Value Settings:**
 - Processor:** JSON Path
 - Value:** \$.l
 - Default value:** ☒ []
 - Type:** Collection
 - Secret:** ☐

At the bottom, there are buttons for 'Delete Attribute' and 'Save changes'.

Add logic to check the user location

About this task

To check the user location by extending the condition logic of your policy rule, perform the following steps:

Steps

1. Click **Policies > Global Decision Point > Random jokes API policy**.
2. Expand the "Block programming jokes" rule.
3. In the **Condition** group box, perform the following steps:
 - a) Click **+ Comparison**.
 - b) From the first drop-down list, select `TokenOwner.city`.
 - c) From the comparator drop-down list, select `Contains`.
 - d) In the final text box, type `Boston`, which is the city that you found when searching for a user.
4. Click **Save Changes**.

Results

Unless one decision is deny, the decision will be permit

Block programming jokes

Description

Effect

Deny

Condition

ALL ANY NONE Clear all

A

Joke.type

Equals

C

programming

A

TokenOwner.city

Contains

C

Boston

+ Comparison

+ Named Condition

+ Group

Drag and Drop any Trust Framework Item from Toolbox

Advice (1 in total)

Name

Send "Bad gateway" response

Obligatory

+ Create new Advice

or

Drag and Drop Advice from Toolbox

Show "Applies to"

Hide Advice

Show Properties

+ Create new Rule

or

Drag and Drop Rules from Toolbox

Test the gateway with cURL

About this task

1. Test the proxied API again with cURL, but this time include the user name of the user whom you located earlier, as follows:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/jokes/random \
-H 'Authorization: Bearer {"active": true, "sub": "user.20"}'
```

Try the API repeatedly until you receive an HTTP 502 response.

2. Test the proxied API again with a different user from a different city, as follows:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/jokes/random \
```

```
-H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
```

Keep trying the API until you receive a programming joke.

Example files

The compressed PingDataGovernance Server file at `PingDataGovernance/resource/policies` includes a policy snapshot and deployment package that contains an example Trust Framework as well as example policies.

Create the first SCIM policies

In the previous section, you used PingDataGovernance Server to filter data that an external REST API returned. In this section, you will develop a set of access-control policies for the PingDataGovernance Server's built-in SCIM REST API.

While PingDataGovernance Server's API security gateway protects existing REST APIs, PingDataGovernance Server's built-in SCIM service provides a REST API for accessing and protecting identity data that might be contained in datastores like LDAP and relational databases.

PingDataGovernance Server uses SCIM in the following ways:

- Internally, user identities are represented as SCIM identities by way of one or more SCIM resource types and schemas. This approach includes access token subjects, which are always mapped to a SCIM identity.
- A SCIM REST API service provides access to user identities through HTTP.

You will now design a set of policies to control access to the SCIM REST API by using OAuth 2 access token rules. This section assumes that you have set up and configured PingDataGovernance Server as described previously.

Before proceeding, make a test request to generate a SCIM REST API response to a request when only the default policies are in place. As in the earlier section, a mock access token is used.

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me \
-H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "nonexistent.scope", "client_id": "nonexistent.client"}'
```

Although the precise attribute values might vary, the response returns the SCIM resource that corresponds to `user.1`.

```
{ "mail": [ "user.1@example.com" ], "initials": [ "RJV" ], "homePhone": [ "+1 091 438 1890" ], "pager": [ "+1 472 824 8704" ], "givenName": [ "Romina" ], "employeeNumber": "1", "telephoneNumber": [ "+1 319 624 9982" ], "mobile": [ "+1 650 622 7719" ], "sn": [ "Valerio" ], "cn": [ "Romina Valerio" ], "description": [ "This is the description for Romina Valerio." ], "street": [ "84095 Maple Street" ], "st": [ "NE" ], "postalAddress": [ "Romina Valerio$84095 Maple Street$Alexandria, NE 39160" ], "uid": [ "user.1" ], "l": [ "Alexandria" ], "postalCode": [ "39160" ], "entryUUID": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "objectClass": [ "top", "person", "organizationalPerson", "inetOrgPerson" ], "entryDN": "uid=user.1, ou=people, o=yeah", "meta": { "resourceType": "Users", "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e", "id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "schemas": [ "urn:pingidentity:schemas:store:2.0:UserStoreAdapter" ] }
```

This response is a success response, although we prefer that it not be one, because it shows that any active access token referencing a valid user can be used to access any data.

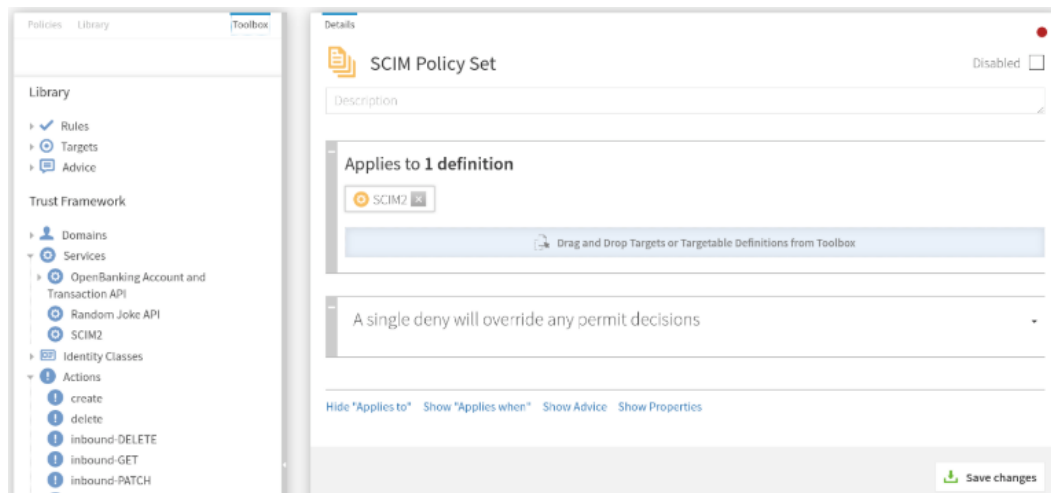
Create the policy tree

About this task

Log on to the PingDataGovernance Policy Administration GUI and click **Policies > Policy Editor**. The default policies include a single policy, named `Token Validation`, under **Global Decision Point**. This policy denies any request by using an access token if its active flag is set to `false`. This policy is augmented with a set of scope-based access control policies.

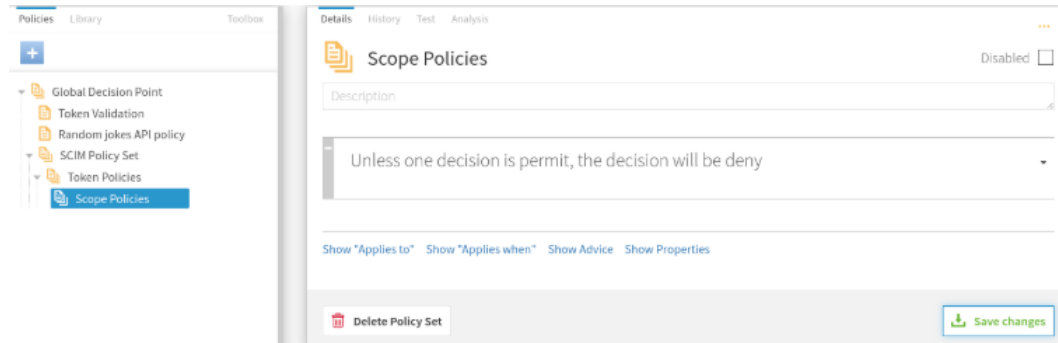
Steps

1. To create a tree structure and ensure that your policies apply only to SCIM requests, perform the following steps:
 - a) Highlight **Global Decision Point**.
 - b) Click **+**.
 - c) Click **Add Policy Set**.
 - d) In the **Name** text box, type `SCIM Policy Set`.
 - e) Click **Unless one decision is deny, the decision will be permit** and change it to **A single deny will override any permit decisions**.
 This step is known as a *combining algorithm*. It determines the manner in which the policy set resolves potentially contending decisions from child policies.
 - f) Click **Show "Applies to"**.
 - g) Click **Toolbox**.
 - h) From the **Services** list, drag `SCIM2` to the blue **Targets** box.
 This step ensures that policies in the SCIM policy set apply only to SCIM requests.
 - i) Click **Save Changes**.



2. To add a branch under the SCIM policy set to hold SCIM-specific access token policies, navigate from **Toolbox** to **Policies** and perform the following steps:
 - a) Highlight **SCIM Policy Set**.
 - b) Click **+**.
 - c) Click **Add Policy Set**.
 - d) In the **Name** text box, type `Token Policies`.
 - e) Change the combining algorithm to **A single deny will override any permit decisions**.
 - f) Click **Save Changes**.
3. To add another branch that holds a policy specific to access token scopes, perform the following steps:
 - a) Highlight **Token Policies**.
 - b) Click **+**.

- c) Click **Add Policy Set**.
- d) In the **Name** text box, type `Scope Policies`.
- e) Change the combining algorithm to **Unless one decision is permit, the decision will be deny**.
- f) Click **Save Changes**.



Create SCIM access token policies

After you define a structure, you are ready to define some policies. In this section, you will define three policies that use a requester's access token to limit its access to data.

Create a policy for permitted access token scopes

About this task

The first policy defines the access token scopes that PingDataGovernance Server accepts for SCIM requests. The following table defines these scopes.

Scope	Allowed actions	Applies to
scimAdmin	search, retrieve, create/modify, delete	Any data
email	retrieve	Requester's email attributes
profile	retrieve	Requester's profile attributes

To create the policy and add rules to define the scopes, perform the following steps:

Steps

1. Highlight **Scope Policies**.
2. Click **Show Advice**.
3. Click **Toolbox**.
4. From the **Advice** list, drag `Insufficient Scope` to the blue **Advice** box.
5. Click **Save Changes**.
6. Highlight **Scope Policies**.
7. Click **+**.
8. Click **Add Policy**.
9. In the **Name** text box, type `Permitted Scopes`.
10. Change the combining algorithm to **A single deny will override any permit decisions**.
11. Click **Save Changes**.

Test the policy with cURL

About this task

If you attempt the same HTTP request that you issued previously, it is now denied:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "nonexistent.scope", "client_id": "nonexistent.client"}'
{"schemas":["urn:ietf:params:scim:api:messages:2.0:Error"],"status":"403", "scimType":"insufficient_scope","detail":"Requested operation not allowed by the granted OAuth scopes."}
```

Define the email scope

Steps

1. Highlight **Permitted Scopes**.
2. From the **Rules** list, drag Permitted SCIM scope for user to the blue **Rules** box.
3. To the right of the copied rule, click ...
4. Click **Replace with clone**.
5. In the **Name** text box, type Scope: email.
6. To expand the rule, click **+**.
7. In the **Description** text box, type Rule that permits a SCIM user to access its own mail attribute if the access token contains the email scope..
8. In the **HttpRequest.AccessToken.scope** row of the **Condition** group box, type email in the **CHANGEME** text box.
9. Within the rule, click **Show "Applies to"**.
10. From **Actions**, drag retrieve to the blue **Targets** box.
Note: This task uses different actions from the previous gateway example.
11. Within the rule, click **Show Advice**.
12. Click **Toolbox**.
13. From **Advice**, drag Include email attributes to the blue **Advice** box.
 Note the payload for this predefined advice. If the condition for this rule is satisfied, the response includes the mail attribute.
14. Click **Save changes**.

Results

A single deny will override any permit decisions

✓ **Scope: email**
🔍 💬 ⋮

Rule that permits a SCIM user to access its own mail attribute if the access token contains the email scope.

Effect Permit

Applies to 1 definition

⚠️ retrieve

Drag and Drop Targets or Targetable Definitions from Toolbox

Condition

ALL ANY NONE Clear all

A
HttpRequest.AccessToken.scope
Contains
C
email

A
HttpRequest.AccessToken.token_
Equals
A
HttpRequest.ResourcePath

+ Comparison + Named Condition + Group Drag and Drop any Trust Framework item from Toolbox

Advice (1 in total)

Name
Include email attributes
Obligatory
☑️
⋮

+ Create new Advice or Drag and Drop Advice from Toolbox

[Hide "Applies to"](#) [Hide Advice](#) [Show Properties](#)

+ Create new Rule or Drag and Drop Rules from Toolbox

Test the email scope with cURL

About this task

If you make the same request as earlier, a 403 is returned because the provided scope is not allowed:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me \
-H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "nonexistent.scope", "client_id": "nonexistent.client"}'
```

If you adjust the request to use the email scope, the request succeeds, and only the mail attribute is returned:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "email", "client_id": "nonexistent.client"}'
{"id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "meta": {"resourceType": "Users", "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"}, "schemas": ["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"], "mail": ["user.1@example.com"]}
```

Define the profile scope

Steps

1. From the **Rules** list, drag Permitted SCIM scope for user to the blue **Rules** box.
2. To the right of the copied rule, click
3. Click **Replace with clone**.
4. In the **Name** text box, type `Scope: profile`.
5. To expand the rule, click **+**.
6. In the **Description** text box, type Rule that permits a SCIM user to access a subset of its own profile attributes if the access token contains the profile scope.
7. In the **HttpRequest.AccessToken.scope** row of the **Condition** group box, type `profile` in the **CHANGEME** text box.
8. Within the rule, click **Show "Applies to"**.
9. From **Actions**, drag `retrieve` to the blue **Targets** box.
10. Within the rule, click **Show Advice**.
11. Click **Toolbox**.
12. From **Advice**, drag `Include profile attributes` to the blue **Advice** box.
Note the payload for this predefined advice. If the condition for this rule is satisfied, the response includes the `uid`, `sn`, `givenName`, and `description` attributes.
13. Click **Save changes**.

Results

The screenshot displays the configuration interface for a rule named "Scope: profile". The rule's description is "Rule that permits a SCIM user to access a subset of its own attributes if the access token contains the profile scope." The rule is set to "Effect: Permit" and "Applies to 1 definition". The action is "retrieve". The condition is defined as "ALL" of the following: "HttpRequest.AccessToken.scope" contains "profile" and "HttpRequest.AccessToken.token_" equals "HttpRequest.ResourcePath". The advice section shows one advice named "Include profile attributes", which is obligatory and has a description: "Allows a client to read a set of specific user profile attributes. Note that these attributes are schema-specific and may need to be changed to reflect your user schema." The code for the advice is "include-attributes", it applies to "Permit", and the payload is ["uid", "sn", "givenName", "description"].

Scope: profile ⓘ 💬 ⋮

Rule that permits a SCIM user to access a subset of its own attributes if the access token contains the profile scope.

Effect: **Permit**

Applies to 1 definition

retrieve ✕

Drag and Drop Targets or Targetable Definitions from Toolbox

Condition

ALL ANY NONE Clear all

A	HttpRequest.AccessToken.scope	Contains	C	profile
A	HttpRequest.AccessToken.token_	Equals	A	HttpRequest.ResourcePath

+ Comparison + Named Condition + Group Drag and Drop any Trust Framework item from Toolbox

Advice (1 in total)

Name 📄 Include profile attributes Obligatory ☒ ⋮

Description Allows a client to read a set of specific user profile attributes. Note that these attributes are schema-specific and may need to be changed to reflect your user schema.

Code include-attributes **Applies To** Permit

Payload ["uid", "sn", "givenName", "description"] 🏷️

+ Create new Advice or Drag and Drop Advice from Toolbox

Test the profile scope with cURL

About this task

Make the same request as earlier, but change the email scope that the access token uses to profile:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "profile", "client_id": "nonexistent.client"}'
{"id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "meta": {"resourceType": "Users", "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"}, "schemas": ["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"], "uid": ["user.1"], "givenName": ["Romina"], "description": ["This is the description for Romina Valerio."], "sn": ["Valerio"]}
```

The attributes defined by the new rule's advice are returned.

Because an access token might contain multiple scopes, confirm that an access token with the email and profile scopes returns the union of the attributes that both scopes grant:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "email profile", "client_id": "nonexistent.client"}'
{"id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "meta": {"resourceType": "Users", "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"}, "schemas": ["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"], "uid": ["user.1"], "mail": ["user.1@example.com"], "givenName": ["Romina"], "description": ["This is the description for Romina Valerio."], "sn": ["Valerio"]}
```

Define the scimAdmin scope

For the `scimAdmin` scope, you will define different behaviors that depend on the action of the request. As a result, the scope definition will be split into multiple rules.

Add the scimAdmin retrieve rule

Steps

1. Highlight **Permitted Scopes**.
2. Click **Create new Rule**.
3. In the **Name** text box, type `Scope: scimAdmin (retrieve)`.
4. From the **Effect** drop-down list, select **Permit**.
5. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.scope`.
 - c) From the comparator drop-down list, select **Contains**.
 - d) In the final text box, type `scimAdmin`.
6. Within the rule, click **Show "Applies to"**.
7. Click **Toolbox**.
8. From **Actions**, drag **retrieve** to the blue **Targets** box.
9. Within the rule, click **Show Advice**.
10. From **Advice**, drag **Include all attributes** to the blue **Advice** box.
11. Click **Save Changes**.

Results

Scope: scimAdmin (retrieve)

Description

Effect: Permit

Applies to 1 definition

retrieve

Drag and Drop Targets or Targetable Definitions from Toolbox

Condition

A HttpRequest.AccessToken.scope Contains C scimAdmin

+ Comparison + Named Condition + Group

Drag and Drop any Trust Framework item from Toolbox

Advice (1 in total)

Name Include all attributes Obligatory

+ Create new Advice or Drag and Drop Advice from Toolbox

[Hide "Applies to"](#) [Hide Advice](#) [Show Properties](#)

Add the scimAdmin create/modify rule

Steps

1. Click **Create new Rule**.
2. In the **Name** text box, type `Scope: scimAdmin (create/modify)`.
3. From the **Effect** drop-down list, select **Permit**.
4. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.scope`.
 - c) From the comparator drop-down list, select **Contains**.
 - d) In the final text box, type `scimAdmin`.
5. Within the rule, click **Show "Applies to"**.
6. Click **Toolbox**.
7. From **Actions**, drag `create` to the blue **Targets** box.
8. From **Actions**, drag `modify` to the blue **Targets** box.
9. Within the rule, click **Show Advice > Create new advice**.
10. In the **Name** text box, type `Allow certain attributes to be created or updated`.
11. Select the **Obligatory** check box.
12. In the **Code** text box, type `allow-attributes`.
13. From the **Applies to** drop-down list, select **Permit**.
14. Click **+ Payload**.
15. In the **Payload** text box, type the following content:

```
[ "manager", "uid", "mail", "sn", "givenName", "cn", "description", "l",
  "st", "country", "postalAddress", "mobile", "homePhone" ]
```

Note: This example arbitrarily restricts the attributes that can be set during a create or modify operation. To allow all attributes, set the **Payload** value to ["*"].

16. Click **Save Changes**.

Results

The screenshot displays the configuration interface for a rule named "Scope: scimAdmin (create/modify)". The interface includes the following sections:

- Description:** A text box for describing the rule.
- Effect:** A dropdown menu set to "Permit".
- Applies to 2 definitions:** Two buttons labeled "create" and "modify" with warning icons.
- Condition:** A section with a dropdown set to "A", a text box containing "HttpRequest.AccessToken.scope", a dropdown set to "Contains", and another text box containing "scimAdmin". Below this are buttons for "+ Comparison", "+ Named Condition", and "+ Group", along with a "Drag and Drop any Trust Framework item from Toolbox" area.
- Advice (1 in total):** A section containing a table with the following details:

Name	Allow certain attributes to be created or updated	Obligatory	<input checked="" type="checkbox"/>
Description			
Code	allow-attributes	Applies To	Permit
Payload	["manager", "uid", "mail", "sn", "givenName", "cn", "description", "l", "st", "country", "postalAddress", "mobile", "homePhone"]		

 Below the table are buttons for "+ Attributes" and "+ Services".
- Footer:** A section with a "+ Create new Advice" button, an "or" separator, and a "Drag and Drop Advice from Toolbox" area. At the bottom, there are links: "Hide 'Applies to'", "Hide Advice", and "Show Properties".

Add the scimAdmin search rule

Steps

1. Click **Create new Rule**.
2. In the **Name** text box, type `Scope: scimAdmin (search)`.
3. From the **Effect** drop-down list, select **Permit**.
4. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.scope`.
 - c) From the comparator drop-down list, select **Contains**.
 - d) In the final text box, type `scimAdmin`.
5. Within the rule, click **Show "Applies to"**.
6. Click **Toolbox**.
7. From **Actions**, drag `search` to the blue **Targets** box.
8. Click **Save Changes**.

Add the scimAdmin delete rule

Steps

1. Click **Create new Rule**.
2. In the **Name** text box, type `Scope: scimAdmin (delete)`.

3. From the **Effect** drop-down list, select `Permit`.
4. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.scope`.
 - c) From the comparator drop-down list, select `Contains`.
 - d) In the final text box, type `scimAdmin`.
5. Within the rule, click **Show "Applies to"**.
6. Click **Toolbox**.
7. From **Actions**, drag `delete` to the blue **Targets** box.
8. Click **Save Changes**.

Create a policy for permitted OAuth2 clients

About this task


A REST service typically allows only requests from a whitelist of OAuth2 clients. In this section, you will define a policy in which each rule specifies an allowed client.

Steps

1. On the **Policies** page, highlight **Token Policies** and click **+**.
2. Click **Add Policy**.
3. In the **Name** text box, type `Permitted Clients`.
4. Change the combining algorithm to **Unless one decision is permit, the decision will be deny**.
5. Click **Create new Rule**.
6. In the **Name** text box, type `Client: client1`.
7. From the **Effect** drop-down list, select `Permit`.
8. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.client_id`.
 - c) From the comparator drop-down list, select `Equals`.
 - d) In the final text box, type `client1`.
9. Click **Create new Rule**.
10. In the **Name** text box, type `Client: client2`.
11. From the **Effect** drop-down list, select `Permit`.
12. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.client_id`.
 - c) From the comparator drop-down list, select `Equals`.
 - d) In the final text box, type `client2`.
13. At the policy level, click **Show Advice**.


Note: Do not click **Show Advice** within the `client1` or `client2` rules.
14. From **Advice**, drag `Unauthorized Client` to the blue **Advice** box.
15. Click **Save changes**.


Results


Permitted Clients
Disabled ☐

Description

Unless one decision is permit, the decision will be deny

+  Client: client1

+  Client: client2

Description


Effect Permit

Condition


HttpRequest.AccessToken.client_id Equals C client2


+ Comparison + Condition + Group

[Show "Applies to"](#) [Hide Advice](#) [Show Properties](#)

+ Create new Rule or  Drag and Drop Rules from Toolbox

Advice (1 in total)

Name  Unauthorized Client Obligatory ☒

+ Create new Advice or  Drag and Drop Advice from Toolbox

Test the client policy with cURL

About this task

After completing the tasks in the previous sections, an access token for any client other than client1 or client2 is rejected.

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "email", "client_id": "nonexistent.client"}'
{"schemas": [
  "urn:ietf:params:scim:api:messages:2.0:Error"], "status": "401", "scimType": "The client is not authorized to request this resource.", "detail": "unauthorized_client"}
```

An access token for client1 is now accepted.

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": "email", "client_id": "client1"}'
{"id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e", "meta": {
  "resourceType": "Users", "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"}, "schemas": [
  "urn:pingidentity:schemas:store:2.0:UserStoreAdapter"], "mail": [
  "user.1@example.com"]}
```

Create a policy for permitted audiences

About this task


An authorization server like PingFederate might set an `audience` field on the access tokens that it issues, naming one or more services that are allowed to accept the access token. A REST service can use the `audience` field to ensure that it does not accept access tokens that are intended for use with a different service.

As with the Permitted Clients policy, each rule in the Permitted Audiences policy defines an acceptable audience value.

Steps


1. Highlight **Token Policies**.
2. Click **+**.
3. Click **Add Policy**.
4. In the **Name** text box, type `Permitted Audiences`.
5. Change the combining algorithm to **Unless one decision is permit, the decision will be deny**.
6. Click **Create new Rule**.
7. In the **Name** text box, type `Audience: https://example.com`.
8. From the **Effect** drop-down list, select `Permit`.
9. In the **Condition** group box, perform the following steps:
 - a) Click **+ Condition**.
 - b) In the first text box, type `HttpRequest.AccessToken.audience`.
 - c) From the comparator drop-down list, select `Equals`.
 - d) In the final text box, type `https://example.com`.
10. At the policy level, click **Show Advice**.
11. From **Toolbox > Advice**, drag `Unauthorized Audience` to the blue **Advice** box.
12. Click **Save changes**.

Results


Permitted Audiences
Disabled ☐

Description

Unless one decision is permit, the decision will be deny


Audience: https://example.com
...

Description

Effect Permit

Condition

HttpRequest.AccessToken.audience
 Equals
 C https://example.com


+ Comparison
+ Condition
+ Group

[Show "Applies to"](#)
[Show Advice](#)
[Show Properties](#)

+ Create new Rule or Drag and Drop Rules from Toolbox

Advice (1 in total)

Name


 Unauthorized Audience

Obligatory ☒
...

+ Create new Advice or Drag and Drop Advice from Toolbox

Test the audience policy with cURL

About this task

An access token without a specific audience value is expected to be rejected.

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": {"email", "client_id": "client1"}}'
```

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:Error"
  ],
  "status": "403",
  "scimType": "invalid_token",
  "detail": "The access token was issued for a different audience."
}
```

An access token with an audience value of https://example.com is accepted.

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.1", "scope": {"email", "client_id": "client1", "aud": "https://example.com"}}'
```

```
{
  "id": "355a133d-58ea-3827-8e8d-b39cf74ddb3e",
  "meta": {
    "resourceType": "Users",
    "location": "https://<your-dg-host>:<your-https-dg-port>/scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"
  },
  "schemas": [
    "urn:pingidentity:schemas:store:2.0:UserStoreAdapter"
  ],
  "mail": [
    "user.1@example.com"
  ]
}
```

Create a policy for role-based access control

About this task

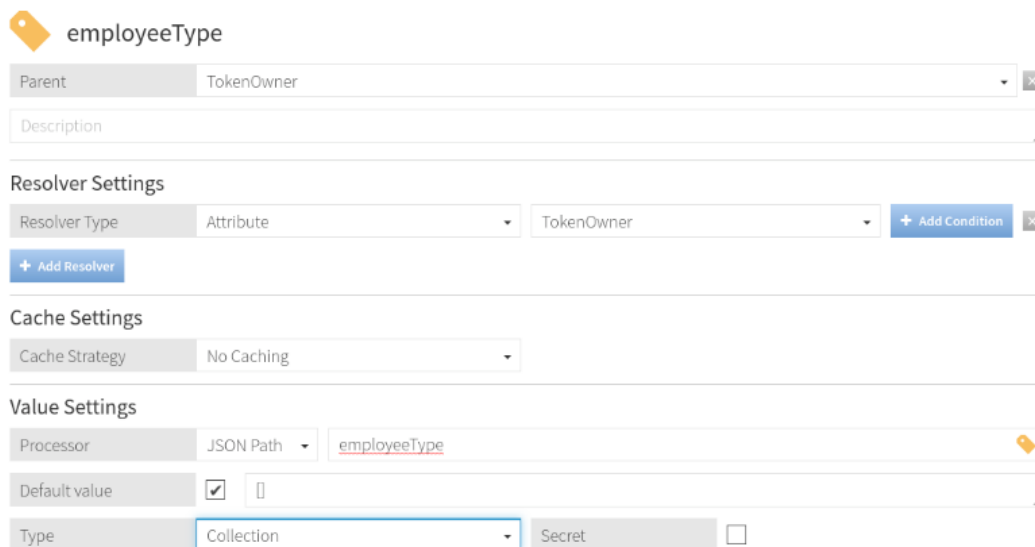
The final policy illustrates the manner in which an access-control rule can base its authorization decision on an attribute of the requesting identity, rather than on an access token claim.

When PingDataGovernance Server authorizes a request, an access token validator resolves the subject of the access token to a SCIM user, and populates a policy request attribute called `TokenOwner` with the SCIM user's attributes. In this scenario, build a policy around the `employeeType` attribute, which must be defined in the Trust Framework.

Steps

1. On **Trust Framework > Attributes**, click **TokenOwner**.
2. Click **+**.
3. Click **Add new Attribute**.
4. In the **Name** text box, type `employeeType`.
5. From the **Parent** drop-down list, select `TokenOwner`.
6. In the **Resolver Settings** section, perform the following steps:
 - a) Click **Add Resolver**.
 - b) From the **Resolver Type** drop-down list, select `Attribute` and specify a value of `TokenOwner`.
7. In the **Value Settings** section, perform the following steps:
 - a) From the **Processor** drop-down list, select `JSON Path` and type a value of `employeeType`.
 - b) Select the **Default Value** check box and type of value of `[]`.
An empty array is specified as the default value because not all users have an `employeeType` attribute. A default value of `[]` ensures that policies can safely use this attribute to define conditions.
 - c) From the **Type** drop-down list, select `Collection`.
8. Click **Save changes**.

Results



The screenshot displays the configuration page for the `employeeType` attribute. The interface is organized into several sections:


- Parent:** A dropdown menu set to `TokenOwner`.
- Description:** A text input field.
- Resolver Settings:**
 - Resolver Type:** A dropdown menu set to `Attribute`.
 - TokenOwner:** A dropdown menu set to `TokenOwner`.
 - + Add Condition:** A button to add a condition.
 - + Add Resolver:** A button to add a resolver.
- Cache Settings:**
 - Cache Strategy:** A dropdown menu set to `No Caching`.
- Value Settings:**
 - Processor:** A dropdown menu set to `JSON Path`.
 - JSON Path:** A text input field containing `employeeType`.
 - Default value:** A checkbox checked, with a text input field containing `[]`.
 - Type:** A dropdown menu set to `Collection`.
 - Secret:** A checkbox unchecked.

Next steps

Add a policy that uses the `employeeType` attribute.


1. Highlight **SCIM Policy Set**.

2. Click **+**.
3. Click **Add Policy**.
4. In the **Name** text box, type `Restrict Intern Access`.
5. Change the combining algorithm to **Unless one decision is deny, the decision will be permit**.
6. Click **Create new Rule**.
7. In the **Name** text box, type `Restrict access for interns`.
8. From the **Effect** drop-down list, select `Permit`.
9. In the **Condition** group box, perform the following steps:
 - a. Click **+ Condition**.
 - b. In the first text box, type `TokenOwner.employeeType`.
 - c. From the comparator drop-down list, select `Contains`.
 - d. In the final text box, type `intern`.
10. Within the rule, click **Show Advice > Create New Advice**.
11. In the **Name** text box, type `Restrict attributes visible to interns`.
12. Select the **Obligatory** check box.
13. In the **Code** text box, type `exclude-attributes`.
14. From the **Applies to** drop-down list, select `Permit`.
15. Click **+ Payload**.
16. In the **Payload** text box, type `["description"]`.
17. Click **Save Changes**.


Restrict Intern Access
Disabled ☐

Description

Unless one decision is deny, the decision will be permit


Restrict access for interns
...

Description

Effect Permit

Condition

TokenOwner.employeeType
 Contains
 C intern

+ Comparison
+ Condition
+ Group

Advice (1 in total)

Name
 Restrict attributes visible to interns
 Obligatory ☒

Description

Code
 exclude-attributes
 Applies To
 Permit

Payload
 ["description"]

+ Attributes

+ Create new Advice
or
Drag and Drop Advice from Toolbox

[Show "Applies to"](#)
[Show Properties](#)

+ Create new Rule
or
Drag and Drop Rules from Toolbox

Test the policy with cURL

About this task

PingDataGovernance's sample user data allows an `employeeType` attribute but does not populate it with values for any users. To modify a user entry, create a file named `user2-to-intern.ldif` with the following contents:

```
dn: uid=user.2,ou=people,dc=example,dc=com
changetype: modify
add: employeeType
employeeType: intern
```

Run the following command to update `user.2`:

```
PingDirectory/bin/ldapmodify --port 1636 --useSSL --trustAll --bindDN
"cn=directory manager" --bindPassword <your-ds-password> -f user2-to-
intern.ldif
```

Confirm that the user cannot read the `description` attribute, even though the profile scope allows it:

```
curl --insecure -X GET https://<your-dg-host>:<your-https-dg-port>/scim/
v2/Me -H 'Authorization: Bearer {"active": true, "sub": "user.2", "scope":
"profile", "client_id": "client1", "aud": "https://example.com"}'
{"id": "c9cbfb8c-d915-3de3-8a2c-a01c0ccc6d09", "meta":
{"resourceType": "Users", "location": "https://<your-dg-host>:<your-https-
dg-port>/scim/v2/Users/c9cbfb8c-d915-3de3-8a2c-a01c0ccc6d09"}, "schemas":
["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"], "uid":
["user.2"], "givenName": ["Billy"], "sn": ["Zaleski"]}
```

Example files

The compressed PingDataGovernance Server file at `PingDataGovernance/resource/policies` includes a policy snapshot and deployment package that contains an example Trust Framework as well as example policies.

About the API security gateway

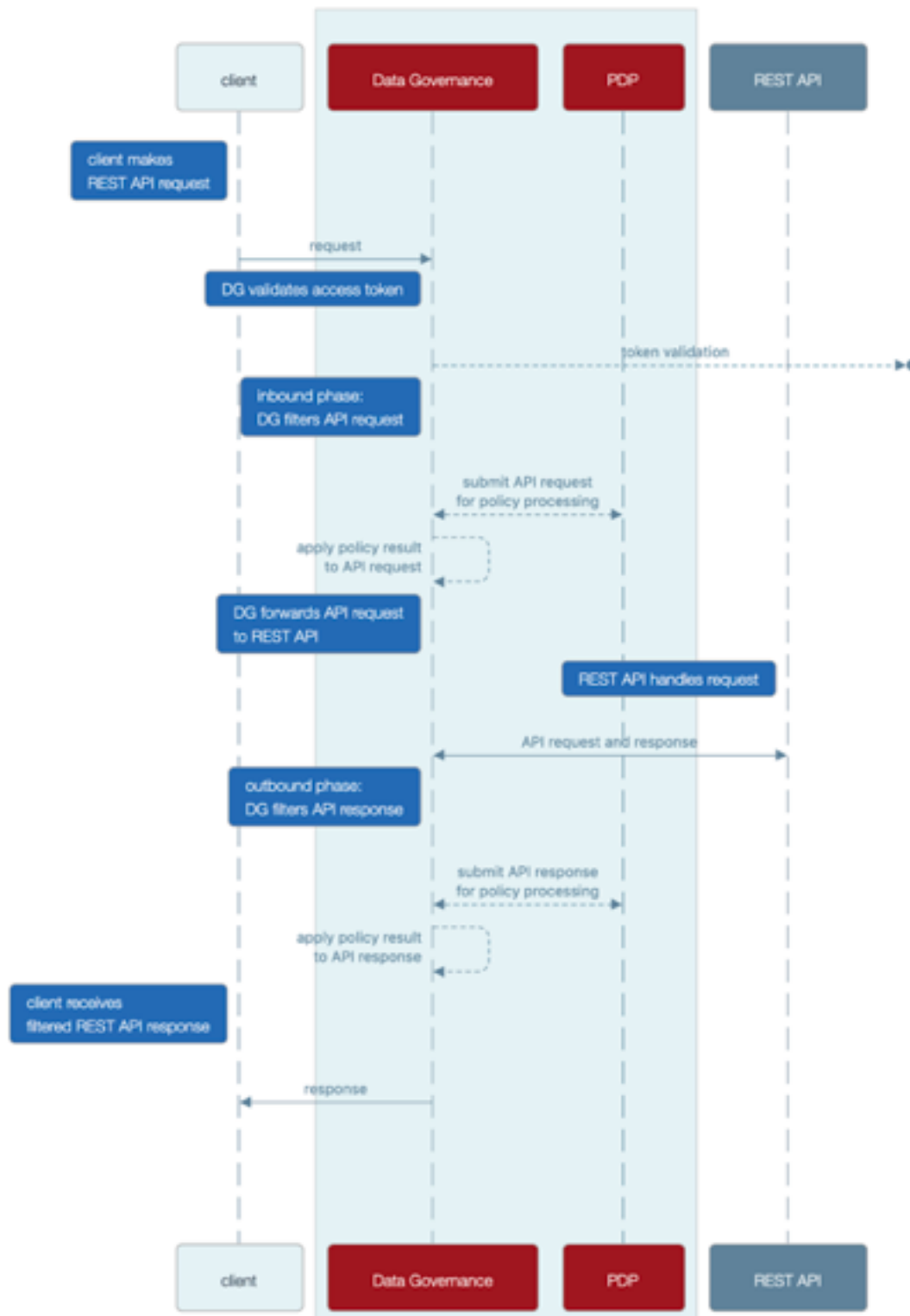
PingDataGovernance Server and its API security gateway act as an intermediary between a client and an API server.

Request and response flow

The gateway handles proxied requests in the following phases:

- Inbound phase – When a client submits an API request to PingDataGovernance Server, the gateway forms a policy request based on the API request and submits it to the PDP for evaluation. If the policy result allows it, PingDataGovernance Server forwards the request to the API server.
- Outbound phase – After PingDataGovernance Server receives the upstream API server's response, the gateway again forms a policy request, this time based on the API server response, and submits it to the PDP. If the policy result is positive, PingDataGovernance Server forwards the response to the client.

Data Governance API Security Gateway sequence diagram



The API gateway supports only JSON requests and responses.

Gateway configuration basics

The API security gateway consists of the following components:

- One or more gateway HTTP servlet extensions
- One or more Gateway API Endpoints
- One or more API external servers

An API external server represents the upstream API server and contains the configuration for the server's protocol scheme, host name, port, and connection security. The server can be created in the PingDataGovernance Administration Console, or with the following example command:

```
PingDataGovernance/bin/dsconfig create-external-server \
  --server-name "API Server" \
  --type api \
  --set base-url:https://api-service.example.com:1443
```

A Gateway API Endpoint represents a public path prefix that PingDataGovernance Server accepts for handling proxied requests. A Gateway API Endpoint configuration defines the base path for receiving requests (inbound-base-path) as well as the base path for forwarding the request to the API server (outbound-base-path). It also defines the associated API external server and other properties that relate to policy processing, such as service, which targets the policy requests generated for the Gateway API Endpoint to specific policies.

The following example commands use the API external server from the previous example to create a pair of Gateway API Endpoints:

```
PingDataGovernance/bin/dsconfig create-gateway-api-endpoint \
  --endpoint-name "Consent Definitions" \
  --set inbound-base-path:/c/definitions \
  --set outbound-base-path:/consent/v1/definitions \
  --set "api-server:API Server" \
  --set service:Consent

PingDataGovernance/bin/dsconfig create-gateway-api-endpoint \
  --endpoint-name "Consent Records" \
  --set inbound-base-path:/c/consents \
  --set outbound-base-path:/consent/v1/consents \
  --set "api-server:API Server" \
  --set service:Consent
```

The gateway HTTP servlet extension is the server component that represents the API security gateway itself. In most cases, you do not need to configure this component.

Changes to these components do not typically require a server restart to take effect. For more information about configuration options, refer to the *Configuration Reference Guide* that is bundled with the product.

API security gateway authentication

Although the gateway does not strictly require the authentication of requests, the default policy set requires bearer token authentication.

To support this approach, the gateway uses the configured access token validators to evaluate bearer tokens that are included in incoming requests. The result of that validation is supplied to the policy request in the `HttpRequest.AccessToken` attribute, and the user identity that is associated with the token is provided in the `TokenOwner` attribute.

Policies use this authentication information to affect the processing of requests and responses. For example, a policy in the default policy set requires all requests to be made with an active access token.

```
Rule: Deny if HttpRequest.AccessToken.active Equals false

Advice:
  Code: denied-reason
  Applies To: Deny
  Payload: {"status":401, "message": "invalid_token", "detail":"Access token
is expired or otherwise invalid"}
```

Gateway API Endpoints include the following configuration properties to specify the manner in which they handle authentication.

Property	Description
http-auth-evaluation-behavior	Determines whether the Gateway API Endpoint evaluates bearer tokens, and if so, whether the bearer token is forwarded to the API server.
access-token-validator	<p>Sets the access token validators that the Gateway API Endpoint uses. By default, this property has no value, and the Gateway API Endpoint can evaluate every bearer token by using each access token validator that is configured on the server. To constrain the set of access token validators that a Gateway API Endpoint uses, set this property to use one or more specific values.</p> <p>If http-auth-evaluation-behavior is set to do-not-evaluate, this setting is ignored.</p>

API security gateway policy requests

Before accepting an incoming request and forwarding it to the API server, the gateway creates a policy request that is based on the request and sends it to the PDP for authorization. Before accepting an API server response and forwarding it back to the client, the gateway creates a policy request that is based on the request and response, and sends it to the PDP for authorization. An understanding of the manner in which the gateway formulates policy requests can help you create and troubleshoot policies more effectively.

We recommend enabling detailed decision logging and viewing all policy request attributes in action, particularly when first developing API security gateway policies. For more information, see [Policy Decision logger](#) on page 109.

Policy request attributes

The following table identifies the attributes of a policy request that the gateway generates.

Policy request attributes	Description	Type
<code>action</code>	Identifies the gateway request processing phase and the HTTP method, such as GET or POST. The value is formatted as <code><phase>-<method></code> . Example values include <code>inbound-GET</code> , <code>inbound-POST</code> , <code>outbound-GET</code> , and <code>outbound-POST</code> .	String
<code>service</code>	Identifies the API service. By default, this attribute is set to the name of the Gateway API Endpoint, which can be overridden by setting the Gateway API Endpoint's <code>service</code> property. Multiple Gateway API Endpoints can use the same service value.	String
<code>domain</code>	Unused.	String
<code>identityProvider</code>	Identifies the access token validator that evaluates the bearer token used in an incoming request.	String
<code>attributes</code>	Identifies additional attributes that do not correspond to a specific entity type in the PingDataGovernance trust framework. For more information about these attributes, see the following table.	Object

The following table identifies the additional attributes that are included in `attributes`.

Attribute	Description	Type
<code>HttpRequest</code>	Identifies the HTTP request.	Object
<code>TokenOwner</code>	The access token subject as a SCIM resource, as obtained by the access token validator.	Object
<code>Gateway</code>	Provides additional gateway-specific information about the request.	Object

The following table identifies the fields that the `HttpRequest` attribute contains.

Attribute	Description	Type
<code>RequestURI</code>	The request URI.	String
<code>Headers</code>	Request/response headers.	Object

Attribute	Description	Type
ResourcePath	Portion of the request URI path following the inbound base path that the Gateway API Endpoint defines.	String
QueryParameters	Request URI query parameters.	Object
AccessToken	Parsed access token. For more information, see the following table.	Object
RequestBody	The request body, if available.	Object
ResponseBody	The response body, if available. This attribute is provided only for outbound policy requests.	Object
ClientCertificate	Properties of the client certificate, if one was used.	Object

The access token validator populates the `HttpRequest.AccessToken` attribute, which contains the fields in the following table. These fields correspond approximately to the fields that the IETF Token Introspection specification ([RFC 7662](#)) defines.

Attribute	Description	Type
client_id	The client ID of the application that was granted the access token.	String
audience	Identifies the recipients for whom the access token is intended. Typically, the authorization server sets this field to indicate the resource servers that might accept the token.	Array
user_token	Flag that the access token validator sets to indicate that the token was issued originally to a subject. If this flag is <code>false</code> , the token does not have a subject and was issued directly to a client.	Boolean
subject	Token subject. This attribute is a user identifier that the authorization server sets.	String
token_owner	User identifier that was resolved by the access token validator's token resource lookup method. This attribute is always a SCIM ID of the form <code><resource type>/<resource ID></code> .	String
username	Subject's user name. This attribute is a user identifier that the authorization server sets.	String

Attribute	Description	Type
issuer	Token issuer. This attribute is usually a URI that identifies the authorization server.	String
issued_at	Date and time at which the access token was issued.	DateTime
expiration	Date and time at which the access token expires.	DateTime
not_before	Date and time before which a resource server does not accept the access token.	DateTime
token_type	The token type, as set by the authorization server. This value is typically set to <code>bearer</code> .	String

The following table identifies the fields that the `HttpRequest.ClientCertificate` attribute contains.

Attribute	Description	Type
algorithm	Name of the certificate signature algorithm, such as <code>SHA256withRSA</code> .	String
algorithmOID	Signature algorithm OID.	String
notBefore	Earliest date on which the certificate is considered valid.	DateTime
notAfter	Expiration date and time of the certificate.	DateTime
issuer	Distinguished name (DN) of the certificate issuer.	String
subject	DN of the certificate subject.	String
valid	Indicates whether the certificate is valid.	Boolean

The following table identifies the fields that the `Gateway` attribute contains.

Attribute	Description	Type
<code>_BasePath</code>	Portion of the HTTP request URI that matches the Gateway API Endpoint's <code>inbound-base-path</code> value.	String
<code>_TrailingPath</code>	Portion of the HTTP request URI that follows the <code>_BasePath</code> .	String
<i>base path parameters</i>	Parameters used in a Gateway API Endpoint's <code>inbound-base-path</code> configuration property are included as fields of the <code>Gateway</code> attribute.	String

Attribute	Description	Type
<i>custom attribute</i>	The Gateway attribute might contain multiple arbitrary custom attributes that are defined by the <code>policy-request-attribute</code> of the Gateway API Endpoint configuration.	String

Gateway API Endpoint configuration properties that affect policy requests

The following table identifies Gateway API Endpoint properties that might force the inclusion of additional attributes in a policy request.

Gateway API Endpoint property	Description
<code>inbound-base-path</code>	<p>Defines the URI path prefix that the gateway uses to determine whether the Gateway API Endpoint handles a request.</p> <p>The <code>inbound-base-path</code> property value can include parameters. If parameters are found and matched, they are included as attributes to policy requests.</p> <p>The following configuration properties reference parameters that the <code>inbound-base-path</code> introduces:</p> <ul style="list-style-type: none"> <code>outbound-base-path</code> <code>service</code> <code>resource-path</code> <code>policy-request-attribute</code>
<code>service</code>	<p>Identifies the API service to the PDP.</p> <p>The service value appears in the policy request as the <code>service</code> attribute.</p> <p>If undefined, the service value defaults to the name of the Gateway API Endpoint.</p>
<code>resource-path</code>	<p>Identifies the REST resource to the PDP.</p> <p>The resource path value appears in the policy request as the <code>HttpRequest.ResourcePath</code> attribute.</p> <p>If undefined, the resource path value defaults to the portion of the request that follows the base path defined by <code>inbound-base-path</code>.</p>
<code>policy-request-attribute</code>	<p>Defines zero or more static, arbitrary key-value pairs. If specified, key-value pairs are always added as attributes to policy requests.</p> <p>These custom attributes appear in the policy request as fields of the Gateway attribute. For example, if a value of <code>policy-request-attribute</code> is <code>foo=bar</code>, the attribute <code>Gateway.foo</code> is added to the policy request with a value of <code>bar</code>.</p>

Path parameters

As stated previously, the `inbound-base-path` property value can include parameters. If parameters are found and matched, they are included in policy requests as fields of the `Gateway` policy request attribute. The previous table identifies additional configuration properties that can use these parameters.

Parameters must be introduced by the `inbound-base-path` property. Other configuration properties cannot introduce new parameters.

Basic example

Given the following example configuration:

Gateway API Endpoint property	Example value
<code>inbound-base-path</code>	<code>/accounts/<accountId>/transactions</code>
<code>outbound-base-path</code>	<code>/api/v1/accounts/<accountId>/transactions</code>
<code>policy-request-attribute</code>	<code>foo=bar</code>

A request URI with the path `/accounts/XYZ/transactions/1234` matches the inbound base path and is mapped to the outbound path `/api/v1/accounts/XYZ/transactions/1234`.

The following properties are added to the policy request:

- `HttpRequest.ResourcePath` : `1234`
- `Gateway.accountId` : `XYZ`
- `Gateway.foo` : `bar`

Advanced example

Given the following example configuration:

Gateway API Endpoint property	Example value
<code>inbound-base-path</code>	<code>/health/<tenant>/<resourceType></code>
<code>outbound-base-path</code>	<code>/api/v1/health/<tenant>/<resourceType></code>
<code>service</code>	<code>HealthAPI.<resourceType></code>
<code>resource-path</code>	<code><resourceType>/<_TrailingPath></code>

A request URI with the path `/health/OmniCorp/patients/1234` matches the inbound base path and is mapped to the outbound path `/api/v1/health/OmniCorp/patients/1234`.

The following properties are added to the policy request:

- `service` : `HealthAPI.patients`
- `HttpRequest.ResourcePath` : `patients/1234`
- `Gateway.tenant` : `OmniCorp`
- `Gateway.resourceType` : `patients`

About error templates

REST API clients are often written with the expectation that the API produces a custom error format. Some clients might fail unexpectedly if they encounter an error response that uses an unexpected format.

When a REST API is proxied by PingDataGovernance Server, errors that the REST API returns are forwarded to the client as is, unless a policy dictates a modification of the response. In the following scenarios, PingDataGovernance Server returns a gateway-generated error:

- When the policy evaluation results in a `deny` response. This scenario typically results in a 403 error.
- When an internal error occurs in the gateway, or when the gateway cannot contact the REST API service. This scenario typically results in a 500, 502, or 504 error.

By default, these responses use a simple error format, as the following example shows:

```
{
  "errorMessage": "Access Denied",
  "status": 403
}
```

The following table describes this default error format.

Field	Type	Description
errorMessage	String	Error message
status	Number	HTTP status code

Because some REST API clients expect a specific error response format, PingDataGovernance Server provides a facility for responding with custom errors, called *error templates*. An error template is written in [Velocity Template Language](#) and defines the manner in which a Gateway API Endpoint produces error responses.

Error templates feature the following context parameters:

Parameter	Type	Description
status	Integer	HTTP status
message	String	Exception message

For more information, see [Error templates](#) on page 64.

Example

The example in this section demonstrates the configuration of a custom error template for a Gateway API Endpoint named `Test API`. Error responses that use this error template feature the following fields:

- `code`
- `message`

Perform the following steps:

1. Create a file named `error-template.vtl` with the following contents:

```
#set ($code = "UNEXPECTED_ERROR")
#if($status == 403)
  #set ($code = "ACCESS_FAILED")
#end
{
  "code": "$code",
  "message": "$message"
}
```

2. Add the error template to the configuration, as follows:

```
dsconfig create-error-template \
  --template-name "Custom Error Template" \
  --set "velocity-template<error-template.vtl"
```

3. Assign the error template to the Gateway API Endpoint, as follows:

```
dsconfig set-gateway-api-endpoint-prop \
  --endpoint-name "Test API" \
  --set "error-template:Custom Error Template"
```

The error template is used whenever the gateway generates an error in response to a request. For example, a policy deny results in a response like the following example:

```
HTTP/1.1 403 Forbidden
Content-Length: 57
Content-Type: application/json;charset=utf-8
Correlation-Id: e7c8fb82-f43e-4678-b7ff-ae8252411513
Date: Wed, 27 Feb 2019 05:54:50 GMT
Request-Id: 56

{
  "code": "ACCESS_FAILED",
  "message": "Access Denied"
}
```

About the Sideband API

As a gateway, PingDataGovernance Server functions as a reverse proxy that performs the following steps:

- Intercepts client traffic to a backend REST API service.
- Authorizes the traffic to a policy decision point (PDP) that operates in one of the following locations:
 - Within the PingDataGovernance process. This mode is known as *Embedded PDP mode*.
 - Outside the PingDataGovernance process. This mode is known as *External PDP mode*.

Using the Sideband API, PingDataGovernance Server can be configured instead as a plugin to an external API gateway. In Sideband mode, an API gateway integration point performs the following steps:

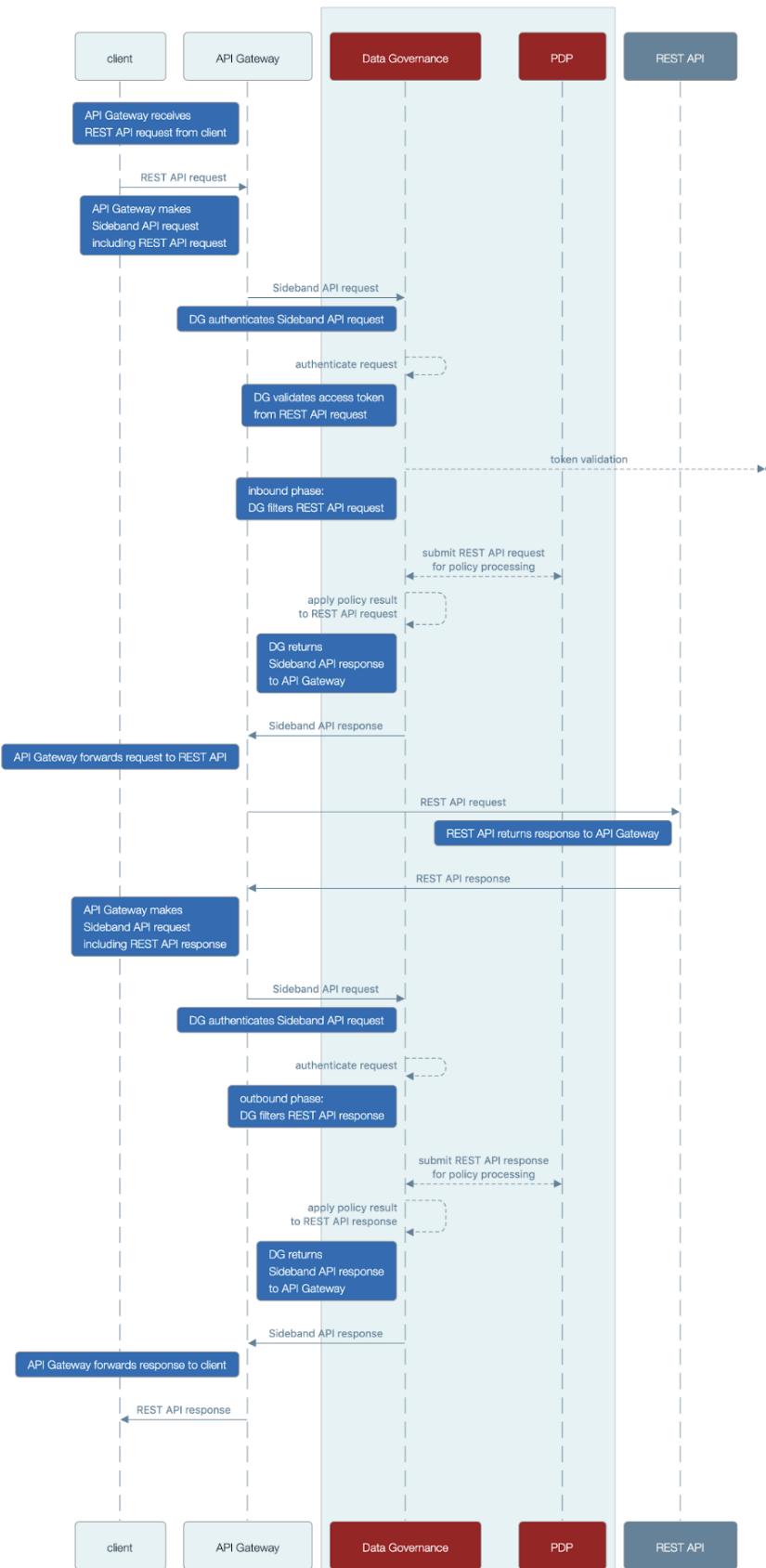
- Intercepts client traffic to a backend REST API service.
- Passes intercepted traffic to the PingDataGovernance Sideband API.

The Sideband API authorizes requests and responses, and returns them in a potentially modified form, which the API gateway forwards to the backend REST API or the client.

API gateway integration

By using an API gateway plugin that acts as a client to the Sideband API, PingDataGovernance Server can be used with an external API gateway.

Data Governance API Gateway Sideband API sequence diagram



After the API gateway receives a request from an API gateway plugin, it makes a call to the Sideband API to process the request. The Sideband API returns a response that contains a modified version of the HTTP client's request, which the API gateway forwards to the REST API.

If the Sideband API returns a response that indicates the request is unauthorized or otherwise not to be forwarded, the response includes the response to return to the client. The API gateway returns the response to the client without forwarding the request to the REST API.

When the API gateway receives a response from the REST API, it makes a call to the Sideband API to process the response. The Sideband API returns a response that contains a modified version of the REST API's response, which the API gateway forwards to the client.

Sideband API configuration basics

The Sideband API consists of the following components:

- Sideband API Shared Secrets – Define the authentication credentials that the Sideband API might require an API gateway plugin to present. For more information, see [Authenticating to the Sideband API](#) on page 57.
- Sideband API HTTP Servlet Extension – Represents the Sideband API itself. If you decide to require shared secrets, you might need to configure this component. For more information, see [Authenticating to the Sideband API](#) on page 57.
- One or more Sideband API Endpoints – Represent a public path prefix that the Sideband API accepts for handling proxied requests. Specifically, a Sideband API Endpoint configuration defines the following items:
 - The base path (`base-path`) for requests that the Sideband API accepts.
 - Properties that relate to policy processing, such as `service`, which targets the policy requests that are generated for the Sideband API Endpoint to specific policies.

PingDataGovernance Server's out-of-the-box configuration includes a Default Sideband API Endpoint that accepts all API requests and generates policy requests for the service `Default`. To customize policy requests further, an administrator can create additional Sideband API Endpoints.

The following example commands create a pair of Sideband API Endpoints that target specific requests to a consent service:

```
PingDataGovernance/bin/dsconfig create-sideband-api-endpoint \
  --endpoint-name "Consent Definitions" \
  --set base-path:/c/definitions \
  --set service:Consent

PingDataGovernance/bin/dsconfig create-sideband-api-endpoint \
  --endpoint-name "Consent Records" \
  --set base-path:/c/consents \
  --set service:Consent
```

For more information about using the Sideband API Endpoint configuration to customize policy requests, see [Sideband API policy requests](#) on page 59

Note: Changes to these components do not typically require a server restart to take effect. For more information about the configuration and configuration options, refer to the *Configuration Reference Guide*, which is bundled with the product.

Sideband API authentication

The Sideband API provides the following levels of authentication:

- Authentication to the Sideband API itself

- Bearer token processing of API gateway requests

The following sections describe these authentication levels in more detail.

Authenticating to the Sideband API

The Sideband API can require an API gateway plugin to authenticate to it by using a *shared secret*. Shared secrets are defined by using Sideband API Shared Secret configuration objects, and are managed by using the Sideband API HTTP Servlet Extension.

Creating a shared secret

About this task

To create a shared secret, run the following example `dsconfig` command, substituting values of your choosing:

```
PingDataGovernance/bin/dsconfig create-sideband-api-shared-secret \
  --secret-name "Shared Secret A" \
  --set "shared-secret:secret123"
```

Note:

- The `shared-secret` property sets the value that the Sideband API requires the API gateway plugin to present. After this value is set, it is no longer visible.
- The `secret-name` property is a label that allows an administrator to distinguish one Sideband API Shared Secret from another.

A new Sideband API Shared Secret is not used until the `shared-secrets` property of the Sideband API HTTP Servlet Extension is updated. To update the `shared-secrets` property, run the following example `dsconfig` command:

```
PingDataGovernance/bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --add "shared-secrets:Shared Secret A"
```

Deleting a shared secret

About this task

To remove a Sideband API Shared Secret from use, run the following example `dsconfig` command, substituting values of your choosing:

```
PingDataGovernance/bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --remove "shared-secrets:Shared Secret A"
```

To delete a Sideband API Shared Secret, run the following example `dsconfig` command:

```
PingDataGovernance/bin/dsconfig delete-sideband-api-shared-secret \
  --secret-name "Shared Secret A"
```

Rotating shared secrets

About this task

To avoid service interruptions, the Sideband API allows multiple, distinct shared secrets to be accepted at the same time. As a result, an administrator can configure a new shared secret that the Sideband API

accepts alongside an existing shared secret. This approach allows time for the API gateway plugin to be updated to use the new shared secret.

Steps

1. Create a new Sideband API Shared Secret and assign it to the Sideband API HTTP Servlet Extension.
2. Update the API gateway plugin to use the new shared secret.
3. Remove the previous Sideband API Shared Secret.

Customizing the shared secret header

About this task

By default, the Sideband API accepts a shared secret from an API gateway plugin by way of the `PDG-TOKEN` header. To customize a shared secret header, change the value of the Sideband API HTTP Servlet Extension's `shared-secret-header` property.

For example, the following command changes the shared secret header to `x-shared-secret`:

```
PingDataGovernance/bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --set shared-secret-header-name:x-shared-secret
```

The following command resets the shared secret header to its default value:

```
PingDataGovernance/bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --reset shared-secret-header-name
```

Authenticating API server requests

About this task

As with PingDataGovernance's API Security Gateway mode, API server requests that the Sideband API authorizes do not strictly require authentication. However, the default policy set requires bearer token authentication.

To support this level of authentication, the Sideband API uses configured Access Token Validators to evaluate bearer tokens that are included in incoming requests. The `HttpRequest.AccessToken` attribute supplies the validation result to the policy request, and the `TokenOwner` attribute provides the user identity that is associated with the token.

Policies use this authentication information to affect the processing requests and responses. For example, a policy in the default policy set requires all requests to be made with an active access token.

```
Rule: Deny if HttpRequest.AccessToken.active Equals false

Advice:
  Code: denied-reason
  Applies To: Deny
  Payload: {"status":401, "message": "invalid_token", "detail":"Access token
is expired or otherwise invalid"}
```

The following table identifies the configuration properties that determine the manner in which Sideband API Endpoints handle authentication.

Table 1:

Property	Description
<code>http-auth-evaluation-behavior</code>	Determines whether the Sideband API Endpoint evaluates bearer tokens and, if so, whether the Sideband API Endpoint forwards them to the API server by way of the API gateway.
<code>access-token-validator</code>	<p>Sets the Access Token Validators that the Sideband API Endpoint uses. Because this property contains no value by default, the Sideband API Endpoint can potentially use each Access Token Validator that is configured on the server to evaluate every bearer token.</p> <p>To constrain the set of Access Token Validators that a Sideband API Endpoint uses, set this property to use one or more specific values.</p> <p>This setting is ignored if <code>http-auth-evaluation-behavior</code> is set to <code>do-not-evaluate</code>.</p>

Sideband API policy requests

To authorize an incoming request, the Sideband API performs the following steps:

- Creates a policy request that is based on the incoming request.
- Sends the policy request to the PDP for evaluation.

An understanding of the manner in which the Sideband API formulates policy requests can help you create and troubleshoot policies more effectively.

Policy request attributes

The following table identifies the attributes that are associated with a policy request that the Sideband API generates.

Attribute	Description	Type
<code>action</code>	<p>Identifies the request-processing phase and the HTTP method, such as GET or POST.</p> <p>The value is formatted as <code><phase>-<method></code>. Example values include <code>inbound-GET</code>, <code>inbound-POST</code>, <code>outbound-GET</code>, and <code>outbound-POST</code>.</p>	String

Attribute	Description	Type
<code>service</code>	Identifies the API service. By default, this value is set to the name of the Sideband API Endpoint. To override the default value, set the Sideband API Endpoint's <code>service</code> property. Multiple Sideband API Endpoints can use the same service value.	String
<code>domain</code>	Unused.	String
<code>identityProvider</code>	Name of the Access Token Validator that evaluates the bearer token in an incoming request.	String
<code>attributes</code>	Additional attributes that do not correspond to a specific entity type in the Symphonic trust framework. For more information, see the following table.	Object

The following table identifies the additional attributes that are included in `attributes`.

Attribute	Description	Type
<code>HttpRequest</code>	HTTP request.	Object
<code>TokenOwner</code>	Access token subject as a SCIM resource, as obtained by the access token validator.	Object
<code>Gateway</code>	Additional information about the request.	Object

The following table identifies the fields that the `HttpRequest` attribute can contain.

Attribute	Description	Type
<code>RequestURI</code>	Request URI.	String
<code>Headers</code>	Request and response headers.	Object
<code>ResourcePath</code>	Portion of the request URI path that follows the inbound base path, which the Sideband API Endpoint defines.	String
<code>QueryParameters</code>	Request URI query parameters.	Object
<code>AccessToken</code>	Parsed access token. For more information, see the following table.	Object

Attribute	Description	Type
RequestBody	Request body, if available.	Object
ResponseBody	Response body, if available. This field is provided only for outbound policy requests.	Object
ClientCertificate	Properties of the client certificate, if one was used.	Object

The following table identifies the fields that are associated with the `HttpRequest.AccessToken` attribute, which is populated by the access token validator.

Note: These fields correspond approximately to the fields that are defined by the IETF Token Introspection specification, [RFC 7662](#).

Attribute	Description	Type
client_id	Client ID of the application that was granted the access token.	String
audience	Identifies the recipients for whom the access token is intended. Typically, the authorization server sets this field to identify the resource servers that can accept the token.	Array
user_token	Flag that the access token validator sets to indicate the token was originally issued to a subject. If the flag is <code>false</code> , the token contains no subject and was issued directly to a client.	Boolean
subject	Token subject. This value represents a user identifier that the authorization server sets.	String
token_owner	User identifier that was resolved by the access token validator's token resource lookup method. This value is always a SCIM ID of the form <code><resource type>/<resource ID></code> .	String
username	Subject's user name. This value represents a user identifier that the authorization server sets.	String
issuer	Token issuer. Typically, this value is a URI that identifies the authorization server.	String
issued_at	Date and time at which the access token was issued.	DateTime
expiration	Date and time at which the access token expired.	DateTime

Attribute	Description	Type
<code>not_before</code>	Date and time before which a resource server does not accept an access token.	DateTime
<code>token_type</code>	Token type, as set by the authorization server. Typically, this value is <code>bearer</code> .	String

The following table identifies the fields that the `HttpRequest.ClientCertificate` attribute can contain.

Attribute	Description	Type
<code>algorithm</code>	Name of the certificate signature algorithm, such as <code>SHA256withRSA</code> .	String
<code>algorithmOID</code>	Signature algorithm OID.	String
<code>issuer</code>	Distinguished name (DN) of the certificate issuer.	String
<code>subject</code>	DN of the certificate subject.	String
<code>notAfter</code>	Expiration date and time of the certificate.	DateTime
<code>notBefore</code>	Earliest date on which the certificate is considered valid.	DateTime
<code>valid</code>	Indicates whether the SSL client certificate is valid.	Boolean

The following table identifies the fields that the `Gateway` attribute can contain.

Attribute	Description	Type
<code>_BasePath</code>	Portion of the HTTP request URI that matches the Sideband API Endpoint's <code>base-path</code> value.	String
<code>_TrailingPath</code>	Portion of the HTTP request URI that follows the <code>_BasePath</code> .	String
<i>base path parameters</i>	Parameters in a Sideband API Endpoint's <code>base-path</code> configuration property are included as fields of the <code>Gateway</code> attribute.	String
<i>base path parameters</i>	The <code>Gateway</code> attribute can contain multiple, arbitrary custom attributes that are defined by the <code>policy-request-attribute</code> of the Sideband API Endpoint configuration.	String

Sideband API Endpoint configuration properties

The following table identifies Sideband API Endpoint properties that might force the inclusion of additional attributes with the policy request.

Property	Description
<code>base-path</code>	<p>Defines the URI path prefix that the Sideband API uses to determine whether the Sideband API Endpoint handles a request.</p> <p>The <code>base-path</code> property value can include parameters. If parameters are found and matched, they are included as attributes to policy requests.</p> <p>The following configuration properties can reference parameters that <code>base-path</code> introduces:</p> <ul style="list-style-type: none"> <code>service</code> <code>resource-path</code> <code>policy-request-attribute</code>
<code>service</code>	<p>Identifies the API service to the PDP. A policy can use this value to target requests.</p> <p>The <code>service</code> value appears in the policy request as the <code>service</code> attribute. If undefined, the <code>service</code> value defaults to the name of the Sideband API Endpoint.</p>
<code>resource-path</code>	<p>Identifies the REST resource to the PDP.</p> <p>The resource path value appears in the policy request as the <code>HttpRequest.ResourcePath</code> attribute. If undefined, the <code>resource-path</code> value defaults to the portion of the request that follows the base path, as defined by <code>base-path</code>.</p>
<code>policy-request-attribute</code>	<p>Defines zero or more static, arbitrary key-value pairs. If specified, the pairs are always added as attributes to policy requests.</p> <p>These custom attributes appear in the policy request as fields of the <code>Gateway</code> attribute. For example, if a value of <code>policy-request-attribute</code> is <code>foo=bar</code>, the attribute <code>Gateway.foo</code> is added to the policy request with the value <code>bar</code>.</p>

Path parameters

If parameters are found and matched for the `base-path` property, they are included in policy requests as fields of the `Gateway` policy request attribute. These parameters are available for use by other configuration properties, as identified in the table in [Sideband API Endpoint configuration properties](#) on page 63.

Parameters must be introduced by the `base-path` property. Other configuration properties cannot introduce new parameters.

Path parameters: Basic example

The following table provides a basic example configuration.

API Endpoint property	Example value
base-path	/accounts/{accountId}/transactions
policy-request-attribute	foo=bar

A request URI with the path /accounts/XYZ/transactions/1234 matches the example base-path value.

The following properties are added to the policy request:

- `HttpRequest.ResourcePath` : 1234
- `Gateway.accountId` : XYZ
- `Gateway.foo` : bar

Path parameters: Advanced example

The following table provides an advanced example configuration:

API Endpoint property	Example value
base-path	/health/{tenant}/{resourceType}
service	HealthAPI.{resourceType}
resource-path	{resourceType}/{_TrailingPath}

A request URI with the path /health/OmniCorp/patients/1234 matches the example base-path value.

The following properties are added to the policy request:

- `service` : HealthAPI.patients
- `HttpRequest.ResourcePath` : patients/1234
- `Gateway.tenant` : OmniCorp
- `Gateway.resourceType` : patients

Error templates

REST API clients are often written to expect a custom error format that the API produces. Some clients might fail unexpectedly if they encounter an error response that uses an unexpected format.

When PingDataGovernance Server proxies a REST API, errors that the API returns are forwarded to the client as they are, unless a policy dictates modifications to the response. In the following scenarios, PingDataGovernance Server return an error that the Sideband API generates:

- The policy evaluation results in a `deny` response. This scenario typically results in a 403 error.
- An internal error occurs in the Sideband API. This scenario typically results in a 500 error.

By default, these responses use a simple error format, as the following example shows.

```
{
  "errorMessage": "Access Denied",
  "status": 403
}
```

The following table describes the default error format.

Field	Type	Description
errorMessage	String	Error message.
status	Number	HTTP status code.

Because some REST API clients expect a specific error-response format, PingDataGovernance Server provides *error templates* as a way to respond with custom errors. Written in [Velocity Template Language](#), error templates define the manner in which a Sideband API Endpoint produces error responses.

The following table identifies the context parameters that are provided with error templates.

Parameter	Type	Description
status	Integer	HTTP status.
message	String	Exception message.

Error templates: Example

This topic demonstrates the configuration of a custom error template for a Sideband API Endpoint called *Test API*.

The following fields are associated with the error responses that use this error template:

- code
- message

To create such an error template, perform the following steps:

1. Create a file named `error-template.vtl` with the following contents:

```
#set ($code = "UNEXPECTED_ERROR")
#if($status == 403)
  #set ($code = "ACCESS_FAILED")
#end
{
  "code": "$code",
  "message": "$message"
}
```

2. Add the error template to the configuration.

```
dsconfig create-error-template \
  --template-name "Custom Error Template" \
  --set "velocity-template<error-template.vtl"
```

3. Assign the error template to the Sideband API Endpoint.

```
dsconfig set-sideband-api-endpoint-prop \
  --endpoint-name "Test API" \
  --set "error-template:Custom Error Template"
```

The error template is used whenever the Sideband API generates an error in response to a request.

About the SCIM service

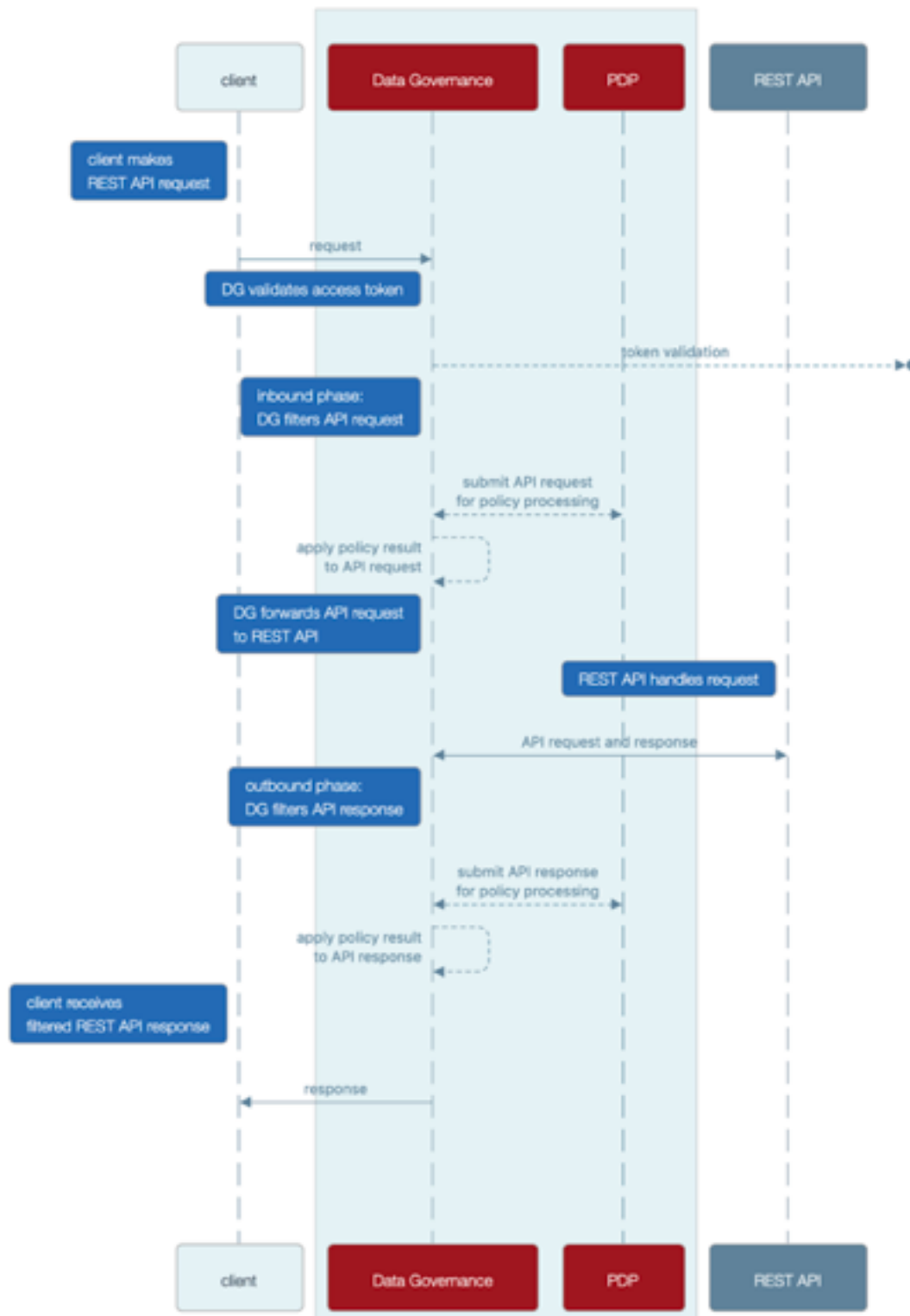
PingDataGovernance Server's built-in SCIM service provides a REST API for data that is stored in one or more external datastores, based on the [SCIM 2.0 standard](#).

Request and response flow

The SCIM REST API provides an HTTP API for data that is contained in a *User Store*. Although User Stores typically consist of a single datastore, such as PingDirectory Server, they can also consist of multiple datastores.

When a SCIM request is received, it is translated into one or more requests to the User Store, and the resulting User Store response is translated into a SCIM response. The SCIM response is authorized by sending a policy request to the PDP. Depending on the policy result, including the advices that are returned in the result, the SCIM response might be filtered or rejected.

Data Governance API Security Gateway sequence diagram



SCIM configuration basics

PingDataGovernance Server's SCIM system consists of the following components:

- SCIM resource types – Define a class of resources, such as users or devices. Every SCIM resource type features at least one *SCIM schema*, which defines the attributes and subattributes that are available to each resource, and at least one *store adapter*, which handles datastore interactions.

The following types of SCIM resource types differ according to the definitions of the SCIM schema:

- Mapping SCIM resource type – Requires an explicitly defined SCIM schema, with explicitly defined mappings of SCIM attributes to store adapter attributes. Use a mapping SCIM resource type to exercise detailed control over the SCIM schema, its attributes, and its mappings.
- Pass-through SCIM resource type – Does not use an explicitly defined SCIM schema. Instead, an implicit schema is generated dynamically, based on the schema that is reported by the store adapter. Use a pass-through SCIM resource type when you need to get started quickly.
- SCIM schemas – Define a collection of SCIM attributes, grouped under an identifier called a *schema URN*. Each SCIM resource type possesses a single *core schema* and can feature *schema extensions*, which act as secondary attribute groupings that the schema URN namespaces. SCIM schemas are defined independently of SCIM resource types, and multiple SCIM resource types can use a single SCIM schema as a core schema or schema extension.

A *SCIM attribute* defines an attribute that is available under a SCIM schema. The configuration for a SCIM attribute defines its data type, regardless of whether it is required, single-valued, or multi-valued. Because it consists of *SCIM subattributes*, a SCIM attribute can be defined as a complex attribute.

- Store adapters – Act as a bridge between PingDataGovernance Server's SCIM system and an external datastore. PingDataGovernance Server provides a built-in LDAP store adapter to support LDAP datastores, including PingDirectory Server and PingDirectoryProxy Server. The LDAP store adapter uses a configurable load-balancing algorithm to spread the load among multiple directory servers. Use the Server SDK to create store adapters for arbitrary datastore types.

Each SCIM resource type features a *primary store adapter*, and can also define multiple *secondary store adapters*. Secondary store adapters allow a single SCIM resource to consist of attributes that are retrieved from multiple datastores.

Store adapter mappings define the manner in which a SCIM resource type maps the attributes in its SCIM schemas to native attributes of the datastore.

About the create-initial-config tool

The `create-initial-config` tool helps to quickly configure PingDataGovernance Server for SCIM. Run this tool after completing setup to configure a SCIM resource type named `Users`, along with a related configuration.

For an example of using `create-initial-config` to create a pass-through SCIM resource type, see [Configure the PingDataGovernance User Store](#) on page 10.

Example: Mapped SCIM resource type for devices

This example demonstrates the addition of a simple mapped SCIM resource type, backed by the standard `device` object class of a PingDirectory Server.

To add data to PingDirectory Server, create a file named `devices.ldif` with the following contents:

```
dn: ou=Devices,dc=example,dc=com
objectClass: top
objectClass: organizationalUnit
ou: Devices

dn: cn=device.0,ou=Devices,dc=example,dc=com
```

```

objectClass: top
objectClass: device
cn: device.0
description: Description for device.0

dn: cn=device.1,ou=Devices,dc=example,dc=com
objectClass: top
objectClass: device
cn: device.1
description: Description for device.1

```

Use the `ldapmodify` tool to load the data file, as follows:

```
PingDirectory/bin/ldapmodify --defaultAdd --filename devices.ldif
```

Start configuring PingDataGovernance Server by adding a store adapter, as follows:

```

dsconfig create-store-adapter \
  --adapter-name DeviceStoreAdapter \
  --type ldap \
  --set enabled:true \
  --set "load-balancing-algorithm:User Store LBA" \
  --set structural-ldap-objectclass:device \
  --set include-base-dn:ou=devices,dc=example,dc=com \
  --set include-operational-attribute:createTimestamp \
  --set include-operational-attribute:modifyTimestamp \
  --set create-dn-pattern:entryUUID=server-
generated,ou=devices,dc=example,dc=com

```

The previous command creates a store adapter that handles LDAP entries found under the base DN `ou=devices,dc=example,dc=com` with the object class `device`. This example uses the User Store load-balancing algorithm that is created when you use the `create-initial-config` tool to set up a users SCIM resource type.

The following command creates a SCIM schema for devices with the schema URN `urn:pingidentity:schemas:Device:1.0`:

```

dsconfig create-scim-schema \
  --schema-name urn:pingidentity:schemas:Device:1.0 \
  --set display-name:Device

```

Under this schema, add the string attributes `name` and `description`, as follows:

```

dsconfig create-scim-attribute \
  --schema-name urn:pingidentity:schemas:Device:1.0 \
  --attribute-name name \
  --set required:true
dsconfig create-scim-attribute \
  --schema-name urn:pingidentity:schemas:Device:1.0 \
  --attribute-name description

```

After you create a store adapter and schema, create the SCIM resource type, as follows:

```

dsconfig create-scim-resource-type \
  --type-name Devices \
  --type mapping \
  --set enabled:true \
  --set endpoint:Devices \
  --set primary-store-adapter:DeviceStoreAdapter \
  --set lookthrough-limit:500 \
  --set core-schema:urn:pingidentity:schemas:Device:1.0

```

Map the two SCIM attributes to the corresponding LDAP attributes. The following commands map the SCIM `name` attribute to the LDAP `cn` attribute, and map the SCIM `description` attribute to the LDAP `description` attribute:

```
dsconfig create-store-adapter-mapping \
  --type-name Devices \
  --mapping-name name \
  --set scim-resource-type-attribute:name \
  --set store-adapter-attribute:cn \
  --set searchable:true

dsconfig create-store-adapter-mapping \
  --type-name Devices \
  --mapping-name description \
  --set scim-resource-type-attribute:description \
  --set store-adapter-attribute:description
```

To confirm that the new resource type has been added, send the following request to the SCIM resource types endpoint:

```
curl -k https://localhost:8443/scim/v2/ResourceTypes/Devices
{"schemas":
[ "urn:ietf:params:scim:schemas:core:2.0:ResourceType" ], "id": "Devices", "name":
"Devices", "endpoint": "Devices", "schema": "urn:pingidentity:schemas:Device:1.0",
"meta": {"resourceType": "ResourceType", "location": "https://localhost:8443/
scim/v2/ResourceTypes/Devices"}}
```

For a more advanced example of a mapped SCIM resource type, refer to the example User schema in `PingDataGovernance/resource/starter-schemas`.

SCIM endpoints

The following table identifies the endpoints that the SCIM 2.0 REST API provides.

Endpoint	Description	Supported HTTP methods
/ServiceProviderConfig	Provides metadata that indicates PingDataGovernance Server's authentication scheme, which is always OAuth 2.0, and its support for features that the SCIM standard considers optional. This endpoint is a metadata endpoint and is not subject to policy processing.	GET
/Schemas	Lists the SCIM schemas that are configured for use on PingDataGovernance Server, and that define the various attributes available to resource types. This endpoint is a metadata endpoint and is not subject to policy processing.	GET

Endpoint	Description	Supported HTTP methods
/Schemas/<schema>	Retrieves a specific SCIM schema, as specified by its ID. This endpoint is a metadata endpoint and is not subject to policy processing.	GET
/ResourceTypes	Lists all of the SCIM resource types that are configured for use on PingDataGovernance Server. Clients can use this information to determine the endpoint, core schema, and extension schemas of any resource types that the server supports. This endpoint is a metadata endpoint and is not subject to policy processing.	GET
/ResourceTypes/<resourceType>	Retrieves a specific SCIM resource type, as specified by its ID. This endpoint is a metadata endpoint and is not subject to policy processing.	GET
/<resourceType>	Creates a new resource (POST), or lists and filters resources (GET).	GET, POST
/<resourceType>/ .search	Lists and filters resources.	POST
/<resourceType>/<resourceId>	Retrieves a single resource (GET), modifies a single resource (PUT, PATCH), or deletes a single resource (DELETE).	GET, PUT, PATCH, DELETE
/Me	Alias for the resource that the subject of the access token identifies. Retrieves the resource (GET), modifies the resource (PUT, PATCH), or deletes the (DELETE).	GET, PUT, PATCH, DELETE

SCIM authentication

All SCIM requests must be authenticated by using OAuth 2.0 bearer token authentication.

Bearer tokens are evaluated by using access token validators. The `HttpRequest.AccessToken` attribute supplies the validation result to the policy request, and the `TokenOwner` attribute provides the user identity that is associated with the token.

Policies use this authentication information to affect the processing of requests and responses.

SCIM policy requests

An understanding of the manner in which the SCIM service formulates policy requests will help you to create and troubleshoot policies more effectively.

For every SCIM request or response, one or more policy requests are sent to the PDP for authorization. Policies can use a policy request's `action` value to determine the processing phase and to act accordingly.

Most SCIM operations are authorized in the following phases:

1. The operation itself is authorized.
2. The outgoing response is authorized with the `retrieve` action.

In most cases, policies that target the `retrieve` action can be reused to specify read-access control rules.

Operation	Actions
POST /scim/v2/<resourceType>	create, retrieve
GET /scim/v2/<resourceType>/<resourceId>	retrieve
PUT /scim/v2/<resourceType>/<resourceId> PATCH /scim/v2/<resourceType>/<resourceId>	modify, retrieve
DELETE /scim/v2/<resourceType>/<resourceId>	delete
GET /scim/v2/<resourceType> POST /scim/v2/<resourceType>/search	search, retrieve -OR- search, search-result For more information about authorizing searches, see About SCIM searches on page 76.

We recommend enabling detailed decision logging and viewing all policy request attributes in action, particularly when learning how to develop SCIM policies. For more information, see [Policy Decision logger](#) on page 109.

Policy request attributes

The following table identifies the attributes associated with a policy request that the SCIM service generates.

Policy request attribute	Description	Type
action	Identifies the SCIM request as one of the following types: <ul style="list-style-type: none"> • create • modify • retrieve • delete • search • search-request 	String
service	Identifies the SCIM service, which is always SCIM2.	String
domain	Unused.	String
identityProvider	Name of the access token validator that evaluates the bearer token used in an incoming request.	String
attributes	Additional attributes that do not correspond to a specific entity type in the PingDataGovernance Trust Framework. For more information, see the following table.	Object

The following table identifies the additional attributes that are included in `attributes`.

Attribute	Description	Type
HttpRequest	HTTP request.	Object
impactedAttributes	Provides the set of attributes that the request modifies.	Collection
TokenOwner	Access token subject as a SCIM resource, as obtained by the access token validator.	Object
SCIM2	Provides additional, SCIM2-specific information about the request.	Object

The following table identifies the fields that the `HttpRequest` attribute contains.

Attribute	Description	Type
RequestURI	The request URI.	String
Headers	Request and response headers.	Object

Attribute	Description	Type
ResourcePath	Uniquely identifies the SCIM resource that is being requested, in the format <code><Resource Type>/<SCIM ID></code> , as the following example shows: <code>Users/0450b8db-f055-35d8-8e2f-0f203a291cd1</code>	String
QueryParameters	Request URI query parameters.	Object
AccessToken	Parsed access token. For more information, see the following table.	Object
RequestBody	The request body, if available. This attribute is available for POST, PUT, and PATCH requests.	Object
ClientCertificate	Properties of the client certificate, if one is used.	Object

The access token validator populates the `HttpRequest.AccessToken` attribute, which contains the fields in the following table. These fields correspond approximately to the fields that the IETF Token Introspection specification ([RFC 7662](#)) defines.

Attribute	Description	Type
client_id	The client ID of the application that was granted the access token.	String
audience	Identifies the recipients for whom the access token is intended. Typically, the authorization server sets this field to indicate the resource servers that might accept the token.	Array
user_token	Flag that the access token validator sets to indicate that the token was issued originally to a subject. If this flag is <code>false</code> , the token does not have a subject and was issued directly to a client.	Boolean
subject	Token subject. This attribute is a user identifier that the authorization server sets.	String
token_owner	User identifier that was resolved by the access token validator's token resource lookup method. This attribute is always a SCIM ID of the form <code><resource type>/<resource ID></code> .	String

Attribute	Description	Type
username	Subject's user name. This attribute is a user identifier that the authorization server sets.	String
issuer	Token issuer. This attribute is usually a URI that identifies the authorization server.	String
issued_at	Date and time at which the access token was issued.	DateTime
expiration	Date and time at which the access token expires.	DateTime
not_before	Date and time before which a resource server does not accept the access token.	DateTime
token_type	The token type, as set by the authorization server. This value is typically set to <code>bearer</code> .	String

The following table identifies the fields that the `HttpRequest.ClientCertificate` attribute contains.

Attribute	Description	Type
algorithm	Name of the certificate signature algorithm, such as <code>SHA256withRSA</code> .	String
algorithmOID	Signature algorithm OID.	String
notBefore	Earliest date on which the certificate is considered valid.	DateTime
notAfter	Expiration date and time of the certificate.	DateTime
issuer	Distinguished name (DN) of the certificate issuer.	String
subject	DN of the certificate subject.	String
valid	Indicates whether the certificate is valid.	Boolean

The following table identifies the fields that the `SCIM2` attribute contains.

Attribute	Description	Type
<code>resource</code>	<p>Complete SCIM resource that the request targets. This attribute is available for GET, PUT, PATCH, and DELETE requests.</p> <p>The <code>resource</code> attribute is also available in the policy requests that are performed for each matching SCIM resource in a search result. For more information, see About SCIM searches on page 76.</p>	Object
<code>modifications</code>	Contains a normalized SCIM 2 PATCH request object that represents all of the changes to apply. This attribute is available for PUT and PATCH requests.	Object

About SCIM searches

A request that potentially causes the return of multiple SCIM resources is considered a *search request*. Perform such requests in one of the following manners:

- Make a GET request to `/scim/v2/<resourceType>`.
- Make a POST request to `/scim/v2/<resourceType>/search`.

To constrain the search results, we recommend that clients supply a search filter through the `filter` parameter. For example, a GET request to `/scim/v2/Users?filter=st+eq+"TX"` returns all SCIM resources of the `Users` resource type in which the `st` attribute possesses a value of "TX". Additionally, the Add Filter policy can be used to add a filter automatically to search requests.

SCIM search policy processing

Policy processing for SCIM searches occurs in the following phases:

1. Policies deny or modify a search request.
2. Policies filter the search result set.

Search request authorization

In the first phase, a policy request is issued for the search itself, using the `search` action. If the policy result is a `deny`, the search is not performed. Otherwise, advices in the policy result are applied to the search filter, giving advices a chance to alter the filter.

Note: Only advice types that are written specifically for the `search` action can be used. For example, the Add Filter advice type can be used to constrain the scope of a search.

The Combine SCIM Search Authorizations advice type can also be used at this point. If this advice is used, search results are authorized by using a special mode, which the next section describes.

Search response authorization

After a search is performed, the resulting `search` response is authorized in one of two ways.

The default authorization mode simplifies policy design but can generate a large number of policy requests. For every SCIM resource that the search returns, a policy request is issued by using the `retrieve` action. If the policy result is a `deny`, the SCIM resource is removed from the search response. Otherwise, advices in the policy result are applied to the SCIM resource, which gives advices a chance to alter the resource.

Because the `retrieve` action is used, policies that are already written for single-resource `GET` operations are reused and applied to the search response.

Optimized search response authorization

If the search request policy result includes the Combine SCIM Search Authorizations advice type, an optimized authorization mode is used instead. This mode reduces the number of overall policy requests but might require a careful policy design. Instead of generating a policy request for each SCIM resource that the search returns, a single policy request is generated for the entire result set. To distinguish the policy requests that this authorization mode generates, the action `search-result` is used.

Write policies that target these policy requests to accept an object that contains a `Resources` array with all matching results. Advices that the policy result returns are applied iteratively to each member of the result set. The input object that is provided to advices also contains a `Resources` array, but it contains only the single result that is currently being considered.

The following code provides an example input object:

```
{
  "Resources": [{
    "name": "Henry Flowers",
    "id": "40424a7d-901e-45ef-a95a-7dd31e4474b0",
    "meta": {
      "location": "https://example.com/scim/v2/Users/40424a7d-901e-45ef-a95a-7dd31e4474b0",
      "resourceType": "Users"
    },
    "schemas": [
      "urn:pingidentity:schemas:store:2.0:UserStoreAdapter"
    ]
  }]
}
```

The optimized search response authorization mode checks policies efficiently, and is typically faster than the default authorization mode. However, the optimized search response authorization mode might be less memory-efficient because the entire result set, as returned by the datastore, is loaded into memory and processed by the PDP.

Lookthrough limit

Because a policy evaluates every SCIM resource in a search result, some searches might exhaust server resources. To avoid this scenario, cap the total number of resources that a search matches.

The configuration for each SCIM resource type contains a `lookthrough-limit` property that defines this limit, with a default value of 500. If a search request exceeds the lookthrough limit, the client receives a 400 response with an error message that resembles the following example:

```
{
  "detail": "The search request matched too many results",
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:Error"
  ],
  "scimType": "tooMany",
  "status": "400"
}
```

To avoid this error, a client must refine its search filter to return fewer matches.

Disable the SCIM REST API

About this task

If you have no need to expose data through the SCIM REST API, disable it by removing the SCIM2 HTTP servlet extension from the HTTPS connection handler, or from any other HTTP connection handler, and restart the handler, as follows:

```
dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --remove http-servlet-extension:SCIM2 \
  --set enabled:false
dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --set enabled:true
```

When the SCIM REST API is disabled, access token validators still use PingDataGovernance Server's SCIM system to look up token owners.

Policy administration

Create policies in a development environment

About this task

Policies are developed in the PingDataGovernance Policy Administration GUI, which is sometimes referred to as the *Policy Administration Point (PAP)*. PingDataGovernance can be configured to evaluate policy in the following modes:

- Embedded
- External

In a development environment, the External mode is used. PingDataGovernance authorizes requests by submitting policy requests to the PAP.

Change the active policy branch

About this task

The PingDataGovernance Policy Administration GUI manages multiple sets of Trust Framework attributes and policies by storing data sets in different branches. In a development environment, the ability to quickly reconfigure PingDataGovernance Server between policy branches is highly beneficial.

To change the active policy branch, perform the following steps:

Steps

1. Define a Policy External Server configuration for each branch.
2. Change the Policy Decision service's `policy-server` property as needed.

Example configuration

The following example involves a policy branch named `Default Policies` and a policy branch named `SCIM Policies`. Create a Policy external server for each branch, as follows:

```
dsconfig create-external-server \
  --server-name "Default Policies" \
  --type policy \
  --set base-url:http://localhost:4200 \
  --set user-id:admin \
  --set decision-node:e51688ff-1dc9-4b6c-bb36-8af64d02e9d1 \
  --set "branch:Default Policies"
dsconfig create-external-server \
  --server-name "SCIM Policies" \
  --type policy \
  --set base-url:http://localhost:4200 \
  --set user-id:admin \
  --set decision-node:e51688ff-1dc9-4b6c-bb36-8af64d02e9d1 \
  --set "branch:SCIM Policies"
```

To use the `Default Policies` branch, configure the Policy Decision Service to use the corresponding Policy external server, as follows:

```
dsconfig set-policy-decision-service-prop \
  --set "policy-server:Default Policies"
```

To use the `SCIM Policies` branch, configure the Policy Decision Service to use the corresponding Policy external server, as follows:

```
dsconfig set-policy-decision-service-prop \
  --set "policy-server:SCIM Policies"
```

Use policies in a production environment

PingDataGovernance Server can be configured to evaluate policy in the following modes:

- Embedded
- External

In staging and production environments, configure PingDataGovernance Server in Embedded mode so that it does not depend on an external server.

To configure the Embedded mode of the Policy Decision Service, perform the following steps:

1. On the main configuration page of the PingDataGovernance Administration Console, click **Policy Decision Service**.
2. From the **PDP Mode** drop-down list, select **Embedded**.
3. After you complete the following section, upload a policy deployment package file.

For more information about modes, see [External PDP mode](#) on page 82 and [Embedded PDP mode](#) on page 88.

Default policies

To use the default policies that are distributed with PingDataGovernance Server, select the deployment package and locate the default policies deployment package that loads directly into the embedded PDP. The policy deployment package is located at `resource/policies/defaultPolicies.SDP`.

The following `dsconfig` command configures the policy service in Embedded mode with the default policies:

```
PingDataGovernance/bin/dsconfig set-policy-decision-service-prop \
  --set pdp-mode:embedded \
  --set "deployment-package<resource/policies/defaultPolicies.SDP"
```

Customized policies

About this task

To install a new set of policies into the PingDataGovernance embedded PDP, based on the changes that you made through the PingDataGovernance Policy Administration GUI, perform the following steps:

Steps

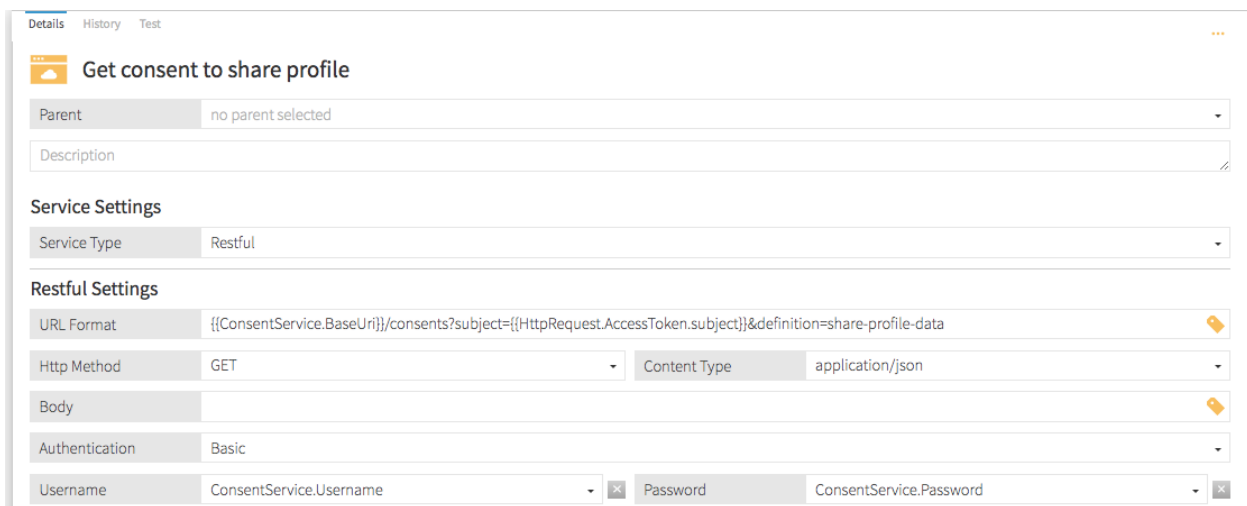
1. In the PingDataGovernance Policy Administration GUI, click **Change Control**.
2. Verify that you are viewing the **Version Control** child tab
3. Select **Commit New Changes** and enter a commit message.
4. From the submenu in the upper-left corner, select the **Deployment Packages** tab.
5. To create a new deployment package, click **+**.
6. From the **Branch** drop-down list, select the policy branch to export.
7. From the **Snapshot** drop-down list, select the option that matches your most recent commit message.
8. To include only particular policies and policy sets, from the **Policy Node** drop-down list, select the branch in the policy tree to export.
9. Click **Create Package**.
10. After the deployment package has been created, click **Export Package** to download a file to your system.
11. To load the new deployment package into the PingDataGovernance embedded PDP, use the PingDataGovernance Administration Console or enter a `dsconfig` command like the following example:

```
PingDataGovernance/bin/dsconfig set-policy-decision-service-prop \
  --set "deployment-package</path/to/policies/customPolicies.SDP"
```

Environment-specific Trust Framework attributes

Within dynamic authorization, policies must be able to retrieve attributes frequently from Policy Information Providers (PIPs) at runtime. The services and datastores from which additional policy information is retrieved range from development and testing environments to preproduction and production environments.

For example, you might use a Trust Framework service to retrieve a user's consent from PingDirectory's Consent API. This service depends on the URL of the Consent API, the user name and password that are used for authentication, and other items that vary between development, preproduction, and production environments.



Details History Test

Get consent to share profile

Parent: no parent selected

Description:

Service Settings

Service Type: Restful

Restful Settings

URL Format: {{ConsentService.BaseUri}}/consents?subject={{HttpRequest.AccessToken.subject}}&definition=share-profile-data

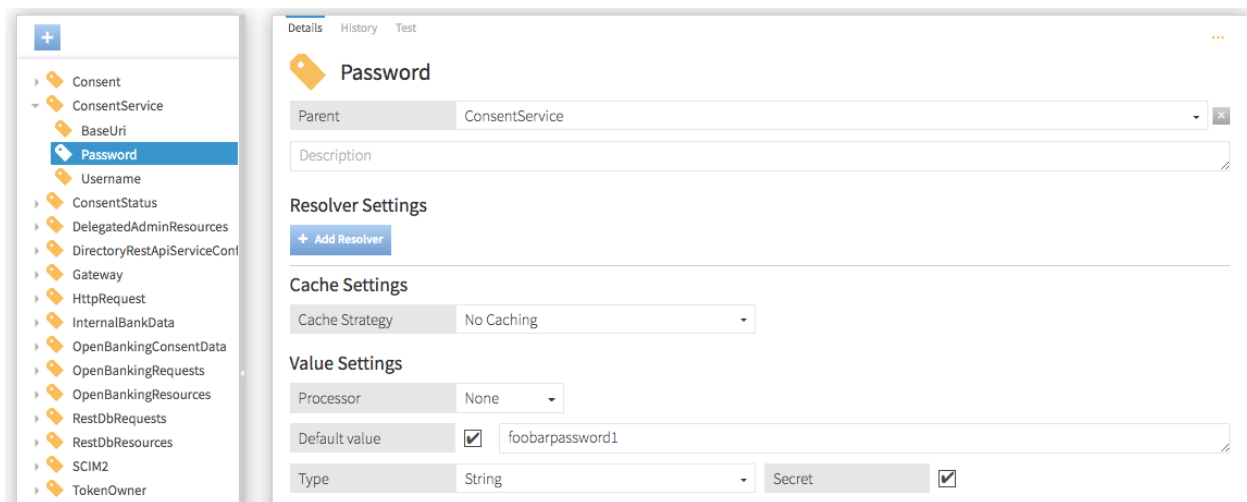
Http Method: GET Content Type: application/json

Body:

Authentication: Basic

Username: ConsentService.Username Password: ConsentService.Password

When you begin creating policies, you can define these values easily in the Trust Framework attributes, as the following image shows.



Details History Test

Password

Parent: ConsentService

Description:

Resolver Settings

+ Add Resolver

Cache Settings

Cache Strategy: No Caching

Value Settings

Processor: None

Default value: ☒ foobarpassword1

Type: String Secret: ☒

Before sharing your policies with others or moving to production, remove these hard-coded values from the Trust Framework. For more information, see [Remove the hard-coded password](#) on page 86.

Store keys and values in PingDataGovernance Server

About this task

Environment-specific attribute keys and values are stored in PingDataGovernance Server's configuration, which can be encrypted. By using this approach, you can configure different values in each of your PingDataGovernance development, preproduction, and production environments.

The hard-coded values will be removed from your Trust Framework attributes at a later time.

Define a policy configuration key

About this task

To define a policy configuration key, perform the following steps:

Steps

1. Use a web browser to access the PingDataGovernance Administration Console at `https://<your-dg-host>:<your-dg-https-port>/console`.
2. Navigate to **Authorization and Policies > Policy Decision Service**.
3. In the **Name** text box, type `ConsentServicePassword`.
4. In the **Policy Configuration Value** text box, type `foobarpassword1`.

Next steps

If you utilize scripted deployment automation, use the command-line tools to configure a different value for each of your environments. For example, in Ansible you might use a Jinja2 template to replace `foobarpassword1` with the appropriate Consent API password for the deployment environment, as follows:

```
PingDataGovernance/bin/dsconfig create-policy-configuration-key
\
--key-name ConsentServicePassword \
--set "policy-configuration-value:foobarpassword1"
```

Repeat this step for the `ConsentServiceBaseUri` and `ConsentServiceUsername` configuration keys.

External PDP mode

When you develop policies, you are using the PDP in your PingDataGovernance Policy Administration GUI server. This mode is referred to as *External PDP mode*. To grant an external PDP access to the passwords that are stored in PingDataGovernance Server, create a service that retrieves the keys and values from your development PingDataGovernance Server's configuration API.

Ultimately, you will create services like the following example, each of which retrieves a specific value from your development PingDataGovernance Server:

The screenshot displays the 'Definition Editor' for the 'ConsentServicePassword' service. The interface includes a top navigation bar with 'Definition Editor' and 'Relationship Search' tabs, and a sub-navigation bar with 'Domains', 'Services', 'Attributes', 'Identity Classes', 'Identity Providers', 'Identity Properties', 'Actions', and 'Conditions'. The main panel shows the 'Details' tab for the service configuration.

ConsentServicePassword

Parent: PolicyConfigurationService

Description:

Service Settings

Service Type: Restful

Restful Settings

URL Format: {{ConfigurationKeyService.BaseUri}}/policy-decision-service/policy-configuration-keys/ConsentServicePassword

Http Method: GET

Content Type: application/json

Body:

Authentication: Basic

Username: ConfigurationKeyService.Username

Password: ConfigurationKeyService.Password

Headers

+ Header

Value Settings

Processor: JSON Path

Value: \$.policyConfigurationValue

Type: String

Secret: ☒

To complete this task, configure the credentials that grant access to your development PingDataGovernance Server's API.

Note: Configure credentials to grant access to your development PingDataGovernance Servers only when operating in External PDP mode. Such credentials are unnecessary when operating in Embedded PDP mode, which is used in production environments.

Never store credentials in the Trust Framework attributes. Instead, save them to the server on which you installed the PingDataGovernance Policy Administration GUI.

Store PingDataGovernance credentials as environment variables

About this task

To create environment variables, run the following commands in a terminal window:

```
export ConfigurationKeyServiceBaseUri="https://<your-dg-host>:<your-dg-httpsport>/config/v2"
export ConfigurationKeyServiceUsername="cn=<your-dg-username>"
export ConfigurationKeyServicePassword="<your-dg-password>"
```

Add PingDataGovernance environment variables to the configuration file

About this task

To add an attribute value to the configuration file, perform the following steps:

Steps

1. In a text editor, open the configuration file `PingDataGovernance-PAP/config/configuration.yml`.
2. Locate the `core:` section.

3. Add the user name and password to PingDataGovernance Server, as follows:

```
ConfigurationKeyServiceBaseUri: ${ConfigurationKeyServiceBaseUri}
ConfigurationKeyServiceUsername: ${ConfigurationKeyServiceUsername}
ConfigurationKeyServicePassword: ${ConfigurationKeyServicePassword}
```

4. Stop the PingDataGovernance Policy Administration GUI server.
5. Restart the PingDataGovernance Policy Administration GUI server.

Results

core:

```
Database.Type: H2
Database.H2.Mode: file
datanucleus.generateSchema.database.mode: create
ConfigurationKeyServiceBaseUri: ${ConfigurationKeyServiceBaseUri}
ConfigurationKeyServiceUsername: ${ConfigurationKeyServiceUsername}
ConfigurationKeyServicePassword: ${ConfigurationKeyServicePassword}
```

The `${ }` points to the server environment variables. It is added to `configuration.yml` so that the PingDataGovernance Policy Administration GUI can use environment variables as attributes.

In the following section, we will create those attributes within the PingDataGovernance Policy Administration GUI.

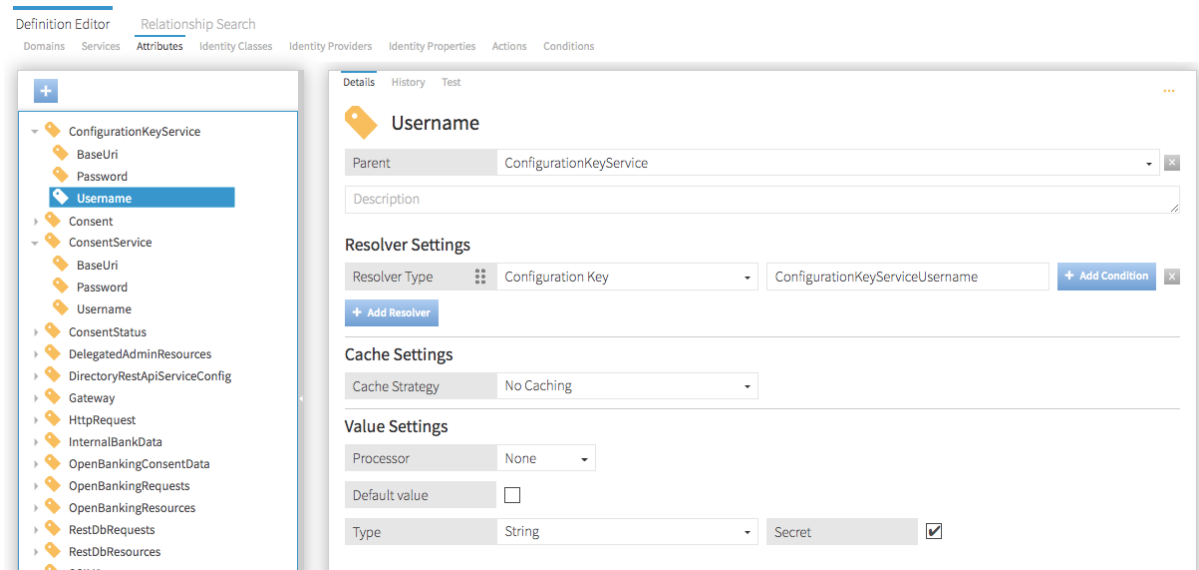
Define a new attribute

About this task

To define a new attribute, perform the following steps:

Steps

1. From the PingDataGovernance Policy Administration GUI, navigate to **Trust Framework**.
2. Click **Attributes**.
3. Click **+Add new attribute**.
4. In the **Name** text box, type `Username`.
5. In the **Resolver Settings** section, perform the following steps:
 - a) From the **Resolver Type** drop-down list, select `Configuration Key`.
 - b) In the corresponding text box, type `ConfigurationKeyServiceUsername`.
6. Click **Save Changes**.



Next steps

Repeat this task for `ConfigurationKeyServiceBaseUri` and `ConfigurationKeyServicePassword`.

Retrieve the `ConsentServicePassword` value

About this task

Create a new Trust Framework service to retrieve the `ConsentServicePassword` value from your development PingDataGovernance Server.

Steps

1. From the PingDataGovernance Policy Administration GUI, navigate to **Trust Framework**.
2. Click **Services**.
3. Click **+Add new service**.
4. In the **Name** text box, type `ConsentServicePassword`.
5. From the **Service Type** drop-down list, select `Restful`.
6. In the **Restful Settings** section, perform the following steps:
 - a) In the **URL Format** text box, type `{{ConfigurationKeyServiceBaseUri}}/policy-decision-service/policy-configuration-keys/ConsentServicePassword`.
 - b) From the **Authentication** drop-down list, select `Basic`.
 - c) From the **Username** drop-down list, select the attribute that you created, `<Attribute>.Username`.
 - d) From the **Password** drop-down list, select the attribute that you created, `<Attribute>.Password`.
7. In the **Value Settings** section, perform the following steps:
 - a) From the **Processor** drop-down list, select `JSONPath`.
 - b) In the corresponding text box, type `$.policyConfigurationValue`.
8. Click **Save Changes**.

Results

Definition Editor Relationship Search

Domains **Services** Attributes Identity Classes Identity Providers Identity Properties Actions Conditions

Details History Test

ConsentServicePassword

Parent PolicyConfigurationService

Description

Service Settings

Service Type Restful

Restful Settings

URL Format {{ConfigurationKeyService.BaseUri}}/policy-decision-service/policy-configuration-keys/ConsentServicePassword

Http Method GET Content Type application/json

Body

Authentication Basic

Username ConfigurationKeyService.Username Password ConfigurationKeyService.Password

Headers

+ Header

Value Settings

Processor JSON Path \$.policyConfigurationValue

Type String Secret ☒

Next steps

Repeat this task for `ConsentServiceBaseUri` and `ConsentServiceUsername`.

Remove the hard-coded password

About this task

Remove the hard-coded password from your Trust Framework attributes and add a resolver to use the new Trust Framework service. To reduce REST API calls to your development PingDataGovernance Server, ensure that you add attribute caching.

Steps

1. From the PingDataGovernance Policy Administration GUI, navigate to **Trust Framework**.
2. Click **Attributes**.
3. Expand **ConsentService**.
4. Click **Password**.
5. In the **Resolver Settings** section, perform the following steps:
 - a) From the **Resolver Type** drop-down list, select **Service**.
 - b) In the corresponding text box, type `PolicyConfigurationService.ConsentServicePassword`.
6. Click **Save Changes**.

Results

Definition Editor Relationship Search

Domains Services **Attributes** Identity Classes Identity Providers Identity Properties Actions Conditions

Password

Parent: ConsentService

Description

Resolver Settings

Resolver Type: Service PolicyConfigurationService.ConsentServicePasswor [+ Add Condition](#)

[+ Add Resolver](#)

Cache Settings

Cache Strategy: Time Limited Caching

Time to Live (s): 3600

Value Settings

Processor: None

Default value: ☐

Type: String Secret ☒

Next steps

Repeat this task for ConsentServiceUsername and ConsentServiceBaseUri.

Test your changes

About this task

Definition Editor Relationship Search

Domains Services **Attributes** Identity Classes Identity Providers Identity Properties Actions Conditions

Password

Testing Scenario

Request

Domain: Select to add Domain to the testing scenario

Service: Select to add Service to the testing scenario

Identity Provider: Select to add Identity Provider to the testing scenario

Action: Select to add Action to the testing scenario

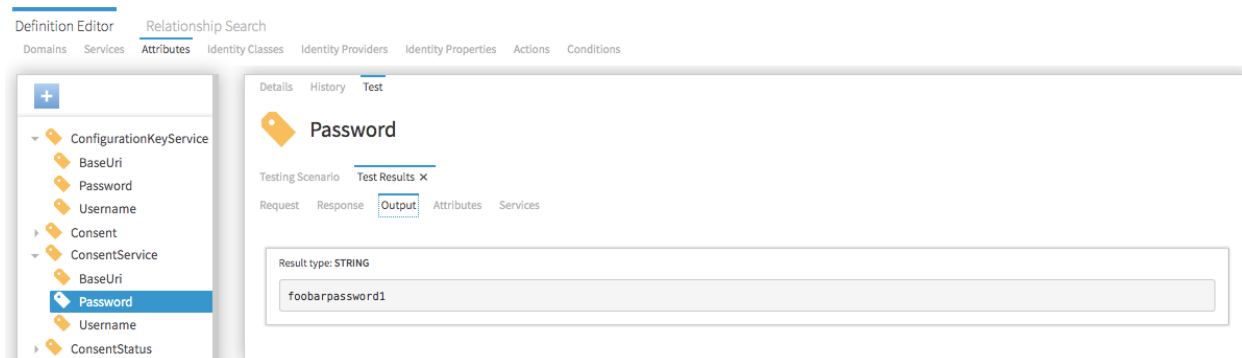
Attributes: Select an attribute to add it to the testing scenario

Overrides

Attributes: Select an attribute to add it to the testing scenario

Services: Select a service to add it to the testing scenario

[Import](#) [Execute](#)



Steps

Embedded PDP mode

When you perform regression testing or work in a production environment, you are using the PDP that is embedded within PingDataGovernance Server. This mode is referred to as *Embedded PDP mode*.

PingDataGovernance Server automatically passes all configured policy configuration keys and values to the embedded PDP.

Access a policy configuration key

About this task

To access the keys and values in Trust Framework attributes, add a corresponding resolver for using the Configuration Key type, and specify the matching key name. Make certain to drag the Configuration Key resolver to the top of the preference order.

Steps

1. From the PingDataGovernance Policy Administration GUI, navigate to **Trust Framework**.
2. Click **Attributes**.
3. Click **+Add new attribute**.
4. In the **Name** text box, type `Password`.
5. In the **Resolver Settings** section, perform the following steps:
 - a) From the **Resolver Type** drop-down list, select `Configuration Key`.
 - b) In the corresponding text box, type `ConsentServicePassword`.
 - c) Click **+Add Resolver**.
 - d) From the **Resolver Type** drop-down list, select `Service`.
 - e) In the corresponding text box, type `PolicyConfigurationService.ConsentServicePassword`.
6. Click **Save Changes**.

Advice property	Description
Payload	Set of parameters governing the actions that the advice performs when it is applied. The appropriate payload value depends on the advice type.

PingDataGovernance supports the following advice types:

- Add Filter
- Allow Attributes
- Combine SCIM Search Authorizations
- Denied Reason
- Exclude Attributes
- Filter Response
- Include Attributes
- Prohibit Attributes

The following sections describe these advice types in more detail. To develop custom advice types, use the Server SDK.

Note: Many advice types let you use the [JSONPath](#) expression language to specify JSON field paths. To experiment with JSONPath, use the [Jayway JSONPath Evaluator](#) tool.

Add Filter

Advice ID: `add-filter`

Description: Adds administrator-required filters to SCIM search queries.

Applicable to: SCIM

The Add Filter advice places restrictions on the resources that are returned to an application that can otherwise use SCIM search requests. The filters that the advice specifies are **AND**ed with any filter that the SCIM request includes.

The payload for this advice is a string that represents a valid SCIM filter, which can contain multiple clauses that are separated by **AND** or **OR**. If multiple instances of Add Filter advice are returned from policy, they are **AND**ed together to form a single filter that is passed with the SCIM request. If the original SCIM request body included a filter, it is **AND**ed with the policy-generated filter to form the final filter value.

Allow Attributes

Advice ID: `allow-attributes`

Description: Specifies the attributes that a JSON request body can create or modify for POST, PUT, or PATCH.

Applicable to: All, although only SCIM is supported when the HTTP method is PATCH.

The payload for this advice is a JSON array of strings. Each string is interpreted as the name of a resource attribute that the client can modify, create, or delete. If the client request contains changes for an attribute that the advice does not name, the request is denied with a 403 Forbidden response. If multiple instances of Allow Attributes advice are returned from policy, the union of all named attributes is allowed. The optional wildcard string "*" indicates that the request can modify all attributes, and can override the other paths that are present in the policy result.

Combine SCIM Search Authorizations

Advice ID: `combine-scim-search-authorizations`

Description: Optimizes policy processing for SCIM search responses.

Applicable to: SCIM

By default, SCIM search responses are authorized by generating multiple policy decision requests with the `retrieve` action, one for each member of the result set. The default mode enables policy reuse but might result in greater overall policy processing time.

When this advice type is used, the current SCIM search result set is processed by using an alternative authorization mode in which all search results are authorized by a single policy request that uses the `search-results` action. The policy request includes an object with a single `Resources` field, which is an array that consists of each matching SCIM resource. Advices that are returned in the policy result are applied iteratively against each matching SCIM resource, allowing for the modification or removal of individual search results.

This advice type does not use a payload.

For more information about SCIM search handling, see [About SCIM searches](#) on page 76.

Denied Reason

Advice ID: `denied-reason`

Description: Allows a policy writer to provide an error message that contains the reason for denying a request.

Applicable to: All.

The payload for Denied Reason advice is a JSON object string with the following fields:

- `status` – Contains the HTTP status code that is returned to the client. If this field is absent, the default status is 403 Forbidden.
- `message` – Contains a short error message that is returned to the client
- `detail` (optional) – Contains additional, more detailed error information.

The following example might be returned for a request made with insufficient scope:

```
{ "status": 403, "message": "insufficient_scope", "detail": "Requested operation
not allowed by the granted OAuth scopes." }
```

Exclude Attributes

Advice ID: `exclude-attributes`

Description: Specifies the attributes that are excluded from a JSON response.

Applicable to: All

The payload for this advice is a JSON array of strings. Each string is interpreted as a JSONPath into the response body of the request that is being authorized. The portions of the response that each JSONPath selects are removed before the response is returned to the client. Each JSONPath can point to multiple attributes in the object, all of which are removed.

The following example instructs PingDataGovernance Server to remove the attributes `secret` and `data.private`:

```
[ "secret", "data.private" ]
```

For more information about the processing of SCIM searches, see [Filter Response](#) on page 91.

Filter Response

Advice ID: `filter-response`

Description: Directs PingDataGovernance Server to invoke policy iteratively over each item of a JSON array that is contained within an API response.

Applicable to: Gateway

Filter Response advice allows policies, when presented with a request to permit or deny a multi-valued response body, to require that a separate policy request be made to determine whether the client can access each individual resource that a JSON array returns.

The following table identifies the fields of the JSON object that represents the payload for this advice.

Field	Required	Description
Path	Yes	JSONPath to an array within the API's response body. The advice implementation iterates over the nodes in this array and makes a policy request for each node.
Action	No	Value to pass as the <code>action</code> parameter on subsequent policy requests. If no value is specified, the action from the parent policy request is used.
Service	No	Value to pass as the <code>service</code> parameter on subsequent policy requests. If no value is specified, the service value from the parent policy request is used.
ResourceType	No	Type of object contained by each JSON node in the array, selected by the <code>Path</code> field. On each subsequent policy request, the contents of a single array element are passed to the policy decision point as an attribute with the name that this field specifies. If no value is specified, the resource type of the parent policy request is used.

On each policy request, if policy returns a `deny` decision, the relevant array node is removed from the response. If the policy request returns a `permit` decision with additional advice, the advice is fulfilled within the context of the request. For example, this advice allows policy to decide whether to exclude or obfuscate particular attributes for each array item.

For a response object that contains complex data, including arrays of arrays, this advice type can descend through the JSON content of the response.

Note: Performance ramifications might arise as the total number of policy requests increases.

Include Attributes

Advice ID: `include-attributes`

Description: Limits the attributes that a JSON response can return.

Applicable to: All

The payload for this advice is a JSON array of strings. Each string is interpreted as a JSONPath into the response body of the request that is being authorized. The response includes only the portions that one of the JSONPaths selects. When a single JSONPath represents multiple attributes, all of them are included. If

multiple instances of Include Attributes advice are returned from a policy, the response includes the union of all selected attributes.

For more information about the processing of SCIM searches, see [Filter Response](#) on page 91.

Prohibit Attributes

Advice ID: `prohibit-attributes`

Description: Specifies the attributes that a JSON request body cannot create or modify with POST, PUT, or PATCH methods.

Applicable to: All, although only SCIM is supported when the HTTP method is PATCH.

The payload for this advice is a JSON array of strings. Each string is interpreted as the name of a resource attribute that the client is not permitted to modify, create, or delete. If the client request contains changes for an attribute that the advice specifies, the request is denied with a 403 Forbidden response.

Access token validators

Access token validators verify the tokens that client applications submit when they request access to protected resources. Specifically, they translate an access token into a data structure that constitutes part of the input for policy processing.

To authenticate to PingDataGovernance Server's HTTP services, clients use [OAuth 2 bearer token authentication](#) to present an access token in the HTTP Authorization Request header. To process the incoming access tokens, PingDataGovernance Server uses access token validators, which determine whether to accept an access token and translate it into a set of properties, called *claims*.

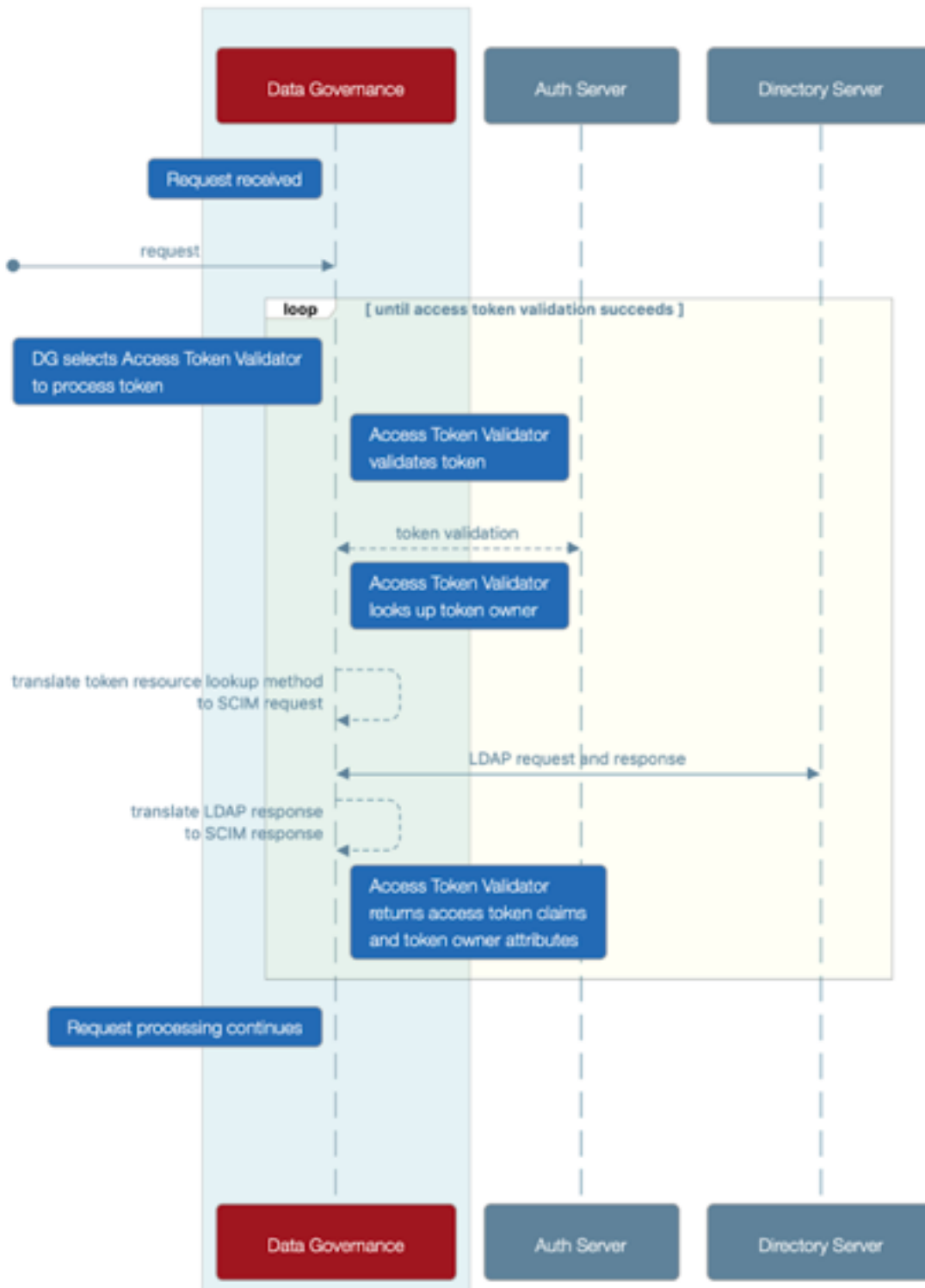
Most access tokens identify a user, also called the *token owner*, as its subject, and access token validators retrieve the token owner's attributes from the User Store. All the data that an access token validator produces is sent to the PDP so that policies can determine whether to authorize the request.

About access token validator processing

Any number of access token validators can be configured for PingDataGovernance Server. Each access token validator possesses an *evaluation order index*, an integer that determines its processing priority. Lower evaluation order index values take precedence over higher values.

The following image shows the validation process.

Data Governance Access Token Validator Processing



1. If an incoming HTTP request contains an access token, the token is sent to the access token validator with the lowest evaluation order index.
2. The access token validator validates the access token.

Validation logic varies by access token validator type, but the validator generally verifies the following information:

- A trusted source issued the token
- The token is not expired

If the token is valid, its `active` flag is set to `true`. The flag and other access token claims are added to the `HttpRequest.AccessToken` attribute of the policy request.

3. If the access token contains a subject, the access token validator sets the `user_token` flag to `true`, and uses a token resource lookup method to fetch the token owner through SCIM.

A *token resource lookup* defines a SCIM filter that locates the token owner. If the lookup succeeds, the resulting SCIM object is added to the policy request as the `TokenOwner` attribute.

4. If the access token validator is unable to validate the access token, the token is passed to the access token validator with the next lowest evaluation order index, and the previous two steps are repeated.
5. HTTP request processing continues, and the policy request is sent to the PDP.
6. Policies inspect the `HttpRequest.AccessToken` and `TokenOwner` attributes to make access control decisions.

Access tokens issued using the OAuth 2 client credentials grant type are issued directly to a client and do not contain a subject. An access token validator always sets the `HttpRequest.AccessToken.user_token` flag to `false` for such tokens, which are called *application tokens*, in contrast to tokens with subjects, which are called *user tokens*. Because authorization policies often grant a broad level of access for application tokens, we recommend that such policies always check the `HttpRequest.AccessToken.user_token` flag.

Access token validators determine whether PingDataGovernance Server accepts an access token and uses it to provide key information for access-control decisions, but they are neither the sole nor the primary means of managing access. The responsibility for request authorization falls upon the PDP and its policies. This approach allows an organization to tailor access-control logic to its specific needs.

Access token validator types

PingDataGovernance Server provides the following types of access token validators:

- PingFederate access token validator
- JSON Web Token (JWT) access token validator
- Mock access token validator
- Third-party access token validator

PingFederate access token validator

To verify the access tokens that a PingFederate authorization server issues, the PingFederate access token validator uses HTTP to submit the tokens to PingFederate Server's token introspection endpoint. This step allows the authorization server to determine whether a token is valid.

Because this step requires an outgoing HTTP request to the authorization server, the PingFederate access token validator might perform slower than other access token validator types. Regardless, the validation result is guaranteed to be current, which is an important consideration if the authorization server permits the revocation of access tokens.

Before attempting to use a PingFederate access token validator, create a client that represents the access token validator in the PingFederate configuration. This client must use the Access Token Validation grant type.

Example configuration

In PingFederate, create a client with the following properties:

- Client ID: PingDataGovernance
- Client authentication: Client Secret
- Allowed grant types: Access Token Validation

Take note of the client secret that is generated for the client, and use PingDataGovernance Server's `dsconfig` command to create an access token validator, as follows:

```
# Change the host name and port below, as needed
dsconfig create-external-server \
  --server-name "PingFederate External Server" \
  --type http \
  --set base-url:https://example.com:9031
# Create the Access Token Validator
dsconfig create-access-token-validator \
  --validator-name "PingFederate Access Token Validator" \
  --type ping-federate \
  --set enabled:true \
  --set "authorization-server:PingFederate External Server" \
  --set client-id:PingDataGovernance \
  --set "client-secret:<client secret>" \
  --set evaluation-order-index:2000
# Match the token's subject (sub) claim to the uid attribute
# of a SCIM resource
dsconfig create-token-resource-lookup-method \
  --validator-name "PingFederate Access Token Validator" \
  --method-name "User by uid" \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:1000
```

JWT access token validator

The JWT access token validator verifies access tokens that are encoded in JSON Web Token format, which can be signed (JWS) or signed and encrypted (JWE). The JWT access token validator inspects the JWT token without presenting it to an authorization server for validation.

To ensure that a trusted source issued a particular token, the token's signature is validated by using the public keys of the authorization server in one of the following manners:

- Store the keys as trusted certificates in PingDataGovernance Server's configuration.
- Retrieve the keys by way of HTTP from the authorization server's JSON Web Key Set (JWKS) endpoint when the JWT access token validator is initialized. This method ensures that the JWT access token validator uses updated copies of the authorization server's public keys.

Because the JWT access token validator is not required to make a token introspection request for every access token that it processes, it performs better than the PingFederate access token validator. The access token is self-validated, however, so the JWT access token validator cannot determine whether the token has been revoked.

Supported JWS/JWE features

For signed tokens, the JWT access token validator supports the following JWT web algorithm (JWA) types :

- RS256
- RS384
- RS512

For encrypted tokens, the JWT access token validator supports the RSA-OAEP key-encryption algorithm and the following content-encryption algorithms:

- A128CBC-HS256
- A192CBC-HS384
- A256CBC-HS512

Example configuration

In the following example, a JWT access token validator is configured to retrieve public keys from a PingFederate authorization server's JWKS endpoint:

```
# Change the host name and port below, as needed
dsconfig create-external-server \
  --server-name "PingFederate External Server" \
  --type http \
  --set base-url:https://example.com:9031
# Create the Access Token Validator
dsconfig create-access-token-validator \
  --validator-name "JWT Access Token Validator" \
  --type jwt \
  --set enabled:true \
  --set evaluation-order-index:1000 \
  --set "authorization-server:PingFederate External Server" \
  --set jwks-endpoint-path:/ext/oauth/jwks
# Match the token's subject (sub) claim to the uid attribute
# of a SCIM resource
dsconfig create-token-resource-lookup-method \
  --validator-name "JWT Access Token Validator" \
  --method-name "User by uid" \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:1000
```

Mock access token validator

A *mock access token validator* is a special access token validator type that is used for development or testing purposes. A mock access token validator accepts arbitrary tokens without validating whether a trusted source issued them. This approach allows a developer or tester to make bearer token-authenticated requests without first setting up an authorization server.

Mock access tokens are formatted as plain-text JSON objects using standard JWT claims. Always provide the boolean `active` claim. If this value is `true`, the token is accepted. If this value is `false`, the token is rejected. If the `sub` claim is provided, a token owner lookup populates the `TokenOwner` policy request attribute, as with the other access token validator types.

The following example cURL command provides a mock access token in an HTTP request:

```
curl -k -X GET https://localhost:8443/scim/v2/Me -H 'Authorization:
  Bearer {"active": true, "sub":"user.3", "scope":"email profile",
  "client":"client1"}'
```

Important: Never use mock access token validators in a production environment because they do not verify whether a trusted source issued an access token.

Example configuration

The configuration for a mock access token validator resembles the configuration for a JWT access token validator. However, the JWS signatures require no configuration because mock tokens are not authenticated.

```
# Create the Access Token Validator
dsconfig create-access-token-validator \
  --validator-name "Mock Access Token Validator" \
  --type mock --set enabled:true \
  --set evaluation-order-index:9999
# Match the token's subject (sub) claim to the uid attribute
# of a SCIM resource
dsconfig create-token-resource-lookup-method \
  --validator-name "Mock Access Token Validator" \
  --method-name "User by uid" \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:1000
```

Third-party access token validator

To create custom access token validators, use the Server SDK.

Server configuration

This section covers basic server configuration. For a detailed look at configuration, refer to the *Ping Identity PingDataGovernance Server Configuration Reference*, which is located in the server's `docs` directory.

PingDataGovernance Server is built upon the same foundation as PingDirectory Server. Both servers use a common configuration system, and their configurations use the same tools and APIs.

The configuration system is fundamentally LDAP-based, and configuration entries are stored in a special LDAP backend called `cn=config`. The structure is a tree structure, and configuration entries are organized in a shallow hierarchy under `cn=config`.

Administration accounts

Administration accounts called *Root DNs* are stored in a branch of the configuration backend, `cn=Root DNs,cn=config`. When setup is run, the process creates a superuser account that is typically named `cn=Directory Manager`. Although PingDataGovernance Server is not an LDAP directory server, it follows this convention by default. As a result, its superuser account is also typically named `cn=Directory Manager`.

To create additional administration accounts, use `dsconfig` or the PingDataGovernance Administration Console to add Root DN users.

About the dsconfig tool

Use the `dsconfig` tool whenever you administer the server from a shell. When run without arguments, `dsconfig` enters an interactive mode that permits the browsing and updating of the configuration from a menu-based interface. Use this interface to list, update, create, and delete configuration objects.

When viewing any configuration object in `dsconfig`, use the `d` command to display the command line that is necessary to recreate a configuration object. A command line in this form can be used directly from a shell or placed in a `dsconfig` batch file, along with other commands.

Batch files are a powerful feature that enable scripted deployments. By convention, these scripts use a file extension of `DSCONFIG`. Batch files support comments by using the `#` character, and they support line continuation by using the `\` (backslash) character.

For example, the following `dsconfig` script configures PingDataGovernance Server's policy service:

```
# Define an external PingDataGovernance PAP
dsconfig create-external-server \
  --server-name "PingDataGovernance PAP" \
  --type policy \
  --set base-url:http://localhost:4200 \
  --set user-id:admin \
  --set decision-node:e51688ff-1dc9-4b6c-bb36-8af64d02e9d1 \
  --set "branch:Default Policies"
# Configure the policy service
dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:PingDataGovernance PAP" \
  --set "decision-response-view:request" \
  --set "decision-response-view:decision-tree"
```

To load a `dsconfig` batch file, run `dsconfig` with the `--batch-file` argument, as follows:

```
$ PingDataGovernance/bin/dsconfig -n --batch-file example.dsconfig

Batch file 'example.dsconfig' contains 2 commands.

Pre-validating with the local server ..... Done

Executing: create-external-server -n --server-name "PingDataGovernance PAP"
--type policy --set base-url:http://localhost:4200 --set
decision-node:e51688ff-1dc9-4b6c-bb36-8af64d02e9d1 --set "branch:Default
Policies"

Arguments from tool properties file: --useSSL --hostname localhost --port
8636 --bindDN cn=root --bindPassword ***** --trustAll

The Policy External Server was created successfully.

Executing: set-policy-decision-service-prop -n --set pdp-mode:external --set
"policy-server:PingDataGovernance PAP" --set
decision-response-view:request --set decision-response-view:decision-tree

The Policy Decision Service was modified successfully.
```

PingDataGovernance Administration Console

The PingDataGovernance Administration Console is a web-based application that provides a graphical configuration and administration interface. It is available by default from the `/console` path.

About the configuration audit log

All successful configuration changes are recorded to the file `logs/config-audit.log`, which records the configuration commands that represent these changes as well as the configuration commands that undo the changes.

```
$ tail -n 8 PingDataGovernance/logs/config-audit.log
```

```
# [23/Feb/2019:23:16:24.667 -0600] conn=4 op=12 dn='cn=Directory
Manager,cn=Root DNs,cn=config' authtype=[Simple] from=127.0.0.1
to=127.0.0.1
# Undo command: dsconfig delete-external-server --server-name
"PingDataGovernance PAP"
dsconfig create-external-server --server-name "PingDataGovernance PAP"
--type policy --set base-url:http://localhost:4200 --set decision-
node:e51688ff-1dc9-4b6c-bb36-8af64d02e9d1 --set "branch:Default Policies"

# [23/Feb/2019:23:16:24.946 -0600] conn=5 op=22 dn='cn=Directory
Manager,cn=Root DNs,cn=config' authtype=[Simple] from=127.0.0.1
to=127.0.0.1
# This change was made to mirrored configuration data, which is
automatically kept in sync across all servers.
# Undo command: dsconfig set-policy-decision-service-prop --set "policy-
server:PingDataGovernance (Gateway Policy Example)"
dsconfig set-policy-decision-service-prop --set "policy-
server:PingDataGovernance PAP"
```

About the config-diff tool

The `config-diff` tool compares server configurations and produces a `dsconfig` batch file that lists the differences.

When run without arguments, the `config-diff` tool produces a list of changes to the configuration, as compared to the server's baseline or out-of-the-box configuration. Because this list captures the customizations of your server configuration, it is useful when you transition from a development environment to a staging or production environment.

```
$ PingDataGovernance/bin/config-diff
# No comparison arguments provided, so using "--sourceLocal --sourceTag
postSetup --targetLocal" to compare the local configuration with the post-
setup configuration.
# Run "config-diff --help" to get a full list of options and example usages.

# Configuration changes to bring source (config-postSetup.gz) to target
(config.ldif)
# Comparison options:
#   Ignore differences on shared host
#   Ignore differences by instance
#   Ignore differences in configuration that is part of the topology
registry

dsconfig create-external-server --server-name "DS API Server" --type api
--set base-url:https://localhost:1443 --set hostname-verification-
method:allow-all --set "trust-manager-provider:Blind Trust" --set user-
name:cn=root --set "password:AADaK6dtmjJQ7W+urtx9RGhSvKX9qCS8q5Q="

dsconfig create-external-server --server-name "FHIR Sandbox" --type api
--set base-url:https://fhir-open.sandboxcerner.com
...
```

Certificates

Depending on the circumstances, PingDirectory Server uses one of the following certificates:

- Inter-server certificate – Used for internal purposes, like the following examples:
 - Replication authentication

- Inter-server authentication in the topology registry
- Reversible password encryption
- Encrypted backups and LDIF exports
- Server certificate – Presented by the server when a client uses a protocol like LDAPS or HTTPS to initiate a secure connection. A client must trust the server's certificate to obtain a secure connection to it.

The following sections describe these certificates in more detail.

Inter-server certificate

Generated during installation, the inter-server certificate is stored under the alias `ads-certificate` in a file named `ads-truststore`, which resides in the server's `/config` directory. This certificate contains the key pair for the local server as well as for the certificates of all trusted servers, and has a lifetime of 20 years before expiring.

The local server's public key is signed by its own private key, making it a *self-signed certificate*. The alias is hard-coded to `ads-certificate`, and the keystore file is hard-coded to `ads-truststore`. This behavior cannot be modified during setup.



Warning:

- Although some customers feel uncomfortable with the self-signed nature of the inter-server certificate, we recommend that you do not replace it with a CA-signed certificate for the following reasons:
 - If the inter-server certificate is replaced incorrectly, serious problems can occur during topology authentication.
 - The inter-server certificate is used for internal purposes only.
- If the server's access logs contain authentication (bind) errors, the inter-server certificate is most likely configured inappropriately. In the topology registry, this certificate is persisted in the `inter-server-certificate` property of a server instance.

Replace the inter-server certificate

About this task

Because the inter-server certificate is also stored in the topology registry, it can be replaced on one server and mirrored to all other servers in the topology. Changes are mirrored automatically to the other servers in the topology.

Important: Before attempting to replace the inter-server certificate, ensure that all servers in the topology are updated to version 7.0 or later.

The inter-server certificate is stored in human-readable, PEM-encoded format and can be updated by using the `dsconfig` tool. While the certificate is being replaced, existing authenticated connections continue to work. If the server is restarted, or if a topology change requires a reset of peer connections, the server continues authenticating with its peers, all of whom trust the new certificate.

To replace the inter-server certificate with no downtime, complete the following tasks:

Steps

1. Prepare a new keystore with the replacement key pair.
2. Import the earlier trusted certificates into the new keystore.
3. Update the server configuration to use the new certificate by adding it to the server's list of certificates in the topology registry.
After this step is performed, other servers will trust the certificate.
4. Replace the server's `ads-truststore` file with the new one.

5. Retire the previous certificate by removing it from the topology registry.

Next steps

The following sections describe these tasks in more detail.

Prepare a new keystore with the replacement key pair

The self-signed certificate can be replaced with an existing key pair. As an alternative, the certificate that is associated with the original key pair can be used.

Use an existing key pair

If a private key and certificate in PEM-encoded format already exist, both the original private key and the self-signed certificate can be replaced in `ads-truststore` by using the `manage-certificates` tool. Depending on your operating system, the `manage-certificates` tool is located in the server's `bin` or `bat` directory.

Important: If the existing key pair is not in PEM-encoded format, convert it to a format that is compatible with the server's `ads-truststore` keystore file format before proceeding.

If you replace the entire key pair instead of only the certificate that is associated with the original private key, your existing backups and LDIF exports might be rendered invalid. To avoid this scenario, perform this step immediately after setup, or at least before the key pair is used. After the first use, change only the certificate associated with the private key to extend its validity period, or to replace it with a certificate that is signed by a different CA.

The following command imports existing certificates into a new keystore file named `ads-truststore.new`:

```
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
```

Order the certificates that use the `--certificate-file` option in such a manner that each subsequent certificate functions as the issuer for the previous one. The server certificate is listed first, any intermediate certificates are listed next, and the root CA certificate is listed last. Because some deployments do not feature an intermediate issuer, you might need to import only the server certificate and a single issuer.

Replace the certificate associated with the original key pair

About this task

Alternatively, to replace the certificate that is associated with the original server-generated, `ads-certificate` private key, perform the following steps:

Steps

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

```
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`, as follows:

```
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 – including the CA-signed server certificate, the root CA certificate, and any intermediate certificates – into `ads-truststore.new`, as follows:

```
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
```

Import earlier trusted certificates into the new keystore

About this task

The new `ads-truststore` file, `ads-truststore.new`, contains only the server's new key pair. You must import the currently trusted certificates of other servers in the topology.

To export trusted certificates from `ads-truststore` and import them into `ads-truststore.new`, perform the following steps for each trusted certificate:

Steps

1. Locate the currently trusted certificates, as follows:

```
manage-certificates list-certificates \
  --keystore ads-truststore
```

2. For each alias other than `ads-certificate`, or whose fingerprint does not match `ads-certificate`, perform the following steps:

- a) Export the trusted certificate from `ads-truststore`, as follows:

```
manage-certificates export-certificate \
  --keystore ads-truststore \
  --keystore-password-file ads-truststore.pin \
  --alias <trusted-cert-alias> \
  --export-certificate-chain \
  --output-file <trust-cert-alias>.crt
```

- b) Import the trusted certificate into `ads-truststore.new`, as follows:

```
manage-certificates import-certificate \
  --keystore ads-truststore.new \
  --keystore-type JKS \
  --keystore-password-file ads-truststore.pin \
  --alias <trusted-cert-alias> \
  --certificate-file <trusted-cert-alias>.crt
```

Update the server configuration to use the new certificate

About this task

Before updating the server to use the appropriate key pair, update the `inter-server-certificate` property for the server instance in the topology registry. To support the transition from an existing certificate to a new one, earlier and newer certificates might appear within their own beginning and ending headers in the `inter-server-certificate` property.

To update the server configuration to use the new certificate, perform the following steps:

Steps

1. Export the server's previous `ads-certificate` into `old-ads.crt`, as follows:

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

2. Concatenate the previous and new certificate into one file.

On Windows, use a text editor like Notepad. On Unix, use the following command:

```
cat old-ads.crt new-ads.crt > old-new-ads.crt
```

3. Use `dsconfig` to update the `inter-server-certificate` property for the server instance in the topology registry, as follows:

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

Replace the previous `ads-truststore` file with the new one

Because the server still uses the previous `ads-certificate`, you must replace the previous `ads-truststore` file with `ads-truststore.new` in the server's `config` directory when you want the new `ads-certificate` to go into effect:

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

Retire the previous certificate

Retire the previous certificate by removing it from the topology registry after it expires, as follows:

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

Existing encrypted backups and LDIF exports remain unaffected. Because the public key is the same in the previous and new server certificates, the private key can decrypt them.

Server certificate

During setup, administrators have the option of using self-signed certificates or CA-signed certificates for the server certificate. Where possible, we encourage the use of CA-signed certificates. Self-signed certificates are recommended only for demonstration and proof-of-concept environments.

If you specify the option `--generateSelfSignedCertificate` during setup, the server certificate is generated automatically with the alias `server-cert`. The key pair consists of the private key and the self-signed certificate, and is stored in a file named `keystore`, which resides in the server's `/config`

directory. The certificates for all the servers that the server trusts are stored in the `truststore` file, which is also located under the server's `/config` directory.

To override the server certificate alias and the files that store the key pair and certificates, use the following arguments during setup:

- `--certNickname`
- `--use*Keystore`
- `--use*Truststore`

For more information about these arguments, refer to the setup tool's Help and the Installation Guide.

Important: If the server's access logs contain authentication (bind) errors, the inter-server certificate is most likely configured inappropriately. In the topology registry, this certificate is persisted in a Server Instance Listener's listener-certificate property.

Replace the server certificate

About this task

Regardless of whether the server was set up with self-signed or CA-signed certificates, the steps to replace the server certificate are nearly identical.

This task makes the following assumptions:

- You are replacing the self-signed server certificate.
- The certificate alias is `server-cert`.
- The private key is stored in `keystore`.
- The trusted certificates are stored in `truststore`.
- The `keystore` and `truststore` use the JKS keystore format.

If a PKCS#12 keystore format was used for the `keystore` and `truststore` files during setup, change the `--keystore-type` argument in the `manage-certificate` commands to `PKCS12` in the relevant steps.

Important: Before attempting to replace the inter-server certificate, ensure that all servers in the topology are updated to version 7.0 or later.

While the certificate is being replaced, existing secure connections continue to work. If the server is restarted, or if a topology change requires a reset of peer connections, the server continues authenticating with its peers, all of whom trust the new certificate.

To replace the server certificate with no downtime, complete the following tasks:

Steps

1. Prepare a new keystore with the replacement key pair.
2. Import the earlier trusted certificates into the new `truststore` file.
3. Update the server configuration to use the new certificate by adding it to the server's list of listener certificates in the topology registry.
After this step is performed, other servers will trust the certificate.
4. Replace the server's `keystore` and `truststore` files with the new ones.
5. Retire the previous certificate by removing it from the topology registry.

Next steps

The following sections describe these tasks in more detail.

Prepare a new keystore with the replacement key pair

The self-signed certificate can be replaced with an existing key pair. As an alternative, the certificate that is associated with the original key pair can be used.

Use an existing key pair

If a private key and certificate already exist in PEM-encoded format, they can replace both the original private key and the self-signed certificate in `keystore` (instead of replacing the self-signed certificate associated with the original server-generated private key). Use the `manage-certificates` tool that, depending on your operating system, is located in the server's `bin` or `bat` directory.

The following command imports existing certificates into a new keystore file named `keystore.new`:

```
manage-certificates import-certificate \
  --keystore keystore.new \
  --keystore-type JKS \
  --keystore-password-file keystore.pin \
  --alias server-cert \
  --private-key-file existing.key \
  --certificate-file existing.crt \
  --certificate-file intermediate.crt \
  --certificate-file root-ca.crt
```

Order the certificates that use the `--certificate-file` option in such a manner that each subsequent certificate functions as the issuer for the previous one. The server certificate is listed first, any intermediate certificates are listed next, and the root CA certificate is listed last. Because some deployments do not feature an intermediate issuer, you might need to import only the server certificate and a single issuer.

*Replace the certificate associated with the original key pair***About this task**

If the certificate that is associated with the original server-generated private key (`server-cert`) has expired or must be replaced with a certificate from a different CA, perform the following steps to replace it:

Steps

1. Create a CSR file for the `server-cert`, as follows:

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

2. Submit `server-cert.csr` to a CA for signing.
3. Export the server's private key into `server-cert.key`, as follows:

```
manage-certificates export-private-key \
  --keystore keystore \
  --keystore-password-file keystore.pin \
  --alias server-cert \
  --output-file server-cert.key
```

4. Import the certificates obtained from the CA – including the CA-signed server certificate, the root CA certificate, and any intermediate certificates – into `keystore.new`, as follows:

```
manage-certificates import-certificate \
  --keystore keystore.new \
  --keystore-type JKS \
  --keystore-password-file keystore.pin \
  --alias server-cert \
  --private-key-file server-cert.key \
```

```
--certificate-file server-cert.crt \
--certificate-file intermediate.crt \
--certificate-file root-ca.crt
```

Import earlier trusted certificates into the new keystore

About this task

The trusted certificates of other servers in the topology must be imported into the new `truststore` file. To export trusted certificates from `truststore` and import them into `truststore.new`, perform the following steps for each trusted certificate:

Steps

1. Locate the currently trusted certificates, as follows:

```
manage-certificates list-certificates \
--keystore truststore
```

2. For each alias other than `server-cert`, or whose fingerprint does not match `server-cert`, perform the following steps:

- a) Export the trusted certificate from `truststore`, as follows:

```
manage-certificates export-certificate \
--keystore truststore \
--keystore-password-file truststore.pin \
--alias <trusted-cert-alias> \
--export-certificate-chain \
--output-file trusted-cert-alias.crt
```

- b) Import the trusted certificate into `truststore.new`, as follows:

```
manage-certificates import-certificate \
--keystore truststore.new \
--keystore-type JKS \
--keystore-password-file truststore.pin \
--alias <trusted-cert-alias> \
--certificate-file trusted-cert-alias.crt
```

Update the server configuration to use the new certificate

About this task

Before updating the server to use the appropriate key pair, update the `listener-certificate` property for the server instance's LDAP listener in the topology registry. To support the transition from an existing certificate to a new one, earlier and newer certificates might appear within their own beginning and ending headers in the `listener-certificate` property.

To update the server configuration to use the new certificate, perform the following steps:

Steps

1. Export the server's previous `server-cert` into `old-server-cert.crt`, as follows:

```
manage-certificates export-certificate \
--keystore keystore \
--keystore-password-file keystore.pin \
--alias server-cert \
--output-file old-server-cert.crt
```

2. Concatenate the previous and new certificate into one file.

On Windows, use a text editor like Notepad. On Unix, use the following command:

```
cat old-server-cert.crt new-server-cert.crt > old-new-server-cert.crt
```

3. Use `dsconfig` to update the `listener-certificate` property for the server instance's LDAP listener in the topology registry, as follows:

```
$ bin/dsconfig -n set-server-instance-listener-prop \
  --instance-name instance-name> \
  --listener-name ldap-listener-mirrored-config \
  --set "listener-certificate<old-new-server-cert.crt"
```

Replace the keystore and truststore files with the new ones

Because the server still uses the previous `server-cert`, you must replace the earlier `keystore` and `truststore` files with the new ones in the server's `config` directory when you want the new `server-cert` to take effect.

```
$ mv keystore.new keystore
   mv truststore.new truststore
```

Retire the previous certificate

Retire the previous certificate by removing it from the topology registry after it expires, as follows:

```
$ dsconfig -n set-server-instance-listener-prop \
  --instance-name <instance-name> \
  --listener-name ldap-listener-mirrored-config \
  --set "listener-certificate<new-server-cert.crt"
```

Capture debugging data

This section provides instructions for exporting all Trust Framework and policy data from the PingDataGovernance Policy Administration GUI, which is powered by Symphonic, to a *snapshot* that captures all of the policy data contained within a branch of the PingDataGovernance Policy Administration GUI. Snapshots provide a convenient way to load policy data into a separate PingDataGovernance Policy Administration GUI instance.

Export policy data

About this task

To export policy data, perform the following steps:

Steps

1. Navigate to **Change Control**.
2. Click **Version Control**.
3. Click the name of the branch to export.
4. Click **Options** and select **Export Snapshot**.
A snapshot file is downloaded to your computer.

Enable detailed logging

This section provides instructions for enabling detailed debug logging for troubleshooting purposes. This level of logging captures request and response data that contains potentially sensitive information.

Note: Do not use this level of logging when working with actual customer data.

Policy Decision logger

Enabled by default, the *Policy Decision logger* records decision responses that are received from the PDP. Regardless of whether PingDataGovernance Server is configured to evaluate a policy in Embedded or External mode, a policy-decision file logs every policy decision per request. This file is located at `PingDataGovernance/logs/policy-decision` and contains the following information:

- Policy-decision response – Each client request triggers a *policy-decision response* that specifies the inbound actions to perform, and another policy-decision response that specifies the outbound actions to perform. If you think of a policy-decision response as a set or decision tree of policies, all inbound and outbound requests are read from that set or tree.

Policy rules determine whether a request is denied, permitted, or indeterminate.

- Most recent policy decision – To debug the most recent inbound request, open the policy-decision log file and locate the highest `DECISION requestID` in the section near the bottom of the file. In the following example, `[08/May/2019:15:35:04.791 -0500] "DECISION requestID=46"` represents the most recent request, and `action equals "inbound-GET"`.

```
[08/May/2019:15:35:04.791 -0500] DECISION requestID=46 correlationID="0349a205-6aeb-4bd6-923b-c777bcef2241" product="Ping Identity
Data Governance Server" instanceName="dgl" startupID="XNM9Hw==" threadID=140 from=[0:0:0:0:0:1]:49882 method=GET url="https://
0:0:0:0:0:0:1:8443/jokes/random" clientId="" action="inbound-GET" service="Random Joke API" domain="" identityProvider="Mock
Access Token Validator" resourcePath="" deploymentPackageId="95c5864c-b7ab-4588-a3d6-99d99d09fafc"
decisionId="734fc520-ff1e-4f80-970a-12100cdd7646" authorized="true" decision="PERMIT" decisionStatusCode="OKAY" adviceIds=""
adviceNames=""
```

```
{
  "id" : "734fc520-ff1e-4f80-970a-12100cdd7646",
  "deploymentPackageId" : "95c5864c-b7ab-4588-a3d6-99d99d09fafc",
  "elapsedTime" : 1036,
  "request" : {
```

Alternatively, you can use the most recent request timestamp to locate the most recent request.

- Policy advice – If the policy contains advice, it is logged after the policy-decision response JSON. Advice features the same corresponding `requestID`, as the following example shows:

```
[08/May/2019:15:35:05.377 -0500] ADVICE requestID=46 correlationID="0349a205-6aeb-4bd6-923b-c777bcef2241" product="Ping Identity
Data Governance Server" instanceName="dgl" startupID="XNRLuQ==" threadID=139 from=[0:0:0:0:0:1]:56475 method=GET url="https://
0:0:0:0:0:0:1:8443/jokes/random" clientId="" action="outbound-GET" service="Random Joke API" resourcePath=""
deploymentPackageId="026ab83d-5ed5-41f1-ada7-a50af5d02133" decisionId="0331232d-cd9e-43fc-8804-c2f8b0c23674" authorized="false"
decision="DENY" decisionStatusCode="OKAY" adviceImplId="denied-reason" adviceImplName="Denied Reason Advice" obligatory="false"
resourceModified="true"
```

To increase the level of detail that is returned in PDP decision responses, configure the Policy Decision Service as follows:

```
dsconfig set-policy-decision-service-prop \
  --add decision-response-view:decision-tree \
  --add decision-response-view:request
```

Debug Trace logger

The *Debug Trace logger* records detailed information about the processing of HTTP requests and responses. The following example enables this log:

```
dsconfig set-log-publisher-prop \
  --publisher-name "Debug Trace Logger" \
  --set enabled:true
```

By default, the corresponding log file is located at `PingDataGovernance/logs/debug-trace`.

Debug logger

The *Debug logger* records debugging information that a developer might find useful. The following example enables this log:

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

dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name com.unboundid.directory.broker.http.gateway \
  --set debug-level:verbose

dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name \
  com.unboundid.directory.broker.config.GatewayConfigManager \
  --set debug-level:verbose

dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name \
  com.unboundid.directory.broker.core.policy.PolicyEnforcementPoint \
  --set debug-level:verbose

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

By default, the corresponding log file is located at `PingDataGovernance/logs/debug`.

Trace a policy-decision response

About this task

Before attempting to troubleshoot or trace a policy-decision response, make certain that the Trace log is enabled within PingDataGovernance Server. For more information, see [Configure PingDataGovernance logging](#) on page 11.

Each policy-decision response is presented in JSON format. To view the details of a policy-decision response, perform the following steps:

Steps

1. From within the policy-decision file, copy the policy-decision response JSON.
2. In the Policy Administration GUI, navigate to **Policies**.
3. Click the **Log Visualizer** tab.
4. In the **Log Input** text box, paste the policy-decision response JSON.
5. Click **Visualize**.

The screenshot shows the PingDataGovernance interface. On the left is a sidebar with navigation links: Change Control, Trust Framework, Policies (selected), and API Reference. The main area has three tabs: Policy Editor, Relationship Search, and Log Visualiser. The Log Visualiser tab is active, displaying a 'Log Input' window with a JSON policy set. Below the JSON is a 'Visualise' button. At the bottom left, it says 'Powered by SYMPHONIC'.

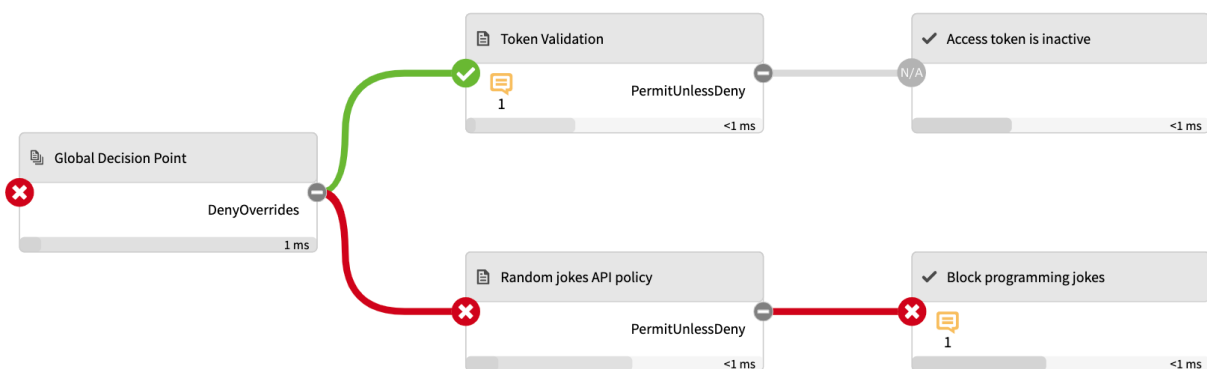
```

"SCIM Policy Set" : {
  "id" : "aa50a4c3-1652-4f41-8167-e80bf9bf4528",
  "nodeType" : "PolicySet",
  "name" : "SCIM Policy Set",
  "sequence" : 0,
  "targets" : [ {
    "name" : "Inline target",
    "domains" : [ ],
    "services" : [ "SCIM2" ],
    "actions" : [ ],
    "identityClasses" : [ ],
    "attributes" : { }
  } ],
  "elapsedTime" : 26,
  "combiningAlgorithm" :
  "DenyOverrides{",
  "decision" : "NOT_APPLICABLE",
  "statements" : [ ]
  },
  "statements" : [ ]
},
"evaluationLog" : [ ]
}

```

Results

An interactive decision tree of your policies is displayed.



This image depicts the final decision that is sent to the client. The node to the far left, *Global Decision Point*, represents the root node, and the children nodes contain the subset of policies and rules.

The following color-coded icons convey important information:

- Green **check mark** – Indicates that the request *permit* on the policy or rule.
- Red **X** – Indicates that the request *deny* on the policy or rule.
- Gray **N/A** – Indicates that the request is not applicable to the policy or rule.

In the previous example, the client received a final decision of *deny*. The Token Validation policy permitted the request initially but was overridden after the Random Jokes API policy was applied.

Capture debugging data with the collect-support-data tool

Run the `collect-support-data` tool to capture the PingDataGovernance Server's configuration, server state, environment, and other information that is useful for troubleshooting issues. When you run `collect-support-data`, the tool generates a compressed file that can be attached to a message or report.

```
PingDataGovernance/bin/collect-support-data
```

By default, the tool excludes log files that might contain sensitive customer information, including the debugging logs that are described in [Enable detailed logging](#) on page 109. When you use test data, send the following log files alongside `collect-support-data`'s compressed output file:

- `PingDataGovernance/logs/policy-decision`
- `PingDataGovernance/logs/debug-trace`
- `PingDataGovernance/logs/debug`

PingDataGovernance Policy Administration GUI single sign-on

To improve security and ensure a consistent authentication experience across all enterprise applications, enable single sign-on (SSO) for administrators, security professionals, and business analysts to authenticate and access the PingDataGovernance Policy Administration GUI, which is powered by Symphonic.

Reconfigure the PingDataGovernance Policy Administration GUI

Steps

1. To stop the application, run the `stop-server` command.
2. To reconfigure the application, run the `setup` command again.
3. Answer the following on-screen questions:
 - a) Select **OpenID Connect**.
 - b) Ensure that the API URL and the callback URL use the public DNS name of the PingDataGovernance Policy Administration GUI server.
Use the paths `/api/` and `/idp-callback`, as specified.
 - c) Ensure that the PingFederate discovery endpoint uses PingFederate Server's public DNS name.

Note: You must specify the PingFederate discovery endpoint twice.
4. To restart the application, run the `start-server` command.

Next steps

Before using a browser to log on to the GUI, configure the PingFederate dependencies.

The following transcript represents an example SSO configuration:

```
$ bin/setup
Use Basic auth (for testing) or OpenID Connect (for production) [b, o]? o
Provide a license file [bin/PingDataGovernance.lic]: /home/centos/
pingdatagovernance.70.lic
```



```

What port should the application run on? [8080]: 8080
What URL should be used to access the Symphonic API? (Port must match.)
  [http://localhost:8080/api/]: http://pap.example.com:8080/api/
What is the URL to your PingFederate discovery endpoint? [https://
localhost:9031/.well-known/openid-configuration]: https://
sso.example.com:9031/.well-known/openid-configuration
What is the URL to your PingFederate discovery endpoint? (As above)
  [https://localhost:9031/.well-known/openid-configuration]: https://
sso.example.com:9031/.well-known/openid-configuration
What should the callback URL for OIDC be? (Port must match.) [https://
localhost:8080/idp-callback]: http://pap.example.com:8080/idp-callback

```

Configuration Summary:

```

server.applicationConnectors[0].port: 8080
ui.REST_URL: http://pap.example.com:8080/api/
security.oidcConfigurationEndpoint: https://
sso.example.com:9031/.well-known/openid-configuration
ui.auth.configurationEndpoint: https://
sso.example.com:9031/.well-known/openid-configuration
security.redirectUri: http://pap.example.com:8080/idp-
callback

```

```
>>>> Configuration written to /home/centos/PingDataGovernance-PAP/bin/
configuration.yml
```

To start the server, run the bin/start-server script.

PingFederate dependencies

For OIDC-based SSO, the PingDataGovernance Policy Administration GUI requires the following configuration details in PingFederate:

- OAuth client named `pingdatagovernance-pap`
- Redirect URL configuration matches the OAuth redirect URL provided to `bin/setup`, like `http://<server>:8080/idp-callback`
- Cross-origin resource sharing (CORS) configuration includes the scheme, host, and port of the PingDataGovernance Policy Administration GUI, like `http://<server>:8080`
- Support for the response type token `id_token`
- Support for the following OIDC scopes:
 - `openid`
 - `email`
 - `profile`
- Support for the following claims:
 - `sub`
 - `name`
 - `email`
- The issuance criteria is configured so that only entitled users can access the application

For information about supporting these dependencies, refer to your PingFederate documentation.

PingFederate example configuration

The PingDataGovernance Policy Administration GUI relies on external access authorization, which is handled in one of the following manners:

- PingFederate authorizes external access through token-issuance criteria.
- PingAccess uses access rules to authorize external access.

The following example configuration assumes that a group of policy administrators is configured in PingDataGovernance, and expects the PingFederate token-issuance criteria to verify group membership.

OAuth server settings

Steps

1. If necessary, perform the following steps to add OIDC scopes:
 - a) Click **OAuth Server > Scope Management > Common Scopes**.
 - b) Add the following scopes:
 - email
 - profile
 - c) Click **Save**.
2. To create an authentication policy contract, perform the following steps:
 - a) Click **Identity Provider > Policy Contracts**.
 - b) Click **Create**.
 - c) Extend the contract with `name` and `email` attributes.
 - d) For group-based access authorization, extend the contract with `isMemberOf`, which the issuance criteria uses at a later time.
 - e) Click **Next**.
 - f) Click **Done**.
 - g) Click **Save**.
3. To map your authentication policy contract for persistent grants, perform the following steps:
 - a) Click **OAuth Server > Authentication Policy Contract Mapping**.
 - b) Select the authentication policy contract that you created in the previous step.
 - c) Click **Add Mapping**.
 - d) Click **Next**.
 - e) Fulfill the contract with the following values from the authentication policy contract:
 - For the `USER_KEY`, specify the `subject`.
 - For the `USER_NAME`, specify the `name`.
 - f) Click **Save**.
4. To create the access token manager, perform the following steps:
 - a) Click **OAuth Server > Access Token Management > Create**.
 - b) From the **Type** drop-down list, select `JSON Web Tokens`.
 - c) Add a row to the symmetric keys, using 64 hexadecimal characters.
 - d) Select **JWS Algorithm HMAC using SHA-256**.
 - e) From the **Active Symmetric Key Id** drop-down list, select your symmetric key.
 - f) Click **Next**.
 - g) Select **Check all**.
 - h) Click **Next**.
 - i) Add the following attributes:
 - sub
 - email
 - name
 - j) Click **Next**.
 - k) Click **Save**.
5. To map your authentication policy contract to your access token manager, perform the following steps:

- a) Click **OAuth Server > Access Token Mapping**.
 - b) Select the authentication policy contract and the access token manager that you created during the previous steps.
 - c) Click **Add Mapping**.
 - d) Click **Next**.
 - e) Fulfill the contract with the following values from the authentication policy contract:
 - For the `email`, specify the `email`.
 - For the `name`, specify the `name`.
 - For the `sub`, specify the `subject`.
 - f) Click **Next**.
 - g) To authorize access, add the following issuance criteria:
 1. From the **Source** drop-down list, select `Authentication Policy Contract`.
 2. From the **Attribute Name** drop-down list, select `isMemberOf`.
 3. From the **Condition** drop-down list, select `multi-value contains DN`.
 4. Type the full DN of the group of policy administrators, like `cn=Policy Admins,ou=Groups,dc=example,dc=com`.
 5. Click **Add**.
 - h) Click **Next**.
 - i) Click **Save**.
6. To create the OIDC policy, perform the following steps:
- a) Click **OAuth Server > OpenID Connect Policy Management**.
 - b) Click **Add Policy**.
 - c) Select the access token manager that you created earlier.
 - d) Select the **include server information in the ID token** check box.
 - e) Extend the token lifetime to 120 minutes.
 - f) Click **Next**.
 - g) Delete all attributes except `email` and `name`.
 - h) Click **Next**.
 - i) Add the following scopes:
 - `email`
 - `profile`

The scopes are associated automatically with the `email` and `name` attributes.
 - j) Click **Next**.
 - k) Click **Next**.
 - l) Fulfill the contract from the matching attributes of the access token.
 - m) Click **Next**.
 - n) Click **Done**.
 - o) Click **Save**.
7. To create the OAuth client for the PingDataGovernance Policy Administration GUI, perform the following steps:
- a) In the **Clients** section of the **OAuth Server** page, click **Create**.
 - b) In the **Client ID** text box, type `pingdatagovernance-pap`.
 - c) Select **No client authentication**.
 - d) Add the redirect URL of the PingDataGovernance Policy Administration GUI that you entered while installing the GUI, like `http://pap.example.com:8080/idp-callback`.
 - e) For **Scopes**, select the following values:
 - `email`
 - `profile`

Note: If you are restricting common scopes, select `openid`, as well.

- f) From the **Grant Type** drop-down list, select `Implicit`.
 - g) From the **Default Access Token Manager** drop-down list, select the access token manager that you created earlier.
 - h) From the **Connect Policy** drop-down list, select `OpenID`.
 - i) Click **Save**.
8. To configure CORS, perform the following steps:
- a) Click **OAuth Server > Authorization Server Settings**.
 - b) Under **Cross-Origin Resource Sharing Settings**, add the base URL of the server on which you installed the PingDataGovernance Policy Administration GUI.
Specify the base URL as the scheme, host, and port, like `http://pap.example.com:8080`.
 - c) Click **Save**.

Identity provider settings

About this task

Create an authentication policy that fulfills your authentication policy contract with the required attributes from your identity provider. This process depends on your environment and typically involves the following steps:

Steps

1. An Identity Provider adapter provides values for the following required attributes:
 - `name`
 - `email`
 - `subject`
2. The Identity Provider adapter also provides the `isMemberOf` attribute for the group-based authorization that the previous section describes.
3. An authentication policy maps the authentication adapter.
4. Contract fulfillment maps the attributes of the Identity Provider adapter to the Authentication Policy contract.

Upgrade PingDataGovernance Server

Ping Identity periodically issues software release builds that introduce new features, enhancements, and fixes for improved server performance. To upgrade the current server code version, administrators can use PingDataGovernance Server's `update` utility.

This section presents an `update` scenario and discusses various implications that must be considered when you upgrade your server code.

Upgrade overview and considerations

The process of upgrading PingDataGovernance Server involves the following tasks:

1. Downloading a new version of the PingDataGovernance Server distribution `.zip` file, and extracting its contents to the server that you want to update.
2. Running the `update` utility, making certain to point the `--serverRoot` or `-R` option value from the new root server toward the installation that requires upgrading.

When performing an upgrade, consider the following points:

- An upgrade affects only the configuration of the server that is being upgraded. To upgrade multiple servers, perform the upgrade process on them individually. For information about updating the PingDataGovernance Policy Administration GUI server, refer to the *PingDataGovernance Policy Administration Guide*.
- The `update` tool verifies whether the version of Java that is installed meets the new server requirements. To simplify the process, install the version of Java that the new server supports before you run the `update` tool.
- Upgrades for PingDataGovernance Server are supported only from version 7.0.0.0 or later. If you attempt to upgrade from a version that is earlier than 7.3.0.0, you will sustain configuration loss. When you run the `update` tool, a message reminds you of this possibility.

Upgrading PingDataGovernance Server

About this task

This task makes the following assumptions:

- The existing server installation is located in `/prod/PingDataGovernance`.
- The new server version is extracted to `/home/stage/PingDataGovernance`.

Steps

1. Download and extract the contents of the new version of the PingDataGovernance Server distribution `.zip` file to a location outside the existing server's installation.
2. Copy the PingDataGovernance license file for the new version to `/home/stage/PingDataGovernance`.

-OR-

Use the `--licenseKeyFile` option to specify the location of the license file when you run the `update` tool in the next step.

3. Run the `update` tool that is provided with the new server package.

```
$ /home/stage/PingDataGovernance/update --serverRoot/prod/
PingDataGovernance
```

If the `update` tool detects changes to the server configuration, it might prompt you to confirm them.

Results

The existing version of PingDataGovernance Server is upgraded to the new version.

Reverting an update

About this task

Use the `revert-update` tool to revert back one level to the previous version of PingDataGovernance Server. The `revert-update` tool returns the file system to its prior state by accessing a log of file actions that the `update` tool performed. If you have performed multiple updates, run `revert-update` multiple times to revert to each prior update sequentially.

Important: You can revert back only one level. For example, if you have run the `update` tool twice since initially installing PingDataGovernance Server, run the `revert-update` tool to revert the installation to its previous state, and then run `revert-update` again to return the installation to its original state.

To revert back to the most recent version of PingDataGovernance Server, run the `revert-update` tool from the server root directory, as follows:

```
$ PingDataGovernance-old/revert-update
```

Next steps

When starting PingDataGovernance Server for the first time after a reversion, the server display warnings about offline configuration changes. These warnings are not critical and do not appear during subsequent startups.

PingDataGovernance Server 7.3.0.3 Release Notes

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance Server:

Ticket ID	Description
PDSTAGING-840	Fixed an issue that could cause the server to leak a small amount of memory each time it failed to establish an LDAP connection to another server.
DS-40371, DS-40382, DS-40427	<p>SCIM 2 search responses can now be authorized and filtered with an optimized authorization mode that uses a single policy request to process an entire result set. This authorization mode is optional. By default, the server creates a policy request for each member of a result set.</p> <p>This authorization mode is enabled on a per-request basis. To enable, a policy that targets the <code>SCIM2</code> service and the <code>search</code> action must provide an advice with the ID <code>combine-scim-search-authorizations</code> but with no payload. The subsequent search response is then authorized by using a single policy request with the 'SCIM2' service and the 'search-result' action. If advices are returned in the policy results, they are applied iteratively to each SCIM resource in the result set.</p> <p>For more information, refer to the <i>PingDataGovernance Server Administration Guide</i>.</p>

PingDataGovernance Server Release Notes archive

Release Notes for earlier versions of PingDataGovernance Server are included for reference.

PingDataGovernance Server 7.3.0.2 Release Notes

Upgrade Considerations

Important considerations for upgrading to this version of PingDataGovernance Server:

- If you are upgrading from PingDataGovernance 7.3.0.0 to 7.3.0.1 or 7.3.0.2, an updated version of the Policy Administration GUI is required.
- The Allow Attributes and Prohibit Attributes advices have been deprecated. If a deployment requires the behavior that these advices provided, use a Server SDK to implement the appropriate behavior.
- API Endpoints, which were introduced in 7.3.0.0, have been renamed to *Gateway API endpoints*.



Warning: When performing an update, existing API Endpoint configuration objects are migrated automatically. To reflect this change, manually update your `dsconfig` scripts and other automated deployments or configurations.

What's New

As a gateway, PingDataGovernance Server functions as a reverse proxy while in deployment mode. With 7.3.0.2, the Sideband API introduces an alternate deployment mode in which PingDataGovernance Server uses a plugin to connect to an existing API Lifecycle Gateway. In sideband deployment, the API Lifecycle Gateway handles requests between API clients and backend API services. The integration plugin intercepts all request data and passes it through PingDataGovernance Server, which authorizes requests and responses, and modifies request and response data.

Resolved Issues

The following table identifies issues that have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-38832	Added a property to Advice types that limits their application to <code>PERMIT</code> or <code>DENY</code> decisions.
DS-39037	The provided PingDataGovernance policies and deployment packages now apply access token validation policies only to the following requests: <ul style="list-style-type: none"> • Inbound • SCIM • OpenBanking
DS-39490, DS-39616	The API Endpoint configuration type has been renamed to <i>Gateway API Endpoint</i> . Update any existing <code>dsconfig</code> scripts that reference an API Endpoint. For example, a <code>dsconfig</code> command of <code>create-api-endpoint</code> must be changed to <code>create-gateway-api-endpoint</code> .
DS-39592	HTTP External Servers feature a new attribute, <code>certificate-alias</code> , which defines the alias of a specific certificate within the keystore to be used as a client certificate.
DS-39681	When PingDataGovernance Server receives a <code>401 - Unauthorized</code> response from an external policy decision server, it converts the status to <code>503 - Service Unavailable</code> for the upstream client.

Ticket ID	Description
DS-40234	The Open Banking account request endpoint no longer requires a value for <code>x-fapi-financial-id</code> . Instead, it now includes the configured <code>fapi-financial-id</code> value in policy requests through the <code>Gateway.FapiFinancialId</code> attribute. A policy can deny account requests based on the presence and value of this attribute.

PingDataGovernance Server 7.3.0.1 Release Notes

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance Server:

Ticket ID	Description
DS-17278	<p>Added a <code>cn=Server Status Timeline,cn=monitor</code> monitor entry to track a history of the local server's last 100 status changes and their timestamps.</p> <p>Updated the LDAP external server monitor to include attributes that track health-check state changes for external servers. The new attributes include the following information:</p> <ul style="list-style-type: none"> • Number of times a health-check transition has occurred • Timestamps of the most recent transitions • Messages associated with the most recent transitions
DS-37504, DS-38765, DS-39011	Fixed an issue in the <code>Passthrough SCIM</code> resource type that could cause an access token validator's token subject lookup to fail if the user store was unavailable when PingDataGovernance Server was started. This issue typically manifested as a SCIM schema error in the debug trace log, such as "Attribute uid in path uid is undefined."
DS-39176, DS-39308	<p>Updated the Groovy scripting language version to 2.5.7. For a list of changes, visit groovy-lang.org and view the Groovy 2.5 Release Notes.</p> <p>As of this release, only the core Groovy runtime and the <code>groovy-json</code> module are bundled with the server. To deploy a Groovy-scripted Server SDK extension that requires a Groovy module not bundled with the server, such as <code>groovy-xml</code> or <code>groovy-sql</code>, download the appropriate JAR file from groovy-lang.org and place it in the server's <code>lib/extensions</code> directory.</p>

Ticket ID	Description
DS-39564	Fixed an issue in which the gateway responded with a 404 for requests that were handled by a Gateway API Endpoint with an <code>inbound-base-path</code> of <code>"/</code> .
DS-39593	Fixed an issue in which policy decision logs contained content that the Policy Administration GUI Log Visualizer considered invalid.

PingDataGovernance Server 7.3.0.0 Release Notes

Upgrade Considerations

Important considerations for upgrading to this version of PingDataGovernance Server:

- **WARNING:** OAuth scope configurations for resource access control, including fine-grained access control, and JEXL-based policies are no longer supported. Manual steps are necessary to migrate configuration and policies in order to restore the functionality of SCIM APIs. Please contact your account executive to schedule time for migration assistance.

What's New

These are new features for this release of PingDataGovernance Server:

- New features for data encryption in transit and at rest: added support for TLS 1.3, ability to encrypt and automatically decrypt sensitive files such as `tools.properties` and keystore pin files using the server data encryption keys, and the ability to more easily and securely separate master keys from data encryption keys by protecting the server encryption settings database using either Amazon Key Management Service (AWS KMS) or HashiCorp Vault.
- Added support for Amazon Corretto JDK 8, Windows Server 2019, Red Hat Enterprise Linux 7.6, CentOS 7.6, Amazon Linux 2, and Docker 18.09.0 on Ubuntu 18.04 LTS.
- Fine-grained data access control for JSON-based APIs. Configured as a reverse proxy to existing customer API endpoints, PingDataGovernance enforces dynamic authorization policies to inbound API calls or outbound API responses. For inbound calls, policies can inspect request attributes and request bodies to allow or deny the HTTP call. For outbound responses, policies can whitelist or blacklist JSON objects and specific attributes, thus sanitizing the HTTP response data per use case.
- New Policy Administration GUI. Data owners and other stakeholders can now collaborate with IT and developers to build and test data access control policies. IT and developers configure services and attributes that gather, extract, and transform data dynamically from REST APIs, RDBMS, LDAP, and more. Data owners and other stakeholders build expressions to check and compare these attributes as part of a hierarchy of policies and rules. The Policy Administration GUI supports testing with mock input data, and it displays test results in a graphical tree to help policy writers understand and troubleshoot policy logic.

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance Server:

Ticket ID	Description
PDSTAGING-570,DS-38334	<p>The following enhancements were made to the topology manager to make it easier to diagnose the connection errors described in PDSTAGING-570:</p> <ul style="list-style-type: none">- Added monitoring information for all the failed outbound connections (including the time since it's been failing and the last error message seen when the failure occurred) from a server to one of its configured peers and the number of failed outbound connections.- Added alarms/alerts for when a server fails to connect to a peer server within a configured grace period.
PDSTAGING-570,DS-38344	<p>The topology manager will now raise a mirrored-subtree-manager-connection-asymmetry alarm when a server is able to establish outbound connections to its peer servers, but those peer servers are unable to establish connections back to the server within the configured grace period. The alarm is cleared as soon as there is connection symmetry.</p>
PDSTAGING-570,DS-38335	<p>The dsreplication tool has been fixed to work when the node being used to enable replication is currently out-of-sync with the topology master.</p>
DS-15734	<p>Added a cipher stream provider that can be used to protect the contents of the encryption settings database with a key from the Amazon Key Management Service.</p>

Ticket ID	Description
DS-18060	<p>Added an HTTP servlet extension that can be used to retrieve the server's current availability state. It accepts any GET, POST, or HEAD request sent to a specified endpoint and returns a minimal response whose HTTP status code may be used to determine whether the server considers itself to be AVAILABLE, DEGRADED, or UNAVAILABLE. The status code for each of these states is configurable, and the response may optionally include a JSON object with an "availability-state" field with the name of the current state.</p> <p>Two instances of this servlet extension are now available in the default configuration. A request sent to /available-state will return an HTTP status code of 200 (OK) if the server has a state of AVAILABLE, and 503 (Service Unavailable) if the server has a state of DEGRADED or UNAVAILABLE. A request sent to the /available-or-degraded-state will return an HTTP status code of 200 for a state of AVAILABLE or DEGRADED, and 503 for a state of UNAVAILABLE. The former may be useful for load balancers that you only want to have route requests to servers that are fully available. The latter may be useful for orchestration frameworks if you wish to destroy and replace any instance that is completely unavailable.</p>
DS-37617	<p>HTTP Connection Handlers now accept client-provided correlation IDs by default. To adjust the set of HTTP request headers that may include a correlation ID value, change the HTTP Connection Handler's correlation-id-request-header property.</p>
DS-37753	<p>PingDataGovernance now contains Server SDK support for Advices.</p>
DS-37839	<p>Make Fingerprint Certificate Mapper and Subject DN to User Attribute Certificate Mapper disabled by default on fresh installations. This will not affect upgrades from installations where these mappers are enabled.</p>

Ticket ID	Description
DS-37959	<p data-bbox="850 216 1464 275">Added support for insignificant configuration archive attributes.</p> <p data-bbox="850 296 1464 926">The configuration archive is a collection of the configurations that have been used by the server at some time. It is updated whenever a change is made to data in the server configuration, and it is very useful for auditing and troubleshooting. However, because the entries that define root users and topology administrators reside in the configuration, changes to those entries will also cause a new addition to the configuration archive. This is true even for changes that affect metadata for those entries, like updates to the password policy state information for one of those users. For example, if last login time tracking is enabled for one of those users, especially with high-precision timestamps, a new configuration may be generated and added to the configuration archive every time that user authenticates to the server. While it is important for this information to be persisted, it is not as important for it to be part of the server's configuration history.</p> <p data-bbox="850 947 1464 1199">This update can help avoid the configuration archive from storing information about updates that only affect this kind of account metadata. If a configuration change only modifies an existing entry, and if the only changes to that entry affect insignificant configuration archive attributes, then that change may not be persisted in the server's configuration archive.</p> <p data-bbox="850 1220 1464 1312">By default, the following attributes are now considered insignificant for the purpose of the configuration archive:</p> <p data-bbox="850 1333 1464 1675">* ds-auth-delivered-otp * ds-auth-password-reset-token * ds-auth-single-use-token * ds-auth-totp-last-password-used * ds-last-access-time * ds-pwp-auth-failure * ds-pwp-last-login-ip-address * ds-pwp-last-login-time * ds-pwp-password-changed-by-required-time * ds-pwp-reset-time * ds-pwp-retired-password * ds-pwp-warned-time * modifiersName * modifyTimestamp * pwdAccountLockedTime * pwdChangedTime * pwdFailureTime * pwdGraceUseTime * pwdHistory * pwdReset</p>

Ticket ID	Description
DS-38050	<p>Updated the server to support encrypting the contents of the PIN files needed to unlock certificate key and trust stores. If data encryption is enabled during setup, then the default PIN files will automatically be encrypted.</p> <p>Also, updated the command-line tool framework so that the tools.properties file (which can provide default values for arguments not provided on the command line), and passphrase files (for example, used to hold the bind password) can be encrypted.</p>
DS-38072	<p>Updated the server to enable TLSv1.3 by default on JVMs that support it (Java 11 and higher).</p>
DS-38085	<p>Fixed an issue in the installer where the Administrative Console's trust store type would be incorrectly set if it differed from the key store type.</p>
DS-38089,DS-38705	<p>The Open Banking Account Request servlet now supports versions 1.1, 2.0, and 3.0 of the Open Banking Read/Write Data API.</p> <p>Error responses returned by the Account Request servlet are now formatted as described in the Open Banking Read/Write Data API specification, v3.0.</p>
DS-38090,DS-38564,DS-38567	<p>The response header used for correlation IDs may now be set at the HTTP Servlet Extension level using the correlation-id-response-header configuration property. If set, this property overrides the HTTP Connection Handler's correlation-id-response-header property.</p>
DS-38109	<p>Added the --skipHostnameCheck command line option to the setup script, which bypasses validation of the provided host name for the server.</p>
DS-38403	<p>Fixed an issue that could prevent certain types of initialization failures from appearing in the server error log by default.</p>
DS-38512	<p>Added a cipher stream provider that can be used to protect the contents of the encryption settings database with a secret passphrase obtained from a HashiCorp Vault instance.</p>
DS-38550	<p>Fixed an issue in which backups of the encryption settings database could be encrypted with a key from the encryption settings database.</p>

Ticket ID	Description
DS-38670	Fixed a bug where the startIndex value for SCIM requests would be incorrect if the used LDAPSearch element had more than one baseDN defined in the scim-resources XML file.
DS-38737	Fixed an issue where inter-server bind requests would fail if the cipher used reported a maximum unencrypted block size of 0.
DS-38864	Changed the default value of the HTTP Configuration property include-stack-traces-in-error-pages from 'true' to 'false'. Disabling this property prevents information about exceptions thrown by servlet or web application extensions from being revealed in HTTP error responses.

Ticket ID	Description
DS-38897,DS-38908	<p>Fixed two issues in which the server could have exposed some clear-text passwords in files on the server file system.</p> <p>* When creating an encrypted backup of the alarms, alerts, configuration, encryption settings, schema, tasks, or trust store backends, the password used to generate the encryption key (which may have been obtained from an encryption settings definition) could have been inadvertently written into the backup descriptor. This problem does not affect local DB backends (like userRoot), the LDAP changelog backend, or the replication database.</p> <p>* When running certain command-line tools with an argument instructing the tool to read a password from a file, the password contained in that file could have been written into the server's tool invocation log instead of the path to that file. Affected tools include backup, create-initial-config, create-initial-proxy-config, dsreplication, enter-lockdown-mode, export-ldif, import-ldif, ldappasswordmodify, leave-lockdown-mode, manage-tasks, manage-topology, migrate-ldap-schema, parallel-update, prepare-endpoint-server, prepare-external-server, realtime-sync, rebuild-index, re-encode-entries, reload-http-connection-handler-certificates, reload-index, remove-defunct-server, restore, rotate-log, and stop-server. Other tools are not affected. Also note that this only includes passwords contained in files that were provided as command-line arguments; passwords included in the tools.properties file, or in a file referenced from tools.properties, would not have been exposed.</p> <p>In each of these cases, the files would have been written with permissions that make their contents only accessible to the system account used to run the server. Further, while administrative passwords may have been exposed in the tool invocation log, neither the passwords for regular users, nor any other data from their entries, should have been affected. We have introduced new automated tests to help ensure that such incidents do not occur in the future.</p> <p>We recommend changing any administrative passwords you fear may have been compromised as a result of this issue. If you are concerned that the passphrase for an encryption settings definition may have been exposed, then we recommend creating a new encryption settings definition that is preferred for all subsequent encryption operations, exporting your data to LDIF, and re-importing so that it will be encrypted with the new key. You also may wish to re-encrypt or destroy any existing backups, LDIF exports, or other data encrypted with a compromised key, and you may wish to sanitize or destroy any existing tool invocation log files that may contain clear-text passwords.</p>

Ticket ID	Description
DS-38913	Added a set of message types to Trace Log Publishers that records events related to access token validation.
DS-39086	Removed the version information page from the docs/build-info.txt endpoint. This information is now available in build-info.txt, which is located in the root directory.
DS-39102	Updated the server SDK class AccessTokenValidator's method initializeTokenValidator's parameters. The method's first parameter is now of type ServerContext instead of BrokerContext. This change is incompatible with earlier versions of the server SDK.

Index

D

document copyright [5](#)