

PingDataGovernance



Contents

PingDataGovernance Server Release Notes.....	5
PingDataGovernance Server 8.0.0.5 release notes.....	5
PingDataGovernance Server Release Notes archive.....	5
PingDataGovernance Server 8.0.0.4 Release Notes.....	5
PingDataGovernance Server 8.0.0.3 Release Notes.....	5
PingDataGovernance Server 8.0.0.2 Release Notes.....	6
PingDataGovernance Server 8.0.0.1 Release Notes.....	7
PingDataGovernance Server 8.0.0.0 Release Notes.....	10
PingDataGovernance Server 7.3.0.10 release notes.....	17
PingDataGovernance Server 7.3.0.9 Release Notes.....	18
PingDataGovernance Server 7.3.0.8 Release Notes.....	18
PingDataGovernance Server 7.3.0.7 Release Notes.....	19
PingDataGovernance Server 7.3.0.6 Release Notes.....	20
PingDataGovernance Server 7.3.0.5 Release Notes.....	20
PingDataGovernance Server 7.3.0.4 Release Notes.....	21
PingDataGovernance Server 7.3.0.3 Release Notes.....	21
PingDataGovernance Server 7.3.0.2 Release Notes.....	22
PingDataGovernance Server 7.3.0.1 Release Notes.....	23
PingDataGovernance Server 7.3.0.0 Release Notes.....	24
PingDataGovernance Server Administration Guide.....	31
PingDataGovernance™ Product Documentation.....	31
Introduction to PingDataGovernance Server.....	31
Key components.....	32
Explore PingDataGovernance Server.....	32
Install and configure PingDataGovernance Server.....	32
Install and configure the PingDataGovernance Policy Administration GUI.....	35
Import default policies.....	37
Create the first API policy.....	39
Create the first SCIM policies.....	53
Install PingDataGovernance Server.....	68
Before you begin.....	71
Installing PingDataGovernance Server.....	76
Running PingDataGovernance Server.....	91
About the API security gateway.....	94
Request and response flow.....	94
Gateway configuration basics.....	95
API security gateway authentication.....	96
API security gateway policy requests.....	97
About error templates.....	102
About the Sideband API.....	104
API gateway integration.....	104
Sideband API configuration basics.....	106
Sideband API authentication.....	106
Sideband API policy requests.....	109
Error templates.....	114
About the SCIM service.....	115
Request and response flow.....	116

SCIM configuration basics.....	118
SCIM endpoints.....	120
SCIM authentication.....	121
SCIM policy requests.....	122
Lookthrough limit.....	127
Disable the SCIM REST API.....	127
About the PDP API.....	127
Request and Response Flow.....	128
Policy administration.....	132
Create policies in a development environment.....	132
Use policies in a production environment.....	133
Environment-specific Trust Framework attributes.....	134
Advice.....	142
Add Filter.....	143
Allow Attributes.....	143
Combine SCIM Search Authorizations.....	144
Denied Reason.....	144
Exclude Attributes.....	144
Filter Response.....	145
Include Attributes.....	145
Modify Attributes.....	146
Modify Query.....	146
Prohibit Attributes.....	146
Access token validators.....	146
About access token validator processing.....	147
Access token validator types.....	148
Server configuration.....	151
Administration accounts.....	151
About the dsconfig tool.....	151
PingDataGovernance Administration Console.....	152
About the configuration audit log.....	152
About the config-diff tool.....	153
Certificates.....	153
Manage monitoring.....	161
StatsD monitoring endpoint.....	161
Sending metrics to Splunk.....	162
Capture debugging data.....	163
Export policy data.....	163
Enable detailed logging.....	163
Trace a policy-decision response.....	165
Capture debugging data with the collect-support-data tool.....	167
Upgrade PingDataGovernance Server.....	167
Upgrade overview and considerations.....	167
Upgrading PingDataGovernance Server.....	167
Reverting an update.....	168
Upgrade the PingDataGovernance Policy Administration GUI.....	168
PingDataGovernance Policy Administration Guide.....	171
Getting started.....	171
Introduction.....	171
Version control.....	173
Branches.....	173
Snapshots.....	174
Trust framework.....	177
Trust Framework overview.....	177

Domains (PDP API only).....	177
Services.....	178
Attributes.....	183
Actions.....	188
Identity classifications and IdP support.....	188
Named conditions.....	190
Testing.....	191
Policy management.....	192
Policy management overview.....	192
Policy sets, policies, and rules.....	192
Policies and policy sets.....	192
Testing.....	202
Analysis.....	203
Change control.....	204
Deployment packages.....	205
Advice types.....	205
Advice types overview.....	205
Add Filter.....	206
Allow Attributes.....	206
Combine SCIM Search Authorizations.....	206
Denied Reason.....	206
Exclude Attributes.....	207
Filter Response.....	207
Include Attributes.....	208
Modify Attributes.....	208
Modify Query.....	208
Prohibit Attributes.....	209
REST API.....	209
REST API documentation.....	209

PingDataGovernance Server Release Notes

PingDataGovernance Server 8.0.0.5 release notes

Critical fixes

This release of the Data Governance Server addresses critical issues from earlier versions. Update all affected servers appropriately.

No critical issues have been identified.

PingDataGovernance Server Release Notes archive

Release Notes for earlier versions of PingDataGovernance Server are included for reference.

PingDataGovernance Server 8.0.0.4 Release Notes

Critical fixes

This release of the PingDataGovernance addresses critical issues from earlier versions. Update all affected servers appropriately.

- Fixed an issue where mirrored subtree polling could produce config archive files that were identical or ignored the configured insignificant attributes list.
 - Fixed in: 8.0.0.4
 - Introduced in: 7.0.0.0
 - Support identifiers: DS-41762 SF#00675207 SF#00683777

Resolved issues

The following issues have been resolved with this release of the PingDataGovernance:

Ticket ID	Description
DS-41762	Fixed an issue where mirrored subtree polling could produce config archive files that were identical or ignored the configured insignificant attributes list.

PingDataGovernance Server 8.0.0.3 Release Notes

Resolved issues

The following issues have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-43288	<p>Updated setup and the replace-certificate tool to improve the way we generate self-signed certificates and certificate signing requests to make them more palatable to clients.</p> <p>To reduce the frequency with which administrators had to replace self-signed certificates, we previously used a very long lifetime for self-signed certificates generated by setup or the replace-certificate tool. However, some clients (especially web browsers and other HTTP clients) have started more strenuously objecting to certificates with long lifetimes, so we now generate self-signed certificates with a one-year validity period. The inter-server certificate (which is used internally within the server and does not get exposed to normal clients) is still created with a twenty-year lifetime.</p> <p>Also, the replace-certificate tool's interactive mode has been updated to improve the process that it uses to obtain information to include in the subject DN and subject alternative name extension for self-signed certificates and certificate signing requests. The following changes have been made in accordance with CA/Browser Forum guidelines:</p> <ul style="list-style-type: none"> ▪ When selecting the subject DN for the certificate, we listed a number of common attributes that might be used, including CN, OU, O, L, ST, and C. We previously indicated that CN attribute was recommended. We now also indicate that the O and C attributes are recommended as well. ▪ When obtaining the list of DNS names to include in the subject alternative name extension, we previously suggested all names that we could find associated with interfaces on the local system. In many cases, we now omit non-qualified names and names that are associated with loopback interfaces. We will also warn about any attempts to add unqualified or invalid names to the list. ▪ When obtaining the list of IP addresses to include in the subject alternative name extension, we previously suggested all addresses associated with all network interfaces on the system. We no longer suggest any IP addresses associated with loopback interfaces, and we no longer suggest any IP addresses associated in IANA-reserved ranges (for example, addresses reserved for private-use networks). The tool now warns about attempts to add these addresses for inclusion in the subject alternative name extension.
DS-43480	<p>Updated the system information monitor provider to restrict the set of environment variables that can be included. Previously, the monitor entry included information about all defined environment variables, which can be useful for diagnostic purposes. However, some deployments might include credentials, secret keys, or other sensitive information in environment variables, and that should not be exposed in the monitor. The server now only includes values from a predefined set of environment variables that are expected to be the most useful for troubleshooting problems and are not expected to contain sensitive information.</p>
DS-38535	<p>Fixed an issue that could cause the server to generate an administrative alert about an uncaught exception when trying to send data on a TLS-encrypted connection that is no longer valid.</p>
DS-43632	<p>Fixed an issue where the "format" field is omitted from the list of operational attribute schemas in the Directory REST API.</p>

PingDataGovernance Server 8.0.0.2 Release Notes

Resolved issues

The following issues have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-40551	<p>Fixed an issue that could prevent some tools from running properly with an encrypted <code>tools.properties</code> file.</p>

Ticket ID	Description
DS-41332	The use of an internal ScimInterface for Server SDK extensions is now deprecated. Support for this was removed from PingDataGovernance Server 8.1.0.0. This was previously available using the <code>getInternalScimInterface()</code> method of the <code>BrokerContext</code> class.
DS-40828	Fixed an issue where some state associated with a JMX connection was not freed after the connection was closed. This led to a slow memory leak in servers that were monitored by an application that created a new JMX connection each polling interval.
DS-42609	Fixed an issue in which the Directory REST API could fail to decode certain credentials when using basic authentication.
DS-41289	Fixed an issue that prevented password changes for topology administrators unless their password policy was configured to allow pre-encoded passwords.
DS-41236	To avoid inconsistencies, changing clustered configuration now requires all servers in the cluster to be on the same product version. Servers will not pull any clustered configuration from the master of the cluster if they are on a different product version.
DS-41235	Updated the <code>cn=Cluster</code> subtree to prevent clustered configuration changes when servers in the cluster have mixed versions. To make clustered configuration changes, either update all servers in the cluster to the same version, or temporarily create separate clusters by server version by changing the <code>cluster-name</code> property on the server instance configuration objects.
DS-41261	Fixed an issue with <code>manage-profile replace-profile</code> where certain configuration changes for recurring task chains were not being applied.
DS-41126	Updated the server to make the general monitor entry available to JMX clients.
DS-41054	Fixed an issue that stopped new extensions from being installed.
DS-42812	Upgraded to jetty 9.4.30.
DS-41074	Fixed an issue with the way the server reports memory usage after completing an explicitly requested garbage collection.
DS-42218, DS-42232	Fixed an issue in which the PingDataGovernance Gateway generated error responses that did not include a correlation ID.
DS-41234, DS-41264	Fixed an issue where the SCIM Impacted Attributes Provider would return all the attributes of a SCIM PUT request instead of only those that have been modified.

PingDataGovernance Server 8.0.0.1 Release Notes

Upgrade Consideration

Important consideration for upgrading to this version of PingDataGovernance Server:

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 8.0.0.1, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

Critical Fixes

This release of the Data Governance Server addresses critical issues from earlier versions. Update all affected servers appropriately.

- Addressed an issue that could lead to slow, off-heap memory growth. This only occurred on servers whose `cn=Version,cn=monitor` entry was retrieved frequently.
 - Fixed in: 8.1.0.0
 - Introduced in: 5.2.0.0
 - Support identifiers: DS-41301
- The following enhancements were made to the topology manager to make it easier to diagnose connection errors:
 - Added monitoring information for all the failed outbound connections (including the time since it has 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.
 - Fixed in: 7.3.0.0
 - Introduced in: 7.0.0.0
 - Support identifiers: DS-38334 SF#00655578
- The topology manager now raises a `mirrored-subtree-manager-connection-asymmetry` alarm when a server can establish outbound connections to its peer servers but those peer servers cannot establish connections back to the server within the configured grace period. The alarm is cleared as soon as there is connection symmetry.
 - Fixed in: 7.3.0.0
 - Introduced in: 7.0.0.0
 - Support identifiers: DS-38344 SF#00655578
- Fixed two issues in which the server could have exposed some clear-text passwords in files on the server file system.
 - 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.
 - 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`, `ldappasswordmodify`, `manage-tasks`, `manage-topology`, `reload-http-connection-handler-certificates`, `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.

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 might 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.

We recommend changing any administrative passwords you fear might have been compromised as a result of this issue. If you are concerned that the passphrase for an encryption settings definition might have been exposed, then we recommend creating a new encryption settings definition that is preferred for all subsequent encryption operations. You also might want to re-encrypt or destroy any existing backups, LDIF exports, or other data encrypted with a compromised key, and you

might want to sanitize or destroy any existing tool invocation log files that might contain clear-text passwords.

- Fixed in: 7.3.0.0
- Introduced in: 7.0.0.0
- Support identifiers: DS-38897 DS-38908

Known Issues/Workarounds

The following item is a known issue in the current version of PingDataGovernance Server:

- The internal SCIM interface in the BrokerContext class of the Server SDK has been deprecated. It will be removed in a future version of the product. Extensions that need to interact with the SCIM service should use an HTTP client SDK or other means.

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance.

Ticket ID	Description
DS-40532	<p>Added a <code>logging-error-behavior</code> property to the log publisher, periodic stats logger plugin, and monitor history plugin configuration that you can use to specify the behavior the server should exhibit if an error occurs while attempting logging-related processing. By default, the server preserves its previous behavior of writing a message to standard error; however, you can configure it to enter lockdown mode on a logging error. In this mode, the server reports itself as unavailable and only accepts requests from accounts with the <code>lockdown-mode</code> privilege and only from clients communicating over a loopback interface.</p>
DS-40767, DS-41229	<p>Fixed an issue in which a PingDataGovernance Server could return an HTTP 500 error while logging the policy decision response if using these items:</p> <ul style="list-style-type: none"> ▪ External PDP mode ▪ The Policy Decision Service with a "decision-tree" decision response view ▪ A policy that uses a service with HTTP authentication <p>Also, the Policy Decision Logger now records external policy decisions to the policy decision log as a single line for easier use with the Policy Administration GUI Log Visualizer.</p>
DS-40980	<p>PingDataGovernance Server no longer prevents a server with an expired license from restarting.</p>

Ticket ID	Description
DS-41087	The Policy Administration GUI now includes decision evaluation details in the <code>decision-audit.log</code> by default. With this change, policy writers can visualize decisions by copying and pasting the JSON into the Log Visualizer.
DS-41301	Addressed an issue that could lead to slow, off-heap memory growth. This only occurred on servers whose <code>cn=Version,cn=monitor</code> entry was retrieved frequently.

PingDataGovernance Server 8.0.0.0 Release Notes

PingDataGovernance 8.0.0.0 Release Notes

Upgrade Considerations

Important considerations for upgrading to this version of the PingDataGovernance Server:

- Changes have been made to the Trust Framework in the default policies shipped with PingDataGovernance Server. Refer to the PingDataGovernance Server Administration Guide for instructions on updating existing policy deployments.
- Token Resource Lookup Methods, which are invoked after access token validation to obtain an access token owner's attributes from an external identity store, have been updated so that they do not strictly require SCIM. In this release, the existing SCIM-based method is provided, in addition to a new ability to create custom Token Resource Lookup Methods using the Server SDK.
- Token Resource Lookup Methods which were configured in existing deployments will be automatically migrated as SCIM Token Resource Lookup Methods during upgrade. Any existing `dsconfig` scripts that create Token Resource Lookup Methods should be updated to specify the `--type` parameter with the value "scim" before using these scripts with an upgraded server.
- An issue has also been fixed in which Token Resource Lookup Methods were not invoked after validating an access token with an Access Token Validator which was created using the Server SDK. The `TokenValidationResult` object returned by third-party Access Token Validators no longer includes the `tokenOwner` field, and extensions that set this field must be updated.
- API Endpoints, which were introduced in 7.3.0.0, have been renamed to Gateway API endpoints as of version 7.3.0.2.

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.

- The Allow Attributes and Prohibit Attributes advices have been deprecated. If a deployment requires the behavior that these advices provided, use the Server SDK to implement the appropriate behavior.
- Changes to the server configuration in this release of PingDataGovernance are incompatible with previous releases. This entails special consideration when upgrading a topology of servers that were set up using the setup tool's peer setup option. Once a server has been upgraded to the new version, an admin must manually apply configuration changes that could not be automatically applied by the update tool. The update tool will print out the instructions on how to do this.
- Changes to the server configuration in this release of PingDataGovernance are incompatible with previous releases. This entails special consideration when reverting a topology of servers to their

previous versions. All servers must be put into their own cluster before running revert-update using a `dsconfig` command like the following:

```
dsconfig set-server-instance-prop --instance-name <server-instance-name>
\
--set cluster-name:<unique-cluster-name>
```

In the above command, it is recommended that the cluster name be set to the server instance name, which is guaranteed to be unique.

- If you are upgrading from PingDataGovernance 7.3.0.x to 8.0.0.0, an updated version of the Policy Administration GUI is required
- Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. We encourage deployers to manage server configuration using server profiles, which support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide*.
- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 8.0.0.0, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

What's New

These are new features for this release of the PingDataGovernance Server:

- Use Server Profiles to reduce risk and improve consistency following the DevOps principle of infrastructure-as-code. Administrators can export the configuration of a PingDataGovernance instance to a directory of text files called a Server Profile, track changes to these files in version control like Git, and install new instances of PingDataGovernance or update existing instances of PingDataGovernance from a Server Profile. Server Profiles support variable substitution in order to remove the settings unique to each pre-production or production environment from the Server Profile that is stored in version control.
- Use PingDataGovernance with existing API Lifecycle Gateways. Previously, the PingDataGovernance Server functioned only as a reverse proxy. A new 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 data and passes it through the PingDataGovernance Sideband API. PingDataGovernance continues to enforce policy, authorizing requests and responses, and filtering or modifying request and response data.
- Improved handling of sensitive data through API Lifecycle Gateways. The Sideband API, which is the integration method between API Lifecycle Gateways and PingDataGovernance (introduced in 7.3.0.2), now supports filtering and modifying of API responses, in addition to authorizing requests. In this configuration, the integration plugin intercepts all response data, and passes it through the PingDataGovernance server, which filters and modifies the response data based on policies.
- Use external authorization in non-API use cases. Organizations can now externalize the authorization logic from other enforcement points, like legacy web applications, and manage these authorization policies centrally in PingDataGovernance. With this licensing option, other authorization enforcement points can call into the core policy engine of PingDataGovernance via a Policy Decision Point API (PDP API) that complies with the XACML JSON Profile Request API.
- More API request and response modification capabilities. Policy administrators can take advantage of new advice to replace JSON data values, even attributes that are deeply nested within API requests or responses. Also, administrators can define policy to manipulate the query string of API requests, useful for limiting upstream API calls based on the attributes of the caller or context.

- Additional HTTP request/response data is now provided to policies when an outbound response is processed by either the API Security Gateway or the Sideband API. A policy request for an outbound response may now include the following attributes, in addition to those already supported:
 - `HttpRequest.Response`
 - `HttpRequest.RequestHeaders`
 - `HttpRequest.RequestBody`
 - `HttpRequest.ResponseHeaders`
 - `HttpRequest.AccessToken`
 - `TokenOwner`

The existing `HttpRequest.Headers` policy request attribute is deprecated and will be removed in a future release of PingDataGovernance.

Known Issues/Workarounds

The following are known issues in the current version of the PingDataGovernance Server:

- When the PDP API receives a valid request, it first authorizes the client request itself before sending the client's policy request to the decision engine. As currently implemented, the PDP API ignores any advices in the decision, so the policy writer has no control over either the HTTP status code or the error message.
- The following are suggested solutions for problems with slow DNS:
 - Maintain a connection pool in the client app rather than opening new connections for each bind.
 - Add appropriate records, including PTR records, to DNS.
 - Add `options timeout:1` in the `/etc/resolv.conf` file and/or `options single-request`
 - If IPv6 requests specifically are causing issues, add `-Djava.net.preferIPv4Stack=true` to the `start-server.java-args` line in PingDirectory's `config/java.properties` file, run `bin/dsjavaproperties`, and restart the server to stop the issuance of IPv6 PTR requests.
 - Some server tools, such as `collect-support-data` and `rebuild-index`, will fail with errors if they are run with an encrypted `tools.properties` file.

Workaround: Add the `--noPropertiesFile` argument to the server tools to prevent them from pulling information from the encrypted file.

- The working directory value used by `exec` tasks is not implemented for recurring `exec` tasks.
- Deploying the Admin Console to an external container using JDK 11 requires downloading the following dependencies and making them available at runtime (for example, by copying them to the `WEB-INF/lib` directory of the exploded WAR file).
 - `groupId:jakarta.xml.bind, artifactId:jakarta.xml.bind-api, version:2.3.2`
 - `groupId:org.glassfish.jaxb, artifactId:jaxb-runtime, version:2.3.2`

Workaround: Deploy the Console in an external container using JDK 8.

Resolved Issues

The following issues have been resolved with this release of the PingDataGovernance Server:

Ticket ID	Description
DS-17278	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. Updated the LDAP external server monitor to include attributes tracking health check state changes for external servers. The new attributes include the number of times a health check transition has occurred, timestamps of the most recent transitions, and messages associated with the most recent transitions.

Ticket ID	Description
DS-37504, DS-38765, DS-39011	Fixed an issue in the Passthrough SCIM 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 would typically manifest as a SCIM schema error in the debug trace log, such as "Attribute uid in path uid is undefined."
DS-37565	A new advice type has been added, <code>modify-attributes</code> , which can modify the values of attributes.
DS-37720	<p>Added Token Resource Lookup Methods as a new type of Server SDK extension. A Token Resource Lookup Method can be used to customize the way that PingDataGovernance looks up access token owners to populate the TokenOwner attribute used to make policy decisions.</p> <p>For example, a developer could build a Token Resource Lookup Method that maps an access token subject to an identity stored in an RDBMS or an arbitrary REST API.</p>
DS-37881	The PingFederate Access Token Validator will now refresh its cached value of the PingFederate server's token introspection endpoint. A new attribute, <code>endpoint-cache-refresh</code> , has been added to the PingFederate Access Token Validator, which will determine how often this refresh occurs.
DS-37955	To support multiple trace loggers, each trace logger now has its own resource key, which is shown in the <code>Resource</code> column in the output of status. This key allows multiple alarms, due to sensitive message types for multiple trace loggers.
DS-38053	The JWT Access Token Validator no longer requires a restart after a change to one of its signing certificates.
DS-38515	The <code>requestURI</code> , <code>requestQueryParams</code> , <code>headers</code> , and <code>correlationID</code> attributes of the HTTP request have been made available when constructing an Error Template.
DS-38560	Updated <code>manage-profile replace-profile</code> to apply configuration changes directly, when possible. If the new server profile used by <code>replace-profile</code> has changed only the <code>dsconfig</code> batch files from the original profile, then only the <code>dsconfig</code> files are applied. If no changes are detected between profiles, <code>replace-profile</code> takes no action. If changes other than <code>dsconfig</code> are detected, the full <code>replace-profile</code> process is followed.
DS-38597	The Policy Administration GUI setup has been redesigned, allowing users to generate configuration through a command line tool more consistent with other Ping products.
DS-38777	Added support for updating the server version during <code>manage-profile replace-profile</code> . The server must have been originally set up with a server profile.
DS-38832	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-38832	A property has been added to Advice types that can limit their application to PERMIT or DENY decisions.

Ticket ID	Description
DS-38863	Updated the <code>manage-profile setup</code> subcommand to set a server's cluster name to match its instance name by default. This prevents servers in the same replication topology from being in the same cluster, reducing the risk of unintentionally overwriting parts of an existing server's configuration in a DevOps environment. The <code>--useDefaultClusterName</code> argument can be used to leave the cluster name unchanged.
DS-38867	Updated the PBKDF2 password storage scheme to add support for variants that use the 256-bit, 384-bit, and 512-bit SHA-2 digest algorithms. At present, the SHA-1 variant remains the default to preserve backward compatibility with older versions. Also, in accordance with the recommendations in NIST SP 800-63B, we have increased the default iteration count from 4096 to 10,000, and the default salt length from 64 bits to 128 bits.
DS-38869	Updated the <code>remove-defunct-server</code> tool's <code>--ignoreOnline</code> option. When using <code>--ignoreOnline</code> in a mixed-version environment, all servers must support the option.
DS-38968	A new advice type, <code>modify-query</code> , has been added which can modify the request query parameters before the request to the upstream server is made.
DS-39037	The provided PingDataGovernance policies and deployment packages now apply access token validation policies to inbound, SCIM, and OpenBanking requests only. With this change, an access token is no longer required to issue a Sideband API response request.
DS-39176, DS-39308	Updated the Groovy scripting language version to 2.5.7. For a list of changes, go to groovy-lang.org and view the Groovy 2.5 release notes. 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.
DS-39253	Added a <code>replace-certificate</code> tool, which can help an administrator replace the listener or inter-server certificate for a server instance.
DS-39322	Added support for PingDataGovernance to the <code>manage-profile</code> tool and its subcommands.
DS-39490, DS-39616	The API Endpoint configuration type has been renamed to Gateway API Endpoint. Any existing <code>dsconfig</code> scripts referencing an API Endpoint should be updated. For example, a <code>dsconfig</code> command of <code>create-api-endpoint</code> would need to be changed to <code>create-gateway-api-endpoint</code> .
DS-39518	Fixed an issue in which escaped characters in schema extensions may not be handled properly. If used in attribute type constraints (such as <code>X-VALUE-REGEX</code>), this could cause unexpected or incorrect behavior.

Ticket ID	Description
DS-39564	Fixed an issue in which the Gateway would respond with a 404 for requests handled by an API Endpoint with an inbound-base-path of "/".
DS-39592	HTTP External Servers have a new attribute, <code>ssl-cert-nickname</code> , which defines the alias of a specific certificate within their keystore to be used as a client certificate.
DS-39593	Fixed an issue where policy decision logs contained content that was considered invalid by the Policy Administration GUI Log Visualizer.
DS-39603	Fixed an issue where Server SDK extensions could not be configured by <code>dsconfig</code> batch files in the <code>manage-profile</code> tool.
DS-39626, DS-40357	The trace log publisher will now record an access token's scopes after the token is successfully validated.
DS-39643	Fixed an issue where a PUT request that attempted to delete less than 50 percent of the total items of a multivalued sub-attribute object resulted in the deletion of all items for that object.
DS-39654	Added support for the <code>--topologyFilePath</code> argument to the <code>manage-topology add-server</code> subcommand.
DS-39671	Updated the <code>manage-topology add-server</code> subcommand to require being run from the older server in a mixed-version environment.
DS-39681	When PingDataGovernance receives a 401 Unauthorized response from an external policy decision server, it will now convert the status to a 503 Service Unavailable for the upstream client.
DS-39715	Updated the Server SDK to add support for sending email messages.
DS-39735	The Server SDK's Advice API has been updated to provide the ability to modify multiple attributes of an HTTP request/response rather than just the body. Existing Advice extensions must be updated to use the new API.
DS-39857	Added the StatsD monitoring endpoint. When the Stats Collector Plugin is enabled, this endpoint sends metric data from the server in StatsD format to the configured destination.
DS-39877	Fixed an issue in which using an empty Error Template would cause the Sideband API to respond with a 500 Internal Server Error.
DS-39908	Added a new JVM-default trust manager provider that can be used to automatically trust any certificate signed by an authority included in the JVM's default set of trusted issuers. Also, updated other trust manager providers to offer an option to use the JVM-default trust addition to the trust that they normally provide.
DS-39913	Fixed a rare <code>NullPointerException</code> that could occur when recording advice metadata to the policy decision log.
DS-40114	Added a new <code>cn=Status Health Summary,cn=monitor</code> monitor entry that provides a summary of the server's current assessment of its health. This simplifies monitoring with third party tools that support retrieving monitoring data over JMX. The Periodic Stats Logger has also been updated to allow some of this monitoring information to be logged. No new information is logged by default.

Ticket ID	Description
DS-40234	The Open Banking account request endpoint no longer requires the <code>x-fapi-financial-id</code> to be present. Instead, it now includes the configured <code>fapi-financial-id</code> value in policy requests via <code>Gateway.FapiFinancialId</code> attribute. A policy can choose to deny account requests based on the presence and value of this attribute.
DS-40332	A check has been added to all DataGovernance policy requests which will cause them to fail if the version of the configured external Ping DataGovernance Policy Administration Point is not the same as the DataGovernance server. This will prevent potential errors that may otherwise arise from mismatched versions.
DS-40344	The API security gateway no longer forwards CORS-related request headers to upstream API servers. Likewise, it no longer forwards CORS-related response headers to clients. To use CORS with an API protected by the API security gateway, assign an HTTP Servlet Cross Origin Policy to the Gateway HTTP Servlet Extension.
DS-40354	Fixed a problem with <code>config-diff</code> when writing properties that span multiple lines using the <code>--prettyPrint</code> argument.
DS-40360	A new gauge has been created, DataGovernance Servlet Average Response Time (Milliseconds), which watches the average response time from Ping DataGovernance servlets, and can generate alarms and/or affect the server's available or degraded state. This gauge must be configured before it will have any effect; see the DataGovernance Administration Guide for details.
DS-40366	Fixed an issue where the server was attempting to connect by an IP address rather than a hostname when DNS lookup was successful.
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 the entire result set. This authorization mode is optional; by default, the server will continue to create 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 'SCIM2' 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 using a single policy request with the 'SCIM2' service and the <code>search-results</code> action. Any advices returned in the policy results are applied iteratively to each SCIM resource in the result set.</p> <p>For more information, see the PingDataGovernance Server Administration Guide.</p>
DS-40372	Added a new PDP API endpoint servlet extension. The PDP API endpoint accepts XACML-JSON requests and hands them off to the policy decision engine. It then converts the resulting policy decision to a XACML-JSON response for consumption by the client. To use this feature, customers must configure their PingDataGovernance servers with PDP API-enabled licenses.
DS-40377	Added support for logging to a JSON file in the Periodic Stats Logger Plugin.

Ticket ID	Description
DS-40517	Added metrics for status summary, replication database, and LDAP changelog to the Stats Collector Plugin.
DS-40542, DS-40554	Because the API Security Gateway may alter requests and responses as a result of policy processing, it no longer forwards request and response headers used for HTTP resource versioning and conditional requests. This includes the following headers: <code>If-None-Match</code> , <code>If-Modified-Since</code> , <code>If-Unmodified-Since</code> , <code>ETag</code> , and <code>Last-Modified</code> .
DS-40543	Updated <code>manage-profile</code> <code>replace-profile</code> to copy the tool log file to the server being updated.
DS-40556	Added support for specifying a working directory for exec tasks.
DS-40730	Updated the <code>encrypt-file</code> tool to prevent using the same path for both the input file and the output file.
DS-40771	Added a <code>--duration</code> argument to <code>collect-support-data</code> . When used, only the log files covering the specified duration before the current time will be collected.
DS-40784	Access Token Validator extensions built with the Server SDK may now provide the original access token value in addition to parsed claims when building a <code>TokenValidationResult</code> object. This access token value may be used by Token Resource Lookup Method extensions that do not need the parsed token claims to perform a subject lookup.

PingDataGovernance Server 7.3.0.10 release notes

Critical fixes

This release of PingDataGovernance Server addresses critical issues from earlier versions. Update all affected servers appropriately.

- Fixed an issue where mirrored subtree polling could produce config archive files that were identical or ignored the configured insignificant attributes list.
 - Fixed in: 7.3.0.10
 - Introduced in: 7.0.0.0
 - Support identifiers: DS-41762 SF#00675207 SF#00683777

Resolved issues

The following issues have been resolved with this release of the Data Governance Server.

Ticket ID	Description
DS-41762	Fixed an issue where mirrored subtree polling could produce config archive files that were identical or ignored the configured insignificant attributes list.

PingDataGovernance Server 7.3.0.9 Release Notes

Upgrade Considerations

Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. If you plan to upgrade to PingDataGovernance 8.0 at some point, consider using server profiles to manage server configuration. Introduced in PingDataGovernance 8.0, server profiles support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide* for PingDataGovernance 8.0 or later

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-38535	Fixed an issue that could cause the server to generate an administrative alert about an uncaught exception when trying to send data on a TLS-encrypted connection that is no longer valid.
DS-43480	Updated the system information monitor provider to restrict the set of environment variables that can be included. Previously, the monitor entry included information about all defined environment variables, as that information can be useful for diagnostic purposes. However, some deployments might include credentials, secret keys, or other sensitive information in environment variables, and that should not be exposed in the monitor. The server now only includes values from a predefined set of environment variables that are expected to be the most useful for troubleshooting problems, and that are not expected to contain sensitive information.

PingDataGovernance Server 7.3.0.8 Release Notes

Upgrade Considerations

This upgrade moves to Jetty 9.4. As a result, the HTTPS connection handler will no longer support TLS_RSA ciphers by default. If you use any legacy HTTPS clients that still require TLS_RSA ciphers, modify the `ssl-cipher-suite` property of the HTTPS Connection Handler to include them.

Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. If you plan to upgrade to PingDataGovernance 8.0 at some point, consider using server profiles to manage server configuration. Introduced in PingDataGovernance 8.0, server profiles support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide* for PingDataGovernance 8.0 or later

Resolved Issues

The following issues have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-40551	Fixed an issue that could prevent some tools from running properly with an encrypted tools.properties file.

Ticket ID	Description
DS-41126	Updated the server to make the general monitor entry available to JMX clients.
DS-41235	Updated the cn=Cluster subtree to prevent clustered configuration changes when servers in the cluster have mixed versions. To make clustered configuration changes, either update all servers in the cluster to the same version, or temporarily create separate clusters by server version by changing the cluster-name property on the server instance configuration objects.
DS-41236	To avoid inconsistencies, changing clustered configuration will now require all servers in the cluster to be on the same product version. Servers will not pull any clustered configuration from the master of the cluster if they are on a different product version.
DS-41261	Fixed an issue with manage-profile replace-profile where certain configuration changes for recurring task chains were not being applied.
DS-41289	Fixed an issue that prevented password changes for topology administrators unless their password policy was configured to allow pre-encoded passwords.
DS-42687	Upgrade to Jetty 9.4.30.

PingDataGovernance Server 7.3.0.7 Release Notes

Upgrade considerations

- Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. If you plan to upgrade to PingDataGovernance 8.0 at some point, consider using server profiles to manage server configuration. Introduced in PingDataGovernance 8.0, server profiles support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide* for PingDataGovernance 8.0 or later
- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.7, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

Resolved issues

The following issues have been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-37955	To support multiple trace loggers, each trace logger now has its own resource key, which is shown in the "Resource" column in the output of "status". This key allows multiple alarms, due to sensitive message types for multiple trace loggers.
DS-39799	Allows users who were migrated from the admin backend to the topology to manage the topology. Migrated users are granted the "manage-topology" privilege if they do not already have it.
DS-40366	Fixed an issue where the server was attempting to connect by an IP address rather than a hostname when DNS lookup was successful.
DS-40771	Added a <code>--duration</code> argument to <code>collect-support-data</code> . When used, only the log files covering the specified duration before the current time are collected.
DS-41054	Fixed an issue that stopped new extensions from being installed.

PingDataGovernance Server 7.3.0.6 Release Notes

Upgrade considerations

- Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. If you plan to upgrade to PingDataGovernance 8.0 at some point, consider using server profiles to manage server configuration. Introduced in PingDataGovernance 8.0, server profiles support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide* for PingDataGovernance 8.0 or later
- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.6, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

Critical fixes

This release has no critical fixes.

PingDataGovernance Server 7.3.0.5 Release Notes

Upgrade considerations

- Peer setup and clustered configuration are deprecated and will be removed in PingDataGovernance 9.0. If you plan to upgrade to PingDataGovernance 8.0 at some point, consider using server profiles to manage server configuration. Introduced in PingDataGovernance 8.0, server profiles support deployment best practices such as automation and Infrastructure-asCode (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide* for PingDataGovernance 8.0 or later

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.5, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

Critical fixes

This release has no critical fixes.

PingDataGovernance Server 7.3.0.4 Release Notes

Upgrade consideration

Important consideration for upgrading to this version of PingDataGovernance Server:

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.4, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

Resolved issue

The following issue has been resolved with this release of PingDataGovernance Server.

Ticket ID	Description
DS-40828	Fixed an issue where some state associated with a JMX connection was not freed after the connection was closed. This led to a slow memory leak in servers that were monitored by an application that created a new JMX connection each polling interval.

PingDataGovernance Server 7.3.0.3 Release Notes

Upgrade Consideration

Important consideration for upgrading to this version of PingDataGovernance Server:

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.3, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

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.

Ticket ID	Description
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 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.

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.2, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

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.
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

Upgrade Consideration

Important consideration for upgrading to this version of PingDataGovernance Server:

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.1, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

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	<p>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."</p>
DS-39176, DS-39308	<p>Updated the Groovy scripting language version to 2.5.7. For a list of changes, go to 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>
DS-39564	<p>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>.</p>
DS-39593	<p>Fixed an issue in which policy decision logs contained content that the Policy Administration GUI Log Visualizer considered invalid.</p>

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.

- If you are updating a multi-server topology from PingDataGovernance 7.0.x to 7.3.0.0, you must use the `--skipMirroredSubtreeUpdateTask` option for the updater or the update fails. Alternatively, you can uninstall all but one of the servers to retain the base configuration, update the standalone server, install fresh servers on the new version, and add them back to the topology with the peer options. However, using the `--skipMirroredSubtreeUpdateTask` option is the recommended path.

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, RBDMS, 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.

Ticket ID	Description
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>
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>
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>

Ticket ID	Description
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>
DS-37959	<p>Added support for insignificant configuration archive attributes.</p> <p>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>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>By default, the following attributes are now considered insignificant for the purpose of the configuration archive:</p> <ul style="list-style-type: none"> * 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

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> <ul style="list-style-type: none">* 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.* 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, ldappasswordmodify, 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>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. 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.

PingDataGovernance Server Administration Guide

PingDataGovernance™ Product Documentation

© Copyright 2004-2020 Ping Identity® Corporation. All rights reserved.
 © Copyright 2014-2020 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.

Explore PingDataGovernance Server

This section provides a tutorial for PingDataGovernance Server. As you complete this section, you will quickly get up and running with PingDataGovernance Server and its policy administration GUI. You will also learn how to implement data access policies for REST APIs and SCIM.

For more information about installing PingDataGovernance Server, including prerequisites and deployment options, see [Install PingDataGovernance Server](#) on page 68.

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 167.

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-8x-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-8x-license> \
  --rootUserDN "cn=directory manager" \
  --rootUserPassword <your-dg-password> \
  --ldapPort 8389 --ldapsPort 8636 \
  --httpsPort 8443 \
  --generateSelfSignedCertificate \
  --maxHeapSize 1g \
  --instanceName dg1 \
  --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 146.

```
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" \
--type 'scim' \
--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 163.

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 would you like to configure the Policy Administration GUI?**

Use option 1 (Quickstart) to set up a demo server with credentials `admin/password123`, and to use a self-signed certificate for SSL.

- **On which port should the Policy Administration GUI listen for HTTPS communications?**

You can use any unused port here, but most of the examples in this guide assume that port 9443 is used for the PingDataGovernance Policy Administration GUI.

- **Enter the fully qualified host name or IP address that users' browsers will use to connect to this GUI**

Unless you are testing on `localhost`, ensure that the provided API URL uses the public DNS name of the PingDataGovernance Policy Administration GUI server as shown in the following example:

```
pap.example.com
```

5. Copy and record any generated values needed to configure external servers.

The Shared Secret is used in PingDataGovernance, under **External Servers# Policy External Server# Shared Secret**.

6. 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:

```
[/opt/PingDataGovernance-PAP]$ bin/setup

Please enter the location of a valid PingDataGovernance with Symphonic license file
[/opt/PingDataGovernance-PAP/PingDataGovernance.lic]: /opt/PingDataGovernance/
PingDataGovernance.lic

PingDataGovernance Policy Administration GUI
=====

How would you like to configure the Policy Administration GUI?

  1) Quickstart (DEMO PURPOSES ONLY): This option configures the server with a form
     based authentication and generates a self-signed server certificate
  2) OpenID Connect: This option configures the server to use an OpenID Connect
     provider such as PingFederate
  3) Cancel the setup

Enter option [1]: 1

On which port should the Policy Administration GUI listen for HTTPS communications? [9443]: 9443

Enter the fully qualified host name or IP address that users' browsers will use to
connect to this GUI [centos.localdomain]: pap.example.com

Setup Summary
=====
Host Name:      pap.example.com
Server Port:    9443
Secure Access:  Self-signed certificate

Command-line arguments that would set up this server non-interactively:
  setup demo --port 9443 --certNickname server-cert \
    --licenseKeyFile /opt/PingDataGovernance/PingDataGovernance.lic \
    --pkcs12KeyStorePath config/keystore.pl2 --generateSelfSignedCertificate \
    --hostname pap.example.com

What would you like to do?
```

```

1) Set up the server with the parameters above
2) Provide the setup parameters again
3) Cancel the setup

Enter option [1]:

Setup completed successfully

Please configure the following values
=====
PingDataGovernance Server - Policy External Server
Base URL:                https://pap.example.com:9443
Shared Secret:           7ed6f52d6e71411ca9e58f9567c7de2e
Trust Manager Provider:  Blind Trust

Please start the server by running bin/start-server

```

Next steps

In this example, the PingDataGovernance Policy Administration GUI is now running and listening on port 9443. To log in to the interface, go to `https://<host>:9443`. The default credentials are `admin` and `password123`.

Note: Use the default user name and password logon credentials for demo and testing purposes only, such as this initial walk-through. To configure the PingDataGovernance Policy Administration GUI for PingFederate OIDC SSO, see [Configure Authentication Server for OpenID Connect single sign-on](#) on page 82.

Import default policies

About this task

After you log in to the PingDataGovernance Policy Administration GUI, 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

You configure PingDataGovernance Server to evaluate a 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.

Embedded mode removes the reliance on an external server for improved performance in a production environment and is covered in the chapter on Policy administration.

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.

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:	Do this:
Name	Specify <code>PingDataGovernance Policy Administration GUI</code> .
Base URL	Specify <code>https://<your_PingDataGovernance_Policy_Administration_GUI_host>:9443</code> .
Host Name Verification Method	Specify <code>allow-all</code> for test environments. If you are specifying a host name verification method for a nontest environment, you might need to configure additional mechanisms.
Trust Manager Provider	Specify <code>Blind Trust</code> and click the Edit Value button to ensure that this provider is <code>Enabled</code> .
User Id	Specify <code>admin</code> .
Branch	Specify the name of the branch that you created while importing the default policy snapshot (Default Policies).
Shared Secret	Click Set Value and enter the value previously generated in Install and configure the PingDataGovernance Policy Administration GUI on page 35 step 5.

3. To specify a value for the **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 **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 8636 --useSSL --trustAll \
  --bindDN "cn=directory manager" \
  --bindPassword <your-dg-password> \
  --server-name "PingDataGovernance Policy Administration GUI" \
  --type policy \
  --set base-url:https://<your-
PingDataGovernance_Policy_Administration_GUI_host>:9443 \
  --set hostname-verification-method:allow-all \
  --set "trust-manager-provider:Blind Trust" \
  --set user-id:admin \
```

```

--set "shared-secret:<your-shared-secret>" \
--set decision-node:<global-decision-point-id> \
--set "branch:Default Policies"

PingDataGovernance/bin/dsconfig set-trust-manager-provider-prop \
--no-prompt --port 8636 --useSSL --trustAll \
--bindDN "cn=directory manager" \
--bindPassword <your-dg-password> \
--provider-name "Blind Trust" \
--set enabled:true

```

Configure the policy service in External mode

About this task

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

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.

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 37.
4. Keep all the other default values.
5. Click **Save To Data Governance Server Cluster**.

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 Policy Administration GUI"

```

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](#)

Name * ?

Description ?

Base URL * ?

Hostname Verification Method ?

Key Manager Provider ?

Trust Manager Provider ?

Connect Timeout ?

Response Timeout ?

Allowed Header ?

If this property is not specified, then all end-to-end request headers will be forward

User Name ?

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:

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.

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**.

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
<input type="text" value="Search"/> 	<input type="text" value="Search"/> 
<div style="border: 1px solid #ccc; height: 100px;"></div>	<div style="border: 1px solid #ccc; padding: 5px;"> <div style="background-color: #007bff; color: white; padding: 2px;">Mock Access Token Validator</div> </div>  
 	
 	

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 manager" \
  --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-https-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.
 - a. Click **Add Resolver**.
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

The screenshot displays the configuration page for an attribute named "Joke". The interface is organized into several sections:

- Parent:** A dropdown menu showing "no parent selected".
- Description:** A text input field.
- Resolver Settings:**
 - Resolver Type:** A dropdown menu set to "Attribute".
 - Path:** A dropdown menu set to "HttpRequest.ResponseBody".
 - + Add Condition:** A button to add a condition.
 - + Add Resolver:** A button to add a new resolver.
- Cache Settings:**
 - Cache Strategy:** A dropdown menu set to "No Caching".
- Value Settings:**
 - Processor:** A dropdown menu set to "None".
 - Default value:** A checkbox that is currently unchecked.
 - Type:** A dropdown menu set to "JSON".
 - Secret:** A checkbox that is currently unchecked.

At the bottom of the form, there are two buttons: "Delete Attribute" (with a trash icon) and "Save changes" (with a download icon).

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`.
 - a. Click **Add Resolver**.
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 is organized into several sections:

- Parent:** A dropdown menu set to "Joke".
- Description:** A text input field.
- Resolver Settings:**
 - Resolver Type:** A dropdown menu set to "Attribute".
 - Condition:** A dropdown menu set to "Joke".
 - + Add Condition:** A button to add more conditions.
 - + Add Resolver:** A button to add more resolvers.
- Cache Settings:**
 - Cache Strategy:** A dropdown menu set to "No Caching".
- Value Settings:**
 - Processor:** A dropdown menu set to "JSON Path".
 - Value:** A text input field containing "\$.type".
 - Default value:** A checkbox that is unchecked.
 - Type:** A dropdown menu set to "String".
 - Secret:** A checkbox that is unchecked.

At the bottom right of the form, there is a **Save changes** button 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

The screenshot shows the configuration form for the 'Random Joke API'. It includes a gear icon and the title 'Random Joke API'. There is a 'Parent' dropdown menu currently set to 'no parent selected'. Below it is a 'Description' text area. Under the 'Service Settings' section, there is a 'Service Type' dropdown menu set to 'None'. At the bottom right, there is a 'Save changes' button with a green download icon.

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, go 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

The screenshot shows the 'Random jokes API policy' configuration page. The page is titled 'Random jokes API policy' and has a 'Disabled' checkbox. There is a 'Description' text area. Below this, the 'Applies to 2 definitions' section shows two items: 'outbound-GET' and 'Random Joke API'. A button below this section says 'Drag and Drop Targets or Targetable Definitions from Toolbox'. Below that, the 'Unless one decision is deny, the decision will be permit' section has a dropdown menu and a button that says '+ Create new Rule' or 'Drag and Drop Rules from Toolbox'. At the bottom, there are links for 'Hide "Applies to"', 'Show "Applies when"', 'Show Advice', and 'Show Properties'. A 'Save changes' button is at the bottom right.

Add logic to reject programming jokes

About this task

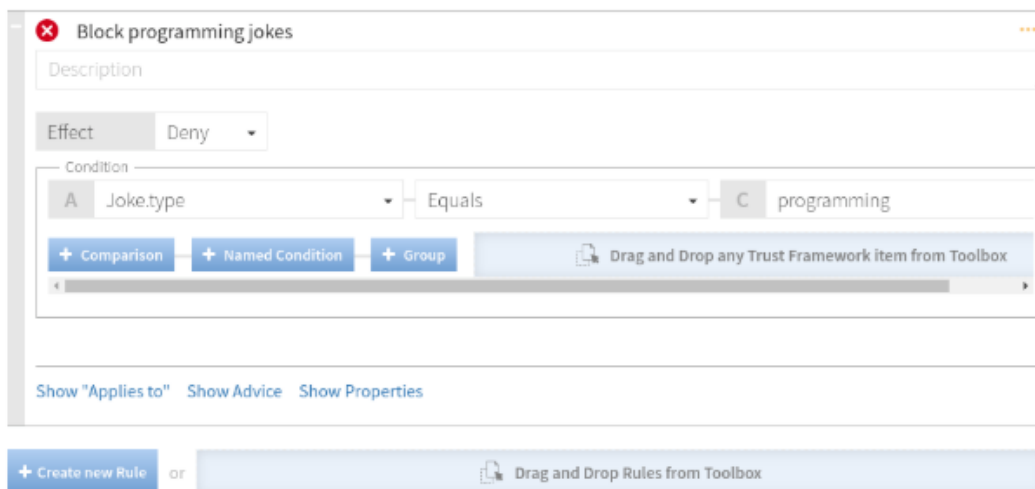
To add a rule that blocks responses with `programming` as the `joke.type` attribute value, perform the following steps:

Steps

1. Click **Create new Rule**.
2. For the **Name**, specify `Block programming jokes`.
3. For the **Effect**, select `Deny`.
 - a. Click **Create new Condition**.
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



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. Go 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:

The screenshot shows the 'Global Decision Point' interface in the 'Test' tab. The 'Request' section is configured with the following details:

- 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 with the following JSON body:


```
{
  "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"}}
```

Below the request configuration, there are 'Overrides' sections for 'Attributes' and 'Services', both currently empty. At the bottom right, there are 'Import' and 'Execute' buttons.

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 displays the configuration interface for a new attribute named "city". The interface is divided 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".
 - Value:** A text input field set to "TokenOwner".
 - + Add Resolver:** A button to add more resolvers.
- Cache Settings:**
 - Cache Strategy:** A dropdown menu set to "No Caching".
- Value Settings:**
 - Processor:** A dropdown menu set to "JSON Path".
 - Value:** A text input field set to "\$.l".
 - Default value:** A checked checkbox next to a text input field set to "[]".
 - Type:** A dropdown menu set to "Collection".
 - Secret:** A checkbox that is currently unchecked.
- Actions:** 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

The screenshot displays the configuration interface for a rule named "Block programming jokes". The rule's effect is set to "Deny". The condition is configured with two comparison rules: "A Joke.type Equals C programming" and "A TokenOwner.city Contains C Boston". The interface includes buttons for "+ Comparison", "+ Named Condition", and "+ Group", along with a "Drag and Drop any Trust Framework item from Toolbox" button. Below the condition section, there is an "Advice" section with one advice named "Send 'Bad gateway' response" and an "Obligatory" checkbox. At the bottom, there is a "Rule" section with a "+ Create new Rule" button and a "Drag and Drop Rules from Toolbox" button.

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", "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.

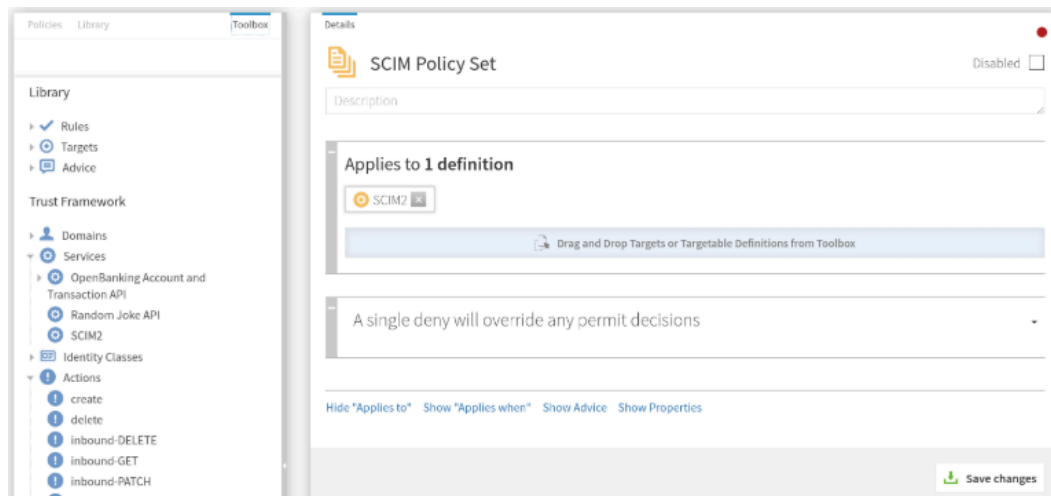
Create the policy tree

About this task

Log in 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.

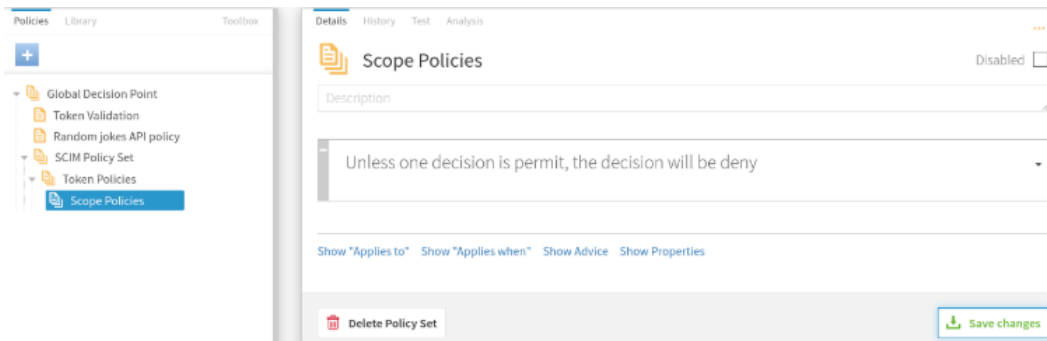
Steps

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, go 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**.
 - a. Click the **+** icon to add 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.

a. Click **Toolbox**.

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. 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.

13. 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

Applies to 1 definition

! retrieve ✕

Drag and Drop Targets or Targetable Definitions from Toolbox

Condition

ALL ANY NONE

A C

A A

→Advice (1 in total)

Name Obligatory ⋮ ◀

or

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

or

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 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 "Permit" and applies to 1 definition, which is the "retrieve" action.

The condition is defined with two rows:

Operator	Field	Comparison	Value
A	HttpRequest.AccessToken.scope	Contains	profile
A	HttpRequest.AccessToken.token_	Equals	HttpRequest.ResourcePath

The advice section shows one advice named "Include profile attributes", which is obligatory. Its description is "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 is "include-attributes", it applies to "Permit", and the payload is ["uid", "sn", "givenName", "description"].

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, select `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

The screenshot displays the configuration for a rule named "Scope: scimAdmin (retrieve)".

- Description:** A text input field.
- Effect:** A dropdown menu set to "Permit".
- Applies to 1 definition:** A box containing a warning icon and the text "retrieve".
- Condition:** A group box containing:
 - A dropdown menu set to "A", a text box containing "HttpRequest.AccessToken.scope", a dropdown menu set to "Contains", and a dropdown menu set to "C", followed by a text box containing "scimAdmin".
 - Buttons: "+ Comparison", "+ Named Condition", "+ Group", and "Drag and Drop any Trust Framework item from Toolbox".
- Advice (1 in total):** A box containing:
 - A dropdown menu set to "Name", a text box containing "Include all attributes", and a dropdown menu set to "Obligatory" with a checked checkbox.
 - Buttons: "+ Create new Advice" and "Drag and Drop Advice from Toolbox".

At the bottom, there are links: "Hide 'Applies to'", "Hide Advice", and "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, select `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. At the top, the scope is set to 'scimAdmin (create/modify)'. The rule's effect is 'Permit', and it applies to two definitions: 'create' and 'modify'. The condition is configured as 'HttpRequest.AccessToken.scope' contains 'scimAdmin'. The rule name is 'Allow certain attributes to be created or updated', and it is marked as 'Obligatory'. The payload is set to a list of attributes: ["manager", "uid", "mail", "sn", "givenName", "cn", "description", "l", "st", "country", "postalAddress", "mobile", "homePhone"].

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, select `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, select `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, select `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 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, select `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-https-dg-host>:<your-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 `[]`.
A 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

 **employeeType**

Parent: TokenOwner

Description:

Resolver Settings

Resolver Type: Attribute | TokenOwner + Add Condition

+ Add Resolver

Cache Settings

Cache Strategy: No Caching

Value Settings

Processor: JSON Path | `employeeType`

Default value: `[]`


Type: Collection | Secret

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, select `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.

Install PingDataGovernance Server

As you plan your PingDataGovernance deployment, review the components to install as well as the potential deployment architectures and environments.

Components

- Policy Administration GUI – Powered by Symphonic®, the PingDataGovernance Policy Administration GUI gives policy administrators the ability to develop and test data-access policies.
- PingDataGovernance Server – Enforces policies to control fine-grained access to data. REST APIs access data through PingDataGovernance Server, which applies the data-access policies to allow, block, filter, or modify data resources and data attributes.

Deployment architectures

PingDataGovernance Server supports the following options of deployment architectures for enforcing fine-grained access to data:

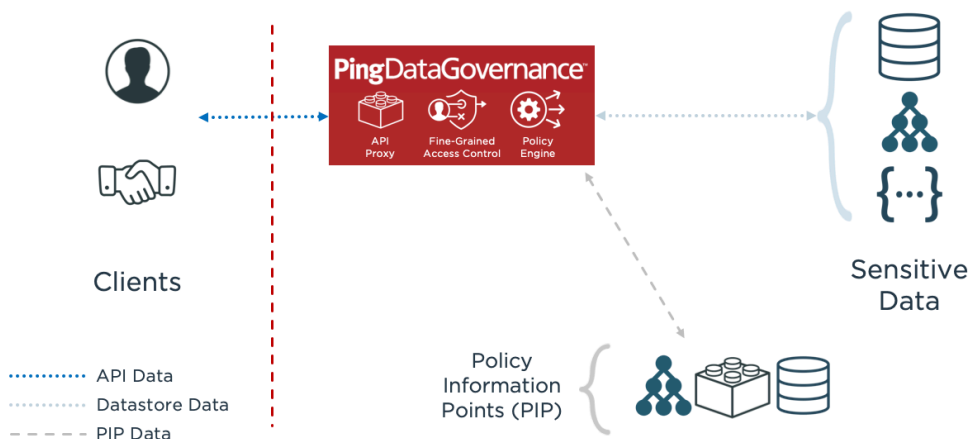
- SCIM API to datastores
- API Security Gateway as reverse proxy

- API Security Gateway in Sideband configuration

The following sections describe these deployment architectures in more detail.

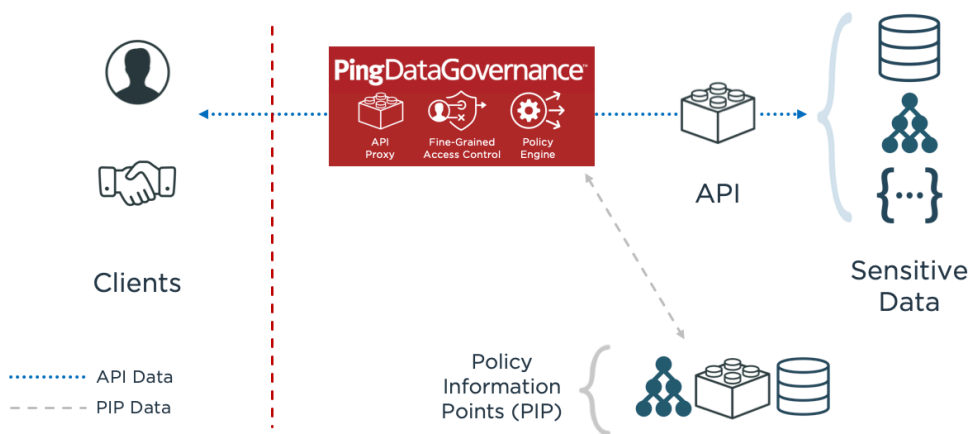
SCIM API to datastores

PingDataGovernance Server SCIM service provides a REST API for data that is stored in one or more external datastores, based on the SCIM 2.0 standard. Policy is enforced by the SCIM service.



API Security Gateway as reverse proxy

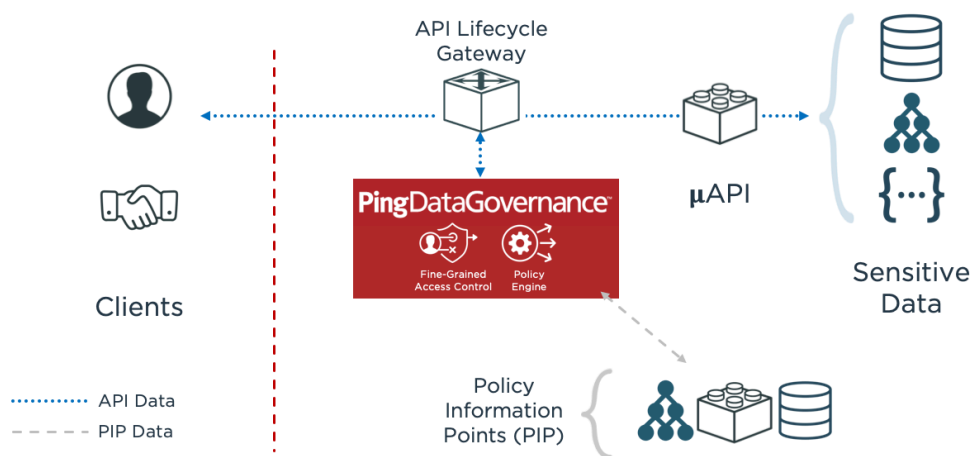
PingDataGovernance Server's API security gateway can be deployed as a reverse proxy to an existing JSON-based REST API. In this configuration, PingDataGovernance Server acts as an intermediary between clients and existing API services. Policy is enforced by the API security gateway.



API Security Gateway in Sideband configuration

PingDataGovernance Server's API security gateway can be deployed as an extension to an existing API Lifecycle Management Gateway, which is commonly known as a *sideband* configuration. In this configuration, the API Lifecycle Management Gateway functions as the intermediary between clients and

existing API services. However, API request and response data still flows through PingDataGovernance Server to enforce policy.



Deployment environments

PingDataGovernance Server can be deployed in either of the following environments:

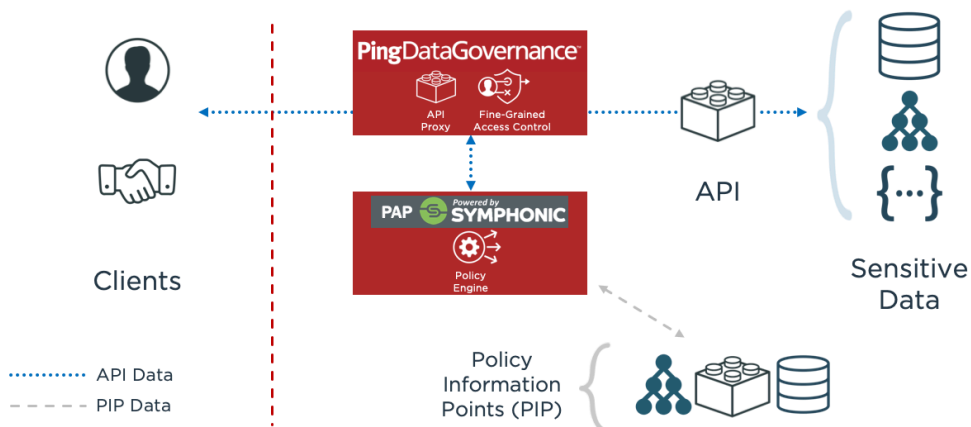
- Development environment – PingDataGovernance Server and the Policy Administration GUI are used together during the development of policies.
- Other pre-production and production environments – After policies are developed, they are tested in other pre-production environments and eventually put into production.

The following sections describe these deployment environments in more detail.

Development environment

To allow teams to test data-access policies during their development, PingDataGovernance Server is configured to obtain policy decisions from the Policy Administration GUI. The development environment supports all deployment architectures. In this configuration, the Policy Decision Service is set to External mode.

The following image shows PingDataGovernance Server configured in the Reverse Proxy architecture.

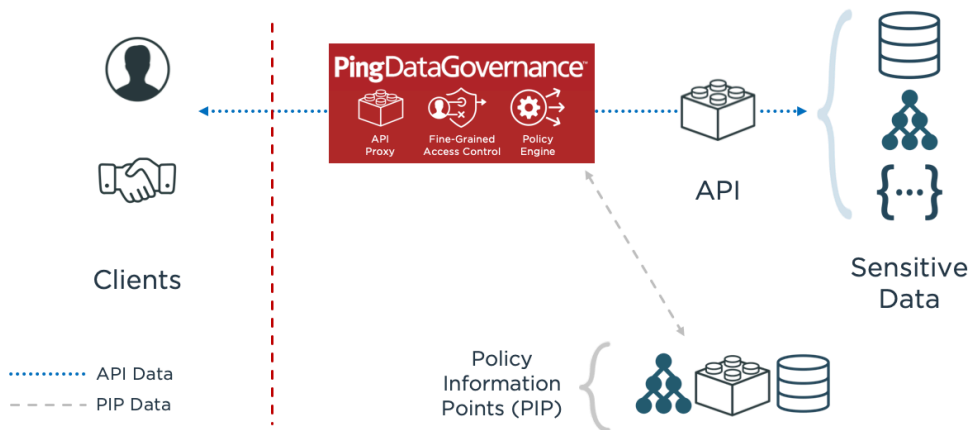


As test API requests are proxied through PingDataGovernance Server's API security gateway, policy decisions are obtained from the Policy Administration GUI, and are enforced by the API security gateway.

Other pre-production and production environments

The Policy Administration GUI is not a part of so-called *higher* environments. Instead, policy is exported from the Policy Administration GUI, and is imported into PingDataGovernance Server.

In the following configuration, the Policy Decision Service is set to Embedded mode.



Before you begin

The following components are required to install PingDataGovernance Server:

- Supported Linux, Windows, or Docker platform
- Valid license key
- Java

The following sections describe these prerequisites in more detail.

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 8
- Red Hat Enterprise Linux ES 7.6
- Red Hat Enterprise Linux ES 7.5
- CentOS 7.6
- CentOS 7.5
- 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

Note: The [Ping Identity Java Support Policy](#) applies to your Java Runtime Environment.

Browsers

Administration Console

- Chrome
- Firefox
- Internet Explorer 11 and later

About license keys

License keys are required to install all Ping products. To obtain a license key, contact your account representative or use the [Ping Identity licensing portal](#).

A license is required for setting up a new single server instance and can be used site-wide for all servers in an environment. Additionally, a new license must be obtained when updating a server to a new major version, such as when upgrading from 7.3 to 8.0.

Note: A prompt for a new license is displayed during the update process.

Each license expires on a particular date. Before a license expires, obtain a new one and install it by using `dsconfig` or the Admin Console.

Note: The server provides a notification as the expiration date approaches.

To view the details of a license, use the server's `status` tool.

Installing Java

About this task

PingDataGovernance Server requires Java for 64-bit architectures. Even if Java is already installed on your system, we recommend that you create a separate Java installation for PingDataGovernance Server. This setup ensures that updates to the system-wide Java installation do not inadvertently impact PingDataGovernance Server.

Note: This setup requires that you install the JDK, rather than the JRE.

Steps

1. Download and install a JDK.
2. Set the `JAVA_HOME` environment variable to the Java installation directory path.
3. Add the `bin` directory to the `PATH` environment variable.

Preparing a Linux environment

About this task

This section describes the following tasks, which must be completed before you install PingDataGovernance Server in a Linux environment:

Steps

1. Setting the file descriptor limit.
2. Setting the maximum user processes.
3. Disabling file system swapping.
4. Managing system entropy.
5. Enabling the server to listen on privileged ports.

Set the file descriptor limit

About this task

By default, PingDataGovernance Server allows for an unlimited number of connections. However, the server is restricted by the file descriptor limit on the operating system. This topic describes how to increase the file descriptor limit on the operating system.

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

Steps

1. Display the current hard limit of the system, as follows:

```
ulimit -aH
```

The *hard limit* is the maximum server limit that can be set without tuning the kernel parameters in the `proc` file system.

2. Edit the `/etc/sysctl.conf` file.

If the `fs.file-max` property is defined in the file, verify that its value is set to at least 65535. If this property does not exist, add the following line to the end of the file:

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

3. Edit the `/etc/security/limits.conf` file.

If the file contains lines that set the soft and hard limits for the number of file descriptors, verify that the values are set to 65535. If the properties are absent, add the following lines to the end of the file (before #End of file), making certain to insert a tab between the columns:

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

4. Restart the server.
5. Use the `ulimit` command to verify that the file descriptor limit is set to 65535, as follows:

```
ulimit -n
```

Results

After the operating system limit is set, use one of the following methods to configure the number of file descriptors that the server uses:

- Use a `NUM_FILE_DESCRIPTOR` environment variable.
- Create a `config/num-file-descriptors` file with a single line, such as `NUM_FILE_DESCRIPTOR=12345`.

If these values are not set, the default value of 65535 is used.

Note: This optional step ensures that the server shuts down safely before it reaches the file descriptor limit.

Next steps

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

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

Add or edit the following lines if they do not already exist:

```
* soft nproc 65536
* soft nofile 65536
* hard nproc 65536
* hard nofile 65536
root soft nproc unlimited
```

Set the maximum user processes

About this task

On some Linux distributions, such as RedHat Enterprise Linux Server/CentOS 6.0 or later, the default maximum number of user processes is set to 1024, which is considerably lower than the same parameter on earlier distributions, such as RHEL/CentOS 5.x. The default value of 1024 leads to some JVM memory errors when running multiple servers on a machine, due to each Linux thread being counted as a user process.

At startup, PingDataGovernance Server attempts to raise this limit to 16383 if the value reported by `ulimit` is less than that number. If the value cannot be set, an error message is displayed. In such a scenario, explicitly set the limit in `/etc/security/limit.conf`, as the following example shows.

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

The 16383 value can also be set in the `NUM_USER_PROCESSES` environment variable, or by setting the same variable in `config/num-user-processes`.

Disable file system swapping

About this task


Disable all performance-tuning services, like `tuned`. If performance tuning is required, perform the following steps to set `vm.swappiness`:

Steps

1. Clone the existing performance profile.
2. Add `vm.swappiness = 0` to the new profile's `tuned.conf` file in `/usr/lib/tuned/`
`profilename/tuned.conf`.
3. Select the updated profile by running `tuned-adm profile customized_profile`.

Manage 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.

Enable the server to listen on privileged ports

About this task

Linux systems provide *capabilities* that grant specific commands the ability to complete tasks that are normally permitted only by a root account. Instead of granting an ability to a specific user, capabilities are granted to a specific command. For the sake of convenience, you might want to enable the server to listen on privileged ports while running as a non-root user.

Use the `setcap` command to assign capabilities to an application. The `cap_net_bind_service` capability enables a service to bind a socket to privileged ports, which are defined as ports with numbers less than 1024. If Java is installed in `/ds/java`, and if the Java command to run the server is `/ds/java/bin/java`, then the Java binary can be granted the `cap_net_bind_service` capability by using the following command:

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

The Java binary requires an additional shared library, `libjli.so`, as part of the Java installation. Because additional limitations are imposed on where the operating system looks for shared libraries to load for commands with assigned capabilities, you must create the file `/etc/ld.so.conf.d/libjli.conf` with the path to the directory that contains the `libjli.so` file. This step informs the operating system where to look for the library. For example, if the Java installation is located in `/ds/java`, the contents must be as follows:

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

Run the following command for the change to take effect:

```
$ sudo ldconfig -v
```

Installing PingDataGovernance Server

About this task

The following options are available when installing PingDataGovernance Server:

- Install PingDataGovernance Server manually
- Use server profiles to install PingDataGovernance Server in an automated environment
- Use Docker to install PingDataGovernance Server

The following sections describe these installation options in more detail.

Installing PingDataGovernance Server manually

About this task

After you prepare your hardware and software system based on the preceding instructions, you can start the setup process for PingDataGovernance Server, which consists of the following steps:

Steps

1. Obtaining the PingDataGovernance Server installation packages.
2. Installing PingDataGovernance Server.
3. Installing the PingDataGovernance Policy Administration GUI.
4. Performing additional configuration steps.

The following sections describe these installation and configuration steps in more detail.

Obtaining the installation packages

About this task

The PingDataGovernance distribution consists of two compressed files, one for each of the following server components:

- PingDataGovernance Server
- PingDataGovernance Policy Administration GUI

To start the installation process, obtain the latest compressed release bundles from Ping Identity and expand them into folders of your choice.

Installing the server

About this task

To unpack the build distribution

The PingDataGovernance release bundle contains the PingDataGovernance Server code, tools, and package documentation.

Steps

1. Download the latest compressed distribution of the PingDataGovernance Server software.
2. Unzip the compressed zip archive to a directory of your choice.

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

Results

You can now set up PingDataGovernance Server.

About the layout of the PingDataGovernance Server folders

After you extract the contents of the PingDataGovernance Server distribution file, you can access the folders and command-line utilities that the following table describes.

Directories, files, and tools	Description
README	README file that describes the steps to set up and start PingDataGovernance Server.
bak	Stores the physical backup files used with the backup command-line tool.

Directories, files, and tools	Description
bat	Stores Windows-based command-line tools for PingDataGovernance Server.
bin	Stores UNIX/Linux-based command-line tools for PingDataGovernance Server.
build-info.txt	Contains build and version information for PingDataGovernance Server.
collector	Used by the server to make monitored statistics available to PingDataMetrics Server.
config	Stores the configuration files for the backends (admin, config) as well as the directories for messages, schema, tools, and updates.
docs	Provides the product documentation.
legal	Stores any legal notices for dependent software used with PingDataGovernance Server.
lib	Stores any scripts, jar, and library files needed for the server and its extensions.
locks	Stores any lock files in the backends.
logs	Stores log files for PingDataGovernance Server.
metrics	Stores the metrics that can be gathered for this server and surfaced in PingDataMetrics Server.
resource	Stores supporting files such as default policies, a sample server profile template, and MIB files for SNMP.
revert-update	The revert-update tool for UNIX/Linux systems.
revert-update.bat	The revert-update tool for Windows systems.
setup	The setup tool for UNIX/Linux systems.
setup.bat	The setup tool for Windows systems.
uninstall	The uninstall tool for UNIX/Linux systems.
uninstall.bat	The uninstall tool for Windows systems.
update	The update tool for UNIX/Linux systems.
update.bat	The update tool for Windows systems.
velocity	Stores any customized Velocity templates and other artifacts (CSS, Javascript, images), or Velocity applications hosted by the server.
webapps	Stores web application files such as the Administrative Console.

About the server installation modes

PingDataGovernance Server provides the following tools to help install and configure the system:

- The **setup** tool performs the initial tasks needed to start PingDataGovernance Server, including configuring JVM runtime settings and assigning listener ports for the PingDataGovernance Server's HTTP services.
- The **create-initial-config** tool configures connectivity between a SCIM 2 user store and PingDataGovernance Server. During the process, the **prepare-external-store** tool prepares each PingDirectory Server to serve as a user store by PingDataGovernance Server. Configuration can be written to a file to use for additional installations.
- After the initial setup is finished, you can use the **dsconfig** tool and the Administrative Console to perform additional configuration.

To install a server instance, run the **setup** tool in one of the following modes:

- Interactive Command-Line mode – Prompts for information during the installation process. To run the installation in this mode, use the **setup --cli** command.
- Non-Interactive Command-Line mode – Designed for setup scripts to automate installations or for command-line usage. To run the installation in this mode, setup must be run with the **--no-prompt** option as well as the other arguments required to define the appropriate initial configuration.

Perform all installation and configuration steps while logged on to the system as the user or the role under which PingDataGovernance Server will run.

Install the server interactively

About this task

- The **setup** tool prompts you interactively for the information that it needs. Be prepared to provide the following information:
- The location of a valid license file.
- The name and password for an administrative account, which is also called the *root user DN*.
- An available port for PingDataGovernance Server to accept HTTPS requests.
- An available LDAPS port for PingDataGovernance Server to accept administrative requests.
- Information related to the server's connection security, including the location of a keystore that contains the server certificate, the nickname of that server certificate, and the location of a truststore.
- The amount of memory to reserve for usage by the JVM.
- A unique instance name for the server.

Steps

1. Run the **setup** command.

```
$ ./setup
```

2. To start and stop PingDataGovernance Server, use the **start-server** and **stop-server** commands, respectively.

For additional options, see [Starting PingDataGovernance Server](#) on page 91.

Log in to the Administrative Console

About this task

After the server is installed, access the Administrative Console at `https://<host>/console/login` to verify the configuration and to manage the server. To log in to the Administrative Console, use the initial root user DN specified during setup (by default, `cn=Directory Manager`).

Installing PingDataGovernance Policy Administration GUI

About this task

To unpack the build distribution

The PingDataGovernance Policy Administration GUI release bundle contains the PingDataGovernance Policy Administration GUI code and tools.

Steps

1. Download the latest compressed distribution of the PingDataGovernance Policy Administration GUI software.
2. Extract the compressed archive to a directory of your choice.

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

Results

You can now set up PingDataGovernance Policy Administration GUI.

About the layout of the PingDataGovernance Policy Administration GUI folders

After you have extract the contents of the PingDataGovernance Policy Administration GUI distribution file, you can access the folders and command-line utilities that the following table describes.

Directories, files, and tools	Description
admin-point-application	Stores any jar and library files needed for the server.
bin	Stores UNIX/Linux-based command-line tools for the PingDataGovernance Policy Administration GUI.
build-info.txt	Contains build and version information for the PingDataGovernance Policy Administration GUI.
config	Stores the configuration, including the keystore for the web server HTTPS certificate.
lib	Stores any jar and library files needed by the command-line tools.
logs	Stores log files for the PingDataGovernance Policy Administration GUI.
resource	Stores supporting files such as policy snapshots.

Install the PingDataGovernance Policy Administration GUI interactively

About this task

The **setup** tool prompts you interactively prompt for the information that it needs. Be prepared to provide the following information:

- The location of a valid license file.
- An available port for the PingDataGovernance Policy Administration GUI to accept HTTPS requests.

Additionally, you must choose one of the two following authentication modes for the PingDataGovernance Policy Administration GUI:

- Demo mode – Configures the PingDataGovernance Policy Administration GUI to use form-based authentication with a fixed set of credentials. Unlike OIDC mode, this mode does not require an external authentication server. However, it is inherently insecure and is recommended only for demonstration purposes.
- OpenID Connect (OIDC) mode – configures the PingDataGovernance Policy Administration GUI to delegate authentication and sign-on services to an OpenID Connect provider, such as PingFederate.

If you choose OIDC mode, be prepared to provide the following additional information:

- The host name and port of an OpenID Connect provider.
- Information related to the server's connection security, including the location of a keystore that contains the server certificate, the nickname of that server certificate, and the location of a truststore.

Steps

1. Run the `setup` command.
2. To start and stop the PingDataGovernance Policy Administration GUI, use the `start-server` and `stop-server` commands, respectively.

Log in to the PingDataGovernance Policy Administration GUI

After completing setup for demo mode, you can log in immediately to the PingDataGovernance Policy Administration GUI by going to the following URL in a web browser:

```
https://<host>:<port>
```

where you substitute the host name and port that you specified during setup.

Use the following demo credentials to log in to the PingDataGovernance Policy Administration GUI:

- User name: `admin`
- Password: `password123`

If you set up the PingDataGovernance Policy Administration GUI to use OIDC mode, you must also configure an OpenID Connect provider, see [Configure Authentication Server for OpenID Connect single sign-on](#) on page 82

To log in to the PingDataGovernance Policy Administration GUI, go to the following URL in a web browser:

```
https://<host>:<port>
```

where you substitute the host name and port that you specified during setup.

The GUI prompts you to proceed to the OpenID Connect provider to log in. After OpenID Connect authentication is complete, you are redirected back to the PingDataGovernance Policy Administration GUI.

Change the Policy Administration GUI authentication mode

About this task

If you decide later to change the authentication mode that the PingDataGovernance Policy Administration GUI uses, re-run the `setup` tool and choose a different authentication mode. This action overwrites the PingDataGovernance Policy Administration GUI's existing configuration.

Steps

1. Stop the Policy Administration GUI.

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

2. Run the setup command.

```
$ bin/setup
```

3. Start the Policy Administration GUI.

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

*Configure Authentication Server for OpenID Connect single sign-on***About this task**

If you chose OIDC mode when setting up the PingDataGovernance Policy Administration GUI, you need to configure an OpenID Connect provider, such as PingFederate, to accept sign-on requests from the PingDataGovernance Policy Administration GUI.

Create an OAuth 2 client

Use the following configuration to create an OAuth 2 client that represents the PingDataGovernance Policy Administration GUI.

OAuth 2 client configuration	Configuration value
Client ID	pingdatagovernance-pap
Redirect URI	https://<host>:<port>/idp-callback
Grant type	Implicit
Response type	token id_token
Scopes	<ul style="list-style-type: none"> ▪ openid ▪ email ▪ profile

Additionally, access tokens and ID tokens issued for this client must be configured to include the following claims:

- sub
- name
- email

Allow a CORS origin

Configure the OpenID Connect provider to accept a CORS origin that matches the PingDataGovernance Policy Administration GUI's scheme, public host, and port, such as `https://<host>:<port>`.

Authorizing logon attempts

Configure the OpenID Connect provider to issue tokens to the PingDataGovernance Policy Administration GUI only when the authenticated user is authorized to administer policies according to your organization's access rules.

For PingFederate, this level of authorization is controlled by using issuance criteria. For more information, refer to the PingFederate documentation.

Additional configuration steps**About this task**

After the components are installed, complete the following configuration tasks:

Steps

1. Configure the Policy Decision service.
2. Configure a user store.
3. Configure Access Token Validation

The following sections describe these configuration tasks in more detail.

Configure the Policy Decision Service

About this task

Configure the Policy Decision Service before policies are enforced on data access. For development environments in which policy administrators will be building and testing policies, configure the Policy Decision Service to External mode. For other pre-production and production environments in which policies will be tested and deployed, configure the Policy Decision Service for Embedded mode.

For information about configuring the Policy Decision Service, see [Policy administration](#) on page 132.

Configure a user store

About this task

PingDataGovernance Server uses 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 Server assumes that PingDirectory Server is the default user store, other LDAPv3-compliant directories are also supported.

prepare-external-store

When using PingDirectory Server as the user store, first prepare the server by running **prepare-external-store**. This tool completes the following tasks:

- Creates the PingDataGovernance Server user account on your instance of PingDirectory Server.
- Sets the correct password.
- Configures the account with the required privileges.
- Installs the schema that PingDataGovernance Server requires.

create-initial-config

The **create-initial-config** command configures connectivity between PingDataGovernance Server and the user store. It also creates a SCIM resource type through which PingDataGovernance Server obtains the user attributes.

Although the **create-initial-config** command is recommended for first-time installers, its use is optional. If you elect not to use **create-initial-config**, you must configure the following objects:

- Store adapter
- SCIM resource type
- SCIM schema (optional)

For more information about configuring SCIM, see [About the SCIM service](#) on page 115.

Configure Access Token Validation

About this task

Clients authenticate themselves to HTTP APIs and the SCIM service by using OAuth2 bearer token authentication. PingDataGovernance Server uses Access Token Validators to translate and decode a bearer token to a set of attributes that it represents.

For user-authorized bearer tokens, Access Token Validators are required to map the subject of the access token to the user in the user store, to evaluate the user's attributes as part of policy.

For more information about configuring Access Token Validation, see [Access token validators](#) on page 146.

Next steps

About this task

After the components are installed and configured, start developing policies for enforcing fine-grained access to data. Consider performing the following steps:

Steps

1. Log in to the Admin Console to configure endpoints for existing JSON APIs.
For more information, see [About the API security gateway](#) on page 94.
2. Log in to the Admin Console to define SCIM APIs for data in databases
For more information, see [About the SCIM service](#) on page 115.
3. Log in to the PingDataGovernance Policy Administration GUI to create policy.
For more information, refer to the *PingDataGovernance Policy Administration Guide*.

Using server profiles to install PingDataGovernance Server

About this task

Organizations are adopting DevOps practices to reduce risk while providing quicker time-to-value for the services that they provide to their business and customers. Examples of such practices that are central to DevOps include automation and Infrastructure-as-Code (IaC). Organizations that combine these principles can manage the following infrastructure and service operations in the same manner as preparing application code for general release:

- Appropriate versioning
- Continuous integration
- Quality control
- Release cycles

Server profiles enable organizations to adopt these DevOps practices more easily.

Administrators can export the configuration of a PingDataGovernance Server instance to a directory of mostly text files called a *server profile*. Administrators can also track changes to these files in a version-control system like Git, and can install new instances of PingDataGovernance Server (or update existing instances of PingDataGovernance Server) from a server profile.

The scripts and other files in the `server-profile` directory are declarative of the desired state of the environment. Consequently, the definitions in the `server-profile` directory directly influence the servers. No one needs to identify a server's current configuration and compute the differences that must be applied to attain the appropriate end state.

The primary goal of a server profile is to simplify the deployment of PingDataGovernance Server by using deployment automation frameworks. By using server profiles, the amount of scripting that is required across automation frameworks – like Docker, Kubernetes, and Ansible – is reduced considerably.

As a declarative form of a full server configuration, a server profile provides the following advantages:

- Provides a more complete and easily comparable method of defining the configuration of an individual server. Changes between different servers are easier to review and understand, and incremental changes to a server's configuration are easier to track.
- Ensures that each server instance is configured identically to its peers.
- Applies to installing new instances as well as updating the configuration of previously installed instances.

- Shares a common configuration across a deployment environment of development, test, and production without unnecessary duplication and error-prone, environment-specific modifications. For more information about substituting variables that differ by environment, see [Variable substitution](#) on page 85.
- Reduces the number of additional configuration steps that are required to place a server into production.
- Makes the execution of various configuration changes more consistent and repeatable. The strategy of using a server profile to represent the final state of a server is less error-prone than recording a step-by-step process to attain that state.
- Can be managed easily in a version-control system.
- Simplifies the management of servers outside deployment-automation frameworks.

Variable substitution

The `manage-profile` tool uses the format `${VARIABLE}` to support the substitution of variables in profiles. This format can be escaped by using another `$`. For example, after substitution, `$$${VARIABLE}` becomes `${VARIABLE}`.

Variable values can be read from a profile variables file or from environment variable values. If both options are used, the values that are specified in the file overwrite any environment variables.

The following code provides an example profile variables file.

```
MY_VARIABLE=value
MY_OTHER_VARIABLE=anothervalue
```

In this example, the tool provides the `PING_SERVER_ROOT` and `PING_PROFILE_ROOT` variables. The following table describes the built-in variables.

Built-in variable	Description
<code>PING_SERVER_ROOT</code>	Evaluates to the absolute path of the server's root directory.
<code>PING_PROFILE_ROOT</code>	Evaluates to the individual profile's root directory.

For more information about the tool's usage, run the command `bin/manage-profile --help`.

Layout of a server profile

Use either of the following methods to create a server profile:

- Extract the template named `server-profile-template-dg.zip`, which is located in the `resource` directory.
- Run the `manage-profile generate-profile` subcommand. The `manage-profile` tool references the file system directory structure.

Files can be added to each directory as needed.

The following hierarchy represents the file structure of a basic server profile.

```
-server-profile/
  |-- dsconfig/
  |-- misc-files/
  |-- server-root/
  | |-- post-setup/
  | /-- pre-setup/
  |-- server-sdk-extensions/
  |-- setup-arguments.txt
```

setup-arguments.txt

When creating a profile, the first step is to add arguments to the file `setup-arguments.txt`. When **manage-profile** setup is run, these arguments are passed to the server's setup tool. To view the arguments that are available in this file, run the server's **setup --help** command.

To provide the equivalent, non-interactive CLI arguments after any prompts have been completed, run **setup** interactively. The `setup-arguments.txt` file in the profile template contains an example set of arguments that can be changed.

`setup-arguments.txt` is the only required file in the profile.

dsconfig/

dsconfig batch files can be added to the `dsconfig` directory. These files, each of which must include a `.dsconfig` extension, contain **dsconfig** commands to apply to server.

Because the **dsconfig** batch files are ordered lexicographically, `00-base.dsconfig` runs before `01-second.dsconfig`, and so on.

To produce a `dsconfig` batch file that reproduces the current configuration, run **bin/config-diff**.

server-root/

Any server root files can be added to the `server-root` directory, including schema files, email template files, custom password dictionaries, and other files that must be present on the final server root. Add these files to the `server-root/pre-setup` or `server-root/post-setup` directory, depending on when they need to be copied to the server root. Most server root files are added to the `server-root/pre-setup` directory.

server-sdk-extensions/

Add server SDK extension ZIP files to the `server-sdk-extensions` directory. Use the **manage-extension** tool to install them, and include any configuration that is necessary for the extensions in the profile's **dsconfig** batch files.

variables-ignore.txt

The `variables-ignore.txt` file is an optional component of the server profile. It is useful when adding bash scripts to the server root because such files often contain expressions that the **manage-profile** tool normally interprets as variables.

Add `variables-ignore.txt` to a profile's root directory to indicate the relative paths of any files that are not to have their variables substituted.

The following example shows the contents of a typical `variables-ignore.txt` file:

```
server-root/pre-setup/script-to-ignore.sh
server-root/post-setup/another-file-to-ignore.txt
```

server-root/permissions.properties

The `permissions.properties` file, located in the `server-root` directory, is an optional file that specifies the permissions to apply to files that are copied to the server root. These permissions are represented in octal notation. By default, server root files maintain their permissions when copied.

The following example shows the contents of a typical `permissions.properties` file:

```
default=700
file-with-special-permissions.txt=600
new-subdirectory/file-with-special-permissions.txt=644
bin/example-script.sh=760
```

misc-files/

Additional documentation and other files can be added to the `misc-files` directory, which the `manage-profile` tool does not use. Use the variable `PING_PROFILE_ROOT` to refer to files in this directory from other locations, such as `setup-arguments.txt`.

For example, a password file named `password.txt` in the `misc-files` directory could be referenced with `${PING_PROFILE_ROOT}/misc-files/password.txt` in `setup-arguments.txt`. Use a reference like this example to supply the file for the `--rootUserPasswordFile` argument in `setup-arguments.txt`.

Workflows

This section describes how to use the `manage-profile` tool to accomplish typical server-management tasks, like the following examples:

- [Creating a server profile](#) on page 87.
- [Installing a new environment](#) on page 88.
- [Scaling up your environment](#) on page 89.
- [Rolling out an update](#) on page 89.

The following sections describe these tasks in more detail. For more information about the `manage-profile` tool, run `manage-profile --help`. For more information about each individual subcommand and its options, run `manage-profile <subcommand> --help`.

Creating a server profile

About this task

To create a server profile from a configured server, use the `generate-profile` subcommand.

Steps

1. Create a profile directory.

```
$ mkdir -p /opt/server-profiles/dg
```

2. Run `generate-profile`.

```
$ bin/manage-profile generate-profile --profileRoot /opt/server-profiles/dg
```

3. Customize the resulting profile to suit your needs and to remove deployment environment-specific values.

- Specify a consistent location for the license key file:
 - a. Copy the license key file to the server profile's `misc-files` directory.

```
$ cp PingDataGovernance.lic /opt/server-profiles/dg/misc-files/
```

- b. Open the `setup-arguments.txt` file in a standard text editor.
- c. Locate the `--licenseKeyFile` argument.
- d. Change the value of `--licenseKeyFile` to the following value:

```
${PING_PROFILE_ROOT}/misc-files/PingDataGovernance.lic
```

- e. Save your changes.
- Remove deployment environment-specific values and replace them with variables. For example, to refer to a different PingFederate server in your development environments versus your test environments, perform the following steps:
 - a. Open the `/opt/server-profiles/dg/dsconfig/00-config.dsconfig` file in a standard text editor.
 - b. Locate the value specified for `base-url` for the external server that identifies your PingFederate server.
 - c. Replace the value with a variable, like `${PF_BASE_URL}`.
 - d. Save your changes.
 - e. Create or update a server profile variables file for your development environment.
 - f. Add a row like the following example to the variables file:

```
PF_BASE_URL=https://sso.dev.example.com:9031
```

- g. Save your changes.
- h. Continue replacing deployment environment-specific values with variables until the server profile contains no more deployment environment-specific values.

At this point, the server profile can be checked into a version-control system, like Git, shared with your team, and integrated into your deployment automation.

Installing a new environment

Before you begin

The steps in this section make the following assumptions:

- A server profile has already been created at the path `~/git/server-profiles/dg`
- Your development environment's variables file is saved at the path `~/dg-variables-dev.env`

About this task

After you create and customize a server profile, use the `manage-profile setup` subcommand to set up new server instances and additional deployment environments.

The `setup` subcommand completes the following tasks:

- Copies the server root files
- Runs the `setup` tool
- Runs the `dsconfig` batch files
- Installs the server SDK extensions

- Sets the server's cluster name to a unique value

Note: Cluster-wide configuration is automatically mirrored across all servers in the topology with the same cluster name. In a DevOps deployment with immutable servers, configuration mirroring introduces risk. Therefore, in most cases, cluster names should be unique for each server to avoid configuration mirroring.

Steps

1. Extract the contents of the compressed archive to a directory of your choice.

```
$ mkdir /opt/dg
$ cd /opt/dg
$ unzip PingDataGovernance-<version>.zip
```

2. Change directories.

```
$ cd PingDataGovernance
```

3. Run **setup**.

```
$ bin/manage-profile setup \
  --profile ~/git/server-profiles/dg \
  --profileVariablesFile ~/dg-variables-dev.env
```

Scaling up your environment

About this task

The automation for this task is identical to the previous task of installing a new server in a new environment. Because each instance of PingDataGovernance Server requires a unique instance name and host name, each instance must also be set up from a unique server profile variables file.

Rolling out an update

Before you begin

The steps in this section make the following assumptions:

- A server profile has been created at the path `~/git/server-profiles/dg`
- The server's server profile variables file is saved at the path `/opt/dg/dg-variables.env`
- The existing server with the earlier configuration is installed at `/opt/dg/PingDataGovernance`

About this task

Run the **replace-profile** subcommand on a server that was originally set up with a server profile to replace its configuration with a new profile. The **replace-profile** subcommand applies a specified server profile to an existing server while also preserving its configuration.

While **manage-profile replace-profile** is running, the existing server is stopped and moved to a temporary directory that the `--tempServerDirectory` argument specifies. A fresh, new server is subsequently installed and set up with the new profile. The final server is left running if it was running before the command was started, and remains stopped if it was stopped.

If files have been added or modified in the server root since the most recent **manage-profile setup** or **manage-profile replace-profile** was run, they are included in the final server with the replaced profile. Otherwise, files added specifically from the `server-root` directory of the previous server profile are absent from the final server with the replaced profile.

If errors occur while running the subcommand, such as the new profile having an invalid `setup-arguments.txt` file, the existing server returns to its original state from before **manage-profile replace-profile** was run.

Steps

1. Extract the same or a new version of PingDataGovernance Server to a location outside the existing server's installation.

```
$ mkdir ~/stage
$ cd ~/stage
$ unzip PingDataGovernance-<version>.zip
```

2. Change directories.

The **replace-profile** subcommand must be run from the location of the distribution package, not from the location of the existing server.

```
$ cd PingDataGovernance
```

3. Run **replace-profile**.

```
$ bin/manage-profile replace-profile \
  --serverRoot /opt/dg/PingDataGovernance \
  --profile ~/git/server-profiles/dg \
  --profileVariablesFile ~/dg-variables-dev.env
```

Using Docker to install PingDataGovernance Server

About this task

Docker images for Ping Identity's on-premises server products, including PingDataGovernance Server, are available from the Docker Hub repository at the following URL:

```
https://hub.docker.com/u/pingidentity/
```

Clustering and scaling

Because PingDataGovernance Servers are stateless, they do not require intra-cluster communication to scale. Instead, similarly configured independent server instances can be added behind the same network load balancer to achieve higher throughput while maintaining low latency.

Automated environments

To maintain identically configured PingDataGovernance Server instances behind your load balancer, we recommend that you use DevOps principles of Infrastructure-as-Code (IaC) and Automation. For information about using server profiles to scale upward by installing a new, identically configured instance of PingDataGovernance Server, see [Using server profiles to install PingDataGovernance Server](#) on page 84.

Non-automated environments

For customers without infrastructure and configuration automation, PingDataGovernance supports intra-cluster communication to maintain consistent configuration more easily among PingDataGovernance Server instances behind your network load balancer.

In this model, the server instances are joined into a topology configuration that automatically enables the grouping of servers as well as the mirroring of configuration changes. To mirror shared data across a topology, this model uses a master/slave architecture. All writes and updates are forwarded to the master, which forwards them to all other servers.

To learn more about this clustering model, contact Ping Professional Services.

Running PingDataGovernance Server

About this task

To start PingDataGovernance Server as a background process on a UNIX or Linux system, run `bin/start-server`. On Microsoft Windows systems, an analogous command is located in the `bat` folder.

To run PingDataGovernance Server as a foreground process, run `bin/start-server` with the `--nodetach` option.

Starting PingDataGovernance Server

About this task

To starting PingDataGovernance Server, use the `bin/start-server` command.

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

Running PingDataGovernance Server as a foreground process

About this task

To launch PingDataGovernance Server as a foreground process, run `bin/start-server` with the `--nodetach` option.

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

To stop PingDataGovernance Server, perform one of the following steps:

- In the terminal window in which the server is running, press and hold `CTRL+C`.
- Run `bin/stop-server` from another window.

Starting PingDataGovernance Server at boot time (Unix/Linux)

About this task

By default, PingDataGovernance Server does not start automatically when the system is booted. Instead, you must use the `bin/start-server` command to start it manually.

To configure PingDataGovernance Server to start automatically when the system boots, complete one of the following tasks:

- Use the `create-systemd-script` utility to create a script.
- Create a script manually.

Steps

1. Create the service unit configuration file in a temporary location.

```
$ bin/create-systemd-script \
  --outputFile /tmp/ping-data-governance.service \
  --userName dg
```

In this example, `dg` represents the user that PingDataGovernance Server runs as

2. As a root user, copy the `ping-data-governance.service` configuration file to the `/etc/systemd/system` directory.

3. Reload `systemd` to read the new configuration file.

```
$ systemctl daemon-reload
```

4. To start PingDataGovernance Server, use the `start` command.

```
$ systemctl start ping-directory.service
```

5. To configure PingDataGovernance Server to start automatically when the system boots, use the `enable` command.

```
$ systemctl enable ping-data-governance.service
```

6. Log off from the system as the root user.

Next steps

If you are working on an RC system, perform the following steps to complete this task:

1. Run `bin/create-rc-script` to create the startup script.
2. Move the script to the `/etc/init.d` directory.
3. Create symlinks to the script from the `/etc/rc3.d` directory.

To ensure that the server is started, begin the symlinks with an `S`.

4. Create symlinks to the script from the `/etc/rc0.d` directory.

To ensure that the server is stopped, begin the symlinks with a `K`.

Starting PingDataGovernance Server at boot time (Windows)

About this task

PingDataGovernance Server can run as a service on Windows Server operating systems. This approach allows the server to start at boot time, and allows the administrator to log off from the system without stopping the server.

Registering PingDataGovernance Server as a Windows service

About this task

Note: The following options are not supported when PingDataGovernance Server is registered to run as a Windows service:

- Command-line arguments for the `start-server.bat` and `stop-server.bat` scripts
- Using a task to stop the server

Steps

1. Run `bin/stop-server` to stop PingDataGovernance Server.
A server cannot be registered while it is running.
2. From a Windows command prompt, run `bat/register-windows-service.bat` to register the server as a service.
3. Use one of the following methods to start PingDataGovernance Server:
 - The **Windows Services Control Panel**
 - The `bat/ start-server.bat` command

Running multiple service instances

About this task

Only one instance of a particular service can run at a time. Services are distinguished by the `wrapper.name` property in the `<server-root>/config/wrapper-product.conf` file.

To run additional service instances, change the `wrapper.name` property on each additional instance. Service descriptions can also be added or changed in the `wrapper-product.conf` file.

Deregistering and uninstalling services

About this task

When a server is registered as a service, it cannot run as a non-service process or be uninstalled.

To remove the service from the Windows registry, use the `bat/deregister-windows-service.bat` file. The server can then be uninstalled by running the `uninstall.bat` script.

Log files for services

Log files are stored in `<server-root>/logs`, and file names begin with `windows-service-wrapper`. Log files are configured to rotate each time the wrapper starts or due to file size. Only the three most recent log files are retained.

These configurations can be edited in the `<server-root>/config/wrapper.conf` file.

Stopping PingDataGovernance Server

About this task

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

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

You can run `bin/stop-server` manually from the command line or within a script.

Restarting PingDataGovernance Server

About this task

To restart PingDataGovernance Server, use the `bin/stop-server` command with the `--restart` or `-R` option. Running this command is equivalent to shutting down PingDataGovernance Server, exiting the JVM session, and starting the server again.

Steps

1. Go to the PingDataGovernance Server root directory.
2. Run `bin/stop-server` with the `--restart` or `-R` option.

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

Uninstalling PingDataGovernance Server

About this task

PingDataGovernance Server provides an `uninstall` tool to remove its components from the system.

Steps

1. Go to the PingDataGovernance Server root directory.

2. Run the `uninstall` command.

```
$ ./uninstall
```

3. Select the option to remove all components, or select the components to be removed.

To remove selected components, enter `yes` when prompted.

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

4. Manually delete any remaining files or directories.

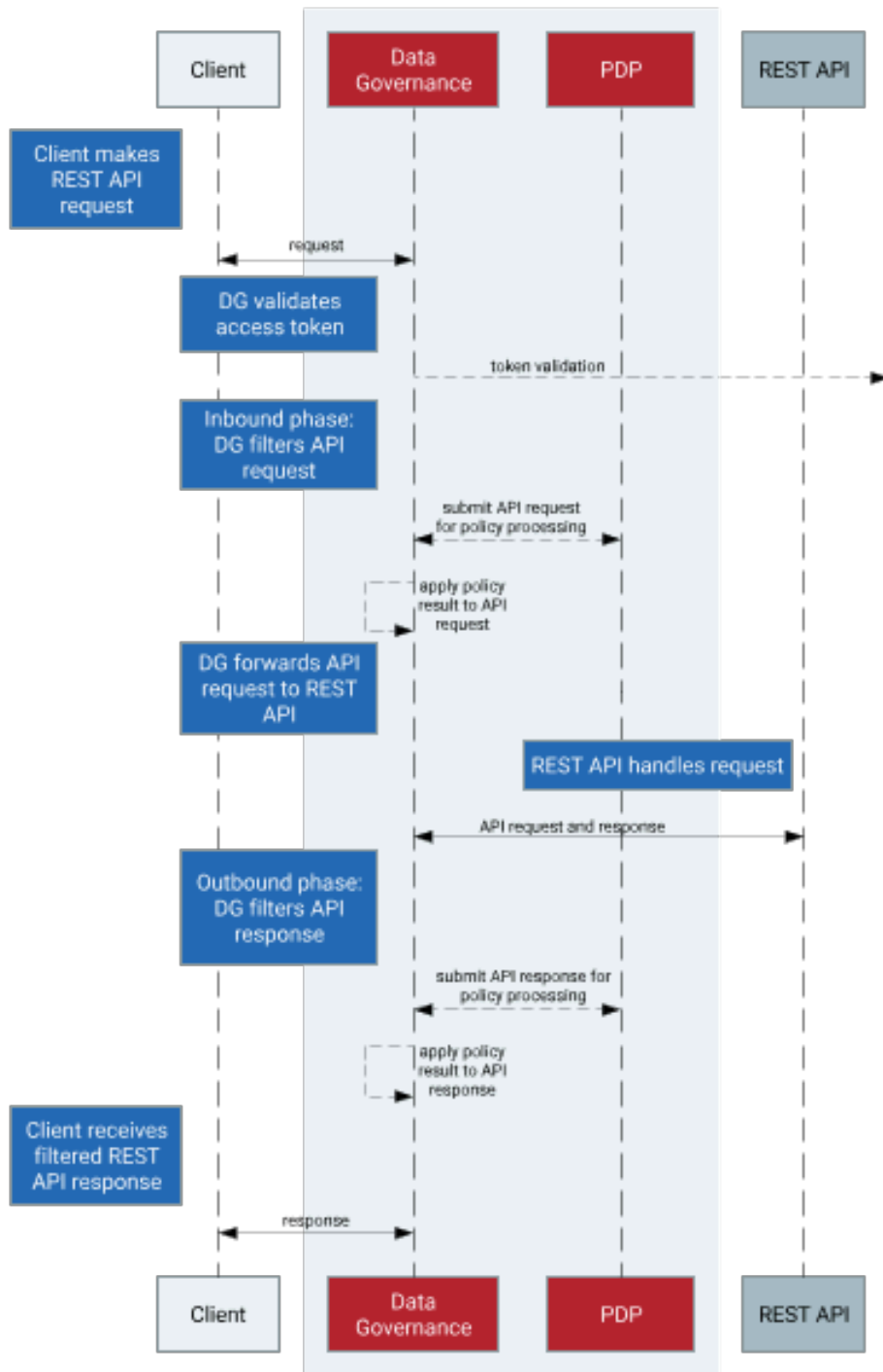
About the API security gateway

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.



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
<code>http-auth-evaluation-behavior</code>	Determines whether the Gateway API Endpoint evaluates bearer tokens, and if so, whether the bearer token is forwarded to the API server.
<code>access-token-validator</code>	<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 <code>http-auth-evaluation-behavior</code> is set to <code>do-not-evaluate</code>, 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 163.

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

Policy request attributes	Description	Type
domain	Unused.	String
identityProvider	Identifies the access token validator that evaluates the bearer token used in an incoming request.	String
attributes	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
HttpRequest	Identifies the HTTP request.	Object
TokenOwner	The access token subject as a SCIM resource, as obtained by the access token validator.	Object
Gateway	Provides additional gateway-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
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
ResponseStatus	The HTTP response status code, if available.	Number
RequestHeaders	The HTTP request headers.	Object
ResponseHeaders	The HTTP response headers, if available.	Object

Attribute	Description	Type
ClientCertificate	Properties of the client certificate, if one was used.	Object
CorrelationId	A unique value that identifies the request and response, if available.	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
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

Attribute	Description	Type
<code>token_type</code>	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
<code>algorithm</code>	Name of the certificate signature algorithm, such as <code>SHA256withRSA</code> .	String
<code>algorithmOID</code>	Signature algorithm OID.	String
<code>notBefore</code>	Earliest date on which the certificate is considered valid.	DateTime
<code>notAfter</code>	Expiration date and time of the certificate.	DateTime
<code>issuer</code>	Distinguished name (DN) of the certificate issuer.	String
<code>subject</code>	DN of the certificate subject.	String
<code>valid</code>	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
<code>base path parameters</code>	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
<code>custom attribute</code>	The <code>Gateway</code> 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 <code>service</code> value appears in the policy request as the <code>service</code> attribute.</p> <p>If undefined, the <code>service</code> 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 <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 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
inbound-base-path	/accounts/{accountId}/transactions
outbound-base-path	/api/v1/accounts/{accountId}/ transactions
policy-request-attribute	foo=bar

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
inbound-base-path	/health/{tenant}/{resourceType}
outbound-base-path	/api/v1/health/{tenant}/{resourceType}
service	HealthAPI.{resourceType}
resource-path	{resourceType}/{_TrailingPath}

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
requestURI	String	Original Request URI
requestQueryParams	Object	Query parameters as JSON object
headers	Object	Request headers as JSON object
correlationID	String	Request correlation ID

For more information, see [Error templates](#) on page 114.

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*.

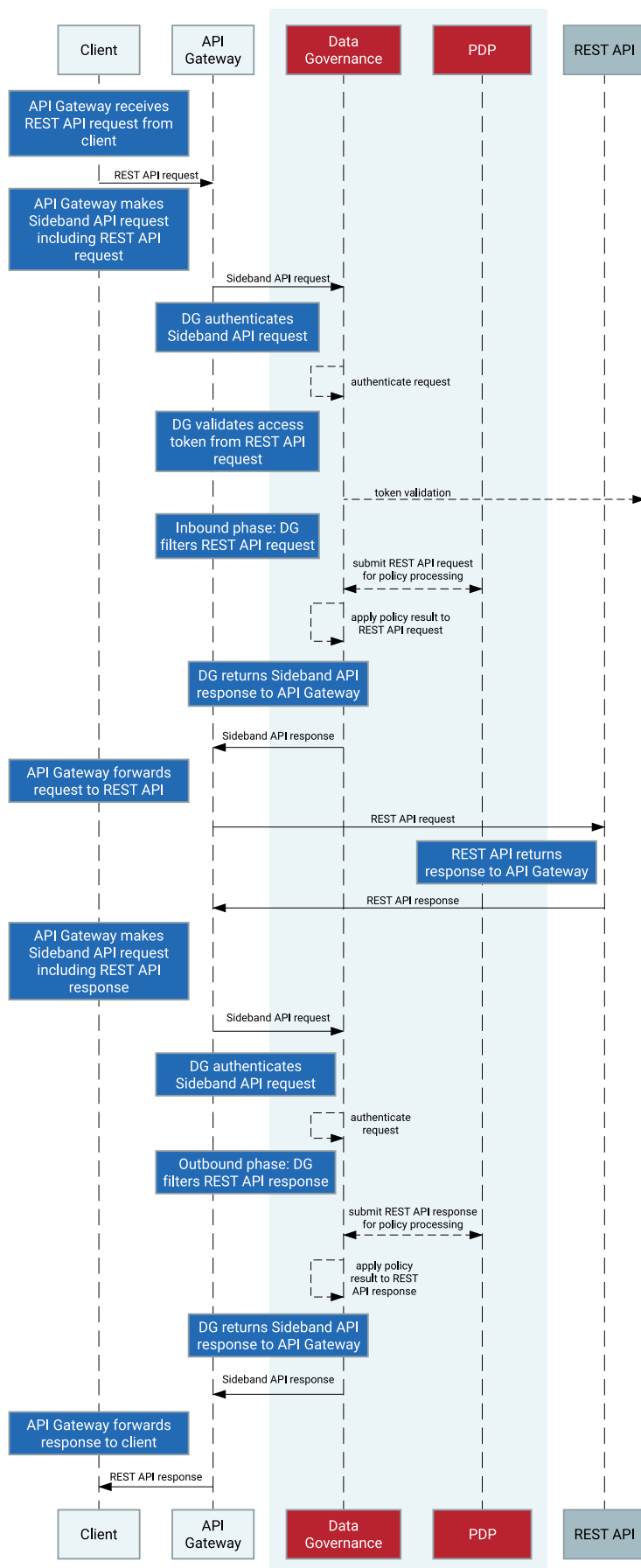
Using the Sideband API, PingDataGovernance Server can be configured instead as a plugin to an external API gateway. In Sideband mode, an API gateway integration point performs the following steps:

- 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.



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 107.
- 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 107.
- 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 109

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.

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.

Property	Description
<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 <code>GET</code> or <code>POST</code>.</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
<code>service</code>	<p>Identifies the API service. By default, this value is set to the name of the Sideband API Endpoint.</p> <p>To override the default value, set the Sideband API Endpoint's <code>service</code> property.</p> <p>Multiple Sideband API Endpoints can use the same service value.</p>	String
<code>domain</code>	Unused.	String

Attribute	Description	Type
<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>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
<code>RequestBody</code>	Request body, if available.	Object
<code>ResponseBody</code>	Response body, if available. This field is provided only for outbound policy requests.	Object
<code>ResponseStatus</code>	The HTTP response status code, if available.	Number
<code>RequestHeaders</code>	The HTTP request headers.	Object
<code>ResponseHeaders</code>	The HTTP response headers, if available.	Object
<code>ClientCertificate</code>	Properties of the client certificate, if one was used.	Object

Attribute	Description	Type
CorrelationId	A unique value that identifies the request and response, if available.	Object

Note: When handling an outbound response, HTTP request data is only available if specifically provided by the API gateway plugin.

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
<code>base path parameters</code>	Parameters in a Sideband API Endpoint's <code>base-path</code> configuration property are included as fields of the <code>Gateway</code> attribute.	String
<code>base path parameters</code>	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 113.

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

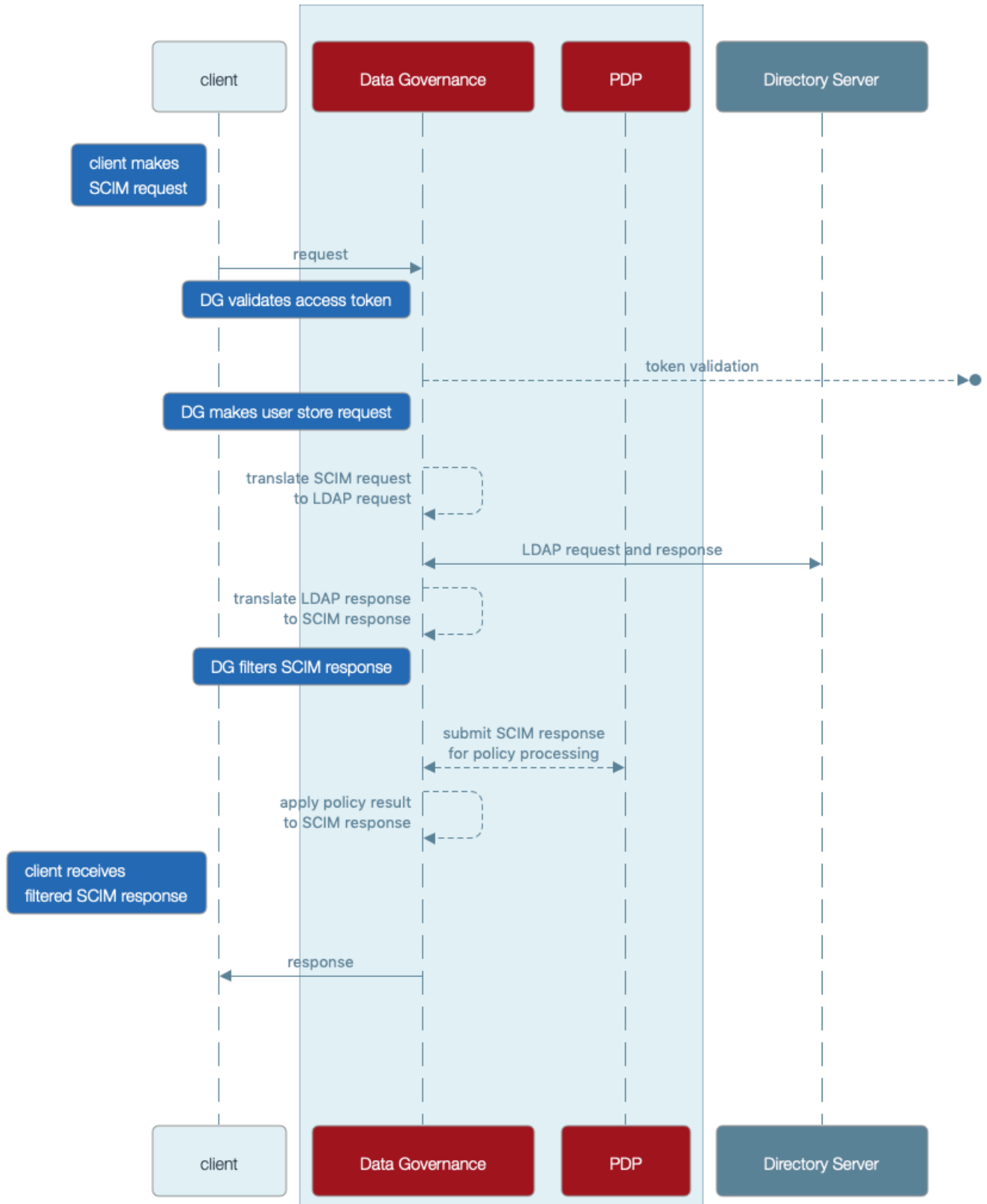
PingDataGovernance Server's built-in SCIM service provides a REST API for data that is stored in one or more external datastores, based on the [SCIM 2.0 standard](#).

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 SCIM 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 33.

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
```

The response is:

```
{"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-results For more information about authorizing searches, see About SCIM searches on page 125.

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 163.

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

Policy request attribute	Description	Type
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
ResourcePath	Uniquely identifies the SCIM resource that is being requested, in the format <code><Resource Type>/<SCIM ID></code> , as the following example shows: Users/0450b8db-f055-35d8-8e2f-0f203a291cd1	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

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

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

Attribute	Description	Type
<code>resource</code>	Complete SCIM resource that the request targets. This attribute is available for GET, PUT, PATCH, and DELETE requests. 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 125.	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-results** 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.

About the PDP API

Important: The PDP API feature requires PingDataGovernance Premier. For information about access to PingDataGovernance Premier, contact your Ping Identity account representative.

While the PingDataGovernance Server's main functionality is to enforce fine-grained policies for data accessed via APIs, organizations might have requirements to use the core Policy Decision Service for other, non-API use cases. For example, an application server may use it to request policy decisions when generating dynamic web content. In this configuration, the PingDataGovernance Server becomes the Policy Decision Point (PDP), and the application server becomes the Policy Enforcement Point (PEP).

PingDataGovernance Server's PDP API provides an endpoint to support these use cases. Enforcement points request policy decisions based on a subset of the XACML-JSON standard (XACML 3.0 JSON Profile 1.1).

Note: The PDP API can indicate when a particular request or response triggers advice, but it is up to the application server to actually implement this advice.

For the PDP API to be available, an administrator must configure the PingDataGovernance Server with a feature-enabled license during setup. In addition, the admin must configure an Access Token Validator (see [Access Token Validators](#)), and the Policy Decision Point Service (see [Use policies in a production environment](#)).

Request and Response Flow

The PDP API is implemented as a single endpoint, which consuming application servers can access via POST requests to the `/pdp` path. The requests must include the appropriate `Content-Type` and `Accept` headers, and request bodies must adhere to the XACML-JSON standard (see [XACML-JSON response conversion](#) on page 131 for a concrete example).

PDP API Endpoint path	Action	Content-Type/Accept	Request data
<code>/pdp</code>	POST	application/xacml+json	XACML-JSON

The PDP API provides an HTTP API for decisions determined based on the policies configured within the PingDataGovernance Server's Policy Decision Service. At a high level, a successful PDP API request goes through the following two-phase flow:

- First, the client makes the XACML-JSON request, which is received by the PDP API. The PDP API converts the request to a PingDataGovernance Server decision request and attempts to authorize the client.
- On authorize success, the request is handed off to the Policy Decision Service which processes a decision for the PDP API. The PDP API then converts the decision to a XACML-JSON response and writes the response to the client.

The following sections describe these stages in more detail.

Requests

The PDP API first converts the XACML-JSON request to a decision request for the policy decision point to be consumed later by the Policy Decision Service. Policies may match a decision request by `Service`, `Domain`, `Action`, or other attributes. The following table describes how these trust framework entities map to a XACML-JSON request:

XACML-JSON	PingDataGovernance decision request
<code>\$.Request.AccessSubject[0].Attribute[0]</code>	Domain
<code>\$.Request.Action[0].Attribute[0].Value</code>	Action
<code>\$.Request.Resource[0].Attribute[0].Value</code>	Service
<code>\$.Request.Environment[0].Attribute[*].Attribute</code>	Attribute Name
<code>\$.Request.Environment[0].Attribute[*].Value</code>	Attribute Value (as a JSON escaped string)

The following example XACML-JSON request body represents a request that an application server might make while generating the content for a protected Performance Dashboard Summary page for an employee of a company. The employee can access the application, but only has access to limited content. The employee has authenticated using a social network called "Spacebook."

```
{
  "Request": {
    "AccessSubject": [{
      "Attribute": [{
        "AttributeId": "Domain",
        "Value": "PerformanceDashboard"
      }]
    }]
  }
}
```



```

  }],
  "Action": [{
    "Attribute": [{
      "AttributeId": "Action",
      "Value": "read"
    }]
  }],
  "Resource": [{
    "Attribute": [{
      "AttributeId": "Service",
      "Value": "Pages.Summary"
    }]
  }],
  "Environment": [{
    "Attribute": [{
      "AttributeId": "role",
      "Value": "employee"
    }]
  }],
  "Category": [{
    "CategoryId": "symphonic-idp",
    "Attribute": [{
      "AttributeId": "Identity Provider",
      "Value": "Spacebook"
    }]
  }],
}

```

Note: Besides Environment, the AttributeIds are not consumed by the system and may be used for readability as they are in the example.

The response for this request body can be seen in [XACML-JSON response conversion](#) on page 131.

Authorization

Before calculating a decision, the PDP API attempts to authorize the client making the PDP API request by invoking the Policy Decision Service. A PDP authorization request can be targeted in policy as having service PDP with action authorize. The default policies included with the PingDataGovernance Server perform this authorization by only permitting requests with active access tokens which contain the `urn:pingdatagovernance:pdp scope`.

This policy can be seen in **Global Decision Point > PDP API Endpoint Policies > Token**

The screenshot displays the configuration for the 'Token Authorization' policy. At the top, there are tabs for 'Details', 'History', 'Test', and 'Analysis'. The policy name 'Token Authorization' is shown with a 'Disabled' checkbox. Below this is a 'Description' field. The 'Applies to 1 definition' section shows a dropdown menu with 'authorize' selected. A message indicates to 'Drag and Drop Targets or Targetable Definitions from Toolbox'. The 'A single deny will override any permit decisions' section contains two rules: 'Token is inactive' and 'Token does not contain PDP scope', both with red 'x' icons. Below the rules is a '+ Create new Rule' button and a 'Drag and Drop Rules from Toolbox' button. The 'Advice (1 in total)' section shows one advice rule, 'Invalid Token', with an 'Obligatory' checkbox checked. Below the advice is a '+ Create new Advice' button and a 'Drag and Drop Advice from Toolbox' button. At the bottom, there are links for 'Hide "Applies to"', 'Show "Applies when"', 'Hide Advice', and 'Show Properties'.

Authorization.

Note: The parent of the Token Authorization policy, PDP API Endpoint Policies, is what constrains the Token Authorization policy to apply only to the PDP service.

For example, under the default policies, the following request would result in an authorized client when the PDP is configured with a Mock Access Token Validator:

```
curl --insecure -X POST \
  -H 'Authorization: Bearer
  {"active":true,"scope":"urn:pingdatagovernance:pdp", "sub":"<valid-
  subject>"}' \
  -H 'Content-Type: application/xacml+json' \
  -d '{"Request":{}}' "https://<your-dg-host>:<your-dg-port>/pdp"
```

The default policies are intended to be a sensible starting point, and the policy writer may decide to modify these policies if additional authorization logic is required.

Decision Processing

On successful client authorization, the Policy Decision Service is invoked with the decision request converted from the XACML-JSON request. When writing policy for the PDP API endpoint, it is important to note the mapping between the XACML-JSON schema and the PingDataGovernance Server decision request, described in [XACML-JSON response conversion](#) on page 131. Once the decision response is determined, it is handed back to the PDP API to provide to the client.

XACML-JSON response conversion

The PDP API converts a decision response to a XACML-JSON response, which includes a decision (Permit/Deny/etc.), and any obligations or advice that were found to match during policy processing. Note that it is up to the Policy Enforcement Point (PEP) to actually apply any of the obligations and advice.

The mapping from decision response to XACML-JSON response is indicated in the table below.

PingDataGovernance Server decision response	XACML-JSON response
Decision	Decision
Advice (obligatory)	Obligations
▪ Advice code	Obligations[].Id
▪ Advice payload	Obligations[].AttributeAssignments[].Value
Advice (non-obligatory)	AssociatedAdvice
▪ Advice code	AssociatedAdvice[].Id
▪ Advice payload	AssociatedAdvice[].AttributeAssignments[].Value

Going back to the hypothetical example from XACML-JSON request conversion where an employee attempts to access a Performance Dashboard application, an appropriate response could be:

```
{
  "Response": {
    "Decision": "Permit",
    "Obligations": [
      {
        "Id": "disclaimer",
        "AttributeAssignments": [
          {
            "AttributeId": "payload",
            "Value": "confidential"
          }
        ]
      }
    ],
    "AssociatedAdvice": [
      {
        "Id": "content-gen",
        "AttributeAssignments": [
          {
            "AttributeId": "payload",
            "Value": "limited"
          }
        ]
      }
    ]
  }
}
```

In the above example, it is up to the application server to know to interpret these statements by generating limited content and including a confidentiality disclaimer.

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 135 and [Embedded PDP mode](#) on page 141.

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.

The screenshot shows the configuration page for a Trust Framework service. The service is named "Get consent to share profile". The "Parent" is set to "no parent selected". The "Service Settings" section shows "Service Type" as "Restful". The "Restful Settings" section includes:

- URL Format: `{{ConsentService.BaseUri}}/consents?subject={{HttpRequest.AccessToken.subject}}&definition=share-profile-data`
- Http Method: GET
- Content Type: application/json
- Body: (empty)
- 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.

The screenshot shows the configuration page for a Trust Framework attribute named "Password". The "Parent" is set to "ConsentService". The "Resolver Settings" section includes a "+ Add Resolver" button. The "Cache Settings" section shows "Cache Strategy" as "No Caching". The "Value Settings" section includes:

- 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 139.

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. Select **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 shows the 'Definition Editor' interface for a service named 'ConsentServicePassword'. The interface is divided into several sections:

- Parent:** PolicyConfigurationService
- Description:** (Empty text field)
- Service Settings:** Service Type is Restful.
- Restful Settings:**
 - URL Format:** {{ConfigurationKeyService.BaseUri}}/policy-decision-service/policy-configuration-keys/ConsentServicePassword
 - Http Method:** GET
 - Content Type:** application/json
 - Body:** (Empty text field)
 - Authentication:** Basic
 - Username:** ConfigurationKeyService.Username
 - Password:** ConfigurationKeyService.Password
- Headers:** (Empty list with a '+ Header' button)
- 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

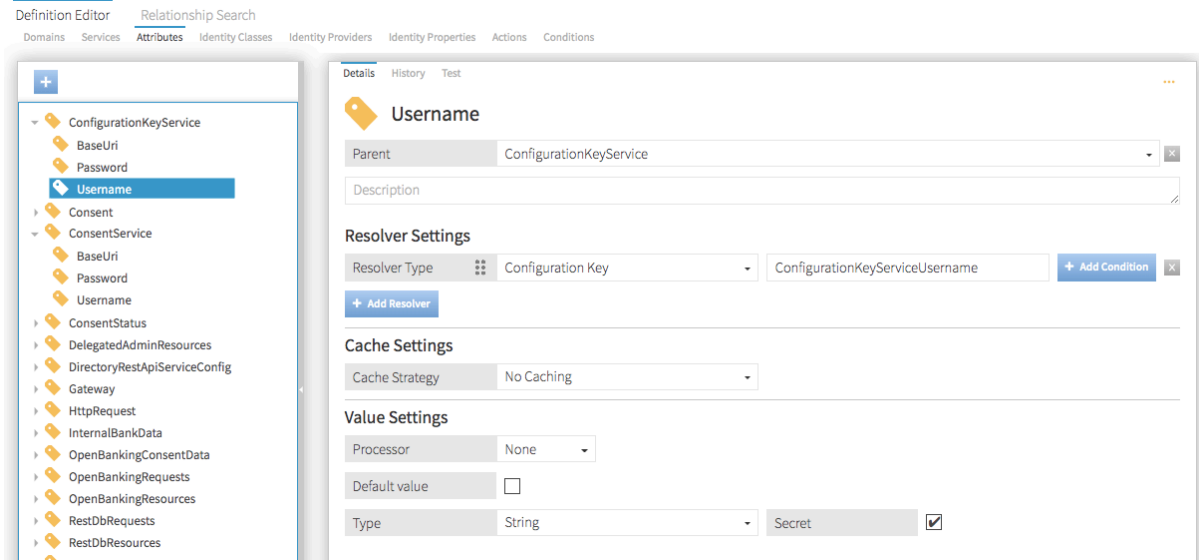
About this task

To define a new attribute, perform the following steps:

Steps

1. In the PingDataGovernance Policy Administration GUI, go 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. In the PingDataGovernance Policy Administration GUI, go 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.

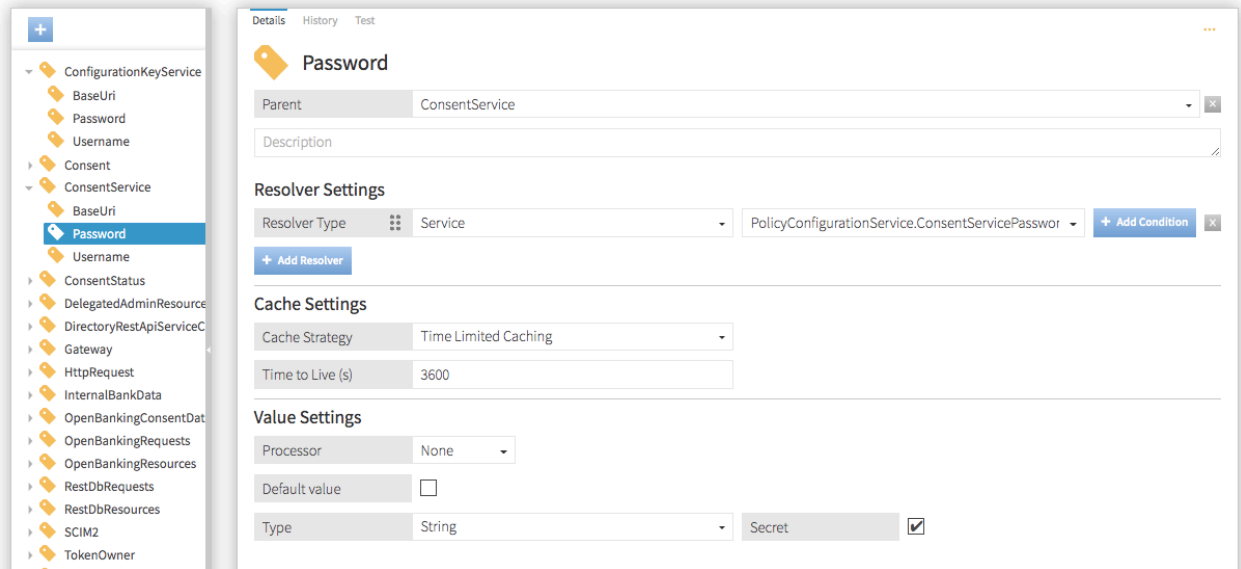
Steps

1. In the PingDataGovernance Policy Administration GUI, go 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



Details History Test

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

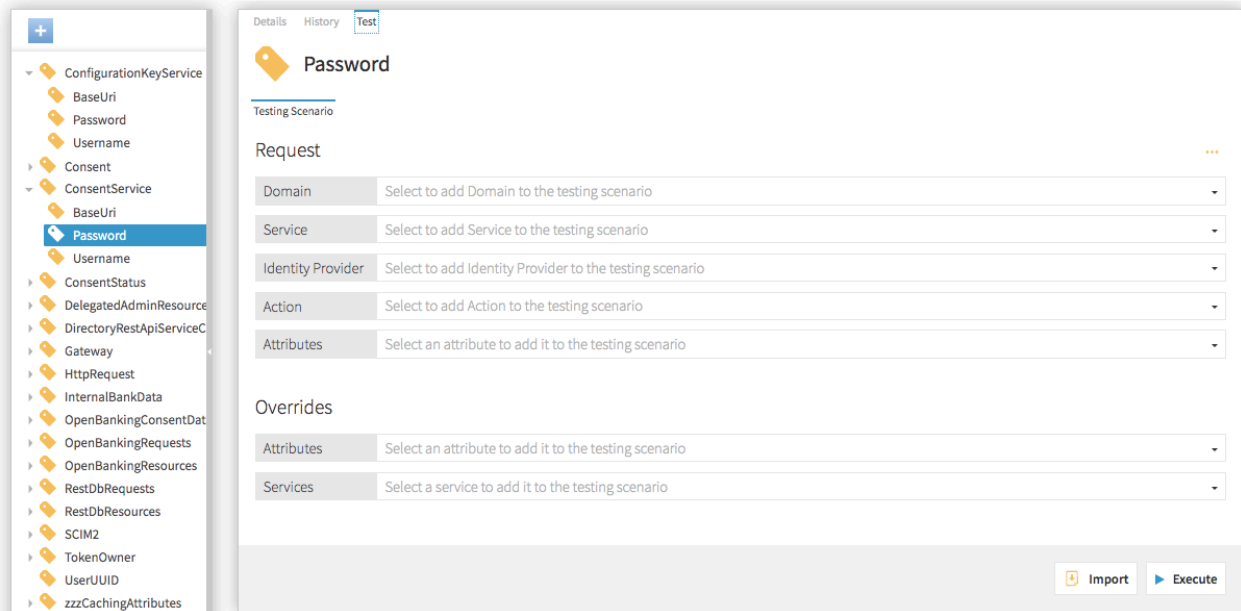
Repeat this task for **ConsentServiceUsername** and **ConsentServiceBaseUri**.

Test your changes

About this task

Definition Editor Relationship Search

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



Details History **Test**

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

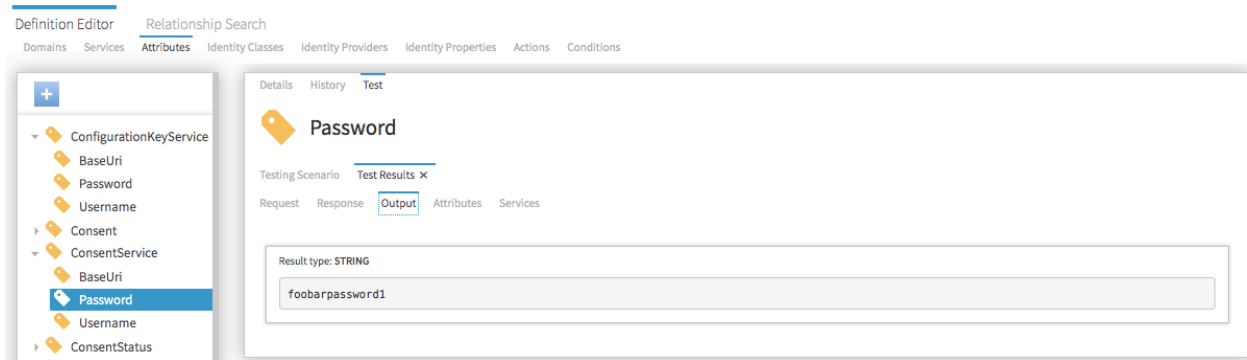
Attributes: Select an attribute to add it to the testing scenario

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. In the PingDataGovernance Policy Administration GUI, go 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**.

Results

Definition Editor Relationship Search

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

Details History Test

Password

Parent: ConsentService

Description:

Resolver Settings

Resolver Type	Configuration Key	ConsentServicePassword	+ Add Condition
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 **ConsentServiceBaseUri** and **ConsentServiceUsername**.

Advice

When a policy is applied to a request or response, the policy result might include one or more *advices*. An advice is a directive that instructs the policy enforcement point to perform additional processing in conjunction with an authorization decision. In this example, PingDataGovernance Server functions as the policy enforcement type.

Advices allow PingDataGovernance Server to do more than simply allow or deny access to an API resource. For example, an advice might cause the removal of a specific set of fields from a response.

An advice can be added directly to a single policy or rule, or it can be defined in the Toolbox for use with multiple policies or rules. Advices possess the following significant properties:

Advice property	Description
Name	Friendly name for the advice.
Obligatory	If true, the advice must be fulfilled as a condition of authorizing the request. If PingDataGovernance cannot fulfill an obligatory advice, it fails the operation and returns an error to the client application. If a non-obligatory advice cannot be fulfilled, an error is logged, but the client's requested operation continues.
Code	Identifies the advice type. This value corresponds to an advice ID that the PingDataGovernance configuration defines.
Applies To	Specifies the policy decisions, such as <code>Permit</code> or <code>Deny</code> , that include the advice with the policy result.

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
- Modify Attributes
- Modify Query
- Prohibit Attributes

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

i 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 125.

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: DENY decisions.

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: PERMIT decisions, although [Include or Exclude] Attributes advice cannot be applied directly to a SCIM search.

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 145.

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: PERMIT decisions from Gateway, although Filter Response advice cannot be applied directly to a SCIM search. However, the SCIM service performs similar processing automatically when it handles a search result. For every candidate resource in a search result, a policy request is made for the resource with an Action value of retrieve.


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	Requ	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 might degrade 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: PERMIT decisions, although [Include or Exclude] Attributes advice cannot be applied directly to a SCIM search.

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 145.

Modify Attributes

Advice ID: `modify-attributes`

Description: Modifies the values of attributes in the JSON request or response.

Applicable to: All, although Modify Attributes advice cannot be applied directly to a SCIM search

The payload for this advice is a JSON object. Each key/value pair is interpreted as an attribute modification on the request or response body of the request being authorized. For each pair, the key is a JSONPath pointing to the attribute to be modified, and the value is the value to set for the attribute. The value can be any valid JSON value (including a complex value such as an object or array).

Modify Query

Advice ID: `modify-query`

Description: Modifies the query string of the request sent to the API server.

Applicable to: All

The payload for this advice is a JSON object. The keys are the names of the query parameters which must be modified, and the values are the new values of the parameters. A value can be one of the following:

- Null, in which case the query parameter is removed from the request
- A string, in which case the parameter is set to that specific value
- An array of strings, in which case the parameter is set to all of the values in the array

If the query parameter already exists on the request, it will be overwritten. If it does not already exist, it will be added.

As an example, if a request is made to a proxied API, with a request URL of `https://example.com/users?limit=1000`, a policy may be used to limit certain groups of users to only request 20 users at a time, and a payload of `{"limit": 20}` will cause the URL to be rewritten as `https://example.com/users?limit=20`.

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*.

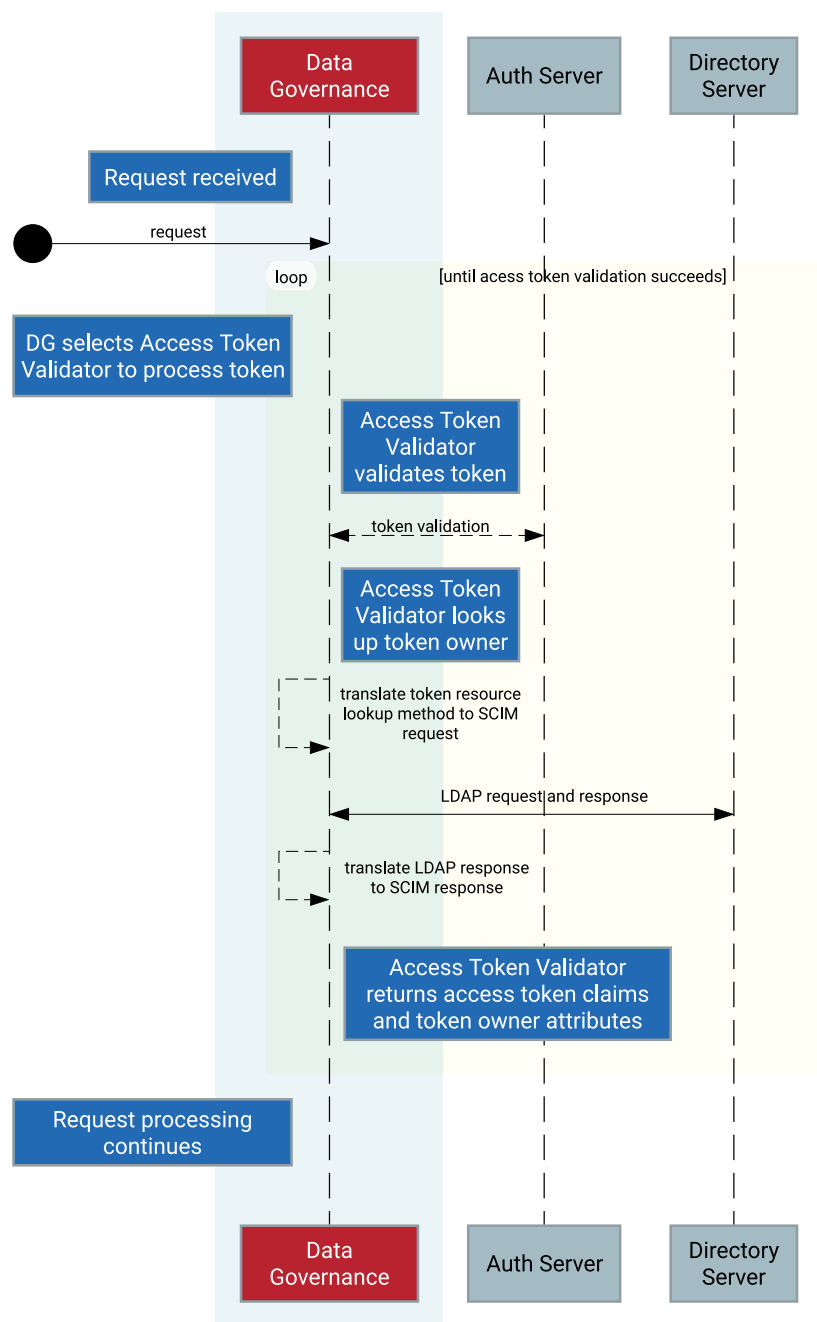
Most access tokens identify a user, also called the *token owner*, as its subject. Access token validators can retrieve the token owner's attributes from the User Store using a related component called a *token*

resource lookup method. The user data obtained by a token resource lookup method 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.



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.

 **Note:** For deployments that do not use SCIM, token owner attributes can be retrieved from other user store types by writing a token resource lookup method extension with the Server SDK.

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-lookup-method \
  --validator-name "PingFederate Access Token Validator" \
  --method-name "User by uid" \
  --type scim \
  --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 "PingFederate Access Token Validator" \
  --method-name "User by uid" \
  --type scim \
  --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" \
  --type scim \
  --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 Policy Administration GUI" \
  --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 \
  --type scim \
  --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.

i 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.

i 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.

i 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 \  
--alias server-cert
```



```
--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"
```

Manage monitoring

StatsD monitoring endpoint

The Monitoring Endpoint configuration type provides the StatsD Endpoint type that you can use to transfer metrics data in the StatsD format.

Examples of metrics you can send are:

- Busy worker thread count
- Garbage collection statistics
- Host system metrics such as CPU and memory

For a list of available metrics, use the interactive `dsconfig` menu for the Stats Collector Plugin or in the Administrative Console, edit the **Stats Collector** plugin as explained in the second example.

You configure the Monitoring Endpoint using the `dsconfig` command. When you configure Monitoring Endpoint, you include:

- The endpoint's hostname
- The endpoint's port
- A toggle to use TCP or UDP
- A toggle to use SSL if you use TCP

For example, to configure a new StatsD Monitoring Endpoint to send UDP data to localhost port 8125 using `dsconfig`:

```
dsconfig create-monitoring-endpoint \
  --type statsd \
  --endpoint-name StatsDEndpoint \
  --set enabled:true \
  --set hostname:localhost \
  --set server-port:8125 \
  --set connection-type:unencrypted-udp
```

If you are using the Administrative Console:

1. Click **Show Advanced Configuration**.
2. In the **Logging, Monitoring, and Notifications** section, click **Monitoring Endpoints**.
3. Click **New Monitoring Endpoint**.

You can send data to any number of monitoring endpoints.

The Stats Collector Plugin controls the metrics used by the StatsD monitoring endpoint. To send metrics with the StatsD monitoring endpoint, you must enable the Stats Collector Plugin. Also, you must configure the Stats Collector Plugin to indicate the metrics to send.

To enable the Stats Collector Plugin or to configure the type of data sent, use the `dsconfig` command or the Administrative Console. For example, to enable the Stats Collector Plugin to send host CPU metric, memory metrics, and server status metrics using `dsconfig`:

```
dsconfig set-plugin-prop \
  --plugin-name "Stats Collector" \
  --set enabled:true \
  --set host-info:cpu \
  --set host-info:disk \
  --set status-summary-info:basic
```

If you are using the Administrative Console:

1. Click **Show Advanced Configuration**.
2. In the **LDAP (Administration and Monitoring)** section, click **Plugin Root**
3. Edit the **Stats Collector** plugin.

After you enable the Stats Collector and create the StatsD monitoring endpoint, you can:

- Use the data with Splunk as explained in [Sending metrics to Splunk](#) on page 162.
- Configure other tools that support StatsD, such as CloudWatch or a Prometheus StatsD exporter, to use the data. For more information about this configuration, see your tool's StatsD documentation. Configure the StatsD monitoring endpoint to use the correct host and port. The `dsconfig create-monitoring-endpoint` example above uses a host of localhost and a port of 8125. You can also set these values in the Administrative Console.

Sending metrics to Splunk

About this task

With the StatsD Endpoint type, you can send metric data to a Splunk installation.

In Splunk, you can use SSL to secure ports that are open for StatsD.

 **Note:**

StatsD metrics are typically sent over UDP. By using UDP, the client sending metrics does not have to block as it would if using TCP. However, using TCP guarantees order and ensures no metrics are lost.

You can configure open UDP (or TCP) ports in Splunk to accept only connections from a certain hostname or IP address.

To securely send UDP (or TCP) data to Splunk, you can:

Steps

1. Send the data to a Splunk Universal Forwarder.
2. Have the forwarder communicate with the Splunk Indexer over SSL.

Capture debugging data

If problems arise (whether from issues in PingDataGovernance Server itself or a supporting component, such as the JVM, the operating system, or the hardware), you can capture diagnostic data. With this data, you can troubleshoot the problem quickly to determine the underlying cause and the best course of action to take to resolve it. This chapter describes how to capture the diagnostic data.

Export policy data

About this task

This section provides instructions to export 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.


To export policy data, perform the following steps:

Steps

1. Go 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="dg1" startupID="XNM9Hw==" threadID=140 from=[0:0:0:0:0:0:1]: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="dg1" startupID="XNRLuQ==" threadID=139 from=[0:0:0:0:0:0:1]: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

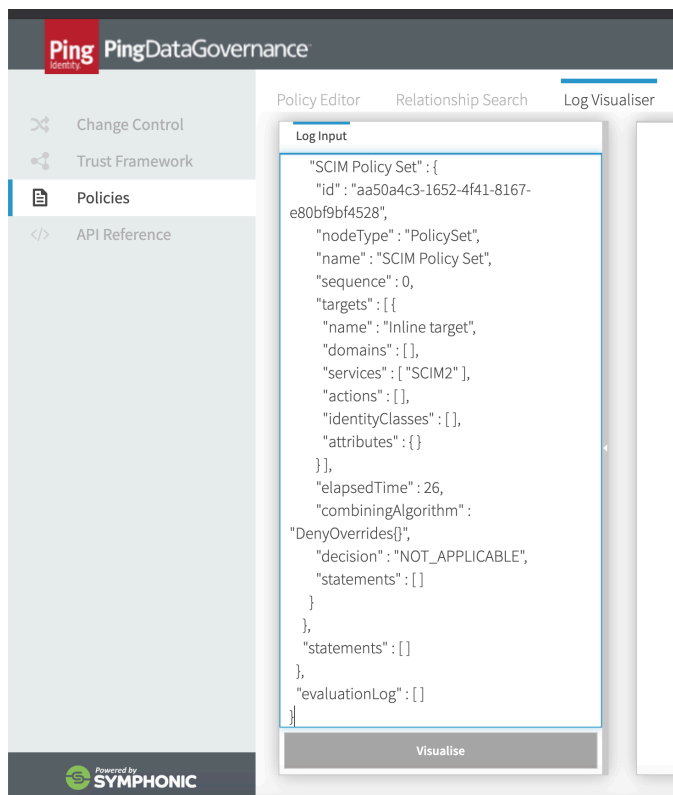
Before attempting to troubleshoot or trace a policy-decision response, make certain that the Trace log is enabled within PingDataGovernance Server. For more information, see [Configure PingDataGovernance logging](#) on page 35.

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

Steps

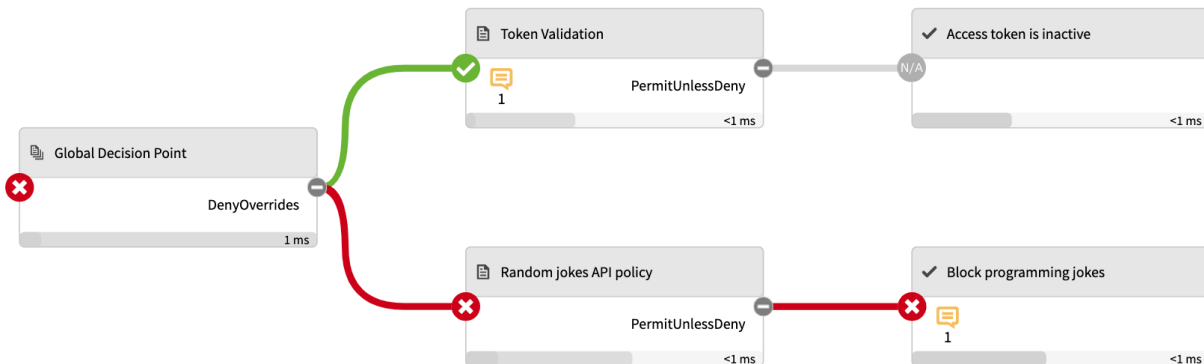
1. From within the policy-decision file, copy the policy-decision response JSON.
2. In the Policy Administration GUI, go to **Policies**.
3. Click the **Log Visualizer** tab.
4. In the **Log Input** text box, paste the policy-decision response JSON.

5. Click Visualize.



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 163. 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`

Upgrade PingDataGovernance Server

Ping Identity issues software release builds periodically with new features, enhancements, and fixes for improved server performance. PingDataGovernance is unique in that it includes two server applications that must be upgraded in tandem — the main PingDataGovernance Server, and the Policy Administration GUI.

Note: A PingDataGovernance Server used in external PDP mode requires a Policy Administration GUI with the same version. When upgrading PingDataGovernance Server, the Policy Administration GUI must also be upgraded.

Upgrade overview and considerations

The upgrade process involves downloading and unzipping a new version of the PingDataGovernance Server ZIP file on the server to be updated and running the update utility with the `--serverRoot` or `-R` option value from the new root server pointing to the installation to be upgraded.

Consider the following when upgrading:

- The update affects only the server being upgraded. The process does not alter the configuration of other servers, so updates of those servers need to be performed separately. Instructions for updating the PingDataGovernance Policy Administration GUI server can be found in a later section of this chapter.
- The update tool will verify that the version of Java that is installed meets the new server requirements. To simplify the process, install the version of Java that is supported by the new server before running the tool.
- Upgrade for PingDataGovernance Server is only supported from versions 7.0.0.0 or higher. If upgrading from a version of PingDataGovernance prior to 7.3.0.0 there will be configuration loss. The update tool has a warning message about this.

Upgrading PingDataGovernance Server

About this task

Perform the following steps to upgrade a PingDataGovernance Server.

Steps

1. Download and unzip the new version of the PingDataGovernance Server in a location outside the existing server's installation. For these steps, assume the existing server installation is in `/opt/dg/PingDataGovernance` and the new server version has been unzipped into `/home/stage/PingDataGovernance`.
2. Provide a copy of the PingDataGovernance license file for the version to which you are upgrading in the `/home/stage/PingDataGovernance` directory or give the location of the license file to the tool using the `--licenseKeyFile` option.
3. Run the `update` tool provided with the new server package to update the existing PingDataGovernance Server. The `update` tool might prompt for confirmation on server configuration changes if it detects customization.

```
/home/stage/PingDataGovernance/update --serverRoot /opt/dg/
PingDataGovernance
```

Reverting an update

About this task

After PingDataGovernance Server has been updated, you can revert to the previous version (one level back) using the `revert-update` tool. The `revert-update` tool accesses a log of file actions taken by the updater to put the file system back to its previous state. If you have run multiple updates, you can run the `revert-update` tool multiple times to sequentially revert to each prior update. You can only revert back one level at a time with the `revert-update` tool. For example, if you had to run the update twice since first installing PingDataGovernance Server, you can run the `revert-update` tool to revert to its previous state, then run the `revert-update` tool again to return to its original state.

When starting the server for the first time after a revert has been run, the server will display warnings about "offline configuration changes," but these are not critical and will not appear during subsequent start-ups.

To revert a server update, perform the following step:

- Run `revert-update` in the server root directory to revert back to the most recent previous version of the server

```
/opt/dg/PingDataGovernance/revert-update
```

Upgrade the PingDataGovernance Policy Administration GUI

About this task

This section describes how a server administrator can upgrade the PingDataGovernance Policy Administration GUI when a new version is released. You can use the PingDataGovernance Server's update utility to upgrade the server to a new version. The following topics describe a scenario where an administrator wants to transfer an existing server's configuration to a new server installation. In these examples, assume that the current Policy Administration GUI is located in `/opt/dg/PingDataGovernance-PAP`.

Back Up Policies

About this task

Policy writers may want to back up existing policies before upgrading the Policy Administration GUI. Do this by exporting policy snapshots as described in the following steps:

Steps

1. Log in to the Policy Administration GUI and choose any existing branch to go to the main landing page.
2. Select **Change Control# Version Control** to display your current branches.
3. Click a branch from the list that you wish to export. You should see a list of the commits for that branch. Identify the commit that represents the snapshot that you want to export (the most recent version of the branch is named **Uncommitted changes**) and click the hamburger icon in the **Options** column.
4. Choose to export the snapshot. Your browser will download the file.
5. Repeat for any additional branches that you wish to back up.

Transfer Configuration Files

About this task

This section describes how to copy key configuration files to a new server installation.

Note: Log files will not be transferred to the new server location.

Steps

1. Go to your existing PingDataGovernance Policy Administration GUI installation directory and stop the server.

```
cd /opt/dg/PingDataGovernance-PAP
bin/stop-server
```

2. Rename the existing PingDataGovernance Policy Administration GUI directory.

```
mv /opt/dg/PingDataGovernance-PAP /opt/dg/PingDataGovernance-PAP-previous
```

3. Unzip the new PingDataGovernance Policy Administration GUI distribution to the desired location. In this example, the new Policy Administration GUI directory will be `/opt/dg/PingDataGovernance-PAP`.

```
unzip PingDataGovernance-PAP-x.x.x.x.zip -d /opt/dg
```

4. Copy the following files from the existing PingDataGovernance Policy Administration GUI directory to the new location.

```
cp /opt/dg/PingDataGovernance-PAP-previous/Symphonic.mv.db /opt/dg/
PingDataGovernance-PAP
```

Note: If your existing PingDataGovernance Policy Administration GUI installation uses a custom certificate keystore, you should also copy this file to the new location and specify it when running setup in **Configure and Start the Server**. For example:

```
cp /opt/dg/PingDataGovernance-PAP-previous/keystore.p12 /opt/dg/
PingDataGovernance-PAP
```

Upgrade the Policy Database

About this task

The new server installation may require changes to the policy database structure. The server comes with a script that performs these upgrades.

Steps

1. To upgrade the policy database file (`Symphonic.mv.db`), run the following command:

```
cd /opt/dg/PingDataGovernance-PAP
./bin/update-db
```

2. Once the database has been upgraded, move it to the expected location:

```
mv /opt/dg/PingDataGovernance-PAP/Symphonic.mv.db /opt/dg/
PingDataGovernance-PAP/admin-point-application/db/Symphonic.mv.db
```

Configure and Start the Server

About this task

Use the setup tool to generate a new `configuration.yml` file. This will ensure that any configuration properties required by the new server version are present.

Note: The setup tool requires a valid license file, which you can place either in the server's root directory or specify by using the `--licenseKeyFile` option.

```
./bin/setup --licenseKeyFile /path/to/PingDataGovernance.lic
```

Follow the prompts for your desired installation.

Note: When upgrading from a DataGovernance version prior to 8.0.0.0, you will need to modify the output configuration file located at `<server-root>/config/configuration.yml`. In a text editor, change the following value:

```
Database.User: "pap_user"
```

to:

```
Database.User: "sa"
```

After performing the steps above, start the server by running the following command.

```
./bin/start-server
```

Upgrade Trust Framework and Policies

About this task

PingDataGovernance ships with a default trust framework and policy snapshot that policy writers should use as a starting point when developing their policies. Occasionally a server upgrade results in changes to the default trust framework and policies, and policy writers will need to upgrade any policies that had been based on `defaultPolicies.SNAPSHOT`.

Steps

1. Log in to the Policy Administration GUI and choose any branch to go to the main landing page.
2. Select **Change Control** from the navigation bar on the left, and open the **Merge Snapshot** tab.
3. Click the file selection option, and go to the `resources/policies/upgrade-snapshots` folder of the new Policy Administration GUI deployment.

4. Select the correct `SNAPSHOT` file based on the version you are upgrading from and the version to which you are upgrading (for example, when upgrading from version 7.3.0.0 to version 8.0.0.0, use `resources/policies/upgrade-snapshots/7.3.0.0-to-8.0.0.0.SNAPSHOT`).
5. Merge the partial snapshot.
6. Merge conflicts may occur where objects have been updated. If you have not modified the objects in conflict, you may safely select **Keep Snapshots**.

PingDataGovernance Policy Administration Guide

Getting started

Introduction

About this task

This guide introduces the features of the PingDataGovernance Policy Administration GUI (Policy Admin GUI), which is powered by Symphonic®, and provides information about creating access control policies that reflect your business requirements. It also provides a tour of the various concepts that are involved in modeling policies in the Policy Admin GUI.

To get started with the Policy Admin GUI, complete the following tasks:


Steps

1. Log in to the Policy Admin GUI – In demo environments, you can use the default user name `admin` and the password `password123`.
2. Create a branch – This branch stores your policies and other entities.
3. Define the Trust Framework – This allows you to define the elements that will form the building blocks of your policies – the WHO, WHAT, WHERE, WHY, and WHEN.
4. Define your policies and policy sets – Build your policies to reflect your business needs.
5. Test polices and policy sets – Verify that your policies correctly implement your business rules.
6. Conduct analysis – Carry out detailed, cross-policy analysis to determine potential conflicts, redundancies, failure impacts, and so on.
7. Commit changes – This creates a *snapshot*, which is an immutable representation of the Trust Framework and Policies at a point in time.
8. Create a deployment package – This creates a file that can be deployed to PingDataGovernance Server instances across multiple environments.

Next steps


After you log in to the Policy Admin GUI, you are prompted to set the branch on which to work. You can create a new (empty) branch, select an existing branch, or import a branch from a snapshot file.

The PingDataGovernance Policy Administration GUI embraces similar principles to general software source control. As such, it begins with the creation of a branch. When you first deploy the Policy Admin GUI, the `Branches` repository is empty, and the system prompts you to create or import a branch. You must complete one of these actions to continue using the product.




SYMPHONIC

A branch must be set in order to continue


 Create a Branch

Branch name

 Select an existing Branch

Branch


Or

 Import a Branch from a Snapshot

Snapshot

Name

To create a new branch, enter a name for your branch in the **Branch name** field in the **Create a Branch** section, and click **Create new branch**.

 Create a Branch

Branch name

Version control

Branches

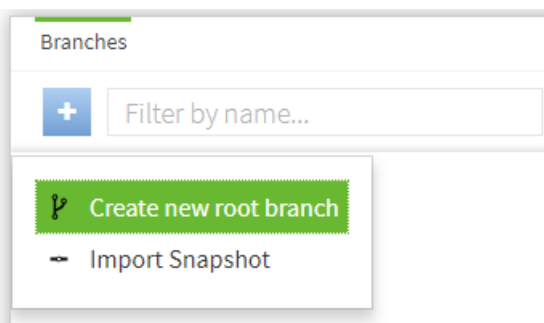
Creating a new top-level branch

About this task

There are two ways in which you can create a new branch.

Note: Branch names must be unique. No two branches in the PingDataGovernance Policy Administration GUI can share the same name.

- From the **Version control** tab, you can create a new root, or top-level, branch:
 - Click **+** and select **Create new root branch**.
 - Fill in the **Name** field.
 - Click **Create new branch**.



- You can also create a new branch from the startup screen displayed when you log in to the Policy Admin GUI.

Importing a branch

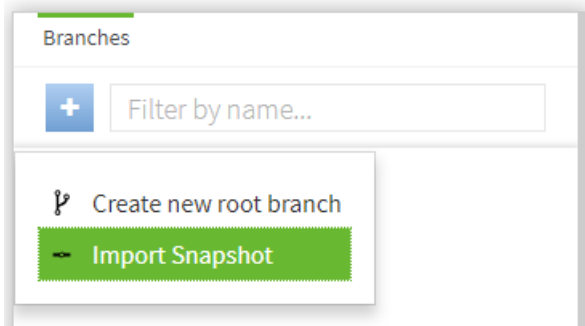
About this task

Branches can be imported from previously exported snapshot files. This is a useful way of sharing and restoring Trust Framework definitions and Policies across users and environments.

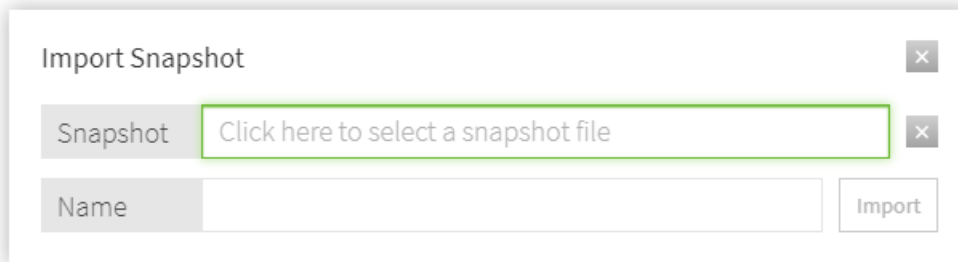
Note: A snapshot file contains all the entities and policies from an existing branch and can be shared like any other file. For more information about creating snapshots, see [Creating a branch from a snapshot](#) on page 174.

Steps

1. Click **+** and select **Import Snapshot**.



2. Select the appropriate snapshot file.



3. Specify a name for the branch.
4. Click **Import**.

Deleting a branch

About this task

Deleting a branch removes the branch, its history, and any snapshots created on it from the system. Because this operation is irreversible, the system prompts you to confirm whether to proceed.

Steps

1. Select the branch to delete.
2. Click **Delete Branch**.

Next steps

To recover data from a deleted branch, load a snapshot that was exported from the branch if one exists. If no such snapshot is available, contact your system administrator, who might be able to recover the deleted branch from a database backup.

Snapshots

Creating a branch from a snapshot

About this task

You can create a branch from a snapshot. This branch shares the history and contents of the branch on which it is based up to the time of snapshot.

Steps

1. Select the snapshot from which to branch.

The snapshot must be a committed snapshot. To branch from the latest uncommitted changes, make certain to commit before proceeding.

2. Click the menu burger and select **Create new branch from commit**.
3. Specify a name for the branch.
4. Click **Save Branch**.

Results

The system creates a new branch with the selected snapshot as the branch-point.

The screenshot shows the 'Commits' section of a software interface. At the top right, there are two buttons: 'Set branch as Merge Source' and 'Set branch as Merge Target'. Below these is a table with columns: 'Options', 'Commit Message', 'Committed on', and 'Creator'. The table contains three rows: 'Uncommitted Changes' (N/A), 'Added risk score policies' (02/08/2018, 14:14:23, Symphonic), and another row (23/07/2018, 11:34:34, SYSTEM). A context menu is open over the second row, listing options: '# Copy ID to Clipboard', '- Export to Snapshot', 'Create new branch from commit' (highlighted in green), 'Select source commit to compare', and 'Select target commit to compare'. At the bottom right of the table area is a 'Save Branch' button with a green download icon.

Partial snapshot export and merging

With the partial snapshot export feature, you can package a subset (partial) of the policies or trust framework entities for export. Then you can import the partial snapshot, either as a new branch (import) or into an existing branch (merge).

Version Control Deployment Packages **Export Partial Snapshot** Merge Snapshot

Partial export

About this task

Partial export lets you build an export snapshot of specifically selected entities from a combination of the Trust Framework, Policy Sets, and the Library set.

The screenshot shows the 'Partial Snapshot' export interface. On the left, under 'Available entities', there is a tree view with a blue '+ Add selection to Snapshot' button. The tree includes 'Policy Manager', 'Bank Root Policy Set', 'Open Banking' (highlighted in green), 'Online Banking', 'Mobile Banking', 'Trust Framework', and 'Library'. On the right, under 'Selected entities', there is a section titled 'Partial Snapshot' with a warning icon and the text 'Please note, all related dependencies will be included automatically'. Below this is a table with columns 'Type', 'Name', and 'include Children'. The table contains one row: 'Open Banking' with a checked 'include Children' checkbox. At the bottom of the table, it says '1 total'. An 'Export' button is located at the bottom right of the interface.

Steps

1. Select the desired items from the list on the left.

2. Click **Add selection to snapshot** at the top of the pane on the left.

This step adds the entity to the selected table on the right. Because all dependencies are included automatically in the exported snapshot, you do not need to explicitly select each individual dependency.

3. Click **Export**.

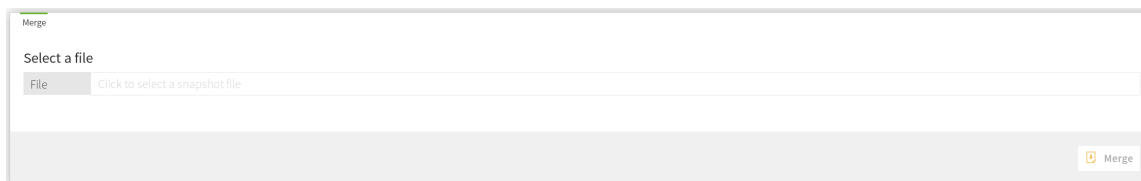
Merging

About this task

Merging a snapshot adds or updates all of the entities into the currently selected branch.

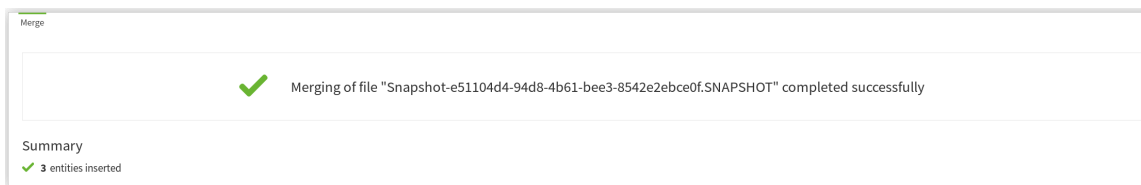
Steps

1. Select the appropriate file.
2. Click **Merge**.



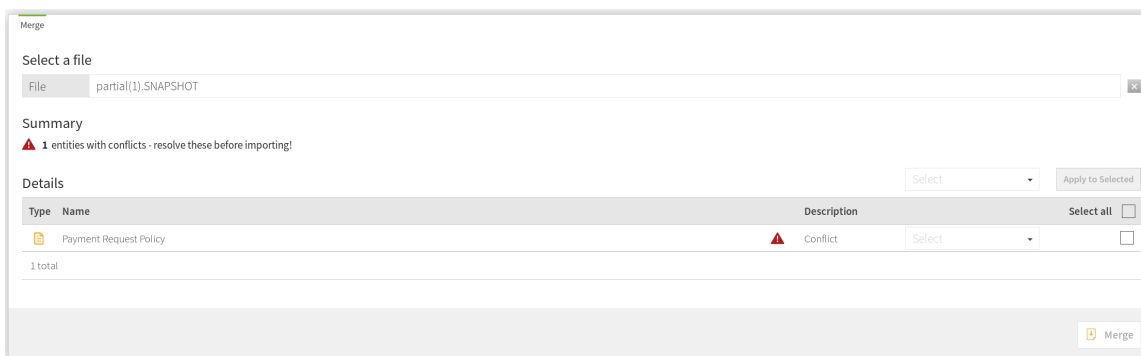
Results

The system displays a **Summary** page that details the result of the merge.

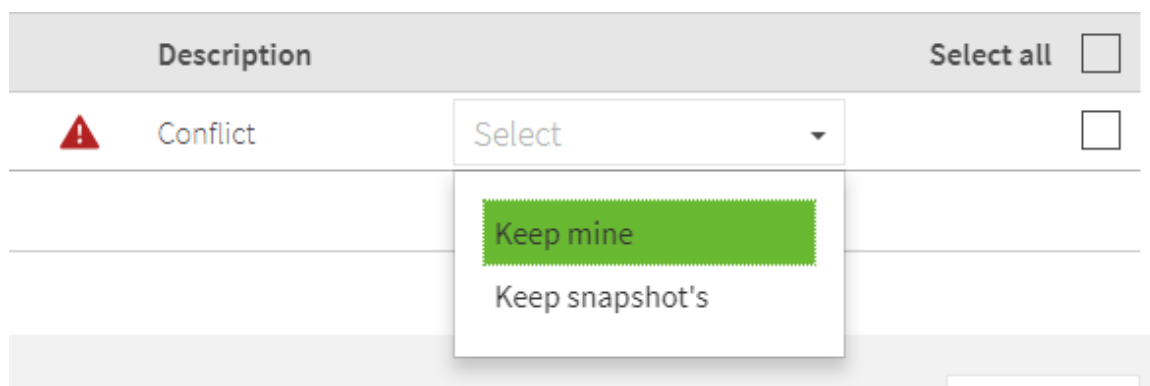


Next steps

In some cases, the merge function detects conflicts that arise when the current branch version differs from the snapshot version of the same entity. For example, this situation might occur if you update one of the merged entities in your current branch and then try to re-merge the snapshot. In such a scenario, the system displays the following **Merge Conflict Resolution** page.



For each conflict that is detected, you can choose whether to keep your local changes or to overwrite them with the changes from the merged snapshot.



After you resolve the conflicts, click **Merge**.

Trust framework

Trust Framework overview

The Trust Framework tool lets you define all the entities within your organizations about which you want to build policies at a later time. Anything that you want to express in your policies must be defined in the Trust Framework. As a result, your policies become strongly typed to the definitions in your Trust Framework, with strict restrictions on intermixing of values with differing data types.

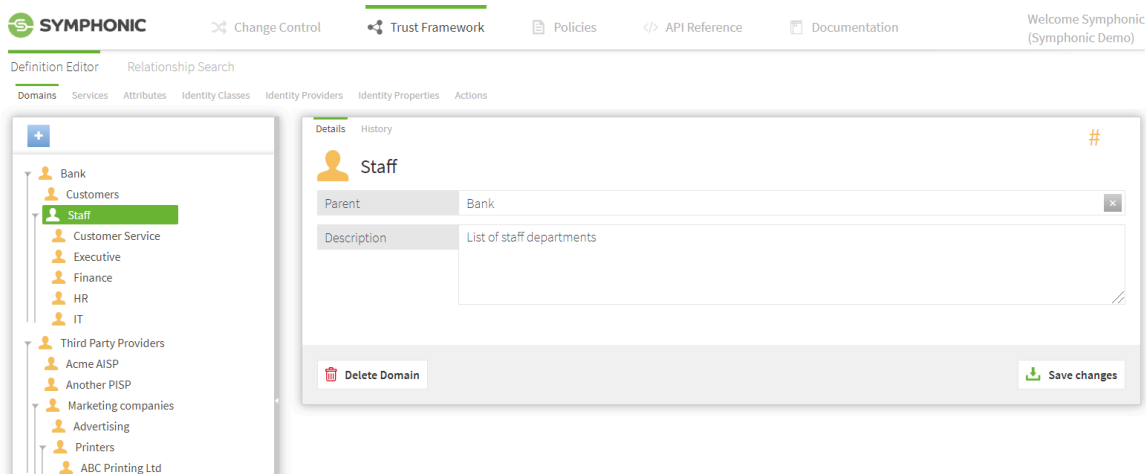
Definitions are broken down into the following types:

- [Domains](#)
- [Services](#)
- [Attributes](#)
- [Actions](#)
- [Identity properties](#)
- [Identity providers](#)
- [Identity classifications](#)
- [Named conditions](#)

Domains (PDP API only)

Use the **Domains** section, which is available only on PDP API-enabled servers, to define the organizational structure, as well as any other organizations with which you intend to interact and, consequently, on which you want to specify authorization policies. We recommend that you keep the domain ontology relatively clean and simple, as it can always be extended later if you choose to get to more granular levels.

You can import these values from your existing organizational directory, such as Active Directory. Make certain that you do not import redundant and unnecessary entities.



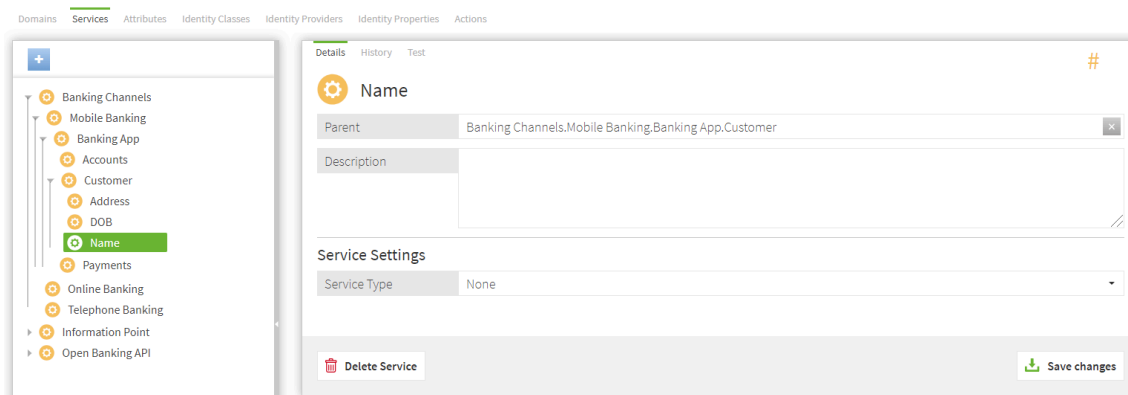
Services

The **Services** section enables the definition of the following types of services:

- The **resources** to which you want to control access (what your policies will protect)
- The **information points** that are used as a source of data for the attributes that comprise policy decisions

Resources

For a resource, define only the top-level fields, such as **Name**, **Parent**, and **Description**. Unless you also plan to also use the service as an information point, leave the **Service Type** as **None**.



Information points

Setting up services as information points makes use of various service connectors. To complete this task, make a selection from the **Service Type** drop-down list and fill in the fields.

Many common settings apply to all service endpoints. When a service is invoked during attribute resolution and returns a value, the response can be mapped to a type, or a processor can be applied either to transform the response or to extract a specific part of it.

The following processors are supported:

- JSON Path
- XPath
- Spring Expression Language (SpEL)

This approach is useful when existing services return more information than is required, or return information that must be converted to a different format.

Value Settings

Processor	JSON Path	\$.risk.score	
Type	Number	Secret	<input type="checkbox"/>

Timeout and Retry

Request Timeout (ms)	2000		
Number of Retries	2	Retry Strategy	Fixed Interval
Retry Delay (ms)	1000	Delay Jitter (%)	10

Rate Limits

Concurrent Requests	2	Requests Per Second	1000000
---------------------	---	---------------------	---------

Delete Service
 Save changes

Common settings

The settings in this section apply to all service types.

Request timeout

The number of milliseconds that PingDataGovernance Server waits for the request to complete. If this time elapses before a successful response is received, the request is canceled. If retries are configured, the request is attempted again. If all requests fail to complete in time, the service result is an error that represents the timeout.

Number of retries

If the initial request fails or times out, this value indicates the number of times PingDataGovernance Server attempts the request again. To try the request only once, set this value to zero.

Retry strategy

Options are:

- Fixed Interval (default)

PingDataGovernance Server waits for the retry delay between each attempt to perform a service request.

- Exponential Backoff

PingDataGovernance Server waits for an exponentially increasing amount of time between attempts.

Retry delay

For a fixed interval strategy, this value represents the number of milliseconds that PingDataGovernance Server waits between request attempts.

For Exponential Backoff, PingDataGovernance Server multiplies this value by 2^n , where n represents the number of retries already made. For example, if the retry delay is 1000 and Exponential Backoff is selected, PingDataGovernance Server makes the initial request, then waits 1000ms before making a second attempt, 2000ms before the third attempt, 4000ms before the fourth attempt, and so on.

Delay jitter

This setting is a percentage value that indicates the amount of variability to apply to the retry delay on each attempt. For example, if this value is set to 10%, the delays in the previous example are $1000 \pm 100\text{ms}$, $2000 \pm 100\text{ms}$, $4000 \pm 100\text{ms}$, and so on.

RESTful services

PingDataGovernance Server can perform the following requests to HTTP services:

- GET
- POST
- PUT
- DELETE
- HEAD

Text, JSON, and XML content can be sent and received with such requests.

HTTP authentication is supported by using a simple user name and password, or by using an OAuth2 token. Additionally, you can send custom headers with any request, which you can make dynamically in various ways by interpolating attribute values into various parameters.

Core settings

- URL format – URL for the REST endpoint that is accessed. Attributes can be interpolated anywhere in the URL. Because no escaping of attribute values takes place, make certain that this action is completed in the attribute definition, if necessary.
- HTTP method – Method to send in the HTTP request.
- Content type – Content-Type header to send, which relates to the body of the request.
- Body – Body to send with the request. Attributes can be interpolated anywhere in the body with no escaping.

Authentication

The Authentication picker selects one of the following HTTP authentication types, which correspond to an authorization header that is sent with the request:

- None – Default value that indicates no authorization header is sent.
- Basic – Reveals the choices for attributes whose values are sent as the user name and password of an HTTP request with basic authentication.
- OAuth2 – Reveals a token selector. The selected attribute is sent as the authorization token in an HTTP request with bearer authentication.

Headers

Any number of custom headers can be added to the request. The header names are fixed strings, but their values can be constants or attribute values. To switch between constant and attribute, toggle **C / A**, which is located next to a header value.

Value settings

For a RESTful service, value settings describe the expected response from the request. If no preprocessing of the response is required, leave the **Processor** set to `None` and set the **Type** field to `String` for plain text, or to `JSON` or `XML` for those types.

If the response requires preprocessing, select the required SpEL, XPath, or JSON Path processor and enter the appropriate expression. The **Type** field is set to the type of the result of this expression.

For example, if the RESTful service returns the following JSON body:

```
{
  "id": 123,
  "name": "John Smith"
}
```

and a JSON Path processor is selected with expression `$.name`, then the **Type** must be `String`, and the final value for the **Service** is `John Smith`.

Secret

To mark a service's response as secret and to ensure that the data is never leaked to log files, enable the **Secret** button.

The screenshot shows the 'Service Settings' configuration page. The 'Value Settings' section is expanded, showing the following configuration:

- Processor:** None
- Type:** Number
- Secret:**

Other visible settings include:

- Service Type:** Restful
- URL Format:** https://devicerisk/score
- Http Method:** GET
- Content Type:** application/json
- Authentication:** OAuth2
- Token:** Customer.Credentials
- Request Timeout (ms):** 2000
- Number of Retries:** 2
- Retry Strategy:** Fixed Interval
- Retry Delay (ms):** 1000
- Delay Jitter (%):** 10
- Concurrent Requests:** 2
- Requests Per Second:** 1000000

LDAP services

PingDataGovernance Server can make LDAP queries to resolve attribute values.

Configuration

Many settings are required to configure an LDAP service. A publicly available LDAP service is used as an example.

Host and port

The host name and port number of the LDAP server.

```
Host: ldap.forumsys.com
Port: 389
```

User name or bind DN and password

The user or bind credentials for the LDAP server.

```
Bind DN: cn=read-only-admin,dc=example,dc=com
Password: password
```

Search base DN or LDAP filter

These settings define the LDAP query that is made.

```
Search Base DN: dc=example,dc=com
LDAP Filter: ou=mathematicians
```

Results

Because the result of an LDAP query is converted to an XML document, the service value type must be set to XML. The previous example query results in the following document.

```
<searchResponse>
  <searchResultEntry dn="OU=MATHEMATICIANS,DC=EXAMPLE,DC=COM">
    <attr name="ou">mathematicians</attr>
    <attr name="objectClass">groupOfUniqueNames</attr>
    <attr name="objectClass">top</attr>
    <attr name="uniqueMember">uid=euclid,dc=example,dc=com</attr>
    <attr name="uniqueMember">uid=riemann,dc=example,dc=com</attr>
    <attr name="uniqueMember">uid=euler,dc=example,dc=com</attr>
    <attr name="uniqueMember">uid=gauss,dc=example,dc=com</attr>
    <attr name="uniqueMember">uid=test,dc=example,dc=com</attr>
    <attr name="cn">Mathematicians</attr>
  </searchResultEntry>
</searchResponse>
```

Individual parts or collections of the data can be extracted from the resulting XML document by using XPath processors.

Service Settings			
Service Type	LDAP		
LDAP Settings			
Host	ldap.mybank.com	Port	389
Bind DN or User	cn=read-only-admin,dc=mybank,dc=com	Bind Password	{{ldap-password}}
Use SSL	<input checked="" type="checkbox"/>		
Search Base DN	dc=mybank,dc=com		
LDAP Filter	ou=customers		

Camel services

Overview

In addition to retrieving information from HTTP and LDAP information points, you can retrieve information from any endpoint that the [Apache Camel](#) enterprise integration platform supports. To view the full list of supported systems, go to the [list of Camel components](#) on the Apache Camel website.

Camel components are configured by using a combination of URI, Headers, Body, and Configuration settings. The appropriate values to provide for each setting depend on the component that is used. Refer to the documentation on the Camel website for the particular component that you want to use.

URI

Camel endpoints are identified by URIs. As well as identifying the system, URIs can specify configuration options for components. For information about configuring a URI for the component to which you want to connect, go to the Apache Camel website.

Note: Attribute values can be interpolated into the URI.

Headers

Additional information can be sent to the external information point by using Camel headers. If the component to which you will connect uses headers, you can read more about them in the instructions for your component on the Apache Camel website.

Note: Attribute values can be interpolated into headers.

Body

Some Camel components operate on a message body, which you can provide by using this setting. If the component to which you will connect requires a message body, you can read more about it in the instructions for your component on the Apache Camel website.

Note: Attribute values can be interpolated into the body.

Configuration

Some Camel components require you to configure helper components for them to work. Specify these components by using the [Groovy](#) scripting language to write a *Spring Bean* configuration block. For information about writing such a configuration, go to [Class GroovyBeanDefinitionReader](#).

The Camel JDBC component makes use of the Headers and Body settings, and requires a JDBC data source to be set up in the Camel Configuration setting.

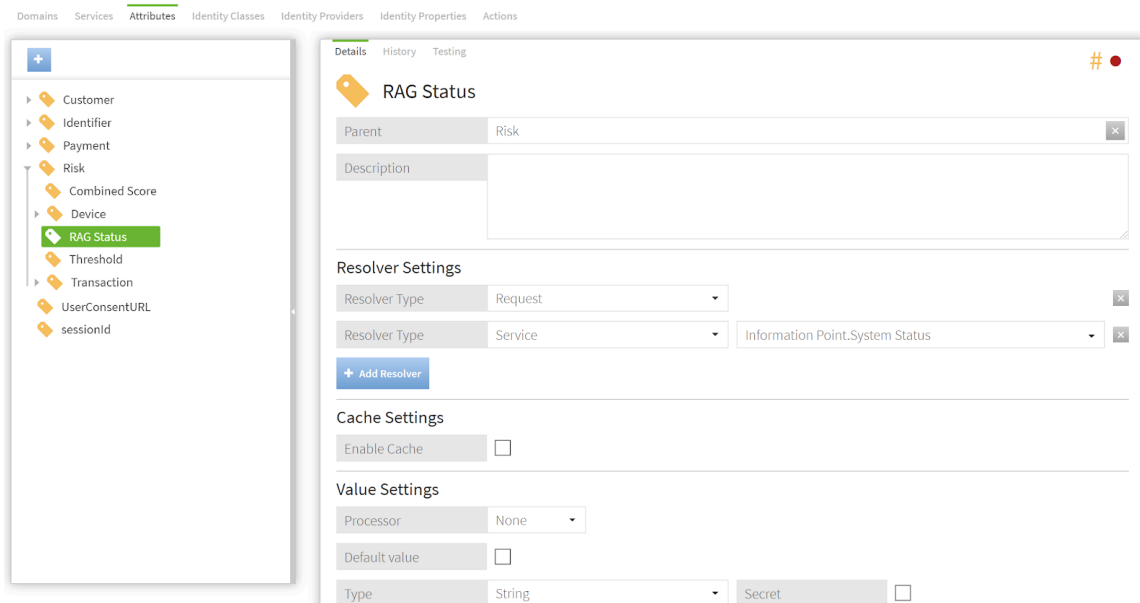
Warning: Attribute values cannot be interpolated into the configuration.

Attributes

Attributes provide the context that enables fine-grained policies. Attributes can retrieve data from multiple information endpoints and can allow for the inclusion of values and results inside the rules of a particular policy.

Consider the manner in which the attributes will be structured, as well as the naming conventions that will be used, so that policy writers and editors can build policies without requiring a deep understanding of the complex underlying data endpoints. The intention is to abstract this complexity and to expose the attributes in business terms that business users and policy builders can understand.

To create an attribute, click **+** on the **Attributes** tab.



Attribute naming

Attributes can be given any name that is unique and does not contain a period (.). To ensure that the attribute can be interpolated, avoid the following characters:

- {
- }
- |

Resolver settings

Attributes are resolved only when an applicable rule requires the value of the attribute to evaluate the conditions of one or more rules. For example, if a rule requires the Risk Score for a customer's device and compares it to a value of High, Medium, or Low, the attribute resolver attempts to resolve the attribute only when it reaches the policy. Depending on the order of the policies, you might not need to resolve all attributes.

Each attribute can have multiple resolver types and is resolved in the order in which it was defined. Attributes can be reordered by dragging and dropping them to the appropriate position.

The following sections describe the various resolver types.

Request

This resolver type looks inside the authorization request itself to determine whether the attribute has been provided by the caller.

Constant

This resolver setting takes a constant value that is defined on the resolver itself. The type and value of the constant must be specified.

Note: As with all other resolved values, constants undergo any value processing that is defined on the attribute. To define a constant that does not undergo value processing, consider using a default value.

Service

This resolver setting uses a **Trust Framework# service endpoint** to invoke the service at runtime and to resolve the attribute. The service might rely on other attributes that are supplied to invoke it.

PingDataGovernance Server handles this process automatically.

Attribute

Attributes can also be resolved from other attributes. This approach is useful when you have attributes that contain multiple pieces of information and you want to create nested or child attributes as subset extracts from them.

For example, the Customer.Name attribute might return the following JSON representation:

```
{ "firstname": "Joe", "middlename": "Bod", "surname": "Bloggs" }
```

In this example, the Customer.Name.Surname attribute could be created to resolve against the Customer.Name attribute and could use a JSON parser to extract only the Surname property of the JSON.

System

The PingDataGovernance Policy Administration GUI provides many of out-of-the-box System attributes that can be used without additional configuration. For example, the CurrentDateTime returns the current system datetime, as the name implies.

Configuration key

Attributes can be resolved against configuration key items. These key-pair values are defined in the `configuration.yml` file as part of the initial PingDataGovernance Server and PingDataGovernance Policy Administration GUI configuration.

Conditional resolvers

Because all resolver types support the ability to add conditional logic, the resolver is invoked only under certain defined conditions. To add a conditional logic builder to a resolver, click **Add Condition** beside the appropriate resolver item.

The screenshot shows the 'Resolver Settings' section of the GUI. It contains two rows of configuration:

- Row 1: Resolver Type is 'Request'. To its right is a '+ Add Condition' button with an 'x' icon.
- Row 2: Resolver Type is 'Service'. To its right is a dropdown menu showing 'Information Point.Credential Datastore', followed by a '+ Add Condition' button with an 'x' icon.

At the bottom left of the settings area is a '+ Add Resolver' button.

The following example implies that the service resolver `Information Point.Credential Datastore` is used only when the attribute `Customer.Status` has a value of `Confirmed`.

The screenshot shows the conditional logic builder interface. It features a dropdown menu for 'Resolver Type' set to 'Service' and another dropdown menu for 'Information Point.Credential Datastore'. Below these is a comparison rule: 'A Customer.Status' followed by a dropdown set to 'Equals' and 'C Confirmed'. At the bottom, there are buttons for '+ Comparison', '+ Named Condition', and '+ Group', along with a 'Drag and Drop any Definition from Toolbox' button.

Attribute caching

PingDataGovernance Server and the PingDataGovernance Policy Administration GUI support caching for attributes. The ability to cache resolved attributes can deliver significant performance gains for PingDataGovernance Server. As a result, we recommend that you carefully consider this concept to ensure optimum configuration.

This section focuses on the individual cache options that can be set at the attribute level.

The cache settings for attributes offer the following approaches:

- Time-based – Allows you to set the duration for which the cache lives (Time to Live) before expiring. If the attribute does not exist in the cache, PingDataGovernance Server resolves automatically by using the appropriate attribute resolvers. After it is resolved, it is added to the cache. All subsequent attribute usages use the cached value until it expires from the cache, which results in another attribute resolution.

- Attribute-based – Allows the application of an additional scope to the cache. This approach lets you attach the value of an attribute to the scope of the cache, like customerId or sessionId. Consequently, the cache is returned only when the scope is matched.

Note: The cache key for a Trust Framework attribute value includes a hash of the values that are required for it to resolve. If one of these values changes, the cache key is invalidated automatically. You can think of this arrangement as an aggregation of scope parameters that guard against inconsistencies between your cached values.

Value settings

The **Value Settings** section of services and attributes lets you dictate the value type. It also provides optional processor steps to transform the resolved value.

The PingDataGovernance Policy Administration GUI supports the following value processors:

- JSON Path
- XPath
- Spring Expression language (SpEL)

JSON Path

JSON Path can be used to extract data from JSON objects. For example, you might want to extract the price fields of all requested items from a service that resolves as follows:

```
{
  "name": "Joe Bloggs",
  "requestedItems": [
    {
      "id": "b5f963fa-111e-49ff-994b-b89a20a2c1d5",
      "price": 125.00
    },
    {
      "id": "84e204dd-44f5-4a84-8e58-972c2a9c80b4",
      "price": 299.99
    }
  ]
}
```

In such a scenario, you can create an attribute that resolves against the service and can use the following expression to set the value processor to JSON Path:

```
$.requestedItems[*].price
```

This attribute can be used in conditions or in further attribute resolution. For more information about JSON Path expressions, refer to [these articles](#) on the creator's website.

XPath

As the XML-equivalent of JSON Path, XPath follows a similar syntax. For more information about XPath expressions, refer to the [XPath tutorial](#) on w3schools.com.

i Important: Support is provided for XPath 1.0 only. Functions that have been added in later versions might be unavailable.

SpEL

The Spring Expression language (SpEL) can be used to perform complicated data processing. With SpEL, expressions are applied directly to the resolved value. For example, you might want to search for a substring that matches the following regular expression:

```
\ [[0-9]*\.[0-9]\]
```

To accomplish the task, set the processor to SpEL, and set the expression as follows:

```
matches (\ [[0-9]*\.[0-9]\])
```

Attribute values can be interpolated into the SpEL expression directly, which is useful if you want to combine multiple attribute values into a single value. To interpolate an attribute, wrap its full name in double curly brackets, as follows:

```
{{Customer.Age}} - {{State.Drinking Age}} >= 0
```

For more information, refer to the official [Spring Framework documentation](#).

Because SpEL can be powerful, you might want to limit the tasks that users can complete with it. For instance, you can configure a list of allowed classes under `core`, `AttributeProcessing.SpEL.AllowedClasses`. The following example whitelists, or allows, `DateTimeFormatter` and `URLEncoder`.

```
core:
```

```
AttributeProcessing.SpEL.AllowedClasses:java.time.format.DateTimeFormatter,java.net.URLEncoder
```

If this configuration option is absent, all `java.lang` classes are blacklisted, or disallowed, for use in SpEL expression, with the following exceptions:

- Byte
- ChronoUnit
- Date
- DayOfWeek
- Double
- Instant
- Integer
- LocalDate
- Long
- Math
- Random
- SimpleDateFormat
- String
- UUID

All other classes are allowed.

If this configuration option is present, all classes are blacklisted unless they are specified explicitly in the previous list of fully qualified classnames.

Default value

An attribute can be given an optional default value in the event that it cannot be resolved. This approach can also be used to encode constant attributes within the Trust Framework by not setting any resolvers and, consequently, always resolving to the default value.

Actions

Actions represent arbitrary values that a typical authorization request might ask to perform on a specific resource, such as view or update. The following actions are typically configured in the PingDataGovernance Policy Administration GUI:

- `inbound-GET`
- `inbound-PATCH`
- `inbound-POST`
- `inbound-PUT`
- `outbound-GET`
- `outbound-PATCH`
- `outbound-POST`
- `outbound-PUT`
- `create`
- `delete`
- `modify`
- `retrieve`
- `search`
- `search-results`

Identity classifications and IdP support

The PingDataGovernance Policy Administration GUI provides the ability to generate smart identity classifications. The purpose of these classifications is to abstract the underlying identity providers (IdPs) from their presumed level of trust. The outcome is that you will be able to build policies that target levels of trust instead of specific IdPs.

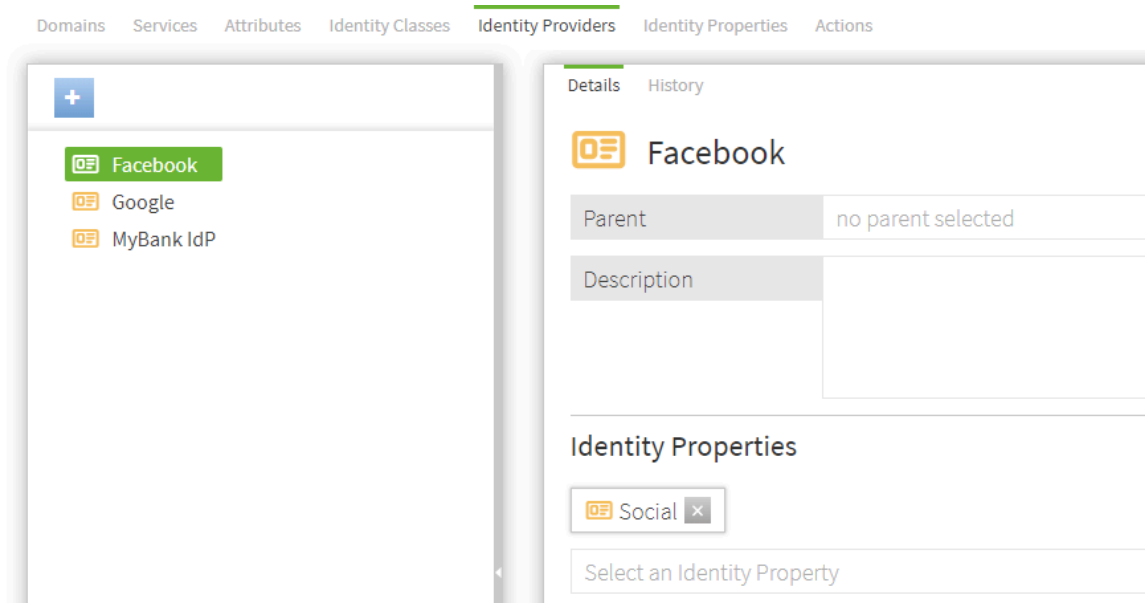
Trust levels are defined by the following distinct parts:

- [Identity properties](#) – Arbitrary properties that can relate to specific IdPs
- [Identity providers](#)
- [Identity classifications](#) – Levels of classifications

The following sections describe these parts in more detail.

Identity providers

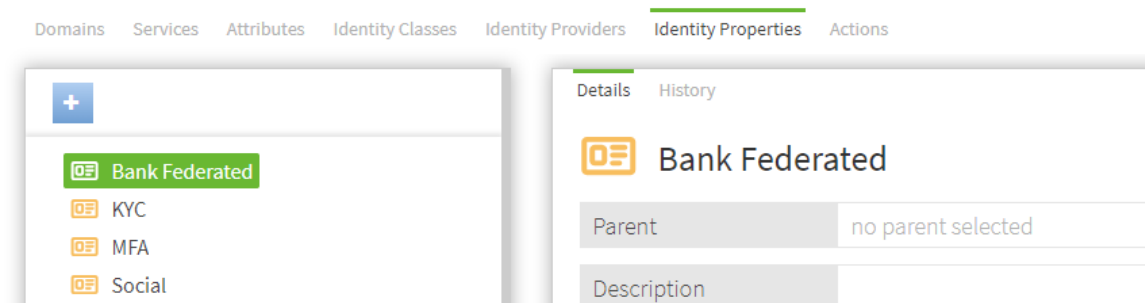
Use the **Identity Providers** page to define different IdPs and to attach identity properties to them.



This step might appear irrelevant when your enterprise expects to use only one or two identity providers, but it provides significant abstraction for more complicated ecosystems in which tens or hundreds of IdPs are participating.

Identity properties

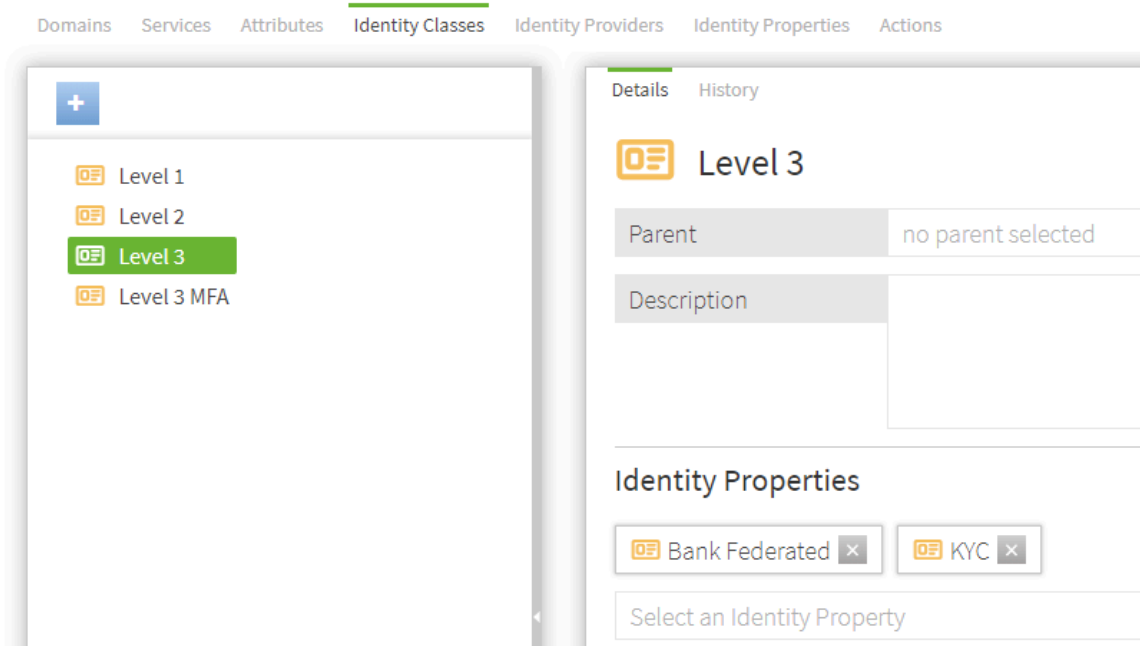
Use the **Identity Properties** page to define objects and elements to attach to specific identity providers.



These elements will be used later to map IdPs to specific identity classification levels. This example features a Social property that can be attached to any social IdP.

Identity classifications

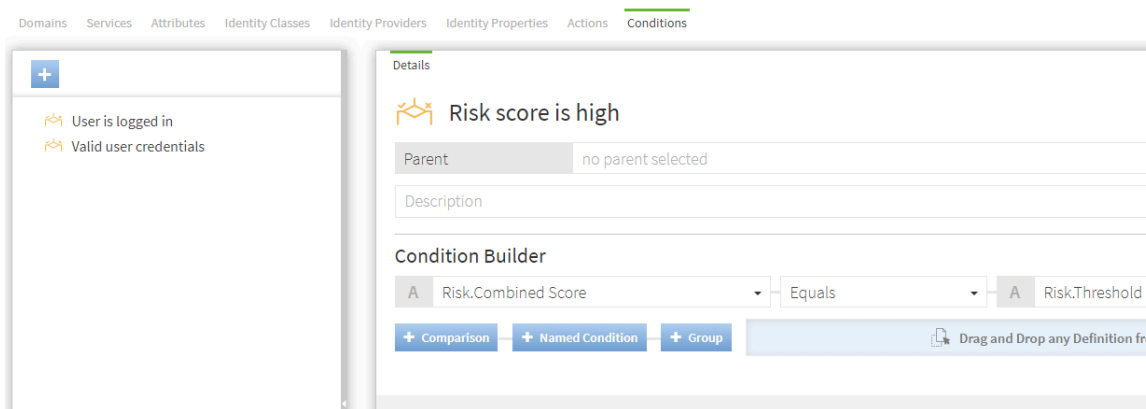
Use the **Identity Classes** page to create different levels of classification.



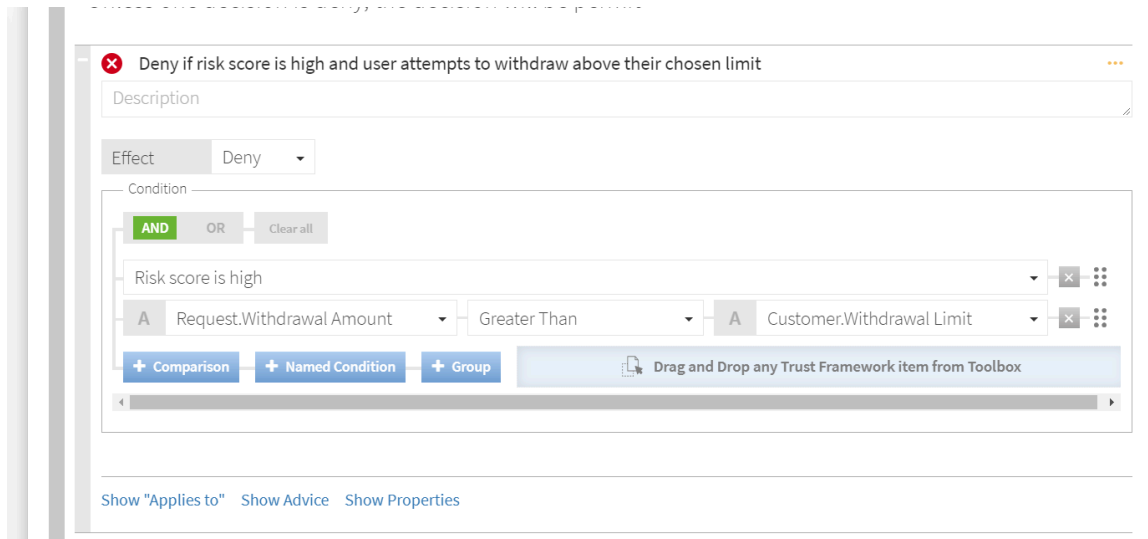
For each classification level, attach the properties that an IdP must have to be considered inside the level.

Named conditions

Named conditions provide the ability to create reusable conditional logic that helps abstract some of the logical complexity from the people who build the policies.



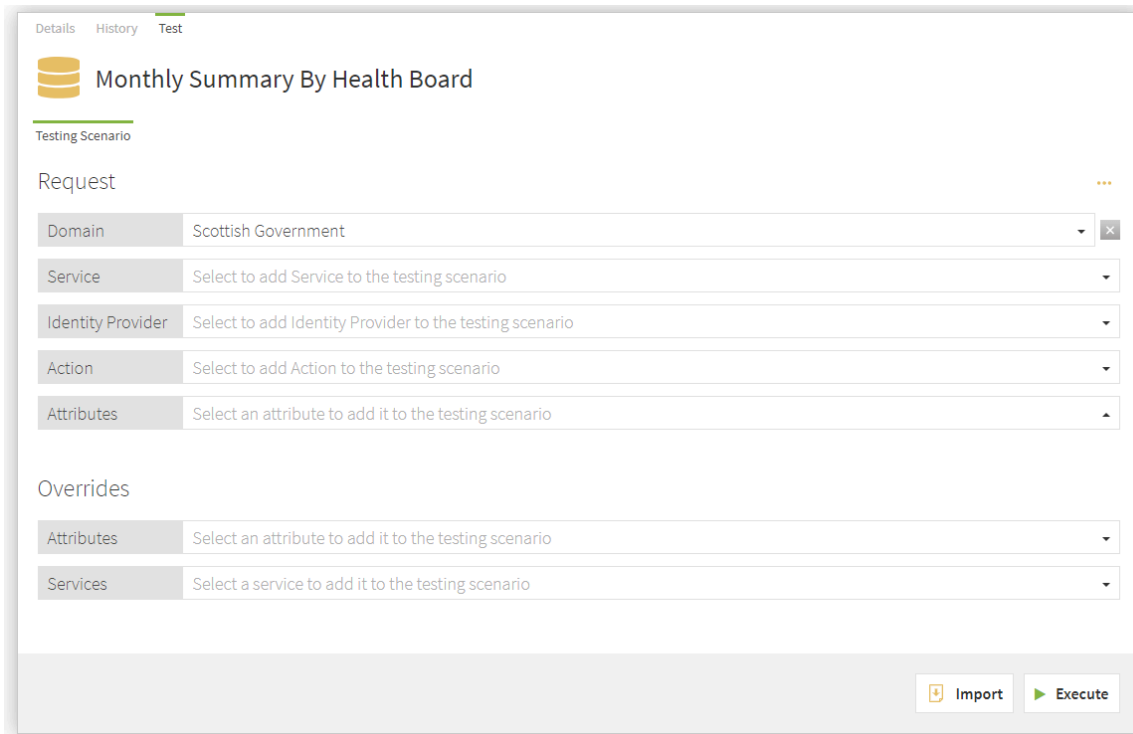
Named conditions also provide an effective way to minimize repetition throughout policies. Policy builders remain able to create their own conditions, which can coexist with the named conditions.



Further, named conditions can be used to replace entire conditions, and can also function as components of more complicated condition expressions. To add a named condition within the condition builder, click **+ Named Condition**.

Testing

The PingDataGovernance Policy Administration GUI provides testing capabilities for applicable definition types. To prepare a test request, select a definition of type attribute or service and go to the **Test** tab.



To form a request, select the following main elements:

- **Domain**
- **Service**
- **IdP**
- **Action**

- **Attributes**

If the information endpoints that your attribute resolvers require are running, click **Execute**. If your endpoints are not running or are otherwise unavailable, as is often the case in development, use the **Overrides** section to provide stubbed values for attributes and services that might be required during the evaluation process. This step overrides the attribute and service resolution, and uses the specified values instead.

After the request is evaluated, you are presented with the following set of result tabs:

- **Request** – Shows the actual JSON request that is sent to the decision engine.
- **Response** – Contains the complete (high verbosity) response for the decision.
- **Output** – Provides a summary of the decision.
- **Attributes** – Contains an expandable list of all attributes that are executed as part of the test.
- **Services** – Contains an expandable list of all services that are executed as part of the test.

Policy management

Policy management overview

The Policy Manager provides the tools for implementing fine-grained and dynamic, access-control policies, allowing you to govern the manner in which your organization's services and data are used.

Use the Policy Manager to create policies that answer the question, "Should this resource-access request be permitted or denied"? In a traditional role-based access control (RBAC) system, this question can be rephrased as, "Who is the user making the access request, and have they been assigned a role that is permitted access to the resource?" Although you can model such a policy, the PingDataGovernance Policy Administration GUI functions essentially as an attribute-based access-control (ABAC) system. In such a system, the question can be rephrased as, "Given the facts that I know about the user, the resource being accessed, what the user wants to do with the resource, how sure I am the user is who she says she is, and any other pertinent facts about the world at this point in time, should the user's access request be permitted, and must anything else be done in addition to permitting or denying access?"

That length of that question speaks to the inherent power of the Policy Administration GUI. Fortunately, the Policy Manager makes harnessing this power quite straightforward.

Policy sets, policies, and rules

A typical enterprise-level organization might impose hundreds or thousands of conditions and constraints around access control. Such constraints comprise the business rules that define the circumstances under which certain resources can be accessed.

These rules can be grouped together naturally, which allows people to reason about them without focusing on all of them at once. For example, a set of policies around authentication might require a user to authenticate to a certain level before they can access a certain resource. Another set of policies might gather together all of the business rules around accessing the resources of a particular business unit. Yet another set of policies might define the audit processes that are triggered whenever access to a set of restricted resources is attempted.

This structure is inherent in the problem domain of resource-access control and is reflected in the PingDataGovernance Policy Administration GUI by three types of entities – policy sets, policies, and rules – and the relationship between them. This section examines these entity types, discusses how they are composed together, and provides an overview of their properties.

Policies and policy sets

To view the Policy Manager, click **Policies** on the main navigation bar.

The screenshot shows the Symphonic Policy Administration interface. At the top, there is a navigation bar with the Symphonic logo and several menu items: Change Control, Trust Framework, Policies (highlighted), API Reference, and Documentation. Below this, there are sub-menu items: Policy Editor, Relationship Search, and Log Visualiser. The main interface has three tabs: Policies (selected), Library, and Toolbox. A search bar with a plus icon and the text 'Filter by name...' is located below the tabs. The main content area displays a tree structure of policy nodes on the left side, with a vertical arrow indicating the hierarchy. The nodes are:

- Bank Root Policy Set
- Open Banking
- Online Banking
- Allow access to authenticated users
- Require strong authentication for transactions
- Require transaction authorisation when risk threshold exceeded
- Mobile Banking

Existing policy nodes are organized in a tree structure within the navigation panel on the left side of the page. We recommend adding a root policy set to contain all other policy sets. This tactic is useful when you build a deployment package from the entire policy tree.

Creating policies and policy sets

1. On the policy navigation panel, click the **Policies** tab.
2. Click **Creation** (blue button) to open a popup window.
3. Select and start the creation process of a policy or policy set, as appropriate.

Policies and policy sets can be named anything you like. However, we recommend that you use relevant and contextual names, especially as the policy tree grows larger and more complex. When naming policies, consider the business rule that they are trying to model and verify that the names check adequately represent the operational policies of the organization.

Inspector

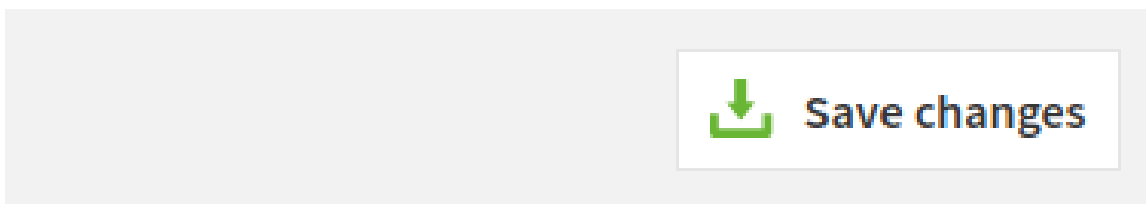


My Basic Policy

Disabled

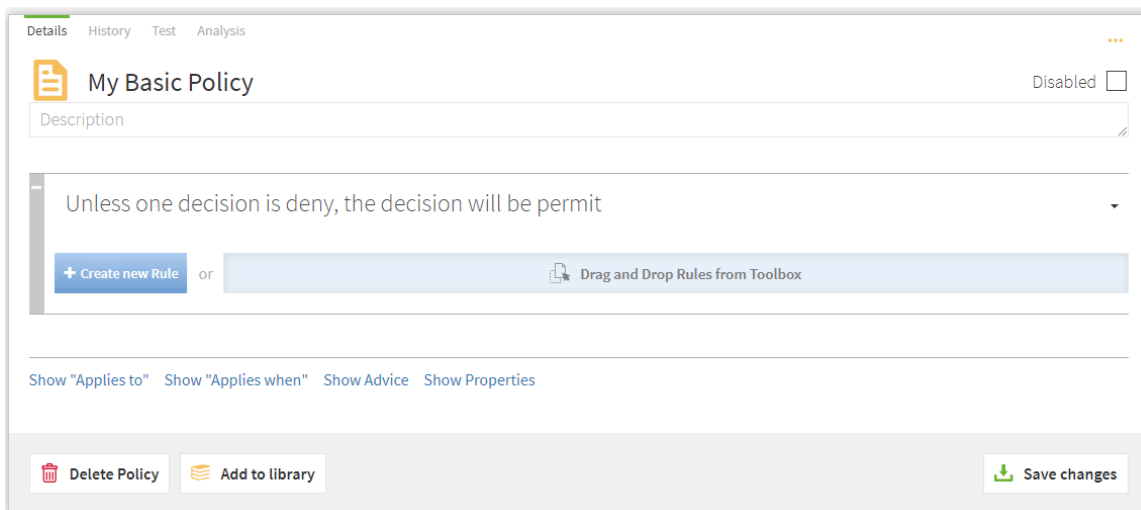
In this example, the policy is named `My Basic Policy`. The red dot in the upper-right corner signifies that, because the name has been changed, the policy contains unsaved changes. If you try to leave the page, a popup window prompts you to save your changes.

To save a policy, click **Save Changes**.



Adding targets to a policy

After you change the name of a policy, add targets to identify the requests to which the policy applies.



To expand the target section in the Policy, clicking **Show "Applies to"**.



A target defines a set of access requests to which a policy applies. To make a policy apply for all requests to a certain database, for example, add the database domain as a target. Because no targets have been attached yet to the policy in this example, it applies to all requests.

Policies Library **Toolbox**

Filter by name...

Library

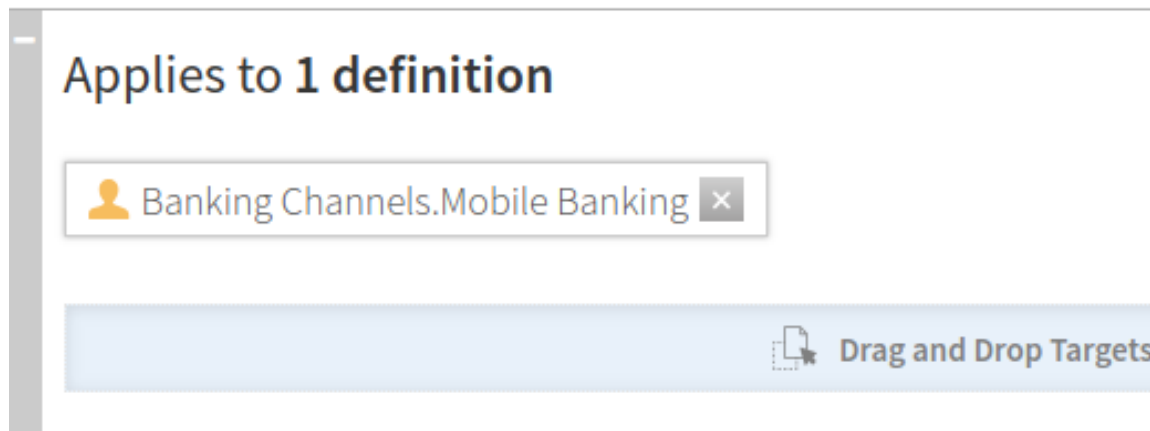
- ▶ ✓ Rules
- ▶ ⦿ Targets
- ▶ 💬 Advice

Trust Framework

- ▶ 👤 Domains
 - ▶ 👤 Banking Channels
 - ▶ 👤 Mobile Banking
 - ▶ 👤 Online Banking
 - ▶ 👤 Third Party Providers
- ▶ ⚙️ Services
 - ▶ ⚙️ Information Point
 - ▶ ⚙️ Internal

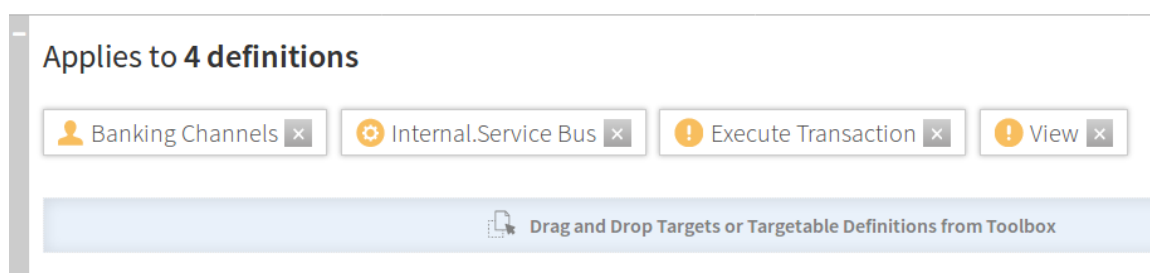
Use the target section in conjunction with the toolbox in the left panel, which displays the elements created in the Trust Framework. Drag the appropriate domains, services, identity classes, and actions from the toolbox to the target section on the policy. For example, to target **Mobile Banking** requests, drag that domain to the target section. To target all banking groups, add the **Banking Channels** domain, which is the parent of the **Online Banking** domain as well as the **Mobile Banking** domain. Because the top level is also a target, this step adds a total of three targets.

In the following image, the **Mobile Banking** domain has been dragged to the target section in the Policy.



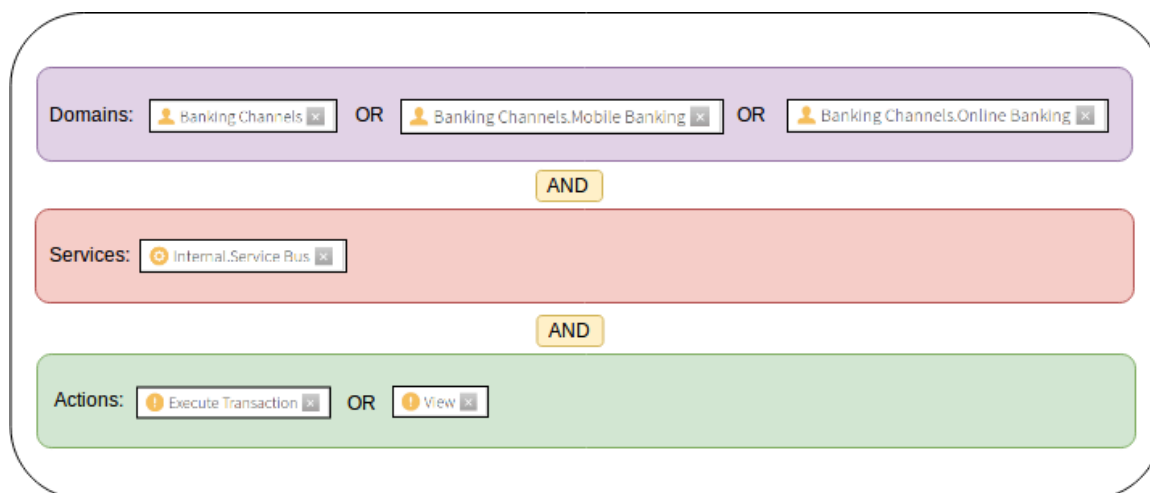
The target is displayed as a label that can be removed by clicking **X**.

In the following image, four definitions have been added as targets to the policy.



This example features three domains because the **Banking Channels** definition is the parent of the other definitions. Logically, one of the channels is selected by applying an OR operation within the definition type.

The following graph shows the manner by which the group of targets is evaluated.



Conditional targets

Conditional targets extend the capability of the "Applies to" concept, as follows:

- Permit the interweaving of targets with other conditional logic
- Allow standalone logic to determine if and when a policy or rule applies

To enable this functionality, click **Show "Applies When"**.

The following types of conditions can be included in a logical expression:

- Attribute comparison – Allows the comparison of an attribute with another attribute or with a constant.
- Request comparison – Allows the matching of incoming requests by answering questions like, "Is the requested service equal to Banking.Payment?"
- Named condition – Click **+ Named Condition** to add a **Named Condition** drop-down list that displays pre-saved named conditions.

The following image provides an example.

The screenshot shows a configuration window titled "Applies when". At the top, there are logical operators "AND" (highlighted in green) and "OR", along with a "Clear all" button. Below this, three conditions are listed in a vertical stack:

- Condition 1: A (Attribute Comparison) | Risk.Combined Score | Greater Than | C (Constant) | 100
- Condition 2: R (Request Comparison) | Service | Matches | Banking.Payment
- Condition 3: User is logged in

At the bottom, there are buttons for "+ Comparison", "+ Named Condition", and "+ Group". To the right, there is a "Drag and Drop any Definition from Toolbox" button.

To switch between Attribute Comparison mode and Request Comparison mode, click **A** and **R**, respectively, to the left of the comparator.



Advice

Advice is additional information that can be attached to a decision response. It is returned to the governance engine so that, depending on the evaluation response from the policy, the appropriate action can be taken. If a policy is set up to verify the authentication level of a user, and if the policy evaluates that a user does not possess the required access privileges, details can be sent about the reason for denying access.

The screenshot shows a configuration window titled "Advice (1 in total)". It contains the following fields and controls:

- Name:** Invalid authentication level advice
- Code:** invalid_auth_level
- Payload:** Unauthorised customer authentication level
- Applies To:** Deny
- Obligatory:** A checkbox that is currently unchecked.
- + Attributes:** A button to add attributes.
- + Create new Advice:** A button to create a new advice.
- Drag and Drop Advice from Toolbox:** A button to import advice from a toolbox.

To indicate that the final decision applies only if an advice can be fulfilled, advice can be marked as *obligatory*. Typically, the service that calls PingDataGovernance Server handles this responsibility.

Each advice contains the following mandatory fields:

- **Name** – Human-readable label for reference in the Policy Manager.
- **Code** – Identifier that distinguishes between different types of advice.
- **Applies To** – Type of decision to which the advice is attached.

If an advice applies, it is used in the final response if its origin decision contributes to the final result. The decision agrees with every decision between its origin and the top-level policy or policy set.

Advice carries additional data in the form of payloads and attributes, as follows:

- The optional field **Payload** can consist of static or interpolated data.
- The **Attributes** field lets you return a key-value mapping of attributes that might be relevant to the advice.

The following table identifies significant advice properties.

Property	Description
Name	Friendly name for the advice.
Obligatory	If true, the advice must be fulfilled as a condition of authorizing the request. If PingDataGovernance cannot fulfill an obligatory advice, it fails the operation and returns an error to the client application. If a non-obligatory advice cannot be fulfilled, an error is logged, but the client's requested operation continues.
Code	Identifies the advice type. This value corresponds to an advice ID that the PingDataGovernance configuration defines.
Applies To	Specifies the policy decisions, such as <code>permit</code> or <code>deny</code> , that include the advice with the policy result.
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 Server supports the following advice types:

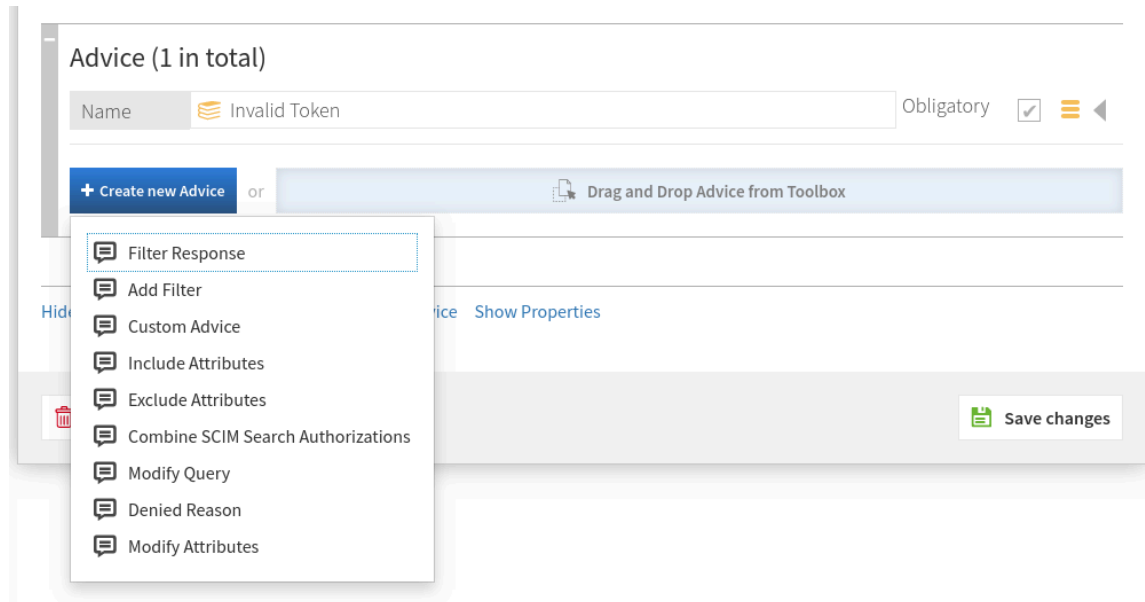
- [Add Filter](#) on page 143
- [Allow Attributes](#) on page 143
- [Combine SCIM Search Authorizations](#) on page 144
- [Denied Reason](#) on page 144
- [Exclude Attributes](#) on page 144
- [Filter Response](#) on page 145
- [Include Attributes](#) on page 145
- [Modify Attributes](#) on page 208
- [Modify Query](#) on page 208
- [Prohibit Attributes](#) on page 146

To develop custom advice types, use the Server SDK.

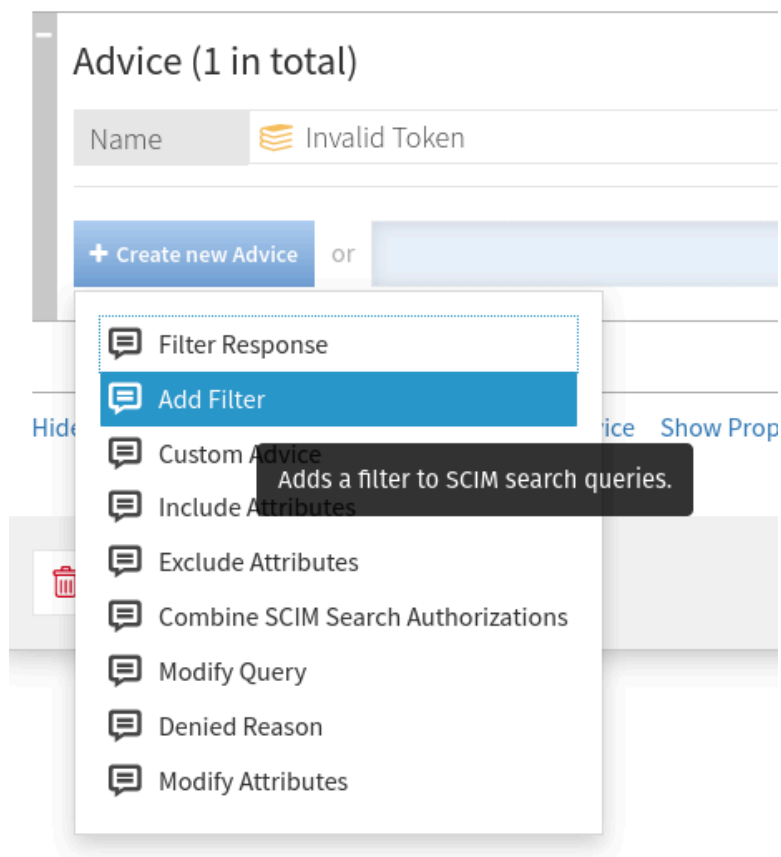
Note: Many advice types let you use the JSONPath expression language to specify JSON field paths. To experiment with JSONPath, use the [Jayway JSONPath Evaluator](#).

Provided advice

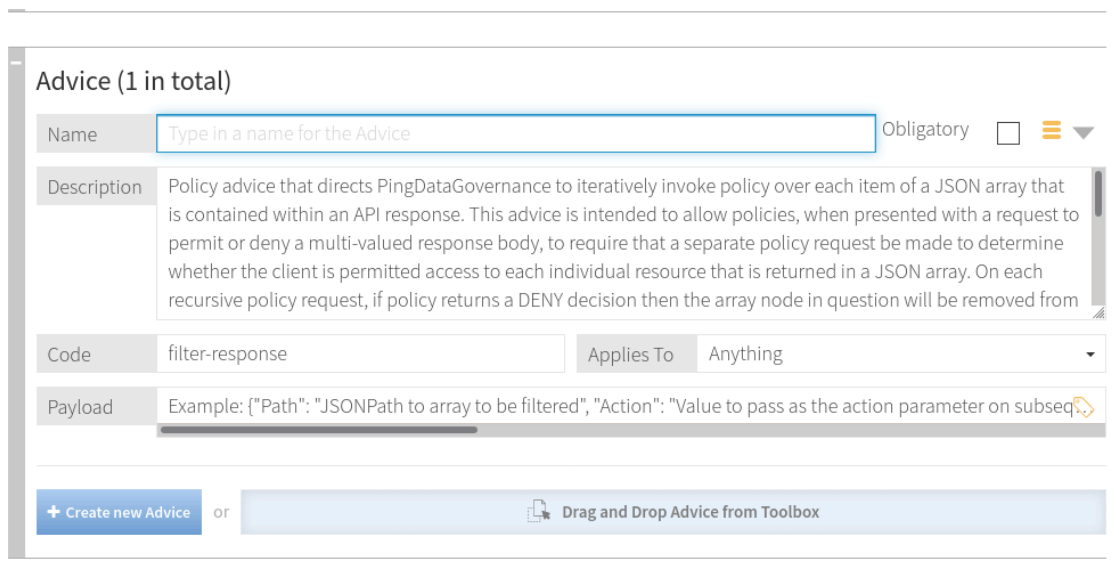
The PingDataGovernance Policy Administration GUI comes with preconfigured advice types that are also configured in PingDataGovernance Server. As a result, policy writers can use this advice out of the box, and PingDataGovernance Server fulfills the advice as documented. To view the full set of provided advice types, click **Create new Advice**.



The documentation for the provided advice types can be viewed from within the Policy Admin GUI. After you click **Create new Advice**, hover over an advice type to view its description.



Selecting an advice type prepopulates the **Description** and **Code** fields and provides an example **Payload** value. Most users replace the example **Payload** value with one that is appropriate for their policy.



For more information, see [Advice types](#).

Custom advice

Policy writers are not limited to the Advice types that are available out of the box in the PingDataGovernance Policy Administration GUI. They can also use a custom advice that leverages the

PingDataGovernance Server SDK. For information about the implementation and configuration of such advice, refer to the *PingDataGovernance Server Administration Guide*.

After the advice is configured properly, it can be used in a policy by selecting **Custom Advice** from the **Create new Advice** drop-down list.

Properties

Properties are used to add metadata to a policy in the format of a key-value pair.

Rules and combining algorithms

Each policy can include multiple rules to produce a permit, deny, indeterminate, or not applicable. To evaluate the overall decision of a policy, a combining algorithm is applied. The default algorithm that is set on a new policy is **Unless One Decision is Deny, the Decision will be Permit**. This algorithm always evaluates to permit unless the decision is deny.

The following list identifies the available combining algorithms and describes their effects:

Combining algorithm	Summary	Details
PermitUnlessDeny	Unless one decision is deny, the decision is permit.	The policy defaults to permit unless any of its children produce the decision deny.
DenyUnlessPermit	Unless one decision is permit, the decision is deny.	The policy defaults to deny unless any of its children produce the decision permit.
PermitOverrides	A single permit overrides any deny decisions.	If any children produce the decision permit, the policy returns permit. If this action does not occur, the policy returns indeterminate if a child produces indeterminate. Otherwise, the policy returns deny if a child produces deny, and returns not applicable if all children are not applicable.
DenyOverrides	A single deny overrides any permit decisions.	If any children produce the decision deny, the policy returns deny. If this action does not occur, the policy returns indeterminate if a child produced indeterminate. Otherwise, the policy returns permit if a child produces permit, and returns not applicable if all children are not applicable.
FirstApplicable	The first applicable decision is the final decision.	The children are evaluated in turn until one produces an applicable value of permit, deny, or indeterminate. If no applicable decisions are produced, the policy returns not applicable.
OnlyOneApplicable	Only one child can produce a decision. If more than one is produced, the result is indeterminate.	The children are evaluated in turn. If at any point two children produce a decision other than not applicable, the policy returns indeterminate. Otherwise, if precisely one child produces an applicable decision, the policy uses it. If no children produce applicable decisions, the policy returns not applicable.
DenyUnlessThreshold	Permit if the weighted average of applicable child decisions meets the threshold; otherwise deny.	The policy's children are assigned weights between 0 and 100. If a child returns permit, the weight is added to a running total. If a child returns deny, the weight is subtracted from the running total. After all children are evaluated, the total is divided by the number of children and is compared against the threshold. If the average is greater than or equal to the threshold, the policy returns permit. Otherwise, the policy returns deny.

Rule structure

Rules contain logical conditions that evaluate to true or false. Each rule can be given an effect of permit or deny. The *effect* is what the rule evaluates to when its child condition or group of conditions evaluates to true. A rule can be set so that, if a condition evaluates to true and the effect is set to deny, the rule evaluates to deny.

Like policies and policy sets, rules can include targets that can be applied to achieve a more fine-grained approach. For example, one rule can target the Mobile Banking channel, and another rule can target the Online Banking channel.

Unless one decision is permit, the decision will be deny

✓ Check authentication level

Effect: Permit

Condition: Customer.Authentication.Level Equals A Customer.Authentication.Level.Strong

+ Condition + Group

Add Targets Add Advice Add Properties

If the condition in this example evaluates to true, the effect is permit.

Testing

The PingDataGovernance Policy Administration GUI provides testing capabilities to evaluate test authorization requests against any or all policy nodes. To specify the node or nodes against which policies are tested, select the root node from the tree on the left side of the page.

In the following example, the evaluation is run against all policies because the root policy set is selected.

Details History Test Analysis

root policy set

Testing Scenario

Request

Domain: Partnership & Local Authority.NHS Board.NHS Fife

Service: Publication.Prescribing Data.Glossary of Terms

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

Import Execute

Select the following main elements to form a request:

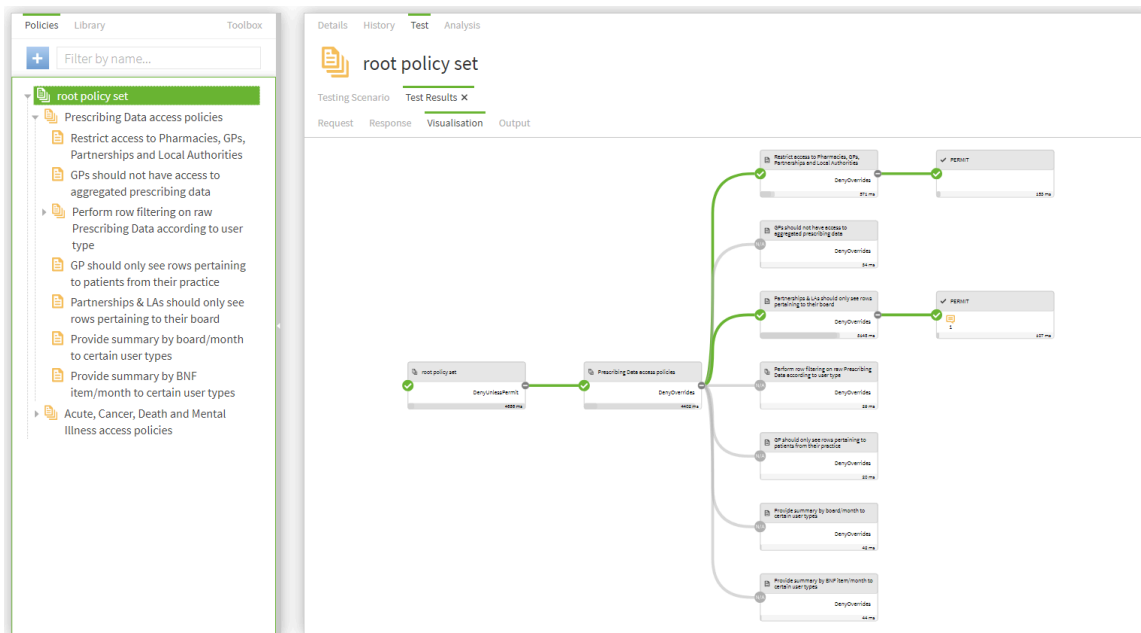
- Domain
- Service
- IdP

- **Action**

If the information endpoints that your attribute resolvers require are running, click **Execute**. If your endpoints are not running or are otherwise unavailable, as is often the case in development, use the **Overrides** section to provide stubbed values for the attributes and services that might be required during evaluation. This step overrides the attribute resolution and uses these values instead.

After a request is evaluated, you are presented with the following set of result tabs:

- **Request** – Shows the actual JSON request that is sent to the decision engine.
- **Response** – Contains the complete, high-verbosity response for the decision.
- **Attributes** – Contains an expandable list of the attributes that are executed as part of the test.
- **Services** – Contains an expandable list of the services that are executed as part of the test.
- **Visualization** – Contains a visual representation of the decision tree.
- **Output** – Provides a summary of the decision.



Analysis

The PingDataGovernance Policy Administration GUI provides full policy-analysis capabilities to generate a report of potential conflicts, redundancies, shadows, and failure-impact assessments, based on the selected policy root node and its children. To execute the analysis across your complete policy landscape, select the root node.

The following key options are available for analysis:

- **Conflicts** – Highlights real policy conflicts, such as the conflict that arises when a policy permits access to a resource while another policy denies access under the same conditions.
- **Redundancy** – Highlights policies that are redundant, based one or more policies, and whose presence makes no difference to the response.
- **Shadows** – Highlights policies that another policy can potentially replace.
- **Global Redundancy** – Similar to **Redundancy** but applies to library policies that are used in multiple locations.
- **Failure Impact** – Highlights information points whose failure might alter the decision.

The screenshot displays the 'Root Policy Set' analysis results. The left pane shows a tree view of policies including 'Root Policy Set', 'Open Banking', 'Payment Request Policy', 'Mobile Banking', and several authentication and risk-related policies. The right pane shows the analysis results for the 'Root Policy Set' under the 'Results' tab. The results are filtered by priority (Important, Ignored, All) and analysis type (Including All). The results table shows five items:

Type	Items Involved	State	Ignore
Conflict	Open Banking RAG Status Check	Unresolved	✘
Conflict	Mobile Banking RAG Status Check	Unresolved	✘
Failure Impact	Require ownership of resource Require SCA for high value transaction Require SCA when device risk threshold exceeded PERMIT if user is authenticated Step up to strong authentication if necessary Trigger transaction authorisation when combined risk score exceeds threshold	Unresolved	✘
Failure Impact	Require ownership of resource Require SCA for high value transaction Require SCA when device risk threshold exceeded Step up to strong authentication if necessary Trigger transaction authorisation when combined risk score exceeds threshold Deny all channels if RAG is RED	Unresolved	✘
Failure Impact	PERMIT if user is authenticated Step up to strong authentication if necessary Trigger transaction authorisation when combined risk score exceeds threshold Deny all channels if RAG is RED	Unresolved	✘

Select the options to analyze and click **Execute**.

Change control

After you build, test, and analyze your policies, commit your changes to move them to a state from which they can be deployed to an instance of PingDataGovernance Server.

Change Control Trust Framework Policies </> API Reference Welcome Symphonic (Bank Demo)

Details #

Bank Demo

Commits

Options	Commit Message	Committed on	Creator
☰	Uncommitted Changes	N/A	N/A

Changes

Entity Type	Name	Operation	Changed on	Creator	DSL	Revert
Rule	RAG	DELETE	14/08/2018, 16:01:02	Symphonic	<>	🔄
Target	Inline target	DELETE	14/08/2018, 16:01:02	Symphonic	<>	🔄
Policy	RAG check for Mobile	DELETE	14/08/2018, 16:01:02	Symphonic	<>	🔄
PolicySet	Root Policy Set	UPDATE	14/08/2018, 16:01:02	Symphonic	<>	🔄

4 items

☰	t	09/08/2018, 15:44:37	Symphonic
☰	SYSTEM BOOTSTRAP	06/08/2018, 17:18:30	SYSTEM

3 items

🗑️ Delete Branch
➡️ Commit New Changes
📄 Save Branch

The **Change Control** page lists previously committed snapshots and all uncommitted changes. To commit new changes, click **Commit New Changes**. This step creates a new snapshot that forms the starting point of a deployment package.

Deployment packages

A deployment package represents a compiled version of the policy tree and is the key element that is deployed to PingDataGovernance Server.

Version Control Deployment Packages

Packages

+

Details

Test Deploy

Branch: Bank Demo

Snapshot: t (by Symphonic at 2018-08-09T14:44:37.690Z)

Node: Root Policy Set

📄 Create Package

Advice types

Advice types overview

This section describes the advice types that are built into PingDataGovernance Server.

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 125.

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: DENY decisions.

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: PERMIT decisions, although [Include or Exclude] Attributes advice cannot be applied directly to a SCIM search.

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 145.

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: PERMIT decisions from Gateway, although Filter Response advice cannot be applied directly to a SCIM search. However, the SCIM service performs similar processing automatically when it handles a search result. For every candidate resource in a search result, a policy request is made for the resource with an Action value of retrieve.

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	Requ	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.

Field	Requ Description
ResourceType	<p>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.</p>

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 might degrade 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: PERMIT decisions, although [Include or Exclude] Attributes advice cannot be applied directly to a SCIM search.

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 145.

Modify Attributes

Advice ID: `modify-attributes`

Description: Modifies the values of attributes in the JSON request or response.

Applicable to: All, although the Modify Attributes advice cannot be applied directly to a SCIM search.

The payload for this advice is a JSON object. Each key-value pair is interpreted as an attribute modification on the request or response body of the request that is being authorized. For each pair, the key is a JSONPath that points to the attribute that requires modification, and the value is the value to set for the attribute. The value can be any valid JSON value, including a complex value like an object or array.

Modify Query

Advice ID: `modify-query`

Description: Modifies the query string of the request that is sent to the API server.

Applicable to: All.

The payload for this advice is a JSON object. The keys are the names of the query parameters that must be modified, and the values are the new values of the parameters. A value can be one of the following options:

- null – Query parameter is removed from the request.
- String – Parameter is set to that specific value.
- Array of strings – Parameter is set to all of the values in the array.

If the query parameter already exists on the request, it is overwritten. If the query parameter does not already exist, it is added. For example, if a request is made to a proxied API with a request URL of `https://example.com/users?limit=1000`, a policy can be used to limit certain groups of users to request only 20 users at a time. A payload of `{"limit": 20}` causes the URL to be rewritten as `https://example.com/users?limit=20`.

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.

REST API

REST API documentation

Swagger documentation is available through the PingDataGovernance Policy Administration GUI and has full testing capabilities. For more information, click **API Reference** in the Policy Administration GUI.