PingDataGovernance

Contents

PingDataGovernance Server Release Notes

PingDataGovernance Server 8.1.0.6 release notes

Critical fixes

This release of PingDataGovernance 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 8.1.0.5 release notes

Critical fixes

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

No critical issues have been identified.

Resolved issues

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

Ticket ID	Description
DS-43288	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. 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 to 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. 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: * When selecting the subject DN for the certificate, we listed a number of common attributes that may 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 will now warn about attempts to add these addresses for inclusion in the subject alternative name extension.

Ticket ID	Description
DS-44316	Reduced the JVM memory requirements for many command line tools. This avoids memory pressure when multiple tools, such as a scheduled collect-support-data task, are run concurrently to the server process. For most tools, the initial heap size has been reduced to 128 MB, and for certain tools the maximum heap size has capped at 512 MB. On systems with larger amounts of memory, these tools previously were allotted unnecessarily large heaps. The maximum heap size has not been reduced for any tool that especially benefits from having more memory.

PingDataGovernance Server 8.1.0.4 release notes

Critical Fixes

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

No critical issues have been identified

Resolved Issues

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

Ticket ID	Description
DS-43288	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. 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 to 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. 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: * When selecting the subject DN for the certificate, we listed a number of common attributes that may 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 will now warn about attempts to add these addresses for inclusion in the subject alternative name extension.

Ticket ID	Description
DS-43305	Increased the maximum number of RDN components that a DN may have from 50 to 100.
DS-43480	Updated the system information monitor provider to restrict the set of environment variables that may 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 may include credentials, secret keys, or other sensitive information in environment variables, and that should not be exposed in the monitor. The server will now only include 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.
DS-43632	Fixed an issue where the "format" field is omitted from the list of operational attribute schemas in the Directory REST API.
DS-43817	Fixed an issue where Directory Server sometimes reports erroneous warnings about duplicate jar files.

PingDataGovernance Server 8.1.0.3 Release Notes

Upgrade Considerations

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-as-Code (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide*.

For more considerations, see *PingDataGovernance Server 8.1.0.0 Release Notes* on page 13.

Resolved Issues

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

Ticket ID	Description
DS-40650	Updated the **`collect-support-data`** tool so you can specify how much data should be captured from the beginning and end of each log file to include in the support data archive. You can also specify the capture size when invoking the tool through an administrative task, recurring task, or extended operation.
DS-42527	Fixed an issue that could cause an exception when creating a resource in SCIM 1.1 using certain types of DNTemplate.
DS-42609	Fixed an issue in which the Directory REST API could fail to decode certain credentials when using basic authentication.

PingDataGovernance Server 8.1.0.2 Release Notes

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 the 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: 7.2.1.0
  - 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 when connection symmetry is achieved.
  - Fixed in: 7.2.1.0
  - 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.
  - Fixed in: 7.0.1.3
  - Introduced in: 7.0.0.0
  - Support identifiers: DS-38897 DS-38908

Upgrade Considerations

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-as-Code (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide*.

For more considerations, see *PingDataGovernance Server 8.1.0.0 Release Notes* on page 13.

Resolved Issues

The following issues have been resolved with this release of the Data Governance 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.
DS-41964	Fixed an issue with the **`manage-profile`** tool where files in a server profile's `dsconfig/` directory without a `.dsconfig` extension could cause failures in **`manage-profile replace-profile`** when validating updated **`dsconfig`** files.
DS-42456	Fixed an issue where POLICY REQUEST-SKIPPED messages were being logged when response processing was not skipped by the Gateway, rather than when it was skipped.
DS-42687	Upgrade to Jetty 9.4.30.

PingDataGovernance Server 8.1.0.1 Release Notes

Upgrade Considerations

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-as-Code (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide*.

For more considerations, see *PingDataGovernance Server 8.1.0.0 Release Notes* on page 13.

Critical Fixes

This release has no critical fixes.

PingDataGovernance Server 8.1.0.0 Release Notes

PingDataGovernance 8.1.0.0 Release Notes

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

- 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: 7.2.1.0
    - 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: 7.2.1.0
    - 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: 7.0.1.3
    - Introduced in: 7.0.0.0
    - Support identifiers: DS-38897 DS-38908

Upgrade considerations

Keep in mind the following important considerations for upgrading to this version of PingDataGovernance Server.

General

- PingDataGovernance 8.1.0.0 uses a new policy request format that requires changes to the Trust Framework.

  If you are using policies intended for a previous release, you can continue to use your existing policies by setting the `trust-framework-version` property of the Policy Decision Service to `v1`. If you upgrade your server using the **`update`** tool, this property is set for you automatically.

  The `v1` format is deprecated, however, and you are strongly encouraged to update your Trust Framework as soon as possible. To do this, load your existing policies in the Policy Administration GUI and apply the Trust Framework changes by going to **Branch Manager**# **Merge Snapshot** and selecting the `resource/policies/upgrade-snapshots/8.0.0.0-to-8.1.0.0.SNAPSHOT` file included with the server. Then, configure PingDataGovernance Server to issue policy requests using the new Trust Framework by setting the `trust-framework-version` property of the Policy Decision Service to `v2`.
- If you are upgrading to PingDataGovernance 8.1.0.0, an updated version of the Policy Administration GUI is required.
- The PingDataGovernance Policy Administration GUI no longer uses the UNIX environment variable PING_HOSTNAME. Instead, server administrators should use PING_EXTERNAL_BASE_URL to specify both the domain and the port. For more information, see the *PingDataGovernance Server Administration Guide*.

Policy processing and advice

- The Allow Attributes advice and the Prohibit Attributes advice have been removed and can no longer be used. Requests involving policies that refer to these advice types will fail.
- The `HttpRequest.Headers` policy request attribute is not available starting with Trust Framework version v2. It has been replaced by the `HttpRequest.RequestHeaders` and `HttpRequest.ResponseHeaders` policy request attributes. Update existing policies or Trust Framework entities that refer to `HttpRequest.Headers` to refer to `HttpRequest.RequestHeaders`.
- SCIM 2 requests now include the resource type in the service value during policy processing. For example, for a SCIM 2 request that affects the "Users" resource type, the service value will now be "SCIM2.Users" instead of "SCIM2". Existing policy rules or targets that rely on an exact equality match for "SCIM2" must be updated. For example, a condition of "Service Equals SCIM2" would need to be updated to "Service Matches SCIM2".
- For security, by default, the policy engine's SpEL processor now invokes Java classes only in the `allow-list` presented in the *PingDataGovernance Server Administration Guide*. To use other classes, add a key to the `core` section of the Policy Administration GUI's configuration called `AttributeProcessing.SpEL.AllowedClasses` with a list of the classes to include. If you are using embedded PDP mode, add a policy configuration key of the same name to the PingDataGovernance Server configuration.

PDP API

- The XACML-JSON PDP API now requires a different request format. With this new format, you can make multiple decisions using a single HTTP request. In addition, the response format is now compliant with the XACML-JSON specification. The 8.0 PDP API request format is no longer supported. For more information, see the *PingDataGovernance Server Administration Guide*.

Peer setup and clustered configuration

- 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-as-

Code (IaC). For more information about server profiles, see the *PingDataGovernance Server Administration Guide*.

- If you have upgraded a server that is in a cluster (that is, has a cluster name set in the Server Instance configuration object) to version 8.1, you will not be able to make cluster configuration changes until all servers with the same cluster name have been upgraded to version 8.1. If needed, you could create temporary clusters based on server versions and modify each server's cluster name appropriately to minimize the impact while you are upgrading.

What's new

These are new features for this release of PingDataGovernance Server:

- Updated the Policy Administration GUI for common tasks during policy development. Now the GUI shows decision trace graphs for the most recent policy decisions, including their attributes. Also, administrators can reuse and chain together attribute processors as well as add attribute processing as an additional step to attribute resolution. Combined, this greatly improves the capabilities of attribute processing while removing any clutter of intermediate attributes in the Trust Framework.
- Added more actions for fine-grained enforcement on API and SCIM requests and responses. Using the `modify-headers` advice, now policy can modify an API's request and response headers. Using the `regex-replace-attributes` advice, now policy can search and replace known or potentially sensitive values or value patterns within requests and responses.
- Updated the core attributes used in policy decisions for SCIM and API transactions to add use cases, simplify policy testing, and improve performance. Added attributes for the raw OAuth2 Access Token and the client's IP address. Also, you can mock all `HttpRequest` child attributes individually during policy testing in the Policy Administration GUI. This avoids the complexity of testing with a large, complex `HttpRequest` mock object.
- Improved support for highly automated or orchestrated environments that provide auto-healing and auto-scaling. A new, simple HTTP status endpoint now reports overall instance health and availability to a cluster orchestrator like Kubernetes or to a network load balancer like AWS Network Load Balancer. You can determine overall instance health through the configuration of any combination of internal monitoring gauges and thresholds.
- Updated the Policy Administration GUI to support single sign-on with other OpenID Connect Providers besides PingFederate.
- Changed the Policy Decision Point API to support batches of requests and decision responses. Previously, you could externalize business logic from non-API use cases, like legacy web applications, using the PDP API, but only one decision at a time. For better performance, now an enforcement point can submit a batch of requests and receive a batch of decision responses.
- Added TLS security options for REST and LDAP Trust Framework Services that give more flexibility in preproduction environments and more security in production environments. Now administrators can relax TLS certificate checks, configure specific certificate trust, and provide client certificates for full mutual TLS security.
- Improved the Policy Administration GUI setup process to support automated deployments and Docker containers. Now you can use the same deployment scripts or Docker image across different preproduction and production environments by using environment variables to provide instance- and environment-specific values. Also, it is now easier to move the policy database to a persistent volume, thereby retaining policy history across Docker image updates.
- Simplified the Policy Administration GUI upgrade process. Now you can use the **`setup`** tool to update an existing Policy Administration GUI. Doing so automatically updates the policy database, if necessary.
- Several improvements to **`collect-support-data`** to help troubleshoot PingDataGovernance servers when running in containers. To build an archive of support data outside of the container, administrators can schedule the **`collect-support-data`** tool to run as a recurring task and direct its output to a volume mounted to a host directory. To get support data on-demand, administrators can use **`collect-support-data`** on a client system, directing it to run the task remotely and download the results.

Known issues / workarounds

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

- The Policy Administration GUI produces an error when a user attempts to import an exported snapshot that contains references to named value processors.
- Several known issues can occur when you use the Administrative Console with Tomcat 9.0.31. You can resolve these issues by upgrading to Tomcat 9.0.33 or later.
- If you use the **`create-systemd-script`** tool to create a forking **`systemd`** service, the service is stopped by the **`systemctl stop ping-data-governance.service`** command. At that time, you can see the status using the **`systemctl status ping-data-governance.service`** command. That status might contain an indication of failure: `Active: failed (Result: exit-code)`. This error has to do with the way the service exits. It is harmless.

Resolved issues

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

Ticket ID	Description
DS-1046, DS-1204, DS-36547	Added support for remotely invoking the **`collect-support-data`** tool using an administrative task and for invoking the tool on a regular basis as a recurring task. The tool has also been updated to add an `outputPath` argument to allow specifying the path or name to use for the output file.
DS-37829	The **`create-systemd-script`** tool now creates a "forking" service file because Ping services are started by a process (the **`start-server`** script) that is different than the actual service process.
DS-38122	Added support for an extended operation that can be used to invoke the **`collect-support-data`** tool from a remote system and stream the output and resulting support data archive back to the client. The **`collect-support-data`** command-line tool has been updated to support this capability through the new `--useRemoteServer` argument.
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-39076	The Policy Decision Service's `decision-response-view` configuration property now accepts more options to configure the level of detail of records in the policy decision log. For information about the options, see the *Configuration Reference Guide* in the server's docs directory.
DS-39587	The payload formats of the include-attributes and exclude-attributes advices are more permissive. If only one path is needed, you can enter a JSONPath directly; previously, you had to enter an array of strings. For example, both the payload '$.secret' and the payload '["$.secret"]' now remove the "secret" attribute from the response.
DS-39733	A new advice type, `modify-headers`, has been added which can modify both the request headers before the request to the upstream server is made, and the response headers before the response is returned to the client.
DS-39734	The advice type `regex-replace-attributes` is now available. With this advice type, you can search for attribute values based on a regular expression and replace the values in place.

Ticket ID	Description
DS-39791	The "service" value used in policy requests for SCIM 2 operations now includes the SCIM resource type, using the format "SCIM2.<resource type>". For example, if the current operation targets the "Users" resource type, then the service value used in the corresponding policy request will be "SCIM2.Users". This allows policy writers to easily match SCIM 2 requests by resource type.
DS-39798	Fixed a bug in which SEMI_AGGRESSIVE and AGGRESSIVE JVM Tuning Parameters were previously allowed to both be selected.
DS-40119	Fixed an issue where the SCIM attributes `id`, `schemas`, and `meta` could be removed using the Exclude Attributes Advice.
DS-40356	Updated the **`manage-profile`** tool to prevent displaying warnings about offline config changes when starting the server.
DS-40410	Previously, the `HttpRequest` policy request attribute used by DataGovernance to represent an HTTP request or response to the policy engine was serialized as a single JSON object. Each field of `HttpRequest` is now submitted to the policy engine as a distinct policy request attribute. This can improve the policy engine's policy request parsing performance and should also allow policy administrators to more effectively cache and test `HttpRequest` attributes.
DS-40551	Fixed an issue that could prevent some tools from running properly with an encrypted `tools.properties` file.
DS-40567	A license is now always required when using the **`manage-profile replace-profile`** tool.
DS-40577	The PingDataGovernance Gateway no longer retains the changes that policy advice performs on hop-by-hop, resource versioning, or other HTTP headers intended for proxy use.
DS-40649	The Sideband API now accepts prevalidated access token claims provided by an API gateway plugin. This prevents PingDataGovernance Server from duplicating work already performed by the API gateway, potentially improving overall performance in some scenarios. For information about configuring this feature, see the *PingDataGovernance Server Administration Guide*.
DS-40746	Updated the logic that the server uses to select an appropriate default set of TLS cipher suites.
DS-40767, DS-41229	Fixed an issue in which a PingDataGovernance Server could return an HTTP 500 error while logging the policy decision response if using these items: ▪ External PDP mode ▪ The Policy Decision Service with a "decision-tree" decision response view ▪ A policy that uses a service with HTTP authentication 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 Decision Visualizer.
DS-40790	Server SDK extensions for PingDataGovernance Server no longer support the use of an internal ScimInterface. This was previously available using the `getInternalScimInterface()` method of the BrokerContext class.
DS-40806	Fixed an issue that could cause the shutdown process to stall if the server is configured to use TCP to communicate with a StatsD endpoint that has become unresponsive.

Ticket ID	Description
DS-40823	The PingDataGovernance Policy Administration GUI **`setup`** tool now uses relative paths when configuring the Advice JSON schema files.
DS-40889	Fixed an issue with recurring exec tasks where the working-directory attribute was ignored.
DS-40909	All policy files, including snapshots, deployment packages, and upgrade snapshots, are now bundled with both PingDataGovernance Server and the PingDataGovernance Policy Administration GUI in the `resource/policies` directory.
DS-40963	You can now specify a custom OpenID Connect client ID when setting up the Policy Administration GUI.
DS-40980	PingDataGovernance Server no longer prevents a server with an expired license from restarting.
DS-40984	The `include-attributes`, `exclude-attributes`, `modify-attributes`, and `filter-response` advice now support request and response bodies that are JSON Arrays as well as Objects.
DS-41054	Fixed an issue that stopped new extensions from being installed.
DS-41074	Fixed an issue with the way the server reports memory usage after completing an explicitly requested garbage collection.
DS-41086	Updated the StatsD monitoring endpoint to replace any spaces, commas, or colons with underscores, and remove and single quotes or double quotes in sent metric lines. This simplifies parsing of the produced metrics.
DS-41087	The Policy Administration GUI now includes decision evaluation details in `decision-audit.log` by default. With this change, policy writers can visualize decisions by copying and pasting the JSON into the Decision Visualizer.
DS-41115	Setup no longer supports adding servers to a topology with mirrored configuration when run interactively.
DS-41118	PingDataGovernance now provides a gauge called HTTP Processing (Percent) that measures the capacity that the server has to process new incoming HTTP requests.
DS-41126	Updated the server to make the general monitor entry available to JMX clients.
DS-41131	The XACML-JSON PDP API now requires a different request format. With this new format, you can make multiple decisions using a single HTTP request. In addition, the response format is now compliant with the XACML-JSON specification. For more information, see the *PingDataGovernance Server Administration Guide*.
DS-41142	Improved debugging support for Server SDK extensions. If debugging is enabled, the server will now generate a debug message whenever it invokes an extension. For some extension methods that return a value, the server will also generate a debug message with that return value.

Ticket ID	Description
DS-41198	Updated the PingDataGovernance setup process to support joining an existing PingDirectory topology in noninteractive mode. To view the noninteractive arguments for joining a PingDirectory topology, in the output of **`setup --help`**, look in the "Join an Existing Directory Server Topology Options" section. Alternatively, after setup is complete, you can run the **`manage-topology add-server`** command to join a PingDirectory topology.
DS-41201, DS-41615, DS-41693	You can now configure load-balancing algorithms to automatically detect PingDirectory Servers that handle SCIM 2 API requests and token owner lookups made by SCIM Token Resource Lookup Methods. For more information, see the *PingDataGovernance Server Administration Guide*.
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 a 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-41244	The Policy Administration GUI **`setup`** now allows users to define policy configuration keys, trust store details, and other settings in a YAML file using the `--optionsFile` command-line option. For more information, see the *PingDataGovernance Server Administration Guide*.
DS-41261	Fixed an issue with `manage-profile replace-profile` where certain configuration changes for recurring task chains were not being applied.
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.
DS-41265	The embedded PDP now automatically loads new, updated, or deleted policy configuration keys. Previously, any policy configuration key change required you to restart the embedded PDP.
DS-41273	The PingDataGovernance Policy Administration GUI **`setup`** tool now stores certain configuration values, including their default values, as environment variables. For example, the configuration property `server.applicationConnectors[0].port` has the value `${PING_PORT:-443}`. An administrator can override this value by setting a PING_PORT environment variable before starting the Policy Administration GUI. If the environment variable is not present, then the GUI uses the default value of 443.
DS-41289	Fixed an issue that prevented password changes for topology administrators unless their password policy was configured to allow pre-encoded passwords.
DS-41294	Fixed an issue that could cause the PingDataGovernance license to be deleted when joining a PingDirectory topology using **`manage-topology add-server`**.
DS-41301	**Critical:** 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.

Ticket ID	Description
DS-41309	When setting up the Policy Administration GUI in noninteractive mode, you can now specify the base URL of an OpenID Connect provider instead of a hostname and port. With this change, you can use the Policy Administration GUI with OpenID Connect providers that include a customer-specific ID in their URLs, such as PingOne.
DS-41313, DS-41800, DS-41839	PingDataGovernance Server now requires a Trust Framework version to be explicitly specified in the Policy Decision Service configuration. The Trust Framework version configuration determines the format used by the server to generate policy requests and must be compatible with the actual Trust Framework used by your policies. For more information about Trust Framework versions, see the *PingDataGovernance Server Administration Guide*. PingDataGovernance Server will now also raise an alarm and mark the server as UNAVAILABLE if the Policy Decision Service is not ready to evaluate policies and requires further configuration. This will happen, for example, after installing the server for the first time.
DS-41329, DS-41330	Services in the Trust Framework now support more flexible handling of TLS connection security: A service can use a client certificate provided by a key store to handle mutual TLS authentication with an external server; also, a service can use a custom trust store to determine whether the certificate presented by an external server should be accepted. For embedded PDP mode, you can configure the Policy Decision Service with any necessary key stores or trust stores using the `service-key-store` and `service-trust-store` properties, respectively.
DS-41366	Updated the base monitor entry to include `locationName` and `locationDN` attributes if the server is configured with a location.
DS-41396	Updated the Server SDK to add ClientContext and OperationContext methods for obtaining the name and DN of the associated client connection policy.
DS-41400	Updated the file servlet HTTP servlet extension to add support for requiring authentication to access the content. You can limit access to members of a specified set of groups.
DS-41482, DS-41812	Added the `HttpRequest.IPAddress` and `HttpRequest.AccessToken.access_token` attributes to the default Trust Framework. The `HttpRequest.IPAddress` attribute contains the client IP address, while the `HttpRequest.AccessToken.access_token` attribute contains the raw access token provided by the client. The latter can be useful when authenticating to HTTP services from the Trust Framework. Please note that these attributes are only available when using Trust Framework v2.
DS-41659	DataGovernance will now enter an UNAVAILABLE state when all of the LDAP external servers backing the UserStoreAdapter are unavailable.
DS-41731	Fixed an issue that could prevent **`setup`** from generating a self-signed certificate for systems with non-ASCII hostnames.
DS-41751, DS-41752	The values of Trust Framework attributes marked as secret are now recorded to the policy decision log in encrypted form when using embedded PDP mode. In addition, the trace logger now supports two new options for the **`pdp-message-type`** property, "info" and "warning". When these options are enabled, the trace log will record additional details about embedded PDP processing, such as summary information about policy information provider invocations.
DS-41760	The Policy Administration GUI **`setup`** tool now automatically upgrades the policy database if an older version is detected.

Ticket ID	Description
DS-41761	The Policy Administration GUI now allows users to override additional configuration values at runtime using UNIX environment variables for the policy database credentials (PING_DB_APP_USERNAME, PING_DB_APP_PASSWORD) and the file location (PING_H2_FILE). For more information, see the *PingDataGovernance Server Administration Guide*.
DS-41762	Fixed an issue where mirrored subtree polling could produce config archive files that were identical or ignored the configured insignificant attributes list.
DS-41818	Added the `--zip` argument to the **`manage-profile generate-profile`** subcommand, which you can use to generate a zipped server profile.
DS-41820	Added an administrative task that you can use to generate a server profile. Also added a corresponding recurring task that you can use to invoke the task on a regular basis.
DS-41821	Added an instance root file servlet to the default configuration. HTTPS requests to /instance-root by authenticated users with the `file-servlet-access` privilege will be granted access to files within the server instance root.
DS-41823	Fixed an issue where using the `modify-query` advice would cause special characters to be percent-encoded twice.
DS-41850	Servers running on Linux will now log a warning about possible performance impacts if the current memory control group has `memory.swappiness` set to a nonzero value.
DS-41869	Fixed an issue in which the Sideband API would respond with an HTTP 500 error if a request to /sideband/response was missing required subfields of the `request` field.
DS-41908	Added a `disable-response-processing` property to SCIM Resource Types. Use this property to prevent policy calls for "retrieve" after a "create", "modify", or "replace". Also use it to prevent policy calls for "retrieve" or "search-results" after a "search".
DS-41909	Added a `disable-response-processing` property to Gateway API Endpoints. Use this property to prevent outbound policy calls and advice processing for Gateway requests.
DS-41914	PingDataGovernance users no longer need to set the Decision Node when configuring Policy External Servers if they are using policy snapshots provided by or created from those provided with the distribution.
DS-42006	The server now warns the administrator at startup if there are multiple versions of the same jar listed in the classpath and the first one in the classpath is not the newest one.
DS-42033	Addressed an issue where some tools would throw a NullPointerException if a server was configured with a custom global result code map.
DS-42150, DS-42163	Fixed an issue in which the `HttpRequest.RequestURI` attribute was malformed and the `HttpRequest.QueryParameters` attribute was missing during the retrieve phase of policy processing for SCIM 2 searches.
DS-42218	Fixed an issue in which the PingDataGovernance Gateway generated error responses that did not include a correlation ID.
DS-42387	Updated the **`manage-profile generate-profile`** subcommand to exclude files in the `ldif/` and `bak/` directories by default when generating a server profile. If necessary, you can manually include those directories using the `--includePath` argument.

Ticket ID	Description
No ID	In the Policy Decision Point, improved LDAP service executor thread safety and XML interpolation. Also, added support in the HTTP service executor for MA-TLS.
No ID	Fixed an issue in the Policy Decision Point in which services were called twice when an Attribute is marked as secret and used in a Statement.
No ID	In the PingDataGovernance Policy Administration GUI, you can now resolve branch merge conflicts within Version Control. Also, branch merges no longer break when merging a source branch with a deleted entity to a target branch where that entity still exists.
No ID	In the Policy Administration GUI: ▪ The Library has been promoted to a separate subsection of the Policy Manager. ▪ The previous selected entity is now selected when switching back from tabs in Trust Framework. ▪ The use of language in the UI has been cleaned up. Toolbox is now Components, and editor screens are no longer postfixed with Editor. ▪ The GUI can now parse testing responses with failed value processing.
No ID	Fixed an issue in the Policy Administration GUI in which changes were lost when you reordered Saved Rules.
No ID	Fixed an issue in the Policy Administration GUI in which creating a condition on a constant Attribute Resolver would throw an error when selecting an Attribute comparand.
No ID	The Policy Administration GUI now maintains a buffer of recent policy decision requests that you can view in the Decision Visualizer. This view provides useful details about policy decision requests and responses, attribute resolution, and service calls that would otherwise only be available in the server's policy decision log.
No ID	This release of the Policy Administration GUI includes various improvements to processors and attribute resolvers: ▪ You can now give a custom name to attribute resolvers. ▪ You can now give a custom name to all processors, including processors defined within another Trust Framework entity. For example, you can name a processor defined within an attribute. ▪ Anywhere you can define a processor, you can also define a chain of processors.
No ID	HTTP services you define in the Trust Framework no longer perform hostname validation if server certificate validation is set to **No Validation**.
No ID	When you define a new policy in the Policy Administration GUI, the default combining algorithm for the new policy is now **The first applicable will be the final decision**. This algorithm stops evaluating as soon as a decision other than **NOT_APPLICABLE** is reached. The previous default combining algorithm was **Unless One Decision is Deny, the Decision will be Permit**.
No ID	Fixed an issue in which the Policy Administration GUI login page could fail to behave correctly when loaded directly from a URL or through the web browser history.
No ID	Fixed an issue in the Policy Administration GUI in which importing a snapshot would fail with the error message "Unable to decode object".
No ID	Fixed various drag-and-drop issues in the Policy Administration GUI.
No ID	Fixed a policy engine issue in which a validation exception could be thrown if an attribute containing a processor with named attributes was interpolated in an advice payload.

Ticket ID	Description
No ID	The following changes to data types have been made in the policy engine: ▪ The Date Time value type has been deprecated in place of four new types: Date, Time, Date-Time, and Zoned Date-Time. ▪ The Time Period value type has been deprecated in place of two new types: Duration (an amount of time using units such as seconds or milliseconds) and Period (an amount of time expressed in calendar units, such as days or months).
No ID	Fixed an issue in which multiple uses of the system "Current DateTime" attribute resolver in a single decision request or batch of requests did not yield the same value.
No ID	Fixed an issue in the policy engine in which Zoned Date Time values were not represented in textual form using the correct ISO-8601 encoding.
No ID	Fixed a policy engine issue in which converting the string "-1" to a boolean would yield a result of False. This will now cause a type conversion error.
No ID	Fixed a policy engine issue in which converting the number -1 to a boolean would yield a result of False. This will now return True.

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 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	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. 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. 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: ▪ 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	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.
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-43632	Fixed an issue where the "format" field is omitted from the list of operational attribute schemas in the Directory REST API.

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	Fixed an issue that could prevent some tools from running properly with an encrypted `tools.properties` file.

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 getInternalScimInterface() method of the BrokerContext 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 `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-41261	Fixed an issue with **`manage-profile replace-profile`** 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 ha 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	Added a `logging-error-behavior` 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 `lockdown-mode` privilege and only from clients communicating over a loopback interface.
DS-40767, DS-41229	Fixed an issue in which a PingDataGovernance Server could return an HTTP 500 error while logging the policy decision response if using these items: ▪ External PDP mode ▪ The Policy Decision Service with a "decision-tree" decision response view ▪ A policy that uses a service with HTTP authentication 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.
DS-40980	PingDataGovernance Server no longer prevents a server with an expired license from restarting.

Ticket ID	Description
DS-41087	The Policy Administration GUI now includes decision evaluation details in the `decision-audit.log` 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 `cn=Version,cn=monitor` 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. See 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. After 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 previous 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
- 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 policy 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` or `options single-request` in the `/etc/resolv.conf` file.
  - If IPv6 requests specifically are causing issues, add `-Djava.net.preferIPv4Stack=true` to the `start-server.java-args` line in the PingDirectory `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 `cn=Server Status Timeline,cn=monitor` 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, `modify-attributes`, which can modify the values of attributes.
DS-37720	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. 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.
DS-37881	The PingFederate Access Token Validator will now refresh its cached value of the PingFederate server's token introspection endpoint. A new attribute, `endpoint-cache-refresh`, 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 `Resource` 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 `requestURI`, `requestQueryParams`, headers, and `correlationID` attributes of the HTTP request have been made available when constructing an Error Template.
DS-38560	Updated `manage-profile replace-profile` to apply configuration changes directly, when possible. If the new server profile used by `replace-profile` has changed only the `dsconfig` batch files from the original profile, then only the `dsconfig` files are applied. If no changes are detected between profiles, `replace-profile` takes no action. If changes other than `dsconfig` are detected, the full `replace-profile` 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 `manage-profile replace-profile`. 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 `manage-profile setup` 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 `--useDefaultClusterName` 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 `remove-defunct-server` tool's `--ignoreOnline` option. When using `--ignoreOnline` in a mixed-version environment, all servers must support the option.
DS-38968	A new advice type, `modify-query`, 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 groovy-json 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 `groovy-xml` or `groovy-sql`, download the appropriate jar file from *groovy-lang.org* and place it in the server's `lib/extensions` directory.
DS-39253	Added a `replace-certificate` 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 `manage-profile` tool and its subcommands.
DS-39490, DS-39616	The API Endpoint configuration type has been renamed to Gateway API Endpoint. Any existing `dsconfig` scripts referencing an API Endpoint should be updated. For example, a `dsconfig` command of `create-api-endpoint` would need to be changed to `create-gateway-api-endpoint`.
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 `X-VALUE-REGEX`), 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, `ssl-cert-nickname`, 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 `dsconfig` batch files in the manage-profile 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 `--topologyFilePath` argument to the `manage-topology add-server` subcommand.
DS-39671	Updated the `manage-topology add-server` 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 NullPointerException that could occur when recording advice metadata to the policy decision log.
DS-40114	Added a new `cn=Status Health Summary,cn=monitor` 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 `x-fapi-financial-id` to be present. Instead, it now includes the configured `fapi-financial-id` value in policy requests via `Gateway.FapiFinancialId` 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 `config-diff` when writing properties that span multiple lines using the `--prettyPrint` 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. This gauge can generate alarms and affect the server's AVAILABLE or DEGRADED state. You must configure the gauge for it to have any effect; see the DataGovernance Server 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	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. This authorization mode is enabled on a per-request basis. To enable, a policy that targets the 'SCIM2' service and the `search` action must provide an advice with the ID `combine-scim-search-authorizations` but with no payload. The subsequent search response is then authorized using a single policy request with the 'SCIM2' service and the `search-results` action. Any advices returned in the policy results are applied iteratively to each SCIM resource in the result set. For more information, see the PingDataGovernance Server Administration Guide.
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 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: `If-None-Match`, `If-Modified-Since`, `If-Unmodified-Since`, `ETag`, and `Last-Modified`.
DS-40543	Updated `manage-profile replace-profile` 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 `encrypt-file` tool to prevent using the same path for both the input file and the output file.
DS-40771	Added a `--duration` argument to collect-support-data. 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 TokenValidationResult 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 `--duration` argument to **`collect-support-data`**. 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	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. This authorization mode is enabled on a per-request basis. To enable, a policy that targets the `SCIM2` service and the `search` action must provide an advice with the ID `combine-scim-search-authorizations` but with no payload. The subsequent search response is then authorized by using a single policy request with the 'SCIM2' service and the `search-results` action. If advices are returned in the policy results, they are applied iteratively to each SCIM resource in the result set. For more information, see the *PingDataGovernance Server Administration Guide*.

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 `PERMIT` or `DENY` decisions.
DS-39037	The provided PingDataGovernance policies and deployment packages now apply access token validation policies only to the following requests: - Inbound - SCIM - OpenBanking
DS-39490, DS-39616	The API Endpoint configuration type has been renamed to *Gateway API Endpoint*. Update any existing **`dsconfig`** scripts that reference an API Endpoint. For example, a **`dsconfig`** command of **`create-api-endpoint`** must be changed to **`create-gateway-api-endpoint`**.
DS-39592	HTTP External Servers feature a new attribute, `certificate-alias`, 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 `401 – Unauthorized` response from an external policy decision server, it converts the status to `503 – Service Unavailable` for the upstream client.
DS-40234	The Open Banking account request endpoint no longer requires a value for `x-fapi-financial-id`. Instead, it now includes the configured `fapi-financial-id` value in policy requests through the `Gateway.FapiFinancialId` 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	Added a `cn=Server Status Timeline,cn=monitor` 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 that track health-check state changes for external servers. The new attributes include the following information: ▪ Number of times a health-check transition has occurred ▪ Timestamps of the most recent transitions ▪ Messages associated with the most recent transitions
DS-37504, DS-38765, DS-39011	Fixed an issue in the `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 typically manifested as a SCIM schema error in the debug trace log, such as "Attribute uid in path uid is undefined."
DS-39176, DS-39308	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 `groovy-json` 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 `groovy-xml` or `groovy-sql`, download the appropriate JAR file from *groovy-lang.org* and place it in the server's `lib/extensions` directory.
DS-39564	Fixed an issue in which the gateway responded with a 404 for requests that were handled by a Gateway API Endpoint with an `inbound-base-path` of "/".
DS-39593	Fixed an issue in which policy decision logs contained content that the Policy Administration GUI Log Visualizer considered invalid.

PingDataGovernance Server 7.3.0.0 Release Notes

Upgrade Considerations

Important considerations for upgrading to this version of PingDataGovernance Server:

- WARNING: OAuth scope configurations for resource access control, including fine-grained access control, and JEXL-based policies are no longer supported. Manual steps are necessary to migrate configuration and policies in order to restore the functionality of SCIM APIs. Please contact your account executive to schedule time for migration assistance.

- 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	The following enhancements were made to the topology manager to make it easier to diagnose the connection errors described in PDSTAGING-570: - 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	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.
DS-15734	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.
DS-18060	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. 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 might be useful for orchestration frameworks if you want to destroy and replace any instance that is completely unavailable.
DS-37617	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.
DS-37753	PingDataGovernance now contains Server SDK support for Advices.

Ticket ID	Description
DS-37839	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.
DS-37959	Added support for insignificant configuration archive attributes. 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. 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. By default, the following attributes are now considered insignificant for the purpose of the configuration archive: * 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	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. 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.
DS-38072	Updated the server to enable TLSv1.3 by default on JVMs that support it (Java 11 and higher).
DS-38085	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.
DS-38089,DS-38705	The Open Banking Account Request servlet now supports versions 1.1, 2.0, and 3.0 of the Open Banking Read/Write Data API. Error responses returned by the Account Request servlet are now formatted as described in the Open Banking Read/Write Data API specification, v3.0.
DS-38090,DS-38564,DS-38567	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.
DS-38109	Added the --skipHostnameCheck command-line option to the setup script, which bypasses validation of the provided host name for the server.
DS-38403	Fixed an issue that could prevent certain types of initialization failures from appearing in the server error log by default.
DS-38512	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.
DS-38550	Fixed an issue in which backups of the encryption settings database could be encrypted with a key from the encryption settings database.

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	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, 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. 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. 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 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 may contain clear-text passwords.

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.

Introduction to PingDataGovernance

PingDataGovernance is a solution for centralized, attribute-based entitlement management with a focus on fine-grained access control and data protection.

Organizations worldwide are seeking ways to introduce new use cases and partnerships to accelerate their businesses. At the heart of any new use case or partnership is a question of entitlement. Can a given user perform this action or see some information? Can a given partner access some data or all data?

As use cases become more sophisticated and sensitive data becomes more regulated, the rules that answer these questions of entitlement have become more complex. For example, the user can only perform the action after their account has been open for a month and they've completed on-boarding. Or, the partner can only access user data for those users who have opted-in.

Traditionally, solving complex rules of entitlement requires coding logic into applications, services, and APIs. However, coding entitlement logic creates challenges around visibility, flexibility, time-to-market, duplicated effort, and more.

PingDataGovernance solves the challenge of entitlement for fine-grained access control and data protection.

Key components

PingDataGovernance Policy Administration GUI

Powered by Symphonic, the PingDataGovernance Policy Administration GUI enables nontechnical stakeholders to collaborate with IT to define and test the policies of entitlement. These policies are strictly attribute-based business rules. PingDataGovernance does not store mappings of specific users or groups to actions and resources. Rather, entitlement is determined dynamically at runtime by the PingDataGovernance Server connecting to the attribute sources across the enterprise.

PingDataGovernance Server

The PingDataGovernance Server includes the runtime policy decision service and multiple policy enforcement capabilities. The policy decision service determines whether fine-grained actions can be taken or data can be accessed. Enforcement of these decisions can be handled in several ways:

- Policy Decision Point (PDP) API

  Applications or services call into the policy decision service using the PDP API and enforce the decision in their own application or service code.
- API Security Gateway and Sideband API

  For fine-grained access control and data protection within application, platform, or microservice APIs, customers can integrate the API Security Gateway or Sideband API into their API architecture. In this configuration, the PingDataGovernance Server inspects API requests and responses, and then enforces policy by blocking, filtering, obfuscating, or otherwise modifying request and response data and attributes. This approach requires little or no code changes by the API developer.
- SCIM Service

  For fine-grained access control and data protection to data stored in structured data stores like LDAP and RDBMS, customers can deploy the SCIM Service in front of their data stores. In this configuration, the PingDataGovernance Server provides a SCIM-based microservice API though which clients create, read, update, and delete (CRUD) data. The SCIM Service enforces policy by blocking, filtering, obfuscating, or otherwise modifying data and attributes.

Next steps

To quickly see PingDataGovernance in action, see *Getting started with PingDataGovernance (tutorials)* on page 52.

Getting started with PingDataGovernance (tutorials)

This section provides tutorials for installing and configuring PingDataGovernance Server with different policy options.

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 System for Cross-domain Identity Management (SCIM).

Using the tutorials

Overview

These tutorials provide exercises to familiarize you with the capabilities of PingDataGovernance.

Before you begin

To complete these tutorials, you need:

- To have completed the instructions at *https://devops.pingidentity.com/get-started/getStarted/*
- Git
- To increase your Docker memory limit to at least 4GB. To change this setting, go to **Docker Dashboard# Settings# Resources# Advanced**

The tutorials provide sample requests that use **`curl`**. However, you can use any program that can send HTTP requests, such as **`wget`** or Postman.

Setting up your environment

To help you quickly get started with PingDataGovernance, we provide Docker containers that have everything you need. You deploy these containers using Docker commands and then start using PingDataGovernance.

Clone the GitHub repository that contains the supporting source files.

```
git clone --branch 8.1 https://github.com/pingidentity/pingauthorize-
tutorials && cd pingauthorize-tutorials
```

This command places the files in the `pingauthorize-tutorials` directory and changes to that directory. This directory contains a `docker-compose.yml` file that defines the containers used in the tutorial. You should not need to modify this file or understand its contents to follow the tutorial steps. You might, however, need to change some configuration values that the `docker-compose` environment uses. The `env-template.txt` file contains various configuration values, including the default port definitions used by the `docker-compose` containers. Copy the template to a new file `.env` at the root of the cloned repository and edit its contents using any text editor.

```
cp env-template.txt .env
vi .env
```

You might not need to modify any values if all the default ports are available. However, you must still have a `.env` file in place for the environment to start.

Starting PingDataGovernance

To start the `docker-compose` environment:

1. Go to the `pingauthorize-tutorials` directory you cloned in *Setting up your environment* on page 53.
2. Run the following command.

   ```
   docker-compose up --detach
   ```

Verifying proper startup

To verify that both PingDataGovernance Server and Policy Administration GUI started properly and are running, run the following command.

```
docker container ls --format '{{ .Names }}: {{ .Status }}'
```

The command shows the status of the containers started by the **`docker-compose`** command. Each of the four containers should initially have a status of "`starting`". Eventually, possibly up to 15 minutes, all four containers should reach an equilibrium state of "`healthy`".

If you encounter any issues, check the log files using the `docker-compose logs` command.

Accessing the GUIs

PingDataGovernance has two GUIs.

> ⓘ **Note:** If you have problems connecting because of self-signed certificates, try a different browser.

Administrative Console

Use this console to make configuration changes to PingDataGovernance Server.

URL	https://localhost:5443/console

Details to enter at login	Server: pingdatagovernance Username: administrator Password: 2FederateM0re **Note:** If submitting the form results in a "`Server unavailable`" error, wait longer for the containers to reach an equilibrium "`healthy`" state, as described in *Verifying proper startup* on page 53.

Policy Administration GUI

Use this GUI to make and test policy changes. Also, this GUI calculates decision responses when you configure PingDataGovernance to use the GUI as an external policy decision point.

URL	https://localhost:8443
Details to enter at login	user id: admin password: password123

Stopping PingDataGovernance

If you have completed the tutorials and no longer need the containers, run the following commands to stop and remove the containers.

> **Warning:**
>
> To simplify the prerequisites for using Docker with this tutorial, all of the changes you make are lost when you destroy your Docker Compose environment. For customer installations, persistent volumes are used to maintain data across container deployments.

1. Go to the `pingauthorize-tutorials` directory you cloned in *Setting up your environment* on page 53.
2. Run the following command.

   ```
   docker-compose down
   ```

About the tutorial configuration

The provided Docker containers are preconfigured so that you can begin developing policies immediately.

The following Docker containers are provided through the `docker-compose` environment.

Container	Description
pingdatagovernance	PingDataGovernance Server The server enforces the policies you define.
pingdatagovernancepap	PingDataGovernance Policy Administration GUI Use this GUI to define the policies that determine access control and data protection.

Container	Description
pingdirectory	PingDirectory A directory of user information. **Note:** PingDataGovernance does not require PingDirectory. However, some of the tutorials do use PingDirectory as an attribute provider. You can reference the attributes in your policies.
pingdataconsole	Administrative Console Use this GUI to configure PingDataGovernance.

Tutorial: Importing default policies

This tutorial describes how to use the PingDataGovernance Policy Administration GUI to import default policies for use. Also, it introduces the Trust Framework and describes the default policies.

About this task

Before you can begin writing policies, you must import the default policies from a *snapshot* file. This file contains a minimal set of policies and the default *Trust Framework*. The Trust Framework defines the foundational elements that you use to build policies, such as API services, HTTP methods, and HTTP requests.

The default policies and Trust Framework are stored in a snapshot file named `defaultPolicies.SNAPSHOT`, which is bundled with both PingDataGovernance Server and the Policy Administration GUI. You must base all policies that you create for use with PingDataGovernance on the policies and Trust Framework entities defined in this file.

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

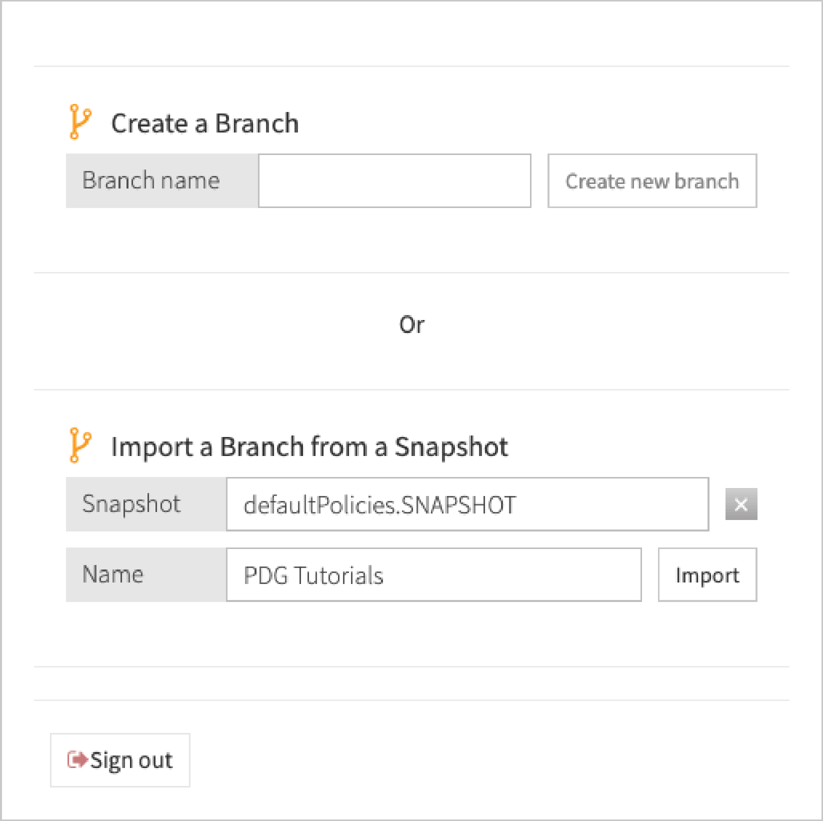
Steps

1. Copy `defaultPolicies.SNAPSHOT` from the PingDataGovernance Policy Administration GUI container to the current directory on your computer. To do this, run the following command. Be sure to include the trailing `.` character.

   ```
   docker cp pingdatagovernancepap:/opt/out/instance/resource/policies/
   defaultPolicies.SNAPSHOT .
   ```

2. Sign on to the Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
3. Under **Import a Branch from a Snapshot**, click where it says **Click here to select a snapshot file**. Choose the file that you just copied to your computer.

4. Name the branch file `PDG Tutorials`.



Create a Branch

Import a Branch from a Snapshot

Snapshot: defaultPolicies.SNAPSHOT

Name: PDG Tutorials

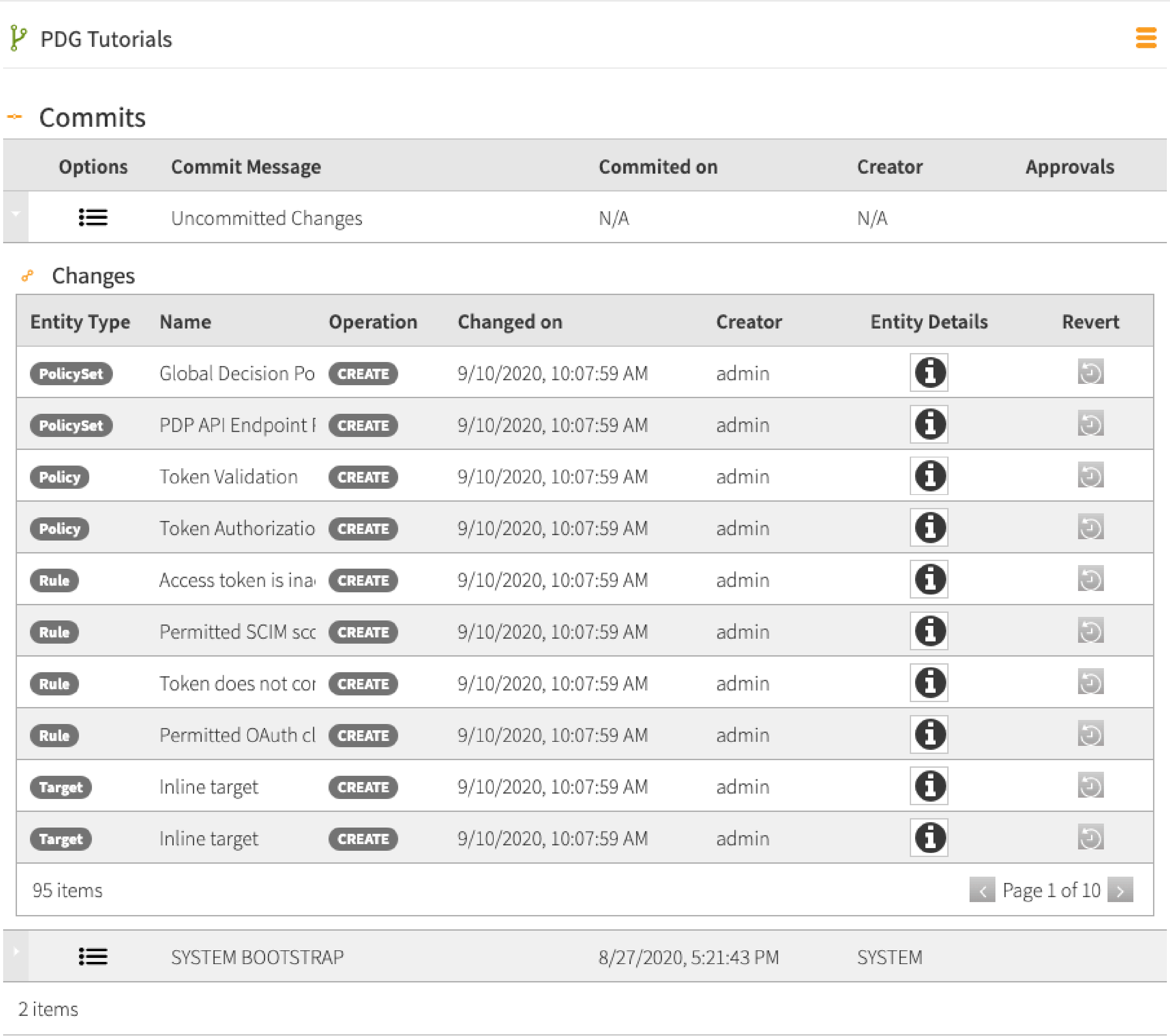
5. Click **Import**.

   The Policy Administration GUI displays the **Version Control** page. From this page, you can manage policy changes similar to how you would in a software source control system.
6. Click **PDG Tutorials** to select the policy branch that you just created.

   A **Commits** table appears. This table provides a log of all changes made to a policy branch.

7. Click the disclosure (triangle) widget at the left of the top line for **Uncommitted Changes**.

   This reveals a list of all changes to the policy branch that are yet to be committed. In this case, the list includes all of the contents of the snapshot that you just imported.



PDG Tutorials

Commits

Options	Commit Message	Commited on	Creator	Approvals
☰	Uncommitted Changes	N/A	N/A	

Changes

Entity Type	Name	Operation	Changed on	Creator	Entity Details	Revert
PolicySet	Global Decision Po	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
PolicySet	PDP API Endpoint I	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Policy	Token Validation	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Policy	Token Authorizatio	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Rule	Access token is ina	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Rule	Permitted SCIM scc	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Rule	Token does not cor	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Rule	Permitted OAuth cl	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Target	Inline target	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲
Target	Inline target	CREATE	9/10/2020, 10:07:59 AM	admin	ℹ	⟲

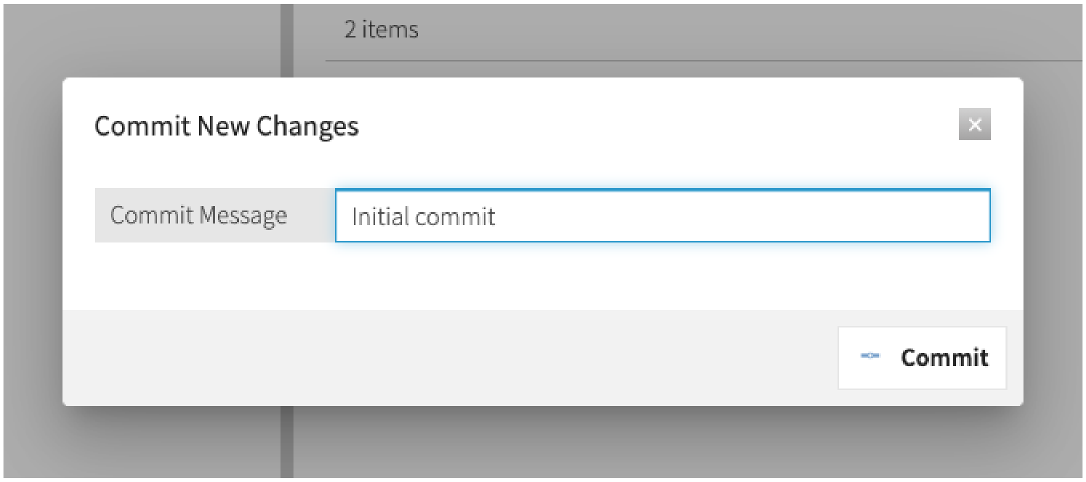
95 items — Page 1 of 10

Options	Commit Message	Commited on	Creator	Approvals
☰	SYSTEM BOOTSTRAP	8/27/2020, 5:21:43 PM	SYSTEM	

2 items

8. Click **Commit New Changes**.

9. Enter the commit message `Initial commit` and click **Commit**.



Commit New Changes

Commit Message: Initial commit

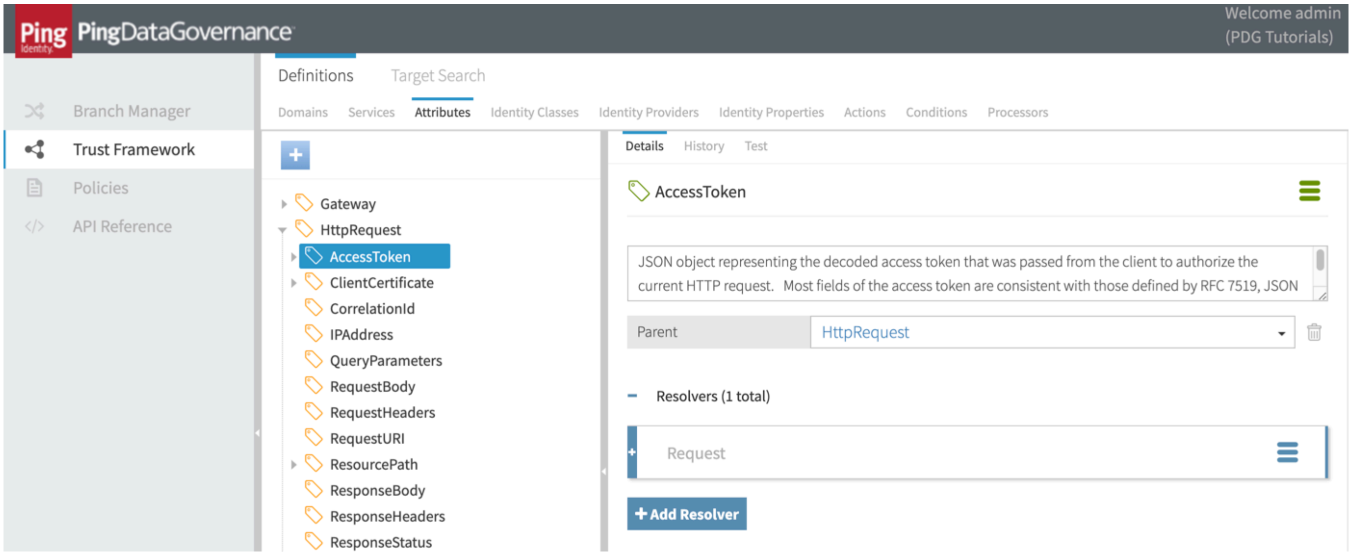
As you work with your own policies, you can use the Policy Administration GUI's version control feature to manage your changes. As you develop policies, a good practice is to set a checkpoint every time you achieve a satisfactory working state by committing your changes.

Introduction to the Trust Framework and default policies

You can now use the Policy Administration GUI with PingDataGovernance Server. First though, explore the interface, paying particular attention to the **Trust Framework** and **Policies** sections in the left pane.

Trust Framework

In the **Trust Framework** section, shown below, you define the foundational elements that you use to build policies and make access control decisions.



AccessToken

JSON object representing the decoded access token that was passed from the client to authorize the current HTTP request. Most fields of the access token are consistent with those defined by RFC 7519, JSON

Parent: HttpRequest

Resolvers (1 total): Request

The Trust Framework provides several types of entities. The following table describes the ones you will use most.

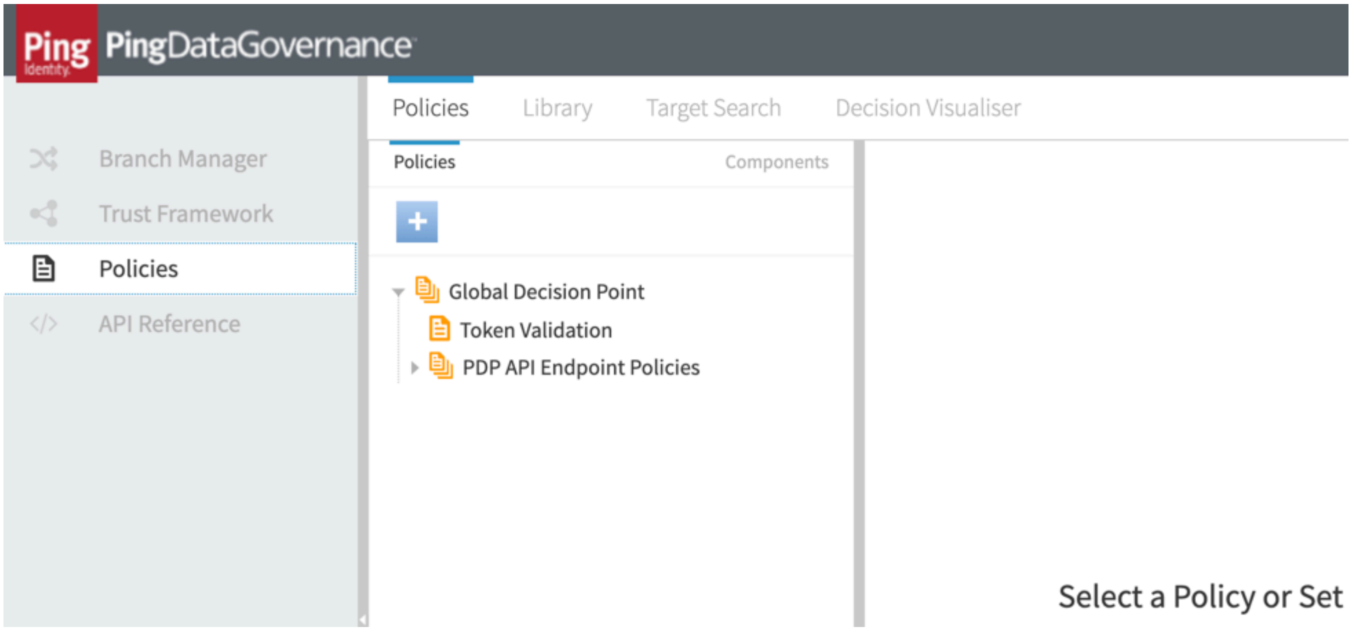
Entity	Description
Services	Services perform two functions. Most often, they represent a specific API service or API resource type to be protected by your policies. They can also define *policy information points*, external data sources (such as APIs or LDAP directory servers) that PingDataGovernance can use to make policy decisions.
Attributes	Attributes provide the context that informs fine-grained policy decisions. Attributes often correspond to elements of an HTTP request, such as an access token subject. However, you can obtain their values from a variety of sources.
Actions	Actions label the type of a request and generally correspond to HTTP methods (GET, POST, and so on) or CRUD actions (create, delete, and so on).

Look at the Trust Framework's default attributes and consider how you could use them in your own policies. Some important Trust Framework attributes include those in the following table.

Attribute	Description
`HttpRequest.AccessToken`	This is the introspected or deserialized access token from the HTTP request.
`HttpRequest.RequestBody`	This is the HTTP request body, typically present for POST, PUT, and PATCH operations.
`HttpRequest.ResponseBody`	This is the upstream API server's HTTP response body.
`SCIM.resource`	For SCIM operations, this is the SCIM resource being retrieved or modified.
`TokenOwner`	For requests authorized using an access token, this is the user who granted the access token.

Policies

In the **Policies** section, shown below, you define your organization's access control policies.



You define your policies as a hierarchical tree of policies. This tree consists of two types of items.

Policy Set

A container for one or more policies.

Policy

A policy, which defines a set of rules that yield a policy decision when evaluated.

When the policy engine receives a policy request from PingDataGovernance Server in response to an API call, it starts at the Global Decision Point and walks down the policy tree, first checking if each policy set or policy is applicable to the current policy request, and then evaluating the rules defined by each policy. Each rule returns a policy decision, typically PERMIT or DENY. Likewise, each policy might return a different policy decision. The policy engine evaluates an overall decision using *combining algorithms*.

The default policy tree contains the following policy sets and policies:

Global Decision Point

This is the root of the policy tree. Place all other policy sets or policies under this point. This node's combining algorithm is set to **A single deny will override any permit**. This algorithm requires no denies and at least one policy to permit the API call.

Token Validation

For most cases, this is the only default policy. It checks for a valid access token. In combination with the Global Decision Point combining algorithm, this is rather permissive. Any API caller can succeed with a valid access token.

PDP API Endpoint Policies

The PingDataGovernance Server PDP API uses these policies. They are not discussed further in this tutorial.

You will use the following items in the UI in a tutorial.

Library

The default policy library contains example advice and rules.

Decision Visualiser

You will use this tool to examine policy decisions in detail.

Tutorial: Configuring fine-grained action access control for an API

This tutorial demonstrates how to use PingDataGovernance to easily configure fine-grained access control for a JSON API.

API access control is often categorized in terms of *granularity*.

Access control granularity type	Description
Coarse-grained	Typically describes scenarios in which users or clients are entitled to all or none of particular applications or APIs.
Medium-grained	Typically applies to URL-based scenarios in which users or clients are entitled to some pages or resources within applications or APIs.
Fine-grained	When applied to the actions a user or client can take on an application page or an API resource, typically implies that *action-specific conditions* dictate whether the user or client is entitled to take the action. For example, a request to transfer bank funds might be denied if the amount exceeds the average of recent transfers by 20% or more.

Scenario

For this tutorial, you are the producer of an online game in which players compete with friends to create the funniest meme. When starting a new game, the first player optionally invites other players by their email addresses. To prevent email spam, you must create a policy that blocks a user from starting a new game with other players if the user's email address comes from a generic mail domain.

Game activities are represented using an example *Meme Game API*.

Tasks

This tutorial teaches you how to configure two fine-grained API access control rules by walking you through the following tasks.

1. Configure a reverse proxy for the Meme Game API.
2. Test the reverse proxy.
3. Add a policy for the Meme Game API's Create Game endpoint.
4. Test the policy from the Policy Administration GUI.
5. Test the reverse proxy by making an HTTP request.
6. Modify the rule for the Meme Game API's Create Game endpoint.

The following sections provide the details for completing these tasks.

Configuring a reverse proxy for the Meme Game API

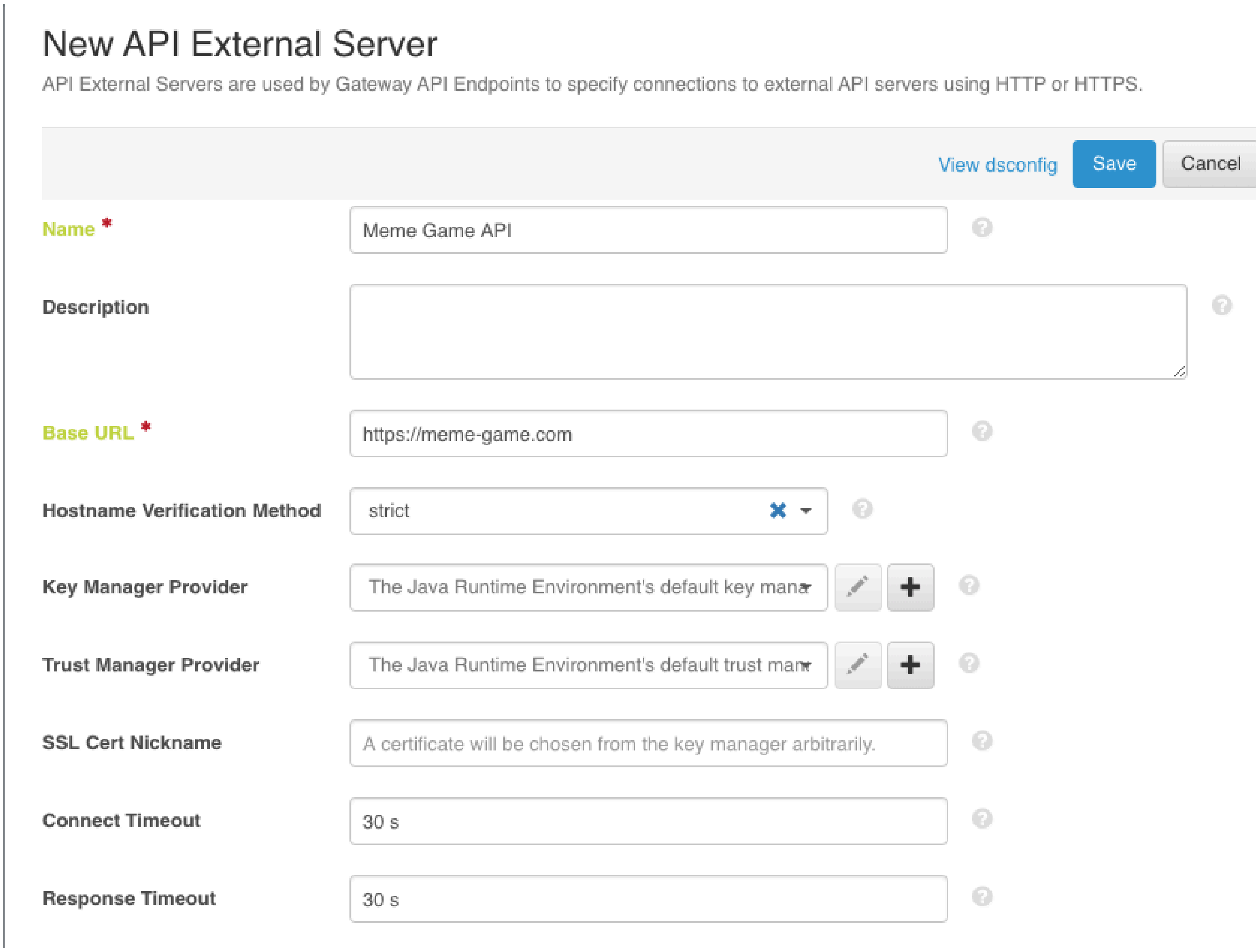
Configure a reverse proxy by configuring an API External Server and a Gateway API Endpoint. The API reverse proxy acts as an intermediary between your HTTP client and the HTTP API, providing fine-grained access control for the API.

About this task

Steps

1. Configure an API External Server for the Meme Game API. An API External Server controls how PingDataGovernance Server handles connections to an HTTPS API server, including configuration related to TLS. In this case, we simply need to provide a base URL.
   a. Sign on to the Administrative Console using the URL and credentials from *Accessing the GUIs* on page 53.
   b. Click **External Servers**.
   c. Click **New External Server** and choose **API External Server**.
   d. For **Name**, specify `Meme Game API`.
   e. For **Base URL**, specify `https://meme-game.com`.

   The following image shows this configuration.



New API External Server

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

Name *: Meme Game API

Description:

Base URL *: https://meme-game.com

Hostname Verification Method: strict

Key Manager Provider: The Java Runtime Environment's default key mana

Trust Manager Provider: The Java Runtime Environment's default trust man

SSL Cert Nickname: A certificate will be chosen from the key manager arbitrarily.

Connect Timeout: 30 s

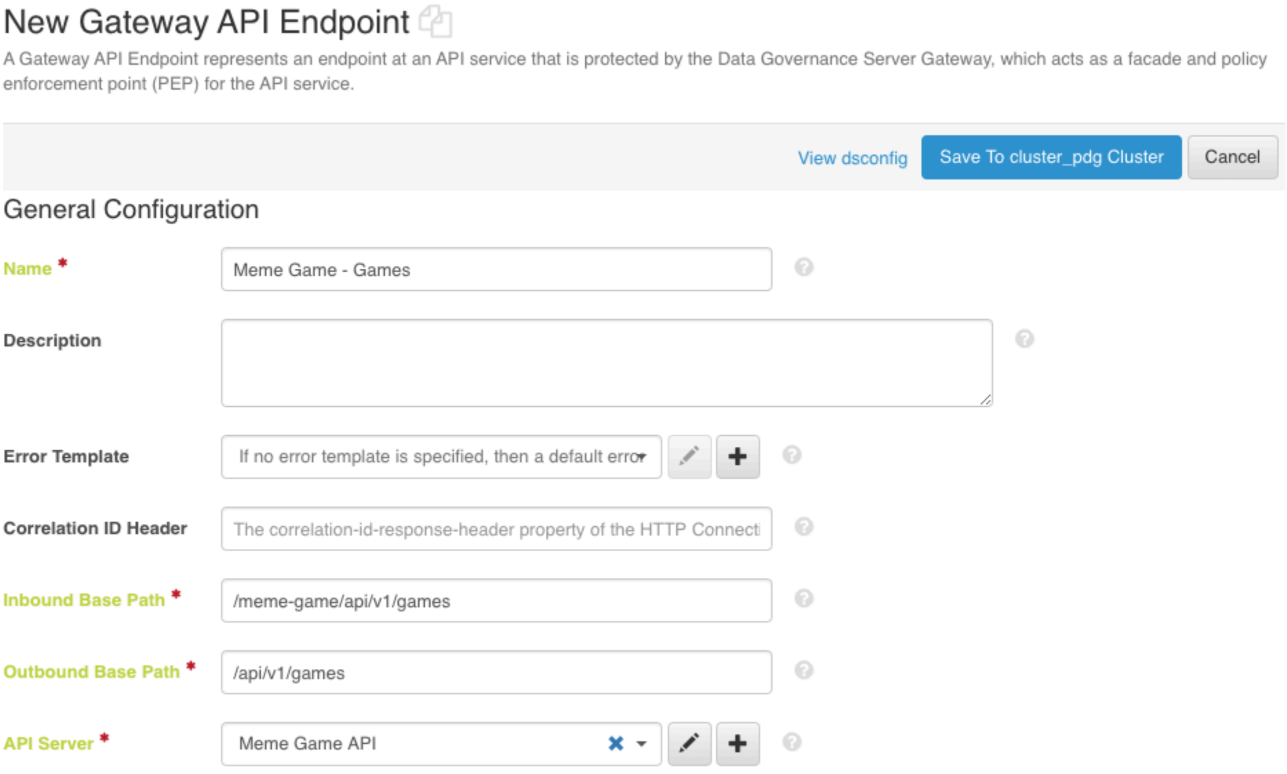
Response Timeout: 30 s

   f. Click **Save**.

2. Configure a Gateway API Endpoint. A Gateway API Endpoint controls how PingDataGovernance Server proxies incoming HTTP client requests to an upstream API server.
   a. In the Administrative Console, click **Configuration** and then **Gateway API Endpoints**.
   b. Click **New Gateway API Endpoint**.
   c. For **Name**, specify `Meme Game - Games`.
   d. For **Inbound Base Path**, specify `/meme-game/api/v1/games`.

      The inbound base path defines the base request path for requests to be received by PingDataGovernance Server.
   e. For **Outbound Base Path**, specify `/api/v1/games`.

      The outbound base path defines the base request path for requests that PingDataGovernance Server forwards to an API server.
   f. For **API Server**, specify `Meme Game API` . This is the API External Server you defined previously.



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.

General Configuration

Name * : Meme Game - Games

Description :

Error Template : If no error template is specified, then a default erro

Correlation ID Header : The correlation-id-response-header property of the HTTP Connecti

Inbound Base Path * : /meme-game/api/v1/games

Outbound Base Path * : /api/v1/games

API Server * : Meme Game API

   g. Save your changes.

Testing the reverse proxy

PingDataGovernance Server is now configured to accept HTTP requests beginning with the path `/meme-games/api/v1/games` and forward them to the Meme Game API. Before proceeding, we will confirm that this configuration is working by making a request to the Meme Game API through the PingDataGovernance Server.

About this task

These tutorials use **`curl`** to make HTTP requests.

The Meme Game API provides an API to create a new game, which looks like this:

```
POST /api/v1/games
{
    "data": {
        "type": "game",
        "attributes": {
            "invitees": ["friend@example.com"]
        }
    }
}
```

We configured a Gateway API Endpoint to forward any requests to `/meme-game/api/v1/games` to the Meme Game API endpoint.

Steps

- Send a request using **`curl`**.

  ```
  curl --insecure --location --request POST 'https://localhost:7443/meme-
  game/api/v1/games' \
  --header 'Authorization: Bearer { "active": true, "sub":
   "user.99@example.com" }' \
  --header 'Content-Type: application/json' \
  --data-raw '{
      "data": {
          "type": "game",
          "attributes": {
              "invitees": [
                  "user.99@example.com"
              ]
          }
      }
  }'
  ```

  This example uses *Bearer token authorization* with a *mock access token*. For an explanation of this authorization, see *For further consideration: The PingDataGovernance API security gateway, part 1* on page 64.

  If the PingDataGovernance Server is configured correctly, then the response status should be `201 Created` with a response body like the following.

  ```
  {
      "data": {
          "id": "130",
          "type": "games"
      },
      "meta": {}
  }
  ```

For further consideration: The PingDataGovernance API security gateway, part 1

Additional concepts to consider include request routing and Bearer token authorization.

Request routing

You configure request routing by defining a Gateway API Endpoint in the PingDataGovernance Server configuration. Each Gateway API Endpoint determines which incoming HTTP requests are proxied to an API server and how PingDataGovernance Server translates the HTTP request into a policy decision request.

Bearer token authorization

The testing in *Testing the reverse proxy* on page 63 uses this authorization. The token itself is a *mock access token*, which is a special kind of Bearer token that a PingDataGovernance Server in test environments can accept. A mock Bearer token is formatted as a single line of JSON, with the same fields used in standard JWT access tokens, plus a boolean `"active"` field, which indicates whether the token should be considered valid. When you use mock access tokens, you do not need to obtain an access token from an actual OAuth 2 auth server, which saves you time during testing.

Adding a policy for the Create Game endpoint

Now that we have confirmed that PingDataGovernance Server is correctly configured to act as a reverse proxy to the Meme Game API, we can define a policy to try out its access control capabilities. This policy will accept or deny a request to create a game based on the identity making the request.

About this task

First, we define a *service* in the Trust Framework. Services have various uses, but at their most basic level, you use them to define a specific API that can be governed by your policies. By defining different services in your Trust Framework, you can target each policy specifically to their applicable APIs.

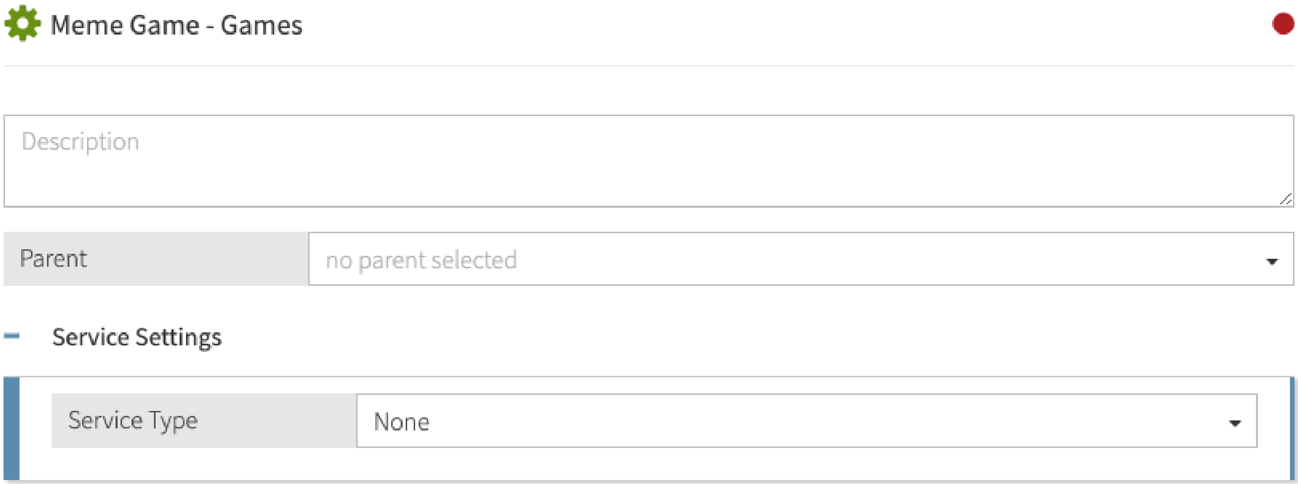
Then, we define a policy. This policy will reject any requests to start a new meme game if the user's identifier ends with `@example.com`. We will identify users using the subject of the request's access token.

Steps

1. Define the service.
   a. Sign on to the Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
   b. Go to **Trust Framework** and click **Services**.
   c. From the **+** menu, select **Add new Service**.
   d. For the name, replace **Untitled** with `Meme Game - Games`.

      The service name must match the endpoint name. To understand why, see *For further consideration: The PingDataGovernance API security gateway, part 2* on page 66.
   e. Verify that in the **Parent** field, no parent is selected.

      To remove a parent, click the trash can icon to the right of **Parent** field.



Meme Game - Games

Description

Parent: no parent selected

Service Settings

Service Type: None

   f. Click **Save changes**.

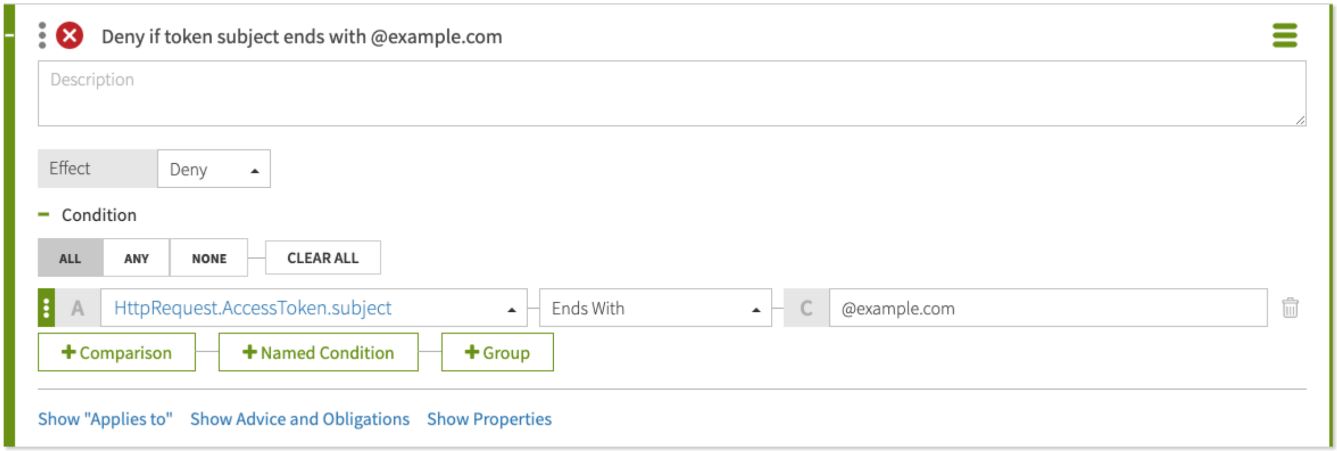
2. Define the policy.
    a. In the Policy Administration GUI, go to **Policies** in the left pane and then click **Policies** along the top.
    b. Select **Global Decision Point**.
    c. From the **+** menu, select **Add Policy**.
    d. For the name, replace **Untitled** with `Users starting a new game`.
    e. Click **+** next to **Applies to**.
    f. In the upper-right corner of the left pane, click **Components**. This reveals a tree of items to target the policy and restrict the types of requests to which the policy applies.
    g. From the **Actions** list, drag **inbound-POST** to the **Drag in a definition or target** box.
    h. From the **Services** list, drag **Meme Games - Games** to the **Drag in a definition or target** box.

        Using these components restricts the policy to incoming POST requests and the Meme Games - Games service.
    i. Set the **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.
    j. Click **+ Add Rule**. This reveals an interface to define a condition. Define the rule as follows.

        1. For the name, replace **Untitled** with `Deny if token subject ends with @example.com`.
        2. For **Effect**, select **Deny**.
        3. Specify the condition.

            a. Click **+ Comparison**.
            b. From the **Select an Attribute** field, select **HttpRequest.AccessToken.subject**.
            c. From the second field, select **Ends With**.
            d. In the third field, type `@example.com`.

        The following screen shows the rule.


Deny if token subject ends with @example.com

Description

Effect: Deny

Condition

ALL | ANY | NONE | CLEAR ALL

A: HttpRequest.AccessToken.subject | Ends With | C: @example.com

    k. Click **Save changes**.

        For more information about API security gateway processing, see *For further consideration: The PingDataGovernance API security gateway, part 2* on page 66.

For further consideration: The PingDataGovernance API security gateway, part 2

Additional concepts to consider include the phases of API security gateway processing and the need for the service name to match the Gateway API Endpoint name.

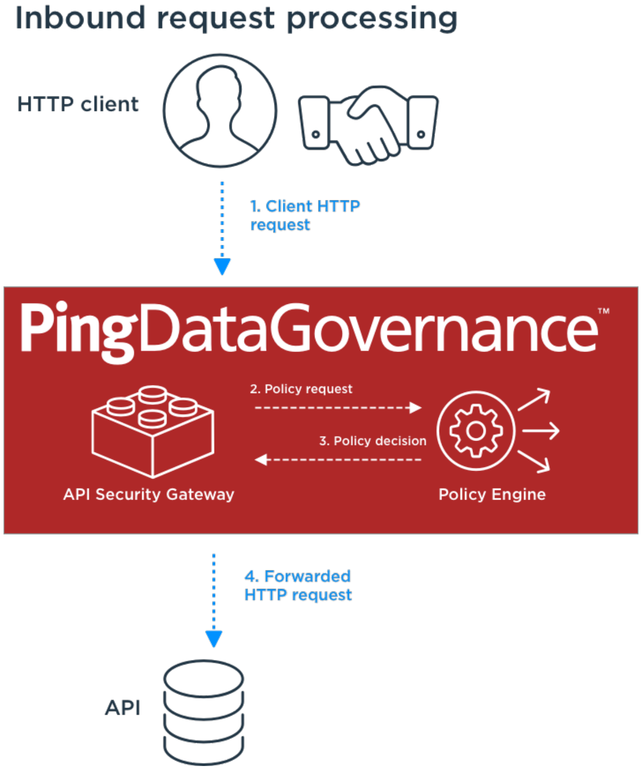
API security gateway processing occurs in two phases

The inbound phase

When the API security gateway receives an HTTP request, it generates a policy request with an action label including the phase and the HTTP method, such as `inbound-POST` or `inbound-GET`.

Based on the result returned by the policy engine, the request might be rejected immediately or it might be forwarded to the API server, potentially with modifications.

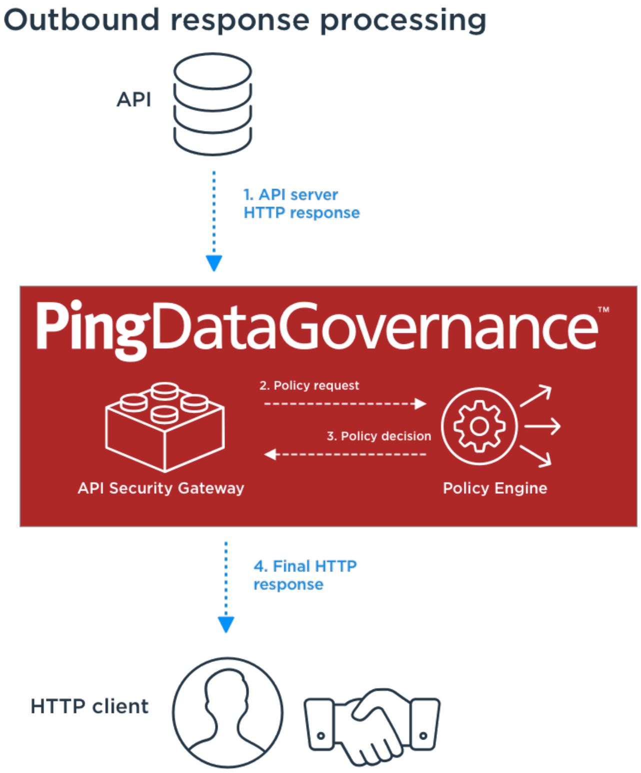
The following diagram illustrates the inbound request processing.



The outbound phase

When the API server returns an HTTP response to the API security gateway, another policy request is generated, again with an action label including the phase and HTTP method, such as `outbound-POST` or `outbound-GET`. Based on the result returned by the policy engine, the response might be modified, and then it is forwarded back to the HTTP client.

The following diagram illustrates the outbound request processing.



Service name must match Gateway API Endpoint name

In *Adding a policy for the Create Game endpoint* on page 65, we named the service to match the name of the Gateway API Endpoint in the PingDataGovernance configuration. This is important. When PingDataGovernance receives an HTTP request, it generates a *policy request* that represents the HTTP request and sends it to its policy engine for processing. The policy request will include a `service` field, and its name will be the name of the Gateway API Endpoint that handled the HTTP request.

Testing the policy from the Policy Administration GUI

We can now test the policy and make sure that it works as we intend. First, we test the policy directly from the Policy Administration GUI's test interface.

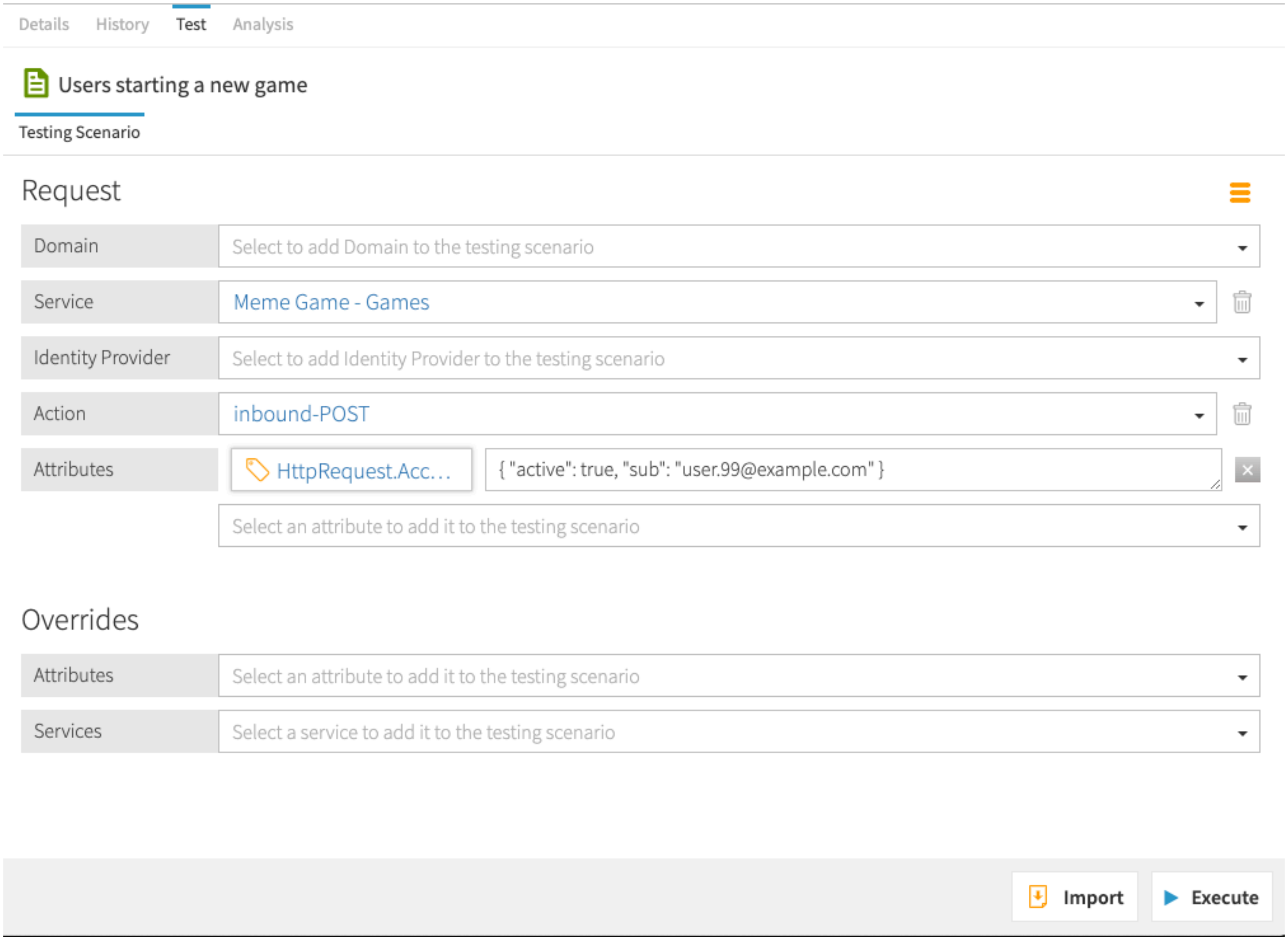
Steps

1. In the Policy Administration GUI, click the **Test** tab at the top of the main pane to display the test interface.
2. Fill out the **Request** section. The test uses this information to simulate the policy request that PingDataGovernance Server makes when it receives an HTTP request.

Service	**Meme Games - Games**
Action	**inbound-POST**

Attributes	**HttpRequest.AccessToken** `{ "active": true, "sub": "user.99@example.com" }`

The following image shows the test.



Users starting a new game

Testing Scenario

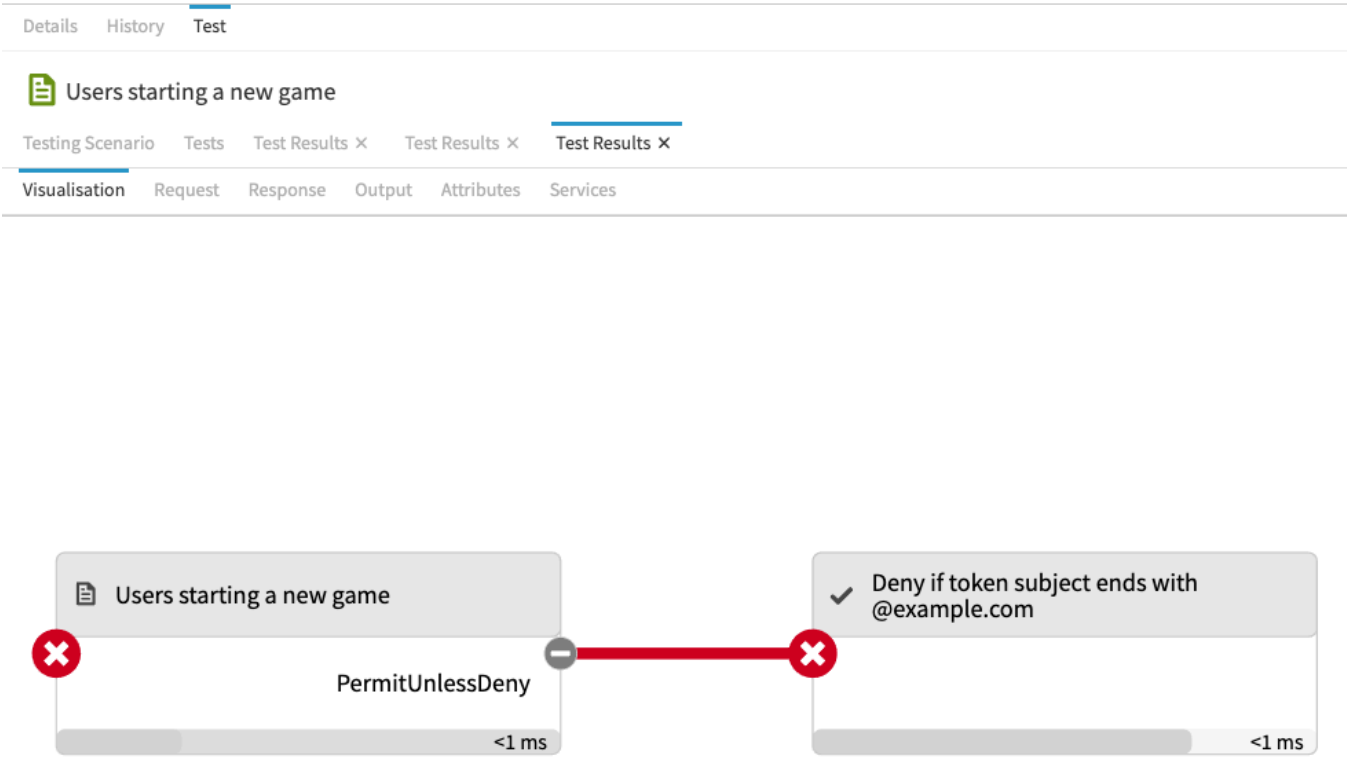
Request

Domain	Select to add Domain to the testing scenario
Service	Meme Game - Games
Identity Provider	Select to add Identity Provider to the testing scenario
Action	inbound-POST
Attributes	HttpRequest.Acc… { "active": true, "sub": "user.99@example.com" }
	Select an attribute to add it to the testing scenario

Overrides

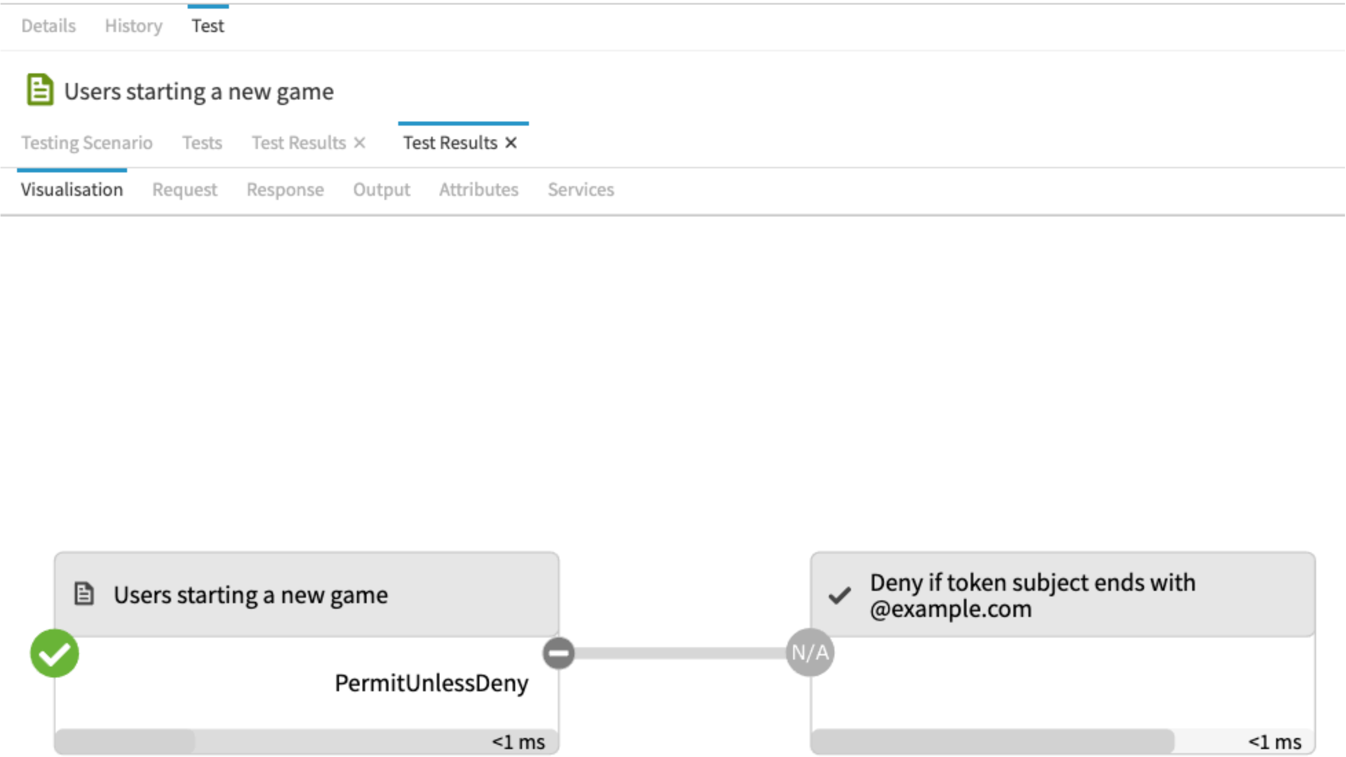
Attributes	Select an attribute to add it to the testing scenario
Services	Select a service to add it to the testing scenario

3. Click **Execute**.
   The policy test result displays. If the policy worked as expected, the leftmost result is red, indicating a `DENY` result.



4. (Optional.) Experiment with testing.

   Click the **Testing Scenario** tab and try different inputs to see how they policy result changes. For example, change the **HttpRequest.AccessToken** attribute value to `{ "active": true, "sub": "user.99@my-company.com" }`. The policy result is now `PERMIT`, as shown in the following image.



Testing the policy by making an HTTP request

Having tested the policy from the Policy Administration GUI to prove the policy works as intended, we can confirm that policy enforcement from end-to-end by sending an HTTP request through the PingDataGovernance Server reverse proxy.

About this task

Steps

1. Send a request using **`curl`**.

```
curl --insecure --location --request POST 'https://localhost:7443/meme-
game/api/v1/games' \
--header 'Authorization: Bearer { "active": true, "sub":
 "user.99@example.com" }' \
--header 'Content-Type: application/json' \
--data-raw '{
    "data": {
        "type": "game",
        "attributes": {
            "invitees": [
                "user.99@example.com"
            ]
        }
    }
}'
```

You should receive an error response with a response status of `403 Forbidden`.

The request has an access token value of `{ "active": true, "sub": "user.99@example.com" }`. The `sub` field of the access token corresponds to the `HttpRequest.AccessToken.subject` Trust Framework attribute that your policy uses to make its decision.

2. As an experiment, edit the access token value in **`curl`** to change the `sub` value to an email address for a different domain. What should happen with this new request?

   Send a request using **`curl`**.

```
curl --insecure --location --request POST 'https://localhost:7443/meme-
game/api/v1/games' \
--header 'Authorization: Bearer { "active": true, "sub": "user.99@my-
company.com" }' \
--header 'Content-Type: application/json' \
--data-raw '{
    "data": {
        "type": "game",
        "attributes": {
            "invitees": [
                "user.99@example.com"
            ]
        }
    }
}'
```

   The HTTP response status should now be `201 Created`.

   To better understand how policy decisions work, see *For further consideration: Decision Visualiser* on page 72.

For further consideration: Decision Visualiser

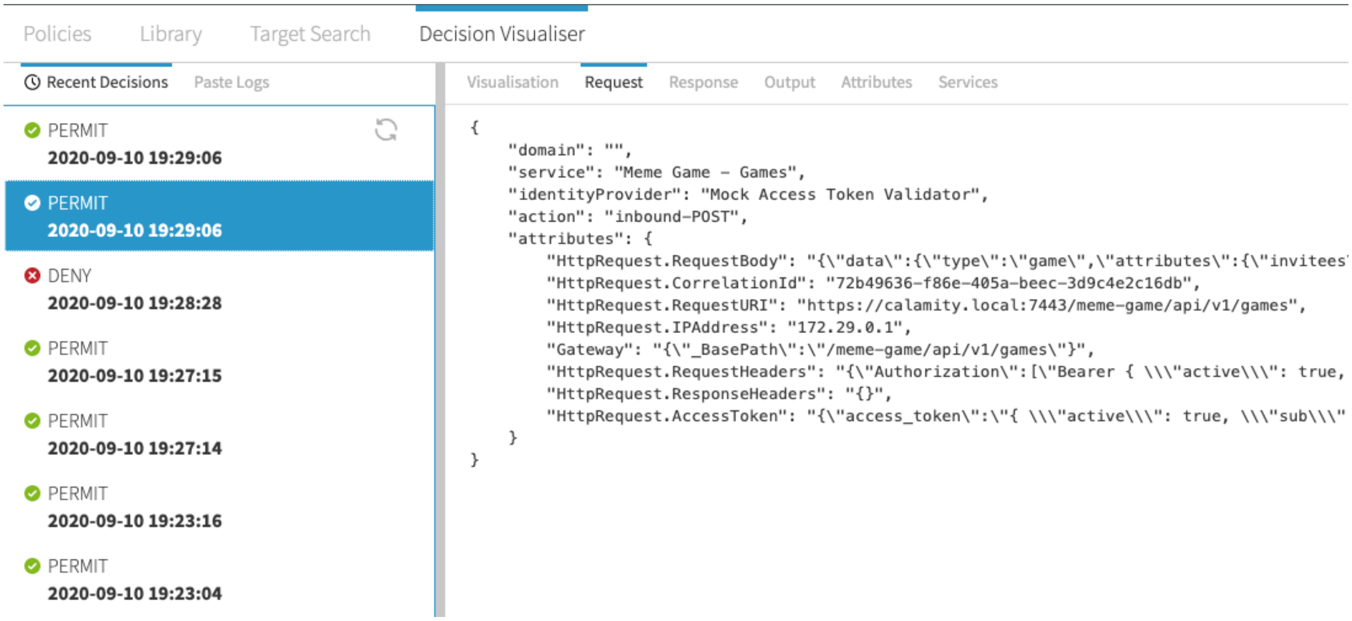
Returning to the Policy Administration GUI, we can view a log of how the policy engine handled the HTTP request.

Steps

1. In the Policy Administration GUI, go to **Policies** and click **Decision Visualiser**.
2. Click the **Recent Decisions** tab. The two most recent items listed correspond to your last HTTP request and response. The first item should correspond to the HTTP response, while the second item should correspond to the HTTP request.
3. Click the second decision. Its visualization appears.

4. Click the **Request** tab. This displays a JSON representation of the policy request that PingDataGovernance generated to represent your HTTP request.

   Here is a request example.



PERMIT
2020-09-10 19:29:06

PERMIT
2020-09-10 19:29:06

DENY
2020-09-10 19:28:28

PERMIT
2020-09-10 19:27:15

PERMIT
2020-09-10 19:27:14

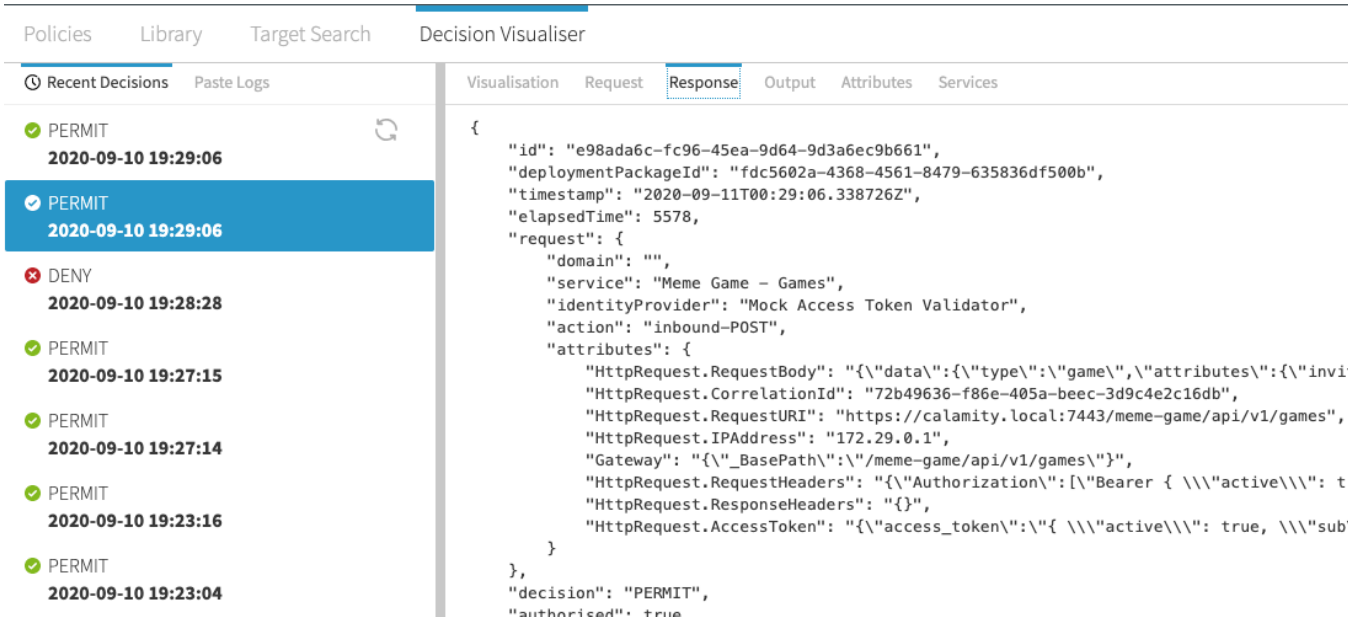
PERMIT
2020-09-10 19:23:16

PERMIT
2020-09-10 19:23:04

```
{
    "domain": "",
    "service": "Meme Game - Games",
    "identityProvider": "Mock Access Token Validator",
    "action": "inbound-POST",
    "attributes": {
        "HttpRequest.RequestBody": "{\"data\":{\"type\":\"game\",\"attributes\":{\"invitees
        "HttpRequest.CorrelationId": "72b49636-f86e-405a-beec-3d9c4e2c16db",
        "HttpRequest.RequestURI": "https://calamity.local:7443/meme-game/api/v1/games",
        "HttpRequest.IPAddress": "172.29.0.1",
        "Gateway": "{\"_BasePath\":\"/meme-game/api/v1/games\"}",
        "HttpRequest.RequestHeaders": "{\"Authorization\":[\"Bearer { \\\"active\\\": true,
        "HttpRequest.ResponseHeaders": "{}",
        "HttpRequest.AccessToken": "{\"access_token\":\"{ \\\"active\\\": true, \\\"sub\\\"
    }
}
```

5. Click the **Response** tab. This displays a JSON representation of the policy response that the policy engine returned after evaluating your policy.

   Here is a response example.



PERMIT
2020-09-10 19:29:06

PERMIT
2020-09-10 19:29:06

DENY
2020-09-10 19:28:28

PERMIT
2020-09-10 19:27:15

PERMIT
2020-09-10 19:27:14

PERMIT
2020-09-10 19:23:16

PERMIT
2020-09-10 19:23:04

```
{
    "id": "e98ada6c-fc96-45ea-9d64-9d3a6ec9b661",
    "deploymentPackageId": "fdc5602a-4368-4561-8479-635836df500b",
    "timestamp": "2020-09-11T00:29:06.338726Z",
    "elapsedTime": 5578,
    "request": {
        "domain": "",
        "service": "Meme Game - Games",
        "identityProvider": "Mock Access Token Validator",
        "action": "inbound-POST",
        "attributes": {
            "HttpRequest.RequestBody": "{\"data\":{\"type\":\"game\",\"attributes\":{\"invi
            "HttpRequest.CorrelationId": "72b49636-f86e-405a-beec-3d9c4e2c16db",
            "HttpRequest.RequestURI": "https://calamity.local:7443/meme-game/api/v1/games",
            "HttpRequest.IPAddress": "172.29.0.1",
            "Gateway": "{\"_BasePath\":\"/meme-game/api/v1/games\"}",
            "HttpRequest.RequestHeaders": "{\"Authorization\":[\"Bearer { \\\"active\\\": t
            "HttpRequest.ResponseHeaders": "{}",
            "HttpRequest.AccessToken": "{\"access_token\":\"{ \\\"active\\\": true, \\\"sub
        }
    },
    "decision": "PERMIT",
    "authorised": true
```

   Both the policy request and the policy response might be hard to understand at the moment, but as you become familiar with PingDataGovernance and its policy engine, you will find that the Decision Visualiser is indispensable for troubleshooting and understanding your policies.

Modifying the rule for the Create Game endpoint

Now that we have defined a policy that permits or denies the ability to create a game based on the email address of the person creating the game, we will modify the rule so that any user can create a game, but only those with real email addresses can create games with invitees. This section demonstrates how a policy can take an action based on data in the request body.

About this task

To review, the Meme Game API offers a game creation endpoint that looks like this:

```
POST /api/v1/games
{
    "data": {
        "type": "game",
        "attributes": {
            "invitees": ["friend@example.com"]
        }
    }
}
```

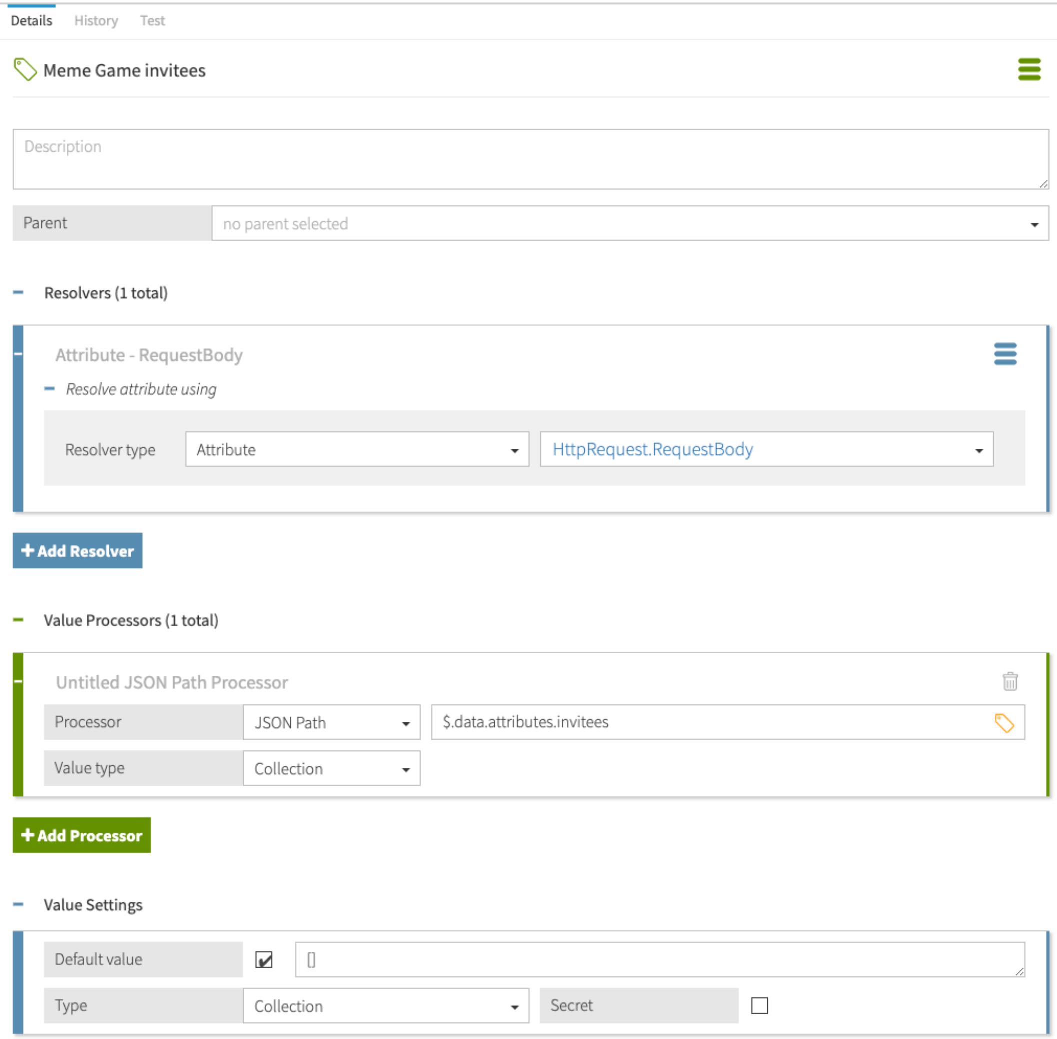
The requester specifies one or more invitees using the `data.attributes.invitees` field. We will update our policy with a second rule that disallows a new game if anybody else is invited to it.

Steps

1. Define a Trust Framework attribute to represent the `data.attributes.invitees` field.
   a. In the Policy Administration GUI, go to **Trust Framework** and click **Attributes**.
   b. From the **+** menu, select **Add new Attribute**.
   c. For the name, replace **Untitled** with `Meme Game invitees`.
   d. Verify that in the **Parent** field, no parent is selected.

      To remove a parent, click the trash can icon to the right of **Parent** field.
   e. Click the **+** next to **Resolvers** and click **+ Add Resolver**.
   f. Set **Resolver type** to **Attribute**.
   g. Select the attribute **HttpRequest.RequestBody**.
   h. Click the **+** next to **Value Processors** and click **+ Add Processor**.
   i. Set **Processor** to **JSON Path**.
   j. Set the value to `$.data.attributes.invitees`.
   k. Set **Value type** to **Collection**.
   l. For **Value Settings**, select **Default value** and specify square brackets (`[]`) to indicate an empty collection.
   m. Set **Type** to **Collection**.
   n. Click **Save changes**.

      The following image shows the new attribute.



Meme Game invitees

Description

Parent: no parent selected

Resolvers (1 total)

Attribute - RequestBody

Resolve attribute using

Resolver type: Attribute | HttpRequest.RequestBody

Value Processors (1 total)

Untitled JSON Path Processor

Processor: JSON Path | $.data.attributes.invitees

Value type: Collection

Value Settings

Default value ☑ []

Type: Collection

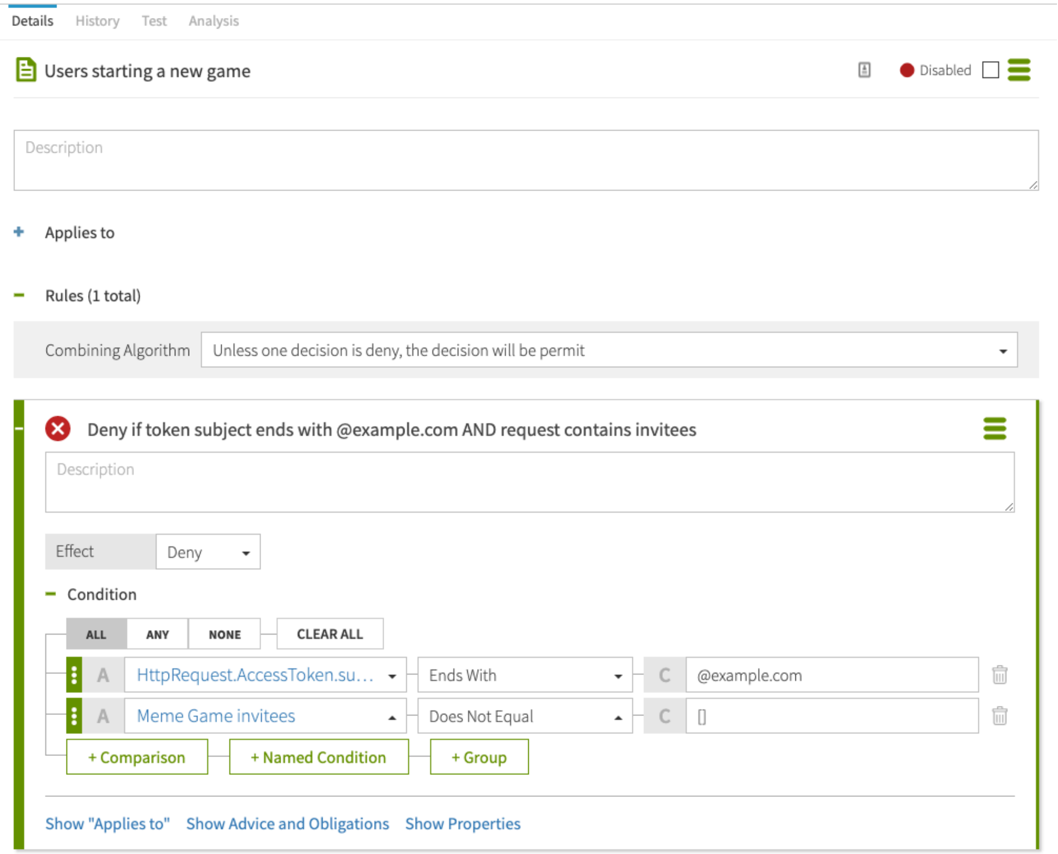
Secret □

This Trust Framework attribute introduces resolvers and value processors, which are two important components. To better understand these components, see *For further consideration: Resolvers and value processors* on page 77.

2. Modify a rule to use the **Meme Game invitees** attribute we just created.
   a. In the Policy Administration GUI, go to **Policies**.
   b. Select the **Users starting a new game** policy.
   c. Rename the **Deny if token subject ends with @example.com** rule to `Deny if token subject ends with @example.com AND request contains invitees`.
   d. Expand the rule by clicking its **+** icon.
   e. For **Effect**, select **Deny**.
   f. Specify a second comparison.

      1. Click **+ Comparison**.
      2. From the **Select an Attribute** field, select **Meme Game invitees**.
      3. From the second field, select **Does Not Equal**.
      4. In the third field, type `[]`.
   g. Click **Save changes**.

      The following image shows the rule.



Users starting a new game

Description

Combining Algorithm: Unless one decision is deny, the decision will be permit

Deny if token subject ends with @example.com AND request contains invitees

Description

Effect: Deny

A HttpRequest.AccessToken.su… | Ends With | C @example.com

A Meme Game invitees | Does Not Equal | C []

3. Test the policy.

   As before, you can test your policy from the Policy Administration GUI using its test interface, and you can test the policy by sending an HTTP request. Try testing using the following combinations of inputs:

   - An access token with the subject `user.0@example.com` and with invitees.

     This should be denied.
   - An access token with the subject `user.0@my-company.com` and with invitees.

     This should be permitted.
   - An access token with the subject `user.0@example.com` and no invitee list.

     This should be permitted.
   - An access token with the subject `user.0@my-company.com` and no invitee list.

     This should be permitted.

For further consideration: Resolvers and value processors

Resolvers and value processors are key components in defining policies.

Modifying the rule for the Create Game endpoint on page 73 introduces their use. Here is more about how you use them in your policies.

- **Resolvers**

  A resolver defines the source of an attribute's value. In this case, the source is the **HttpRequest.RequestBody** policy request attribute, which is set automatically by PingDataGovernance Server. Many other types of sources are available; for example, a resolver might define an attribute value using a constant, or a resolver might call out to an external API to obtain the attribute value.
- **Value Processors**

  Value processors extract and transform values from the source value provided by the resolver. In this case, a value processor uses a JSON Path expression to extract the value of a specific field from the HTTP request body provided by the resolver.

Conclusion

In this tutorial about fine-grained action access control, you added anti-spam protections to the Meme Game API by blocking requests using certain email addresses. In doing so, you learned how to configure PingDataGovernance Server to act as a reverse proxy to a JSON API. You then learned how to use the PingDataGovernance Policy Administration GUI to create a fine-grained access control policy with rules that take effect based on the access token and body of an HTTP request. You also learned how to test policies and inspect policy requests using the Policy Administration GUI.

You also learned:

- Gateway API Endpoint names in the PingDataGovernance Server configuration must match Trust Framework Service names in the Policy Administration GUI.
- Policies can pinpoint different API services and HTTP verbs.
- Policies can PERMIT or DENY transactions based on any combination of attributes.
- Mock access tokens make testing very easy.
- Trust Framework attributes obtain their values using resolvers and transform their values using processors.
- PingDataGovernance Server supplies Attributes for HTTP metadata, request data, and OAuth 2 access token attributes.
- You can test policies directly from the Policy Administration GUI.
- The Policy Administration GUI's Decision Visualiser gives you a detailed view of recent policy decisions.

Tutorial: Configuring attribute-based resource access control for an API

This tutorial describes how to build and test policies that restrict access to a resource based on attributes of both the resource and the caller.

Scenario

In some data use cases, it is necessary to know both the resource being requested and the requesting user. For example, a counselor can only view the records of students in their department. In the scenario of the meme game, users are allowed to invite their friends or family to like or critique their memes. Because some memes are inappropriate for younger audiences, the city of Youngstown, Ohio passes an ordinance that does not allow you to serve its citizens memes rated for age 13 and older. You must create a policy to enforce this by checking the city of the user's profile and the age rating of the shared meme.

> **Note:**
>
> Obviously, not all Youngstown residents are young. In a more realistic scenario, we might compare the age of the requesting user to the age rating of the meme. However, computing the user's age from their date of birth adds unnecessary complexity.

Tasks

This tutorial teaches you how to configure attribute-based API access control rules by walking you through the following tasks.

1. Configure a proxy for the Meme Game API.
2. Create a policy blocking all users from viewing shared memes.
3. Add policy condition logic to allow users not from Youngstown to view shared memes.
4. Add policy condition logic to allow users from Youngstown to view shared memes rated under 13.
5. Add advice to set the API error response when policy blocks access.

The following sections provide the details for completing these tasks.

Configuring the API security gateway

This tutorial describes how to use the API security gateway to allow requests to a parameterized endpoint.

You will configure `https://localhost:7443/meme-game/api/v1/users/{user}/answers` to proxy to `https://meme-game.com/api/v1/users/{user}/answers`, where user can be any username.

Creating the gateway API endpoint

Configure a reverse proxy by configuring an API External Server and a Gateway API Endpoint.

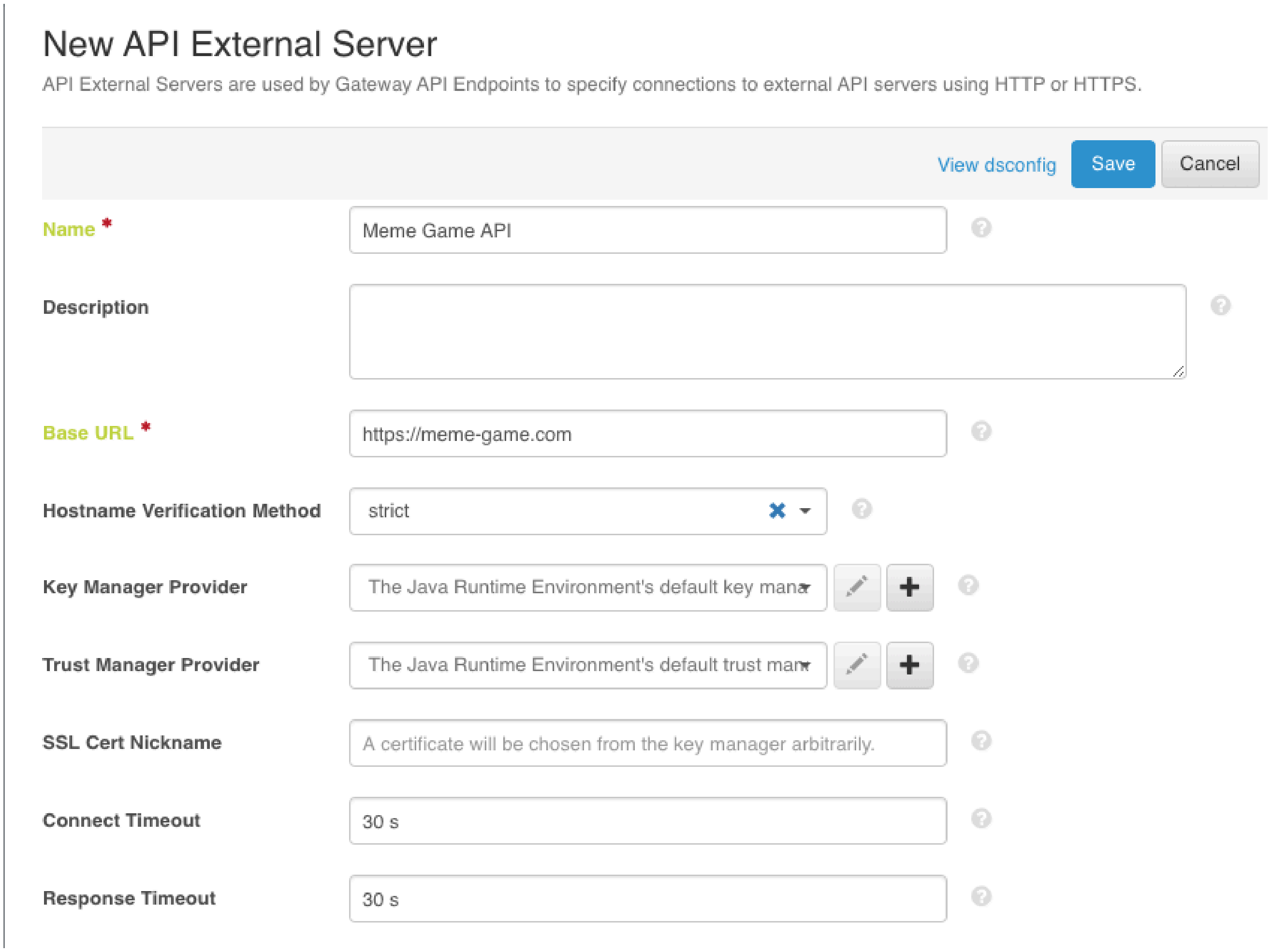
Steps

1. (Optional.) Configure an API External Server for the Meme Game API. An API External Server controls how PingDataGovernance Server handles connections to an HTTPS API server, including configuration related to TLS. In this case, we simply need to provide a base URL.

   > ⓘ **Note:** This step is optional because if you completed *Tutorial: Configuring fine-grained action access control for an API* on page 61, then you already set up this API External Server.

   a. Sign on to the Administrative Console using the URL and credentials from *Accessing the GUIs* on page 53.
   b. Click **External Servers**.
   c. Click **New External Server** and choose **API External Server**.
   d. For **Name**, specify `Meme Game API`.
   e. For **Base URL**, specify `https://meme-game.com`.

      The following image shows this configuration.



New API External Server

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

Name *	Meme Game API
Description	
Base URL *	https://meme-game.com
Hostname Verification Method	strict
Key Manager Provider	The Java Runtime Environment's default key mana
Trust Manager Provider	The Java Runtime Environment's default trust man
SSL Cert Nickname	A certificate will be chosen from the key manager arbitrarily.
Connect Timeout	30 s
Response Timeout	30 s

   f. Click **Save**.

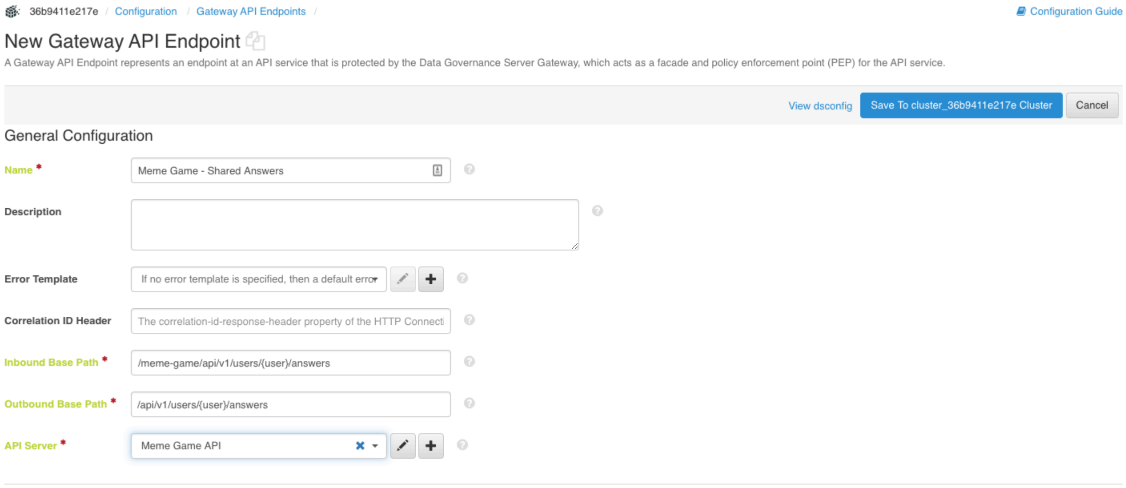
2. Configure a Gateway API Endpoint. A Gateway API Endpoint controls how PingDataGovernance Server proxies incoming HTTP client requests to an upstream API server.
   a. In the Administrative Console, click **Configuration** and then **Gateway API Endpoints**.
   b. Click **New Gateway API Endpoint**.
   c. For **Name**, specify `Meme Game - Shared Answers`.
   d. For **Inbound Base Path**, specify `/meme-game/api/v1/users/{user}/answers`.

      The inbound base path defines the base request path for requests to be received by PingDataGovernance Server.

      By surrounding a value in curly braces, you can add a parameter to a gateway API endpoint's **inbound-base-path**, and use it to fill in a parameter of the same name in the outbound path, as well as to inform other elements of the policy request, such as the service.
   e. For **Outbound Base Path**, specify `/api/v1/users/{user}/answers`.

      The outbound base path defines the base request path for requests that PingDataGovernance Server forwards to an API server.
   f. For **API Server**, specify `Meme Game API`. This is the API External Server you defined in another tutorial, in *Configuring a reverse proxy for the Meme Game API* on page 61.

      Your screen should look like the following one.



36b9411e217e / Configuration / Gateway API Endpoints /
Configuration Guide
View dsconfig
Save To cluster_36b9411e217e Cluster
Cancel

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.

General Configuration

Name *: Meme Game - Shared Answers

Description:

Error Template: If no error template is specified, then a default erro

Correlation ID Header: The correlation-id-response-header property of the HTTP Connecti

Inbound Base Path *: /meme-game/api/v1/users/{user}/answers

Outbound Base Path *: /api/v1/users/{user}/answers

API Server *: Meme Game API

   g. Save your changes.

Testing the gateway

You can test the newly created Gateway API Endpoint with cURL or Postman.

Steps

- Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers`. The following cURL command makes such a request.

```
curl --insecure -X GET \
  https://localhost:7443/meme-game/api/v1/users/user.0/answers \
  -H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
```

  You should get a 200 OK response with a JSON response body that contains a series of answers in an array titled `data`.

Creating a policy based on user credentials

This tutorial describes how to create a policy that acts on information about the user.

Creating a service for the Shared Answers endpoint

Create a service in the Trust Framework to ensure that our policy only affects requests to our new endpoint.

About this task

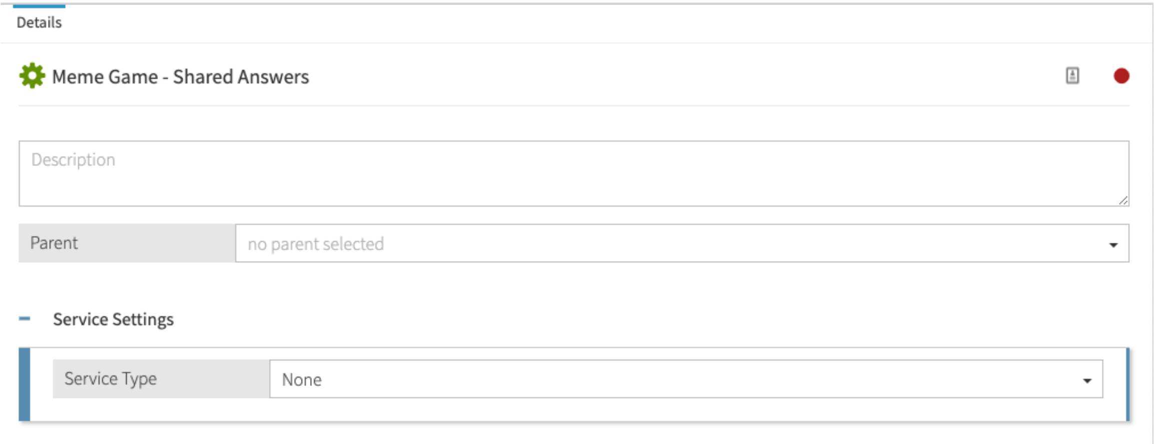
This task passes the name of the Gateway API Endpoint configured in PingDataGovernance Server as the service to the PingDataGovernance policy decision point (PDP).

Steps

1. From the PingDataGovernance Policy Administration GUI, go to **Trust Framework** and click **Services**.
2. From the **+** menu, select **Add new service**.
3. For the name, replace **Untitled** with `Meme Game - Shared Answers`.
4. Verify that in the **Parent** field, no parent is selected.

   To remove a parent, click the trash can icon to the right of the Parent field.
5. Click **Save changes**.

   Your service should look like the one below.



Meme Game - Shared Answers

Description

Parent: no parent selected

Service Settings

Service Type: None

Creating a policy for the Shared Answers endpoint

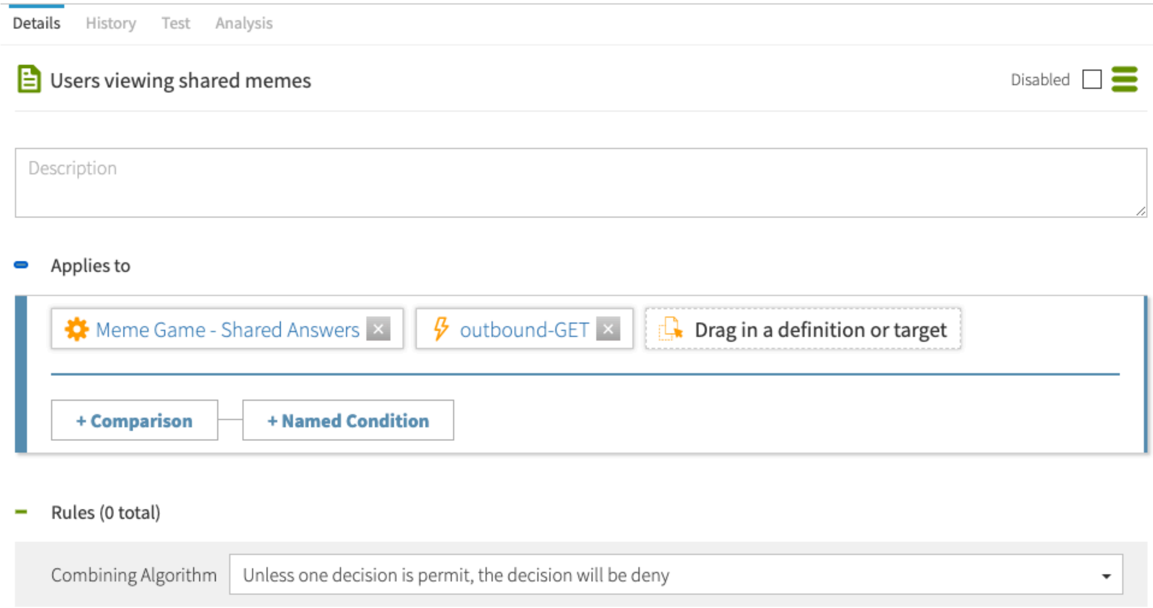
Create a policy to prevent users from accessing the Shared Answers endpoint.

Steps

1. In the PingDataGovernance Policy Administration GUI, go to the **Policies** tab.
2. Select **Global Decision Point**.
3. From the **+** menu, select **Add Policy**.
4. For the name, replace **Untitled** with `Users viewing shared memes`.
5. Click **+** next to **Applies to**.
6. In the upper-right corner of the left pane, click **Components**.
7. From the **Actions** list, drag **outbound-GET** to the **Drag in a definition or target** box.
8. From the **Services** list, drag **Meme Game - Shared Answers** to the **Drag in a definition or target** box.
9. For the combining algorithm, select **Unless one decision is permit, the decision will be deny**.

10. Click **Save changes**.

    Your policy should look like the one shown below.



Details History Test Analysis

Users viewing shared memes

Disabled □

Description

Applies to

Meme Game - Shared Answers ✕ | outbound-GET ✕ | Drag in a definition or target

+ Comparison | + Named Condition

Rules (0 total)

Combining Algorithm: Unless one decision is permit, the decision will be deny

Testing the policy

You can test the new policy with cURL or Postman.

Steps

- Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1`. The following cURL command makes such a request.

```
curl --insecure -X GET \
  https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
   -H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
```

  You should get a 403 Forbidden response with the following body.

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

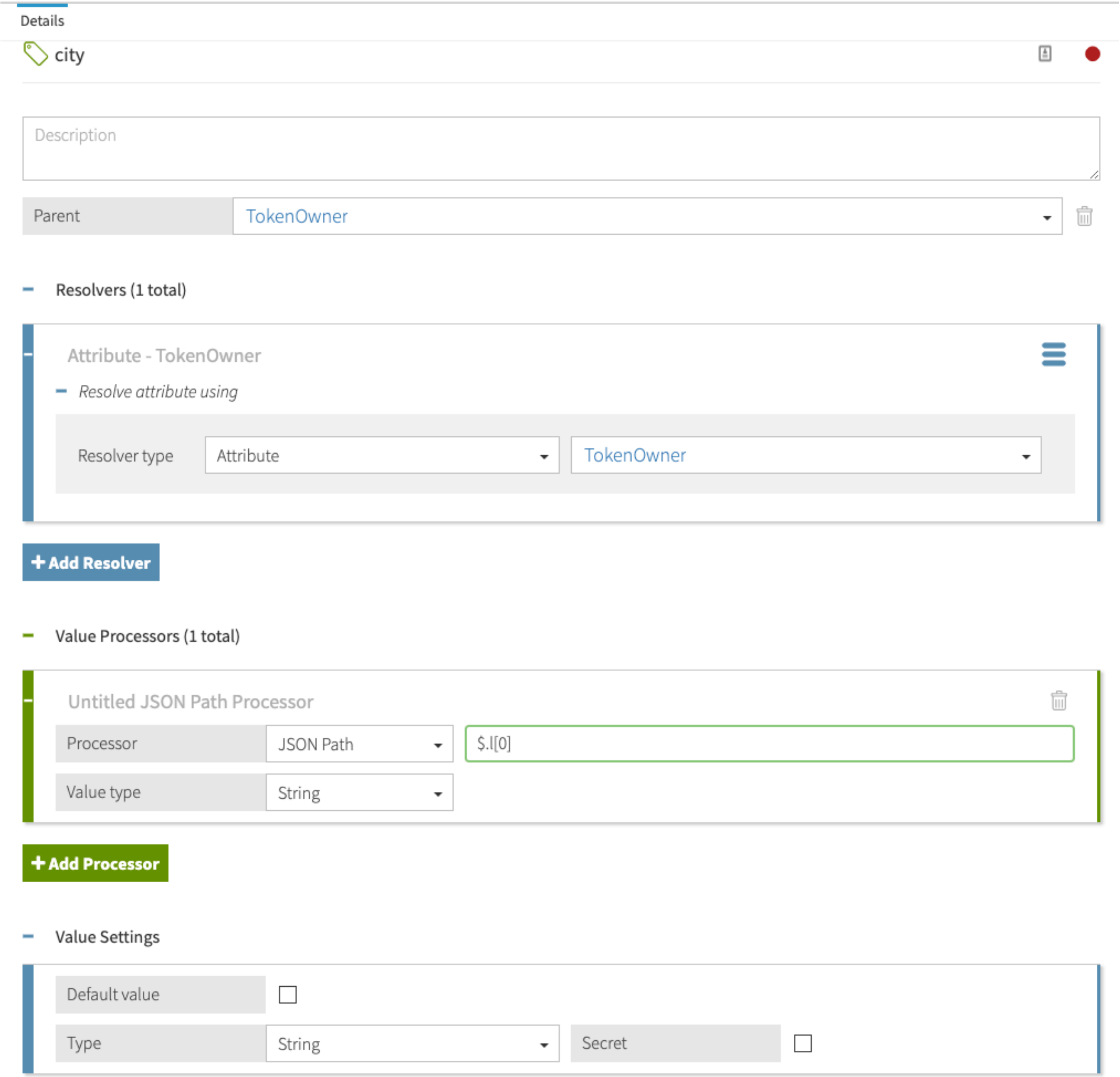
Creating an attribute from user data

Create an attribute to represent the city the user lives in.

Steps

1. In the PingDataGovernance Policy Administration GUI, go to **Trust Framework** and click **Attributes**.
2. From the **+** menu, select **Add new Attribute**.
3. For the name, replace **Untitled** with `city`.
4. For **Parent**, select **TokenOwner**.
5. Click the **+** next to **Resolvers** and click **+ Add Resolver**.
6. For **Resolver type**, select **Attribute** and specify a value of **TokenOwner**.
7. Click the **+** next to **Value Processors** and click **+ Add Processor**.

8. For **Processor**, select **JSON Path** and specify a value of `$.l[0]`. (The LDAP attribute `l` is short for locality.)
9. For the processor's **Value type**, select **String**.
10. For **Value Settings**, set the **Type** to **String**.
11. Click **Save changes**.
    You have an attribute for the user's city, as shown in the following image.



Details

city

Description

Parent: TokenOwner

Resolvers (1 total)

Attribute - TokenOwner

Resolve attribute using

Resolver type: Attribute | TokenOwner

Value Processors (1 total)

Untitled JSON Path Processor

Processor: JSON Path | $.l[0]

Value type: String

Value Settings

Default value □

Type: String

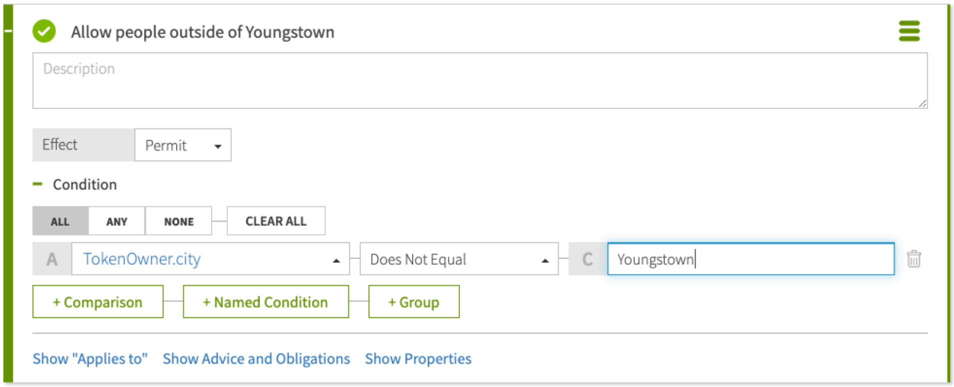
Secret □

Adding logic to allow non-Youngstown users

Add a rule to the **Users viewing shared memes** API policy to allow users who are not from Youngstown to view answers.

Steps

1. From the PingDataGovernance Policy Administration GUI, go to the **Policies** tab.
2. Select **Users viewing shared memes**.
3. Click **+ Add Rule**.
4. For the name, replace **Untitled** with `Allow people outside of Youngstown`.
5. For **Effect**, select **Permit**.
6. To specify a **Condition**, perform the following steps:
   a. Click **+ Comparison**.
   b. From the **Select an Attribute** field, select **TokenOwner.city**.
   c. From the second field, select **Does Not Equal**.
   d. In the third field, type `Youngstown`.
7. Click **Save changes**.
   You have a rule that allows users from outside Youngstown.



Allow people outside of Youngstown

Description

Effect: Permit

Condition

A: TokenOwner.city — Does Not Equal — C: Youngstown

Testing that the policy blocks Youngstown users

You can test the new rule with cURL or Postman.

About this task

Steps

1. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1` as `user.0`. The following cURL command makes such a request.

```
curl --insecure -X GET \
  https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
  -H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
```

A `200 OK` response with the following body.

```
{
 "data": {
     "id": "1",
     "type": "answers",
     "attributes": {
         "url": "https://i.imgflip.com/2fm6x.jpg",
         "captions": [
             "Still waiting for the bus to Jennie’s"
         ],
         "rating": null,
         "created_at": "2020-05-06T22:25:06+00:00"
     }
 },
 "meta": {}
}
```

2. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1` as `user.660`. The following cURL command makes such a request.

   ```
   curl --insecure -X GET \
     https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
     -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
   ```

   The user is from Youngstown, so the result is a `403 Forbidden` response with the following body.

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

Creating a policy based on the API response

This tutorial describes how to create a policy that acts on information about the response received from the API server.

Creating an attribute from response data

Create an attribute to represent the age rating of the meme being requested.

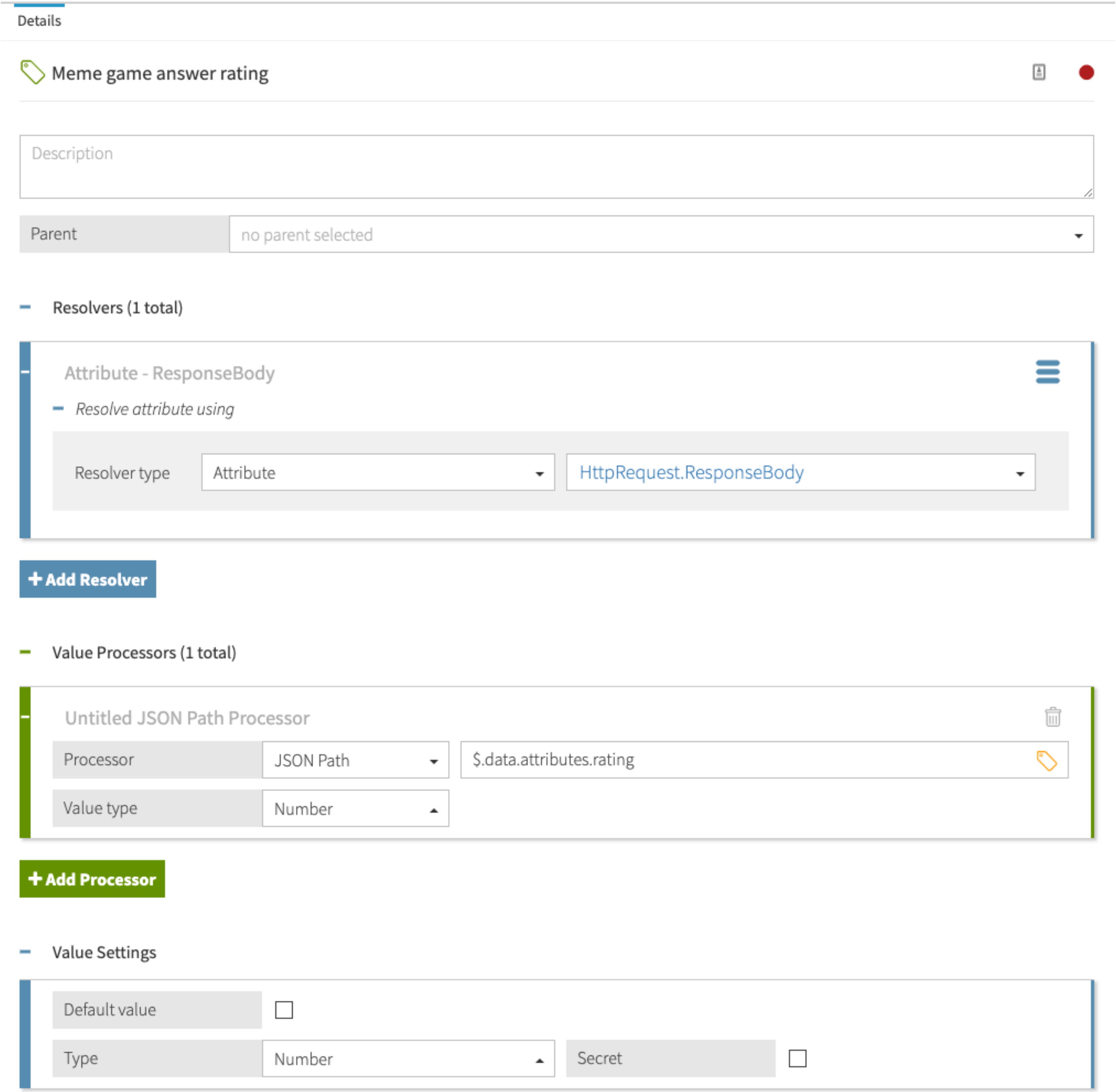
About this task

Steps

1. From the PingDataGovernance Policy Administration GUI, go to **Trust Framework** and click **Attributes**.
2. From the **+** menu, select **Add new Attribute**.
3. For the name, replace **Untitled** with `Meme game answer rating`.
4. Verify that in the **Parent** field, no parent is selected.

   To remove a parent, click the trash can icon to the right of the **Parent** field.
5. Click the **+** next to **Resolvers** and click **+ Add Resolver**.
6. For **Resolver type**, select **Attribute** and specify a value of **HttpRequest.ResponseBody**.
7. Click the **+** next to **Value Processors** and click **+ Add Processor**.
8. For **Processor**, select **JSON Path** and specify a value of `$.data.attributes.rating`.
9. For the processor's **Value type**, select **Number**.
10. For **Value Settings**, set the **Type** to **Number**.

11. Click **Save changes**.
    You have a new attribute for the answer's age rating.



Details

Meme game answer rating

Description

Parent: no parent selected

Resolvers (1 total)

Attribute - ResponseBody

Resolve attribute using

Resolver type: Attribute — HttpRequest.ResponseBody

Value Processors (1 total)

Untitled JSON Path Processor

Processor: JSON Path — $.data.attributes.rating

Value type: Number

Value Settings

Default value □

Type: Number

Secret □

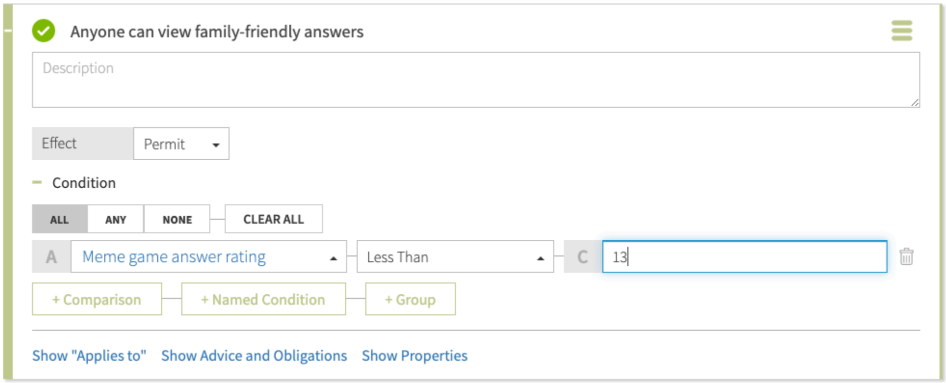
Adding logic to allow family-friendly memes

Add a rule to the **Users viewing shared memes** API policy to allow users to view answers that are rated for ages under 13.

Steps

1. From the PingDataGovernance Policy Administration GUI, go to the **Policies** tab.
2. Select **Users viewing shared memes**.
3. Click **+ Add Rule**.

4. For the name, replace **Untitled** with `Anyone can view family-friendly answers`.
5. For **Effect**, select **Permit**.
6. Specify a **Condition**.
   a. Click **+ Comparison**.
   b. From the **Select an Attribute** field, select **Meme game answer rating**.
   c. From the second field, select **Less Than**.
   d. In the third field, type `13`.
7. 7. Click **Save changes**.
   You have a rule to allow family-friendly memes that looks like the following image.


Anyone can view family-friendly answers

Description

Effect: Permit

Condition

A: Meme game answer rating — Less Than — C: 13

Testing that the policy blocks Youngstown users from viewing age 13+ memes
You can test the newly created rule with cURL or Postman.

About this task

Steps

1. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/2` as `user.0`. The following cURL command makes such a request.

```
curl --insecure -X GET \
  https://localhost:7443/meme-game/api/v1/users/user.0/answers/2 \
  -H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
```

When requesting answer 2 as `user.0`, expect a `200 OK` response with the following body.

```
{
 "data": {
     "id": "2",
     "type": "answers",
     "attributes": {
         "url": "https://i.imgflip.com/23ls.jpg",
         "captions": [
             "There was a spider",
             "it's gone now"
         ],
         "rating": 13,
         "created_at": "2020-05-06T22:25:06+00:00"
     }
 },
 "meta": {}
}
```

2. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/2` as `user.660`. The following cURL command makes such a request.

   ```
   curl --insecure -X GET \
     https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
     -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
   ```

   When requesting answer 2, which is rated age 13, as `user.660`, who is from Youngstown, OH, expect a `403 Forbidden` response with the following body.

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

3. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1` as `user.0`. The following cURL command makes such a request.

   ```
   curl --insecure -X GET \
     https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
     -H 'Authorization: Bearer {"active": true, "sub": "user.0"}'
   ```

   When requesting answer 1 as `user.0`, expect a `200 OK` response with the following body.

   ```
   {
    "data": {
        "id": "1",
        "type": "answers",
        "attributes": {
            "url": "https://i.imgflip.com/2fm6x.jpg",
            "captions": [
                "Still waiting for the bus to Jennie’s"
            ],
            "rating": null,
            "created_at": "2020-05-06T22:25:06+00:00"
        }
    },
    "meta": {}
   }
   ```

4. Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1` as `user.660`. The following cURL command makes such a request.

   ```
   curl --insecure -X GET \
     https://localhost:7443/meme-game/api/v1/users/user.0/answers/2 \
     -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
   ```

   When requesting answer 1, which is unrated, as `user.660`, who is from Youngstown, OH, expect a `403 Forbidden` response with the following body. Be aware that this is not the correct behavior; however, to resolve it, we would need to change our attribute definitions.

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

Allowing unrated memes

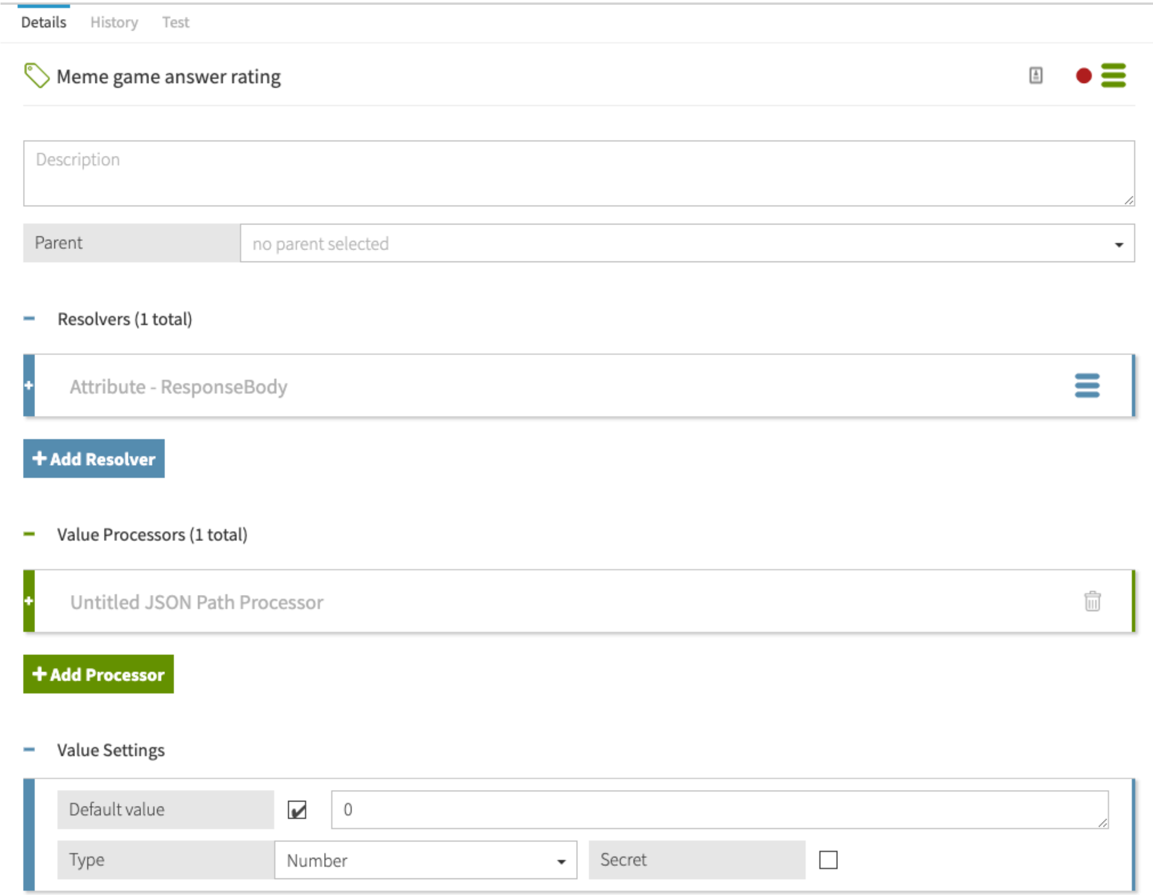
Answer 1 is not being served to `user.660`, even though it has not been rated as 13+. In this scenario, an unrated answer should be considered friendly to all users. Consider why an unrated meme is being blocked for this user. To resolve this, you can add a default value to the age rating.

About this task

Steps

1. In the PingDataGovernance Policy Administration GUI, go to **Trust Framework** and click **Attributes**.
2. Select **Meme game answer rating**.
3. For **Value Settings**, check the `Default Value` box, and specify a value of `0`.
4. Click **Save changes**.

   Your attribute for answer age ratings has a default value of 0, as shown below.



Meme game answer rating

Description

Parent: no parent selected

Resolvers (1 total)

Attribute - ResponseBody

Value Processors (1 total)

Untitled JSON Path Processor

Value Settings

Default value ☑ 0

Type: Number

Secret □

Testing the default value

You can test that the policy now works correctly with cURL or Postman.

Steps

- Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/1` as `user.660`. The following cURL command makes such a request.

```
curl --insecure -X GET \
  https://localhost:7443/meme-game/api/v1/users/user.0/answers/1 \
   -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
```

You should get a `200 OK` response with the following body.

```
{
 "data": {
     "id": "1",
     "type": "answers",
     "attributes": {
         "url": "https://i.imgflip.com/2fm6x.jpg",
         "captions": [
             "Still waiting for the bus to Jennie's"
         ],
         "rating": null,
         "created_at": "2020-05-06T22:25:06+00:00"
     }
 },
 "meta": {}
}
```

Creating an advice to provide a more useful error message

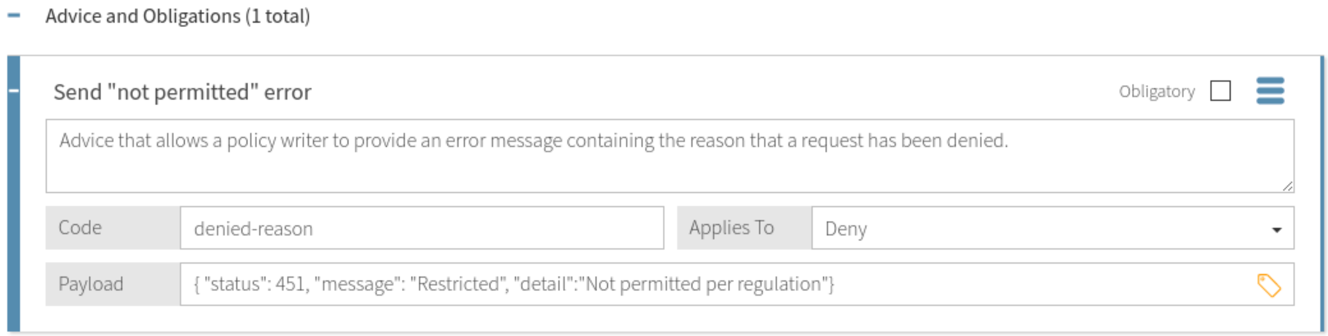
Add a command, known as an advice, that instructs PingDataGovernance to set the HTTP response code and provide a more useful error message when rejecting the outbound response.

About this task

Because this problem is due to an attribute of a user (namely their location), use a 4xx response code to indicate a user issue. The 451 response code has been suggested for use in cases where content cannot be displayed for legal reasons.

Steps

1. From the PingDataGovernance Policy Administration GUI, go to the **Policies** tab.
2. Select **Users viewing shared memes**.
3. Click **+ Advice and Obligations**.
4. Click **+ Add Advice** and select **Denied Reason**.
5. For the name, replace **Untitled** with `Send "not permitted" error`.
6. From the **Applies to** drop-down list, select **Deny**.
7. For a **Payload** value, enter `{"status": 451, "message": "Restricted", "detail": "Not permitted per regulation"}`.
8. Click **Save changes**.
   You have a new advice, which looks something like the following image.



Advice and Obligations (1 total)

Send "not permitted" error

Advice that allows a policy writer to provide an error message containing the reason that a request has been denied.

Code: denied-reason

Applies To: Deny

Payload: { "status": 451, "message": "Restricted", "detail":"Not permitted per regulation"}

Testing the advice

You can test that the advice works correctly with cURL or Postman.

Steps

- Issue a GET request to `https://localhost:7443/meme-game/api/v1/users/user.0/answers/2` as `user.660`. The following cURL command makes such a request.

  ```
  curl --insecure -X GET \
    https://localhost:7443/meme-game/api/v1/users/user.0/answers/2 \
    -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
  ```

  Expect a `451 Unavailable For Legal Reasons` response with the following body.

  ```
  {
   "errorMessage": "Restricted: Not permitted per regulation",
   "status": 451
  }
  ```

Conclusion

In this tutorial, you allowed users to access the meme game's shared answers functionality through PingDataGovernance. Following a request from government authorities, you blocked users from the town of Youngstown, Ohio from viewing memes intended for audiences aged 13 or older. In doing so, you learned about the PingDataGovernance ability to control access to resources based on attributes of both the requesting user and the resource being requested. You also learned how to use advice to modify response bodies.

You also learned:

- Policies can apply "outbound"--upstream server API responses before they are sent to the API client.
- HttpRequest.ResponseBody is the upstream server API response body before it is sent to the client.
- Attributes that cannot be resolved because of any reason including processing errors might impact policy outcomes.
- PingDataGovernance supplies the user profile of access token subject as the Trust Framework attribute `TokenOwner`.
- You must populate the child attributes of the `TokenOwner` that you want to use in policy.
- Many attributes in LDAP are multivalued.
- Advice are the mechanism to modify the API response in some way.
- In this case, denied-reason was used to set the HTTP status code and message body.

Tutorial: Creating SCIM policies

This tutorial describes how to develop a set of access-control policies for the PingDataGovernance Server's built-in System for Cross-domain Identity Management (SCIM) REST API.

In the previous section, you used PingDataGovernance Server to filter data that an external REST API returned.

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.

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 previous section, a mock access token is used.

```
curl --insecure -X GET https://localhost:7443/scim/v2/Me -H 'Authorization:
 Bearer {"active": true, "sub": "user.1", "scope": "nonexistent.scope",
 "client_id": "nonexistent.client"}'
```

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

```
{"mail":["user.1@example.com"],"initials":["RJV"],"homePhone":["+1 091 438
 1890"],
"pager":["+1 472 824 8704"],"givenName":
["Romina"],"employeeNumber":"1","telephoneNumber":["+1 319 624 9982"],
"mobile":["+1 650 622 7719"],"sn":["Valerio"],"cn":["Romina Valerio"],
"description":["This is the description for Romina Valerio."],"street":
["84095 Maple Street"],
"st":["NE"],"postalAddress":["Romina Valerio$84095 Maple Street$Alexandria,
 NE  39160"],
"uid":["user.1"],"l":["Alexandria"],"postalCode":
["39160"],"entryUUID":"355a133d-58ea-3827-8e8d-b39cf74ddb3e",
"objectClass":["top","person","organizationalPerson","inetOrgPerson"],
"entryDN":"uid=user.1,ou=people,o=yeah",
"meta":{"resourceType":"Users",
"location":"https://localhost:7443/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 it is preferred that it not be one, because it shows that any active access token referencing a valid user can be used to access any data.

Scenario

In this tutorial, you use OIDC-like scopes `email` and `profile` to limit data access of the requestor to specific attributes of the profile that granted the access token.

Also, you create a scope `scimAdmin` that has full access to SCIM-based `User` resources.

Tasks

This tutorial walks you through these tasks.

1. Create a basic policy structure for scope-based access to SCIM resources.
2. Create a policy for the `email` scope that only allows access to the subject's `mail` attributes.
3. Create a policy for the `profile` scope that only allows access to a few other profile attributes.
4. Create a policy for the `scimAdmin` scope that allows access to all attributes.

The following sections provide the details for completing these tasks.

Tutorial: Creating the policy tree

This tutorial describes how to create a tree structure and ensure that your policies apply only to System for Cross-domain Identity Management (SCIM) requests.

About this task

The default policies include the policy named `Token Validation`. In the PingDataGovernance Policy Administration GUI, you can find this policy 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.

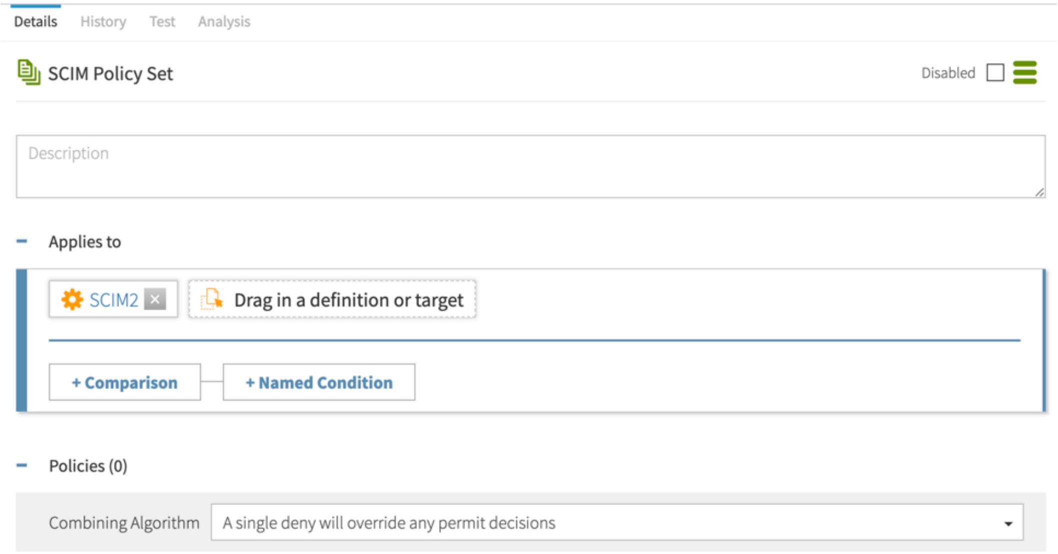
Steps

1. To create the tree structure, perform the following steps:
   a. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
   b. Click **Policies**.
   c. Highlight **Global Decision Point**.
   d. From the **+** menu, select **Add Policy Set**.
   e. For the name, replace **Untitled** with `SCIM Policy Set`.
   f. In the **Policies** section, set the **Combining algorithm** to **A single deny will override any permit decisions**.

      A combining algorithm determines the manner in which the policy set resolves potentially contending decisions from child policies.
   g. Click **+ Applies to**.
   h. Click **Components**.
   i. From the **Services** list, drag **SCIM2** to the **Drag in a definition or target** box.

      This step ensures that policies in the SCIM policy set apply only to SCIM requests.
   j. Click **Save changes**.

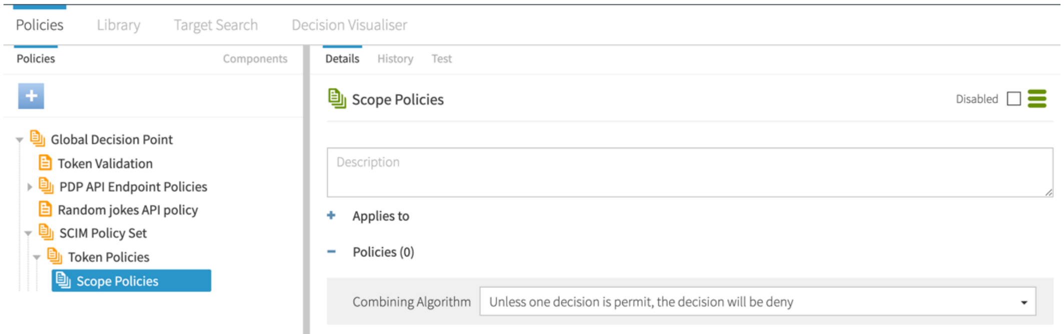
   You should have a screen like the following.



Combining Algorithm: A single deny will override any permit decisions

2. To add a branch under the SCIM policy set to hold SCIM-specific access token policies, go from **Components** to **Policies** and perform the following steps:
   a. Highlight **SCIM Policy Set**.
   b. From the **+** menu, select **Add Policy Set**.
   c. For the name, replace **Untitled** with `Token Policies`.
   d. In the **Policies** section, set the **Combining algorithm** to **A single deny will override any permit decisions**.
   e. 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. From the **+** menu, select **Add Policy Set**.
   c. For the name, replace **Untitled** with `Scope Policies`.
   d. In the **Policies** section, set the **Combining algorithm** to **Unless one decision is permit, the decision will be deny**.
   e. Click **Save changes**.

   After creating the new branches, they should look like the following.



Combining Algorithm: Unless one decision is permit, the decision will be deny

Tutorial: Creating SCIM access token policies

This tutorial describes how to define access token policies after you define a structure.

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

Creating a policy for permitted access token scopes

The first policy defines the access token scopes that PingDataGovernance Server accepts for System for Cross-domain Identity Management (SCIM) requests.

About this task

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. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.

2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, and **Token Policies**.
4. Highlight **Scope Policies**.
5. Next to **Advice and Obligations**, click **+**.
6. Click **Components**.
7. From the **Advice** list, drag **Insufficient Scope** to the area immediately following **Advice and Obligations**. A box appears for you to drop the item into.
8. Click **Save Changes**.
9. Click **Policies** to the left of **Components**.
10. Highlight **Scope Policies**.
11. From the **+** menu, select **Add Policy**.
12. For the name, replace **Untitled** with `Permitted Scopes`.
13. Change the combining algorithm to **A single deny will override any permit decisions**.
14. Click **Save Changes**.

Testing the policy with cURL

Test the newly created policy with cURL.

About this task

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

Steps

- Run the HTTP request to perform the test.

```
curl --insecure -X GET https://localhost:7443/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."}
```

Defining the email scope

Define a permitted access token scope to retrieve email attributes.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, **Token Policies**, and **Scope Policies**.
4. Highlight **Permitted Scopes**.
   a. Click **Components**.
5. From the **Rules** list, drag **Permitted SCIM scope for user** to the **Rules** section.
6. To the right of the copied rule, click the three-line menu.
7. Click **Replace with clone**.
8. Change the name to `Scope: email`.
9. To expand the rule, click **+**.
10. Change the description to `Rule that permits a SCIM user to access its own mail attribute if the access token contains the email scope`.

11. In the **HttpRequest.AccessToken.scope** row of the **Condition** section, type `email` in the **CHANGEME** field.
12. Within the rule, click **Show "Applies to"**.
13. From the **Actions** section, drag **retrieve** to the **Drag in a definition or target** box.

> **Note:**
>
> This task uses different actions from the previous gateway example.

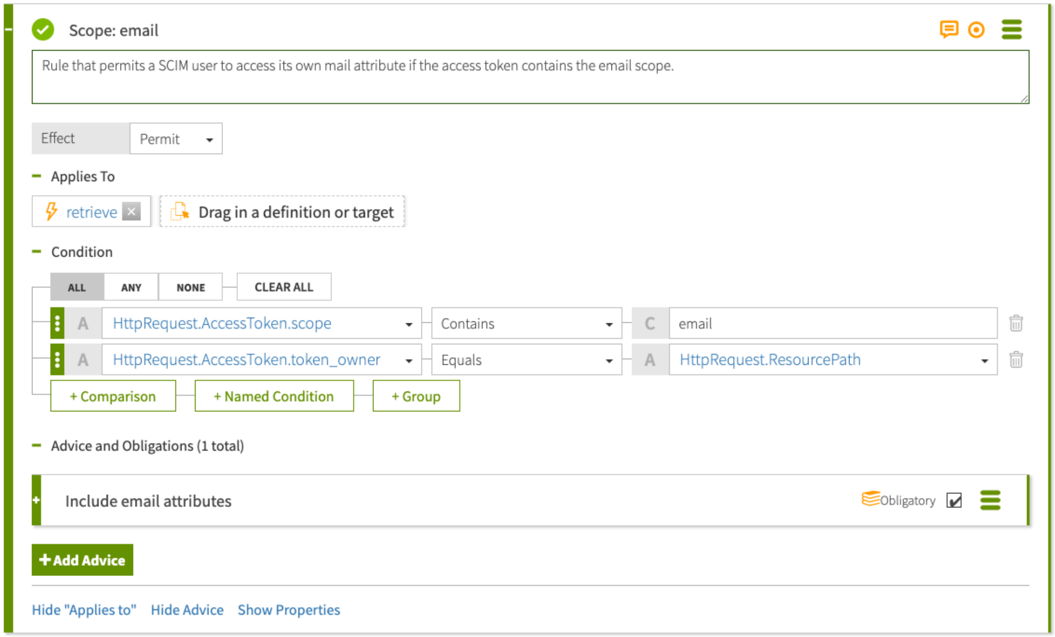
14. Within the rule, click **Show Advice and Obligations**.
15. Click **+** next to **Advice and Obligations**.
16. From the **Advice** section, drag **Include email attributes** to the **Advice and Obligations** section.

> **Note:**
>
> This predefined advice includes a payload. If the condition for this rule is satisfied, the response includes the `mail` attribute.

17. Click **Save changes**.

Results

After completing the configuration, you will have a new email scope, which should look like the following.



Scope: email

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

Effect: Permit

Applies To: retrieve

Condition: ALL

A	HttpRequest.AccessToken.scope	Contains	C email
A	HttpRequest.AccessToken.token_owner	Equals	A HttpRequest.ResourcePath

Advice and Obligations (1 total)

Include email attributes — Obligatory ☑

Testing the email scope with cURL

You can test a newly created 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://localhost:7443/scim/v2/Me -H 'Authorization:
 Bearer {"active": true, "sub": "user.1", "scope": "nonexistent.scope",
 "client_id": "nonexistent.client"}'
```

Steps

- Adjust the request to use the email scope.

```
curl --insecure -X GET https://localhost:7443/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://localhost:7443/
scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"},"schemas":
["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"],"mail":
["user.1@example.com"]}
```

The request succeeds, and only the `mail` attribute is returned.

Defining the profile scope

Define a permitted access token scope to retrieve profile attributes.

Steps

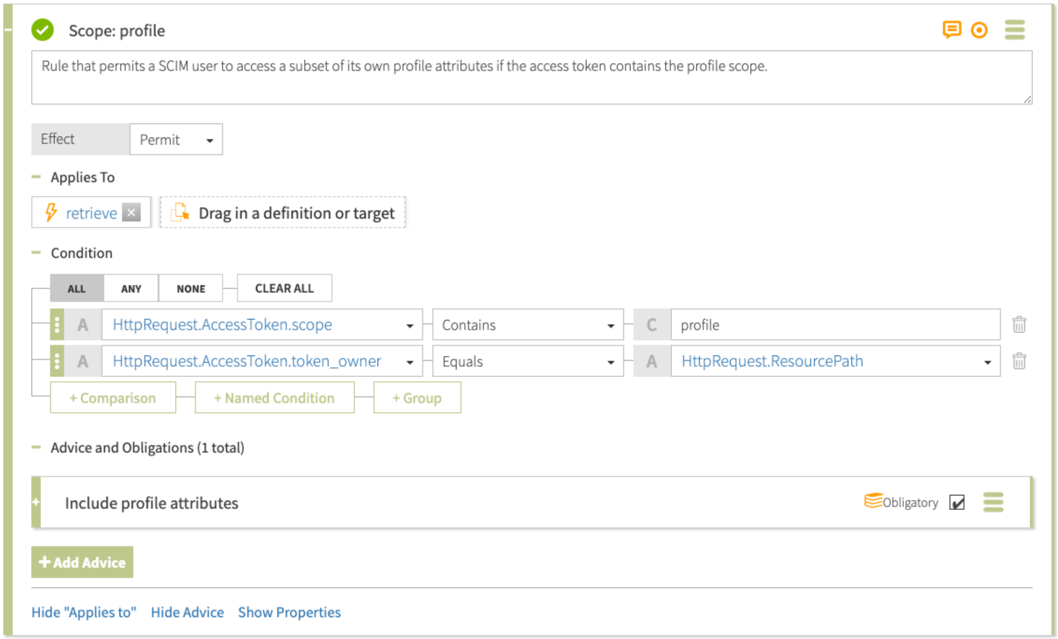
1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, **Token Policies**, and **Scope Policies**.
4. Highlight **Permitted Scopes**.
5. Click **Components**.
6. From the **Rules** list, drag **Permitted SCIM scope for user** to the **Rules** section.
7. To the right of the copied rule, click the three-line menu.
8. Click **Replace with clone**.
9. Change the name to `Scope: profile`.
10. To expand the rule, click **+**.
11. Change the description to `Rule that permits a SCIM user to access a subset of its own profile attributes if the access token contains the profile scope`.
12. In the **HttpRequest.AccessToken.scope** row of the **Condition** section, type `profile` in the **CHANGEME** field.
13. Within the rule, click **Show "Applies to"**.
14. From the **Actions** section, drag **retrieve** to the **Drag in a definition or target** box.
15. Within the rule, click **Show Advice and Obligations**.
16. Next to **Advice and Obligations**, click **+**.
17. From the **Advice** section, drag **Include profile attributes** to the **Advice and Obligations** section.

> ⓘ **Note:**
>
> This predefined advice includes a payload. If the condition for this rule is satisfied, the response includes the uid, sn, givenName, and description attributes.

18. Click **Save changes**.

Results

After completing the configuration, you will have a new profile scope, which should look like the following.



Scope: profile

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

Effect: Permit

Applies To: retrieve

Condition: ALL

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

Advice and Obligations (1 total)

Include profile attributes — Obligatory ☑

Testing the profile scope with cURL

Test your new profile scope with cURL.

Steps

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

  ```
  curl --insecure -X GET https://localhost:7443/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://localhost:7443/
  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://localhost:7443/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://localhost:7443/
  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"]}
  ```

Defining 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.

Adding the scimAdmin retrieve rule

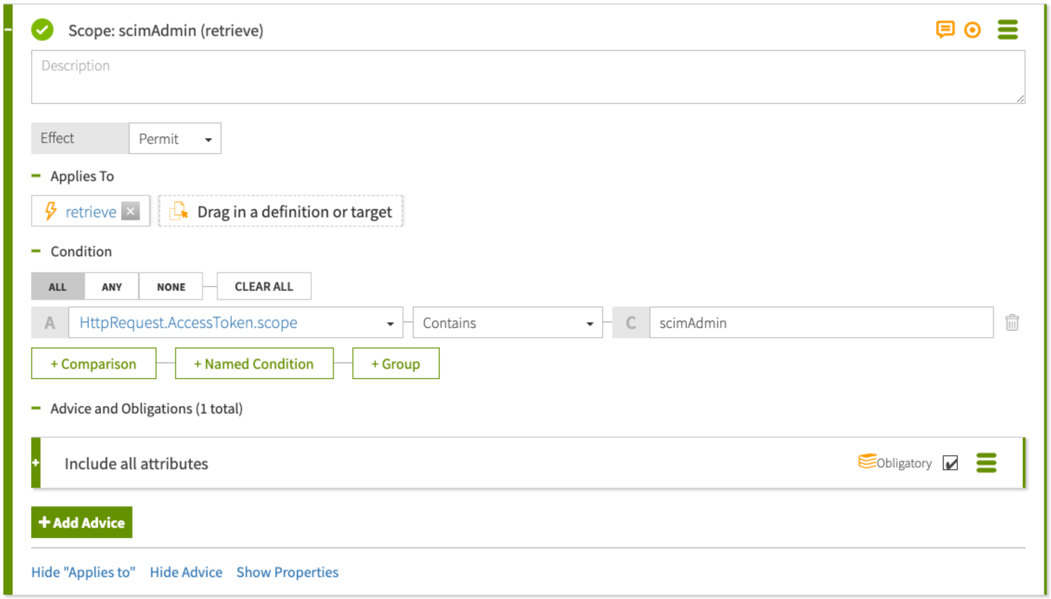
Add the scimAdmin retrieve rule to the Permitted Scopes policy.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Highlight **Permitted Scopes**.
4. Click **+ Add Rule**.
5. For the name, replace **Untitled** with `Scope: scimAdmin (retrieve).`
6. From the **Effect** list, select **Permit**.
7. In the **Condition** section, perform the following steps:
   a. Click **+ Comparison**.
   b. In the first field, select **HttpRequest.AccessToken.scope**.
   c. From the comparator list, select **Contains**.
   d. In the final field, type `scimAdmin`.
8. Within the rule, click **Show "Applies to"**.
9. Click **Components**.
10. From the **Actions** section, drag **retrieve** to the **Drag in a definition or target** box.
11. Within the rule, click **Show Advice and Obligations**.
12. Click **+** next to **Advice and Obligations**.
13. From the **Advice** section, drag **Include all attributes** to the **Advice and Obligations** section.
14. Click **Save Changes**.

Results

After completing the configuration, you will have a new scope for the scimAdmin retrieve rule, that should look like the following.



Scope: scimAdmin (retrieve)

Description

Effect: Permit

Applies To: retrieve

Condition

A HttpRequest.AccessToken.scope — Contains — C scimAdmin

Advice and Obligations (1 total)

Include all attributes — Obligatory ☑

Adding the scimAdmin create/modify rule
Add the scimAdmin create/modify rule to the Permitted Scopes policy.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Highlight **Permitted Scopes**.
4. Click **+ Add Rule**.
5. For the name, replace **Untitled** with `Scope: scimAdmin (create/modify)`.
6. From the **Effect** list, select **Permit**.
7. In the **Condition** section, perform the following steps:
   a. Click **+ Comparison**.
   b. In the first field, select **HttpRequest.AccessToken.scope**.
   c. From the comparator list, select **Contains**.
   d. In the final field, type `scimAdmin`.
8. Within the rule, click **Show "Applies to"**.
9. Click **Components**.
10. From the **Actions** section, drag **create** to the **Drag in a definition or target** box.
11. From the **Actions** sections, drag **modify** to the **Drag in a definition or target** box.
12. Click **Save Changes**.

Adding the scimAdmin search rule
Add the scimAdmin search rule to the Permitted Scopes policy.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Highlight **Permitted Scopes**.
4. Click **+ Add Rule**.
5. For the name, replace **Untitled** with `Scope: scimAdmin (search)`.
6. From the **Effect** list, select **Permit**.
7. In the **Condition** section, perform the following steps:
   a. Click **+ Comparison**.
   b. In the first field, select **HttpRequest.AccessToken.scope**.
   c. From the comparator list, select **Contains**.
   d. In the final field, type `scimAdmin`.
8. Within the rule, click **Show "Applies to"**.
9. Click **Components**.
10. From the **Actions** section, drag **search** to the **Drag in a definition or target** box.
11. Click **Save Changes**.

Adding the scimAdmin delete rule

Add the scimAdmin delete rule to the Permitted Scopes policy.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI using the URL and credentials from *Accessing the GUIs* on page 53.
2. Click **Policies**.
3. Highlight **Permitted Scopes**.
4. Click **+ Add Rule**.
5. For the name, replace **Untitled** with `Scope: scimAdmin (delete)`.
6. From the **Effect** list, select **Permit**.
7. In the **Condition** section, perform the following steps:
   a. Click **+ Comparison**.
   b. In the first field, type `HttpRequest.AccessToken.scope`.
   c. From the comparator list, select **Contains**.
   d. In the final field, type `scimAdmin`.
8. Within the rule, click **Show "Applies to"**.
9. Click **Components**.
10. From the **Actions** section, drag **delete** to the **Drag in a definition or target** box.
11. Click **Save Changes**.

Creating a policy for permitted OAuth2 clients

This tutorial describes how to configure a policy to allow specific OAuth2 clients for a REST service. A REST service typically allows only requests from a whitelist of OAuth2 clients.

About this task

In the PingDataGovernance Policy Administration GUI, define a policy in which each rule specifies an allowed client.

Steps

1. Go to **Policies# Policies**.
2. Expand **Global Decision Point** and **SCIM Policy Set**.
3. Highlight **Token Policies** and click **+** and then **Add Policy**.
4. For the name, replace **Untitled** with `Permitted Clients`.
5. From the **Combining Algorithm** list, select **Unless one decision is permit, the decision will be deny**.
6. Click **+ Add Rule**.
7. For the name, replace **Untitled** with `Client: client1`.
8. From the **Effect** list, select **Permit**.
9. In the **Condition** section:
   a. Click **+ Comparison**.
   b. From the **Select an Attribute** list, select `HttpRequest.AccessToken.client_id`.
   c. From the middle, comparison-type list, select `Equals`.
   d. In the final field, type `client1`.
10. Click **+ Add Rule**.
11. For the name, replace **Untitled** with `Client: client2`.
12. From the **Effect** list, select **Permit**.

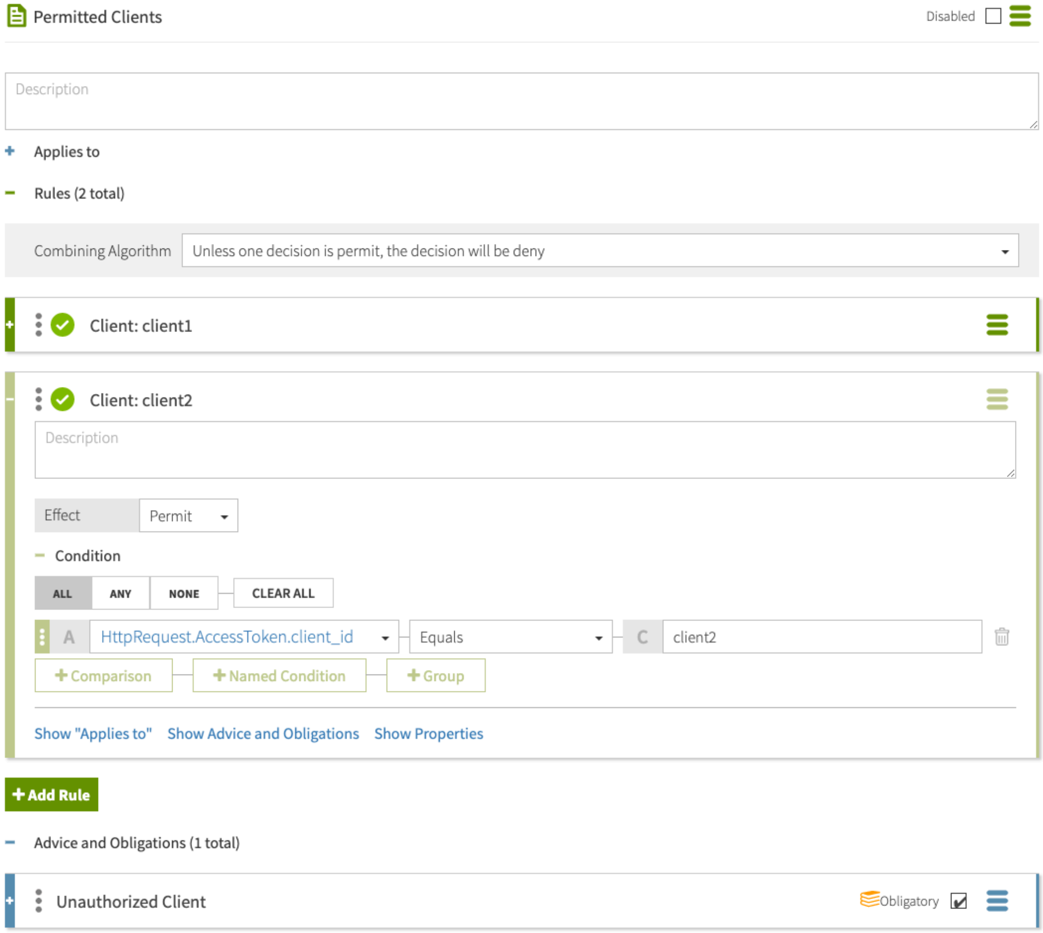
13. In the **Condition** section:
    a. Click **+ Comparison**.
    b. In the **A** field, from the **Select an Attribute** list, select **HttpRequest.AccessToken.client_id**.
    c. From the **Contains** list, select **Equals**.
    d. In the **C** field, enter `client2`.
14. Expand **+ Advice and Obligations**.

> ⓘ **Note:**
>
> Do not click **Show Advice and Obligations** within the client1 or client2 rules.

15. Click **Components**.
16. From **Advice**, drag **Unauthorized Client** to the **Advice and Obligations** box.
17. Click **Save changes**.

Results

The completed configuration should resemble the following image.


Permitted Clients

Disabled □

Description

Applies to

Rules (2 total)

Combining Algorithm: Unless one decision is permit, the decision will be deny

Client: client1

Client: client2

Description

Effect: Permit

Condition

A: HttpRequest.AccessToken.client_id | Equals | C: client2

Advice and Obligations (1 total)

Unauthorized Client | Obligatory ☑

Testing the client policy with cURL

To confirm that you successfully completed the tasks from the previous section, test the client policy with cURL.

About this task

After completing the tasks in the previous sections, test the responses you receive for access tokens for any client other than client1 or client2.

Steps

- To test that an access token for any client other than client1 or client2 is rejected, run the following.

  ```
  curl --insecure -X GET https://localhost:7443/scim/v2/Me -H
   'Authorization: Bearer {"active": true, "sub": "user.1", "scope":
   "email", "client_id": "nonexistent.client"}'
  ```

  Successful completion of the tasks in the previous sections will result in the following response.

  ```
  {"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"}
  ```

- To test that an access token for client1 is accepted, run the following.

  ```
  curl --insecure -X GET https://localhost:7443/scim/v2/Me -H
   'Authorization: Bearer {"active": true, "sub": "user.1", "scope":
   "email", "client_id": "client1"}'
  ```

  Successful completion of the tasks in the previous sections will result in the following response.

  ```
  {"id":"355a133d-58ea-3827-8e8d-b39cf74ddb3e","meta":
  {"resourceType":"Users","location":"https://localhost:7443/
  scim/v2/Users/355a133d-58ea-3827-8e8d-b39cf74ddb3e"},"schemas":
  ["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"],"mail":
  ["user.1@example.com"]}
  ```

Creating a policy for permitted audiences

This tutorial describes how to create a policy for a REST service to control access based on an acceptable audience value.

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. Go to **Policies# Policies**.
2. Expand **Global Decision Point** and **SCIM Policy Set**.
3. Highlight **Token Policies** and click **+**.
4. Click **Add Policy**.
5. For the name, replace **Untitled** with `Permitted Audiences`.
6. From the **Combining Algorithm** list, select **Unless one decision is permit, the decision will be deny**.
7. Click **+ Add Rule**.
8. For the name, replace **Untitled** with `Audience: https://example.com`.
9. From the **Effect** list, select `Permit`.

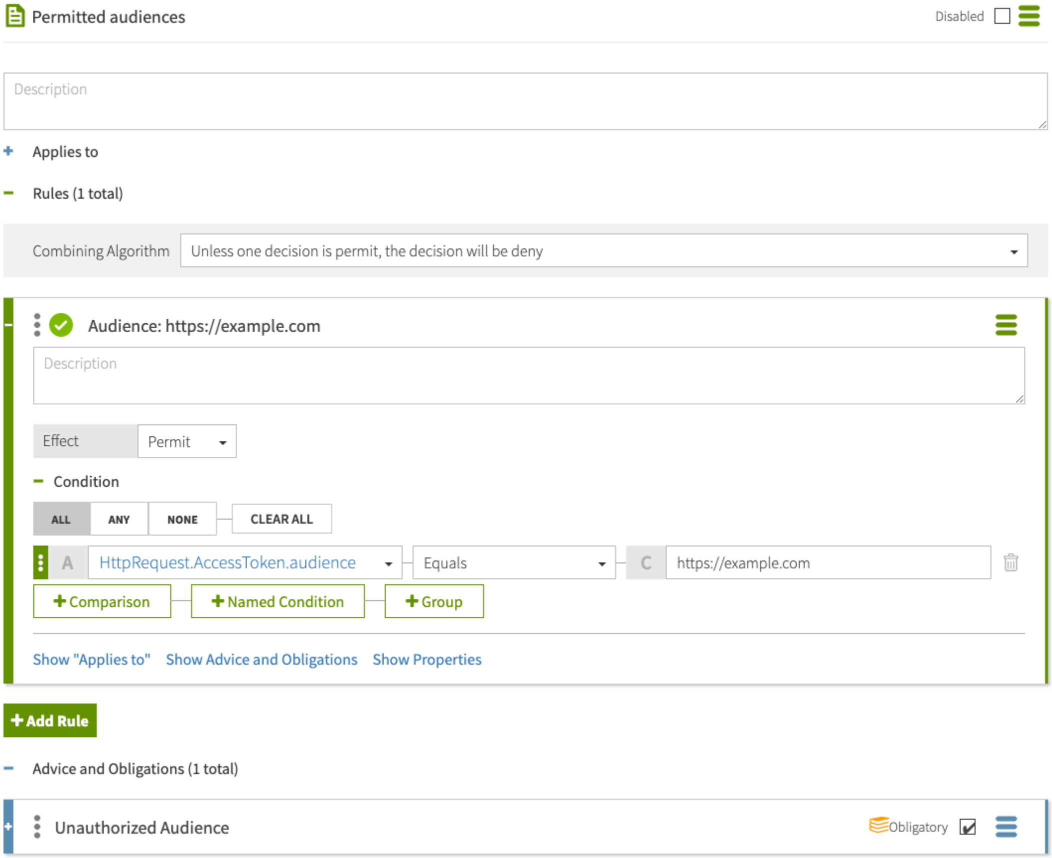
10. In the **Condition** section:
    a. Click **+ Comparison**.
    b. From the **Select an Attribute** list, select `HttpRequest.AccessToken.audience`.
    c. From the middle, comparison-type list, select `Equals`.
    d. In the **C** field, enter `https://example.com`.
11. Expand **+ Advice and Obligations**.
12. Click the **Components** tab, expand **Advice**, and drag `Unauthorized Audience` to the **Advice and Obligations** box.

> **Note:**
>
> Do not click **Show Advice and Obligations** within the "Audience: https://example.com" rule.

13. Click **Save changes**.

Results

The final configuration should resemble the following image.



Permitted audiences

Disabled ☐

Description

+ Applies to

− Rules (1 total)

Combining Algorithm: Unless one decision is permit, the decision will be deny

Audience: https://example.com

Description

Effect: Permit

− Condition

ALL | ANY | NONE | CLEAR ALL

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

− Advice and Obligations (1 total)

Unauthorized Audience — Obligatory ☑

Testing the audience policy with cURL

Test the audience policy with cURL.

Steps

1. To test that an access token without a specific audience value is rejected, run the following.

   ```
   curl --insecure -X GET https://localhost:7443/scim/v2/Me -H
    'Authorization: Bearer {"active": true, "sub": "user.1", "scope":
    "email", "client_id": "client1"}'
   ```

   Successful creation of the audience policy will result in the following.

   ```
   {"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."}
   ```

2. To test that an access token with an audience value of `https://example.com` is accepted, run the following.

   ```
   curl --insecure -X GET https://localhost:7443/scim/v2/Me -H
    'Authorization: Bearer {"active": true, "sub": "user.1", "scope":
    "email", "client_id": "client1", "aud": "https://example.com"}'
   ```

   Successful creation of the audience policy will result in the following.

   ```

   {"id":"355a133d-58ea-3827-8e8d-b39cf74ddb3e","meta":
   {"resourceType":"Users",
   "location":"https://localhost:7443/scim/v2/Users/355a133d-58ea-3827-8e8d-
   b39cf74ddb3e"},
   "schemas":["urn:pingidentity:schemas:store:2.0:UserStoreAdapter"],"mail":
   ["user.1@example.com"]}
   ```

Tutorial: Creating a policy for role-based access control

This tutorial describes how to create the final policy, which is an access-control rule that can base its authorization decision on an attribute of the requesting identity, rather than on an access token claim.

About this task

When PingDataGovernance Server authorizes a request, an access token validator resolves the subject of the access token to a System for Cross-domain Identity Management (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. Go to **Trust Framework** and click the **Attributes** tab. Click **TokenOwner**.
2. Click **+**.
3. Click **Add new Attribute**.
4. For the name, replace **Untitled** with `employeeType`.
5. From the **Parent** list, select `TokenOwner`.
6. In the **Resolvers** section:
   a. Click **+ Add Resolver**.
   b. From the **Resolver type** list, select `Attribute` and in the **Select an Attribute** list, specify a value of `TokenOwner`.

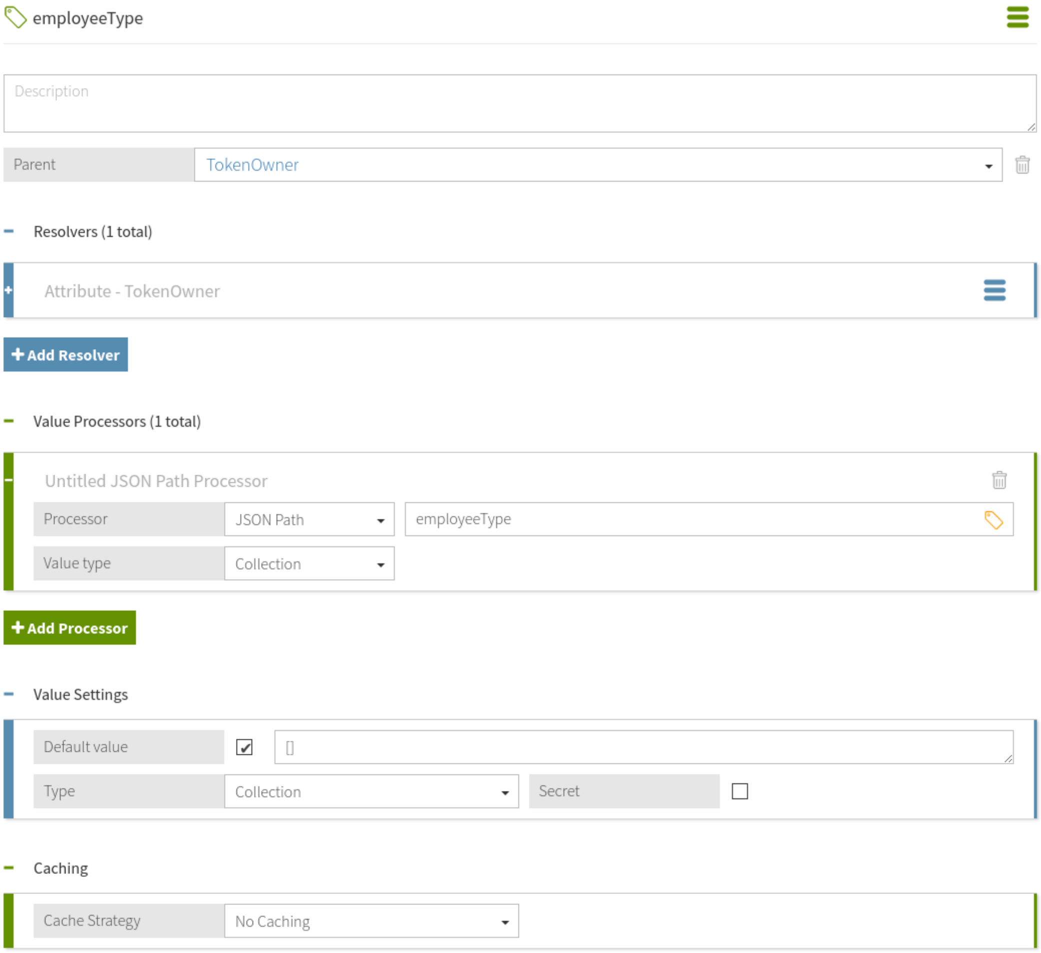
7. Click **+** next to **Value Processors** and then **+ Add Processor**.
8. From the **Processor** list, select `JSON Path` and enter the value `employeeType`.
9. Set the **Value type** to `Collection`.
10. In the **Value Settings** section:
    a. Select the **Default Value** check box and in the **Enter a default value** field, enter the value `[]`.

    > ⓘ **Note:**
    >
    > An empty array is specified as the default value because not all users have an `employeeType` attribute. A default value of `[]` ensures that policies can safely use this attribute to define conditions.

    b. From the **Type** list, select `Collection`.
11. Click **Save changes**.

Results

The final policy configuration should resemble the following image.



employeeType

Description

Parent: TokenOwner

Resolvers (1 total)

Attribute - TokenOwner

Value Processors (1 total)

Untitled JSON Path Processor

Processor: JSON Path — employeeType

Value type: Collection

Value Settings

Default value ☑ []

Type: Collection — Secret □

Caching

Cache Strategy: No Caching

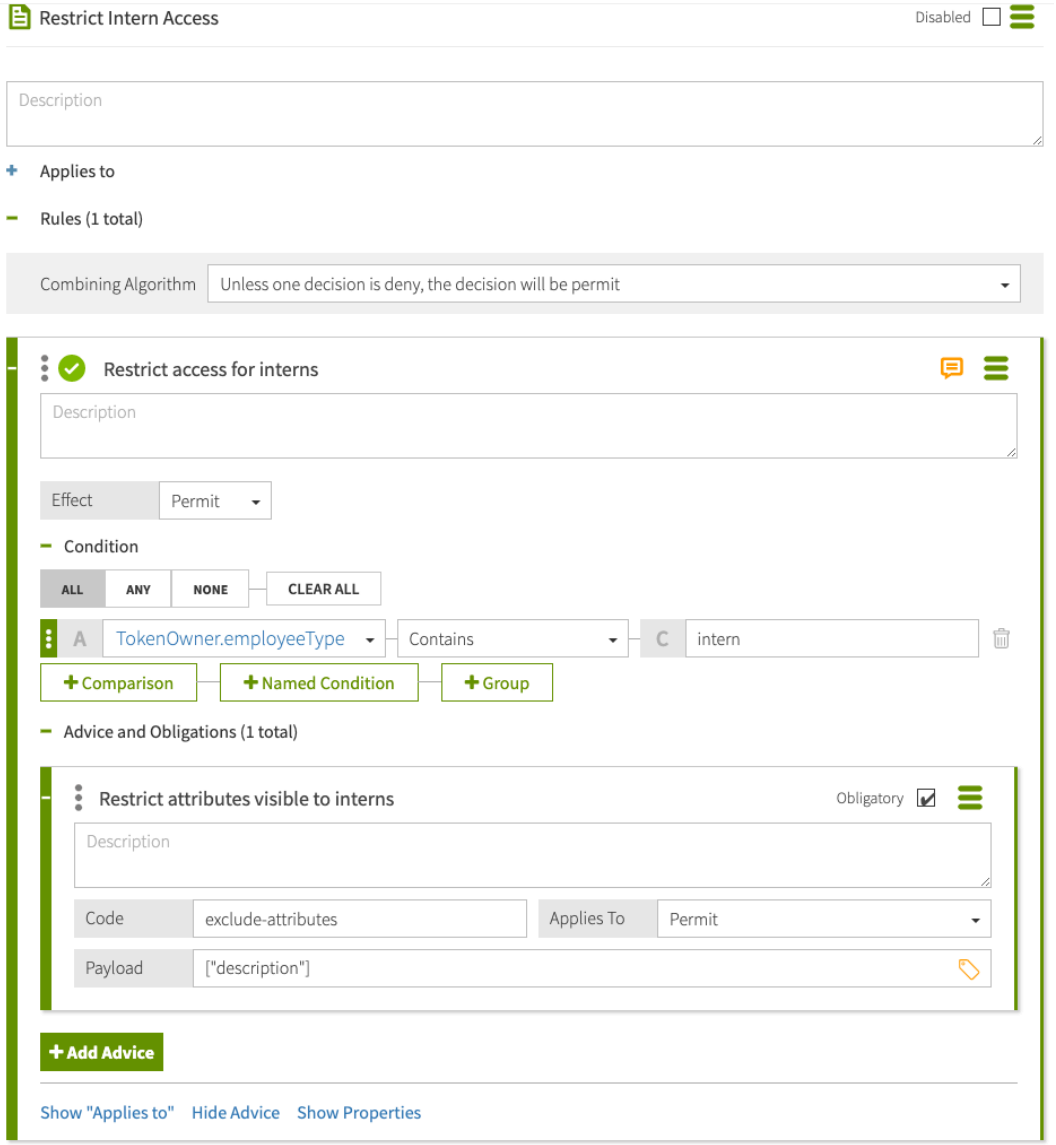
Next steps

Add a policy that uses the `employeeType` attribute.

1. Go to **Policies**, highlight **SCIM Policy Set** and click **+**.
2. Click **Add Policy**.
3. For the name, replace **Untitled** with `Restrict Intern Access`.

4. From the **Combining Algorithm** list, select **Unless one decision is deny, the decision will be permit**.
5. Click **+ Add Rule**.
6. For the name, replace **Untitled** with `Restrict access for interns`.
7. From the **Effect** list, select `Permit`.
8. In the **Condition** section:

   a. Click **+ Comparison**.
   b. In the **Select an Attribute** field, select `TokenOwner.employeeType`.
   c. From the middle, comparison-type list, select `Contains`.
   d. In the **Type in constant value** field, enter `intern`.
9. Within the rule, click **Show Advice and Obligations** and then click the **+** next to **Advice and Obligations**.
10. Click **+ Add Advice**# **Custom Advice**.
11. For the name, replace **Untitled** with `Restrict attributes visible to interns`.
12. Select the **Obligatory** check box.
13. In the **Code** field, enter `exclude-attributes`.
14. From the **Applies To** list, select `Permit`.
15. In the **Payload** field, enter `["description"]`.
16. Click **Save Changes**.



Restrict Intern Access

Disabled ☐

Description

Applies to

Rules (1 total)

Combining Algorithm: Unless one decision is deny, the decision will be permit

Restrict access for interns

Description

Effect: Permit

Condition

ALL | ANY | NONE | CLEAR ALL

A: TokenOwner.employeeType | Contains | C: intern

Advice and Obligations (1 total)

Restrict attributes visible to interns

Obligatory ☑

Description

Code: exclude-attributes

Applies To: Permit

Payload: ["description"]

Testing the policy with cURL

Test the policy for role-based access control using cURL.

About this task

The PingDataGovernance sample user data allows an `employeeType` attribute but does not populate it with values for any users.

Confirm that `user.2` cannot read the `description` attribute, even though the `profile` scope allows it by running the following command.

```
curl --insecure -X GET https://localhost:7443/scim/v2/Me -H 'Authorization:
 Bearer {"active": true, "sub": "user.2", "scope": "profile", "client_id":
 "client1", "aud": "https://example.com"}'
```

The response should be similar to the following response.

```
{"id":"c9cbfb8c-d915-3de3-8a2c-a01c0ccc6d09","meta":
{"resourceType":"Users","location":"https://localhost:7443/
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.

Conclusion

In this tutorial, you set scope-based access to SCIM resources.

You also learned:

- Like `exclude-attributes` used in this tutorial, `include-attributes` filters which attributes can be returned to the caller. `include-attributes` works more like opt-in, while `exclude-attributes` works more like opt-out.
- Multiple attributes can apply from multiple rules or even policies. They are combined by PingDataGovernance to `include` before `exclude`.

Installing PingDataGovernance Server

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

Seeing PingDataGovernance in action

To quickly see PingDataGovernance in action, see *Getting started with PingDataGovernance (tutorials)* on page 52.

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

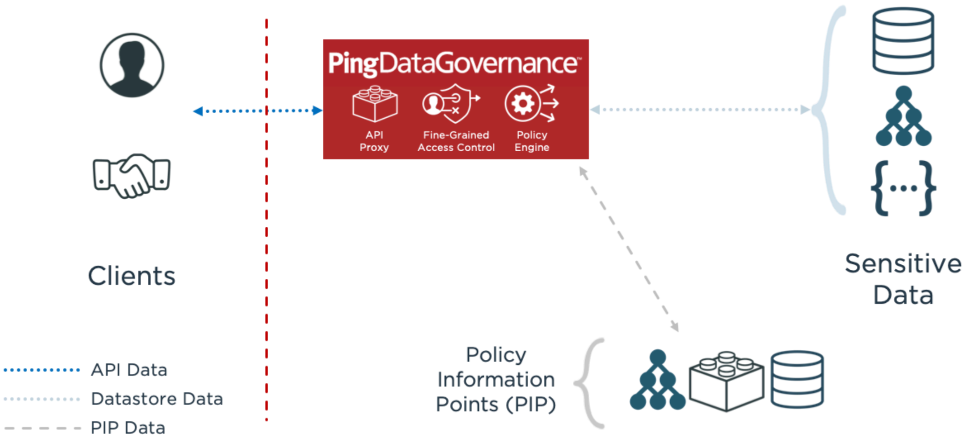
PingDataGovernance Server supports the following options of deployment architectures for enforcing fine-grained access to data:

- System for Cross-domain Identity Management (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.

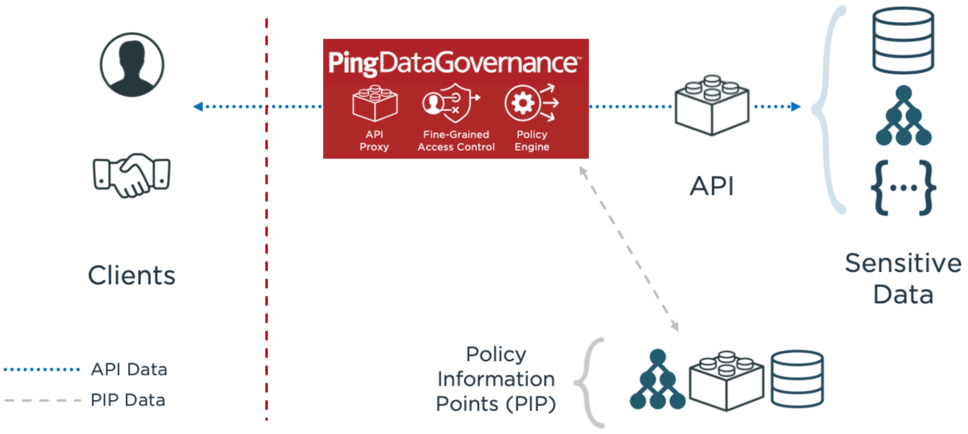
SCIM API to datastores

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. The policy is enforced by the SCIM service.



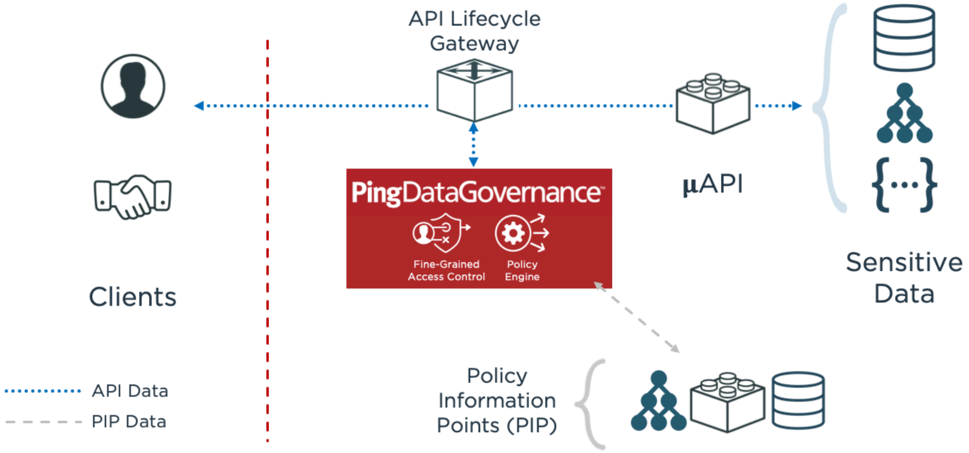
API Security Gateway as reverse proxy

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. The 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

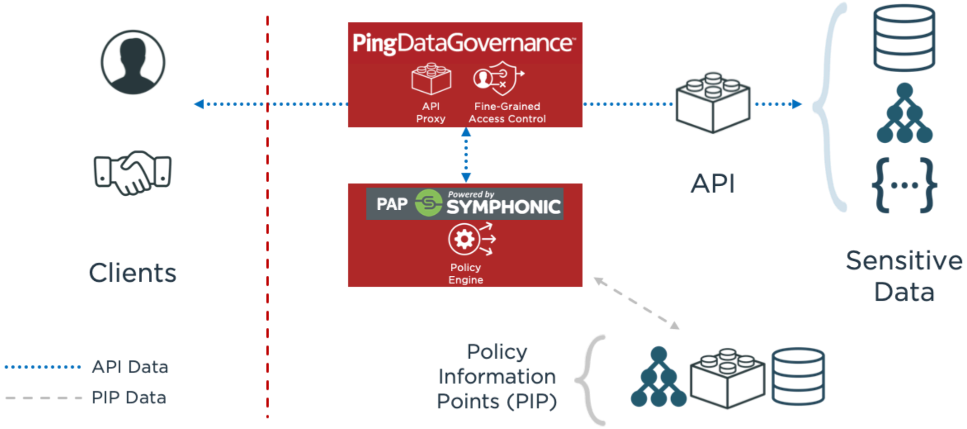
After policies are developed, they are tested in other pre-production environments and eventually put into production.

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.

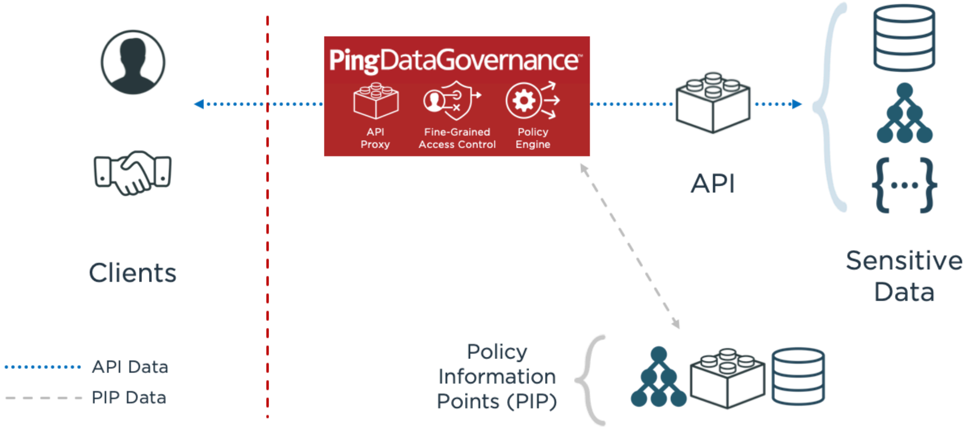


As test API requests are proxied through PingDataGovernance Server's API security gateway, policy decisions are obtained from the Policy Administration GUI and are enforced by the API security gateway.

Other pre-production and production environments

The Policy Administration GUI is not a part of so-called "higher" environments. Instead, the 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

You must install certain components before you can install PingDataGovernance Server.

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

Ensure that your system meets the minimum requirements for PingDataGovernance.

Ping Identity has qualified the configurations in this section and has certified that they are compatible with the product. PingDataGovernance supports differences in operating system versions, service packs, and other platform variations until the platform or other required software is suspected of causing issues.

Platforms

You can run PingDataGovernance on a variety of different platforms and operating systems.

- Windows Server 2019
- Windows Server 2016
- Red Hat Enterprise Linux ES 8.1
- Red Hat Enterprise Linux ES 7.7
- CentOS 8.1
- CentOS 7.7
- SUSE Linux Enterprise 15 SP1
- SUSE Linux Enterprise 12 SP5
- Ubuntu 18.04 LTS
- Ubuntu 16.04 LTS
- Amazon Linux 2

Note:

This product was tested with the default configurations of all operating system components. Customized implementations or third-party plugins might affect the deployment of this product.

Docker

You can review the version, host operating system, and base image operating system specifications for running Docker.

- Version: Docker 19.03.5
- Host operating system: Ubuntu 18.04 LTS
- Base image operating system: Ubuntu 18.04 LTS
- Base image operating system: Alpine Linux 3.11

View the PingDataGovernance Docker Image on *DockerHub: PingDataGovernance*. View the PingDirectory Docker Image on *DockerHub: PingDirectory*. Visit the PingIdentity DevOps *documentation* for more information. Note that only the PingDataGovernance and PingDirectory software is licensed under Ping Identity's end user license agreement, and any other software components contained within the image are licensed solely under the terms of the applicable open source/third party license.

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

Make sure your Java Runtime Environment (JRE) meets the system requirements for PingDataGovernance.

- Amazon Corretto 8
- OpenJDK 11, obtained from AdoptOpenJDK (*https://adoptopenjdk.net/*)

- OpenJDK 8, obtained from AdoptOpenJDK (*https://adoptopenjdk.net/*)
- Oracle Java SE Development Kit 11 LTS
- Oracle Java SE Development Kit 8

ⓘ **Note:**

The *Ping Identity Java Support Policy* applies to your JRE.

Browsers

The PingDataGovernance Administrative Console is compatible with several different web browsers.

Administrative Console

- Chrome
- Firefox
- Internet Explorer 11 and later

About license keys

License keys are required to install, update, and renew all Ping products.

How to obtain a license

To obtain a license key, contact your account representative or use the *Ping Identity licensing portal*.

When do you need a license

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, you must obtain a new license when updating a server to a new major version, such as when upgrading from 7.3 to 8.0. When cloning a server instance with a valid license, you do not need a new license.

ⓘ **Note:**

The update process displays a prompt for a new license.

How to specify a license

- Specify a license at setup

  You have these options:

  - Use the `--licenseKeyFile` *<path-to-license>* option with **`setup`**.
  - Copy the license file to the PingDataGovernance Server root directory and then run the **`setup`** tool. The tool discovers the license file.
- Specify a license after setup

  Use the Administrative Console or **`dsconfig`** (in the Topology section, select License).

  ⓘ **Note:** Placing the new license file in the PingDataGovernance Server root directory does not work in this case.

For information about how to specify the license with the Policy Administration GUI, see *Installing the PingDataGovernance Policy Administration GUI noninteractively* on page 124.

How to view the license status

To view the details of a license, including its expiration, you have these options:

- The server's `status` tool
- The Administrative Console's **Status** page (On the **Monitors** tab, search for License.)

License expiration

The server provides a notification as the expiration date approaches.

Before a license expires, obtain a new one and install it by using `dsconfig` or the Administrative Console.

ⓘ **Note:**

An expiring license causes alerts and alarms but does not affect the functionality of PingDataGovernance Server.

However, PingDataGovernance Policy Administration GUI fails to start if the license has expired.

Installing Java

Create a Java installation for PingDataGovernance Server using the Java Development Kit (JDK).

About this task

PingDataGovernance Server requires Java for 64-bit architectures. Even if Java is already installed on your system, you should 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 Java Runtime Environment (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

Preparing a Linux environment in PingDataGovernance Server requires you to complete a series of tasks, as described in this section

About this task

Complete the following tasks before you install PingDataGovernance Server in a Linux environment:

Steps

1. Set the file descriptor limit
2. Set the maximum user processes
3. Disable file system swapping
4. Manage system entropy
5. Enable the server to listen on privileged ports

Setting the file descriptor limit

PingDataGovernance Server allows for an unlimited number of connections. The following steps describe how to manually increase the file descriptor limit on the operating system.

About this task

Note:

If the operating system relies on **`systemd`**, see the Linux operating system documentation for instructions on setting the file descriptor limit.

Steps

1. Display the current `fs.file-max` limit of the system.

   ```
   sysctl fs.file-max
   ```

   The `fs.file-max` limit is the maximum server-wide file limit you can set without tuning the kernel parameters in the `proc` file system.
2. Edit the `/etc/sysctl.conf` file.

   If there is a line that sets the value of the `fs.file-max` property, make sure that its value is set to at least 1.5 times the per-process limit. If there is no line that sets a value for this property, add the following to the end of the file (100000 is just an example here; specify a value of at least 1.5 times the per-process limit).

   ```
   fs.file-max = 100000
   ```
3. Display the current hard limit of the system.

   ```
   ulimit -aH
   ```

   The `open files (-n)` value is the maximum number of open files per process limit.

   Verify that its value is set to at least 65535.
4. 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, inserting a tab between the columns.

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

   Note:

   The number of open file descriptors is limited by the physical memory available to the host. You can determine this limit with the following command.

   ```
   cat /proc/sys/fs/file-max
   ```

   If the `file-max` value is significantly higher than the 65535 limit, consider increasing the file descriptor limit to between 10% and 15% of the system-wide file descriptor limit. For example, if the `file-max` value is 810752, you could set the file descriptor limit to 100000. If the `file-max` value is lower than 65535, the host is likely not sized appropriately.
5. Reboot the server.

6. Verify that the file descriptor limit is set to 65535.

   ```
   ulimit -n
   ```

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

   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
   ```

Next steps

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_DESCRIPTORS environment variable.
- Create a `config/num-file-descriptors` file with a single line, such as `NUM_FILE_DESCRIPTORS=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.

Setting the maximum user processes

Set the maximum user processes higher than the default to improve memory when running multiple servers on a machine.

About this task

On some Linux distributions, such as RedHat Enterprise Linux (RHEL) 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 Java virtual machine (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, you must explicitly set the limit in `/etc/security/limit.conf`, as the following example shows.

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

Steps

- Set the 1683 value in the NUM_USER_PROCESSES environment variable.
- Set the 1683 value in `config/num-user-processes`.

Disabling file system swapping

To disable the file system swapping in PingDataGovernance, use `vm.swappiness`.

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`**.

Managing system entropy

Entropy is used to calculate random data that the system uses in cryptographic operations.

About this task

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.

Steps

- 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 a system entropy on a UNIX or Linux system, run the following command.

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

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

Enabling the server to listen on privileged ports

To enable PingDataGovernance Server to listen on privileged ports as a non-root user, grant capabilities to specific commands.

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 convenience, you might enable the server to listen on privileged ports while running as a non-root user.

Steps

- To assign capabilities to an application, run the **`setcap`** command.

  For example, 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 you can grant the Java binary the **`cap_net_bind_service`** capability by running 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.

  For example, if the Java installation is located in `/ds/java`, the contents must be as shown in this example.

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

  Run the following command for the change to take effect.

  ```
  $ sudo ldconfig -v
  ```

Installing PingDataGovernance Server

Install PingDataGovernance Server manually, using server profiles in an automated environment or the Docker tool.

Steps

Choose from the following installation options:

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

Installing PingDataGovernance Server manually

Obtain the installation packages and manually install the PingDataGovernance Server and the PingDataGovernance Policy Administration GUI.

About this task

After you prepare your hardware and software system based on the preceding instructions, start the setup process for PingDataGovernance Server:

Steps

1. Obtain the PingDataGovernance Server installation packages.
2. Install PingDataGovernance Server.
3. Install the PingDataGovernance Policy Administration GUI.
4. Perform additional configuration steps.

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

Obtaining the installation packages

To begin the installation process for PingDataGovernance, obtain the server component's 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, complete the following steps.

Steps

1. Obtain the latest compressed release bundles from Ping Identity.
2. Expand the release bundles into the folders of your choice.

Installing the server

Extract the build distribution bundle and install PingDataGovernance Server.

About this task

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. Extract the compressed `.zip` archive to a directory of your choice.

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

Results

You can set up PingDataGovernance Server.

About the server installation modes

There are several different installation modes for PingDataGovernance Server.

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 Java virtual machine (JVM) runtime settings and assigning listener ports for the PingDataGovernance Server's HTTP services.
- The **`create-initial-config`** tool configures connectivity between a System for Cross-domain Identity Management (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.

> ⓘ **Note:**
>
> Using **`create-initial-config`** is optional. However, if you do not use it, you do not get the user's profile (the requester's attributes). For more information, see *Make a user's profile available in policies* on page 241.

- 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.

Noninteractive 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

You can perform all installation and configuration steps while signed on to the system as the user or the role under which PingDataGovernance Server will run.

Installing the server interactively

Run the **`setup`** tool, which prompts you interactively for the information that it needs to install PingDataGovernance Server.

Before you begin

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 distinguished name (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 Java virtual machine (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 in Unix/Linux* on page 143.

Installing the server noninteractively

For an automated installation, run the **`setup`** tool in noninteractive, command-line mode.

Before you begin

Be prepared to provide the following settings using command-line arguments:

- The location of a valid license file
- The name and password for an administrative account, which is also called the root user distinguished name (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 Java virtual machine (JVM)
- A unique instance name for the server

Steps

- Run the **`setup`** tool to install the server noninteractively.
- For more information about the available setup options, run **`setup`** with the `--help` argument, which displays a complete list of setup options, along with examples.

  ```
  $ ./setup --help
  ```

Example

The following example sets up PingDataGovernance with these settings:

- LDAP port 8389
- LDAPS port 8636
- HTTPS port 8443
- An automatically generated self-signed server certificate
- 1 GB of memory reserved for the server's JVM
- A unique server instance name of `dg1`
- A server location of `Austin`

```
$ ./setup \
  --cli --no-prompt --acceptLicense \
  --licenseKeyFile <path-to-license> \
  --rootUserDN "cn=directory manager" \
  --rootUserPassword <your-password> \
  --ldapPort 8389 --ldapsPort 8636 \
  --httpsPort 8443 \
  --generateSelfSignedCertificate \
  --maxHeapSize 1g \
  --instanceName dg1 \
  --location Austin
```

Signing on to the Administrative Console

After you install the server, access the Administrative Console to verify the configuration and to manage the server

Steps

1. To access the Administrative Console, go to `https://<host>:<port>/console/login`.

   The default port is 8443.
2. To sign on to the Administrative Console, use the initial root user distinguished name (DN) and root user password specified during setup.

   The default DN is `cn=Directory Manager`.

Installing PingDataGovernance Policy Administration GUI

Unpack the build distribution and extract the compressed archive to install PingDataGovernance Policy Administration GUI.

About this task

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 set up PingDataGovernance Policy Administration GUI.

Installing the PingDataGovernance Policy Administration GUI interactively

You can run the PingDataGovernance Policy Administration GUI's **`setup`** command interactively in command-line mode.

About this task

> ⓘ **Note:**
>
> You cannot configure some setup options when installing the PingDataGovernance Policy Administration GUI interactively. See *Installing the PingDataGovernance Policy Administration GUI noninteractively* on page 124.

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
- An available port for the PingDataGovernance Policy Administration GUI to accept HTTPS requests

Steps

1. 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 OpenID Connect (OIDC) mode, this mode does not require an external authentication server. However, it is inherently insecure and is recommended only for demonstration purposes.
   - OIDC mode

     Configures the PingDataGovernance Policy Administration GUI to delegate authentication and sign-on services to a PingFederate OIDC provider.

     To use PingDataGovernance Policy Administration GUI with other OIDC providers, such as PingOne, see *Installing the PingDataGovernance Policy Administration GUI noninteractively* on page 124.
2. If you choose OIDC mode, be prepared to provide the following additional information:

   - The host name and port of an OIDC 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 trust store.
3. Run the **`setup`** command.

   > ⓘ **Note:**
   >
   > If you do not want to use the default database credentials, see *Setting database credentials at initial setup* on page 222.
4. Complete the steps in *Post-setup steps* on page 126.

5. Consider additional configuration options in *Specifying custom configuration with an options file* on page 213.

Installing the PingDataGovernance Policy Administration GUI noninteractively

For an automated installation, run the PingDataGovernance Policy Administration GUI's **`setup`** command in noninteractive, command-line mode.

About this task

ⓘ **Note:**

You must run setup in noninteractive, command-line mode instead of interactive mode if you need to do any of the following:

- Configure the Policy Administration GUI with a policy configuration key.
- Configure a key store for a policy information provider.
- Configure a trust store for a policy information provider.
- Customize the Policy Administration GUI's logging behavior.

For more information, see *Specifying custom configuration with an options file* on page 213.

Steps

1. Before you run **`setup`**, 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.
2. Optional: If you choose OIDC mode, be prepared to provide the following additional information:
    - The host name and port of an OpenID Connect provider or its base URL
    - The location of a trust store for the OpenID Connect provider, if the OpenID Connect provider uses a server certificate not issued by a certificate authority not trusted by the JRE
3. Optional: If you do not use the **`setup`** tool to generate a self-signed certificate, you must also provide the following:
    - Information related to the PingDataGovernance Policy Administration GUI's connection security, including the location of a keystore that contains the server certificate and the nickname of that server certificate.

    ⓘ **Note:**

    The **`setup`** tool's `--help` option displays the options available for a noninteractive installation.

4. Run the correct command based on your needs:

> **Note:**
>
> If you do not want to use the default database credentials, see *Setting database credentials at initial setup* on page 222.

- To see the general options for running **setup**:

```
$ bin/setup --help
```

- To see the options for running setup in demo mode:

```
$ bin/setup demo --help
```

- To see the options for running setup in OIDC mode:

```
$ bin/setup oidc --help
```

5. After you complete setup, see *Post-setup steps* on page 126.
6. Consider additional configuration options in *Specifying custom configuration with an options file* on page 213.

Example: Set up the PingDataGovernance Policy Administration GUI in demo mode
This example sets up the PingDataGovernance Policy Administration GUI in demo mode with an automatically generated self-signed server certificate.

After completing setup, the Policy Administration GUI will accept sign-ons using the username `admin` and the password `password123`.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret datagovernance \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --licenseKeyFile <path-to-license>
```

The decision point shared secret is a credential that the PingDataGovernance Server uses to authenticate to the Policy Administration GUI when it uses the Policy Administration GUI as an external policy decision point (PDP). For information about how to configure PingDataGovernance Server to use the decision point shared secret, see *Post-setup steps* on page 126.

Example: Set up the PingDataGovernance Policy Administration GUI in OIDC mode (PingFederate)
Use this example as a reference to set up the PingDataGovernance Policy Administration GUI to handle sign-ons using a PingFederate OpenID Connect (OIDC) provider.

```
$ bin/setup oidc \
  --oidcHostname <ping-federate-hostname> \
  --oidcPort <ping-federate-port> \
  --clientId pingdatagovernance-pap \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret datagovernance \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --licenseKeyFile <path-to-license>
```

The Policy Administration GUI uses the provided OIDC host name and OIDC to query the PingFederate server's autodiscovery endpoint for the information it needs to make OIDC requests. The provided client ID represents the Policy Administration GUI and must be configured in PingFederate. For more information about configuring PingFederate, see *Configuring an Authentication Server for OpenID Connect single sign-on* on page 127.

Example: Set up the PingDataGovernance Policy Administration GUI in OIDC mode (generic OpenID Connect provider)
This example sets up the PingDataGovernance Policy Administration GUI to handle sign-ons using an arbitrary OpenID Connect (OIDC) provider.

This example departs from the previous example by specifying the OIDC provider's base URL, rather than a host name and port. This can be useful if the OIDC provider's autodiscovery and authorization endpoints include an arbitrary prefix, such as a customer-specific environment identifier.

```
$ bin/setup oidc \
  --oidcBaseUrl https://auth.example.com/9595f417-a117-3f24-
a255-5736ab01f543/auth/ \
  --clientId 7cb9f2c9-c366-57e0-9560-db2132b2d813 \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret datagovernance \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --licenseKeyFile <path-to-license>
```

The Policy Administration GUI uses the provided OIDC base URL to query the OIDC provider's autodiscovery endpoint for the information it needs to make OIDC requests. The provided client ID represents the Policy Administration GUI and must be configured in the OIDC provider as well. For more information about configuring an OIDC provider, see *Configuring an Authentication Server for OpenID Connect single sign-on* on page 127.

Post-setup steps
After you successfully set up the PingDataGovernance Policy Administration GUI, you must start the server and then configure PingDataGovernance Server to use the Policy Administration GUI as its policy decision point (PDP).

To start the Policy Administration GUI, run the following command.

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

Then, sign on to the Policy Administration GUI. For more information, see *Signing on to the PingDataGovernance Policy Administration GUI* on page 127 and import a policy snapshot. You can find a set of default policies in the `resource/policies/defaultPolicies.SNAPSHOT` file.

To configure PingDataGovernance Server to use the Policy Administration GUI, use **`dsconfig`** or the Administrative Console to create a Policy External Server to represent the Policy Administration GUI, then assign the Policy External Server to the Policy Decision Service and configure it to use external PDP mode. Also, set the Trust Framework Version to the current version, v2. Consider the following example.

```
dsconfig create-external-server \
  --server-name "Policy Administration GUI" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:datagovernance" \
  --set "branch:Default Policies" \

dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:Policy Administration GUI"
  --set trust-framework-version:v2
```

In the example, the base URL consists of the host name and port chosen for the Policy Administration GUI during setup. Similarly, the shared secret value was chosen during setup. The branch name corresponds to the branch name that you chose when importing your policy snapshot. The decision node is the ID of the root node in your policy tree. If you are using the default policies, then use the ID shown in the example.

Signing on to the PingDataGovernance Policy Administration GUI

You can sign on to the PingDataGovernance Policy Administration GUI by entering your username and password credentials in the appropriate web browser URL.

About this task

Steps

1. After completing setup for demo mode, sign on to the PingDataGovernance Policy Administration GUI by going to the following URL in a web browser: `https://<host>:<port>`

   Substitute the host name and port that you specified during setup.
2. Use the following demo credentials to sign on to the PingDataGovernance Policy Administration GUI:
   - User name: `admin`
   - Password: `password123`
3. Optional: If you set up the PingDataGovernance Policy Administration GUI to use OpenID Connect (OIDC) mode, you must also configure an OIDC provider. For more information, see *Configuring an Authentication Server for OpenID Connect single sign-on* on page 127.

   Then, when you sign on using the URL mentioned previously, the GUI prompts you to proceed to the OIDC provider to sign on. After OIDC authentication is complete, the GUI redirects you back to the PingDataGovernance Policy Administration GUI.

Changing the Policy Administration GUI authentication mode

You can change the authentication mode after the initial setup.

About this task

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 and select a different authentication mode.

   ```
   $ bin/setup
   ```
3. Start the Policy Administration GUI.

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

Configuring an Authentication Server for OpenID Connect single sign-on

You can configure an Open ID Connect (OIDC) provider to accept sign-on requests in PingDataGovernance.

About this task

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

For information about using	See
PingFederate	*Configuring PingFederate as an OIDC provider for PingDataGovernance policy administration*
PingOne	*Configuring PingOne as an OIDC provider for PingDataGovernance policy administration*

Steps

1. 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	▪ `openid` ▪ `email` ▪ `profile`

   a. Configure the access tokens and ID tokens issued for this client with the following claims:

   - `sub`
   - `name`
   - `email`
2. Configure the OIDC provider to accept a cross-origin resource sharing (CORS) origin that matches the PingDataGovernance Policy Administration GUI's scheme, public host, and port, such as `https://<host>:<port>`.
3. Configure the OIDC 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.

   ⓘ **Note:** Sign the tokens with a signing algorithm of RSA using SHA256.

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

Additional configuration steps

After you have installed the PingDataGovernance Server components, you must perform some additional steps to complete your configuration.

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
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 228.

Configure a user store
You can configure a user store using the **`prepare-external-store`** and **`create-initial-config`** commands.

PingDataGovernance Server uses a user store that lets you 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 System for Cross-domain Identity Management (SCIM) resource type through which PingDataGovernance Server obtains the user attributes.

The optional **`create-initial-config`** command is recommended for first-time installers. If you do not use **`create-initial-config`**, you can configure the following objects:

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

ⓘ **Note:**

If you do not configure these objects, you do not get the user's profile (the requester's attributes). For more information, see *Make a user's profile available in policies* on page 241.

For more information about configuring SCIM, see *About the SCIM service* on page 176.

Configure Access Token Validation
You can configure access token validators to translate an access token for policy processing.

Clients authenticate themselves to HTTP APIs and the System for Cross-domain Identity Management (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 251.

Next steps

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

Consider performing the following next steps.

- Sign on to the Administrative Console to configure endpoints for existing JSON APIs.

  For more information, see *About the API security gateway* on page 149.
- Sign on to the Administrative Console to define SCIM APIs for data in databases

  For more information, see *About the SCIM service* on page 176.
- Sign on to the PingDataGovernance Policy Administration GUI to create policies.

  For more information, see the PingDataGovernance Policy Administration Guide.

Server profiles and PingDataGovernance Server installation

Administrators can export the configuration of a PingDataGovernance Server instance to a directory of mostly text files, called a server profile.

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 also track changes to server profile text files in a version-control system, like Git, and can install new instances of PingDataGovernance Server, or update existing instances 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, such as 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 to 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 131.

- 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

You can use the **`manage-profile`** tool to substitute different variables in server profiles.

The **`manage-profile`** tool uses the format **`${VARIABLE}`** to support the substitution of variables in profiles. To escape this format, use another **`$`**. 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 in the file overwrite any environment variables.

The following lines provide an example of how you can set user-defined variables by using a variables file in the server profile.

```
HOSTNAME=testserver.example.com
PORT=389
```

The following table describes built-in variables that you can also reference in the server profile. Use these variables in the format previously described.

Built-in variable	Description
`PING_SERVER_ROOT`	Evaluates to the absolute path of the server's root directory
`PING_PROFILE_ROOT`	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

When you create a server profile, you can review the typical server profile hierarchy structure.

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.

You can add files 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
    |-- variables-ignore.txt
```

setup-arguments.txt

When you create a new profile, you must add arguments to the `setup-arguments.txt` file.

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 you can change.

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

dsconfig/

You can use **`dsconfig`** batch files to apply **`dsconfig`** commands to PingDataGovernance Server.

You can add **`dsconfig`** batch files 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/

You can add a variety of server root files to the `server-root` directory.

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.

Include any configuration that is necessary for the extensions in the profile's **`dsconfig`** batch files.

variables-ignore.txt

You can use the `variables-ignore.txt` file to indicate the relative paths of any files whose variables you do not want to have substituted.

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

You can use `server-root/permissions.properties` to specify permissions you want to apply to files copied to the server root.

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/

You can find additional miscellaneous documentation and other files in the `misc-files` directory.

The **`manage-profile`** tool does not use the `misc-files` directory. 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

You can use the **`manage-profile`** tool to complete a variety of workflows in PingDataGovernance.

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 133.
- *Installing a new environment* on page 134.
- *Scaling up your environment* on page 135.
- *Rolling out an update* on page 135.

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

You can create a server profile from a configured server in PingDataGovernance Server.

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, you can check the server profile in to a version-control system, like Git, share with your team, and integrate into your deployment automation.

Installing a new environment

You can use **`manage-profile setup`** to set up a new server instance and deployment environment in PingDataGovernance Server.

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

You can scale up the environment in your PingDataGovernance Server instance.

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

When you roll out a PingDataGovernance Server update, run **`manage-profile replace-profile`** to use a server profile that you have set up.

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. If the final server was running before the command was started, it is left running. If the final server was stopped, it remains stopped.

If files have been added or modified in the server root since you ran the most recent **manage-profile setup** or **manage-profile replace-profile** subcommand, 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 you ran **manage-profile replace-profile**.

Steps

1. Extract the distribution package for 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.

   You must run the **replace-profile** subcommand from the location of the distribution package, not from 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
   ```

Docker and PingDataGovernance Server installation

You can use Docker, which is available from the Docker Hub repository, to install the PingDataGovernance Server.

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/.

Docker and PingDataGovernance Policy Administration GUI installation

When running the Policy Administration GUI within a Docker container, you can take advantage of the automated policy database update feature by using a mounted volume.

For example, when running the Ping Identity DevOps pingdatagovernancepap Docker container, you could use the following command to ensure that the policy database is on the mounted volume in preparation for future versions of the image. The command:

- Runs a Docker container named pap81
- Creates a mounted volume that maps the host file system location /home/developer/ to the Docker container location /opt/shared
- Indicates to the **setup** tool using the PING_H2_FILE environment variable to use the Docker container file system location /opt/shared/Symphonic.mv.db for the policy database (The environment variable does not need the .mv.db extension.)

Note:

The Ping Identity DevOps Docker image documentation is frequently updated as new features are released. For the most recent instructions about running the Docker images, see *https://devops.pingidentity.com/*.

Also, the Docker image tag used in the example is only a placeholder. For the real tag, see Docker Hub (*https://hub.docker.com/r/pingidentity/pingdatagovernancepap*).

```
$ docker run --name pap81 -p 443:443 \
  -d --env-file ~/.pingidentity/devops \
  --volume /home/developer/PingDataGovernance.lic:/opt/in/instance/
pingdatagovernance.lic \
  --volume /home/developer/:/opt/shared \
  --env PING_H2_FILE=/opt/shared/Symphonic \
  pingidentity/pingdatagovernancepap:8.1.0.0-alpine-az11-ace3
```

Clustering and scaling

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, use DevOps principles of Infrastructure-as-Code (IaC) and Automation. For more information about using server profiles to scale upward by installing a new, identically configured instance of PingDataGovernance Server, see *Server profiles and PingDataGovernance Server installation* on page 130.

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. You enable intra-cluster communication by running **`setup`** using peer setup options such as `--peerHostName` and `--peerPort`.

Warning:

The clustering model is deprecated and will be removed in a future release. For more information, contact Ping Professional Services.

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 primary/secondary architecture. All writes and updates are forwarded to the primary server, which forwards them to all other servers.

- Changes to clustered configurations are not allowed in mixed-version clusters. This applies to configuration in the `cn=Cluster,cn=config` subtree and only applies to servers with matching cluster names. Consider this when updating multiple servers in a cluster.
- To make clustered configuration changes in a mixed-version cluster, choose one of the following options:
  - Update each server to the same version.
  - Temporarily split up the cluster by changing the `cluster-name` property on the server instance configuration objects.
- After you have updated all the servers to the same version, you can again make clustered configuration changes and those changes will mirror across the topology.

Upgrading PingDataGovernance Server

PingDataGovernance includes two server applications you must upgrade in tandem—the main PingDataGovernance Server and the Policy Administration GUI.

Ping Identity issues software release builds periodically with new features, enhancements, and fixes for improved server performance.

ⓘ **Note:**

PingDataGovernance Server used in external PDP mode requires a Policy Administration GUI with the same version. When upgrading PingDataGovernance Server, you must also upgrade the Policy Administration GUI.

Upgrade overview and considerations

When upgrading PingDataGovernance, you must consider factors such as the scope of the update, your installed version of Java, and the PingDataGovernance version from which you are upgrading.

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

Consider the following when upgrading:

- The update affects only the server being upgraded. The process does not alter the configuration of other servers, so you must update those servers separately.
- The update tool verifies that the installed version of Java 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.
- Upgrades for PingDataGovernance Server are only supported from versions 7.0.0.0 or later. If upgrading from a version of PingDataGovernance prior to 7.3.0.0, configuration loss will occur. The update tool has a warning message about this.

ⓘ **Tip:** For additional considerations, see *Planning your upgrade*.

Upgrading PingDataGovernance Server

Perform the following steps to upgrade a PingDataGovernance server.

Steps

1. Download and unzip the new version of 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 is extracted 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

After you've updated PingDataGovernance Server, you can revert to the previous version (one level back) using the `revert-update` tool.

About this task

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 running a revert, the server displays warnings about "offline configuration changes," but these are not critical and will not appear during subsequent start-ups.

Steps

- Run **`revert-update`** in the server root directory to revert back to the most recent previous version of the server, as shown in the following example.

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

Upgrading the PingDataGovernance Policy Administration GUI without Docker

If you originally installed the PingDataGovernance Policy Administration GUI without using Docker, use this procedure to upgrade the GUI when a new version is released.

Steps

1. In your current Policy Administration GUI, complete the steps in *Backing up policies* on page 142.
2. Stop the Policy Administration GUI.

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

3. Obtain and unzip the new version of the PingDataGovernance Policy Administration GUI in a location outside the existing GUI's installation.

4. Copy the existing database.

   The new server installation might require changes to the policy database structure. The server **`setup`** tool performs these upgrades and generates a new `configuration.xml` file.

   This example assumes the old installation is in `/opt/dg/PingDataGovernance-PAP-previous`, and the new installation is in `/opt/dg/PingDataGovernance-PAP`.

To upgrade a database from	Run this command
8.1	`$ cp /opt/dg/PingDataGovernance-PAP-previous/Symphonic.mv.db /opt/dg/PingDataGovernance-PAP`
8.0	`$ cp /opt/dg/PingDataGovernance-PAP-previous/admin-point-application/db/Symphonic.mv.db /opt/dg/PingDataGovernance-PAP`

5. Run **`setup`**.

   Note:

   Updating PingDataGovernance Server uses an **`update`** tool. PingDataGovernance GUI does not have this tool though. Instead of updating the GUI in-place, you install the new GUI.

   Warning:

   The **`setup`** tool uses the default credentials to upgrade the policy database. If the credentials no longer match the default values, the server administrator should pass the correct credentials to the **`setup`** tool using the `--dbAdminUsername`, `--dbAdminPassword`, `--dbAppUsername`, and `--dbAppPassword` command-line options. Otherwise, **`setup`** fails when it cannot access the policy database, or it might reset credentials to their default values. For more information, see *Manage policy database credentials* on page 221.

   Follow the instructions in one of the following topics:

   - *Installing the PingDataGovernance Policy Administration GUI interactively* on page 123
   - *Installing the PingDataGovernance Policy Administration GUI noninteractively* on page 124

6. Start the new GUI.

   Follow the instructions in *Post-setup steps* on page 126.

7. In the new GUI, complete the steps in *Upgrading the Trust Framework and policies* on page 142.

Upgrading the PingDataGovernance Policy Administration GUI with Docker

If you originally installed the GUI with Docker per *Docker and PingDataGovernance Policy Administration GUI installation* on page 136, use this procedure to upgrade the PingDataGovernance Policy Administration GUI when a new version is released.

Steps

1. In your current Policy Administration GUI, complete the steps in *Backing up policies* on page 142.

2. Stop the old Docker container and start the new one.

   When a new Docker image for the PingDataGovernance Policy Administration GUI is available, you stop the existing Docker container and start the new container from the new image while mounting the same volumes. The Ping Identity DevOps Docker images use the PingDataGovernance **`setup`** tool to update the policy database on the mounted volume, as described in *Example: Override the configured policy database location* on page 148.

   Warning:

   If you use a shared volume, you should always stop the Docker container running the older version of the Policy Administration GUI before you start the new container.

   For example, the following commands stop the running container and run a new image named `pap82`. This image uses the volumes from `pap81` to house the policy database. Also, the command uses the same PING_H2_FILE value from *Example: Override the configured policy database location* on page 148 to indicate that the PingDataGovernance **`setup`** tool should use that location.

```
$ docker container stop pap81
$ docker run --name pap82 -p 443:443 -d --env-file ~/.pingidentity/devops
 \
    --volumes-from pap81 \
    --env PING_H2_FILE=/opt/shared/Symphonic \
    pingidentity/pingdatagovernancepap:8.2.0.0-alpine-az11-ace3
```

   Warning:

   The **`setup`** tool uses the default credentials to upgrade the policy database. If the credentials no longer match the default values, the server administrator should pass the correct credentials to the **`setup`** tool using the PING_DB_ADMIN_USERNAME, PING_DB_ADMIN_PASSWORD, PING_DB_APP_USERNAME, and PING_DB_APP_PASSWORD UNIX environment variables.

   For example, if the old policy database admin credentials have been previously set to admin/Passw0rd, and the application credentials have been set to app/S3cret, the docker **`run`** command should include those environment variables as shown in this example:

```
  $ docker container stop pap81
  $ docker run --name pap82 -p 443:443 -d --env-file ~/.pingidentity/
devops \
  --env PING_H2_FILE=/opt/shared/Symphonic \
  --env PING_DB_ADMIN_USERNAME=admin \
  --env PING_DB_ADMIN_PASSWORD=Passw0rd \
  --env PING_DB_APP_USERNAME=app \
  --env PING_DB_APP_PASSWORD=S3cret \
  pingidentity/pingdatagovernancepap:8.2.0.0-alpine-az11-ace3
```

   This command ensures that the **`setup`** tool has the correct credentials to access the policy database, and that it does not reset credentials to their defaults.

3. In the new GUI, complete the steps in *Upgrading the Trust Framework and policies* on page 142.

Backing up policies

Policy writers might want to back up existing policies before upgrading the Policy Administration GUI. Do this by exporting policy snapshots.

Steps

1. Sign on to the Policy Administration GUI and choose any existing branch to go to the main landing page.
2. To display your current branches, select **Branch Manager# Version Control**.
3. From the **Branches** list, click a branch that you want to export.
   You should see a list of the commits for that branch, and the most recent version of the branch is named **Uncommitted Changes**.
4. Identify the commit that represents the snapshot that you want to export and click the three-line icon in the **Options** column.
5. Choose **Export Snapshot**.
   Your browser downloads the file.
6. Repeat for any additional branches that you want to back up.

Upgrading the Trust Framework and policies

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 must upgrade any policies based on `defaultPolicies.SNAPSHOT`.

Steps

1. Sign on to the Policy Administration GUI and choose any branch to go to the main landing page.
2. Select **Branch Manager** from the navigation bar on the left, and open the **Merge Snapshot** tab.
3. Click the **file selection** option, and go to the `resource/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.

   ⓘ **Important:** If you are upgrading from 7.3.0.*x*, use the `7.3.0.`*`x`*`-to-8.0.0.0-SNAPSHOT` and merge that (per the next step) before you select and merge `8.0.0.0-to-8.1.0.0.SNAPSHOT`.

   When upgrading from version 8.0.0.0 to version 8.1.0.0, use `resource/policies/upgrade-snapshots/8.0.0.0-to-8.1.0.0.SNAPSHOT`.
5. Merge the partial snapshot.

   ⓘ **Note:**

   Merge conflicts might occur where objects have been updated. If you have not modified the objects in conflict, you can safely select **Keep Snapshots**.
6. Return to your PingDataGovernance Server installation.
7. Run the following **`dsconfig`** command to configure PingDataGovernance Server to use the latest Trust Framework version.

```
dsconfig set-policy-decision-service-prop \
   --set trust-framework-version:v2
```

Uninstalling PingDataGovernance Server

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 you want to remove.

   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.

PingDataGovernance Server Administration Guide

Running PingDataGovernance Server

Run PingDataGovernance Server from your Unix/Linux or Windows terminal.

Steps

- To start PingDataGovernance Server as a background process on a UNIX or Linux system, run **`bin/start-server`**.
- To start PingDataGovernance Server in the foreground, add the --nodetach option.

  `bin/start-server -nodetach`
- On Microsoft Windows systems, run **`bat/start-server.bat`**.

Starting PingDataGovernance Server in Unix/Linux

To start PingDataGovernance Server in a Unix/Linux environment, use the **`bin/start-server`** command.

Steps

1. In a terminal window, enter `cd ~/<pdgs_install>/PingDataGovernance` to select the directory where you have installed PingDataGovernance Server.
2. Run the `$ bin/start-server` command.

Running PingDataGovernance Server as a foreground process

Run or stop PingDataGovernance Server as a foreground process in Unix/Linux environments.

Steps

- To launch PingDataGovernance Server as a foreground process, run **`$ bin/start-server --nodetach`**.
- To stop a running PingDataGovernance Server, do one of the following:
  - In the terminal window running the server, press and hold `CTRL+C`.
  - In a new terminal window, run **`bin/stop-server`**.

Starting PingDataGovernance Server at boot time (Unix/Linux)

Create a script to run PingDataGovernance Server when the system boots.

About this task

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

Steps

- 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.
    1. Create the service unit configuration file in a temporary location, as in the following example.

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

```
   --userName dg
```

In this example, `dg` represents the username assigned to PingDataGovernance Server.

2. Switch to root user. The command for doing this will vary depending on your distribution.
3. As a root user, copy the `ping-data-governance.service` configuration file to the `/etc/systemd/` system directory as shown.

   ```
   cp ping-data-governance.service /etc/systemd/
   ```

4. Reload `systemd` to read the new configuration file as shown.

   ```
   $ systemctl daemon-reload
   ```

5. To start PingDataGovernance Server, use the **`start`** command.

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

6. To configure PingDataGovernance Server to start automatically when the system boots, use the **`enable`** command, as in the following example.

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

7. Sign off from the system as the root user.

- Create a Run Control (RC) script manually.

  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)

On Windows Server systems you can register PingDataGovernance Server as a service to start it up when booting.

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

Registering PingDataGovernance Server as a service allows you to automate startup when booting.

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.

   > ⓘ **Note:** You cannot register a server 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

You can run multiple instances of PingDataGovernance Server as Windows services by altering the `wrapper-product.conf` file.

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. You can also add or change service descriptions in the `wrapper-product.conf` file.

Steps

1. Open the *`<server-root>`*`/config/wrapper-product.conf` file.
2. Change the wrapper.name property to a unique string, such as `pingdatagovernance1`.
3. **Save** the `wrapper-product.conf` file.
4. Register PingDataGovernance Server as a service. For more information, see *Registering PingDataGovernance Server as a Windows service* on page 145.
5. Repeat these steps for each service instance you want to create.

Deregistering and uninstalling services

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

About this task

Steps

1. To remove the service from the Windows registry, run the `bat/deregister-windows-service.bat` script.
2. To uninstall PingDataGovernance Server, run the `PingDataGovernance/uninstall.bat` script. For more information, see *Uninstalling PingDataGovernance Server* on page 143.

Log files for services

You can configure the log files generated by PingDataGovernance Server running as a Windows service.

Log files are stored in `<server-root>/logs`, and file names begin with `windows-service-wrapper`.

You can edit the log file configurations in the `<server-root>/config/wrapper.conf` file.

Log files are configured to rotate each time the wrapper starts due to file size. You can edit the allowed file size using the wrapper.logfile.maxsize parameter. The default size is 50 Mb.

By default, only the two most recent log files are retained. You can change how many log files to retain by editing the wrapper.logfile.maxfiles parameter.

Starting PingDataGovernance Policy Administration GUI

Use the **start-server** command to start the Policy Administration GUI. Also, you can use environment variables to override configuration variables at startup.

To start PingDataGovernance Policy Administration GUI, use the **bin/start-server** command.

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

Note:

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

Overriding the configuration at startup

You can override a number of Policy Administration GUI settings by defining specific environment variables before starting the server. By overriding some of the configuration, you can redefine certain aspects of the configuration without re-running the **setup** tool.

To override the configuration, stop the Policy Administration GUI, define one or more of the environment variables, and restart the Policy Administration GUI.

Environment variables you can use to override configuration variables

The following table lists the environment variables that you can define.

Environment variable	Example value	Description
PING_EXTERNAL_BASE_URL	pap.example.com:9443	The Policy Administration GUI hostname and port. PingDataGovernance uses this value to construct AJAX requests. The port value must match the value of PING_PORT for web browsers to pass CORS checks.
PING_PORT	443	The Policy Administration GUI HTTPS port. The server binds to this listen port.
PING_KEYSTORE_TYPE	JKS	The Policy Administration GUI's key store type. Valid values include `JKS` and `PKCS12`.
PING_KEYSTORE_PATH	/path/to/keystore.jks	The path to the Policy Administration GUI's key store.
PING_KEYSTORE_PASSWORD	password1234	The Policy Administration GUI's key store password.
PING_CERT_ALIAS	server-cert	The alias for the Policy Administration GUI's server certificate.
PING_SHARED_SECRET	DataGovernance	The Policy Administration GUI's shared secret, which PingDataGovernance Server needs to make policy requests to the Policy Administration GUI.

Environment variable	Example value	Description
PING_OIDC_ CONFIGURATION_ENDPOINT	https://oidc.example.com:9031/.well-known/openid-configuration	The OpenID Connect (OIDC) provider's discovery URL. Used when the Policy Administration GUI is set up in OIDC mode.
PING_CLIENT_ID	8cb9f2c9-c366-47e0-9560-db2132b2d813	The Policy Administration GUI's client ID with the OpenID Connect provider. Used when the Policy Administration GUI is set up in OIDC mode.
PING_USERNAMES	admin, user1, user2	Used in demo mode. A comma-separated list of usernames accepted by the Policy Administration GUI for sign on.
PING_H2_FILE	./Symphonic	The path to the policy database H2 file. Leave off the `.mv.db` extension.
PING_DB_APP_USERNAME	db_user	The username the application uses to access the server database.
PING_DB_APP_PASSWORD	Pa$$w0rd!23	The password the application uses to access the server database.

Example: Override the configured HTTPS port

In this example, the Policy Administration GUI is started using an HTTPS port that differs from the value configured during installation. The override requires two environment variables: PING_PORT and PING_EXTERNAL_BASE_URL.

```
$ bin/stop-server
$ export PING_PORT=9443 PING_EXTERNAL_BASE_URL=pap.example.com:9443; bin/
start-server
```

Example: Override the configured policy database location

This example changes the policy database location. The new value must be a policy server Java Database Connectivity (JDBC) connection string for an H2 embedded database. To use a file located at `/opt/shared/Symphonic.mv.db`, use the following commands.

```
$ bin/stop-server
$ export PING_H2_FILE=/opt/shared/Symphonic
$ bin/setup demo {ADDITIONAL_ARGUMENTS} && bin/start-server
```

ⓘ **Note:**

Even though the actual filename of the policy database includes the extension `.mv.db`, the JDBC connection string excludes the extension.

If `/opt/shared/Symphonic.mv.db` does not exist, **`setup`** creates a new one. If the file does exist and is from an older PingDataGovernance server, **`setup`** updates the file to the latest version.

Troubleshooting startup errors

The **`bin/start-server`** command prints an error message if it detects that an error has occurred during startup. For more information about the error, see the `logs/datagovernance-pap.log` file.

Stopping PingDataGovernance Server

PingDataGovernance Server provides a simple shutdown script to stop the server.

Steps

- To stop the PingDataGovernance Server, run the **`$ bin/stop-server`** command.

  Note:

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

Stopping PingDataGovernance Policy Administration GUI

PingDataGovernance Policy Administration GUI provides a simple shutdown script to stop the system.

Steps

- To stop the PingDataGovernance Policy Administration GUI, run the **`bin/stop-server`** command.

  Note:

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

Restarting PingDataGovernance Server

You can stop and restart PingDataGovernance Server with a single command.

About this task

Running this command is equivalent to shutting down PingDataGovernance Server, exiting the Java virtual machine (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
```

About the API security gateway

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

See the following topics for specific details about the functionality of the API security gateway.

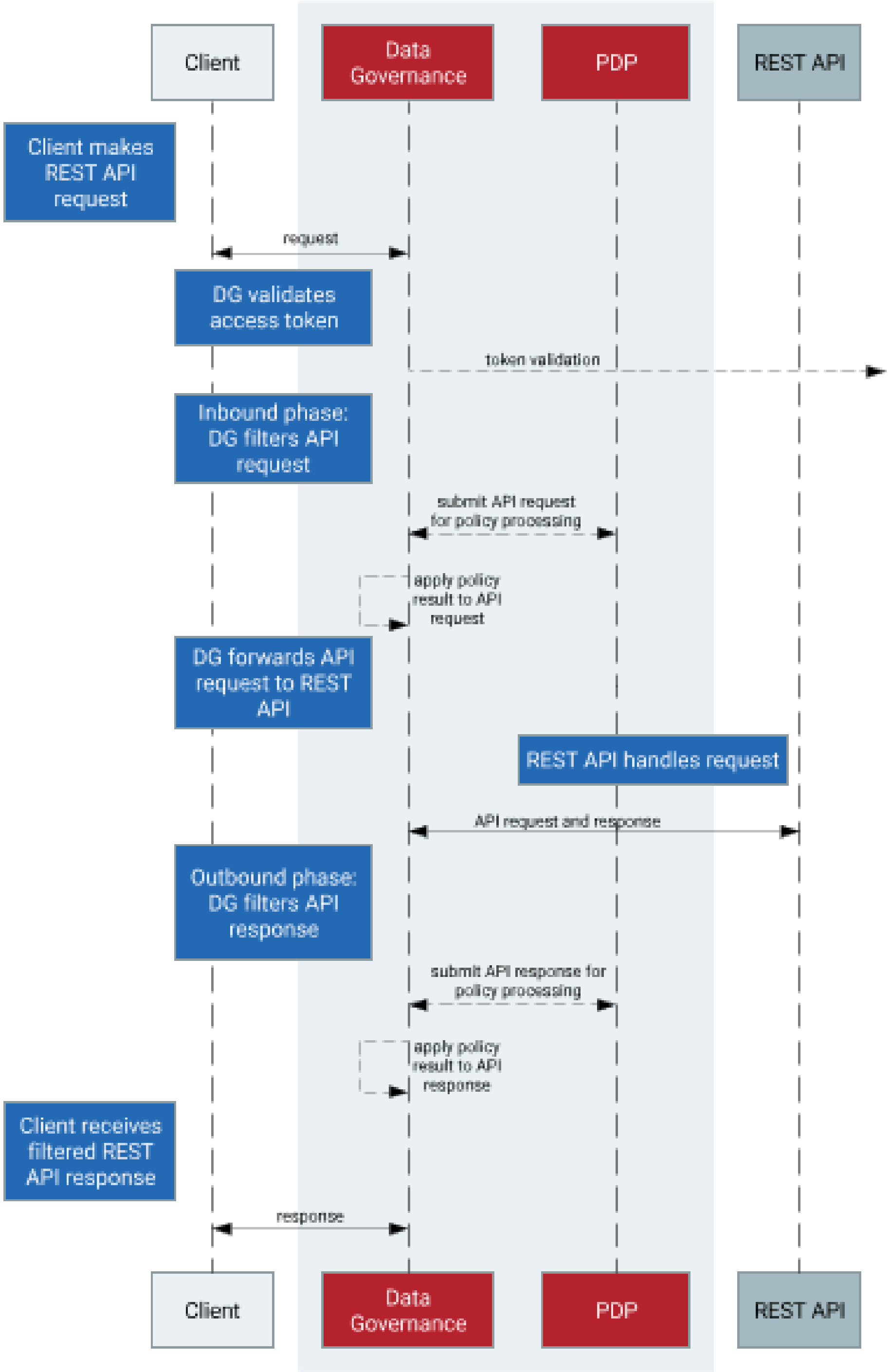
- *Request and response flow* on page 150
- *Gateway configuration basics* on page 151
- *API security gateway authentication* on page 152
- *API security gateway policy requests* on page 153
- *API security gateway HTTP 1.1 support* on page 159
- *About error templates* on page 160

Request and response flow

The API gateway processes JSON requests and responses in two distinct phases according to a defined sequence.

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 policy decision point (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.


Client
Data Governance
PDP
REST API
Client makes REST API request
request
DG validates access token
token validation
Inbound phase: DG filters API request
submit API request for policy processing
apply policy result to API request
DG forwards API request to REST API
REST API handles request
API request and response
Outbound phase: DG filters API response
submit API response for policy processing
apply policy result to API response
Client receives filtered REST API response
response
Client
Data Governance
PDP
REST API

The API gateway supports only JSON requests and responses.

Gateway configuration basics

You can configure the API security gateway by creating and modifying its components.

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. You can create the server in the PingDataGovernance Administrative 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, see the Configuration Reference Guide that is bundled with the product.

API security gateway authentication

The API security gateway authenticates requests through bearer tokens by default, and you can configure it to handle authentication according to your preferences.

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 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 that all requests are made with an active access token.

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

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

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

Property	Description
`http-auth-evaluation-behavior`	Determines whether the Gateway API Endpoint evaluates bearer tokens, and if so, whether the bearer token is forwarded to the API server.
`access-token-validator`	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. If `http-auth-evaluation-behavior` is set to `do-not-evaluate`, this setting is ignored.

API security gateway policy requests

The API security gateway creates policy requests for incoming requests and API responses, and you can observe how it creates them.

Before accepting an incoming request and forwarding it to the API server, the gateway creates a policy request based on the incoming request and sends it to the policy decision point (PDP) for authorization. Before accepting an API server response and forwarding it back to the client, the gateway creates a policy request based on the incoming 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.

You can selectively disable response policy processing on a per-API-Endpoint basis. This ability is useful if the Gateway authorizes requests but does not filter responses. Disabling this processing can improve performance for frequent requests or requests that return very large responses. To disable processing, set the Gateway API Endpoint's `disable-response-processing` property to `true`.

To better understand how the gateway formulates policy requests, enable 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 289.

Policy request attributes

There are many policy request attributes generated by the security gateway, including attributes nested within the `attributes`, `HttpRequest.AccessToken`, `HttpRequest.ClientCertificate`, and `Gateway` fields.

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

Policy request attributes	Description	Type
`action`	Identifies the gateway request processing phase and the HTTP method, such as GET or POST. The value is formatted as `<phase>-<method>`. Example values include `inbound-GET`, `inbound-POST`, `outbound-GET`, and `outbound-POST`.	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
`domain`	Unused.	String
`identityProvider`	Identifies the access token validator that evaluates the bearer token used in an incoming request.	String
`service`	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 service property. Multiple Gateway API Endpoints can use the same service value.	String

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

Attribute	Description	Type
`Gateway`	Provides additional gateway-specific information about the request not provided by the following attributes.	Object
`HttpRequest.AccessToken`	Parsed access token. For more information, see the following table.	Object
`HttpRequest.ClientCertificate`	Properties of the client certificate, if one was used.	Object
`HttpRequest.CorrelationId`	A unique value that identifies the request and response, if available.	String
`HttpRequest.IPAddress`	The client IP address.	String
`HttpRequest.QueryParameters`	Request URI query parameters.	Object
`HttpRequest.RequestBody`	The request body, if available.	Object
`HttpRequest.RequestHeaders`	The HTTP request headers.	Object

Attribute	Description	Type
`HttpRequest.RequestURI`	The request URI.	String
`HttpRequest.ResourcePath`	Portion of the request URI path following the inbound base path that the Gateway API Endpoint defines.	String
`HttpRequest.ResponseBody`	The response body, if available. This attribute is provided only for outbound policy requests.	Object
`HttpRequest.ResponseHeaders`	The HTTP response headers, if available.	Object
`HttpRequest.ResponseStatus`	The HTTP response status code, if available.	Number
`TokenOwner`	The access token subject as a SCIM resource, as obtained by the access token validator.	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
`access_token`	The actual access token from the client request.	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
`client_id`	The client ID of the application that was granted the access token.	String
`expiration`	Date and time at which the access token expires.	DateTime
`issued_at`	Date and time at which the access token was issued.	DateTime
`issuer`	Token issuer. This attribute is usually a URI that identifies the authorization server.	String
`not_before`	Date and time before which a resource server does not accept the access token.	DateTime
`subject`	Token subject. This attribute is a user identifier that the authorization server sets.	String

Attribute	Description	Type
`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 `<resource type>/<resource ID>`.	String
`token_type`	The token type, as set by the authorization server. This value is typically set to `bearer`.	String
`user_token`	Flag that the access token validator sets to indicate that the token was issued originally to a subject. If this flag is `false`, the token does not have a subject and was issued directly to a client.	Boolean
`username`	Subject's user name. This attribute is a user identifier that the authorization server sets.	String

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

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

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

Attribute	Description	Type
`_BasePath`	Portion of the HTTP request URI that matches the Gateway API Endpoint's `inbound-base-path` value.	String
`_TrailingPath`	Portion of the HTTP request URI that follows the `_BasePath`.	String

Attribute	Description	Type
base path parameters	Parameters used in a Gateway API Endpoint's `inbound-base-path` configuration property are included as fields of the `Gateway` attribute.	String
custom attribute	The `Gateway` attribute might contain multiple arbitrary custom attributes that are defined by the `policy-request-attribute` 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
`inbound-base-path`	Defines the URI path prefix that the gateway uses to determine whether the Gateway API Endpoint handles a request. The `inbound-base-path` property value can include parameters. If parameters are found and matched, they are included as attributes to policy requests. The following configuration properties reference parameters that the `inbound-base-path` introduces: ▪ `outbound-base-path` ▪ `service` ▪ `resource-path` ▪ `policy-request-attribute`
`service`	Identifies the API service to the PDP. The service value appears in the policy request as the `service` attribute. If undefined, the service value defaults to the name of the Gateway API Endpoint.
`resource-path`	Identifies the REST resource to the PDP. The resource path value appears in the policy request as the `HttpRequest.ResourcePath` attribute. If undefined, the resource path value defaults to the portion of the request that follows the base path defined by `inbound-base-path`.

Gateway API Endpoint property	Description
`policy-request-attribute`	Defines zero or more static, arbitrary key-value pairs. If specified, key-value pairs are always added as attributes to policy requests. These custom attributes appear in the policy request as fields of the `Gateway` attribute. For example, if a value of `policy-request-attribute` is `foo=bar`, the attribute `Gateway.foo` is added to the policy request with a value of `bar`.

Path parameters

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.

Gateway API Endpoint configuration properties that affect policy requests on page 157 identifies additional configuration properties that can use these parameters.

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

Basic example

The following example configuration demonstrates how request URIs are mapped to the outbound path to alter policy requests.

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

Request URIs are mapped to the outbound path to alter policy requests.

Consider 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`

API security gateway HTTP 1.1 support

In its capacity as a reverse proxy, the API security gateway must modify HTTP requests and responses in addition to the changes required by policy processing.

Forwarded HTTP request headers

HTTP requests often pass through a chain of intermediaries before reaching a destination server. The HTTP 1.1 specifications define two categories of headers that are pertinent to this context.

End-to-end headers

Headers requiring transmission to all recipients on the chain, such as `Content-Type`.

Hop-by-hop headers

Headers that are only relevant to the next recipient on the chain, such as `Connection` and `Keep-Alive`.

The API security gateway never forwards hop-by-hop headers. It generally forwards all end-to-end headers, with the following exceptions:

- Headers related to HTTP resource versioning and conditional requests, such as `If-None-Match` and `If-Modified-Since`, are never forwarded.
- Headers related to CORS, such as `Origin` or `Access-Control-Request-Method`, are never forwarded.
- Headers that you exclude by using the `allowed-headers` configuration property of an API External Server to define a whitelist of forwarded headers.
- Headers that you remove by using a custom advice extension.

The API security gateway always adds the `Host`, `Accept-Encoding`, `Via`, `X-Forwarded-For`, `X-Forwarded-Host`, `X-Forwarded-Port`, and `X-Forwarded-Proto` headers to forwarded requests. If the HTTP Connection Handler is configured to use or generate correlation IDs, then a correlation ID header is also added to the forwarded request.

You can use the `http-auth-evaluation-behavior` property of a Gateway API Endpoint to alter the `Authorization` header of a forwarded request.

Forwarded HTTP response headers

The API security gateway forwards most HTTP response headers, with the following exceptions:

- The `Date` header is replaced with a value generated by the API security gateway.
- The `Content-Length` header is replaced with a value generated by the API security gateway.
- The `Location` header is replaced with a value generated by the API security gateway.
- If the HTTP Connection Handler is configured to use or generate correlation IDs, then a correlation ID header is added to the response.
- Headers related to HTTP resource versioning and conditional requests, such as `ETag` and `Last-Modified`, are never forwarded.
- Headers related to CORS, such as `Access-Control-Allow-Origin` or `Access-Control-Allow-Headers`, are never forwarded.

Unsupported HTTP request header

The API security gateway does not support the `Upgrade` header.

Unsupported advice changes

The API security gateway does not support using advice to add, modify, or delete the following headers:

- Hop-by-hop headers that the gateway always removes, such as `Connection` and `Keep-Alive`
- Conditional request headers that the gateway always removes, such as `If-None-Match` and `ETag`
- Proxy-specific headers that the gateway always adds, such as `Via` and `X-Forwarded-For`

The gateway overrides any changes to these headers.

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 in the following example.

```
{
  "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 175.

Configuring error templates example

The example in this section demonstrates the configuration of a custom error template for a Gateway API Endpoint named `Test API`.

About this task

Error responses that use this error template feature the following fields:

- `code`
- `message`

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"
   ```

Note:

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

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

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.

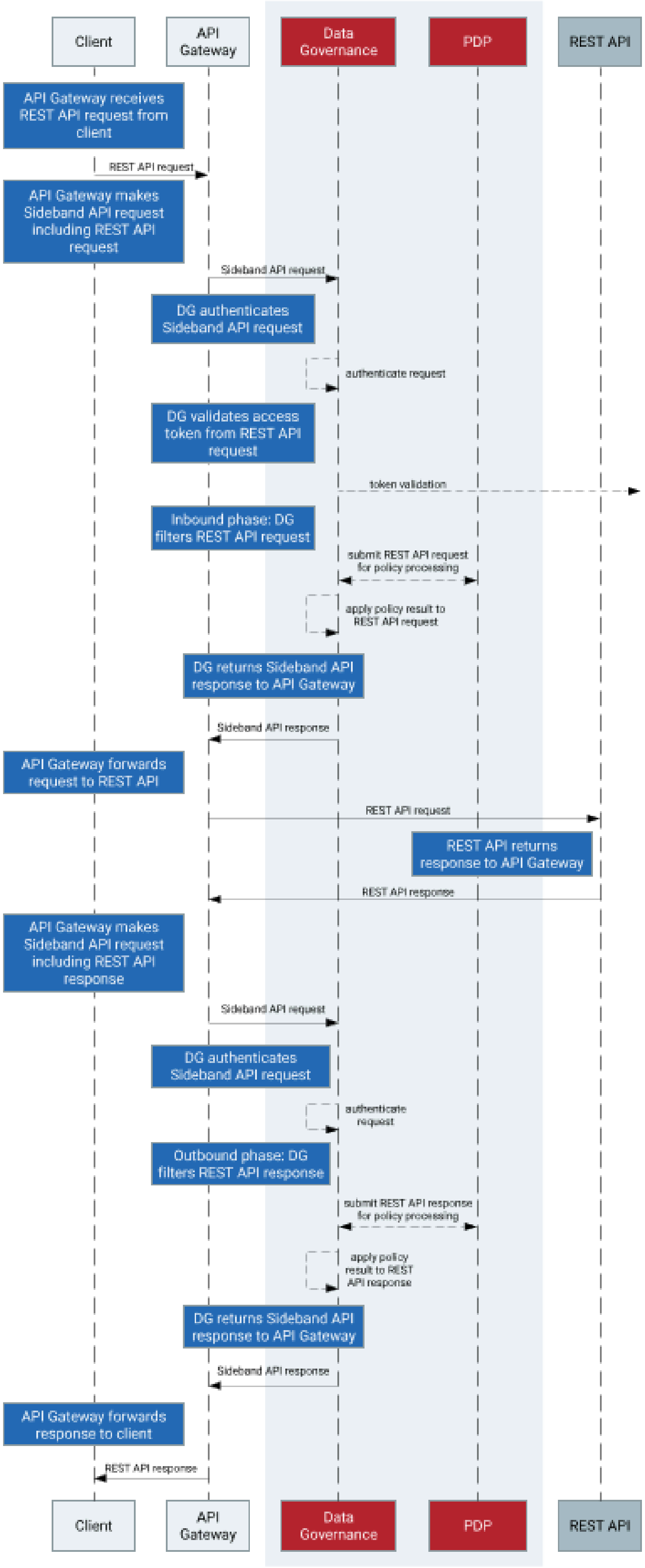
As a gateway, PingDataGovernance Server functions as a reverse proxy that performs the following steps:

- Intercepts client traffic to a backend REST API service
- Authorizes the traffic to a policy decision point (PDP) that operates either within the PingDataGovernance process, called Embedded PDP mode, or outside the PingDataGovernance process, called External PDP mode

Using the Sideband API, you can configure the PingDataGovernance Server instead as a plugin to an external API gateway. In Sideband mode, an API gateway integration point intercepts client traffic to a backend REST API service and passes intercepted traffic to the PingDataGovernance Sideband API.

API gateway integration

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


Client
API Gateway
Data Governance
PDP
REST API
API Gateway receives REST API request from client
REST API request
API Gateway makes Sideband API request including REST API request
Sideband API request
DG authenticates Sideband API request
authenticate request
DG validates access token from REST API request
token validation
Inbound phase: DG filters REST API request
submit REST API request for policy processing
apply policy result to REST API request
DG returns Sideband API response to API Gateway
Sideband API response
API Gateway forwards request to REST API
REST API request
REST API returns response to API Gateway
REST API response
API Gateway makes Sideband API request including REST API response
Sideband API request
DG authenticates Sideband API request
authenticate request
Outbound phase: DG filters REST API response
submit REST API response for policy processing
apply policy result to REST API response
DG returns Sideband API response to API Gateway
Sideband API response
API Gateway forwards response to client
REST API response
Client
API Gateway
Data Governance
PDP
REST API

Processing steps

1. After the API gateway receives a request from an API gateway plugin, it makes a call to the Sideband API to process the request.
2. 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.
3. If the Sideband API returns a response that indicates the request is unauthorized or not to be forwarded, the response includes the response to be returned to the client. The API gateway returns the response to the client without forwarding the request to the REST API.
4. When the API gateway receives a response from the REST API, it makes a call to the Sideband API to process the response.
5. 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

Defines 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 166.

Sideband API HTTP Servlet Extension

Represents the Sideband API itself. If you require shared secrets, you might need to configure this component. For more information, see *Authenticating to the Sideband API* on page 166.

Sideband API Endpoints

Represents a public path prefix that the Sideband API accepts for handling proxied requests. 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 default 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. For more information about using the Sideband API Endpoint configuration to customize policy requests, see *Sideband API policy requests* on page 168.

Note:

Changes to these components do not typically require a server restart to take effect. For more information, see the *Configuration Reference Guide*, which is bundled with the product.

Example

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
```

Authenticating to the Sideband API

The Sideband API can require an API gateway plugin to authenticate to it by using a shared secret.

To define shared secrets, use Sideband API Shared Secret configuration objects. To manage shared secrets, use the Sideband API HTTP Servlet Extension.

Creating a shared secret

Define the authentication credentials that the Sideband API might require an API gateway plugin to present.

Steps

1. 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 you set this value, 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.

2. 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"
```

A new Sideband API Shared Secret is not used until the `shared-secrets` property of the Sideband API HTTP Servlet Extension is updated.

Deleting a shared secret

You can remove a shared secret from use or delete it entirely.

Steps

- 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

To avoid service interruptions, the Sideband API allows multiple, distinct shared secrets to be accepted at the same time.

About this task

You can configure a new shared secret that the Sideband API accepts alongside an existing shared secret. This allows time to update the API gateway plugin 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. For more information, see *Creating a shared secret* on page 166.
2. Update the API gateway plugin to use the new shared secret.
3. Remove the previous Sideband API Shared Secret. For more information, see *Deleting a shared secret* on page 166.

Customizing the shared secret header

By default, the Sideband API accepts a shared secret from an API gateway plugin through the PDG-TOKEN header.

Steps

- To customize a shared secret header, change the value of the Sideband API HTTP Servlet Extension's `shared-secret-header` property.

  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

As with the PingDataGovernance 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.

About this task

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, the following 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
`http-auth-evaluation-behavior`	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.
`access-token-validator`	Sets the Access Token Validators that the Sideband API Endpoint uses. As 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. To constrain the set of Access Token Validators that a Sideband API Endpoint uses, set this property to use one or more specific values. This setting is ignored if `http-auth-evaluation-behavior` is set to `do-not-evaluate`.

Sideband API policy requests

Understanding how the Sideband API formulates policy requests can help you create and troubleshoot policies more effectively.

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 Policy Decision Point (PDP) for evaluation

Policy request attributes

The following tables provide an overview of policy request attributes.

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

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

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

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

Attribute	Description	Type
Gateway	Additional gateway-specific information about the request not provided by the following attributes.	Object
HttpRequest.AccessToken	Parsed access token. For more information, see the following table.	Object
HttpRequest.ClientCertificate	Properties of the client certificate, if one was used.	Object
HttpRequest.CorrelationId	A unique value that identifies the request and response, if available.	String
HttpRequest.IPAddress	The client IP address.	String
HttpRequest.QueryParameters	Request URI query parameters.	Object
HttpRequest.RequestBody	The request body, if available.	Object
HttpRequest.RequestHeaders	The HTTP request headers.	Object
HttpRequest.RequestURI	The request URI.	String
HttpRequest.ResourcePath	Portion of the request URI path that follows the inbound base path that the Sideband API Endpoint defines.	String

Attribute	Description	Type
`HttpRequest.ResponseBody`	The response body, if available. This attribute is provided only for outbound policy requests.	Object
`HttpRequest.ResponseHeaders`	The HTTP response headers, if available.	Object
`HttpRequest.ResponseStatus`	The HTTP response status code, if available.	Number
`TokenOwner`	The access token subject as a SCIM resource, as obtained by the access token validator.	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
`access_token`	The actual access token from the client request.	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
`client_id`	Client ID of the application that was granted the access token.	String
`expiration`	Date and time at which the access token expired.	DateTime
`issued_at`	Date and time at which the access token was issued.	DateTime
`issuer`	Token issuer. Typically, this value is a URI that identifies the authorization server.	String
`not_before`	Date and time before which a resource server does not accept an access token.	DateTime

Attribute	Description	Type
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 <resource type>/<resource ID>.	String
token_type	Token type, as set by the authorization server. Typically, this value is bearer.	String
user_token	Flag that the access token validator sets to indicate the token was originally issued to a subject. If the flag is false, the token contains no subject and was issued directly to a client.	Boolean
username	Subject's user name. This value represents a user identifier that the authorization server sets.	String

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

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

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

Attribute	Description	Type
_BasePath	Portion of the HTTP request URI that matches the Sideband API Endpoint's base-path value.	String

Attribute	**Description**	**Type**
`_TrailingPath`	Portion of the HTTP request URI that follows the `_BasePath`.	String
`base path parameters`	Parameters in a Sideband API Endpoint's `base-path` configuration property are included as fields of the `Gateway` attribute.	String
`base path parameters`	The `Gateway` attribute can contain multiple, arbitrary custom attributes that are defined by the `policy-request-attribute` 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**
`base-path`	Defines the URI path prefix that the Sideband API uses to determine whether the Sideband API Endpoint handles a request. The `base-path` property value can include parameters. If parameters are found and matched, they are included as attributes to policy requests. The following configuration properties can reference parameters that `base-path` introduces: ▪ `service` ▪ `resource-path` ▪ `policy-request-attribute`
`service`	Identifies the API service to the PDP. A policy can use this value to target requests. The `service` value appears in the policy request as the `service` attribute. If undefined, the `service` value defaults to the name of the Sideband API Endpoint.
`resource-path`	Identifies the REST resource to the PDP. The resource path value appears in the policy request as the `HttpRequest.ResourcePath` attribute. If undefined, the `resource-path` value defaults to the portion of the request that follows the base path, as defined by `base-path`.

Property	Description
policy-request-attribute	Defines zero or more static, arbitrary key-value pairs. If specified, the pairs are always added as attributes to policy requests. These custom attributes appear in the policy request as fields of the `Gateway` attribute. For example, if a value of `policy-request-attribute` is `foo=bar`, the attribute `Gateway.foo` is added to the policy request with the value `bar`.

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.

Other configuration properties can use these parameters. For more information, see *Sideband API Endpoint configuration properties* on page 172.

The `base-path` property must introduce parameters. Other configuration properties cannot introduce new parameters.

Path parameters: Basic example

The following table demonstrates a basic configuration of path parameters.

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 demonstrates an advanced configuration of path parameters.

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`

Request context configuration

The API gateway plugin provides data and metadata to the Sideband API about HTTP requests received from a client and HTTP responses received from an API server.

When the Sideband API handles an API server's HTTP response, you can enable the API gateway plugin to also provide data and metadata for the original HTTP request, which can be used to make policy decisions. For example, data about access token claims and the token owner are request data, but they might be useful when authorizing an HTTP response.

The Sideband API provides two methods to supply HTTP request data during HTTP response processing. Select a method according to the API gateway plugin's capabilities. By default, both methods are disabled. You can enable them by configuring the request-context-method property of the Sideband API HTTP Servlet Extension.

Request context using the state field

When enabled, the Sideband API adds a `state` field to its responses for inbound HTTP requests. This field contains an encoded form of the request data, including preprocessed authentication data, such as access token claims and token owner attributes. The API gateway plugin is expected to provide this state data when it next makes a request corresponding to the outbound HTTP response. The Sideband API can then pass this data about the HTTP request in its policy request.

As the state data includes preprocessed authentication information, this information can be made available for policy processing without the overhead of re-invoking an access token validator. However, the size of the state data is proportional to the size of the original HTTP request.

To enable this option, use the following command.

```
PingDataGovernance/bin/dsconfig \
  set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --set request-context-method:state
```

Request context using the request field

When enabled, an API gateway plugin making a request to handle an outbound HTTP response provides all data about the original HTTP request in the `request` field. If this data includes an `Authorization` header with a bearer token, the Sideband API invokes its access token validators to produce a set of access token claims and token owner attributes, which are then made available in the policy request.

To enable this option, use the following command.

```
PingDataGovernance/bin/dsconfig \
  set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --set request-context-method:request
```

Disabling request context handling

The request context feature is disabled by default. If you have enabled it, you can disable it with the following command.

```
PingDataGovernance/bin/dsconfig \
  set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --reset request-context-method
```

Access token validation

HTTP requests often include an access token with an `Authorization` header using the bearer token scheme, as described by *RFC 6750*.

By default, if a Sideband API request contains an `Authorization` header, the Sideband API processes the access token as follows:

- An access token validator parses and validates the access token, and the Sideband API adds the access token parsed claims to the policy request's `HttpRequest.AccessToken` field.
- If the access token has a subject, a token resource lookup method retrieves the subject's attributes, and the Sideband API adds them to the policy request's `TokenOwner` field.

In some cases, the parsing and validation performed by the access token validator might duplicate processing already performed by the API gateway itself. To eliminate redundant processing, you can configure a Sideband API endpoint to use an external API gateway access token validator, which is a unique access token validator that performs no parsing or validation of its own. The API gateway plugin might then pass the parsed access token claims directly to the Sideband API, which would ignore the `Authorization` header and accept the parsed access token claims as-is.

Example configuration

The following example shows how to configure an external API gateway access token validator with a token resource lookup method, and then assign it to an existing Sideband API endpoint.

```
dsconfig create-access-token-validator \
  --validator-name "API Gateway Access Token Validator" \
  --type external-api-gateway \
  --set enabled:true \
  --set evaluation-order-index:0
dsconfig create-token-resource-lookup-method \
  --validator-name "API Gateway Access Token Validator" \
  --method-name "Users by uid" \
  --type scim \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:0
dsconfig set-sideband-api-endpoint-prop \
  --endpoint-name "My API" \
  --set "access-token-validator:API Gateway-Provided Access Token Validator"
```

Error templates

REST API clients often 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, it forwards errors that the API returns to the client as they are, unless a policy dictates modifications to the response. In the following scenarios, PingDataGovernance Server returns an error that the Sideband API generates:

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

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

```
{
  "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 to respond with custom errors. Error templates, which are written in *Velocity Template Language*, 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

Example: Configure error templates
This example 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 the 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 System for Cross-domain Identity Management (SCIM) service provides a REST API for data that is stored in one or more external datastores, based on the *SCIM 2.0 standard*.

For information about the SCIM service, see the following topics:

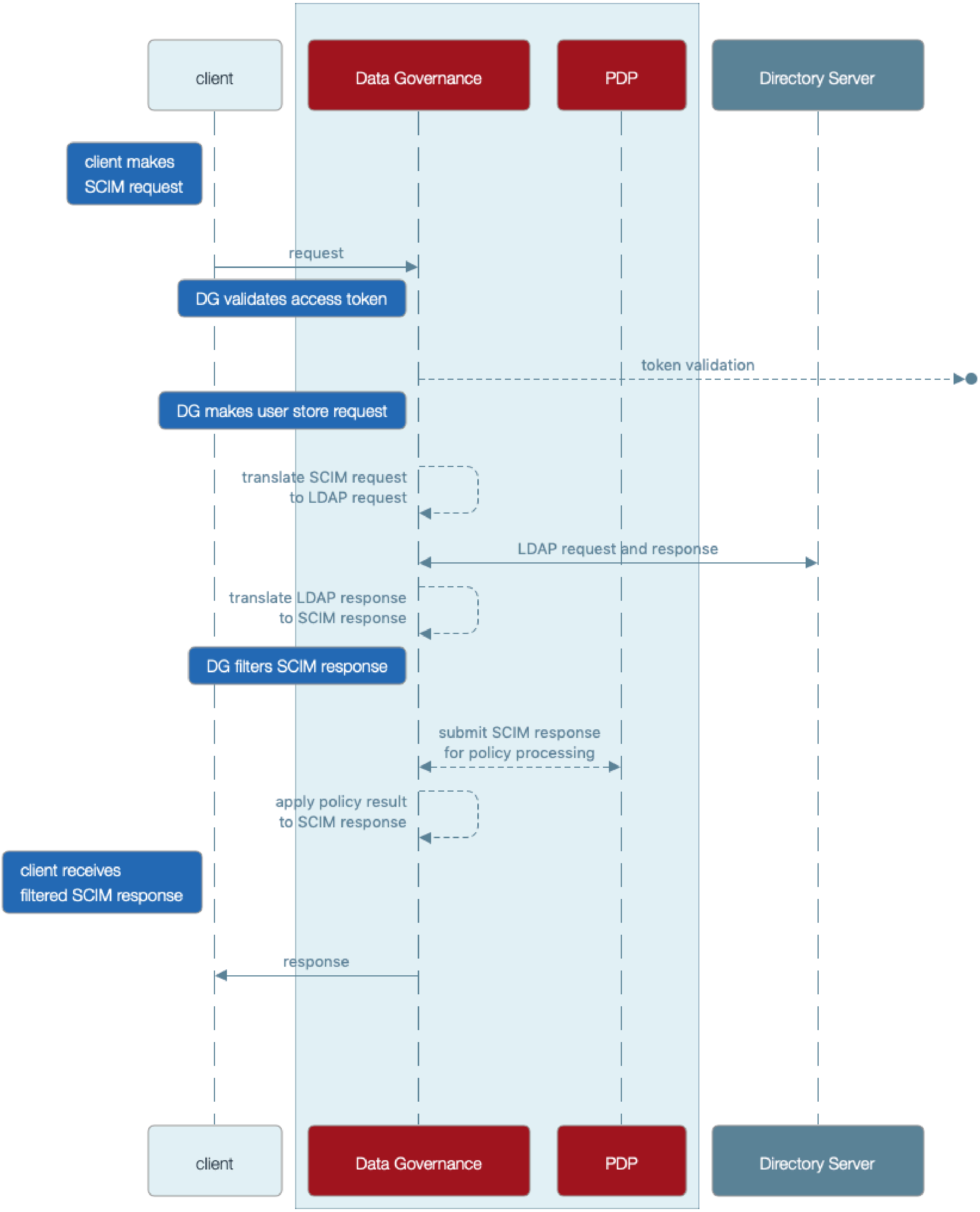
Request and response flow

The System for Cross-domain Identity Management (SCIM) REST API provides an HTTP API for data 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 policy decision point (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


client
Data Governance
PDP
Directory Server
client makes SCIM request
request
DG validates access token
token validation
DG makes user store request
translate SCIM request to LDAP request
LDAP request and response
translate LDAP response to SCIM response
DG filters SCIM response
submit SCIM response for policy processing
apply policy result to SCIM response
client receives filtered SCIM response
response
client
Data Governance
PDP
Directory Server

SCIM configuration basics

PingDataGovernance Server's System for Cross-domain Identity Management (SCIM) subsystem consists of the following components.

SCIM resource types

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 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

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.

ⓘ **Note:**

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

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 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 the System for Cross-domain Identity Management (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 *Configuring the PingDataGovernance User Store*.

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.

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

Start configuring PingDataGovernance Server by adding a store adapter.

```
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`.

```
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.

```
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, see the example User schema in `PingDataGovernance/resource/starter-schemas`.

SCIM endpoints

The following table identifies the endpoints that the System for Cross-domain Identity Management (SCIM) 2.0 REST API provides.

Endpoint	Description	Supported HTTP methods
`/ServiceProviderConfig`	Provides metadata that indicates the PingDataGovernance Server 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
`/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

Endpoint	Description	Supported HTTP methods
/<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

You must authenticate all System for Cross-domain Identity Management (SCIM) requests using OAuth 2.0 bearer token authentication.

Bearer tokens are evaluated using access token validators. The `HttpRequest.AccessToken` attribute supplies the validation result to the policy request, and the `TokenOwner` attribute provides the user identity associated with the token. Policies use this authentication information to affect the processing of requests and responses.

SCIM policy requests

For every System for Cross-domain Identity Management (SCIM) request or response, one or more policy requests are sent to the policy decision point (PDP) for authorization.

Policies can use a policy request's **`action`** value to determine the processing phase and to act accordingly. Understanding how the SCIM service formulates policy requests will help you to create and troubleshoot policies more effectively.

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, you can reuse policies that target the **`retrieve`** action to specify read-access control rules. You can disable this **`retrieve`** action for a SCIM Resource Type if policies are only used for authorization before the operation. To do so, set the SCIM Resource Type's `disable-response-processing` property to `true`. The resource is then returned as-is after the operation completes. This property also affects SCIM searches.

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

Operation	Actions
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 187.

Enable detailed decision logging and view all policy request attributes in action, particularly when learning how to develop SCIM policies. For more information, see *Policy Decision logger* on page 289.

Policy request attributes

The following tables describe policy request attributes and their functions.

The following table identifies the attributes associated with a policy request that the System for Cross-domain Identity Management (SCIM) service generates.

Policy request attribute	Description	Type
action	Identifies the SCIM request as one of the following types: ▪ create ▪ modify ▪ retrieve ▪ delete ▪ search ▪ search-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
domain	Unused.	String
identityProvider	Name of the access token validator that evaluates the bearer token used in an incoming request.	String
service	Identifies the SCIM service and resource type using a value of the form SCIM2.*<resource type>*. For example, for a request using the "Users" resource type, the service value would be SCIM2.Users.	String

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

Attribute	Description	Type
`HttpRequest.AccessToken`	Parsed access token. For more information, see the following table.	Object
`HttpRequest.ClientCertificate`	Properties of the client certificate, if one is used.	Object
`HttpRequest.CorrelationId`	A unique value that identifies the request and response, if available.	String
`HttpRequest.IPAddress`	The client IP address.	String
`HttpRequest.QueryParameters`	Request URI query parameters.	Object
`HttpRequest.RequestBody`	The request body, if available. This attribute is available for POST, PUT, and PATCH requests.	Object
`HttpRequest.RequestHeaders`	The HTTP request headers.	Object
`HttpRequest.RequestURI`	The request URI.	String
`HttpRequest.ResourcePath`	Uniquely identifies the SCIM resource that is being requested, in the format `<Resource Type>/<SCIM ID>`, as the following example shows: `Users/0450b8db-f055-35d8-8e2f-0f203a291cd1`	String
`HttpRequest.ResponseBody`	The response body, if available. This attribute is provided only for outbound policy requests.	Object
`HttpRequest.ResponseHeaders`	The HTTP response headers, if available.	Object
`HttpRequest.ResponseStatus`	The HTTP response status code, if available.	Number
`impactedAttributes`	Provides the set of attributes that the request modifies.	Collection
`SCIM2`	Provides additional, SCIM2-specific information about the request.	Object
`TokenOwner`	Access token subject as a SCIM resource, as obtained by the access token validator.	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
`access_token`	The actual access token from the client request.	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

Attribute	Description	Type
`client_id`	The client ID of the application that was granted the access token.	String
`expiration`	Date and time at which the access token expires.	DateTime
`issued_at`	Date and time at which the access token was issued.	DateTime
`issuer`	Token issuer. This attribute is usually a URI that identifies the authorization server.	String
`not_before`	Date and time before which a resource server does not accept the access token.	DateTime
`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 `<resource type>/<resource ID>`.	String
`token_type`	The token type, as set by the authorization server. This value is typically set to `bearer`.	String
`user_token`	Flag that the access token validator sets to indicate that the token was issued originally to a subject. If this flag is `false`, the token does not have a subject and was issued directly to a client.	Boolean
`username`	Subject's user name. This attribute is a user identifier that the authorization server sets.	String

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

Attribute	Description	Type
`algorithm`	Name of the certificate signature algorithm, such as `SHA256withRSA`.	String
`algorithmOID`	Signature algorithm OID.	String
`issuer`	Distinguished name (DN) of the certificate issuer.	String
`notAfter`	Expiration date and time of the certificate.	DateTime

Attribute	Description	Type
`notBefore`	Earliest date on which the certificate is considered valid.	DateTime
`subject`	DN of the certificate subject.	String
`valid`	Indicates whether the certificate is valid.	Boolean

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

Attribute	Description	Type
`modifications`	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
`resource`	Complete SCIM resource that the request targets. This attribute is available for GET, PUT, PATCH, and DELETE requests. The `resource` 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 187.	Object

About SCIM searches

Search requests are used to return System for Cross-domain Identity Management (SCIM) resources. You can constrain search requests using filters.

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, clients should 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 add a filter automatically to search requests.

SCIM search policy processing

System for Cross-domain Identity Management (SCIM) policy processing involves denying or modifying a search request and then filtering the results.

Policy processing for SCIM searches occurs in the following phases:

1. Policies deny or modify a search request. For more information, see *Search request authorization* on page 188.
2. Policies filter the search result set. For more information, see *Search response authorization* on page 188.

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 **`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:

You can only use advice types that are written specifically for the **`search`** action. For example, you can use the Add Filter advice type to constrain the scope of a search.

You can also use the Combine SCIM Search Authorizations advice type at this point. If you use this advice, search results are authorized by using a special mode, described in *Search response authorization* on page 188.

Search response authorization
After a search is performed, the resulting **`search`** response is authorized in one of three ways: default authorization, optimized search response authorization, and no authorization.

Default authorization

The default authorization mode simplifies policy design but can generate a large number of policy requests. For every System for Cross-domain Identity Management (SCIM) resource that the search returns, a policy request is issued by using the **`retrieve`** action. If the policy result is **`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 currently under consideration.

The following JSON 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 policy decision point (PDP).

No authorization

If you do not need policy processing for the search results on a SCIM Resource Type, such as if policies are only used for authorization before the search and not filtering the results, set that SCIM Resource Type's `disable-response-processing` property to `true`. The search results will be returned as they were received from the external server. This behavior can improve performance for requests that return large numbers of search results. This property also affects other SCIM operations.

Lookthrough limit

Because a policy evaluates every System for Cross-domain Identity Management (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 their search filter to return fewer matches.

Disabling the SCIM REST API

Disable the System for Cross-domain Identity Management (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.

Steps

- Use the following command to remove the extension from the HTTP connection handler and restart it.

  ```
  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
  ```

  Note:

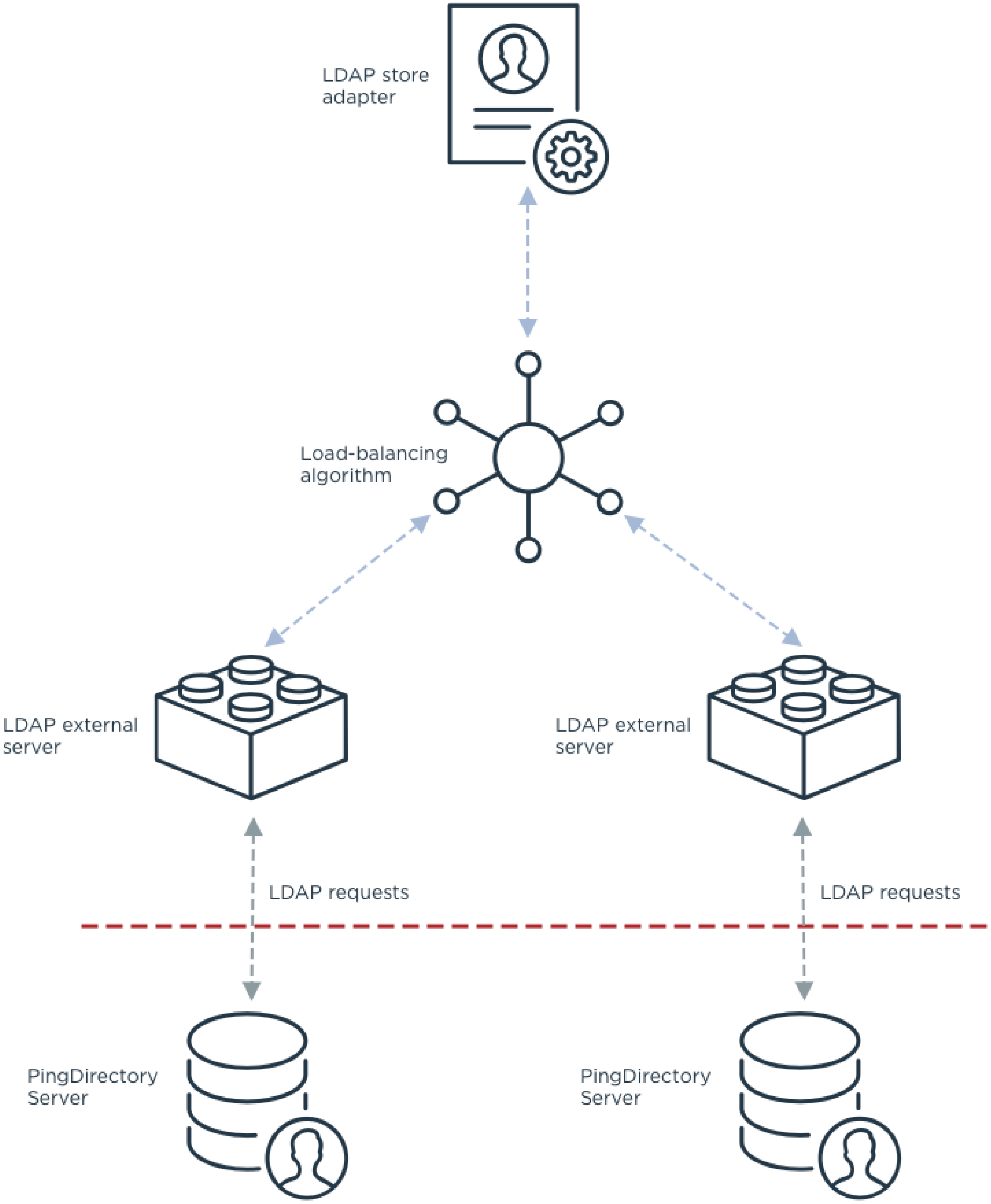
  When the SCIM REST API is disabled, access token validators still use PingDataGovernance Server's SCIM system to look up token owners.

About the SCIM user store

This topic focuses on the relationship between the PingDataGovernance Server SCIM subsystem and its backend data stores, particularly LDAP directory servers.

For general information about SCIM configuration, see *SCIM configuration basics* on page 179.

The PingDataGovernance Server SCIM 2.0 REST API and SCIM token resource lookup methods rely on external data stores, collectively called a *user store*, to locate user records. Typically, a user store is composed of a set of PingDirectory Servers, optionally fronted by a set of PingDirectoryProxy Servers. The SCIM subsystem manages communication with the user store through a *store adapter*, which translates SCIM requests into requests native to the data stores. The following diagram shows an example setup.



PingDataGovernance Server includes a store adapter type for use with LDAP data stores, the *LDAP store adapter*. The LDAP store adapter manages communications to a pool of LDAP servers using a *load-balancing algorithm*. PingDataGovernance Server supports two types of load-balancing algorithms.

Load-balancing algorithm type	Description
Failover load-balancing algorithm	Attempts to always send requests to the same backend LDAP server. If the preferred server is not available, then it fails over to alternate servers.
Fewest operations load-balancing algorithm	Forwards requests to the backend LDAP server with the fewest operations currently in progress. You should only use this load-balancing algorithm when all backend servers are Directory Proxy Servers.

Typically, you connect a load-balancing algorithm to its backend LDAP servers by defining *LDAP external servers* in the configuration and attaching them to the load-balancing algorithm configuration. An LDAP external server configuration manages the actual LDAP connections to a backend LDAP server, such as PingDirectory Server.

ⓘ **Note:**

Alternatively, if all backend LDAP servers are PingDirectory Servers (version 8.0.0.0 and later), you can configure a load-balancing algorithm to automatically discover the backend servers. See *Automatic backend discovery* on page 195.

LDAP external servers monitor and report the availability of backend LDAP servers using LDAP health checks. See *LDAP health checks* on page 199.

Defining the LDAP user store

You can define your user store with the external data servers using **`create-initial-config`**. If you need more flexibility though, you can define the LDAP store manually.

For information about these options, see:

- *Defining the LDAP user store with create-initial-config* on page 191
- *Defining the LDAP user store manually* on page 192

Defining the LDAP user store with create-initial-config

The **`create-initial-config`** tool provides limited support for configuring SCIM and the user store configuration needed to connect the SCIM subsystem to a set of LDAP directory servers.

This tool creates the following configuration:

- An LDAP store adapter named `UserStoreAdapter`
- A load-balancing algorithm named `User Store LBA`
- One or more LDAP external servers
- (Optional) A SCIM resource type named `Users`
- (Optional) SCIM schema, attributes, and attribute mappings for the `Users` resource type

If run interactively, **`create-initial-config`** walks you through the configuration process. You should be prepared to provide connection information for your directory servers.

You can also run **`create-initial-config`** noninteractively, which is useful when performing a scripted deployment. For an example, see *Configuring the PingDataGovernance User Store*.

The following table describes a key subset of the tool's command-line options.

Option	Description
`--governanceBindDN`	The bind DN for a user account that PingDataGovernance Server will use to access backend LDAP servers. Create this account using the **`prepare-external-store`** tool.
`--governanceBindPassword`	The password for the above account.
`--userStore`	The host, LDAP / LDAPS port, and optional location of a backend LDAP server. You can specify this option once per each backend server.
`--userStoreBaseDN`	The base DN under which entries are stored.
`--userObjectClass`	The structural LDAP object class of entries for the SCIM subsystem to handle if `--initialSchema` has the `none` or `pass-through` value.
`--initialSchema`	The SCIM schema and resource type configuration to use. Supports the following values: ▪ `pass-through` Creates a pass-through SCIM resource type called `Users` for the LDAP object class specified by the `--userObjectClass` option. ▪ `user` Creates a mapping SCIM resource type called `Users` with an example schema. For more information about this schema, see *`<server-root>`*`/resource/starter-schemas/README.txt`. ▪ `none` Does not create a SCIM resource type.

For more information about running **`create-initial-config`**, see its help by running the following command.

```
create-initial-config --help
```

When using **`create-initial-config`** noninteractively, you should also run **`prepare-external-store`** for each backend LDAP server. This tool creates a privileged user account on the LDAP server for use by PingDataGovernance Server and configures a set of global access control instructions (ACIs) needed by this account.

Defining the LDAP user store manually

If you require more flexibility than **`create-initial-config`** provides, you can manually configure the SCIM subsystem and its connectivity to the LDAP user store. However, if you have not done this before, first use **`create-initial-config`** to generate an example configuration and then customize that configuration.

About this task

This task shows how to define two backend LDAP servers and a failover load-balancing algorithm. Also, it shows how to connect the load-balancing algorithm to an existing LDAP store adapter named `UserStoreAdapter`.

> **Note:** The example is simplified and does not discuss SSL connection management. When using SSL to connect to an LDAP external server, you must configure PingDataGovernance Server to trust the server certificate presented by the LDAP external server using a trust manager provider.

Steps

1. Run `prepare-external-store` for each backend LDAP server. This tool creates a service account with the access rights needed by PingDataGovernance Server.
   For example:

   ```
   prepare-external-store \
     --hostname ds1.example.com \
     --port 636 \
     --useSSL \
     --trustAll \
     --bindDN "cn=directory manager" \
     --bindPassword password \
     --governanceBindDN 'cn=Governance User,cn=Root DNs,cn=config' \
     --governanceBindPassword password \
     --userStoreBaseDN 'ou=People,dc=example,dc=com'
   ```

2. Create an LDAP external server entry for each backend LDAP server. This configures how PingDataGovernance Server connects to each LDAP server.
   For example:

   ```
   dsconfig create-external-server \
     --server-name DS1 \
     --type ping-identity-ds \
     --set server-host-name:ds1.example.com \
     --set server-port:636 \
     --set location:Minneapolis \
     --set 'bind-dn:cn=Governance User, cn=Root DNs,cn=config' \
     --set password:password \
     --set connection-security:ssl \
     --set key-manager-provider:Null \
     --set trust-manager-provider:JKS

   dsconfig create-external-server \
     --server-name DS2 \
     --type ping-identity-ds \
     --set server-host-name:ds2.example.com \
     --set server-port:636 \
     --set location:Minneapolis \
     --set 'bind-dn:cn=Governance User, cn=Root DNs,cn=config' \
     --set password:password \
     --set connection-security:ssl \
     --set key-manager-provider:Null \
     --set trust-manager-provider:JKS
   ```

3. Create a failover load-balancing algorithm that uses the two LDAP external servers.
   For example:

   ```
   dsconfig create-load-balancing-algorithm \
     --algorithm-name 'User Store LBA' \
     --type failover \
     --set enabled:true \
     --set backend-server:DS1 \
     --set backend-server:DS2
   ```

4. Assign the load-balancing algorithm to an LDAP store adapter. This example assumes that the store adapter `UserStoreAdapter` already exists.
   For example:

   ```
   dsconfig set-store-adapter-prop \
     --adapter-name UserStoreAdapter \
     --set 'load-balancing-algorithm:User Store LBA'
   ```

Location management for load balancing

All PingDirectory and PingDataGovernance servers have a location, which is a label that defines a group of servers with similar response time characteristics. Each location consists of a name and an optional list of preferred failover locations.

The failover and fewest operations load-balancing algorithms, discussed in *About the SCIM user store* on page 190, take server location into account when routing requests. By default, they always prefer LDAP backend servers in the same location as the PingDataGovernance Server. If no servers are available in the same location, they will fall back to any defined failover locations.

You assign a server a location using the `--location` option when you run **`setup`**.

You can manage configuration-level and server-level location settings after setup as explained in the following table.

Task	Corresponding command example

Define a new location.	`dsconfig create-location \`
	`  --location-name Minneapolis`

Define a new location with a failover location. The failover location must already exist.	`dsconfig create-location \`
	`  --location-name Louisville \`
	`  --set preferred-failover-location:Minneapolis`

Add a failover location to an existing location. The failover location must already exist.	`dsconfig set-location-prop \`
	`  --location-name Minneapolis \`
	`  --set preferred-failover-location:Louisville`

Change PingDataGovernance Server's existing location by modifying the global configuration.	`dsconfig set-global-configuration-prop \`
	`  --set location:Minneapolis`

Change a backend LDAP server's location by modifying its LDAP external server entry.	`dsconfig set-external-server-prop \`
	`  --server-name DS1 \`
	`  --set location:Minneapolis`

Configure a load-balancing algorithm to ignore backend LDAP servers' locations when deciding how to route requests.	`dsconfig set-load-balancing-algorithm-prop \`
	`  --algorithm-name "User Store LBA"  \`
	`  --set use-location:false`

Automatic backend discovery

Instead of explicitly specifying all backend LDAP servers in the configuration as LDAP external servers, you can configure PingDataGovernance Server to automatically discover its backend servers.

> ⓘ **Important:** This feature requires that all backend LDAP servers be PingDirectory Servers running version 8.0.0.0 or later. Automatic backend discovery is not supported for PingDirectoryProxy Server or third-party LDAP servers.

To configure automatic backend discovery, you must complete these tasks:

- Join the PingDataGovernance Server to the same topology as the PingDirectory Servers.
- Configure the PingDataGovernance Server's load-balancing algorithm with an LDAP external server template. This template provides the connection and health check settings that PingDataGovernance Server uses for all PingDirectory Servers.
- Configure the topology registry entry for each PingDirectory Server to indicate the name of the PingDataGovernance Server load-balancing algorithm.

Joining a PingDataGovernance Server to an existing PingDirectory Server topology

To use automatic backend discovery, the PingDataGovernance Server must be a member of the same topology of each backend PingDirectory Server.

You can join a PingDataGovernance Server to a PingDirectory Server topology at the time that you set it up or after setup using the **`manage-topology`** command.

For information about these options, see:

- *Joining a topology at setup* on page 195
- *Joining a topology with manage-topology* on page 196

Joining a topology at setup

To join a new PingDataGovernance Server to an existing PingDirectory Server topology during setup, provide connection information for one of the PingDirectory Servers to the **`setup`** tool using its `--existingDSTopology*` options. This PingDirectory Server must be running when you execute the **`setup`** tool.

The following table lists some common setup options for joining a PingDirectory Server topology. For a complete list of options, run **`setup --help`**.

Option	Description
`--existingDSTopologyHostName`	The address of a PingDirectory Server instance in the topology to be joined.
`--existingDSTopologyPort`	The LDAP / LDAPS port for communication with the PingDirectory Server to retrieve information about the topology.
`--existingDSTopologyUseSSL`	Indication that the communication with the PingDirectory Server to retrieve information about the topology should be encrypted with SSL.
`--existingDSTopologyUseJavaTruststore`	The path to a JKS trust store that has the information needed to trust the certificate presented by the PingDirectory Server when using SSL or StartTLS.
`--existingDSTopologyUsePkcs12Truststore`	The path to a PKCS #12 trust store that has the information needed to trust the certificate presented by the PingDirectory Server when using SSL or StartTLS.

Option	Description
`--existingDSTopologyTrustStorePassword`	The password needed to access the contents of the JKS or PKCS #12 trust store. A password is typically required when using a PKCS #12 trust store but is optional when using a JKS trust store.
`--existingDSTopologyBindDN`	The DN of the account to use to authenticate to the PingDirectory Server, such as `cn=Directory Manager`. This account must have full read and write access to the configuration and to manage the topology.
`--existingDSTopologyBindPassword`	The password for the account to use to authenticate to the PingDirectory Server.

Joining a topology with manage-topology

To join an existing PingDataGovernance Server to an existing PingDirectory Server topology, you can use the **`manage-topology add-server`** command to provide connection information for one of the PingDirectory Servers. This PingDirectory Server must be running when you execute the **`setup`** tool.

The following table lists the options that specify connection information for a PingDirectory Server. To see this command's complete set of options, run **`manage-topology add-server --help`**.

Option	Description
`--remoteServerHostname`	The address of a PingDirectory Server in the topology to be joined.
`--remoteServerPort`	The LDAP / LDAPS port for communication with the PingDirectory Server.
`--remoteServerConnectionSecurity`	The type of security to use when communicating with the remote server. This value can be: ▪ `useSSL` Indicates that the communication should be encrypted with SSL ▪ `useStartTLS` Indicates that the communication should be encrypted with the StartTLS extended operation ▪ `noSecurity` Indicates that the communication should not be encrypted
`--remoteServerBindDN`	The DN of the account to use to authenticate to the PingDirectory Server, such as `cn=Directory Manager`. This account must be able to modify the configuration of the target server.
`--remoteServerBindPassword`	The password for the account to use to authenticate to the PingDirectory Server.
`--remoteServerBindPasswordFile`	The path to a file containing the password for the account to use to authenticate to the PingDirectory Server.

Option	Description
`--adminUID`	User ID of the topology-wide administrator. This is typically the account used to enable replication for the PingDirectory Servers.
`--adminPassword`	The password of the topology-wide administrator.

Configuring a load-balancing algorithm with an LDAP external template

When using automatic backend discovery, you configure a load-balancing algorithm with a single LDAP external template instead of one or more LDAP external servers that refer to specific backend LDAP servers.

An LDAP external server template provides a load-balancing algorithm with many of the settings that it should use when communicating with a backend server that has been discovered from the topology registry. An LDAP external server template configuration object has most of the same properties as an LDAP external server configuration object but omits those related to information that it obtains from the topology registry. The omitted properties include:

- `server-host-name`
- `server-port`
- `location`
- `connection-security`

In addition, the `health-check-state` property is also not available for LDAP external server templates because it primarily applies to individual servers rather than all of the servers associated with a load-balancing algorithm.

Because the only LDAP servers that can be in the topology registry are PingDirectory Servers, most of the remaining properties in LDAP external server templates have the same default values as the corresponding properties in the Ping Identity DS External Server type. However, there are some exceptions, including the following:

- The `authentication-method` property has a default value of `inter-server` in LDAP external server templates, while it has a default value of `simple` in Ping Identity DS external servers. The `inter-server` authentication type indicates that the PingDataGovernance Server should authenticate to the PingDirectory Server with a proprietary authentication method that uses inter-server certificates stored in the topology registry.
- The `key-manager-provider` property has a default value of `Null` in LDAP external server templates, while it has no default value in Ping Identity DS external servers. When using the inter-server authentication type, the topology registry is used to obtain the inter-server certificates, so no additional key manager provider is required.
- The `trust-manager-provider` property has a default value of `JVM-Default` in LDAP external server templates, while it has no default value in Ping Identity DS external servers. When using the inter-server authentication type, the topology registry is used to obtain information about the listener certificates that the servers are expected to present.

ⓘ **Note:**

When using automatic backend discovery, it is not necessary to run `prepare-external-store` to create a service account on each PingDirectory Server.

The following example shows how to create an LDAP external template and assign it to a new load-balancing algorithm.

```
dsconfig create-ldap-external-server-template \
  --template-name 'User Store'
```

```
dsconfig create-load-balancing-algorithm \
  --algorithm-name 'User Store LBA'  \
  --type failover  \
  --set enabled:true  \
  --set 'ldap-external-server-template:User Store'
```

Configuring automatic backend discovery

The following example shows how to configure a load-balancing algorithm to automatically discover backend LDAP servers. Also, it shows how to connect the load-balancing algorithm to an existing LDAP store adapter called `UserStoreAdapter`.

About this task

This example assumes that you have already created a topology of PingDirectory Servers and that the servers are currently available.

Steps

1. Create an LDAP external server template. This template configures how PingDataGovernance Server connects to each LDAP server that it discovers. Typically, the default settings are sufficient, so this example only specifies the template name.
   For example:

   ```
   dsconfig create-ldap-external-server-template \
     --template-name 'User Store'
   ```

2. Create a failover load-balancing algorithm that uses the LDAP external server template.
   For example:

   ```
   dsconfig create-load-balancing-algorithm \
     --algorithm-name 'User Store LBA' \
     --type failover \
     --set enabled:true \
     --set 'ldap-external-server-template:User Store'
   ```

3. Assign the load-balancing algorithm to an LDAP store adapter. This example command assumes that the store adapter `UserStoreAdapter` already exists.
   For example:

   ```
   dsconfig set-store-adapter-prop \
     --adapter-name UserStoreAdapter \
     --set 'load-balancing-algorithm:User Store LBA'
   ```

4. Run **`manage-topology add-server`** to connect the PingDataGovernance Server to a running PingDirectory Server.
   For example:

   ```
   manage-topology add-server \
     --remoteServerHostname ds1.example.com \
     --remoteServerPort 636 \
     --remoteServerConnectionSecurity useSSL \
     --remoteServerBindDN "cn=Directory Manager" \
     --remoteServerBindPassword password \
     --adminUID admin \
     --adminPassword password
   ```

5. Configure each PingDirectory Server in the topology to use PingDataGovernance Server's load-balancing algorithm. You should be able to run this command from any server in the topology. The following commands configure two PingDirectory Servers with the instance names `ds1` and `ds2`.
   For example:

```
dsconfig set-server-instance-prop \
  --instance-name ds1 \
  --set 'load-balancing-algorithm-name:User Store LBA'

dsconfig set-server-instance-prop \
  --instance-name ds2 \
  --set 'load-balancing-algorithm-name:User Store LBA'
```

LDAP health checks

LDAP health checks provide information about the health and availability of the LDAP directory servers, which has a direct effect on services, such as the PingDataGovernance Server System for Cross-domain Identity Management (SCIM) 2 service and the SCIM Token Resource Lookup method.

Overview

The LDAP health check component provides information about the availability of LDAP external servers. The health check result includes one of the following server states:

AVAILABLE

Completely accessible for use.

DEGRADED

The server is ready for use if necessary, but it has a condition that might make it less desirable than other servers (for example, it is slow to respond or has fallen behind in replication).

UNAVAILABLE

Completely unsuitable for use (for example, the server is offline or is missing critical data)

Health check results also include a numeric score, which has a value between 1 and 10, that can help rank servers with the same state. For example, if two servers are available, you can configure PingDataGovernance Server to prefer the server with the higher score.

PingDataGovernance Server periodically invokes health checks to monitor each LDAP external server. It might also initiate health checks in response to failed operations. It checks the health of the LDAP external servers at intervals configured in the LDAP server’s health-check-frequency property.

The results of health checks performed by PingDataGovernance Server are made available to the load-balancing algorithms to take into account when determining where to send requests. PingDataGovernance Server attempts to use servers with a state of AVAILABLE before trying servers with a state of DEGRADED. It never attempts to use servers with a state of UNAVAILABLE. Some load-balancing algorithms might also take the health check score into account, such as the health-weighted load-balancing algorithm, which prefers servers with higher scores over those with lower scores. You must configure the algorithms that work best for your environment.

In some cases, an LDAP health check might define different sets of criteria for promoting and demoting the state of a server. A DEGRADED server might need to meet more stringent requirements to meet the criteria for AVAILABLE than it originally took to meet the criteria for DEGRADED. For example, if response time is used to determine the health of a server, then PingDataGovernance Server might have a faster response time threshold for transitioning a server from DEGRADED back to AVAILABLE than the threshold used to consider it DEGRADED in the first place. This threshold difference can help avoid cases in which a server repeatedly transitions between the two states because it is operating near the threshold.

For information about how to configure health checks, see *Configuring a health check using dsconfig* on page 201. To associate a health check with an LDAP external server and set the health check frequency, you must configure the health-check and health-check-frequency properties of the LDAP external server.

ⓘ **Note:**

The default Consume Admin Alerts and Get Root DSE LDAP health checks apply to all LDAP external servers, even if you did not explicitly configure and add them to an LDAP external server's `health-check` property.

To disable this behavior, reset the `use-for-all-servers` property for each LDAP health check. For example:

```
dsconfig set-ldap-health-check-prop \
  --check-name 'Consume Admin Alerts' \
  --reset use-for-all-servers
```

Available health checks

PingDataGovernance Server provides the following LDAP health checks.

<table>
<tr><th>Health check</th><th>Description</th></tr>
<tr><td>Measure the response time for searches and examine the entry contents</td><td>The health check might retrieve a monitoring entry from a server and base the health check result on whether the entry was returned, how long it took to be returned, and whether the value of the returned entry matches what was expected.</td></tr>
<tr><td>Monitor the replication backlog</td><td>If a server falls too far behind in replication, then a PingDataGovernance Server can stop sending requests to it. A server is classified as DEGRADED or UNAVAILABLE if the threshold is reached for the number of missing changes, the age of the oldest missing change, or both.</td></tr>
<tr><td>Consume PingDataGovernance Server administrative alerts</td><td>If a PingDirectory Server indicates there is a problem, it flags itself as DEGRADED or UNAVAILABLE. When a PingDataGovernance Server detects this, it stops sending requests to the server.
You can configure a PingDataGovernance Server to detect administrative alerts as soon as they are issued by maintaining an LDAP persistent search for changes within the <code>cn=alerts</code> branch of a PingDirectory Server. When PingDataGovernance Server is notified by the PingDirectory Server of a new alert, it can immediately retrieve the base <code>cn=monitor</code> entry of the PingDirectory Server.
<table>
<tr><th>When <code>cn=monitor</code> entry has value for this attribute:</th><th>PingDataGovernance Server should consider PingDirectory Server to be:</th></tr>
<tr><td><code>unavailable-alert-type</code></td><td>UNAVAILABLE</td></tr>
<tr><td><code>degraded-alert-type</code></td><td>DEGRADED</td></tr>
</table>
</td></tr>
<tr><td>Monitor the busyness of the server</td><td>If a server becomes too busy, the health check might mark it as DEGRADED or UNAVAILABLE so that less heavily loaded servers are preferred.</td></tr>
</table>

Configuring a health check using dsconfig

Create any health check according to the following instructions.

Steps

1. Use the **dsconfig** tool to configure the LDAP external server locations.

```
$ bin/dsconfig
```

2. Type the host name or IP address for your PingDataGovernance Server, or press **Enter** to accept the default, `localhost`.

```
Data Governance Server host name or IP address [localhost]:
```

3. Type the number corresponding to how you want to connect to PingDataGovernance, or press **Enter** to accept the default, LDAP.

```
How do you want to connect?
  1) LDAP
  2) LDAP with SSL
  3) LDAP with StartTLS
```

4. Type the port number for your PingDataGovernance Server, or press **Enter** to accept the default, 389.

```
Data Governance Server port number [389]:
```

5. Type the administrator's bind distinguished name (DN) or press **Enter** to accept the default (cn=Directory Manager), and then type the password.

```
Administrator user bind DN [cn=Directory Manager]:
Password for user 'cn=Directory Manager':
```

6. Enter the number corresponding to LDAP health checks.
   a. Enter the number to create a new LDAP health check, then press `n` to create a new health check from scratch.
7. Select the type of health check you want to create.

   This example demonstrates the creation of a new search LDAP health check.

```
>>> Select the type of LDAP Health Check that you want to create:

      1)  Admin Alert LDAP Health Check
      2)  Custom LDAP Health Check
      3)  Groovy Scripted LDAP Health Check
      4)  Replication Backlog LDAP Health Check
      5)  Search LDAP Health Check
      6)  Third Party LDAP Health Check
      7)  Work Queue Busyness LDAP Health Check

      ?) help
      c) cancel
      q) quit

Enter choice [c]: 5
```

8. Specify a name for the new health check.

   In this example, the health check is named `Get example.com`.

```
>>>> Enter a name for the search LDAP Health Check that you want to create:
 Get example.com
```

9. Enable the new health check.

```
>>>> Configuring the 'enabled' property

Indicates whether this LDAP health check is enabled for use in the
 server.

Select a value for the 'enabled' property:

   1) true
   2) false

   ?) help
   c) cancel
   q) quit

Enter choice [c]: 1
```

10. Configure the properties of the health check.

You might need to modify the `base-dn` property, as well as one or more response time thresholds for non-local external servers, accommodating WAN latency.

The following example is a search LDAP health check for the single entry `dc=example,dc=com`, which considers non-local responses of up to two seconds healthy.

```
>>>> Configure the properties of the Search LDAP Health Check

           Property                                   Value(s)
           -----------------------------------------------------------
      1)   description                                -
      2)   enabled                                    true
      3)   use-for-all-servers                        false
      4)   base-dn                                    "dc=example,dc=com"
      5)   scope                                      base-object
      6)   filter                                     (objectClass=*)
      7)   maximum-local-available-response-time      1 s
      8)   maximum-nonlocal-available-response-time   2 s
      9)   minimum-local-degraded-response-time       500 ms
      10)  minimum-nonlocal-degraded-response-time    1 s
      11)  maximum-local-degraded-response-time       10 s
      12)  maximum-nonlocal-degraded-response-time    10 s
      13)  minimum-local-unavailable-response-time    5 s
      14)  minimum-nonlocal-unavailable-response-time 5 s
      15)  allow-no-entries-returned                  true
      16)  allow-multiple-entries-returned            true
      17)  available-filter                           -
      18)  degraded-filter                            -
      19)  unavailable-filter                         -

      ?) help
      f) finish - create the new Search LDAP Health Check
      d) display the equivalent dsconfig arguments to create this object
      b) back
      q) quit
```

Connecting non-LDAP data stores

The PingDataGovernance Server SCIM subsystem supports non-LDAP data stores using custom store adapter extensions. For more information, see the Server SDK.

About the PDP API

The PingDataGovernance Server policy decision point (PDP) API provides an endpoint to support non-API use cases.

ⓘ **Important:**

The PDP API feature requires PingDataGovernance Premier. For more information, contact your Ping Identity account representative.

The PingDataGovernance Server's main functionality is to enforce fine-grained policies for data accessed through APIs. However, organizations might need to use the core Policy Decision Service for non-API use cases. For example, an application server might use it to request policy decisions when generating dynamic web content. In this configuration, PingDataGovernance Server becomes the PDP, and the application server becomes the policy enforcement point (PEP).

Enforcement points request policy decisions based on a subset of the XACML-JSON standard. For more information, see *XACML 3.0 JSON Profile 1.1*.

ⓘ **Note:**

The PDP API can indicate when a request or response triggers advice, but the application server must implement the advice.

To make the PDP API client available, you must:

- Configure the PingDataGovernance Server with a feature-enabled license during setup.
- Configure an Access Token Validator. For more information, see *Access Token Validators*.
- Configure the Policy Decision Point Service. For more information, see *Use policies in a production environment*.

The PDP API supports the *MultiRequests JSON object*, which allows a client to make multiple decision requests in a single HTTP request.

ⓘ **Note:**

Because this object also supports single decision requests, it is the only supported XACML-JSON request format.

Request and response flow

The policy decision point (PDP) API provides an HTTP API for decisions determined based on the policies configured within the PingDataGovernance Server Policy Decision Service.

The PDP API is implemented as a single endpoint, which consuming application servers can access using POST requests to the `/pdp` path. The HTTP requests must include the appropriate `Content-Type` and `Accept` headers, and request bodies must adhere to the XACML-JSON standard. For more information, see *Requests* on page 204.

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

A successful PDP API request goes through the following two-phase flow:

1. 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 batch decision request and attempts to authorize the client.
2. On authorize success, the request is handed off to the Policy Decision Service to process decisions in batch for the PDP API. The PDP API then converts the batch decision responses 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 batch decision request for the policy decision point to be consumed by the Policy Decision Service. Policies can match a decision request by `Service`, `Domain`, `Action`, or other attributes.

The following example XACML-JSON request body illustrates the conversion to a batch decision request. For an example with more than one decision request, see *Example* on page 208.

```
{
  "Request": {
    "MultiRequests": {
      "RequestReference": [{
        "ReferenceId": [
          "dom",
          "act",
          "srv",
          "idp",
          "att"
        ]
      }]
    },
    "AccessSubject": [{
      "Id": "dom",
      "Attribute": [{
        "AttributeId": "domain",
        "Value": "Sales.Asia Pacific"
      }]
    }],
    "Action": [{
      "Id": "act",
      "Attribute": [{
        "AttributeId": "action",
        "Value": "Retrieve"
      }]
    }],
    "Resource": [{
      "Id": "srv",
      "Attribute": [{
        "AttributeId": "service",
        "Value": "Mobile.Landing page"
      }]
    }],
    "Environment": [{
      "Id": "idp",
      "Attribute": [{
        "AttributeId": "symphonic-idp",
        "Value": "Social networks.Spacebook"
      }]
    }],
    "Category": [{
      "Id": "att",
      "Attribute": [{
        "AttributeId": "attribute:Prospect name",
        "Value": "B. Vo"
      }]
    }]
  }
}
```

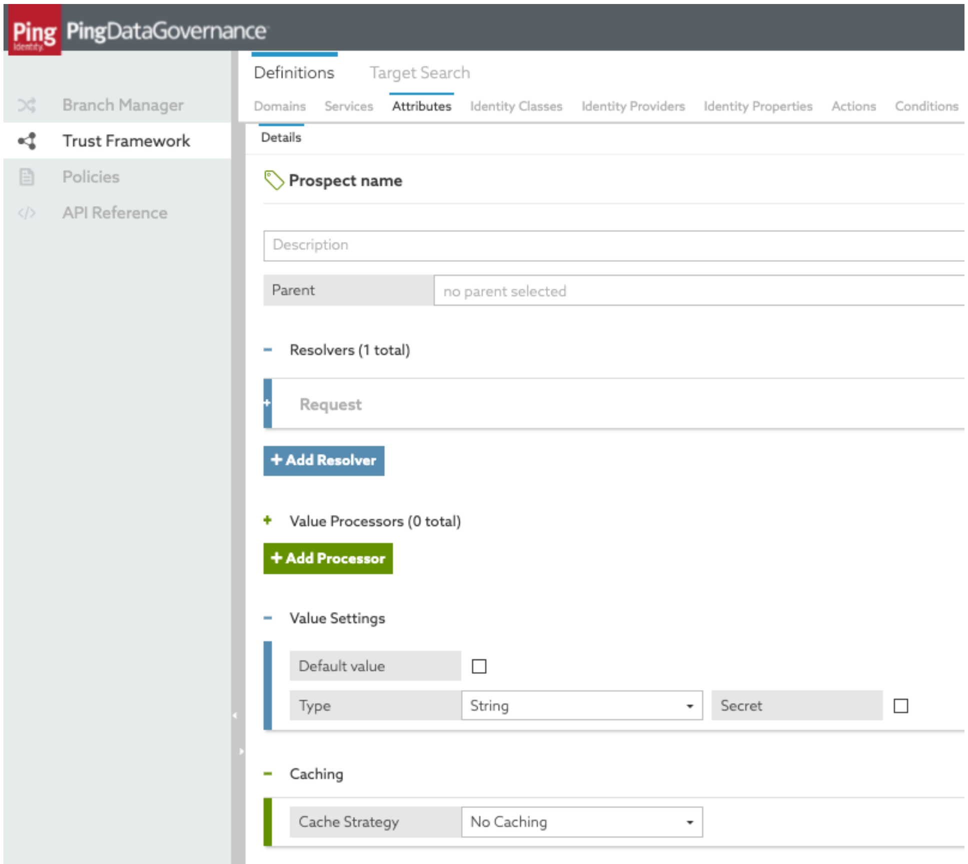
The previous example shows a single decision request with the following attributes:

- A domain of `Sales.Asia Pacific`
- An action of `Retrieve`
- A service of `Mobile.Landing page`
- An identity provider of `Social networks.Spacebook`
- A single attribute named `Prospect name`, with a value of `B. Vo`

The following table shows how these values map from the Trust Framework entities to the XACML-JSON request.

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type	Example value
`$.Request`	`$.AccessSubject[*].Attribute[?(@.AttributeId == "domain")].Value`	Domain	`Sales.Asia Pacific`
	`$.Action[*].Attribute[?(@.AttributeId == "action")].Value`	Action	`Retrieve`
	`$.Resource[*].Attribute[?(@.AttributeId == "service")].Value`	Service	`Mobile. Landing page`
	`$.Environment[*].Attribute[?(@.AttributeId == "symphonic-idp")].Value`	Identity Provider	`Social Networks. Spacebook`
	`$.Category[*].Attribute[?(@.AttributeId == "attribute:Prospect name")].Value`	Other Attribute (`Prospect name` in this case)	`B. Vo`

To illustrate how you can match rules against the `Prospect name` Trust Framework attribute, the following image shows how `Prospect name` is defined in the Policy Administration GUI. In this example, the `Prospect name` attribute has a Request resolver and a **Value Settings Type** of `String`.



Prospect name

Description

Parent: no parent selected

Resolvers (1 total)

Request

Value Processors (0 total)

Value Settings

Default value □

Type: String

Secret □

Caching

Cache Strategy: No Caching

Note:

The Trust Framework attribute name must be a case-sensitive match with the decision request `AttributeId` after the `attribute:` prefix is removed.

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 PingDataGovernance Server perform this authorization by only permitting requests with active access tokens that contain the `urn:pingdatagovernance:pdp` scope. You can see this policy in **Global Decision Point# PDP API Endpoint Policies# Token Authorization**.

Note:

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

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 provide a foundation. You can modify these policies if additional authorization logic is required.

Decision processing

On successful client authorization, the PDP API invokes the Policy Decision Service with the batch decision requests converted from the XACML-JSON request.

When writing policy for the PDP API endpoint, you should note the mapping between the XACML-JSON schema and the PingDataGovernance Server decision request. For more information, see *Requests* on page 204. After the Policy Decision Service determines a decision response, it hands the response back to the PDP API to provide to the client.

Responses

The PDP API converts batch decision responses to a XACML-JSON response.

XACML-JSON responses include decisions, such as `Permit` or `Deny`, and any obligations or advice that matched during policy processing.

> **Note:**
>
> The Policy Enforcement Point (PEP) must apply any obligations or advice.

The following table shows the mapping from a decision response to a XACML-JSON response.

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type
`$.Response[*]`	`$.Decision`	Decision
`$.Response[*].Obligations[*]`		Advice (obligatory)
	`$.Id`	Advice code
	`$.AttributeAssigments[?(@.AttributeId == "payload")].Value`	Advice payload
`$.Response[*].AssociatedAdvice[*]`		Advice (non-obligatory)
	`$.Id`	Advice code
	`$.AttributeAssigments[?(@.AttributeId == "payload")].Value`	Advice payload

The following example is an appropriate response based on the request in *Requests* on page 204.

```
{
  "Response": [{
    "Decision": "Permit",
    "Obligations": [{
      "Id": "obligation-id",
      "AttributeAssignments": [{
        "AttributeId": "payload",
        "Value": "payload-value"
      }]
    }],
    "AssociatedAdvice": [{
      "Id": "advice-id",
      "AttributeAssignments": [{
        "AttributeId": "payload",
        "Value": "payload-value"
      }]
    }]
```

```
   }]
}
```

In this example, it is up to the application server to handle the obligations and advice in the response.

Example

This example shows how to use the PDP API in the context of a peer recognition program.

The example company, AnyCompany, has an internal peer recognition program. The peer recognition program allows employees to recognize each other by awarding each other points. The points can be spent in different categories. Each category requires a minimum number of points for the category to become available. When an employee spends enough points in a category, a related product becomes unlocked in an online catalog that the employee can purchase. AnyCompany has implemented a web application where employees spend their points, view their available catalog, and purchase products.

In this example, the web application that implements the online catalog can make the following XACML-JSON request when an employee spends their points. The request includes three decision requests.

```
{
   "Request":{
      "MultiRequests":{
         "RequestReference":[
            {
               "ReferenceId":[
                  "domain-1",
                  "action-1",
                  "service-1",
                  "idp-1",
                  "attributes-1"
               ]
            },
            {
               "ReferenceId":[
                  "domain-1",
                  "action-2",
                  "service-2",
                  "idp-1",
                  "attributes-2"
               ]
            },
            {
               "ReferenceId":[
                  "domain-1",
                  "action-1",
                  "service-3",
                  "idp-1",
                  "attributes-1"
               ]
            }
         ]
      },
      "AccessSubject":[
         {
            "Id":"domain-1",
            "Attribute":[
               {
                  "AttributeId":"domain",
                  "Value":"AnyCompany.Management"
               }
            ]
         }
      ],
      "Action":[
         {
            "Id":"action-1",
            "Attribute":[
               {
                  "AttributeId":"action",
                  "Value":"Update"
               }
            ]
         },
         {
            "Id":"action-2",
            "Attribute":[
               {
                  "AttributeId":"action",
                  "Value":"Retrieve"
               }
            ]
         }
      ],
      "Resource":[
         {
            "Id":"service-1",
            "Attribute":[
               {
                  "AttributeId":"service",
                  "Value":"Peer Recognition.Point allocation"
               }
            ]
         },
         {
            "Id":"service-2",
            "Attribute":[
               {
                  "AttributeId":"service",
                  "Value":"Peer Recognition.Points unspent"
               }
            ]
         },
         {
            "Id":"service-3",
            "Attribute":[
               {
                  "AttributeId":"service",
                  "Value":"Peer Recognition.Products"
               }
            ]
         }
      ],
      "Category":[
         {
            "Id":"attributes-1",
            "Attribute":[
               {
                  "AttributeId":"attribute:User input.User Id",
                  "Value":"self"
               },
               {
                  "AttributeId":"attribute:User input.Entertainment",
                  "Value":8
               },
               {
                  "AttributeId":"attribute:User input.Travel",
                  "Value":5
               },
               {
                  "AttributeId":"attribute:User input.Academics",
                  "Value":6
               },
               {
                  "AttributeId":"attribute:User input.Electronics",
                  "Value":5
               },
               {
                  "AttributeId":"attribute:User input.Sports",
                  "Value":5
               },
               {
                  "AttributeId":"attribute:User input.Food",
                  "Value":7
               },
               {
                  "AttributeId":"attribute:User input.Music",
                  "Value":4
               }
            ]
         },
         {
            "Id":"attributes-2",
            "Attribute":[
               {
                  "AttributeId":"attribute:User input.User Id",
                  "Value":"self"
               }
            ]
         }
      ],
      "Environment":[
         {
            "Id":"idp-1",
            "Attribute":[
               {
                  "AttributeId":"symphonic-idp",
                  "Value":"AnyCompany SSO"
               }
            ]
         }
      ]
   }
}
```

The three decision requests are summarized in the `RequestReference` JSON array. Each JSON object in the array contains a single field, `ReferenceId`. Each `ReferenceId` field contains an array of `Id` references that represent the content of the decision request. The following tables highlight the key components of each decision request.

ⓘ **Note:**

For brevity, only one Trust Framework attribute is listed in each decision request.

First decision request

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type	Example value
`$.Request.AccessSubject[*]`	`$.Attribute[?(@.AttributeId == "domain")].Value`	Domain	`AnyCompany.Management`
`$.Request.Action[*]`	`$.Attribute[?(@.AttributeId == "action")].Value`	Action	`Update`
`$.Request.Resource[*]`	`$.Attribute[?(@.AttributeId == "service")].Value`	Service	`Peer Recognition.Point allocation`
`$.Request.Environment[*]`	`$.Attribute[?(@.AttributeId == "symphonic-idp")].Value`	Identity Provider	`AnyCompany SSO`
`$.Request.Category[*]`	`$.Attribute[?(@.AttributeId == "attribute:User input.Entertainment")]`	Attribute	`8`

Second decision request

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type	Example value
`$.Request.AccessSubject[*]`	`$.Attribute[?(@.AttributeId == "domain")].Value`	Domain	`AnyCompany.Management`
`$.Request.Action[*]`	`$.Attribute[?(@.AttributeId == "action")].Value`	Action	`Retrieve`
`$.Request.Resource[*]`	`$.Attribute[?(@.AttributeId == "service")].Value`	Service	`Peer Recognition.Points unspent`
`$.Request.Environment[*]`	`$.Attribute[?(@.AttributeId == "symphonic-idp")].Value`	Identity Provider	`AnyCompany SSO`
`$.Request.Category[*]`	`$.Attribute[?(@.AttributeId == "attribute:User input.User Id")]`	Attribute	`self`

Third decision request

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type	Example value
`$.Request.AccessSubject[*]`	`$.Attribute[?(@.AttributeId == "domain")].Value`	Domain	`AnyCompany.Management`
`$.Request.Action[*]`	`$.Attribute[?(@.AttributeId == "action")].Value`	Action	`Retrieve`
`$.Request.Resource[*]`	`$.Attribute[?(@.AttributeId == "service")].Value`	Service	`Peer Recognition.Product`
`$.Request.Environment[*]`	`$.Attribute[?(@.AttributeId == "symphonic-idp")].Value`	Identity Provider	`AnyCompany SSO`

Parent (JSON Path)	Field (JSON Path)	PingDataGovernance Trust Framework type	Example value
`$.Request.Category[*]`	`$.Attribute[?(@.AttributeId == "attribute:User input.Travel")]`	Attribute	`5`

The following is an example response to the previous example request.

The XACML-JSON response contains the decision responses for each of the three decision requests. The order of the decision responses corresponds to the order of the decision requests. In the first decision response, the system policy does not detect any problems and permits the employee to change her point allocation. In the second decision response, the system policy allows the employee to view her own unspent points and indicates that the value is now 0. In the third decision response, the system permits the retrieval of the employee's own product catalog and indicates which of the products should be unlocked for purchase.

Given the response, the web application can now render the content for the three decision requests. It renders the 0 unspent points and all catalog products, with purchase buttons disabled where appropriate.

```
{
  "Response": [
    {
      "Decision": "Permit",
      "Obligations": [],
      "AssociatedAdvice": []
    }, {
      "Decision": "Permit",
      "Obligations": [],
      "AssociatedAdvice": [{
        "Id": "remaining-points",
        "AttributeAssignments": [{
          "AttributeId": "payload",
          "Value": "0"
        }]
      }]
    }, {
      "Decision": "Permit",
      "Obligations": [],
      "AssociatedAdvice": [{
        "Id": "catalog",
        "AttributeAssignments": [{
          "AttributeId": "attribute:Derived.Product availability.Trip to
 exotic country",
          "Value": "false"
        }, {
          "AttributeId": "attribute:Derived.Product availability.Super Bowl
 tickets",
          "Value": "false"
        }, {
          "AttributeId": "attribute:Derived.Product availability.Movie
 theater gift card",
          "Value": "true"
        }, {
          "AttributeId": "attribute:Derived.Product
 availability.Encyclopedia subscription",
          "Value": "false"
        }, {
          "AttributeId": "attribute:Derived.Product availability.Dinner at
 5-star restaurant",
          "Value": "true"
        }, {
          "AttributeId": "attribute:Derived.Product availability.Expensive
 laptop",
          "Value": "false"
        }, {
          "AttributeId": "payload",
          "Value": "2020-03-17T16:21:20.175132-05:00"
        }]
      }]
    }]
}
```

Policy Administration GUI configuration

You can configure the PingDataGovernance Administration GUI in several ways.

With an options file, for example, you can define policy configuration keys, a key store, or a trust store.

Also, you can set:

- Database credentials at setup or later
- SpEL Java classes to use for value processing
- The number of requests that appear in the Decision Visualizer

Specifying custom configuration with an options file

You can configure the Policy Administration GUI by editing and implementing the options file.

About this task

You must run **`setup`** in non-interactive command-line mode instead of interactive mode if you need to do any of the following:

- Configure the Policy Administration GUI with a policy configuration key. A policy configuration key is an arbitrary key/value pair that can be referenced by name in the policy Trust Framework. This allows the policy trust store to be defined without hard-coding environment-specific data such as host names and credentials in the trust store.
- Configure a key store for a policy information provider. This defines a client certificate that the policy engine can use for MTLS connections to a policy information provider.
- Configure a trust store for a policy information provider. This defines the set of certificates or root certificates that the policy engine uses to determine whether it trusts the server certificate presented by a policy information provider.
- Customize the Policy Administration GUI's logging behavior.

Steps

1. Make a copy of the default options file provided at `config/options.yml` and then customize the copy to suit your needs.

   The **`setup`** tool supports configuring these options through the use of a YAML options file

   > ⓘ **Note:**
   >
   > When you customize your options file, do not remove or alter the logging section. For guidance about customizing logging behavior, contact Ping support.

2. Configure the Policy Administration GUI with an options file.
   a. Stop the Policy Administration GUI.

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

   b. Run **setup** normally.
   c. Provide the options file using the `--optionsFile` argument.

   For example, the following **setup** command configures a Policy Administration GUI in demo mode using an options file named `my-options.yml`.

   ```
   $ bin/setup demo \
     --adminUsername admin \
     --generateSelfSignedCertificate \
     --decisionPointSharedSecret datagovernance \
     --hostname <pap-hostname> \
     --port <pap-port> \
     --licenseKeyFile <path-to-license> \
     --optionsFile my-options.yml
   ```

   a. Start the Policy Administration GUI.

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

Example: Configure policy configuration keys
You can define one or more policy configuration keys under the options file's `core` section.

These are arbitrary key/value pairs that are typically used to define environment-specific details such as host names and credentials. After you define a policy configuration key, you can reference it by name in the PingDataGovernance Policy Administration GUI Trust Framework. By using a reference, you do not need to hard-code the values in the Trust Framework.

Example

Consider an organization that has two development environments, US-East and US-West. The organization's policies call out to a PingDirectory Consent API policy information provider (PIP), and the Consent API's host name differs depending on the development environment being used. If the Consent API host name was hard-coded in the Trust Framework, then a different Trust Framework would need to be used for each development environment. Instead, you can declare the host name as a policy configuration key in the Policy Administration GUI's configuration.

To set up this policy configuration key, complete the following steps.

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

2. Edit the new options file to define a policy configuration key in the `core` section called `ConsentHostname`.

   ```
   core:
     ConsentHostname: consent-us-east.example.com
   # Other options omitted for brevity...
   ```

3. Stop the Policy Administration GUI.

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

4. Run **setup** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret datagovernance \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

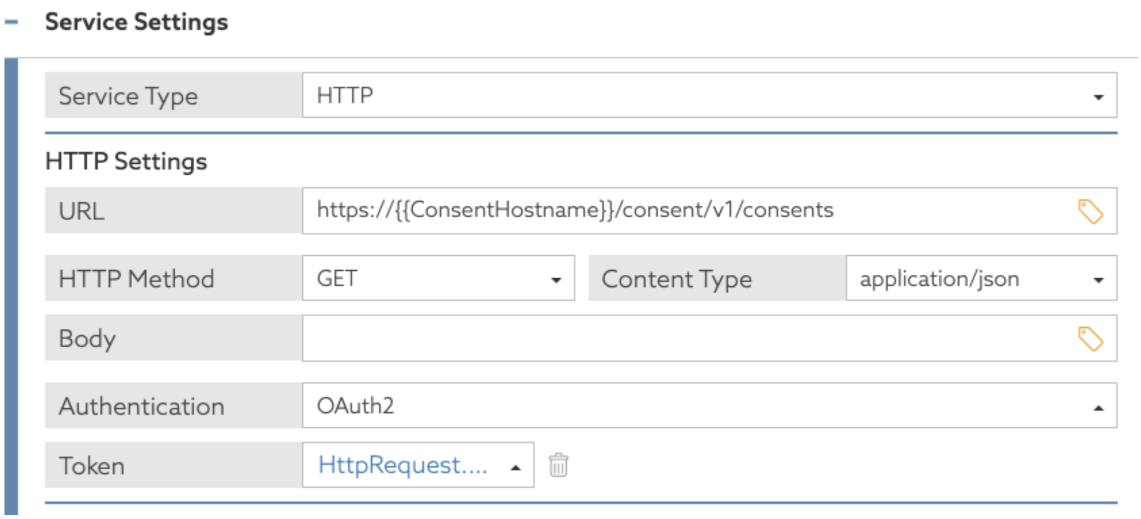
5. Start the Policy Administration GUI.

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

After you define the Consent API service in the Trust Framework, you can refer to the policy configuration key that you defined in the Policy Administration GUI configuration. To do this, you must first create an attribute in the Trust Framework to hold the policy configuration key value. Add an attribute with the following settings.

Property	Value
Name	ConsentHostname
Resolver Type	Configuration Key
Resolver Value	ConsentHostname

Now when you create a service in the Trust Framework, you can refer to this attribute using the `{{AttributeName}}` notation. For example, where the URL https://consent-us-east.example.com/consent/v1/consents is otherwise used, you would use the URL https://{{ConsentHostname}}/consent/v1/consents, as shown in the following image.



Service Settings

Service Type: HTTP

HTTP Settings

URL: https://{{ConsentHostname}}/consent/v1/consents

HTTP Method: GET

Content Type: application/json

Body:

Authentication: OAuth2

Token: HttpRequest....

Example: Configure a key store for a policy information provider

The policy engine supports the use of policy information providers (PIPs) to dynamically retrieve data from external services at runtime. You can configure a key store for a PIP in PingDataGovernance.

Some policy information providers might use MTLS, in which a client presents a client certificate to establish TLS communications with a server. In such cases, the policy engine can use a client certificate contained in a Java KeyStore (JKS) or PKCS12 key store. The key store details are then configured in an options file in the `keystores` section. A JKS key store file should use the extension `.jks`, while a PKCS12 key store file should use the extension `.p12`.

Example

Given a JKS key store named `my-client-cert-keystore.jks` with the password `password123` and a client certificate with the alias `my-cert`, create an options file with details about the key store.

To set up this key store, complete the following steps.

1. Make a copy of the default options file.

```
$ cp config/options.yml my-options.yml
```

2. Edit the new options file and define the key store details by adding an item under the `keystores` section.

```
keystores:
  - name: MyClientCertKeystore
    resource: /path/to/my-client-cert-keystore.jks
    password: password123
# Other options omitted for brevity...
```

3. Stop the Policy Administration GUI.

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

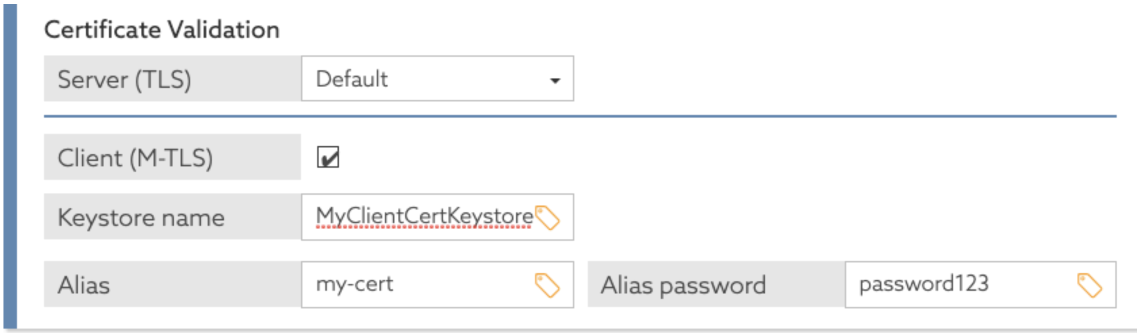
4. Run **`setup`** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret datagovernance \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

5. Start the Policy Administration GUI.

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

After you define the policy information provider in the Trust Framework, you can refer to the key store that you configured using the name `MyClientCertKeystore`.



Certificate Validation

Server (TLS)	Default
Client (M-TLS)	☑
Keystore name	MyClientCertKeystore
Alias	my-cert
Alias password	password123

Example: Configure a trust store for a policy information provider

The policy engine supports the use of policy information providers (PIPs) to dynamically retrieve data from external services at runtime. You can configure a trust store for a PIP in PingDataGovernance.

By default, the policy engine determines whether it should accept a PIP's server certificate using the Java Runtime Environment's (JRE's) default trust store, which contains public root certificates for common certificate authorities. If your PIP uses a server certificate issued by some other certificate authority, such as a private certificate authority operated by your organization, then you can provide a custom Java KeyStore (JKS) or PKCS12 trust store. Configure details about the trust store in an options file in the `truststores` section. A JKS trust store file should use the extension `.jks`, while a PKCS12 trust store file should use the extension `.p12`.

Example

Given a JKS trust store named `my-ca-truststore.jks` with the password `password123` and a trusted root certificate with the alias `my-ca`, create an options file with details about the trust store.

To set up this trust store, complete the following steps.

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

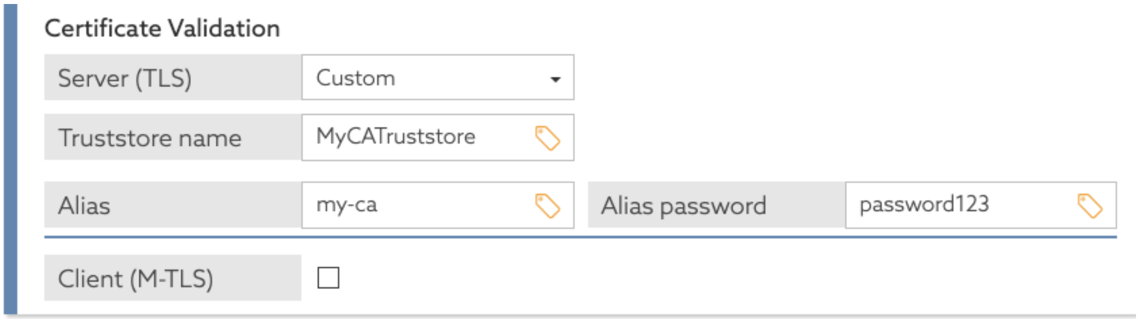
2. Edit the new options file to define the key store details by adding an item under the `truststores` section.

   ```
   truststores:
     - name: MyCATruststore
       resource: /path/to/my-ca-truststore.jks
       password: password123
   # Other options omitted for brevity...
   ```

3. Run **`setup`** using the `--optionsFile` argument. Customize all other options as needed.

   ```
   $ bin/setup demo \
     --adminUsername admin \
     --generateSelfSignedCertificate \
     --decisionPointSharedSecret datagovernance \
     --hostname <pap-hostname> \
     --port <pap-port> \
     --licenseKeyFile <path-to-license> \
     --optionsFile my-options.yml
   ```

After you define the policy information provider in the Trust Framework, you can see the trust store that you configured using the name `MyCATruststore`.



Certificate Validation

Server (TLS)	Custom		
Truststore name	MyCATruststore		
Alias	my-ca	Alias password	password123
Client (M-TLS)	□		

Example: Use environment variables

You do not have to hard-code values for policy configuration keys in an options file in the Policy Administration GUI configuration. You can specify values for policy configuration keys at runtime using environment variables.

To use environment variables, specify a policy configuration key value in the options file using the `${variableName}` notation, and then define the environment variable before starting the Policy Administration GUI.

Example: Set policy information provider host name using an environment variable

This example takes the scenario in *Example: Configure policy configuration keys* on page 214 and modifies it to specify the Consent API host name at runtime using an environment variable.

To specify the host name using an environment variable:

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

2. Edit the new options file and define a policy configuration key in the core section called `ConsentHostname`. Instead of hard-coding its value, specify a variable called `CONSENT_HOSTNAME`.

   ```
   core:
     ConsentHostname: ${CONSENT_HOSTNAME}
   # Other options omitted for brevity...
   ```

3. Stop the GUI server.

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

4. Run **`setup`** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

   ```
   $ bin/setup demo \
     --adminUsername admin \
     --generateSelfSignedCertificate \
     --decisionPointSharedSecret datagovernance \
     --hostname <pap-hostname> \
     --port <pap-port> \
     --licenseKeyFile <path-to-license> \
     --optionsFile my-options.yml
   ```

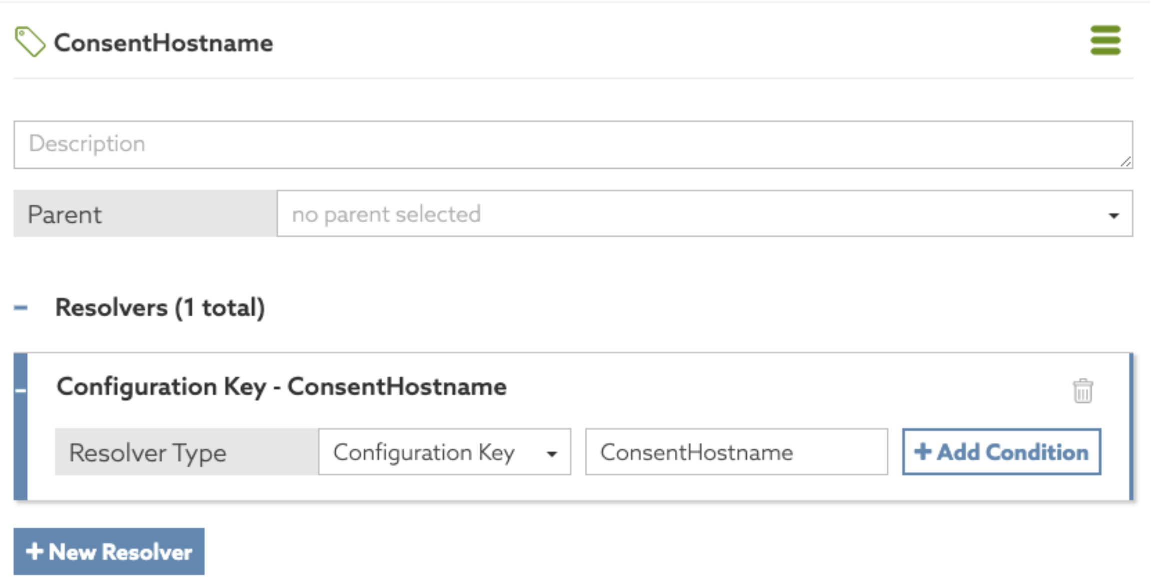
5. Set the value of the CONSENT_HOSTNAME environment variable and then start the server.

   ```
   $ export CONSENT_HOSTNAME=consent-us-east.example.com; bin/start-server
   ```

After you define the Consent API service in the Trust Framework, you can refer to the policy configuration key that you defined in the Policy Administration GUI configuration (ConsentHostName), which will use the environment variable that you also defined. You must first create an attribute in the Trust Framework to hold the policy configuration key value. To do so, add an attribute with the following settings.

Property	Value
Name	ConsentHostname
Resolver Type	Configuration Key
Resolver Value	ConsentHostname

The following image shows the attribute in the Policy Administration GUI.



ConsentHostname

Description

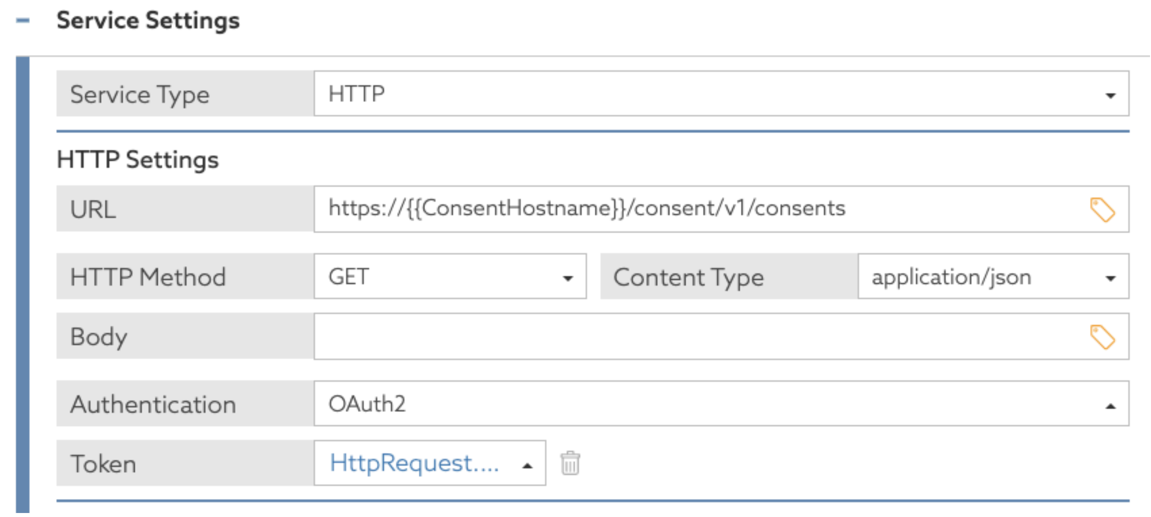
Parent: no parent selected

Resolvers (1 total)

Configuration Key - ConsentHostname

Resolver Type: Configuration Key | ConsentHostname

When you create a service in the Trust Framework, you can refer to this attribute using the `{{AttributeName}}` notation. For example, where the URL https://consent-us-east.example.com/consent/v1/consents would otherwise be used, use the URL https://{{ConsentHostname}}/consent/v1/consents. The following image shows service settings using the `{{AttributeName}}` notation.



Service Settings

Service Type: HTTP

HTTP Settings

URL: https://{{ConsentHostname}}/consent/v1/consents

HTTP Method: GET | Content Type: application/json

Body:

Authentication: OAuth2

Token: HttpRequest....

To set a different host name, redefine the CONSENT_HOSTNAME environment variable and restart the server.

```
$ bin/stop-server
$ export CONSENT_HOSTNAME=consent-us-west.example.com; bin/start-server
```

Example: Set trust store details using an environment variable

This example takes the scenario in *Example: Configure a trust store for a policy information provider* on page 217 and modifies it to specify the trust store password at runtime using an environment variable.

Given a Java KeyStore (JKS) trust store named `my-ca-truststore.jks` with the password `password123` and a trusted root certificate with the alias `my-ca`, create an options file with details about the trust store. Instead of hard-coding the trust store password, specify it as an environment variable.

To specify the password as an environment variable:

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

2. To edit the new options file and define the key store details, add an item in the `truststores` section. Specify the password value using the `${ENVIRONMENT_VARIABLE}` notation. Also, assign the password to a policy configuration key so it can be used in the Trust Framework.

   ```
   core:
     TrustStorePassword: ${TRUST_STORE_PASSWORD}
   truststores:
      - name: MyCATrustStore
        resource: /path/to/my-ca-truststore.jks
        # TRUST_STORE_PASSWORD is an environment variable
        password: ${TRUST_STORE_PASSWORD}
   # Other options omitted for brevity...
   ```

3. Stop the Policy Administration GUI.

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

4. Run **`setup`** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

   ```
   $ bin/setup demo \
     --adminUsername admin \
     --generateSelfSignedCertificate \
     --decisionPointSharedSecret datagovernance \
     --hostname <pap-hostname> \
     --port <pap-port> \
     --licenseKeyFile <path-to-license> \
     --optionsFile my-options.yml
   ```

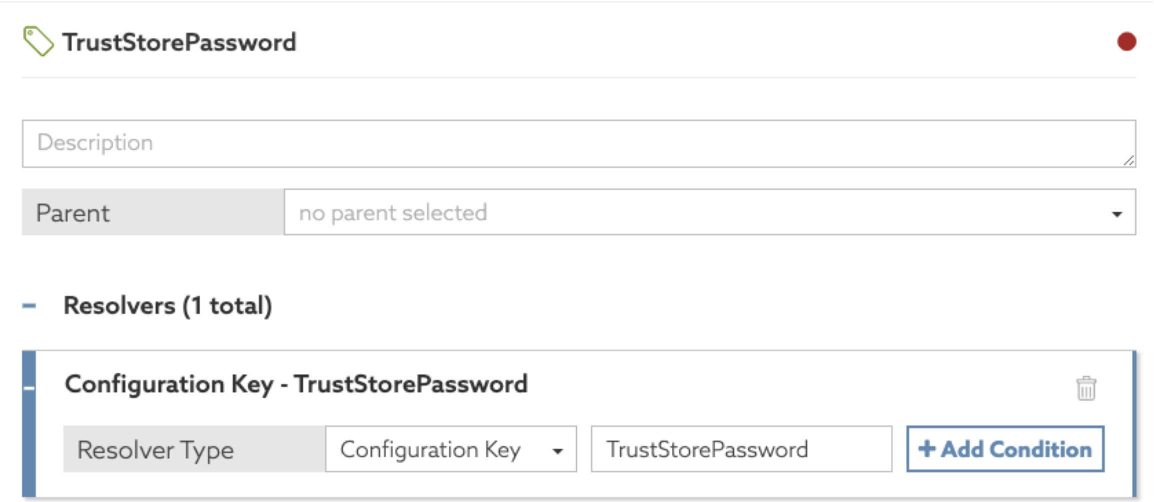
5. Set the value of the TRUST_STORE_PASSWORD environment variable and start the server.

   ```
   $ export TRUST_STORE_PASSWORD=password123; bin/start-server
   ```

The policy configuration key that you defined can be used in the Trust Framework. You must first create an attribute to hold the policy configuration key value. Add an attribute with the following settings.

Property	Value
Name	TrustStorePassword
Resolver Type	Configuration Key
Resolver Value	TrustStorePassword

The following image shows the attribute in the Policy Administration GUI.



TrustStorePassword

Description

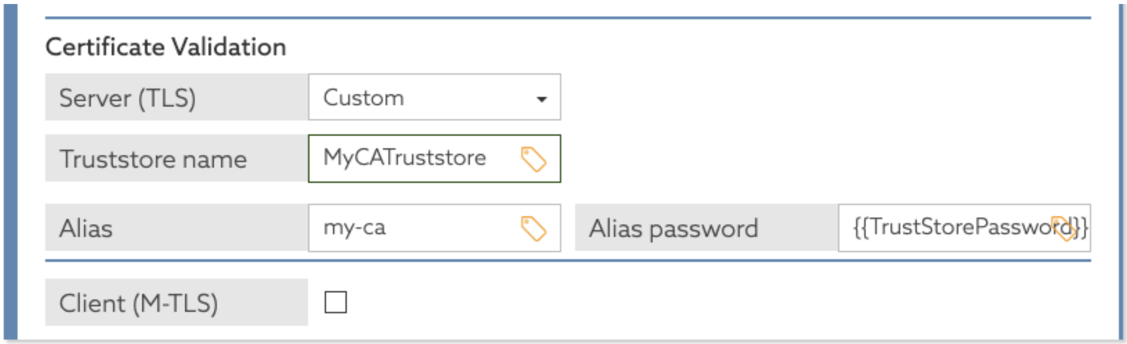
Parent: no parent selected

Resolvers (1 total)

Configuration Key - TrustStorePassword

Resolver Type: Configuration Key | TrustStorePassword

After you define the policy information provider in the Trust Framework, you can refer to the trust store password using the TrustStorePassword attribute.



Certificate Validation

Server (TLS): Custom

Truststore name: MyCATruststore

Alias: my-ca

Alias password: {{TrustStorePassword}}

Client (M-TLS): □

If you later use a trust store with a different password, you can redefine the TRUST_STORE_PASSWORD environment variable and restart the server.

```
$ bin/stop-server
$ export TRUST_STORE_PASSWORD=new-password; bin/start-server
```

Manage policy database credentials

The PingDataGovernance Policy Administration GUI stores policies within an H2 database file on the server. You can set the initial credentials and change them later.

This embedded H2 file, stored in the server root by default, contains two user accounts:

- An admin user

  Setup uses the admin user to perform database upgrades.
- An application user

  The server uses the application user to access the database at runtime.

Each user has its own credentials.

Warning:

If you change either of the default policy database credentials, you must pass the new credentials to **`setup`** when upgrading the server. Otherwise, the **`setup`** tool either cannot upgrade the policy database and fails (if neither default credentials work) or resets the changed credentials back to their defaults (if one of the credential pairs works). For more information about upgrades, see *Upgrading PingDataGovernance Server* on page 138.

Setting database credentials at initial setup

The **`setup`** tool applies credentials to the policy database. Also, this tool generates the `configuration.yml` file that configures the PingDataGovernance Policy Administration GUI.

About this task

Using **`setup`**, you can set credentials for both the admin user and the application user.

Alternatively, you can use environment variables to set credentials for just the application user.

Because this setup is an initial setup, the Policy Admin GUI is not running.

Steps

- Set credentials for both users (**`setup`**) or just the application user (environment variables).
    - Setting credentials for both the admin user and the application user with the **`setup`** tool.

      Include the following options and the credential values with **`setup`**:

        - **`--dbAdminUsername`**
        - **`--dbAdminPassword`**
        - **`--dbAppUsername`**
        - **`--dbAppPassword`**

      For example, the following command sets the policy database admin credentials to `adminuser` / `Passw0rd` and the application credentials to `appuser` / `S3cret`.

      ```
      bin/setup --dbAdminUsername adminuser \
        --dbAdminPassword Passw0rd \
        --dbAppUsername appuser \
        --dbAppPassword S3cret \
        --interactive
      ```

    - Setting credentials for just the application user with environment variables.

      You can set the application user credentials using UNIX environment variables. See *Starting PingDataGovernance Policy Administration GUI* on page 147.

      Using environment variables, you can avoid credentials showing up in process lists and command-line history.

      The following example sets the application user credentials to `app` / `S3cret`.

      ```
      env PING_DB_APP_USERNAME=app \
        PING_DB_APP_PASSWORD=S3cret \
        bin/setup
      ```

      Using environment variables at initial setup generates the `configuration.yml` file with the `app` / `S3cret` credentials instead of the default application credentials.

Changing database credentials

To change the policy database credentials after the initial setup, run the **`setup`** tool again.

About this task

> **Note:** Running the **`setup`** tool regenerates the `configuration.yml` file and regenerates any self-signed certificate keystore.

Steps

1. Stop the Policy Administration GUI.

   ```
   bin/stop-server
   ```

2. Run **`setup`** with the options desired from the following set and specify the new credentials. To change from the default credentials, run **`setup`** one time. To change from nondefault credentials, run **`setup`** combined by double ampersands (`&&`) with a second **`setup`**; in the first command, specify the current credentials for the admin user and the new credentials for the application user, and then in the second command, specify the new credentials for the admin user and the now-current credentials for the application user. See the examples.

   - **`--dbAdminUsername`**
   - **`--dbAdminPassword`**
   - **`--dbAppUsername`**
   - **`--dbAppPassword`**

   The first example changes the credentials for the admin and application accounts from their defaults to `admin` / `Passw0rd` and `app` / `S3cret`, respectively.

   ```
   setup --dbAdminUsername admin \
     --dbAdminPassword Passw0rd \
     --dbAppUsername app \
     --dbAppPassword S3cret \
     --interactive
   ```

   With the credentials no longer the defaults, to change the credentials, you need two **`setup`** commands. The first command uses the current admin credentials (`admin` / `Passw0rd`) and sets new application credentials (`app` and `F0cu5`). The second command then uses the newly set application credentials (`app` and `F0cu5`) to set new admin credentials (`admin` and `S3cure`).

   ```
   setup --dbAdminUsername admin \
     --dbAdminPassword Passw0rd \
     --dbAppUsername app \
     --dbAppPassword F0cu5 \
     --interactive \
     && setup --dbAdminUsername admin \
     --dbAdminPassword S3cure \
     --dbAppUsername app \
     --dbAppPassword F0cu5 \
     --interactive
   ```

3. Start the Policy Administration GUI.

   ```
   bin/start-server
   ```

Specifying database credentials when you start the GUI

You can override database credentials for the application account in the `configuration.yml` file when you start the GUI by using the UNIX environment variables PING_DB_APP_USER and PING_DB_APP_PASSWORD.

About this task

For more information about these and other UNIX environment variables you can use to override configuration settings, see *Starting PingDataGovernance Policy Administration GUI* on page 147.

Steps

1. Stop the Policy Administration GUI.

   ```
   bin/stop-server
   ```

2. Set the PING_DB_APP_USER and PING_DB_APP_PASSWORD variables and start the Policy Administration GUI.

Example

The following example starts the server with the overridden policy database application credentials `app` / `S3cret`. These environment variables override any values in `configuration.yml`.

```
env PING_DB_APP_USER=app \
  PING_DB_APP_PASSWORD=S3cret \
  bin/start-server
```

Docker: Setting the initial database credentials

When using a Docker image, set the database credentials using UNIX environment variables. Specify the environment variables as command-line options in the **`docker run`** command.

Steps

- In the **`docker run`** command, specify the desired following environment variables using the `--env` command-line option:
  - **`--dbAdminUsername`**
  - **`--dbAdminPassword`**
  - **`--dbAppUsername`**
  - **`--dbAppPassword`**

Example

This example initializes the policy database with the admin credentials `admin` / `Passw0rd` and the application credentials `app` / `S3cret`. Also, it uses the Ping DevOps image.

> ⓘ **Note:**
>
> Specify a separate volume to store the policy database to perform future upgrades. See *Docker and PingDataGovernance Policy Administration GUI installation* on page 136.

```
docker run
  --env PING_DB_ADMIN_USERNAME=admin \
  --env PING_DB_ADMIN_PASSWORD=Passw0rd \
  --env PING_DB_APP_USERNAME=app \
  --env PING_DB_APP_PASSWORD=S3cret \
```

```
    pingidentity/pingdatagovernancepap
```

Docker: Changing database credentials

When your Docker container users `/opt` to store the policy database on a separate volume, you can change the database credentials.

About this task

Given that you are changing the credentials, you already have a Docker container running with a mounted volume.

Steps

1. Stop the Docker container.
2. Start the Docker container. In the **`docker run`** command, specify the desired following environment variables using the `--env` command-line option:
    - **`--dbAdminUsername`**
    - **`--dbAdminPassword`**
    - **`--dbAppUsername`**
    - **`--dbAppPassword`**

    Also specify `-p`, `-d`, `--env-file`, `--volumes-from`, and `--env PING_H2_FILE`.

Example

For example, if you have a container named `pap81` with a mounted volume as shown in the example in *Docker and PingDataGovernance Policy Administration GUI installation* on page 136, the following command changes the credentials for the admin and application accounts from their default values to `admin` / `Passw0rd` and `app` / `S3cret`, respectively.

```
docker run -p 443:443 -d \
  --env-file ~/.pingidentity/devops \
  --volumes-from pap81  \
  --env PING_DB_ADMIN_USERNAME=admin \
  --env PING_DB_ADMIN_PASSWORD=Passw0rd \
  --env PING_DB_APP_USERNAME=app  \
  --env PING_DB_APP_PASSWORD=S3cret \
  --env PING_H2_FILE=/opt/shared/Symphonic \
  pingidentity/pingdatagovernancepap
```

Configuring SpEL Java classes for value processing

When you develop policies, you can use value processing to manipulate data that comes from attributes and services. One value processing option is to use the Spring Expression Language (SpEL). Because SpEL is so powerful, you might want to configure the Java classes available through SpEL to limit what users can do with it.

About this task

Use the optional `AttributeProcessing.SpEL.AllowedClasses` parameter in the `core` section of the options file to limit the Java classes available through SpEL.

Steps

1. Make a copy of the default options file.

    ```
    $ cp config/options.yml my-options.yml
    ```

2. Edit the new options file and define `AttributeProcessing.SpEL.AllowedClasses` in the `core` section.

   By default, the `AttributeProcessing.SpEL.AllowedClasses` parameter is not in the options file.

   If `AttributeProcessing.SpEL.AllowedClasses` is not in the options file, all classes except those in the fixed `deny-list` are available. The `deny-list` consists of these classes:

   ```
   "java.lang.*"
   "org.springframework.expression.spel.*"
   ```

   Note: The `java.lang.*` classes in `deny-list` exclude those in the `allow-list` defined next.

   If `AttributeProcessing.SpEL.AllowedClasses` is in the options file without a value, only classes in the fixed `allow-list` are available. The `allow-list` consists of these classes:

   ```
   java.lang.String,
   java.util.Date,
   java.util.UUID,
   java.lang.Integer,
   java.lang.Long,
   java.lang.Double,
   java.lang.Byte,
   java.lang.Math,
   java.lang.Boolean,
   java.time.LocalDate,
   java.time.LocalTime,
   java.time.LocalDateTime,
   java.time.ZonedDateTime,
   java.time.DayOfWeek,
   java.time.Instant,
   java.time.temporal.ChronoUnit,
   java.text.SimpleDateFormat,
   java.util.Collections,
   com.symphonicsoft.spelfunctions.RequestUtilsKt
   ```

   If `AttributeProcessing.SpEL.AllowedClasses` is in the options file with a value, all classes in `allow-list` and in the value are available. Consider the following example.

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

   That setting makes the classes in `allow-list` available in addition to making the `DataTimeFormatter` and `URLEncoder` classes available.

3. Stop the Policy Administration GUI.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.

   ```
   $ bin/setup demo \
    --adminUsername admin \
    --generateSelfSignedCertificate \
    --decisionPointSharedSecret <shared-secret> \
    --hostname <pap-hostname> \
    --port <pap-port> \
   ```

```
 --licenseKeyFile <path-to-license> \
 --optionsFile my-options.yml
```

5. Start the Policy Administration GUI.

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

Setting the request list length for Decision Visualizer

In the PingDataGovernance Policy Administration GUI, you can select **Policies** and then **Decision Visualizer** to view graphs of recent decisions.

About this task

The `RecentRequest.buffer.size` parameter in the configuration file determines the number of recent decisions to choose from. To configure the Policy Administration GUI to use a different value for this parameter, re-run the **`setup`** tool using an options file to generate a new configuration, as shown in the following steps.

Steps

1. Make a copy of the default options file.

```
$ cp config/options.yml my-options.yml
```

2. Edit the new options file and define `RecentRequest.buffer.size` in the `core` section.

   By default, the number of recent decisions is 20.

   To disable the feature, set the value to 0.

```
core:
  RecentRequest.buffer.size: 10
# Other options omitted for brevity...
```

3. Stop the Policy Administration GUI.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.

```
$ bin/setup demo \
 --adminUsername admin \
 --generateSelfSignedCertificate \
 --decisionPointSharedSecret <shared-secret> \
 --hostname <pap-hostname> \
 --port <pap-port> \
 --licenseKeyFile <path-to-license> \
 --optionsFile my-options.yml
```

5. Start the Policy Administration GUI.

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

Policy administration

You define policies for access-control using the PingDataGovernance Policy Administration GUI, which is powered by Symphonic®.

This section covers strategies for policy development and techniques to create environment-specific Trust Framework attributes to use in your policies.

About the Trust Framework

The Trust Framework defines all the entities that your organization can use to build policies. These entities include, for example, the HTTP request attributes that describe API requests protected by PingDataGovernance Server and the services that identify the REST APIs themselves.

To understand how PingDataGovernance Server uses the Trust Framework, you must understand how PingDataGovernance Server interacts with its policy engine, also called the policy decision point (PDP). In general, the flow is:

1. PingDataGovernance Server receives a SCIM 2.0 or API request and translates it to a *policy request*.
2. PingDataGovernance Server submits the policy request to the PDP for evaluation.
3. The PDP applies any matching policies to the policy request and then issues a policy decision.
4. PingDataGovernance Server uses the policy decision to determine how to proceed with the request, depending on the decision result (typically PERMIT or DENY) and any advices included with the decision.

Consider these simple examples.

- A policy decision with a DENY result could cause PingDataGovernance Server to reject a request because it originates from an untrusted IP address.
- A policy decision with the Exclude Attributes advice could cause PingDataGovernance Server to remove specific attributes from an API response because the requesting user lacks a necessary entitlement.

Each policy request that PingDataGovernance Server generates includes a specific set of attributes. These attributes vary based on the service being used. For more information, see the following topics:

- *API security gateway policy requests* on page 153
- *Sideband API policy requests* on page 168
- *SCIM policy requests* on page 183

Policy request structure is tightly coupled to the Trust Framework. If the Trust Framework entity definitions do not match the policy requests generated by PingDataGovernance Server, then PingDataGovernance Server does not function as expected. For this reason, your Trust Framework should always be based on the default policies included with the server installation package in the file `resource/policies/defaultPolicies.SNAPSHOT`.

For information about working with the Trust Framework to customize your organization's policies, see *Trust Framework* on page 302.

Trust Framework versions

The policy request structure used by PingDataGovernance Server is versioned so that it can evolve across releases of the server. You configure the version in the Policy Decision Service using the `trust-framework-version` property. PingDataGovernance Server always supports a minimum of two Trust Framework versions, the current (and preferred) Trust Framework version and the previous Trust Framework version.

When an instance of PingDataGovernance Server is first installed, the Trust Framework version is undefined. The server raises an alarm to indicate this condition and to provide instructions about how to set the preferred version.

You should explicitly set the version to the preferred version. For example, the following **dsconfig** command configures the Policy Decision Service to form policy requests using Trust Framework version v2.

```
dsconfig set-policy-decision-service-prop \
  --set trust-framework-version:v2
```

ⓘ **Tip:** When the Trust Framework version is set, add the configuration to the server profile that you use to deploy new server instances.

New releases of PingDataGovernance Server might introduce changes to the way that the server generates policy requests, potentially in ways that are not backward-compatible with the Trust Framework and policies used in a previous release. In these cases, PingDataGovernance Server will prefer the new Trust Framework version and raises an alarm with instructions to move to the new Trust Framework version. Existing policies will continue to work with the older Trust Framework version. However, the older Trust Framework version will be deprecated, so transitioning to the new Trust Framework version is imperative.

For more information about upgrading the Trust Framework version, see *Upgrading the Trust Framework and policies* on page 142.

Create policies in a development environment

During policy development, configure PingDataGovernance Server in external PDP mode where PingDataGovernance Server forwards all policy requests to the Policy Administration GUI, which acts as PingDataGovernance Server's policy decision point, or PDP.

Any policy changes made while using external PDP mode immediately take effect, allowing for rapid development and troubleshooting.

Develop policies in the PingDataGovernance Policy Administration GUI. To get started, see *Getting started with PingDataGovernance (tutorials)* on page 52 or *Loading a policy snapshot* on page 231.

ⓘ **Note:**

When developing new policies, begin by importing the `defaultPolicies.SNAPSHOT` file bundled with PingDataGovernance Server and using it as the basis for your own customizations. PingDataGovernance Server does not function as expected without many of the Trust Framework entities defined by this snapshot.

Example: Configure external PDP mode

To configure PingDataGovernance Server to use external PDP mode, use **dsconfig** or the Administrative Console to create a Policy External Server to represent the Policy Administration GUI, then assign the Policy External Server to the Policy Decision Service and set the PDP mode.

```
dsconfig create-external-server \
  --server-name "Policy Administration GUI" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:datagovernance" \
  --set "branch:Default Policies" \

dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:Policy Administration GUI"
```

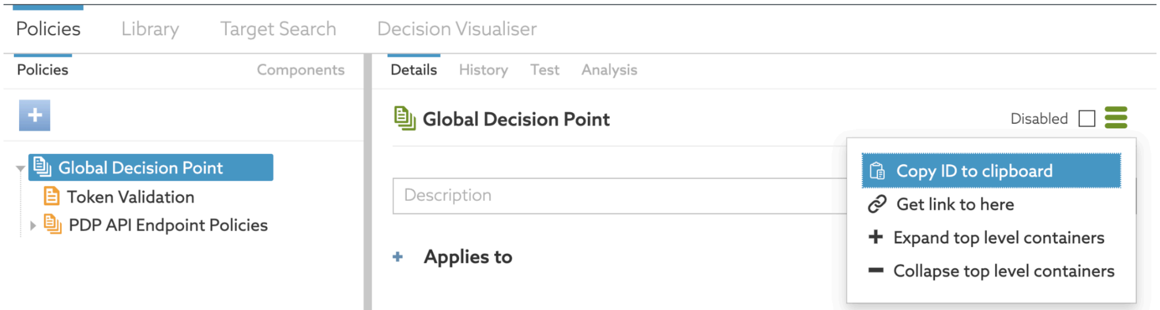
In this example, the shared-secret value corresponds to the decision point shared secret value chosen or generated while installing the Policy Administration GUI. The branch is the name of a policy branch in the

Policy Administration GUI, and the decision-node value is the ID of a node in the policy tree that will be considered first during policy processing.

To find a decision node:

1. In the Policy Administration GUI, go to **Policies**.
2. Select the node that you want to use as the root node.

   This is typically the top-level node of your policy tree.
3. Click the three-line icon and select **Copy ID to clipboard**.



Example: Change the active policy branch

The PingDataGovernance Policy Administration GUI can manage multiple sets of Trust Framework attributes and policies by storing data sets in different branches.

In a development environment, you might need to quickly reconfigure PingDataGovernance Server between policy branches.

To set up branch changes, you must first define a Policy External Server configuration for each branch. Then, you change a branch by changing the Policy Decision Service's `policy-server` property as needed.

Example

Assume that you have two policy branches in the Policy Administration GUI: `Stable Policies` and `Experimental Policies`. Each branch is represented in the PingDataGovernance Server configuration as a Policy External Server. During testing, you can switch back and forth between branches by updating the Policy Decision Service's `policy-server` property.

To change to the `Experimental Policies` branch, run this command.

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

To change back to the `Stable Policies` branch, run this command.

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

Default and example policies

A policy snapshot is a file that contains a complete Trust Framework and policy set.

A policy snapshot is also the data import format for a PingDataGovernance Policy Administration GUI. PingDataGovernance includes a number of default and example policy snapshot files, which are found in the folder `resource/policies`. The following table describes the available snapshot files.

Snapshot filename	Description
`defaultPolicies.SNAPSHOT`	The default Trust Framework for PingDataGovernance Server and a minimal set of policies. Always use this snapshot as the starting point for policy development.
`gatewayPolicyExample.SNAPSHOT`	An example policy set that demonstrates how to apply policies to an external REST API using PingDataGovernance Server as an API security gateway. Based on *Getting started with PingDataGovernance (tutorials)* on page 52.
`scimPolicyExample.SNAPSHOT`	An example policy set that demonstrates how to implement access token-based access control using the SCIM 2 REST API. Based on *Getting started with PingDataGovernance (tutorials)* on page 52.

Importing and exporting policies

PingDataGovernance supports two import and export file formats for Trust Framework and policy data.

The following table describes the snapshot and deployment package formats.

Format	Description
snapshot	Contains all Trust Framework and policy data for a policy branch in the Policy Administration GUI. A snapshot is used to load data into the Policy Administration GUI for development when using external PDP mode.
deployment package	An optimized data format that contains all policies under a specified root policy node and all Trust Framework entities used by those policies. A deployment package is used to load data into the PingDataGovernance Server when using embedded PDP mode.

The following sections describe how to import and export these files from the Policy Administration GUI.

Loading a policy snapshot

To import a policy snapshot into the Policy Administration GUI for policy development, complete the following steps.

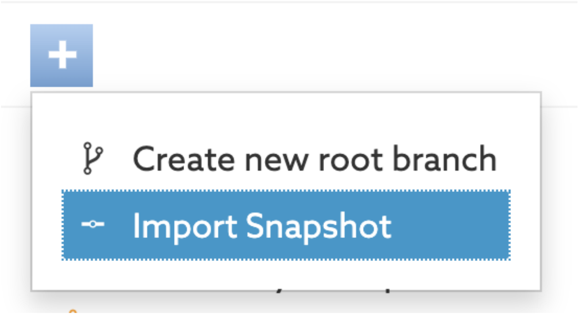
About this task

These steps create a new policy branch with the Trust Framework and policies of the provided snapshot.

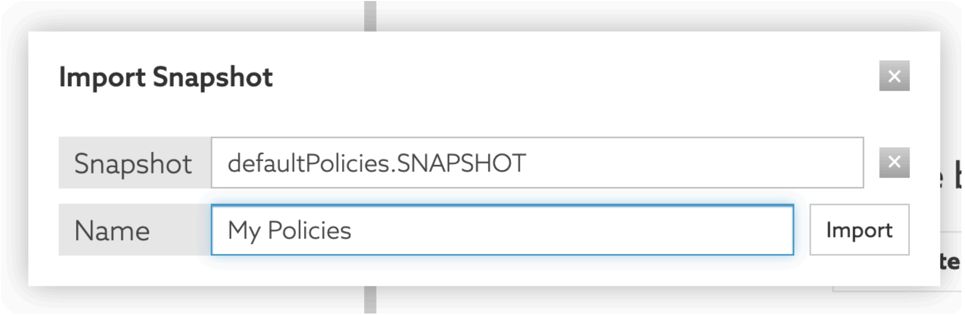
Steps

1. Go to the **Branch Manager** section.
2. Select the **Version Control** tab.

3. From the **+** menu, select **Import Snapshot**.



4. Select a snapshot file and provide a name for your policy branch.



Snapshot: defaultPolicies.SNAPSHOT

Name: My Policies

5. Optionally, click **Commit New Changes** to commit the initial state of the policy branch.

Exporting a policy snapshot

You can export a policy snapshot from the Policy Administration GUI.

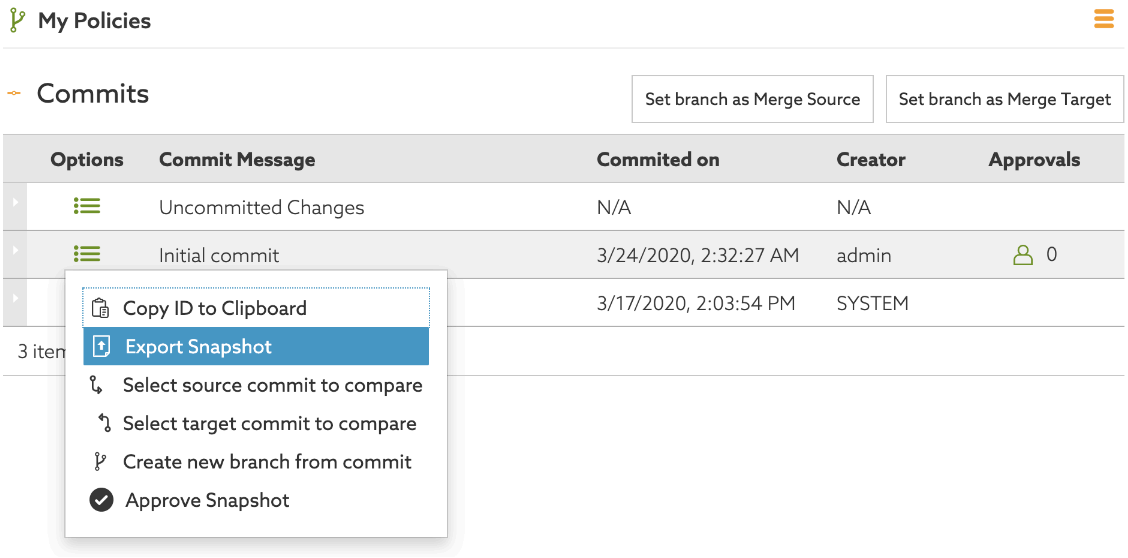
About this task

You can then import it into a different Policy Administration GUI or use it as the basis to create a deployment package to be loaded in the PingDataGovernance Server.

Steps

1. Go to the **Branch Manager** section.
2. Select the **Version Control** tab.
3. Choose the commit message corresponding to the version of the branch that you want to export and click the menu icon to the left of the commit message.

4. Select **Export Snapshot**.


My Policies
Commits
Set branch as Merge Source
Set branch as Merge Target
Copy ID to Clipboard
Export Snapshot
Select source commit to compare
Select target commit to compare
Create new branch from commit
Approve Snapshot
3 item

Options	Commit Message	Commited on	Creator	Approvals
	Uncommitted Changes	N/A	N/A	
	Initial commit	3/24/2020, 2:32:27 AM	admin	0
		3/17/2020, 2:03:54 PM	SYSTEM	

5. Provide a snapshot filename and click **Export**.

Results

The snapshot file is downloaded to your computer.

Exporting a deployment package

When you have completed development and testing of your policies, you can export your Trust Framework and policies to a deployment package for use in embedded PDP mode.

Steps

1. Export a snapshot. See *Exporting a policy snapshot* on page 232.
2. Go to the **Branch Manager** section.
3. Select the **Deployment Packages** tab.
4. Click the **+** icon.
5. Replace **Untitled** with a name for your deployment package.
6. Select a policy branch.
7. Select a commit.
8. Click **Create Package**.
9. Click **Export Package**

Results

The deployment package is downloaded to your computer.

Use policies in a production environment

You can configure PingDataGovernance Server in embedded policy decision point (PDP) mode in preproduction and production environments

When configured to use embedded PDP mode, a policy file, called a deployment package, is exported from the Policy Administration GUI and loaded into PingDataGovernance Server's internal policy engine, which then handles all policy requests.

Because embedded PDP mode does not require PingDataGovernance Server to call out to an external server, it is considerably more performant than external PDP mode. However, any policy changes require a new deployment package to be exported and loaded, so embedded PDP mode is generally unsuited for rapid policy development.

Configure embedded PDP mode

To configure PingDataGovernance Server to use embedded PDP mode, assign a deployment package to the Policy Decision Service and set the PDP mode.

```
dsconfig set-policy-decision-service-prop \
  --set pdp-mode:embedded \
  --set "deployment-package</path/to/my-deployment-package.SDP"
```

In this example, the `deployment-package` value is the full path to a deployment package file.

To create a deployment package, see *Exporting a deployment package* on page 233.

Example: Define policy configuration keys

A policy configuration key is an arbitrary key/value pair that you can reference by name in the policy Trust Framework.

When using embedded PDP mode, policy configuration keys are stored in the PingDataGovernance Server configuration, and the server provides the policy configuration key values to the policy engine at runtime. This allows the Trust Framework to refer to data such as hostnames and credentials without needing those values to be hard-coded in the Trust Framework.

Note:

Policy configuration key values are stored in encrypted form in the PingDataGovernance Server configuration, so they are suitable for storing sensitive values such as server credentials.

Use **`dsconfig`** or the Administrative Console to define policy configuration keys. If using the Administrative Console, you can find policy configuration keys in the Policy Decision Service configuration.

The following example shows how to create a policy configuration key named `ConsentServiceBaseUri` with the value `https://example.com/consent/v1`.

```
dsconfig create-policy-configuration-key \
 --key-name ConsentServiceBaseUri \
 --set policy-configuration-value:https://example.com/consent/v1
```

To learn how to use a policy configuration key in the Trust Framework, see *Environment-specific Trust Framework attributes* on page 236.

Example: Define a policy information provider key store for MTLS

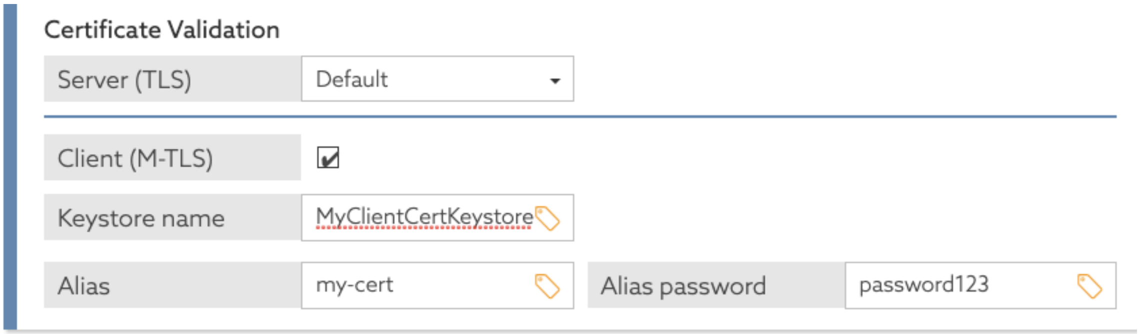
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. In these cases, the policy engine can use a client certificate contained in a Java KeyStore (JKS) or PKCS12 key store.

When using embedded PDP mode, the key store containing the client certificate is represented in the PingDataGovernance Server configuration as a Key Manager Provider, which is then assigned to the Policy Decision Service.

The following example creates a Key Manager Provider named `MyClientCertKeystore` and makes it available to the policy engine.

```
dsconfig create-key-manager-provider \
 --provider-name MyClientCertKeystore \
 --type file-based \
 --set enabled:true \
 --set key-store-file:<full path to a key store> \
 --set key-store-type:JKS \
 --set key-store-pin:<key store password>
dsconfig set-policy-decision-service-prop \
 --set service-key-store:MyClientCertKeystore
```

When you define the PIP in the Trust Framework, you can refer to the key store that you configured, using the name `MyClientCertKeystore`.



Certificate Validation

Server (TLS): Default

Client (M-TLS): ☑

Keystore name: MyClientCertKeystore

Alias: my-cert

Alias password: password123

Example: Define a policy information provider trust store

For a policy information provider (PIP), you can use the Java Runtime Environment (JRE)'s default trust store or you can provide a custom Java KeyStore (JKS) or PKCS12 trust store.

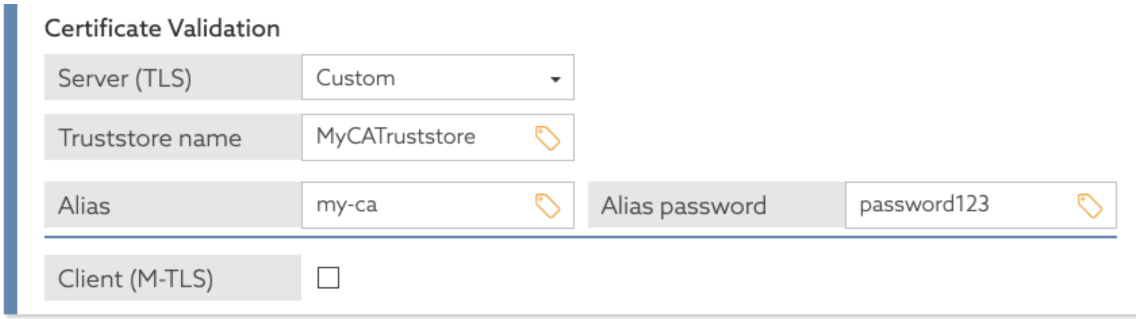
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. By default, the policy engine determines whether it should accept a PIP's server certificate using the Java Runtime Environment (JRE)'s default trust store, which contains public root certificates for common certificate authorities. However, if your PIP uses a server certificate issued by some other certificate authority, for example, a private certificate authority operated by your organization, then you can provide a custom Java KeyStore (JKS) or PKCS12 trust store.

When using embedded PDP mode, the trust store containing the client certificate is represented in the PingDataGovernance Server configuration as a Trust Manager Provider, which is then assigned to the Policy Decision Service.

The following example creates a Trust Manager Provider named `MyCATruststore` and makes it available to the policy engine.

```
dsconfig create-trust-manager-provider \
 --provider-name MyCATruststore \
 --type file-based \
 --set enabled:true \
 --set trust-store-file:<full path to a trust store> \
 --set trust-store-type:JKS
dsconfig set-policy-decision-service-prop \
 --set service-trust-store:MyCATruststore
```

When you define the policy information provider in the Trust Framework, you can refer to the trust store that you configured using the name `MyCATruststore`.



Certificate Validation

Server (TLS): Custom

Truststore name: MyCATruststore

Alias: my-ca

Alias password: password123

Client (M-TLS) □

Environment-specific Trust Framework attributes

With 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.

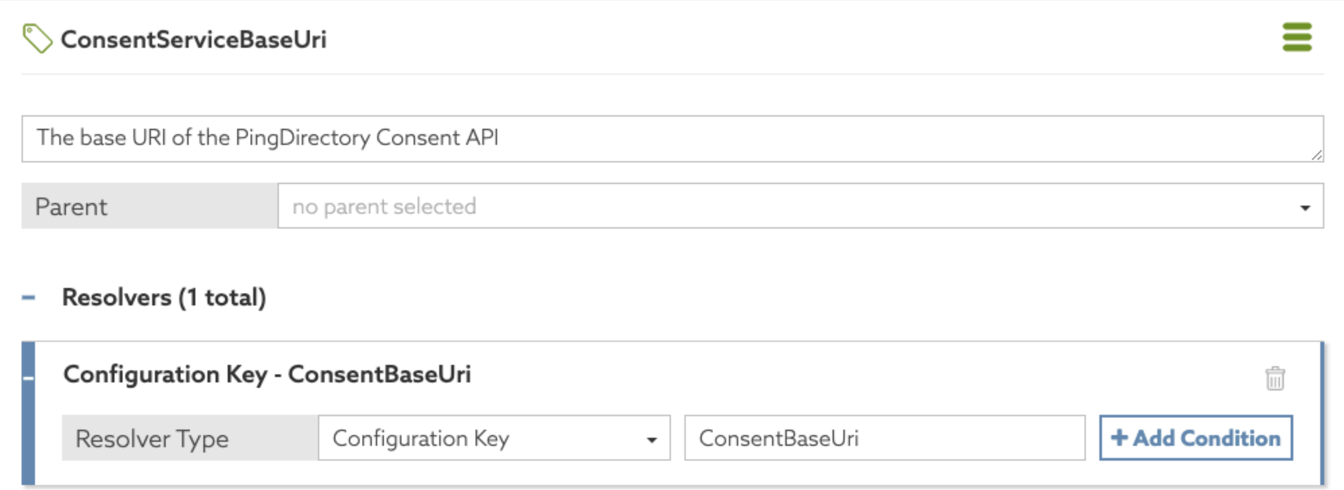
For example, you might use a Trust Framework service to retrieve a user's consent from the PingDirectory Consent API. This service depends on the URL of the Consent API, the username and password that are used for authentication, and other items that vary between development, preproduction, and production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

About policy configuration keys

To avoid hard-coding values such as URLs, usernames, or passwords, Trust Framework attributes can refer to policy configuration keys, which are key/value pairs defined outside of the Trust Framework and provided to the policy engine at runtime.

To define a Trust Framework attribute that uses a policy configuration key, configure the attribute with a **Configuration Key** resolver and the name of the policy configuration key.

For example, in the following image, an attribute called `ConsentServiceBaseUri` is configured to use a policy configuration key called `ConsentBaseUri`.



ConsentServiceBaseUri

The base URI of the PingDirectory Consent API

Parent: no parent selected

Resolvers (1 total)

Configuration Key - ConsentBaseUri

Resolver Type: Configuration Key

ConsentBaseUri

The means by which policy configuration keys are provided to the policy engine differ based on whether the PingDataGovernance Server is configured to use external PDP mode or embedded PDP mode, as shown in the following table.

Mode	**Where to define policy configuration keys**
External PDP mode	An options file and run the Policy Administration GUI's **`setup`** tool. See *Example: Configure policy configuration keys* on page 214.
Embedded PDP mode	The PingDataGovernance Server configuration. See *Example: Define policy configuration keys* on page 234.

Example
In this example, you define a policy information provider (PIP) in the Trust Framework so that various properties needed to connect to the PIP can be changed from those needed for a development environment to those needed for a preproduction environment.

You can complete the PIP definition without needing to update the Trust Framework.

Define a policy information provider for the PingDirectory Consent API that uses the following policy configuration keys:

Policy configuration key	**Description**
`ConsentBaseUri`	The base URL to use when making requests to the Consent API.
`ConsentUsername`	The username for a privileged Consent API account.
`ConsentPassword`	The password for a privileged Consent API account.

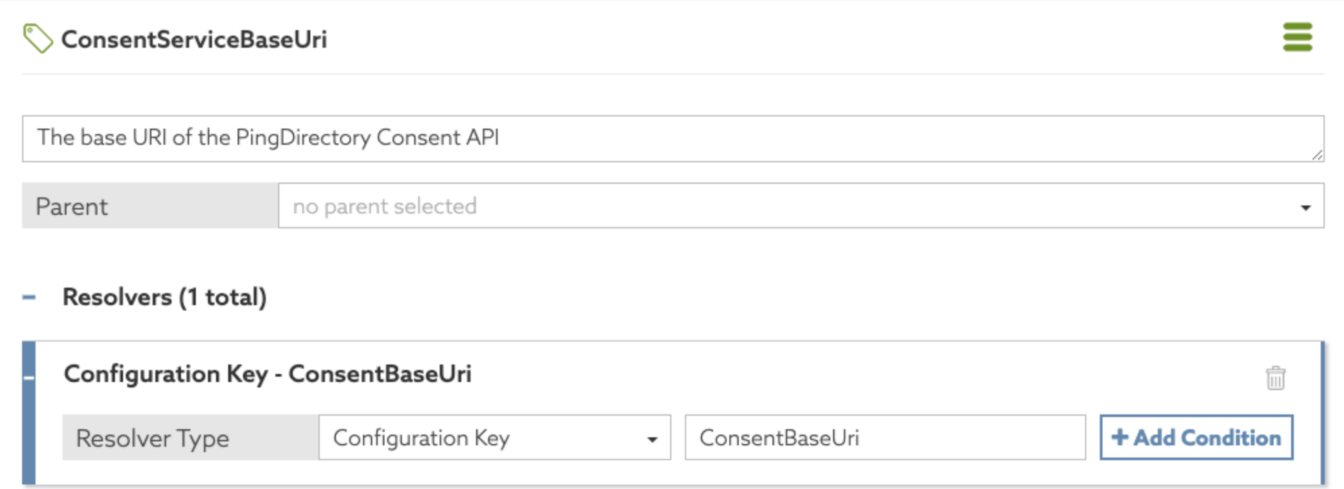
Define the policy information provider in the Trust Framework

Complete the following steps to define the policy information provider (PIP).

Steps

1. Define an attribute in the Trust Framework for the Consent API's base HTTPS URL.
   a. Go to **Trust Framework** and then click **Attributes**.
   b. Add a new attribute.
      1. Name the attribute `ConsentServiceBaseUri`.
      2. Add a resolver.
      3. Set the **Resolver type** to **Configuration Key**.
      4. Set the Resolver value to `ConsentBaseUri`.
      5. Save the attribute.

   The following image shows the attribute configuration.



ConsentServiceBaseUri

The base URI of the PingDirectory Consent API

Parent: no parent selected

Resolvers (1 total)

Configuration Key - ConsentBaseUri

Resolver Type: Configuration Key — ConsentBaseUri

2. Repeat the previous steps for `ConsentUsername` and `ConsentPassword`.

   When complete, you should have defined the following attributes.

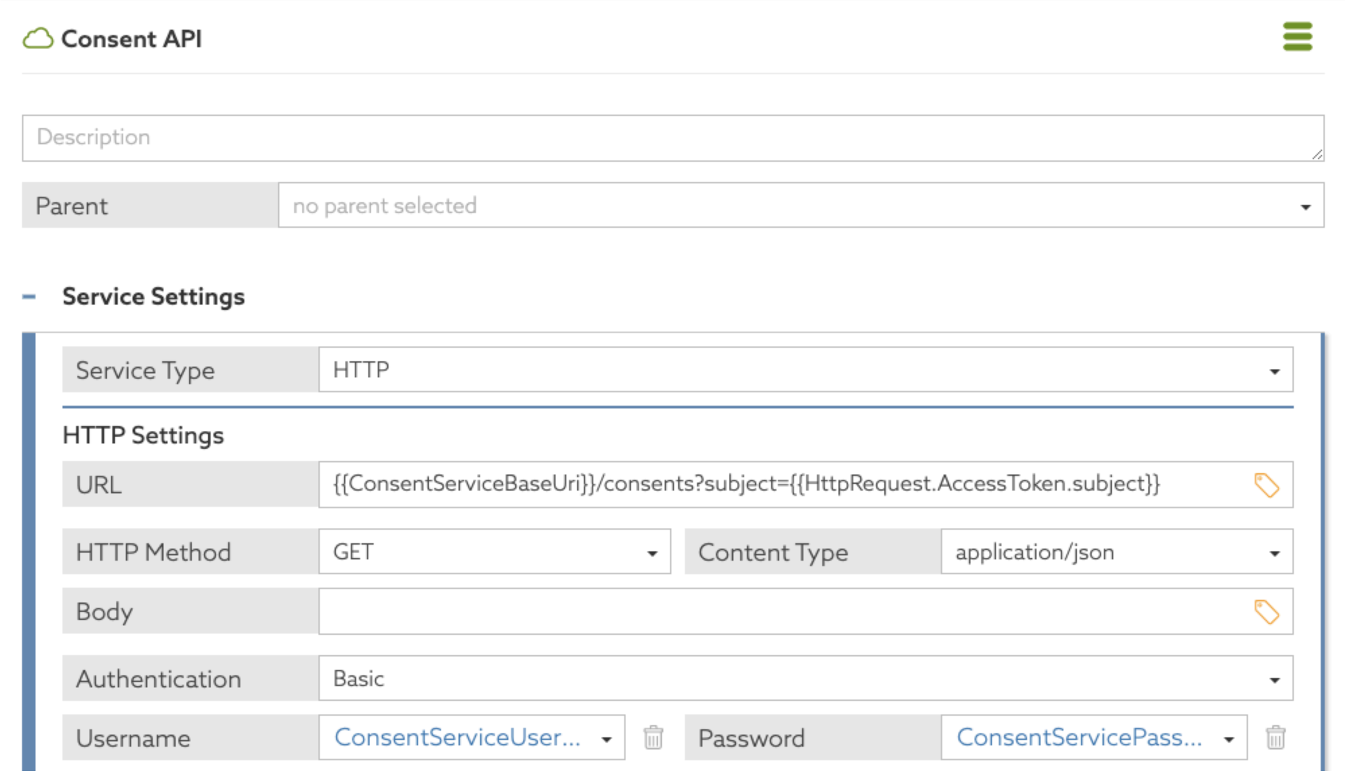
Attribute name	Policy configuration key name
`ConsentServiceBaseUri`	`ConsentBaseUri`
`ConsentServiceUsername`	`ConsentUsername`
`ConsentServicePassword`	`ConsentPassword`

ⓘ **Note:**

Both the attribute names and the policy configuration key names that you use are arbitrary, and you can use any names that you like. For the sake of this example, attribute names do not match configuration key names, but they do not need to differ.

3. Define the policy information provider using the attributes that you just defined.
   a. Go to **Trust Framework** and then **Services**.
   b. Add a new service.
      1. Name the service Consent API.
      2. Leave the **Parent** value blank. If a value is already present, clear it.
      3. Set **Service Type** to HTTP.
      4. Set the **URL** to {{ConsentServiceBaseUri}}/consents?subject={{HttpRequest.AccessToken.subject}}.
      5. Set **Authentication** to Basic.
      6. For **Username**, select the attribute `ConsentServiceUsername`.
      7. For **Password**, select the attribute `ConsentServicePassword`.
   c. Save the new service.

   The following image shows the attributes being used.



Consent API

Description

Parent: no parent selected

− Service Settings

Service Type: HTTP

HTTP Settings

URL: {{ConsentServiceBaseUri}}/consents?subject={{HttpRequest.AccessToken.subject}}

HTTP Method: GET

Content Type: application/json

Body:

Authentication: Basic

Username: ConsentServiceUser...

Password: ConsentServicePass...

   You can use the new Consent API policy information provider to build policies.

Define policy configuration keys in a development environment

Before you can use any policies that you developed with the Consent API policy information provider (PIP), you must configure the Policy Administration GUI to provide values for the PIP’s base URL, username, and password.

About this task

To configure the Policy Administration GUI to provide these values, re-run the **`setup`** tool using an options file to generate a new configuration, as shown in the following steps.

Steps

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

2. Edit the new options file and define the policy configuration keys in the `core` section.

   ```
   core:
     ConsentBaseUri: https://consent-us-east.example.com/consent/v1
     ConsentUsername: cn=consent admin
     ConsentPassword: Passw0rd123
   # Other options omitted for brevity...
   ```

3. Stop the Policy Administration GUI.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.

   ```
   $ bin/setup demo \
    --adminUsername admin \
    --generateSelfSignedCertificate \
    --decisionPointSharedSecret datagovernance \
    --hostname <pap-hostname> \
    --port <pap-port> \
    --licenseKeyFile <path-to-license> \
    --optionsFile my-options.yml
   ```

5. Start the Policy Administration GUI.

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

Define policy configuration keys in a preproduction environment
Do not use the Policy Administration GUI in a pre-production or production environment. Define policy configuration keys in the PingDataGovernance Server configuration.

About this task

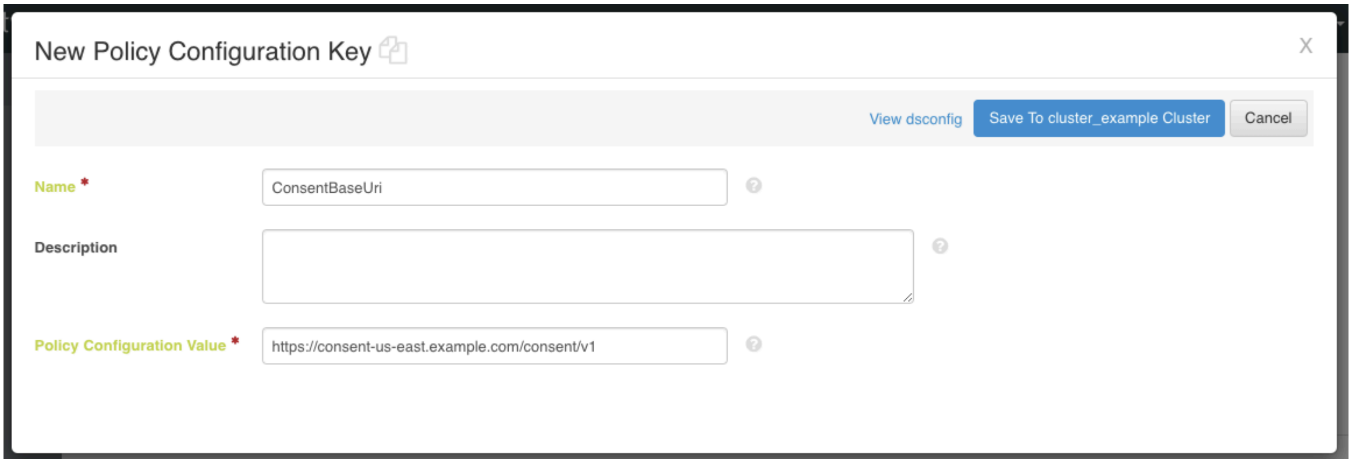
To define policy configuration keys, use either **`dsconfig`** or the Administrative Console, as shown in the following steps.

Steps

1. In the Administrative Console, under **Authorization and Policies**, click **Policy Decision Service**.

2. Click **New Policy Configuration Key**.
   a. For **Name**, enter `ConsentBaseUri`.
   b. For **Policy Configuration Value**, type the base URI. For example, `https://consent-us-east.example.com/consent/v1`.

   The following image shows the window.



New Policy Configuration Key

Name *: ConsentBaseUri

Description:

Policy Configuration Value *: https://consent-us-east.example.com/consent/v1

3. Save the policy configuration key.
4. Repeat the previous steps for the policy configuration keys `ConsentUsername` and `ConsentPassword`.

Make a user's profile available in policies

In a policy, you might need to make a decision based on something about the requesting identity, meaning the access token subject or token owner. PingDataGovernance can automatically look up the token owner's attributes and provide them in the policy request using a token resource lookup method.

Configuring a token resource lookup method

PingDataGovernance provides built-in support for retrieving token owner data using *SCIM token resource lookup methods* on page 257. Using a SCIM token resource lookup method requires a SCIM resource type to be configured, along with its prerequisite configuration objects. For information about SCIM configuration, such as SCIM resource types, store adapters, load-balancing algorithms, and LDAP external servers, see *SCIM configuration basics* on page 179.

For examples that show how to set up a token resource lookup method, see:

- *Configuring the PingDataGovernance OAuth subject search*
- *Access token validation* on page 175
- *SCIM token resource lookup methods* on page 257

Using user profile data in policies

When processing an incoming HTTP request, PingDataGovernance Server invokes any applicable access token validators to parse the request's access token. If an access token validator successfully validates the access token, it then invokes any related token resource lookup methods. If a token resource lookup method succeeds in retrieving the attributes for the token owner, then PingDataGovernance Server includes a `TokenOwner` attribute with the policy request. The contents of the `TokenOwner` attribute are a JSON object containing the user profile.

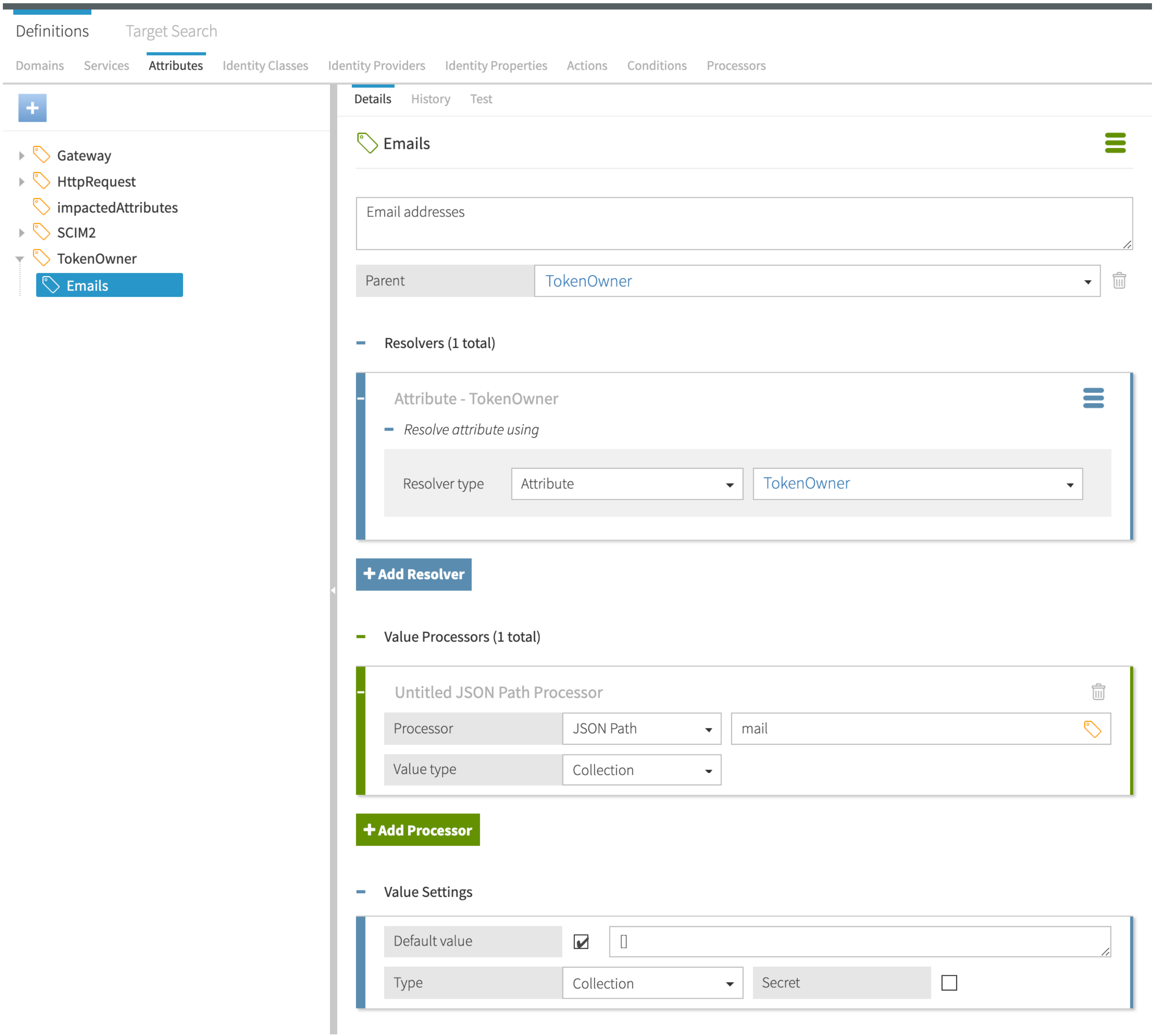
The exact structure of the `TokenOwner` attribute varies from deployment to deployment. When using a SCIM token resource lookup method, the contents of the `TokenOwner` attribute are a SCIM resource using the schema of the SCIM resource type configured for the token resource lookup method, exactly as if the resource had been retrieved via an HTTP GET without policy restrictions. For example, for a `pass-`

`through` SCIM resource type for the LDAP inetOrgPerson object class, a `TokenOwner` value might look like the following.

```
{
    "cn": [
        "Mark E. Smith"
    ],
    "employeeNumber": "1",
    "entryDN": "uid=mark.e.smith,ou=people,dc=example,dc=com",
    "entryUUID": "8ac3d8b5-4f17-33fa-a4b4-854599ed9a89",
    "givenName": [
        "Mark"
    ],
    "id": "8ac3d8b5-4f17-33fa-a4b4-854599ed9a89",
    "initials": [
        "MES"
    ],
    "l": [
        "Manchester"
    ],
    "mail": [
        "mark.e.smith@example.com"
    ],
    "meta": {
        "location": "https://example.com/scim/v2/Users/8ac3d8b5-4f17-33fa-
a4b4-854599ed9a89",
"resourceType": "Users"
    },
    "mobile": [
        "+44 161 872 37676"
    ],
    "modifyTimestamp": "2020-06-03T03:56:54.168Z",
    "objectClass": [
        "top",
        "person",
        "organizationalPerson",
        "inetOrgPerson"
    ],
    "schemas": [
        "urn:pingidentity:schemas:store:2.0:UserStoreAdapter"
    ],
    "sn": [
        "Smith"
    ],
    "uid": [
        "mark.e.smith"
    ]
}
```

The default Trust Framework includes a `TokenOwner` attribute as an empty JSON object. If you need to use a user profile attribute from a policy, add the attribute as a child of `TokenOwner` in the Trust Framework.

For example, the SCIM user profile shown above uses the `mail` attribute to store a user's email addresses. To make policy decisions involving the token owner's email address, you can add an `Emails` attribute under `TokenOwner` in the PingDataGovernance Policy Administration GUI, as shown in the following Trust Framework image.



Emails

Email addresses

Parent: TokenOwner

Resolvers (1 total)

Attribute - TokenOwner

Resolve attribute using

Resolver type: Attribute | TokenOwner

Value Processors (1 total)

Untitled JSON Path Processor

Processor: JSON Path | mail

Value type: Collection

Value Settings

Default value: ☑ []

Type: Collection

Secret: □

Advice types

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 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.

You can add an advice directly to a single policy or rule, or add an advice in **Components** for use with multiple policies or rules. Advices possess the following significant properties.

Advice properties and descriptions

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 `Permit` or `Deny`, 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 supports the following advice types:

- Add Filter
- Combine SCIM Search Authorizations
- Denied Reason
- Exclude Attributes
- Filter Response
- Include Attributes
- Modify Attributes
- Modify Headers
- Modify Query
- Regex Replace Attributes

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

ⓘ **Note:**

Many advice types let you use the *JSONPath* expression language to specify JSON field paths. To experiment with JSONPath, use the *Jayway JSONPath Evaluator* tool.

Add Filter

Use `add-filter` to add administrator-required filters to System for Cross-domain Identity Management (SCIM) search queries.

Applicable to	SCIM.

Additional information	The Add Filter advice places restrictions on the resources 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 separated by `AND` or `OR`. If the policy result returns multiple instances of Add Filter advice, they are `AND`ed together to form a single filter that passes 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.

Combine SCIM Search Authorizations

Use `combine-scim-search-authorizations` to optimize policy processing for System for Cross-domain Identity Management (SCIM) search responses.

Applicable to	SCIM.
Additional information	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 you use this advice type, the current SCIM search result set is processed 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 the policy result returns 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 187.

Denied Reason

Use `denied-reason` to allow a policy writer to provide an error message that contains the reason for denying a request.

Applicable to	DENY decisions.
Additional information	The payload for Denied Reason advice is a JSON object string with the following fields: ▪ `status` – Contains the HTTP status code returned to the client. If this field is absent, the default status is 403 Forbidden. ▪ `message` – Contains a short error message returned to the client. ▪ `detail` (optional) – Contains additional, more detailed error information. The following example shows a possible response for a request made with insufficient scope `{"status":403, "message":"insufficient_scope", "detail":"Requested operation not allowed by the granted OAuth scopes."}`

Exclude Attributes

Use `exclude-attributes` to specify the attributes to exclude from a JSON response.

Applicable to	PERMIT decisions, although you cannot apply Exclude Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.

Additional information	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 being authorized. Each JSONPath can select multiple attributes in the object. The portions of the response that a JSONPath selects are removed before sending the response to the client. 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 246.

Filter Response

Use `filter-response` to direct PingDataGovernance Server to invoke policy iteratively over each item of a JSON array contained within an API response.

Applicable to	PERMIT decisions from Gateway, although you cannot apply Filter Response advice directly to a System for Cross-domain Identity Management (SCIM) search. However, the SCIM service performs similar processing automatically when it handles a search result. For every candidate resource in a search result, the SCIM service makes a policy request for the resource with an Action value of retrieve.

Additional information

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

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

Field	Required	Description
`Path`	Yes	JSONPath to an array within the API's response body. The advice implementation iterates over the nodes in this array and makes a policy request for each node.
`Action`	No	Value to pass as the `action` 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 `service` 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 `Path` field. On each subsequent policy request, the contents of a single array element pass 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

Use `include-attributes` to limit the attributes that a JSON response can return.

Applicable to	PERMIT decisions, although you cannot apply Include Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	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 being authorized. The response includes only the portions that one of the JSONPaths selects. When a single JSONPath represents multiple attributes, the response includes all of them. If the policy result returns multiple instances of Include Attributes advice, the response includes the union of all selected attributes. For more information about the processing of SCIM searches, see *Filter Response* on page 246.

Modify Attributes

Use modify-attributes to modify the values of attributes in the JSON request or response.

Applicable to	All, although you cannot apply the Modify Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	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 that selects the attribute to modify, and the value is the new value to use for the selected attribute. The value can be any valid JSON value, including a complex value like an object or array.

Modify Headers

Use modify-headers to modify the values of request headers before PingDataGovernance sends them to the upstream server or to modify the values of response headers before PingDataGovernance returns them to the client.

Applicable to	All, although you cannot apply the Modify Headers advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	The payload for this advice is a JSON object. The keys are the names of the headers to set, and the values are the new values of the headers. A value can be: ▪ Null, which removes the header ▪ A string, which sets the header to that value ▪ An array of strings, which sets the header to all of the string values If the header already exists, PingDataGovernance overwrites it. If the header does not exist, PingDataGovernance adds it (unless the value is null). If a payload value is an array of strings: ▪ Given a header that supports multiple values, such as Accept, PingDataGovernance repeats the header for each string in the array. ▪ Given a header that does not support multiple values, such as Content-Type, PingDataGovernance sends the last string in the array.

Modify Query

Use modify-query to modify the query string of the request sent to the API server.

Applicable to	All.

Additional information	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`, you can set a policy 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`.

Regex Replace Attributes

Use `regex-replace-attributes` to specify a regex to search for attributes in a request or response body and replace their values with a regex replacement string.

Applicable to	All, although you cannot apply the Regex Replace Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.

Additional information

The payload for this advice is either a JSON object or an array of JSON objects. Each object represents a single replacement operation and has up to four keys.

Key	Description
`"regex"`	Required. Represents the regular expression to use to find the attribute values to replace.
`"replace"`	Required. Represents the regex replacement string to use to replace the attribute values with a new value.
`"path"`	Optional. Is a JSONPath expression that represents the nodes to start searching under.
`"flags"`	Optional. Is a string that contains the regex flags to use. Recognized flags are: ▪ `"i"` Performs case-insensitive matching. ▪ `"l"` Treats the `"regex"` value as a literal string. ▪ `"c"` Performs "canonical equivalence" matching. You can combine flags. For example: `"il"`

PingDataGovernance replaces any portion of the attribute value that matches the regular expression in the `"regex"` value in accordance with the `"replace"` replacement string. If multiple substrings within the attribute value match the regular expression, PingDataGovernance replaces all occurrences.

The regular expression and replacement string must be valid as described in the API documentation for the java.util.regex.Pattern class, including support for capture groups.

For example, consider the following body.

```
{
   "id":5,
   "username":"jsmith",
   "description":"Has a registered ID number of '123-45-6789'.",
   "secrets":{
      "description":"Has an SSN of '987-65-4321."
   }
}
```

Also, consider the following payload.

```
{
   "path":"$.secrets",
   "regex":"(\\\\d{3}-\\\\d{2})-\\\\d{4}",
   "replace":"$1-XXXX"
}
```

Applying the advice produces the following body with a changed `"secrets.description"` value.

```
{
   "id":5,
   "username":"jsmith",
   "description":"Has a registered ID number of '123-45-6789'.",
```

Access token validators

Access token validators verify the tokens that client applications submit when they request access to protected resources.

Specifically, access token validators 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.

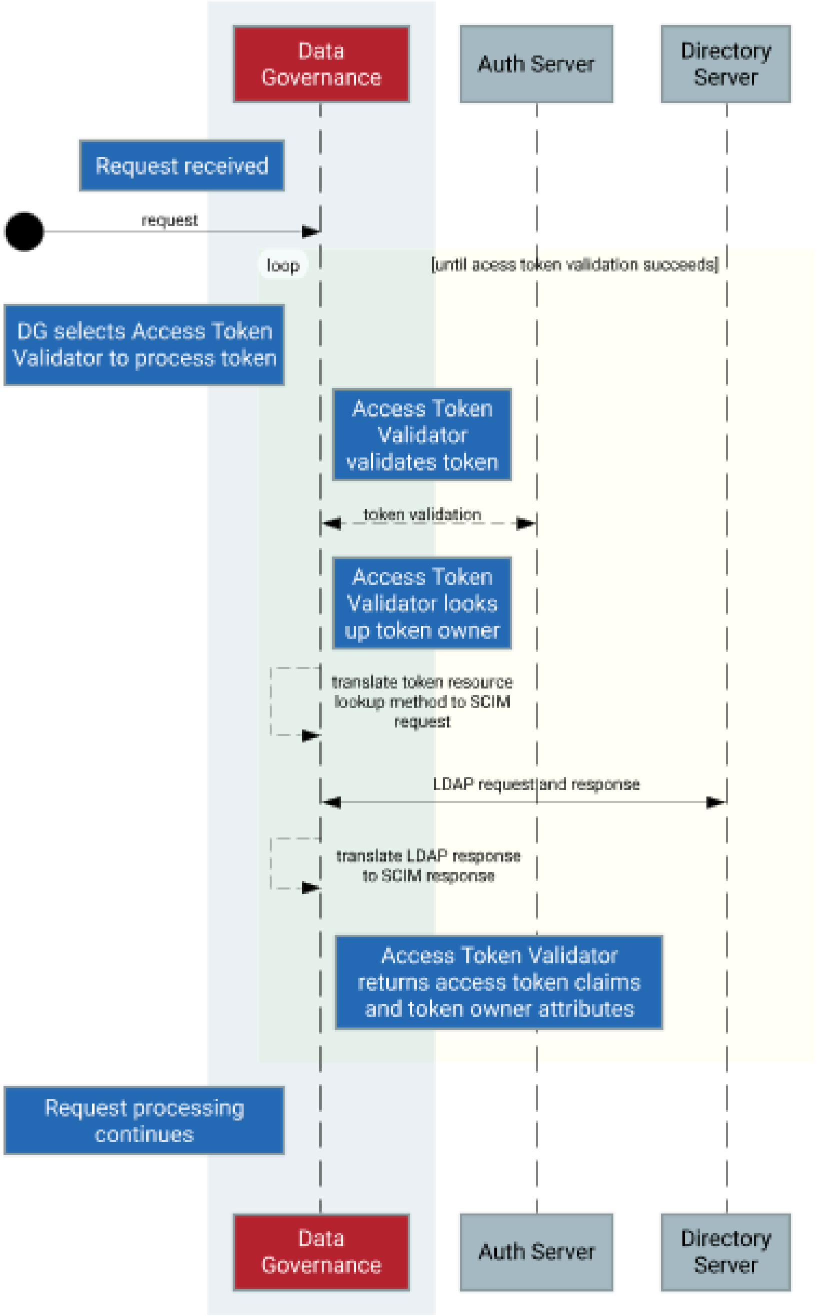
Most access tokens identify a user, also called the token owner, as its subject. Access token validators can retrieve the token owner's attributes from the user store using a related component called a token resource lookup method. The user data obtained by a token resource lookup method is sent to the policy decision point (PDP) so that policies can determine whether to authorize the request.

About access token validator processing

You can configure any number of access token validators for PingDataGovernance Server.

Each access token validator possesses an evaluation order index, an integer that determines its processing priority. Lower values are processed before higher values.

The following image shows the validation process when using an access token validator with the SCIM token resource lookup method.


Data Governance
Auth Server
Directory Server
Request received
request
loop
[until acess token validation succeeds]
DG selects Access Token Validator to process token
Access Token Validator validates token
token validation
Access Token Validator looks up token owner
translate token resource lookup method to SCIM request
LDAP request and response
translate LDAP response to SCIM response
Access Token Validator returns access token claims and token owner attributes
Request processing continues
Data Governance
Auth Server
Directory Server

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 the System for Cross-domain Identity Management (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. For more information, see *Make a user's profile available in policies* on page 241.

4. If the access token validator is unable to validate the access token, it passes the token 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 policy decision point (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, you should configure such policies to 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 works with a variety of access token validators.

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. 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 shown.

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

Replace `<client secret>` with the client secret value generated by the PingFederate client.

JWT access token validator

The JSON web token (JWT) access token validator verifies access tokens that are encoded in JWT format, which can be signed in JSON web signature (JWS) format or signed and encrypted in JSON web encryption (JWE) format.

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 JWT access token validator uses 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 faster 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 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 JSON web token (JWT) claims.

Always provide the boolean `active` claim when creating a mock token. 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 JSON web signature (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.

External API gateway access token validator

An external API gateway access token validator is a special access token validator that the Sideband API can use when the API gateway itself can validate and parse access tokens.

An external API gateway access token validator accepts a set of parsed access token claims from a trusted gateway and performs no further parsing or validation of its own. For information about how the tokens are processed, see *Access token validation* on page 175.

ⓘ **Note:**

External API gateway access token validators are exclusively for use by Sideband API endpoints. If you assign an external API gateway access token validator to any other server component, either explicitly or implicitly, it is ignored.

Example configuration

The following example shows how to configure an external API gateway access token validator with a token resource lookup method, and then assign it to an existing Sideband API endpoint.

```
dsconfig create-access-token-validator \
  --validator-name "API Gateway Access Token Validator" \
  --type external-api-gateway \
  --set enabled:true \
  --set evaluation-order-index:0
dsconfig create-token-resource-lookup-method \
  --validator-name "API Gateway Access Token Validator" \
  --method-name "Users by uid" \
  --type scim \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:0
dsconfig set-sideband-api-endpoint-prop \
  --endpoint-name "My API" \
  --set "access-token-validator:API Gateway-Provided Access Token Validator"
```

Token resource lookup methods

Most access tokens include a *subject*, which identifies the user who granted access to the application using the token. Access token validators can use token resource lookup methods to retrieve the access token subject's attributes from an external data store such as a PingDirectory Server. These attributes are then included in the policy request's `TokenOwner` attribute, allowing policies to make decisions based on some aspect of the user.

Token resource lookup methods work by taking the access token subject, which is usually a string identifier such as a GUID or username, and using that subject value to perform a search in a data store or API providing user data. For this reason, the data store or API must be accessible to PingDataGovernance Server; and in most cases, it should be the same data store or API used by the authorization server that issues the access tokens.

ⓘ **Note:**

Using a token resource lookup method is optional. If your policies do not need user profile information, you do not need to configure token resource lookup methods.

PingDataGovernance Server provides the following types of token resource lookup methods:

- *SCIM token resource lookup methods* on page 257
- *Third-party token resource lookup methods* on page 258

SCIM token resource lookup methods

SCIM token resource lookup methods use PingDataGovernance Server's SCIM subsystem to retrieve a token subject's attributes.

ⓘ **Note:**

Before you create a SCIM token resource lookup method, you must configure SCIM. See *SCIM configuration basics* on page 179.

To configure a SCIM token resource lookup method, you need to know the name of the access token claim that the authorization server uses for the subject identifier (typically, sub). You also need to know which user attribute is used as the subject identifier by the authorization server when it issues access token. If you have configured a mapping SCIM resource type, then the attribute name used by the authorization server and the attribute name in your SCIM schema might differ.

A SCIM token resource lookup method retrieves the token subject's attributes using the combination of the `scim-resource-type` and `match-filter` configuration properties.

Property	Description
`scim-resource-type`	The SCIM resource type that represents users that can be access token subjects.
`match-filter`	A SCIM 2 filter expression that matches a SCIM resource based on one or more access token claims.

The `match-filter` value must be a valid SCIM 2 filter expression that uniquely matches a single resource. The filter expression can include one or more variables that refer to claims found in the access token. These variables are indicated by enclosing a token claim name in percent (%) characters. When the token resource lookup method is invoked, the variable is filled in with the actual value from the access token claim.

For example, if a match filter has the value `id eq "%sub%"` and an access token contains a sub claim with the value `8ac3d8b5-4f17-33fa-a4b4-854599ed9a89`, then the token resource lookup method will perform a SCIM search using the filter `id eq "8ac3d8b5-4f17-33fa-a4b4-854599ed9a89"`.

The following example shows how to create a SCIM token resource lookup method using **`dsconfig`**. It assumes that a SCIM resource type called `Users` and an access token validator called `JWT Access Token Validator` already exist.

```
dsconfig create-token-resource-lookup-method
  --validator-name "JWT Access Token Validator" \
  --method-name "User by uid" \
  --type scim \
  --set evaluation-order-index:10 \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"'
```

Third-party token resource lookup methods

A third-party token resource lookup method is a custom implementation of a token resource lookup method that you write using the Server SDK. A third-party token resource lookup method can be useful for PingDataGovernance Server deployments where SCIM is not otherwise needed. For example, you could use a third-party token resource lookup method to connect a PingDataGovernance Server to a system that stores user data in a cloud directory.

For more information about writing custom server extensions, see the Server SDK documentation.

Server configuration

For a detailed look at configuration, see the Ping Identity PingDataGovernance Server Configuration Reference, located in the server's `docs` directory.

This section covers basic server configuration.

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 distinguished names (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, to add root DN users, use the PingDataGovernance Administrative Console.

About the dsconfig tool

The `dsconfig` tool provides a command-line interface to configure the underlying server configuration.

Use the `dsconfig` tool whenever you administer the server from a shell. When run without arguments, `dsconfig` enters an interactive mode that lets you browse and update 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. You can use a command line in this form 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 `\`, or backslash, character.

This example `dsconfig` script configures the PingDataGovernance Server 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 "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.

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

Setting the console session timeout

The session timeout for the console is 24 hours by default. When this duration is exceeded, all inactive users are logged off automatically.

To set a different timeout value, configure the `server.sessionTimeout` application parameter, which specifies the timeout duration in seconds. You can set the value as an init parameter either in the console or on the command line.

- Console

  In the PingDataGovernance Administrative Console, go to **Web Application Extensions**# **Console**. Specify the timeout value in the **Init Parameter** field.
- Command line

  Use the **`dsconfig`** tool. The following example uses a value of 1800 seconds (30 minutes).

  ```
  dsconfig set-web-application-extension-prop --no-prompt \
  --extension-name Console \
  --add init-parameter:server.sessionTimeout=1800
  ```

For the changes to take effect, restart the HTTP(S) Connection Handler, or the server itself.

About the configuration audit log

The configuration audit log records the configuration commands that represent configuration changes, as well as the configuration commands that undo the changes.

All successful configuration changes are recorded to the file `logs/config-audit.log`.

```
$ 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 "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

The server presents a server certificate 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.

PingDataGovernance Server uses server certificates.

During setup, administrators have the option of using self-signed certificates or certificate authority (CA)-signed certificates for the server certificate. Use CA-signed certificates wherever possible. Use self-signed certificates for demonstration and proof-of-concept environments only.

If you specify the option **`--generateSelfSignedCertificate`** during setup, the server certificate generates 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, see the setup tool's *Help and the Installation Guide*.

Replacing the server certificate

Whether the server was set up with self-signed or certificate authority (CA)-signed certificates, the steps to replace the server certificate are nearly identical.

About this task

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 Java KeyStore (JKS) 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.

While the certificate is being replaced, existing secure connections continue to work. If you restart the server, 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, perform the following steps:

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.
   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.

Preparing a new keystore with the replacement key pair

You can replace the self-signed certificate with an existing key pair. As an alternative, you can use the certificate that is associated with the original key pair.

Using an existing key pair

To use an existing key pair, use the **`manage-certificates`** tool that is located in the server's `bin` or `bat` directory, depending on your operating system.

About this task

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.

Steps

- Import the existing certificates using the `manage-certificates import-certificate`.

  Order the certificates that use the **`--certificate-file`** option so that each subsequent certificate functions as the issuer for the previous one.

  List the server certificate first, then any intermediate certificates, and then list the root certificate authority (CA) certificate. Because some deployments do not feature an intermediate issuer, you might need to import only the server certificate and a single issuer.

  For example, the following command imports the 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
```

Replacing the certificate associated with the original key pair

Replace the certificate associated with the original server-generated private key (`server-cert`) if it has expired or must be replaced with a certificate from a different certificate authority (CA).

About this task

Perform the following steps to replace the certificate associated with the original key pair:

Steps

1. Create a CSR file for the `server-cert`.

```
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`.

```
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`.

```
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
```

Importing earlier trusted certificates into the new keystore

You must import the trusted certificates of other servers in the topology into the new `truststore` file.

About this task

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.

   ```
   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`.

      ```
      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`.

      ```
      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
      ```

Updating the server configuration to use the new certificate

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.

About this task

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`.

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

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

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

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

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

   ```
   $ 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"
   ```

Replacing the key store and trust store files with the new ones

Replace the key store and trust store files in the server's `config` directory to make the new server certificates take effect.

About this task

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.

Steps

- Replace the `keystore` and `truststore` as shown in the following example.

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

Retiring the previous certificate

Retire the previous certificate by removing it from the topology registry after it expires.

Steps

- Remove the previous certificate from the topology registry, as shown in the following example.

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

Server status

You can check server status using the PingDataGovernance Server Administrative Console, the **`status`** command, or the availability servlet.

Administrative Console

You can access status information in the console, in the **Status** tab.

For information about how to access the console, see *PingDataGovernance Administration Console* on page 260.

`status` command

The PingDataGovernance distribution includes the **`bin/status`** command that you can use to see various information about the server, including its status and the status of its LDAP external servers.

Availability servlet

PingDataGovernance provides an HTTP servlet extension that you can use to retrieve the server's current availability state. The servlet accepts any `GET`, `POST`, or `HEAD` request sent to a specified endpoint and returns a minimal response whose HTTP status code can help you 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 can optionally include a JSON object with an `availability-state` field with the name of the current state.

The servlet has these endpoints:

- `/available-state`

  This endpoint can prove useful for load balancers that should only route requests to servers that are fully available.

  The following table shows the responses for this endpoint.

  Endpoint responses and server status

Response	Server state
200 (OK)	AVAILABLE
503 (Service Unavailable)	DEGRADED or UNAVAILABLE

- `/available-or-degraded-state`

  This endpoint can prove useful for orchestration frameworks if you want to destroy and replace any instance that is completely unavailable.

  The following table shows the responses for this endpoint.

  Endpoint responses and server status

Response	Server state
200 (OK)	AVAILABLE or DEGRADED
503 (Service Unavailable)	UNAVAILABLE

Server availability

You can monitor the availability of PingDataGovernance Server and set up load balancing or auto-healing for it.

Use the following gauges to monitor PingDataGovernance Server availability:

- User Store Availability gauge
- Endpoint Average Response Time (Milliseconds) gauge
- HTTP Processing (Percent) gauge

With monitoring, you can set up load balancing or auto-healing.

For auto-healing, configure your container orchestrator to base a health check on the availability servlet mentioned in *Server status* on page 265. If the availability is not as desired, fail the health check. The orchestrator should then start a replacement server for the unhealthy server.

User Store Availability gauge

The `User Store Availability` gauge monitors the directory servers that provide user data to PingDataGovernance.

If PingDataGovernance cannot reach these directory servers, it cannot:

- Retrieve token owner information using a SCIM Token Resource Lookup Method
- Handle SCIM 2 API requests

In this case, this gauge marks the status of PingDataGovernance itself as UNAVAILABLE.

The status appears in the following locations:

- The Administrative Console on the **Status** tab, in the **Operational Status** entry.
- The **Operational Status** line in the **`bin/status`** output.
- The Availability servlet. See *Server status* on page 265.

When PingDataGovernance has a status of UNAVAILABLE, a load balancer can try to route traffic to a different PingDataGovernance server or take some other action. See *Example: auto-healing* on page 269.

If you followed the standard setup and configuration given in *Getting started with PingDataGovernance (tutorials)* on page 52, the `User Store Availability` gauge should automatically work.

Important:

The gauge assumes the PingDataGovernance LDAP Store Adapter name is UserStoreAdapter. If your PingDataGovernance SCIM configuration uses a different name, you must edit the gauge's data source to reflect the custom store adapter name. Use the following **`dsconfig`** command to make this change, replacing *`<CustomStoreAdapter>`* in the last line with the actual name.

```
dsconfig set-gauge-data-source-prop \
  --source-name "User Store Availability"  \
  --set "include-filter:(store-adapter-name=<CustomStoreAdapter>)"
```

If your PingDataGovernance deployment does not use SCIM or SCIM Token Resource Lookup Methods, you can disable the gauge with the following command.

```
dsconfig set-gauge-prop \
  --gauge-name "User Store Availability"  \
  --set enabled:false
```

Endpoint Average Response Time (Milliseconds) gauge

The `Endpoint Average Response Time (Milliseconds)` gauge monitors the average time that PingDataGovernance takes to respond to queries on various endpoints.

The gauge monitors the following types of endpoints:

- Gateway endpoints
- Sideband endpoints
- System for Cross-domain Identity Management (SCIM) 2 endpoints
- OpenBanking endpoints

The gauge can raise alarms or generate a DEGRADED or UNAVAILABLE status that you can use to configure load balancing or auto-healing.

This gauge does not count the time spent waiting for an upstream server response.

By default, this gauge does nothing. To begin using it, set the levels at which the gauge activates to reasonable values for your environment using **`dsconfig`**.

The following table explains the values you set for this gauge.

Value	Description
`minor-value`	This value, in milliseconds, represents a warning condition. An alarm is raised, but the server continues to operate as normal.
`major-value`	This value, in milliseconds, represents the point at which the server is considered DEGRADED.
`critical-value`	This value, in milliseconds, represents the point at which the server is considered UNAVAILABLE.

You can find the server's availability state by using an option discussed in *Server status* on page 265.

The following example shows how to activate the gauge.

Note:

You might need to experiment to find values that work for your environment.

```
dsconfig set-gauge-prop
  --gauge-name "Endpoint Average Response Time (Milliseconds)"
  --set minor-value:200
  --set major-value:500
  --set critical-value:2000
```

HTTP Processing (Percent) gauge

The `HTTP Processing (Percent)` gauge monitors usage of available HTTP worker threads.

The gauge can raise alarms or generate a DEGRADED or UNAVAILABLE status that you can use to configure load balancing or auto-healing.

By default, this gauge raises an alarm at 70% usage, and it raises an alert at 90% usage. Also by default, the gauge does not mark the server as DEGRADED or UNAVAILABLE.

The following table explains the values and descriptions you set for this gauge.

HTTP processing gauge values and descriptions

Value	Description
`warning-value`	This percentage value represents a warning condition. An alarm is raised, but the server continues to operate as normal. It defaults to 70%.
`major-value`	This percentage value represents a severe condition. An alarm is raised, and the server enters a DEGRADED state. It is not set by default. To enable the DEGRADED state, you must set `server-degraded-severity-level`.
`critical-value`	This percentage value represents a critical condition. An alarm is raised, an alert is generated, and the server is put into an UNAVAILABLE state. It defaults to 90%. To enable the UNAVAILABLE state, you must set `server-unavailable-severity-level`.

Value	**Description**
`server-degraded-severity-level`	The alarm level at which the server enters a DEGRADED state. By default, this gauge does not mark the server as DEGRADED. To enable the DEGRADED state, set to `major`.
`server-unavailable-severity-level`	The alarm level at which the server enters an UNAVAILABLE state. By default, this gauge does not mark the server as UNAVAILABLE. To enable the UNAVAILABLE state, set to `critical`.

You can find the server's availability state by using an option discussed in *Server status* on page 265.

The following example shows how to activate the gauge.

ⓘ **Note:**

You might need to experiment to find values that work for your environment.

```
dsconfig set-gauge-prop
  --gauge-name "HTTP Processing (Percent)"
  --set major-value:85
  --set server-degraded-severity-level:major
  --set server-unavailable-severity-level:critical
```

Example: auto-healing

Using gauges, set up auto-healing in a container deployment to address an unavailable server.

Steps

1. Configure one or more of the gauges described in *Server availability* on page 266.
2. Configure the gauges to trigger the UNAVAILABLE status.

   By default, the gauges do not trigger the UNAVAILABLE status.

   As discussed in *Endpoint Average Response Time (Milliseconds) gauge* on page 267 and *HTTP Processing (Percent) gauge* on page 268, use the **`dsconfig`** command to adjust the following values for your environment. Each system is different so you might need to adjust the values several times to determine your ideal configuration.

   a. For the `Endpoint Average Response Time (Milliseconds)` gauge, set `critical-value`.
   b. For the `HTTP Processing (Percent)` gauge, set both `critical-value` and `server-unavailable-severity-level`.
3. Configure the container orchestrator to use the `available-or-degraded-state` endpoint to detect whether the server is alive.

   For information about the endpoint, see *Availability servlet* on page 266.

Available gauges

PingDataGovernance makes the following gauges available. You can manage these gauges using the Administrative Console or the **dsconfig** tool.

Gauge name	Enabled by default	Description
Available File Descriptors	true	Monitors the number of file descriptors available to the server process. The server allows for an unlimited number of connections by default but is restricted by the file descriptor limit on the operating system. You can configure the number of file descriptors that the server uses by either setting the NUM_FILE_DESCRIPTORS environment variable or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_DESCRIPTORS=12345`. If you do not use either of these options, the server uses the default of 65535. Running out of available file descriptors can lead to unpredictable behavior and severe system instability.
Certificate Expiration (Days)	true	Monitors the expiration dates of key server certificates. A server certificate expiring can cause server unavailability, degradation, or loss of key server functionality. Replace certificates nearing the end of their validity as soon as possible. For more information about server certificates and how they are managed, see the **status** tool or **Status** in the Administrative Console.
CPU Usage (Percent)	true	Monitors server CPU use and provides an averaged percentage for the interval defined. The monitored resource is the host system's CPU, which does not include a resource identifier. If CPU use is high, check the server's current workload and other processes on the system and make any needed adjustments. Reducing the load on the system will lead to better response times.

Gauge name	Enabled by default	Description
Disk Busy (Percent)	true	Monitors the percentage of disk use time averaged over the specified update interval. This gauge requires that you enable the Host System Monitor Provider and that you register any monitored disks by using the `disk-devices` property of that configuration object. The resource identifier for this gauge is the disk device name. Use the **`iostat`** command or a similar system utility to see a list of disk device names. A separate gauge monitor entry is created for each monitored disk.
Endpoint Average Response Time (Milliseconds)	false	Monitors the average response time across all endpoints since the server was started. This number does not include requests to the upstream server. There is no resource identifier associated with this gauge. The monitored resource is overall response time of all requests to DataGovernance servlets since the server was started. High response times might be indicative of a number of factors including a disk-bound server, network latency, or misconfiguration. Enabling the Stats Logger plugin can help isolate problems. For more information, see *Endpoint Average Response Time (Milliseconds) gauge* on page 267.
HTTP Processing (Percent)	true	Monitors the percentage of time that request handler threads spend processing HTTP requests. This percentage represents the inverse of the server's ability to handle new requests without queueing. For more information, see *HTTP Processing (Percent) gauge* on page 268.

Gauge name	Enabled by default	Description
JVM Memory Usage (Percent)	true	Monitors the percentage of Java Virtual Machine memory that is in use. This value naturally fluctuates due to garbage collection, so the minimum value within an interval is reported because it is a better indication of overall memory growth. When the memory usage exceeds 90%, contact Ping Customer Support because the server is either misconfigured or has a memory leak. As memory usage approaches 100%, the server is more and more likely to experience garbage collection pauses, which leave the server unresponsive for a long time. Restarting the server is likely the only remedy for this situation. Before you restart the server, run **`collect-support-data`** and capture the output of `jmap -histo <server-pid>` to provide to customer support. The PID of the server is in `<server-root>/logs/server.pid`.
License Expiration (Days)	true	Monitors the expiration date of the product license. An expired license causes warnings to appear in the server's logs and in the **`status`** tool output. Request a license key through the Ping Identity licensing website *https://www.pingidentity.com/en/account/request-license-key.html* or contact sales@pingidentity.com. Use the **`dsconfig`** tool to update the License configuration's license key property.
Memory Usage (Percent)	false	Monitors the percentage of memory use averaged over the update interval defined. The monitored resource is the host system's memory use, which does not have a resource identifier. Some operating systems, including Linux, use the majority of memory for file system cache, which is freed as applications need it. If memory use is high, check the applications that are running on the server.

Gauge name	Enabled by default	Description
Strong Encryption Not Available	true	Indicates the JVM does not appear to support strong encryption algorithms, like 256-bit AES. The server will fall back to using weaker algorithms, like 128-bit AES. To enable support for strong encryption, update your JVM to a newer version that supports it by default; alternatively, install or enable the unlimited encryption strength jurisdiction policy files in your Java installation.
User Store Availability	true	Monitors availability of the SCIM user store. If the LDAP directory servers are unavailable, the "UserStoreAdapter" cannot forward requests. Also, the server cannot process SCIM requests or perform token owner lookups. Ensure that LDAP directory servers are available. For more information, see *User Store Availability gauge* on page 266.

Common alarms

The server uses alarms and alerts to notify administrators of situations that might require intervention.

Policy Decision Service unavailable

PingDataGovernance Server raises this alarm if it cannot process policy decisions because the Policy Decision Service requires further configuration. When this alarm is present, PingDataGovernance Server cannot handle requests for the following services:

- API Security Gateway
- Sideband API
- SCIM 2
- PDP API

The alarm message typically indicates the cause for the Policy Decision Service's UNAVAILABLE state. The administrator should check the Policy Decision Service configuration's `pdp-mode` and `trust-framework-version` properties to ensure that they are set correctly.

Trust framework update needed

The server raises this alarm if the Policy Decision Service is configured with a deprecated `trust-framework-version` value. When this alarm is present, PingDataGovernance does continue to accept requests. However, the administrator is strongly encouraged to take the following actions:

1. Update policies to use a new Trust Framework version. See *Upgrading the Trust Framework and policies* on page 142.
2. Export a new deployment package (if using embedded PDP mode).
3. Load the updated policies and set `trust-framework-version` in the Policy Decision Service to the current version.

The following example uses **dsconfig** to set trust-framework-version to v2.

```
dsconfig set-policy-decision-service-prop \
  --set trust-framework-version:v2
```

LDAP External Server Health Reclassified from AVAILABLE to UNAVAILABLE

The server raises this alarm if an LDAP health check determines that an LDAP external server used by the SCIM subsystem is unavailable. This can occur for a number of reasons; the most typical cause is a network or SSL connectivity problem.

External server initialization failed

You see this alarm at server startup if an LDAP health check determines that an LDAP external server used by the SCIM subsystem is unavailable. This can occur for a number of reasons; the most typical cause is a network or SSL connectivity problem.

User Store Availability

The server raises this alarm if the SCIM subsystem's UserStoreAdapter is unavailable. When this alarm is present, PingDataGovernance Server cannot process SCIM API requests or SCIM token resource lookup method operations. This alarm generally occurs if the underlying data stores are unavailable. To resolve this alarm, determine why the data stores are unavailable and resolve the problem.

If your PingDataGovernance deployment does not require SCIM, you can disable this alarm by disabling the User Store Availability gauge using the following command.

```
dsconfig set-gauge-prop \
  --gauge-name "User Store Availability" \
  --set enabled:false
```

No Enabled Alert Handlers

By default, an administrator can check for server alerts through the error log, the **status** tool, and the Administrative Console. This alarm warns the administrator that they should also configure an alert handler to ensure that the server can actively notify them of current or impending problems. The server provides alert handlers for this purpose. The handlers can deliver alerts by email or through a monitoring application using JMX or SNMP.

The following example shows how to configure an alert handler to send alert emails through the SMTP server <*smtp.example.com*>.

```
dsconfig create-external-server \
  --server-name "SMTP Server" \
  --type smtp \
  --set server-host-name:<smtp.example.com>

dsconfig set-global-configuration-prop \
  --add "smtp-server:SMTP Server"

dsconfig create-alert-handler \
  --handler-name "SMTP Alert Handler" \
  --type smtp \
  --set enabled:true \
  --set 'sender-address:joey@example.com' \
  --set 'recipient-address:deedee@example.com'
```

If you are running a nonproduction environment, you can disable this alarm by running the following **dsconfig** command.

```
dsconfig set-alarm-manager-prop \
  --set suppressed-alarm:no-enabled-alert-handlers
```

Insecure access token validator enabled

This alarm warns the administrator that a mock access token validator is enabled. Mock access token validators can be very useful in test environments because they allow PingDataGovernance Server to accept HTTP API requests without the overhead of setting up an OAuth 2 authorization server. However, because they do not actually authenticate access tokens, they are insecure and should never be used in a production environment.

The following example shows how to disable an access token validator called "Mock Token Validator."

```
dsconfig set-access-token-validator-prop \
  --validator-name "Mock Token Validator" \
  --set enabled: false
```

Sensitive data may be logged

This alarm warns the administrator that a trace log publisher has been configured to record debug messages. Debug log messages are not guaranteed to exclude potentially sensitive data, so their use is strongly discouraged in a production environment. You should not use them with anything but test data.

To disable a trace log publisher called "Debug Trace Logger," run this command.

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

Manage monitoring

You can configure monitoring options in PingDataGovernance Server to suit your needs.

The following sections describe the monitoring options.

- *StatsD monitoring endpoint* on page 275
- *Sending metrics to Splunk* on page 276

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 the 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

The following example shows how 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, perform the following steps.

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. This example shows how 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, perform the following steps.

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 276.
- 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 PingDataGovernance 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

Use a Splunk Universal Forwarder to securely send UDP (or TCP) data 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.

Steps

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

Managing HTTP correlation IDs

An HTTP correlation ID is a unique ID that you can use to track requests as they make their way through the system.

The following sections explain how to configure and use these IDs.

About HTTP correlation IDs

HTTP correlation IDs let you trace requests.

A typical request to a software system is handled by multiple subsystems, which might be distinct servers on distinct hosts across different locations. Tracing the request flow on such distributed systems can be challenging because log messages are scattered across various systems and intermingled with messages for other requests.

To solve this problem, a system can assign a correlation ID to a request that it adds to every associated operation as the request flows through the larger system. With the correlation ID, you can easily locate and group related log messages.

PingDataGovernance, PingDirectory, and their related products support correlation IDs for all HTTP requests received through the HTTP(S) Connection Handler. For more information about HTTP connection handlers in PingDirectory, see *HTTP connection handlers*.

How PingDataGovernance handles correlation IDs

- When any HTTP request is received, PingDataGovernance automatically assigns the request a correlation ID.
- All related activity appears in the trace logs with this correlation ID.
- The PingDataGovernance gateway adds the correlation ID header to requests it forwards.
- The LDAP Store Adapter used by the SCIM 2 service uses the correlation ID as the client request ID value in Intermediate Client Request Controls that it sends to the downstream Ping LDAP server.

  You can find this value in the `via` key of records logged by the LDAP server's access log.

  If the LDAP server is a PingDirectory Proxy Server, the Intermediate Client Request Control is forwarded in turn to the downstream LDAP server.

How other Ping products handle correlation IDs

- When any HTTP request is received, it is automatically assigned a correlation ID.
- You can use this correlation ID to correlate HTTP responses with messages recorded to the HTTP Detailed Operation log and the trace log.
- For specific web APIs, the correlation ID might also be passed to the LDAP subsystem.

- For the SCIM 1, SCIM 2, Delegated Admin, Consent, and Directory REST APIs, the correlation ID appears with associated requests in LDAP logs in the `correlationID` key.

Enabling or disabling correlation ID support

Correlation ID support is enabled by default for each HTTP connection handler, but you can optionally disable it.

Steps

- To disable correlation ID support for the HTTPS connection handler, run the following command.

```
dsconfig set-connection-handler-prop --handler-name "HTTPS Connection
 Handler" --set use-correlation-id-header:false
```

- To enable correlation ID support for the HTTPS connection handler, run the following command.

```
dsconfig set-connection-handler-prop --handler-name "HTTPS Connection
 Handler" --set use-correlation-id-header:true
```

Configuring the correlation ID response header

You can optionally change the correlation ID response header that PingDataGovernance Server sends with HTTP requests.

About this task

By default, PingDataGovernance Server generates a correlation ID for every HTTP request and sends it in the response with the `Correlation-Id` response header.

To customize this response header name:

Steps

- Run the **`dsconfig`** command.
  The following example changes the correlation ID response header to `X-Request-Id`.

```
dsconfig set-connection-handler-prop --handler-name "HTTPS Connection
 Handler" --set correlation-id-response-header:X-Request-Id
```

How the server manages correlation IDs

By default, the server looks for a correlation ID header on the request and uses the value if found. This behavior integrates the server into a larger system of other servers using correlation IDs.

If a correlation ID header is not found, the server generates a new, unique correlation ID for each HTTP request.

The connection handler uses the `correlation-id-request-header` property to determine which request headers are correlation ID headers, as shown in the following configuration. The actual default configuration might differ.

```
dsconfig set-connection-handler-prop --handler-name "HTTPS Connection
 Handler" \
  --set correlation-id-request-header:X-Request-Id \
  --set correlation-id-request-header:X-Correlation-Id \
  --set correlation-id-request-header:Correlation-Id \
  --set correlation-id-request-header:X-Amzn-Trace-Id
```

If a request contains more than one of the previous correlation ID headers, the server checks the configured header names in order, and then uses the first one found.

Server SDK support

For Server SDK extensions that have access to the current `HttpServletRequest`, the extension can retrieve the current correlation ID as a `String` through the `HttpServletRequest`'s `com.pingidentity.pingdata.correlation_id` attribute.

Consider this example.

```
(String) request.getAttribute("com.pingidentity.pingdata.correlation_id");
```

Example: HTTP correlation ID

In this example, you make a request to the PingDirectory REST API and see how the correlation ID allows you to correlate HTTP-specific log messages with LDAP-specific log messages.

First, make a GET request to the PingDirectory REST API.

The response includes a `Correlation-Id` header with the value `ee919049-6710-4594-9c66-28b4ada4b127`. Because the request does not include a correlation ID, the server generates the header and value.

```
GET /directory/v1/me?includeAttributes=mail HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Authorization: Bearer ...
Connection: keep-alive
Host: localhost:1443
User-Agent: HTTPie/0.9.9


HTTP/1.1 200 OK
Content-Length: 266
Content-Type: application/hal+json
Correlation-Id: ee919049-6710-4594-9c66-28b4ada4b127
Date: Fri, 02 Nov 2018 15:16:50 GMT
Request-Id: 369

{
    "_dn": "uid=user.86,ou=People,dc=example,dc=com",
    "_links": {
        "schemas": [
            {
                "href": "https://localhost:1443/directory/v1/schemas/
inetOrgPerson"
            }
        ],
        "self": {
            "href": "https://localhost:1443/directory/v1/
uid=user.86,ou=People,dc=example,dc=com"
        }
    },
    "mail": [
        "user.86@example.com"
    ]
}
```

Use the correlation ID to search the HTTP trace log for matching log records.

```
$  grep 'correlationID="ee919049-6710-4594-9c66-28b4ada4b127"'
 PingDirectory/logs/debug-trace
```

```
[02/Nov/2018:10:16:50.294 -0500] HTTP REQUEST requestID=369
 correlationID="ee919049-6710-4594-9c66-28b4ada4b127" product="Ping Identity
 Directory Server" instanceName="ds1" startupID="W9ikqA==" threadID=52358
 from=[0:0:0:0:0:0:0:1]:58918 method=GET url="https://0:0:0:0:0:0:0:1:1443/
directory/v1/me?includeAttributes=mail"
[02/Nov/2018:10:16:50.526 -0500] DEBUG ACCESS-TOKEN-VALIDATOR-PROCESSING
 requestID=369 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 msg="Identity Mapper with DN 'cn=User ID Identity Mapper,cn=Identity
 Mappers,cn=config' mapped ID 'user.86' to entry DN
 'uid=user.86,ou=people,dc=example,dc=com'"
[02/Nov/2018:10:16:50.526 -0500] DEBUG ACCESS-TOKEN-VALIDATOR-PROCESSING
 requestID=369 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 accessTokenId="201811020831" msg="Token Validator 'Mock Access Token
 Validator' validated access token with active = 'true', sub = 'user.86',
 owner = 'uid=user.86,ou=people,dc=example,dc=com', clientId = 'client1',
 scopes = 'ds', expiration = 'none', not-used-before = 'none', current time
 = 'Nov 2, 2018 10:16:50 AM CDT' "
[02/Nov/2018:10:16:50.531 -0500] HTTP RESPONSE requestID=369
 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 accessTokenId="201811020831" product="Ping Identity Directory Server"
 instanceName="ds1" startupID="W9ikqA==" threadID=52358 statusCode=200
 etime=236.932 responseContentLength=266
[02/Nov/2018:10:16:50.531 -0500] DEBUG HTTP-FULL-REQUEST-AND-RESPONSE
 requestID=369 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 accessTokenId="201811020831" product="Ping Identity Directory
 Server" instanceName="ds1" startupID="W9ikqA==" threadID=52358
 from=[0:0:0:0:0:0:0:1]:58918 method=GET url="https://0:0:0:0:0:0:0:1:1443/
directory/v1/me?includeAttributes=mail" statusCode=200 etime=236.932
 responseContentLength=266 msg="
```

You can also find the LDAP log messages associated with the correlation ID.

```
$  grep 'correlationID="ee919049-6710-4594-9c66-28b4ada4b127"'
 PingDirectory/logs/access
[02/Nov/2018:10:16:50.529 -0500] SEARCH RESULT instanceName="ds1"
 threadID=52358 conn=-371045 op=1657393 msgID=1657394
 origin="Directory REST API" httpRequestID="369"
 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 authDN="uid=user.86,ou=people,dc=example,dc=com" requesterIP="internal"
 requesterDN="uid=user.86,ou=People,dc=example,dc=com"
 requestControls="1.3.6.1.4.1.30221.2.5.2" via="app='PingDirectory-
ds1' clientIP='0:0:0:0:0:0:0:1' sessionID='201811020831'
 requestID='ee919049-6710-4594-9c66-28b4ada4b127'"
 base="uid=user.86,ou=people,dc=example,dc=com" scope=0 filter="(&)"
 attrs="mail,objectClass" resultCode=0 resultCodeName="Success" etime=0.684
 entriesReturned=1
[02/Nov/2018:10:16:50.530 -0500] EXTENDED RESULT
 instanceName="ds1" threadID=52358 conn=-371046 op=1657394
 msgID=1657395 origin="Directory REST API" httpRequestID="369"
 correlationID="ee919049-6710-4594-9c66-28b4ada4b127"
 authDN="cn=Internal Client,cn=Internal,cn=Root DNs,cn=config"
 requesterIP="internal" requesterDN="cn=Internal Client,cn=Internal,cn=Root
 DNs,cn=config" requestControls="1.3.6.1.4.1.30221.2.5.2"
 via="app='PingDirectory-ds1' clientIP='0:0:0:0:0:0:0:1'
 sessionID='201811020831' requestID='ee919049-6710-4594-9c66-28b4ada4b127'"
 requestOID="1.3.6.1.4.1.30221.1.6.1" requestType="Password
 Policy State" resultCode=0 resultCodeName="Success"
 etime=0.542 usedPrivileges="bypass-acl,password-reset"
 responseOID="1.3.6.1.4.1.30221.1.6.1" responseType="Password Policy State"
 dn="uid=user.86,ou=People,dc=example,dc=com"
[02/Nov/2018:10:16:50.530 -0500] SEARCH RESULT instanceName="ds1"
 threadID=52358 conn=-371048 op=1657397 msgID=1657398
 origin="Directory REST API" httpRequestID="369"
 correlationID="ee919049-6710-4594-9c66-28b4ada4b127" authDN="cn=Internal
 Client,cn=Internal,cn=Root DNs,cn=config" requesterIP="internal"
 requesterDN="cn=Internal Client,cn=Internal,cn=Root DNs,cn=config"
 requestControls="1.3.6.1.4.1.30221.2.5.2" via="app='PingDirectory-
ds1' clientIP='0:0:0:0:0:0:0:1' sessionID='201811020831'
 requestID='ee919049-6710-4594-9c66-28b4ada4b127'" base="cn=Default Password
 Policy,cn=Password Policies,cn=config" scope=0 filter="(&)" attrs="ds-
cfg-password-attribute" resultCode=0 resultCodeName="Success" etime=0.065
 preAuthZUsedPrivileges="bypass-acl,config-read" entriesReturned=1
```

Command-line tools

PingDataGovernance Server provides a full suite of command-line tools to administer the server. Most of these tools are in the `bin` directory for Linux systems and the `bat` directory for Microsoft Windows systems; however, some of the tools are in the root directory of the distribution.

Available command-line tools

PingDataGovernance Server provides the following command-line tools. You can run these tools in interactive, noninteractive, or script mode.

Tools help

For	Use this option	Example
Information about arguments and subcommands Usage examples	`--help`	**`dsconfig --help`**
A list of subcommands	`--help-subcommands`	**`dsconfig --help-subcommands`**
More information about a subcommand	`--help` with the subcommand	**`dsconfig list-log-publishers --help`**

For more information and examples, see the *PingDataGovernance Command-Line Tool Reference* at `docs/cli/index.html`.

Command-line tools

Tool	Description
backup	Run full or incremental backups on one or more PingDataGovernance Server backends. This tools supports the use of a properties file to pass command-line arguments. See *Saving options in a file* on page 285.
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.

Tool	Description
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a zip archive that you can send to a technical support representative.
config-diff	Compare PingDataGovernance Server configurations and produce the dsconfig batch file needed to bring the source inline with the target.
create-initial-config	Create an initial PingDataGovernance Server configuration.
create-rc-script	Create a Run Control (RC) script to start, stop, and restart the PingDataGovernance Server on UNIX-based systems.
create-systemd-script	Create a **`systemd`** script to start and stop the PingDataGovernance Server on Linux-based systems.
dsconfig	View and edit the PingDataGovernance Server configuration.
dsjavaproperties	Configure the JVM options used to run PingDataGovernance Server and its associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM options and JAVA_HOME environment variable.
encrypt-file	Encrypt or decrypt data using a key generated from a user-supplied passphrase, a key generated from an encryption settings definition, or a key shared among servers in the topology. The data to be processed can be read from a file or standard input, and the resulting data can be written to a file or standard output. You can use this command to encrypt and subsequently decrypt arbitrary data, or to decrypt encrypted backups, LDIF exports, and log files generated by the server.
encryption-settings	Manage the server encryption settings database.
ldap-diff	Compare the contents of two LDAP servers.
ldap-result-code	Display and query LDAP result codes.
ldapcompare	Perform LDAP compare operations in the PingDataGovernance Server.
ldapdelete	Delete one or more entries from an LDAP directory server. You can provide the DNs of the entries to delete using named arguments, as trailing arguments, from a file, or from standard input. Alternatively, you can identify entries to delete using a search base DN and filter.

Tool	Description
ldapmodify	Apply a set of add, delete, modify, and/or modify DN operations to a directory server. Supply the changes to apply in LDIF format, either from standard input or from a file specified with the `ldifFile` argument. Change records must be separated by at least one blank line.
ldappasswordmodify	Perform LDAP password modify operations in PingDataGovernance Server.
ldapsearch	Process one or more searches in an LDAP directory server.
list-backends	List the backends and base DNs configured in PingDataGovernance Server.
manage-certificates	Manage certificates and private keys in a JKS or PKCS #12 key store.
manage-extension	Install or update PingDataGovernance Server extension bundles.
manage-profile	Generate, compare, install, and replace server profiles.
manage-tasks	Access information about pending, running, and completed tasks scheduled in the PingDataGovernance Server.
manage-topology	Manage the topology registry.
prepare-external-store	Prepare a PingDataGovernance Server and an external server for communication.
reload-http-connection-handler-certificates	Reload HTTPS Connection Handler certificates.
remove-backup	Safely remove a backup and optionally all of its dependent backups from the specified PingDataGovernance Server backend.
remove-defunct-server	Remove a server from this server's topology.
replace-certificate	Replace the listener certificate for this PingDataGovernance Server server instance.
restore	Restore a backup of a PingDataGovernance Server backend.
revert-update	Revert this server package's most recent update.
review-license	Review and/or indicate your acceptance of the license agreement defined in `legal/LICENSE.txt`.
rotate-log	Trigger the rotation of one or more log files.

Tool	Description
sanitize-log	Sanitize the contents of a server log file to remove potentially sensitive information while still attempting to retain enough information to make it useful for diagnosing problems or understanding load patterns. The sanitization process operates on fields that consist of name-value pairs. The field name is always preserved, but field values might be tokenized or redacted if they might include sensitive information. Supported log file types include the file-based access, error, sync, and resync logs, as well as the operation timing access log and the detailed HTTP operation log.
schedule-exec-task	Schedule an exec task to run a specified command in the server. To run an exec task, a number of conditions must be satisfied: the server's global configuration must have been updated to include 'com.unboundid.directory.server.tasks.ExecTask' in the set of allowed-task values, the requester must have the `exec-task` privilege, and the command to execute must be listed in the `exec-command-whitelist.txt` file in the server's `config` directory. The absolute path (on the server system) of the command to execute must be specified as the first unnamed trailing argument to this program, and the arguments to provide to that command (if any) should be specified as the remaining trailing arguments. The server root is used as the command's working directory, so any arguments that represent relative paths are interpreted as relative to that directory.
search-logs	Search across log files to extract lines matching the provided patterns, like the **`grep`** command-line tool. The benefits of using this tool over **`grep`** are its ability to handle multiline log messages, extract log messages within a given time range, and the inclusion of rotated log files.
server-state	View information about the current state of the PingDataGovernance Server process.
setup	Perform the initial setup for a server instance.
start-server	Start the PingDataGovernance Server.
status	Display basic server information.
stop-server	Stop or restart the server.
sum-file-sizes	Calculate the sum of the sizes for a set of files.
uninstall	Uninstall PingDataGovernance Server.

Tool	Description
update	Update the PingDataGovernance Server to a newer version by downloading and unzipping the new server install package on the same host as the server you wish to update. Then, use the **`update`** tool from the new server package to update the older version of the server. Before upgrading a server, you should ensure that it is capable of starting without severe or fatal errors. During the update process, the server is stopped if running, then the update is performed, and a check is made to determine if the newly updated server starts without major errors. If it cannot start cleanly, the update will be backed out and the server returned to its prior state. See the **`revert-update`** tool for information on reverting an update.
validate-file-signature	Validate file signatures. For best results, file signatures should be validated by the same instance used to generate the file. However, it might be possible to validate signatures generated on other instances in a replicated topology.

Saving options in a file

PingDataGovernance Server supports the use of a tools properties file (`config/tools.properties` by default) to simplify command-line invocations by reading in a set of options for each tool from a text file.

Properties files are convenient when quickly testing PingDataGovernance Server in multiple environments.

Each property takes the form of a name-value pair that defines predetermined values for a tool's options.

PingDataGovernance Server supports the following types of properties:

- Default properties that apply to all command-line tools
- Tool-specific properties

Creating a tools properties file

You can set properties that apply to all tools or are tool-specific. These properties serve as defaults for the command-line options they represent.

Steps

1. Use a text editor to open the default tools properties file (`config/tools.properties)` or a different properties file.

   Note:

   If you use a file other than `config/tools.properties`, invoke the tool with the `--propertiesFilePath` option to specify the path to your properties file.

2. Set or change properties that apply to all tools.

   Use the standard Java properties file format (name=value) to set properties. For example, the following properties define a set of LDAP connection parameters.

```
hostname=server1.example.com
port=1389
bindDN=cn=Directory\ Manager
```

```
bindPassword=secret
baseDN=dc=example,dc=com
```

Note:

Properties files do not allow quotation marks of any kind around values.

Escape spaces and special characters.

Whenever you specify a path, do not use `~` to refer to the home directory. The server does not expand the `~` value when read from a properties file.

3. Set or change properties that apply to specific tools.

   Tool-specific properties start with the name of the tool followed by a period. These properties take precedence over properties that apply to all tools. The following example sets two ports: one that applies to all tools (`port=1389`) and a tool-specific one that **`ldapsearch`** uses instead (`ldapsearch.port=2389`).

   ```
   hostname=server1.example.com
   port=1389
   ldapsearch.port=2389
   bindDN=cn=Directory\ Manager
   ```

4. Save your changes and close the file.

Evaluation order

PingDataGovernance Server uses the following evaluation ordering to determine options for a given command-line tool.

- All options on the tool's command line take precedence over any options in any properties file.
- If you use the `--propertiesFilePath` option with no other options, PingDataGovernance Server takes its options from the specified properties file.
- If you use no options on the command line including the `--propertiesFilePath` option (and `--noPropertiesFile`), PingDataGovernance Server searches for the `tools.properties` file in *`<server-root>`*`/config/`.
- If no default properties file is found and a required option is missing, the tool generates an error.
- Tool-specific properties (for example, `ldapsearch.port=3389`) take precedence over general properties (for example, `port=1389`).

Consider this example properties file that is saved as *`<server-root>`*`/bin/tools.properties`:

```
hostname=server1.example.com
port=1389
bindDN=cn=Directory\ Manager
bindPassword=secret
```

PingDataGovernance Server locates a command-line option in a specific priority order.

1. All options presented with the tool on the command line take precedence over any options in any properties file. In the following example, the client request is run with the options specified on the command line (`--port` and `--baseDN`). The command uses the `bindDN` and `bindPassword` values specified in the properties file.

   ```
   $ bin/ldapsearch --port 2389 --baseDN ou=People,dc=example,dc=com \
     --propertiesFilePath bin/tools.properties “(objectclass=*)”
   ```

2. Next, if you specify the properties file using the `--propertiesFilePath` option and no other command-line options, PingDataGovernance Server uses the specified properties file as follows:

```
$ bin/ldapsearch --propertiesFilePath bin/tools.properties \
  “(objectclass=*)”
```

3. If no options are presented with the tool on the command line and the `--noPropertiesFile` option is not present, PingDataGovernance Server attempts to locate the default `tools.properties` file in the following location:

```
<server-root>/config/tools.properties
```

   If you move your `tools.properties` file from `<server-root>/bin` to the `<server-root>/config` directory, you can then run your tools as follows:

```
$ bin/ldapsearch "(objectclass=*)"
```

   You can configure PingDataGovernance Server so that it does not search for a properties file by using the `--noPropertiesFile` option. This option tells PingDataGovernance Server to use only those options specified on the command line. You cannot use the `--propertiesFilePath` and `--noPropertiesFile` options together.
4. If no default `tools.properties` file is found and no options are specified with the command-line tool, the tool generates an error for any missing arguments.

Running task-based tools

PingDataGovernance Server has a Tasks subsystem that allows you to schedule basic operations, such as **`backup`**, **`restore`**, **`rotate-log`**, **`schedule-exec-task`**, and **`stop-server`**. All task-based tools require the `--task` option that explicitly indicates the tool is to run as a task rather than in offline mode.

The following table shows the options you can use for task-based operations.

Options for task-based operations

Option	Description
`--task`	Indicates that the tool is invoked as a task. The `--task` option is required. If you invoke a tool as a task without this `--task` option, then a warning message is displayed stating that it must be used. If the `--task` option is provided but the tool was not given the appropriate set of authentication arguments to the server, then an error message is displayed and the tool exits with an error.
`--start` *`<startTime>`*	Indicates the date and time, expressed in the format 'YYYYMMDDhhmmss', when the operation is to start. A value of '0' causes the task to be scheduled for immediate execution. After the scheduled run, the tool exits immediately.
`--dependency` *`<taskID>`*	Specifies the ID of a task upon which this task depends. A task does not start execution until all its dependencies have completed execution. You can use this option multiple times in a single command.

Option	Description
`--failedDependencyAction` *`<action>`*	Specifies the action this task takes if one of its dependent tasks fail. Valid action values are: ▪ `CANCEL` (the default) Cancels the task. ▪ `DISABLE` Disables the task so that it is not eligible to run until you manually enable it again. ▪ `PROCESS` Runs the task.
`--startAlert`	Generates an administrative alert when the task starts running.
`--errorAlert`	Generates an administrative alert when the task fails to complete successfully.
`--successAlert`	Generates an administrative alert when the task completes successfully.
`--startNotify` *`<emailAddress>`*	Specifies an email address to notify when the task starts running. You can use this option multiple times in a single command.
`--completionNotify` *`<emailAddress>`*	Specifies an email address to notify when the task completes, regardless of whether it succeeded or failed. You can use this option multiple times in a single command.
`--errorNotify` *`<emailAddress>`*	Specifies an email address to notify if an error occurs when this task executes. You can use this option multiple times in a single command.
`--successNotify` *`<emailAddress>`*	Specifies an email address to notify when this task completes successfully. You can use this option multiple times in a single command.

Capture debugging data

For problems with PingDataGovernance Server or a supporting component, such as the Java Virtual Machine (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 resolve it.

For specific details, see the following topics:

- *Exporting policy data* on page 289
- *Enable detailed logging* on page 289
- *Visualizing a policy decision response* on page 291
- *Capture debugging data with the collect-support-data tool* on page 293

Exporting policy data

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.

About this task

Snapshots provide a convenient way to load policy data into a separate PingDataGovernance Policy Administration GUI instance.

To export policy data:

Steps

1. Go to **Branch Manager**.
2. Select the **Version Control** tab.
3. Click the name of the branch to export.
4. Click the branch's **Options** icon and select **Export Snapshot**.
   A snapshot file downloads to your computer.

Enable detailed logging

Enable detailed debug logging for troubleshooting.

> **Note:**
>
> This level of logging captures request and response data that contains potentially sensitive information. 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 policy decision point (PDP).

Regardless of whether PingDataGovernance Server is configured to evaluate a policy in Embedded or External mode, a policy-decision file logs every policy decision per request. This file is located at `PingDataGovernance/logs/policy-decision` and contains the following information:

Policy-decision response

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

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

Most recent policy decision

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

```
[08/May/2019:15:35:04.791 -0500] DECISION requestID=46
 correlationID="0349a205-6aeb-4bd6-923b-c777bcef2241"
product="Ping Identity Data Governance Server" instanceName="dgl"
 startupID="XNM9Hw==" threadID=140
from=[0:0:0:0:0:0:0:1]:49882 method=GET
 url="https://0:0:0:0:0:0:0:1:8443/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-ffle-4f80-970a-12100cdd7646" authorized="true"
 decision="PERMIT" decisionStatusCode="OKAY" adviceIds=""
adviceNames=""

{
  "id" : "734fc520-ffle-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 most recent policy decision, as the following example shows.

```
[08/May/2019:15:35:05.377 -0500] ADVICE requestID=46
 correlationID="0349a205-6aeb-4bd6-923b-c777bcef2241" product="Ping
 Identity
Data Governance Server" instanceName="dgl" startupID="XNRLuQ=="
 threadID=139 from=[0:0:0:0:0:0:0:1]:56475 method=GET url="https:
0:0:0:0:0:0:0:1:8443/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 \
  --add decision-response-view:evaluated-entities \
  --add decision-response-view:evaluation-log-with-attribute-values
```

Debug Trace logger

The Debug Trace logger records detailed information about the processing of HTTP requests and responses.

The following example enables the 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 the 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`.

Visualizing a policy decision response

Visualize a decision by selecting a recent decision or by copying and pasting a decision from a log.

Steps

1. Sign on to the PingDataGovernance Policy Administration GUI.

2. Choose a method for visualizing a decision.
   - Select a recent decision

     a. In the Policy Administration GUI, go to **Policies**.
     b. Click the **Decision Visualiser** tab.
     c. Click **Recent Decisions** and select a decision.
     d. Click **Visualise**.

     > **Note:**
     >
     > You can control the number of recent decisions that appear in the list as explained in *Setting the request list length for Decision Visualizer* on page 227.

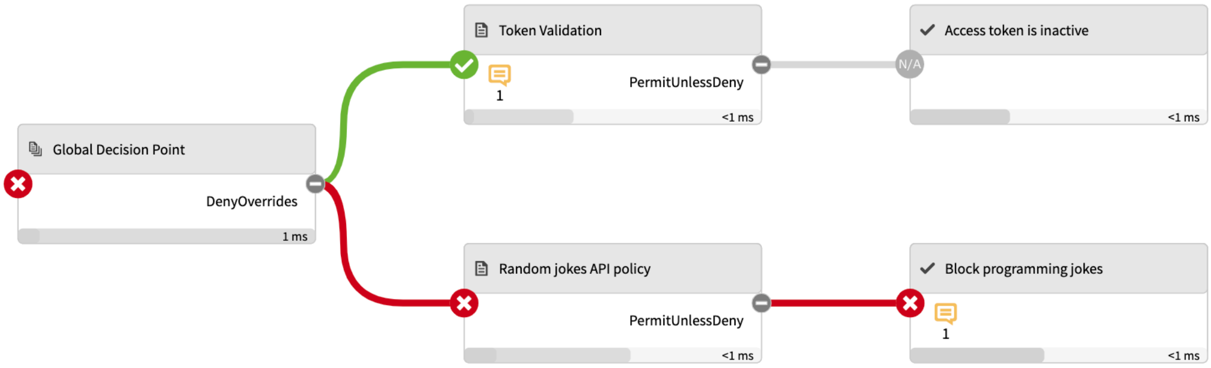
   - Copy and paste a decision from a log

     > **Note:**
     >
     > Before attempting to troubleshoot or trace a policy-decision response, ensure that the Policy Decision logger is enabled. For more information, see *Configuring PingDataGovernance logging*.
     >
     > Each policy-decision response is presented in JSON format. To view the details of a policy-decision response:

     a. From within the policy-decision file, copy the policy-decision response JSON.
     b. In the Policy Administration GUI, go to **Policies**.
     c. Click the **Decision Visualiser** tab.
     d. Click **Paste Logs**.
     e. In the field beneath **Paste Logs**, paste the policy-decision response JSON.
     f. Click **Visualise**.

Results

An interactive decision tree of your policies is displayed.



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

The following color-coded icons convey important information:

- A green check mark indicates that the request `permit` on the policy or rule.
- A red X indicates that the request `deny` on the policy or rule.
- A 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 to use for troubleshooting issues.

When you run `PingDataGovernance/bin/collect-support-data`, the tool generates a compressed file that can be attached to a message or report.

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 289. 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`

About the layout of the PingDataGovernance Server folders

The following table describes the contents of the PingDataGovernance Server distribution file. In addition, the table describes items created as you use PingDataGovernance Server.

PingDataGovernance Server directories, files, and tools

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.
`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.
`extensions`	Stores Server SDK extensions.
`ldif`	Serves as the default location for LDIF exports and imports.
`legal`	Stores any legal notices for dependent software used with PingDataGovernance Server.

Directories, files, and tools	Description
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.
tmp	Stores temporary files and directories used by the server, including extracted WAR files and compiled JSP files used by Web Application Extensions.
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 layout of the PingDataGovernance Policy Administration GUI folders

The following table describes the contents of the PingDataGovernance Policy Administration GUI distribution file.

PingDataGovernance Policy Administration GUI directories, files, and tools

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.

Directories, files, and tools	Description
`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.

PingDataGovernance Policy Administration Guide

Getting started

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

About this task

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

Steps

1. Sign on to the Policy Admin GUI.

   In demo environments, you can use the default credentials:

   - User name: `admin`
   - 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. Commit changes.

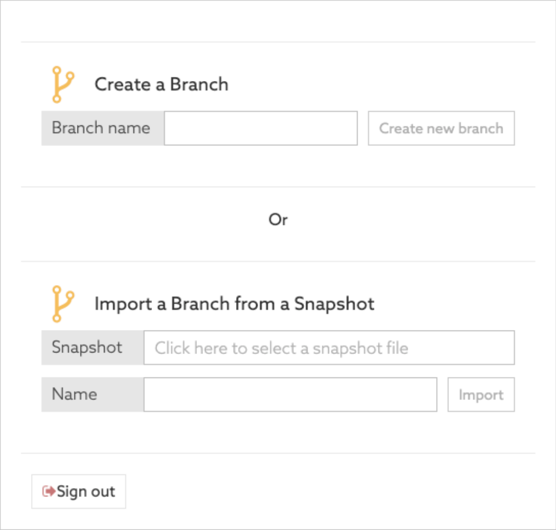
   This creates a *commit*, which is an immutable representation of the Trust Framework and Policies at a point in time.
7. Create a deployment package.

   This creates a file that can you deploy to PingDataGovernance Server instances across multiple environments.

Next steps

After you sign on to the Policy Admin GUI, the system prompts you 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.



Version control (Branch Manager)

Use the **Branch Manager** to manage your branches, commits, snapshots, and deployment packages.

Creating a new top-level branch

The PingDataGovernance Policy Administration GUI (Policy Admin GUI) allows you to create a new branch in two ways: using the startup window or the Branch Manager.

About this task

Note:

Branch names must be unique. No two branches in the Policy Admin GUI can share the same name.

Steps

1. Sign on to the Policy Admin GUI.
2. Choose how to create the branch per the following table.

To create a new top-level branch from	Do this
The startup window	Specify a **Branch name** and click **Create new branch**.
Branch Manager	From **Branch Manager#** **Version Control**, you can create a new root, or top-level, branch: **a.** From the **+** menu, select **Create new root branch**.

To create a new top-level branch from	Do this
	b. For the name, replace **Untitled** with a name for your new branch. **c.** Click **Save Branch**.

Creating a subbranch from a commit

Create a branch from a commit. For more information, see *Committing changes* on page 299.

About this task

This subbranch is a child of the branch from which the commit was selected. The subbranch shares the history and contents of the parent branch up to that commit.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the commit from which to branch.

   To branch from the latest uncommitted changes, make certain to commit before proceeding.
4. Click the three-line menu and select **Create new branch from commit**.
5. Specify a name for the branch.
6. Click **Save Branch**.

Results

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

Importing a branch

Import branches from previously-exported snapshot files to share and restore Trust Framework definitions and policies across users and environments.

About this task

ⓘ **Note:** A snapshot file contains all the entities and policies from an existing branch. You can share the file like any other file. For more information about creating snapshots, see *Generating snapshots* on page 300.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Click **+** and select **Import Snapshot**.
4. Select the appropriate snapshot file.
5. Specify a name for the branch.
6. Click **Import**.

Deleting a branch

Delete a branch to remove the branch, its history, and any commits created on it from the system.

About this task

You cannot delete a branch if a deployment package has been created from that branch.

> ⓘ **CAUTION:**
>
> This operation is irreversible.
>
> To recover data from a deleted branch, load a snapshot 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.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the branch to delete.
4. Click **Delete Branch**.

Merging branches

Merge branches to apply all of the changes made in the source branch to the target branch.

About this task

You can only merge committed branches.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the source branch.

   You can select top-level branches and subbranches.
4. Click **Set branch as Merge Source**.
5. Go to the target branch and click **Set branch as Merge Target**.

   With the source and target branches selected, the **Merge Branches** button should appear.
6. Click **Merge Branches**.

   The PingDataGovernance Policy Administration GUI, which is powered by Symphonic®, checks for merge conflicts.

   If no conflicts are found, the changes are merged from the source branch into the target branch. Your merge is complete, and you can skip the remaining step.

   If conflicts are found, complete the following step to resolve the conflicts.

7. Resolve conflicts.

   If an entity has changed in the both the source and target branches, the Policy Administration GUI flags a conflict. You must resolve the conflict for the merge to continue. Conflicts appear in the Merge Conflicts table.

   a. Click the **Info** icon in the **Show Differences** column.
   b. Compare the differences between the branches.
   c. Decide which change to keep and click either **Keep Source** or **Keep Target**.

      ⓘ **Note:**

      If you need all, or almost all, of the sections from one branch, you can click **Keep All Source** or **Keep All Target**. Then you can click into the Entity Difference of individual entities for fine-grained control.

   d. After you resolve all conflicts, close the entity difference box.

      The **Merge Conflicts** button becomes available.
   e. Click **Merge Conflicts**.

Reverting changes

To undo changes since the last commit, use the **Revert** button.

About this task

Each branch has a list of previous commits and **Uncommitted Changes**. To show the changes since the last commit, click the arrow to the left of the three-line icon in the **Uncommitted Changes** section.

ⓘ **Note:**

Reverting a change reverts all changes that have been made since that change as well. Make sure that you understand all the changes that will be reverted before reverting.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the branch with the uncommitted changes to revert.
4. Expand the **Uncommitted Changes** section by clicking the arrow to the left of the three-line icon to show all the changes that have happened since the last commit.

   To the right of each change is a **Revert** button.
5. Click the **Revert** button and confirm the revert.

Committing changes

To save your policy and Trust Framework changes, commit your changes.

About this task

After you finish building, testing, and analyzing your policies, commit the changes. Committed changes cannot be reverted.

With changes committed, you can create a deployment package from the commit. See *Creating a deployment package* on page 301.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the branch in which to put the commit.
4. Click **Commit New Changes**.

Generating snapshots

A snapshot contains all the details from a commit or from the **Uncommitted Changes** head. You can export a snapshot to import later.

Steps

1. Click **Branch Manager**.
2. Click **Version Control**.
3. Select the three-line icon for item to snapshot.
4. Click **Export Snapshot**.
5. Specify a name for the snapshot.
6. Click **Export**.

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 an imported new branch or merged into an existing branch.

Creating a partial export

Create a partial export to build an export snapshot of specifically selected entities from a combination of the Trust Framework, Policy Sets, and the Library set.

Steps

1. Click **Branch Manager**.
2. Click **Export Partial Snapshot**.
3. Select the desired items from the list on the left.
4. Click **Add selection to Snapshot** at the top of the pane on the left.

   This step adds the entity to the **Selected entities** list. The exported snapshot automatically includes all dependencies so you do not need to explicitly select each individual dependency.
5. Click **Export**.

Merging a partial snapshot

Merge a snapshot to add or update all of the entities into the current branch.

Steps

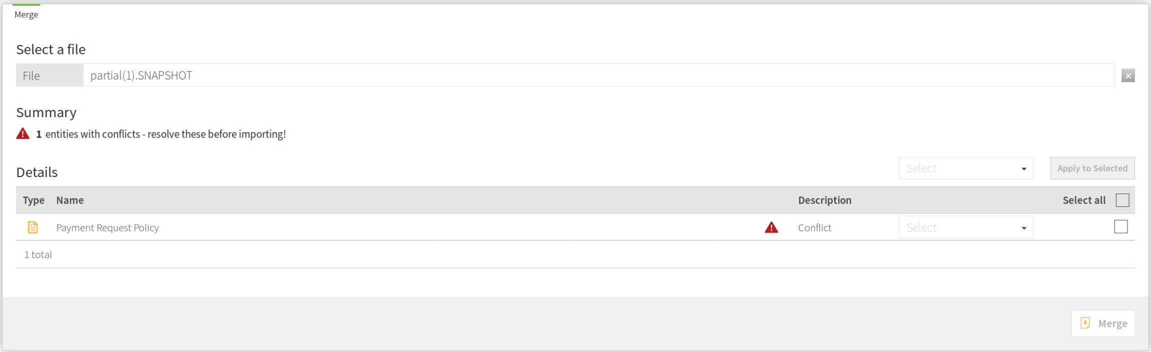
1. Click **Branch Manager**.
2. Click **Merge Snapshot**.
3. Select the appropriate snapshot file from your system.
4. Click **Merge**.

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.



Merge

Select a file

File partial(1).SNAPSHOT

Summary

1 entities with conflicts - resolve these before importing!

Details

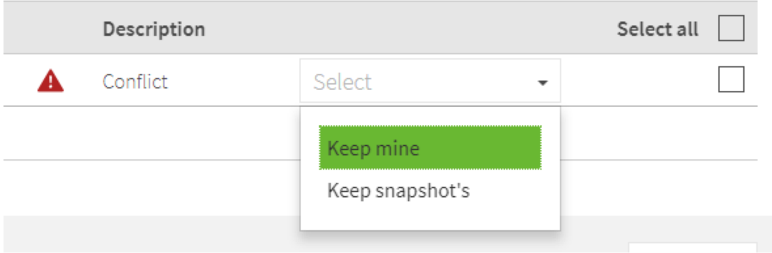
Select ▾ Apply to Selected

Type	Name		Description		Select all □
	Payment Request Policy	⚠	Conflict	Select ▾	□

1 total

Merge

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



	Description		Select all □
⚠	Conflict	Select ▾	□

Keep mine

Keep snapshot's

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

Creating a deployment package

Create a deployment package from committed changes.

About this task

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

Steps

1. Click **Branch Manager**.
2. Click **Deployment Packages**.
3. Click **+**.
4. Replace **Untitled** with a name for the deployment package.
5. Select a **Branch**, **Commit**, and **Policy Node** from which to generate the package.
6. Click **Create Package**.

   The package can be exported any number of times and will remain the same even if further changes are made to the branch.

   To export the deployment package, select the package and click **Export Package**.

Deleting a deployment package

Delete a deployment package to remove it from the **Packages** list.

Steps

1. Click **Branch Manager**.
2. Click **Deployment Packages**.
3. Select the package.
4. Click **Delete Package**.

Trust Framework

The Trust Framework tool lets you define all the entities within your organizations about which you want to build policies at a later time.

You must define anything you want to express in your policies in the Trust Framework. As a result, your policies are tightly coupled to the definitions in your Trust Framework, with strict restrictions on intermixing of values with differing data types.

Domains (PDP API only)

You need to define the organizational structure of any other organizations with which you intend to interact and, consequently, on which you want to specify authorization policies.

Define these organizations under **Trust Framework**, using the **Domains** section, which is available only on PDP API-enabled servers . Start with a relatively clean and simple domain ontology. You can extend it later if you need 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 *policy information providers* 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 plan to also use the service as a policy information provider, leave the **Service Type** as `None`.

Policy information providers

Setting up services as policy information providers makes use of various service connectors.

When you make a selection from the **Service Type** list, settings specific to the service appear. Settings that apply to all service endpoints also appear.

When a service returns a value to resolve an attribute, you can:

- Map the response to a type.
- Apply a processor to the response to transform that response or to extract a specific part of it.

  Use a processor when a service returns more information than is required or returns information that you must convert to a different format.

  For information about processors and how to combine multiple processors, see *Named value processors* on page 317.

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 receiving a successful response, the server cancels request. If the server has retries configured, the server attempts the request 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 you have Exponential Backoff 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±100ms, 2000±100ms, 4000±100ms, and so on.

Value Processors

Specify an optional processor to transform the resolved value. If you do not need to preprocess the response, leave the **Processor** set to `None`. Otherwise, select the required inline `SpEL`, `XPath`, or `JSON Path` processor. Alternatively, add a library processor by selecting `reference`. If the referenced processor has no processing steps defined, it is the equivalent of setting **Processor** to `None`. See *Named value processors* on page 317.

Value Settings

These are required settings that are applied and describe the resolved value after any preprocessing. Set the **Type** field to `String` for plain text, or `JSON` or `XML` for those types, and so forth.

Secret

Select the **Secret** check box to mark a service's response as secret and ensure this data is never leaked to log files.

HTTP services

The policy decision point (PDP) can perform requests to HTTP services. These requests can send and receive Text, JSON, and XML content.

HTTP authentication is supported by using a simple user name and password, or by using an OAuth2 token.

You can send custom headers with any request, which you can make dynamically in various ways by interpolating attribute values into various parameters. See *Attribute interpolation* on page 315.

Core settings

- URL

  URL for the REST endpoint that the PDP accesses. The Policy Manager can interpolate attributes 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. The Policy Manager can interpolate attributes anywhere in the body with no escaping.

Authentication

The Authentication drop-down lists the following HTTP authentication types, which correspond to an authorization header sent with the request:

- None

  Default value that indicates the PDP sends no authorization header.
- Basic

  Reveals the choices for attributes whose values function as the user name and password of an HTTP request with basic authentication.
- OAuth2

  Reveals a token selector. The PDP sends the selected attribute as the authorization token in an HTTP request with bearer authentication.

Headers

You can add any number of custom headers 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 next to a header value.

Certificate validation

With certificate validation, you can define TLS and Mutual-TLS (M-TLS) certificates and keys when connecting to the TLS (or SSL) based service.

When using external PDP mode, you can declare local file-based trust stores and key stores by providing an options file during setup. See *Specifying custom configuration with an options file* on page 213.

When using embedded PDP mode, you do this by assigning Trust Manager Providers and Key Manager Providers to the Policy Decision Service. See *Use policies in a production environment* on page 233.

Server (TLS)

Server (TLS) settings apply when validating the certificate or certificate chain sent from the server. You have three options when validating a server certificate.

- `No Validation`

  Skips validating the server certificates and initiates connection without any restriction.
- `Default`

  This option is the default for Server (TLS).

  Uses the default trust store provided by the runtime environment.

  Use this if you are trying to connect to a service that has a certificate issued from a valid certificate authority.
- `Custom`

  Allows the user to define a custom certificate or certificate chain that is stored in a trust store.

  Custom trust store settings:

  - Source

    Trust store source. Currently, it only supports file-based trust stores.
  - Trust store name

    The name given to the trust store in `configuration.yml`.
  - Alias

    Certificates in the trust stores are mapped by alias. You must set the alias in the trust store to specify which certificate to use for validation.

    Attributes can be interpolated anywhere in the value.
  - Alias password

    If the certificate is password-protected, it might need to provide the password.

    Attributes can be interpolated anywhere in the value.

Client (M-TLS)

Some services might require the client to provide a client certificate when initializing the connection. To provide a client certificate, enable this setting and provide a custom key store to be sent to the service.

Custom key store settings:

- Source

  Key store source. Currently, it only supports file-based key stores.
- Key store name

  The name given to the key store in `configuration.yml`.
- Alias

  Key-value pairs and the certificate entry in the key stores are mapped by alias. You must set the alias in the key store to specify which entry to use for validation.

  Attributes can be interpolated anywhere in the value.

- Alias password

  If the entry is password-protected, it might need to provide the password.

  Attributes can be interpolated anywhere in the value.

LDAP services

The policy decision point (PDP) can make LDAP queries to retrieve information.

You can make requests dynamic by interpolating attribute values into different parameters. See *Attribute interpolation* on page 315.

Configuration

Specify the following settings 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. For example:

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

Username / Bind DN and Password

The user or bind credentials for the LDAP server. For example:

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

Use SSL

If the LDAP server is secured using SSL, enable this setting.

Enabling this setting populates the Certificate Validation section, which is useful when configuring TLS and M-TLS certificates. For more information, see *Certificate validation* on page 304.

Search Base DN / LDAP filter

These settings define the LDAP query. For example:

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

Results

Because the server converts the result of an LDAP query to an XML document, you must set the service value type 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>
```

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

Camel services

In addition to retrieving information from HTTP and LDAP policy information providers, 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.

Overview

Configure Camel components 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. See the documentation on the Camel website for the particular component that you want to use.

You can make requests dynamic by interpolating attribute values into different parameters. See *Attribute interpolation* on page 315.

URI

URIs identify Camel endpoints. 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. The system can interpolate attribute values anywhere in the field.

Headers

You can send additional information to the external policy information provider 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. The system can interpolate attribute values anywhere in the field.

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. The system can interpolate attribute values anywhere in the field.

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*.

ⓘ **Warning:**

The system cannot interpolate attribute values into the configuration.

ⓘ **Note:**

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.

Attributes

Attributes provide the context that enables fine-grained policies.

Attribute values come from a multitude of sources. You can use the original values or modify the values. You can then use the final values in other attributes, *Named conditions* on page 317, or rules.

The system resolves an attribute only when its value is required as part of the decision request evaluation. For example, if a rule checks whether a customer's device "Risk Score" is high, then the system only attempts to resolve the attribute corresponding to "Risk Score" if that rule is required.

Creating an attribute

Create attributes using the business terms that business users and policy writers already understand.

About this task

Consider the manner in which you will structure the attributes and the naming conventions that you will use. You want policy writers to be able to build and manage policies without developing a deep understanding of the often-complex underlying data endpoints or data manipulation.

Steps

1. Click **Trust Framework**.
2. Click **Attributes**.
3. Click **+**.

Attribute name, description, and location

You can give attributes any name that is unique and does not contain a period (.).

To ensure that the system can interpolate the attribute, avoid the following characters:

- {
- }
- |

You can give the attribute a description to help policy editors understand the attribute's purpose. This description is only displayed when a user navigates to the attribute.

You can change the location of an attribute in the attribute tree using the **Parent** field.

Resolvers

Use resolvers to define where the initial data for an attribute comes from.

An attribute can have multiple resolvers, and the resolvers can be conditional. In addition, you can add a processor to a resolver to modify the resolver's value before the attribute uses it.

For more information, see:

- *Resolver types* on page 308
- *Conditional resolvers* on page 310
- *Value processing for a resolver* on page 310

Resolver types

Each attribute can have one or more resolver types.

The resolvers apply in the order listed. You can reorder the resolver types by dragging and dropping them to the appropriate position.

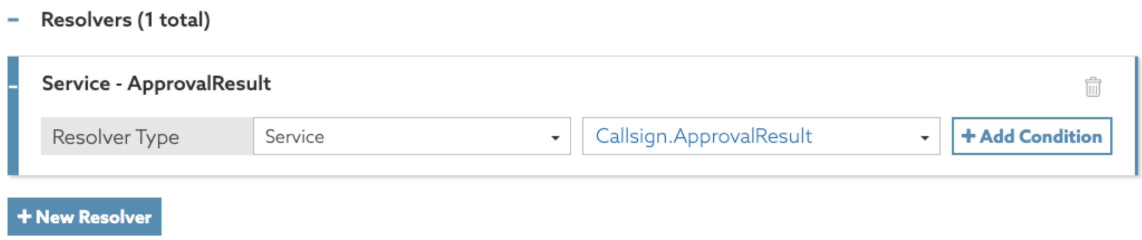
The following table describe the various resolver types.

Resolver type	Description
Request	This resolver type looks inside the authorization request itself to determine whether the attribute has been provided by the caller. Specify the full name of the attribute, including any parents, in the request.
Constant	This resolver setting takes a constant value defined on the resolver itself. The type and value of the constant are required. **Note:** As with all other resolved values, constants undergo any value processing defined on the attribute. To define a constant that does not undergo value processing, consider using a *Default value* on page 315.
Service	This resolver setting uses a **Trust Framework# Services** endpoint to invoke the service at runtime to resolve the attribute. The service might rely on other attributes being supplied to invoke the service. The PDP handles this process automatically.
Attribute	PingDataGovernance Server can also resolve attributes from other attributes. This ability 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, you could create the Customer.Name.Surname attribute 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 you can use without additional configuration. For example, the CurrentDateTime returns the current system datetime according to the **Type** defined for the attribute..
Configuration Key	The policy engine can resolve attribute values using policy configuration keys. When using external PDP mode, you can declare local file-based trust stores and key stores by providing an options file during setup. See *Specifying custom configuration with an options file* on page 213. When using embedded PDP mode, you do this by creating Policy Configuration Keys in the Policy Decision Service. See *Use policies in a production environment* on page 233.

Conditional resolvers

All resolver types support the ability to add conditional logic so that the system invokes the resolver only under certain defined conditions.

To add a conditional logic to a resolver, from the three-line icon beside the appropriate resolver, select **Add Condition**. You can then add a comparison or named condition.

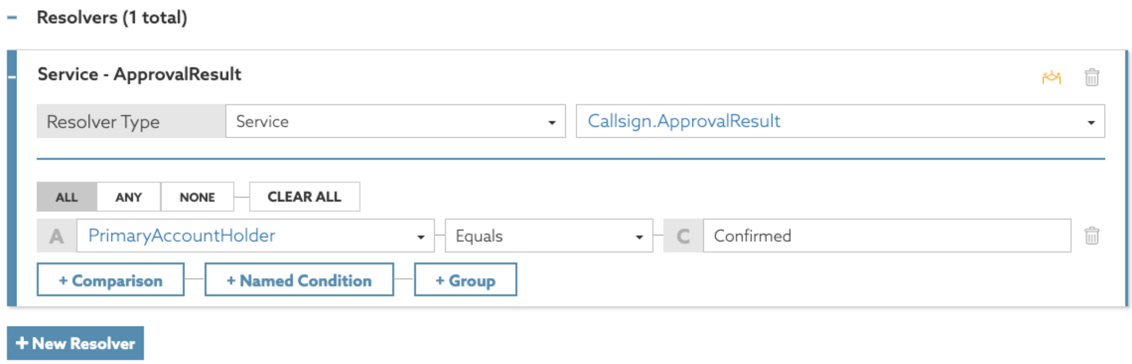


Resolvers (1 total)

Service - ApprovalResult

Resolver Type: Service | Callsign.ApprovalResult

In the following example, the service resolver `Callsign.ApprovalResult` applies only when the attribute PrimaryAccountHolder has a value of Confirmed.



Resolvers (1 total)

Service - ApprovalResult

Resolver Type: Service | Callsign.ApprovalResult

A PrimaryAccountHolder | Equals | C Confirmed

You can combine multiple conditions for a resolver using **ALL, ANY**, or **NONE**. To allow more permutations, create subgroups by clicking **+ Group**.

Value processing for a resolver

Use value processing for a resolver to modify data before using that data as the attribute's final value.

To add or remove a processor to a resolver, within the resolver definition, click the three-line icon in the upper-right corner and choose **Add Processing** or **Remove Processing**.

For information about how to define a processor, see *Value processing* on page 317.

The following examples show how you might use these resolvers.

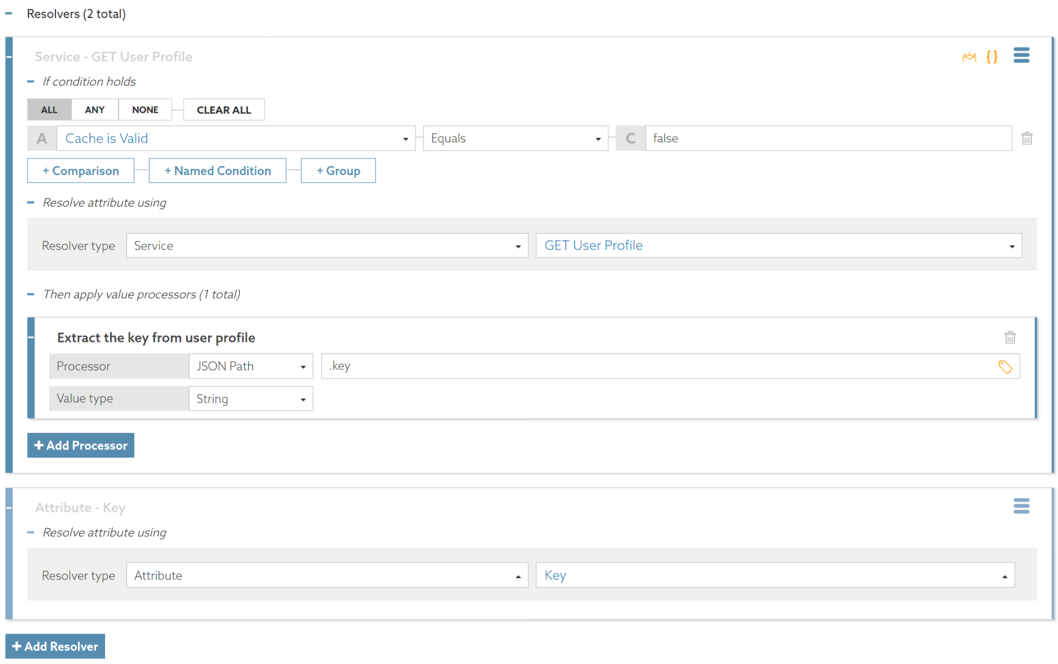
If you expect responses from different resolved sources to vary, you can add a processor to the resolvers to normalize the output. In this example, the attribute's value can come from one of the following resolvers:

- A service named `GET User Profile`

  With this resolver, if the `Cache is Valid` attribute is false, the resolver calls the `GET User Profile` service and uses a JSON Path processor to extract the key from the profile JSON.
- An attribute named `Key`

  In the second resolver, the attribute value comes from the `Key` attribute, and the value requires no processing.

The following image shows the resolvers. The resolvers apply in the order shown.



Resolvers (2 total)

Service - GET User Profile

If condition holds

A Cache is Valid | Equals | C false

Resolve attribute using

Resolver type: Service | GET User Profile

Then apply value processors (1 total)

Extract the key from user profile

Processor: JSON Path | .key

Value type: String

Attribute - Key

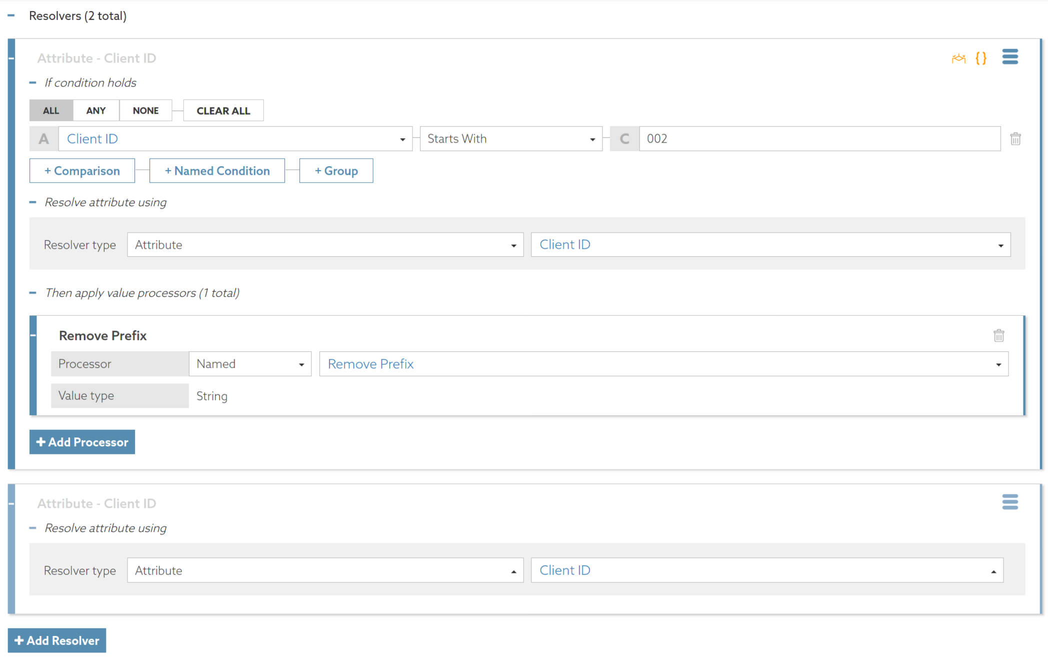
Resolve attribute using

Resolver type: Attribute | Key

This example uses a condition and a processor together to resolve an attribute that might have a prefix. The attribute has two resolvers:

- The first resolver has a condition to check whether the `Client ID` attribute has a prefix of `002`. If so, the value processor removes the prefix.
- The second resolver has no condition and passes the `Client ID` attribute value through with no processing.

The following image shows the resolvers. The resolvers apply in the order shown.



Resolvers (2 total)

Attribute - Client ID

If condition holds

A Client ID | Starts With | C 002

Resolve attribute using

Resolver type: Attribute | Client ID

Then apply value processors (1 total)

Remove Prefix

Processor: Named | Remove Prefix

Value type: String

Attribute - Client ID

Resolve attribute using

Resolver type: Attribute | Client ID

Attribute caching

The policy decision point (PDP) and the PingDataGovernance Policy Administration GUI support caching for attributes. The ability to cache resolved attributes can deliver significant performance gains for the PDP.

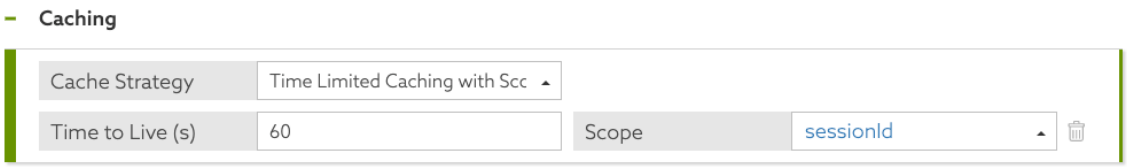
Carefully consider this concept to ensure optimum configuration.

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

Attribute caching can be indefinite or time-limited, with or without the scope of another attribute value.

With time-limited caching, you set the duration for which the cache lives (**Time to Live**) before it expires.

With **Scope** set to an attribute, if the value of that attribute changes, the system invalidates the cache for the attribute you are defining. In the example below, as long as the `sessionId` value remains the same, the value of the attribute you are defining is cached. When the `sessionId` changes, the system invalidates the cache and uses normal resolution.



– Caching

Cache Strategy: Time Limited Caching with Scc

Time to Live (s): 60

Scope: sessionId

If the attribute does not exist in the cache, the PDP resolves the attribute automatically by using the appropriate attribute resolvers and then adds it to the cache. All subsequent attribute usages use the cached value until it expires from the cache, which results in another attribute resolution.

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

Value processing for an attribute

See *Value Processors* on page 303 in *Services* on page 302.

Value settings

Every attribute has a defined data type that constrains the set of allowable values and provides a predictable behavior model for value processing and other data transformations.

Catching type inconsistencies early aids building and testing the Trust Framework. The primary types for accepting data into the system and for producing output data are JSON, XML, and UTF-8 text (known as String). The remaining types are used within a Trust Framework for more fine-grained data processing. All data types have conversions to and from a canonical String representation. Conversion of other formats, such as alternative date or time representations, requires the use of user-defined value processing. See *Value processing for an attribute* on page 312.

Examples of type conversions when data enters the policy decision point (PDP) include:

- Attribute default values you define in the user interface are textual. The system converts these to the type defined by the attribute before use.
- Attributes might take their values from fields in the decision request, which are again textual. The system converts the value to the type defined by the attribute before use.
- The PDP might invoke external services to retrieve data. Typical response formats are JSON, XML and String. JSON Path or XPath value processing can extract components of a response, typically as text, which the system then converts to the types defined by an attribute before use.

Examples of type conversions when exporting data from the PDP include:

- Building a request for a service invocation. Attributes might be request parameters directly or might be used in *Attribute interpolation* on page 315. In both cases, the system uses the canonical conversion to a String format.
- Adding attribute data to Obligations or Advice, either directly or through Attribute Interpolation. Again, the system uses the canonical conversion to String format.
- In all logging and response data that includes attribute values, the system renders those values using their canonical String representations.

The following table lists the data types.

Data type	Description
Boolean	A simple true or false. True can be represented in textual form, such as in default values or decision request parameters, as `true`, `yes` or `1`. False can be represented by `false`, `no` or `0`. Case is insignificant. In value processing contexts such as SpEL expressions, the value is a `java.lang.Boolean` instance.
Number	A numeric value. Decimal integers and reals are supported, including scientific notation. In value processing contexts, the value is a `java.math.BigDecimal` instance.
Date	A date, such as "23 April 2020". The textual representation is ISO-8601; for example, `2020-04-23`. In value processing contexts, the value is a `java.time.LocalDate`. *Date* values can be converted to the following types: ▪ **Date Time** (the time component becomes `00:00:00`) ▪ **Zoned Date Time** (the time zone is assumed to be UTC)
Time	A time of day, such as "4:15pm and 30 seconds". The textual representation is ISO-8601. The maximum resolution is microsecond. For example, `16:15:30`, `16:15:30.783`, and `16:15:30.783239` are all valid. In value processing contexts, the value is a `java.time.LocalTime`. **Time** values cannot be converted to other types.

Data type	Description
Date Time	A date and time of day, such as "4:15pm and 30 seconds on 23 April 2020". The textual representation is ISO-8601. The maximum resolution is microseconds. For example, `2020-04-23T16:15:30` or `2020-04-23T16:15:30.783239`. In value processing contexts, the value is a `java.time.LocalDateTime`. **Date Time** values can be converted to the following types: ▪ **Date** and **Time** (dropping the appropriate information in each case) ▪ **Zoned Date Time** (the time zone is assumed to be UTC)
Zoned Date Time	A date and time of day with a time zone expressed as an offset from UTC. The textual representation is ISO-8601; for example, `2020-04-23T16:15:30.783+01:00`. In value processing contexts, the value is a `java.time.ZonedDateTime`. **Zoned Date Time** values can be converted to the following types, dropping information in each case: ▪ **Date Time** ▪ **Date** ▪ **Time**
Duration	A time duration expressible in seconds or a fraction thereof. The textual representation is ISO-8601; for example: ▪ `PT3H` for 3 hours ▪ `PT2M45.836S` for 2 minutes and 45.836 seconds In value processing contexts, the value is a `java.time.Duration`. **Duration** values cannot be converted to other types.
Period	A time period expressible in calendric units such as a number of days or months. The textual representation is ISO-8601; for example: ▪ `P9Y` for 9 years ▪ `P3M2D` for 3 months and 2 days In value processing contexts, the value is a `java.time.Period`. **Period** values cannot be converted to other types.

Data type	Description
JSON	A JSON document. This type is most useful for bringing data into and out of the PDP. It is the only type that is subject to JSON Path value processors. The textual representation is JSON. In value processing contexts, the value is a `java.util.Map` or `java.util.Collection`.
XML	An XML document. This type is most useful for bringing data into and out of the PDP. It is the only type that is subject to XPath value processors. The textual representation is XML. In value processing contexts, the value is a `org.w3c.Document`.
Collection	An ordered collection of other value types. Only valid value types as described here can be members of collections. JSON-formatted arrays are valid textual representations of collections. In value processing contexts, a collection is a `java.util.Collection`; however, the objects contained are of an internal type. Use only the `get()` method to retrieve items by zero-based integer index.
String	All other data is interpreted as UTF-8 text, stored internally as UTF-16. In value processing contexts, these values are `java.lang.String`.

The legacy **Date Time** and **Time Period** types are ambiguous unions of the types described above. They are retained for backward compatibility only. For new Trust Frameworks, use the more specific types.

Default value

You can give attributes an optional default value in the event that the attribute cannot be resolved.

In addition, you can use a default value to encode constant attributes within the Trust Framework by not setting any resolvers and thus always resolving to the default value.

Attribute interpolation

With attribute interpolation, you reference an attribute in a field. The system resolves the value of the referenced attribute, replacing the reference with the value itself.

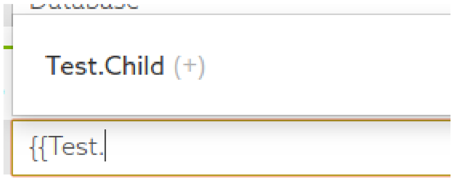
About this task

You can use attribute interpolation in any field that has the label icon, shown below.

🏷

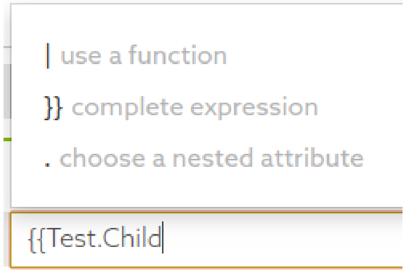
Steps

1. To reference an attribute in one of these fields, type two open curly brackets (`{{`) to open the attribute tree menu. Continue typing the full path to the attribute or select each level of the attribute in the attribute tree menu.



{{Test.

2. Complete the reference by typing two close curly brackets (`}}`) or by selecting the **}} complete expression** item from the attribute tree menu.



{{Test.Child

Actions

Actions represent arbitrary values that a typical authorization request might ask to perform on a specific resource, such as view or update.

Common actions you might want to configure in the PingDataGovernance Policy Administration GUI typically configures the following actions are:

- **`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.

Defining trust levels has the following distinct parts:

- *Identity properties* – Arbitrary properties that can relate to specific IdPs
- *Identity providers*
- *Identity classifications* – Levels of classifications

Identity properties

Use the **Identity Properties** window to define objects and elements to attach to specific identity providers (IdPs).

You use these properties later to map IdPs to specific identity classification levels.

Identity providers

Use the **Identity Providers** window to define different identity providers (IdPs) and to attach identity properties to them.

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

Identity classifications

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

For each classification level, attach the properties that an identity provider (IdP) must have to be in that 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 write 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.

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

Named value processors

Use named value processors to create reusable value processing logic.

Extracting this logic into these reusable components helps abstract some of the complexity in *Services* on page 302 and also reduces repetition. You can still create inline Value Processors that co-exist with Named Value Processors.

To use named value processors, click **Processors**.

Value processing

Use value processing to optionally apply preprocessing steps to returned responses from *Attributes* on page 308 or *Services* on page 302 to transform the resolved value. The PingDataGovernance Policy Administration GUI, which is powered by Symphonic®, currently supports three Value Processors:

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

You can combine these processors to form a chain of processors.

All processors have a type that indicates what the output data type should be after applying the expression. You can use a `reference` processor to refer to a reusable Named Value Processor.

JSON Path

With JSON Path, you can extract data from JSON objects. For example, assume we have a service that resolves to the following JSON.

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

To extract the `price` fields of all requested items, we set the Value Processor to **JSON Path** with the expression `$.requestedItems[*].price`.

For more information about JSON Path expressions, see *https://goessner.net/articles/JsonPath/*.

XPath

XPath is the XML-equivalent for JSON Path and follows a very similar syntax. For more information about XPath expressions, see the *XPath tutorial* on w3schools.com.

ⓘ **Note:**

The Policy Administration GUI only supports the use of *XPath 1.0*. Functions added in later versions are not available.

SpEL

With the Spring Expression Language, you can perform more complicated data processing. Expressions are applied directly to the resolved value. For example, assume you want to search for a substring that matches the following regular expression.

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

You then set the processor to **SpEL** and set the expression to this following text.

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

Attribute values can be interpolated into the SpEL expression directly using curly brackets, which can be useful if you want to combine multiple attribute values into a single value (see *Attribute interpolation* on page 315):

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

For information about the Spring Expression language, see the official *Spring Framework* docs.

For information about the Java classes available for SpEL processing, see *Configuring SpEL Java classes for value processing* on page 225.

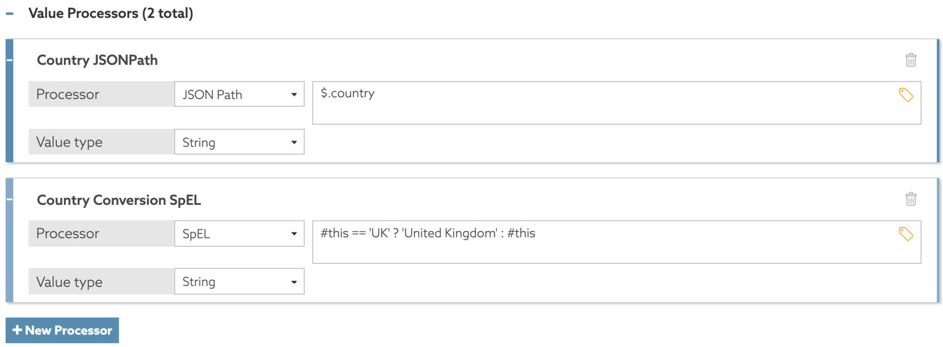
Chained processors

You can chain processors together to combine data preprocessing steps.

For example, you can extract data using JSONPath and then apply a SpEL processor to the extracted data. Assume you have a service that resolves to the following JSON response.

```
{
   "name": "Joe Bloggs",
   "city": "London",
   "country": "UK"
}
```

You have a requirement to extract the country and transform the value to `United Kingdom` whenever the current value is `UK`. You would add a JSONPath processor to select the country followed by a SpEL expression to transform the selection, as shown in the following figure.



Value Processors (2 total)

Country JSONPath

Processor: JSON Path — `$.country`

Value type: String

Country Conversion SpEL

Processor: SpEL — `#this == 'UK' ? 'United Kingdom' : #this`

Value type: String

Reusing chained processors

You can make a chained processor reusable by creating it as a named value processor. Then you can construct more complex processor chains made up of those named value processors.

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 system evaluates the request , you will see the following set of result tabs:

Request

Shows the actual JSON request 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 executed as part of the test

Services

Contains an expandable list of all services executed as part of the test

Policy management

The Policy Manager provides the tools to implement fine-grained and dynamic, access-control policies, allowing you to govern the use of your organization's services and data.

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 might instead be, "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 they say they are, 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?"

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

Policy sets, policies, and rules

The PingDataGovernance Policy Administration GUI reflects the structure of grouping rules for access control with three types of entities and the relationship between them. The entities are 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 users access certain resources.

You can group these rules together naturally, so you can understand them without focusing on all of them at the same time. 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 triggered with each attempt to access a set of restricted resources.

This structure is inherent in the problem domain of resource-access control. This section examines the different entity types, discusses how they are work together, and provides an overview of their properties.

Policies and policy sets

To view the Policy Manager, click **Policies**.

The Policy Manager organizes policy nodes in a tree structure within the navigation panel on the left side of the page. Add 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

Create policies and policy sets to define the circumstances under which users access certain resources.

Steps

1. Click **Policies**.
2. Click **+**.
3. Select **Add Policy Set** or **Add Policy**, as appropriate.

   You can name policies and policy sets 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 adequately represent the operational policies of the organization.
4. Click **Save changes**.

Example

In the following example, the policy name is `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.

Inspector
My Basic Policy
Disabled □

Adding targets to a policy

Add targets to identify the requests to which the policy applies. If no targets are attached to a policy, the policy applies to all requests. To make a policy only apply for all requests to a certain database, for example, add the database domain as a target.

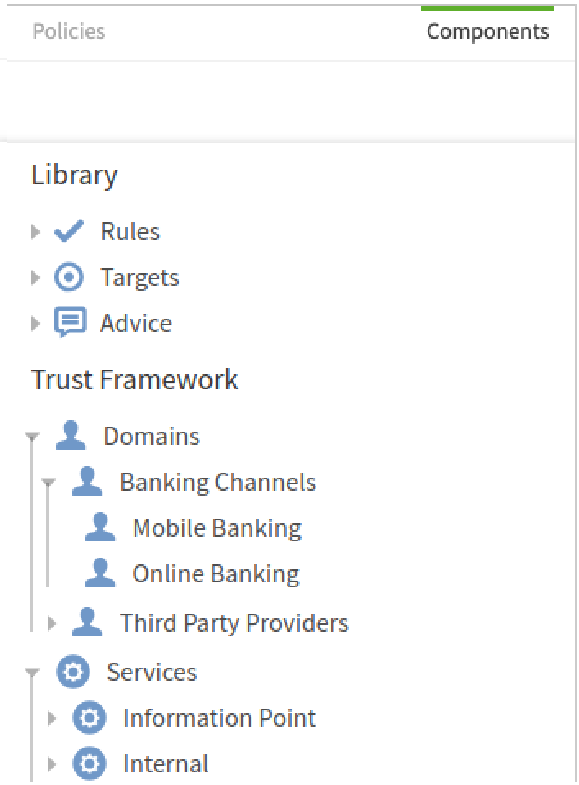
Steps

1. Go to the policy where you want to add targets.
2. Click the **+** next to **Applies to**.

3. In the left pane, click **Components**.

The list of components includes the items you created in the Trust Framework. Drag the appropriate domains, services, identity classes, and actions from the components to the **Applies to** target section on the policy.

For example, to target **Mobile Banking** requests, drag that domain in. 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.

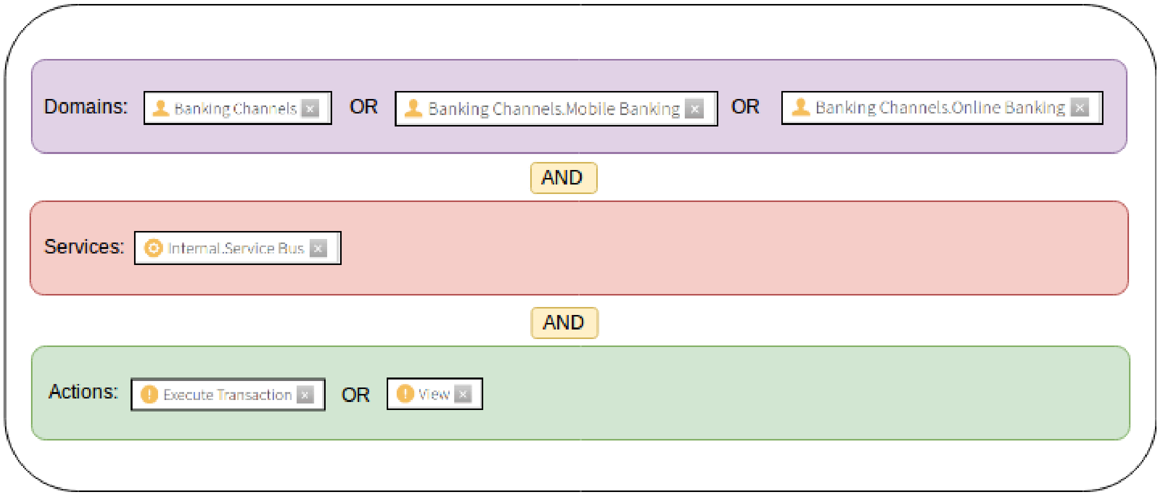


4. Click **Save changes**.

Example

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

The following graph shows how the server evaluates the group of targets.



Domains: Banking Channels OR Banking Channels.Mobile Banking OR Banking Channels.Online Banking

AND

Services: Internal.Service Bus

AND

Actions: Execute Transaction OR View

Conditional targets (applies when)
You can use conditional targets to extend the capability of the "Applies to" concept.

Conditional targets extend the capability of the "Applies to" concept because they:

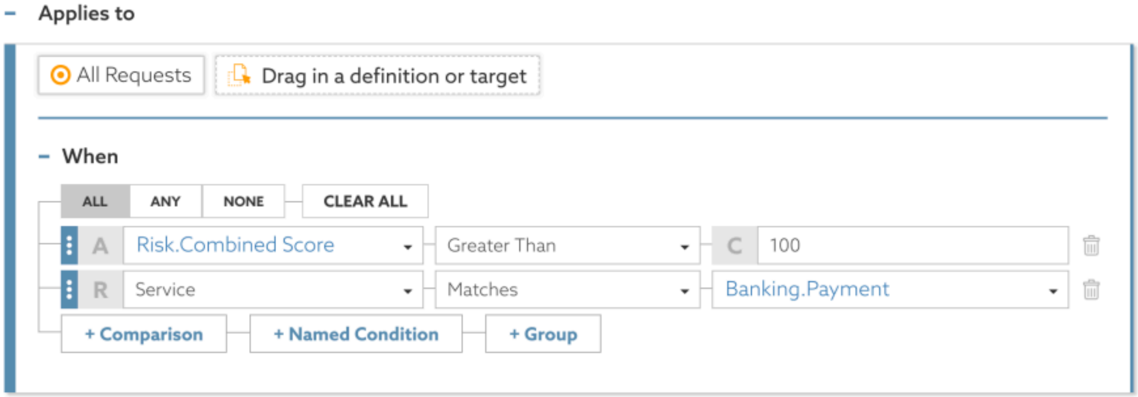
- Permit the interweaving of targets with other conditional logic.
- Allow standalone logic to determine if and when a policy or rule applies.

To enable this functionality, click **Applies to** and then **When**.

You can include the following types of conditions 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 show a **Named Condition** drop-down list that displays named conditions.

The following image provides an example.



Applies to

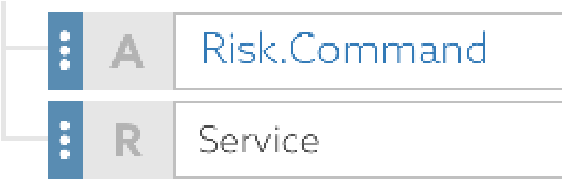
All Requests | Drag in a definition or target

When

A Risk.Combined Score | Greater Than | C 100

R Service | Matches | Banking.Payment

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



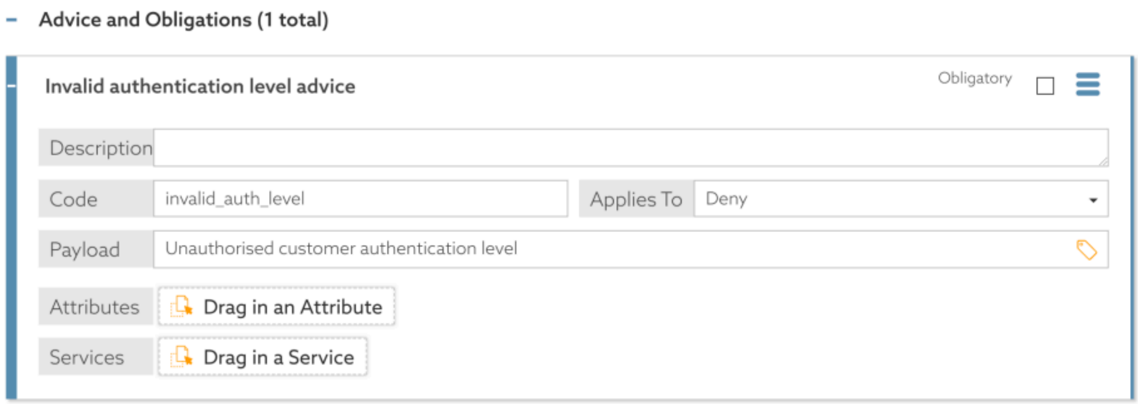
A Risk.Command

R Service

Advice

An advice is additional information you can attach to a decision response.

An advice returns to the governance engine so that, depending on the evaluation response from the policy, PingDataGovernance can take the appropriate action. If you have a policy 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, then PingDataGovernance can send details about the reason for denying access.


– Advice and Obligations (1 total)
Invalid authentication level advice
Obligatory □
Drag in an Attribute
Drag in a Service

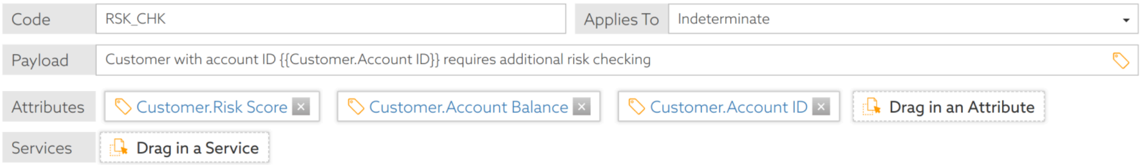
Description:
Code: invalid_auth_level
Applies To: Deny
Payload: Unauthorised customer authentication level
Attributes:
Services:

To indicate that the final decision applies only if an advice can be fulfilled, mark the advice 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, PingDataGovernance uses it 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.



Code: RSK_CHK
Applies To: Indeterminate
Payload: Customer with account ID {{Customer.Account ID}} requires additional risk checking
Attributes: Customer.Risk Score, Customer.Account Balance, Customer.Account ID
Services:

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 PingDataGovernance cannot fulfill a non-obligatory advice, the server logs an error, but the client's requested operation continues.

Property	Description
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 `permit` or `deny`, that include the advice with the policy result.
Payload	Set of parameters governing the actions that the advice performs when PingDataGovernance applies the advice. The appropriate payload value depends on the advice type.

PingDataGovernance Server supports the following advice types:

- *Add Filter* on page 244
- *Combine SCIM Search Authorizations* on page 245
- *Denied Reason* on page 245
- *Exclude Attributes* on page 245
- *Filter Response* on page 246
- *Include Attributes* on page 247
- *Modify Attributes* on page 248
- *Modify Headers* on page 248
- *Modify Query* on page 248
- *Regex Replace Attributes* on page 249

To develop custom advice types, use the Server SDK.

ⓘ **Note:**

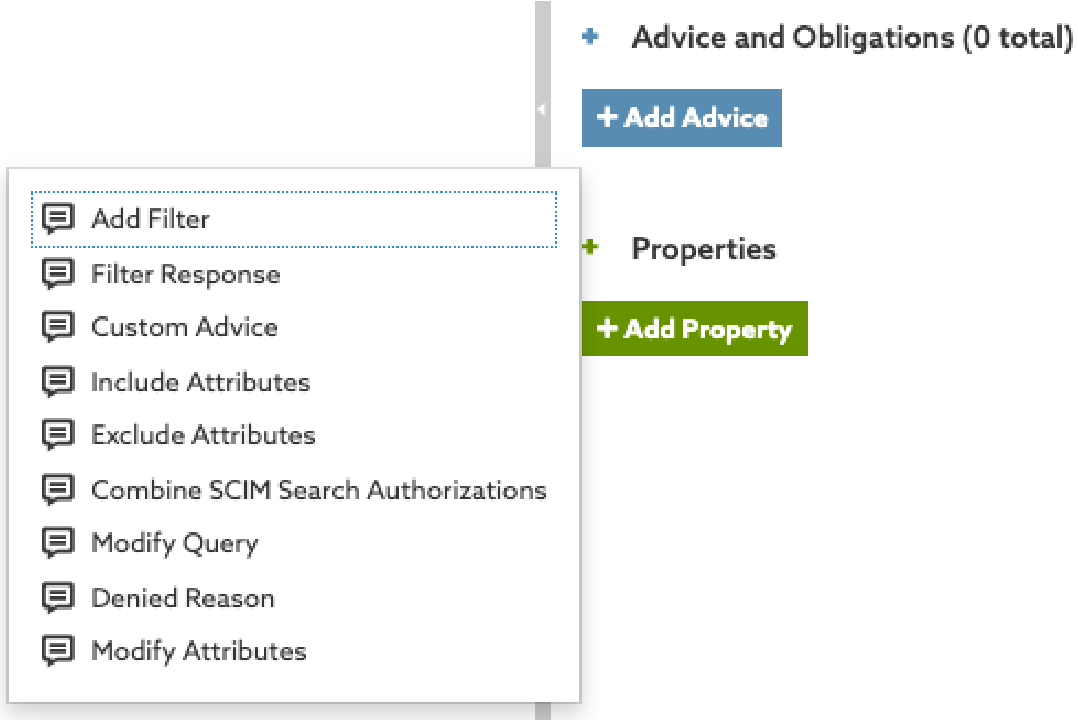
Many advice types let you use the JSONPath expression language to specify JSON field paths. To experiment with JSONPath, use the *Jayway JSONPath Evaluator*

.

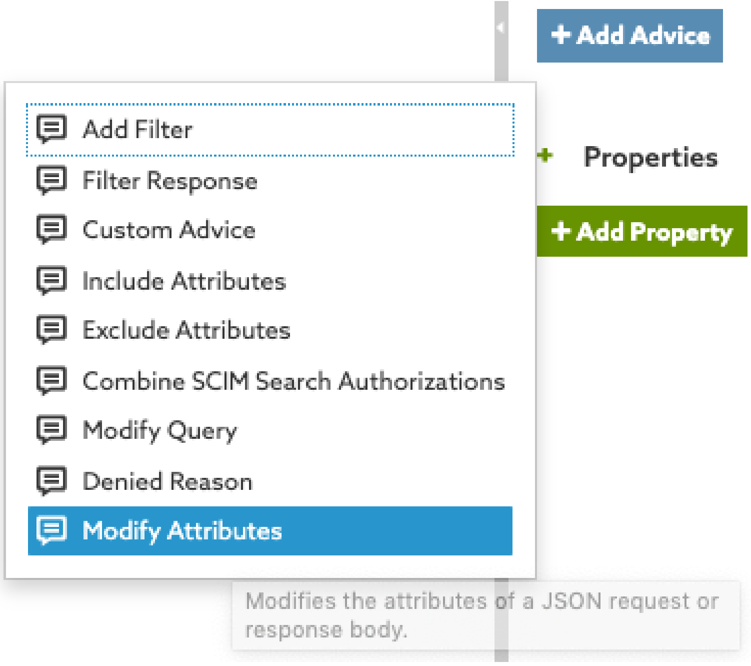
Provided advice

The PingDataGovernance Policy Administration GUI comes with preconfigured advice types that are also in PingDataGovernance Server.

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 **+ Add Advice**.

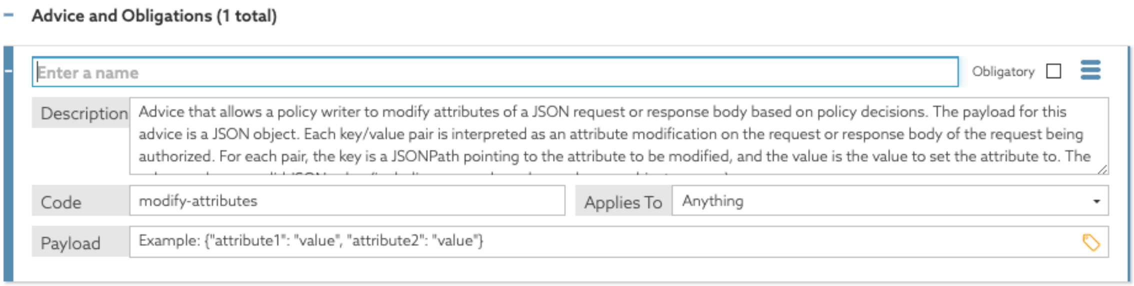


You can see the documentation for the provided advice types from within the Policy Admin GUI. After you click **+ Add Advice**, hover over an advice type to view its description.



Modifies the attributes of a JSON request or response body.

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.



Description: Advice that allows a policy writer to modify attributes of a JSON request or response body based on policy decisions. 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 the attribute to. The

Code: modify-attributes

Applies To: Anything

Payload: Example: {"attribute1": "value", "attribute2": "value"}

For more information, see *Advice types*.

Custom advice

In addition to the advice types that are available out of the box in the PingDataGovernance Policy Administration GUI, policy writers can use a custom advice that leverages the PingDataGovernance Server SDK.

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

After configuring the advice properly, you can use it in a policy by selecting **Custom Advice** from the **Create new Advice** drop-down list.

Properties

Use properties 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 decision.

To evaluate the overall decision of a policy, the policy decision point (PDP) applies a combining algorithm. The default algorithm that is set on a new policy is **The first applicable will be the final decision**. This algorithm stops evaluating as soon as it reaches a decision that is not Not Applicable.

The following table identifies the available combining algorithms and describes their effects.

Combining algorithm descriptions

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. The evaluation of rules stops as soon as a Deny is produced.
DenyUnlessPermit	Unless one decision is permit, the decision is deny.	The policy defaults to Deny unless any of its children produce the decision Permit. The evaluation of rules stops as soon as a Permit is produced.
PermitOverrides	A single permit overrides any deny decisions.	If any children produce the decision Permit, the policy returns Permit and stops evaluating rules. If no Permit is generated, all rules are evaluated; also, the policy returns Indeterminate if a child produces Indeterminate. Otherwise, the policy returns Deny if a child produces Deny. If none of the previous situations occur, the policy returns Not Applicable.

Combining algorithm	Summary	Details
DenyOverrides	A single deny overrides any permit decisions.	If any children produce the decision Deny, the policy returns Deny and stops evaluating rules. If no Deny is generated, all rules are evaluated; also, the policy returns Indeterminate if a child produced Indeterminate. Otherwise, the policy returns Permit if a child produces Permit. If none of the previous situations occur, the policy returns Not Applicable.
FirstApplicable	The first applicable decision is the final decision.	Evaluates the children in turn until one produces an applicable value of Permit, Deny, or Indeterminate. If the evaluation produces no applicable decisions, the policy returns Not Applicable.
OnlyOneApplicable	Only one child can produce a decision. If more than one child produces a decision, the result is indeterminate.	Evaluates the children 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 evaluation produces no applicable decisions, the policy returns Not Applicable.
DenyUnlessThreshold	Permit if the weighted average of applicable child decisions meets the threshold; otherwise deny.	Assigns the policy's children 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 evaluating all children, the PDP divides the total by the number of children and compares that average 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.

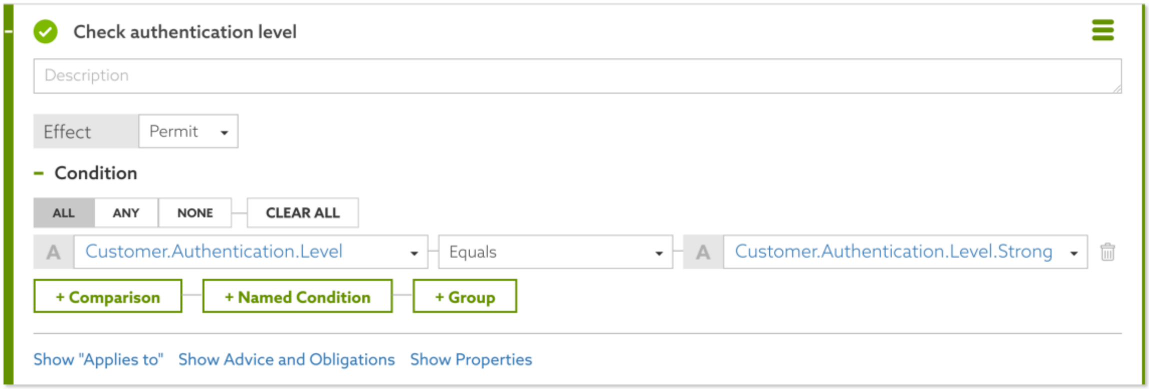
You can give each rule an effect of permit or deny. The effect is what the rule evaluates to when its child condition or group of conditions evaluates to true. You can set a rule so that, if a condition evaluates to true and the effect is set to deny, the rule evaluates to deny.

ⓘ **Important:**

A condition that returns false causes the rule to be Not Applicable. It does not create the opposite effect. You must create a separate and opposite rule to generate the opposite effect. The most consistent way to create such a pair of rules is to use *Named conditions* on page 317, with both rules referencing the same named condition but with the expected outcome being opposite.

Rules can include targets, which work in the same way as on policies and policy sets. However, you cannot associate conditions with these targets. You can apply targets 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.



Check authentication level

Description

Effect: Permit

Condition

A Customer.Authentication.Level | Equals | A Customer.Authentication.Level.Strong

If the condition in this example evaluates to true, the effect is Permit. If it evaluates to false, the effect is Not Applicable.

ⓘ **Tip**:

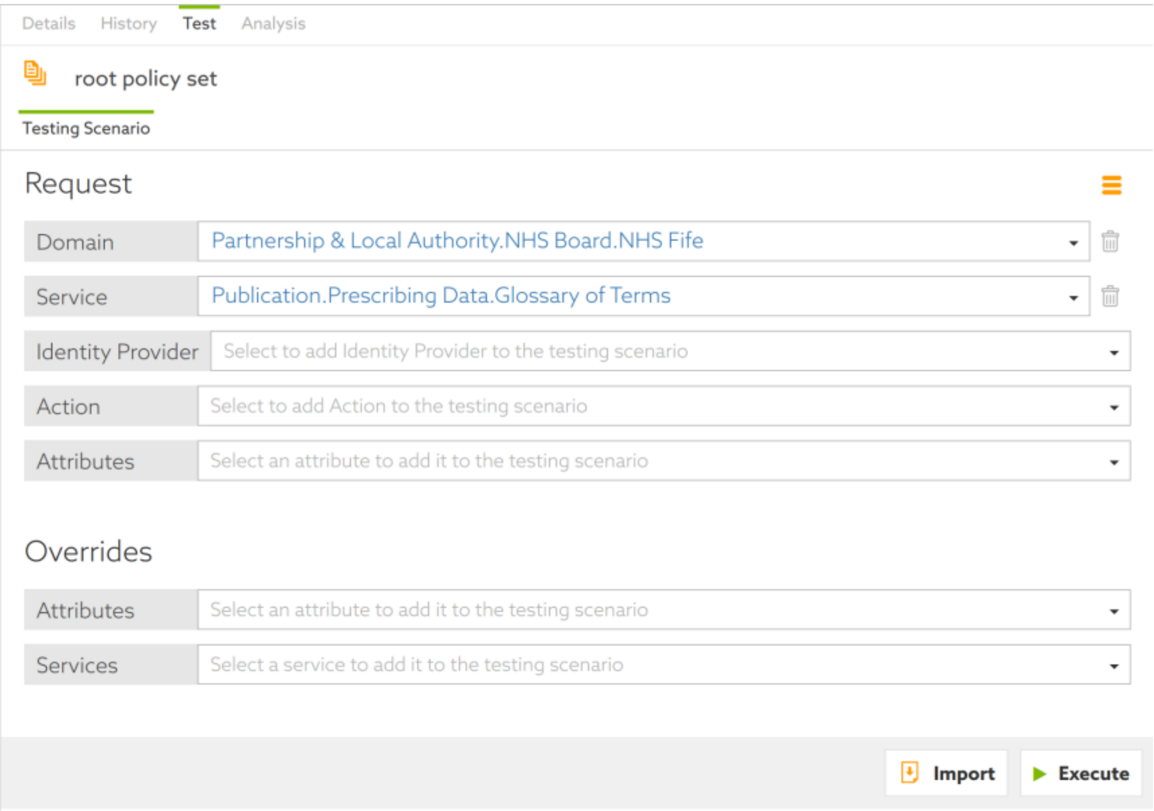
When a logical condition involves comparing two attributes, try to ensure the attributes have the same data type. Comparing different data types requires an implicit conversion that might not always yield the intended result.

Testing

The PingDataGovernance Policy Administration GUI provides testing capabilities to evaluate test authorization requests against any or all policy nodes.

To specify the nodes to test policies against, select the root node from the tree on the left side of the page.

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



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

Overrides

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

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

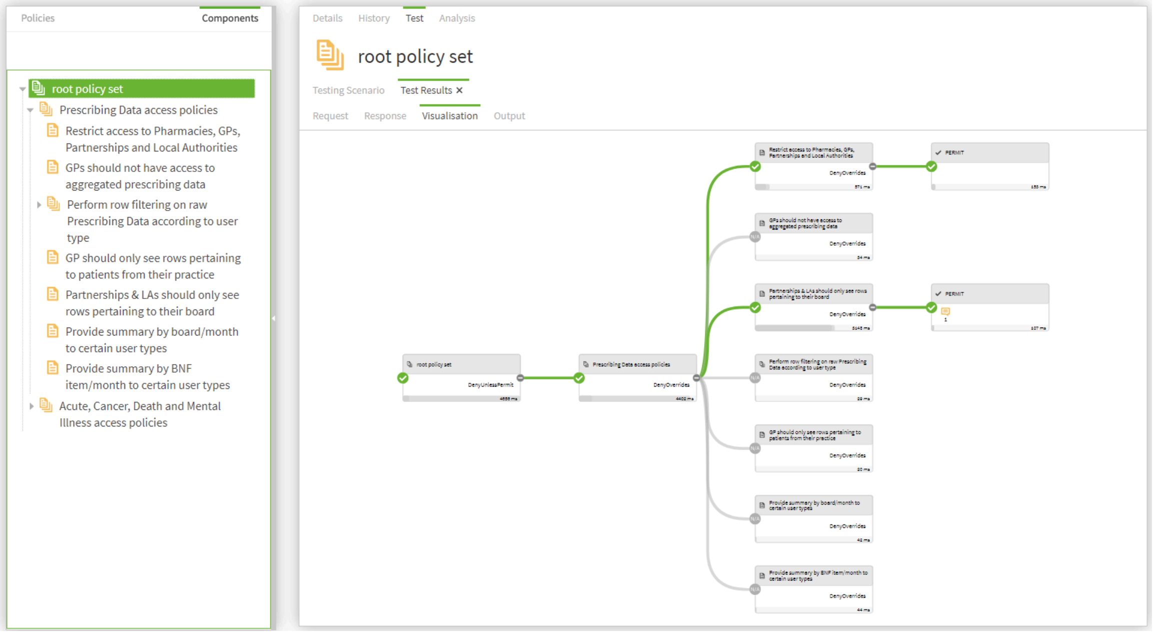
Select the following main elements to form a request:

- 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 will see the following set of result tabs:

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


Policies
Components
root policy set
Prescribing Data access policies
Restrict access to Pharmacies, GPs, Partnerships and Local Authorities
GPs should not have access to aggregated prescribing data
Perform row filtering on raw Prescribing Data according to user type
GP should only see rows pertaining to patients from their practice
Partnerships & LAs should only see rows pertaining to their board
Provide summary by board/month to certain user types
Provide summary by BNF item/month to certain user types
Acute, Cancer, Death and Mental Illness access policies
Details
History
Test
Analysis
root policy set
Testing Scenario
Test Results ✕
Request
Response
Visualisation
Output
root policy set
DenyUnlessPermit
Prescribing Data access policies
DenyOverrides
Restrict access to Pharmacies, GPs, Partnerships and Local Authorities
DenyOverrides
PERMIT
GPs should not have access to aggregated prescribing data
DenyOverrides
Partnerships & LAs should only see rows pertaining to their board
DenyOverrides
PERMIT
1
Perform row filtering on raw Prescribing Data according to user type
DenyOverrides
GP should only see rows pertaining to patients from their practice
DenyOverrides
Provide summary by board/month to certain user types
DenyOverrides
Provide summary by BNF item/month to certain user types
DenyOverrides
N/A

Analysis of policies and policy sets

The PingDataGovernance Policy Administration GUI provides an analysis capability for policies and policy sets. This capability is limited to small trees.

The options available for analysis are:

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 on 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 policy information providers whose failure might alter the decision.

Policy solutions

This section recommends how to implement commonly needed business rules in policy.

Restricting the attributes that can be modified

Starting with PingDataGovernance 8.1, the Allow Attributes advice and Prohibit Attributes advice are no longer supported. If you have policies that use those advices, change them to use the `impactedAttributes` policy attribute.

About this task

The `impactedAttributes` attribute is defined in `resource/policies/defaultPolicies.SNAPSHOT`. If you are using a branch created from that snapshot, the attribute already exists in the branch. If not, create the attribute.

Steps

1. Go to **Trust Framework**, and then click **Attributes**.
2. From the **+** menu, select **Add new Attribute**.
3. Replace **Untitled** with `impactedAttributes`.
4. Verify that in the **Parent** field, no parent is selected. To remove a parent, click the trash can icon to the right of the **Parent** field.
5. Click **+ Add Resolver** and set the **Resolver type** to **Request**.
6. In the **Value Settings** section:
    a. Select the box next to **Default value** and specify square brackets with no space between them (`[]`) as the value.
    b. Set **Type** to **Collection**.
7. Click **Save changes**.

Allowing attributes to be modified

To allow any attribute to be modified, such as for an administrator account, the policy decision point (PDP) does not need to check the `impactedAttributes` attribute.

About this task

To create a policy that allows an administrator to modify any attributes, complete the following step.

Steps

- Create a policy with a rule with **Effect** set to **Permit** the decision based on the **Condition** that the user is an administrator.

  To check the user, for example, you can set up a condition to compare whether `HttpRequest.AccessToken.scope` equals administrator.

Whitelisting attributes

To allow the user to modify a set of attributes limited to a whitelist and return an error if the user attempts to modify any attribute outside the whitelist, create a constant in the Trust Framework and then use the constant in a policy.

Steps

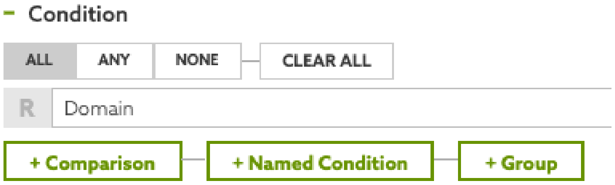
1. Create a constant in the Trust Framework.
   a. Go to **Trust Framework** and then **Attributes**.
   b. From the **+** menu, select **Add new Attribute**.
   c. Replace **Untitled** with `whitelistAttributes`.
   d. Verify that in the **Parent** field, no parent is selected. To remove a parent, click the trash can icon to the right of the **Parent** field.
   e. Click **+ Add Resolver** and set the **Resolver type** to **Constant**.
   f. Set the value of the constant to a set of square brackets that contains a comma-delimited list of the attributes that can be modified.

      For example, to allow the `email` or `userName` attributes to be modified, you would set the value of the constant to `[email, userName]`.

      As another example, to allow the user to modify a property or any of its subproperties, you must explicitly list them. So to allow modification of the `name` field on the default Users pass-through schema, set the value of the constant to `[name, name.formatted, name.givenName, name.familyName]`.
   g. In the **Value Settings** section, set **Type** to **Collection**.
   h. Click **Save changes**.

2. Modify or create a policy to use that constant collection.
   a. Go to **Policies**.
   b. Select a policy or create a new one.
   c. In the **Rules** section:
      1. Set the **Combining Algorithm** to **Unless one decision is permit, the decision will be deny**.
      2. Click **+ Add Rule**.
      3. Replace **Untitled** with `Allow only the email and userName attributes`.
      4. Set the **Effect** to **Permit.**
      5. Under **Condition**, click **+ Comparison**.
      6. In the comparison, we want to compare the constant collection of permitted attributes to the `impactedAttributes` collection.
         - For the left field, select the `whitelistAttributes` attribute, which is the constant collection of permitted attributes defined in the beginning.

           You might see the field as shown below. Click the **R** immediately above **+ Comparison** to toggle to attribute selection.

         - Set the middle field (the operator) to **Contains**.
         - Set the right field to the `impactedAttributes` attribute.

           If that field has a **C** before it, click the **C** to toggle to attribute selection.

> ⓘ **Note:**
>
> If `impactedAttributes` is not available, see *Restricting the attributes that can be modified* on page 331.

When applied to two collections, the **Contains** operator returns true if and only if the right-side collection is a subset of the left-side collection. Thus, the rule only returns PERMIT if the set of `impactedAttributes` is a subset of the list of allowed attributes in `whitelistAttributes`.

Advice types

An advice is additional information you can attach to a decision response. It returns to the governance engine so that, depending on the evaluation response from the policy, PingDataGovernance can take the appropriate action.

This section describes the advice types built into PingDataGovernance Server.

Add Filter

Use `add-filter` to add administrator-required filters to System for Cross-domain Identity Management (SCIM) search queries.

Applicable to	SCIM.

Additional information	The Add Filter advice places restrictions on the resources 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 separated by `AND` or `OR`. If the policy result returns multiple instances of Add Filter advice, they are `AND`ed together to form a single filter that passes 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.

Combine SCIM Search Authorizations

Use `combine-scim-search-authorizations` to optimize policy processing for System for Cross-domain Identity Management (SCIM) search responses.

Applicable to	SCIM.
Additional information	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 you use this advice type, the current SCIM search result set is processed 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 the policy result returns 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 187.

Denied Reason

Use `denied-reason` to allow a policy writer to provide an error message that contains the reason for denying a request.

Applicable to	DENY decisions.
Additional information	The payload for Denied Reason advice is a JSON object string with the following fields: ▪ `status` – Contains the HTTP status code returned to the client. If this field is absent, the default status is 403 Forbidden. ▪ `message` – Contains a short error message returned to the client. ▪ `detail` (optional) – Contains additional, more detailed error information. The following example shows a possible response for a request made with insufficient scope `{"status":403, "message":"insufficient_scope", "detail":"Requested operation not allowed by the granted OAuth scopes."}`

Exclude Attributes

Use `exclude-attributes` to specify the attributes to exclude from a JSON response.

Applicable to	PERMIT decisions, although you cannot apply Exclude Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.

Additional information	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 being authorized. Each JSONPath can select multiple attributes in the object. The portions of the response that a JSONPath selects are removed before sending the response to the client. 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 246.

Filter Response

Use `filter-response` to direct PingDataGovernance Server to invoke policy iteratively over each item of a JSON array contained within an API response.

Applicable to	PERMIT decisions from Gateway, although you cannot apply Filter Response advice directly to a System for Cross-domain Identity Management (SCIM) search. However, the SCIM service performs similar processing automatically when it handles a search result. For every candidate resource in a search result, the SCIM service makes a policy request for the resource with an Action value of retrieve.

Additional information

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

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

Field	Required	Description
`Path`	Yes	JSONPath to an array within the API's response body. The advice implementation iterates over the nodes in this array and makes a policy request for each node.
`Action`	No	Value to pass as the `action` 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 `service` 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 `Path` field. On each subsequent policy request, the contents of a single array element pass 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

Use `include-attributes` to limit the attributes that a JSON response can return.

Applicable to	PERMIT decisions, although you cannot apply Include Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	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 being authorized. The response includes only the portions that one of the JSONPaths selects. When a single JSONPath represents multiple attributes, the response includes all of them. If the policy result returns multiple instances of Include Attributes advice, the response includes the union of all selected attributes. For more information about the processing of SCIM searches, see *Filter Response* on page 246.

Modify Attributes

Use `modify-attributes` to modify the values of attributes in the JSON request or response.

Applicable to	All, although you cannot apply the Modify Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	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 that selects the attribute to modify, and the value is the new value to use for the selected attribute. The value can be any valid JSON value, including a complex value like an object or array.

Modify Headers

Use `modify-headers` to modify the values of request headers before PingDataGovernance sends them to the upstream server or to modify the values of response headers before PingDataGovernance returns them to the client.

Applicable to	All, although you cannot apply the Modify Headers advice directly to a System for Cross-domain Identity Management (SCIM) search.
Additional information	The payload for this advice is a JSON object. The keys are the names of the headers to set, and the values are the new values of the headers. A value can be: ▪ Null, which removes the header ▪ A string, which sets the header to that value ▪ An array of strings, which sets the header to all of the string values If the header already exists, PingDataGovernance overwrites it. If the header does not exist, PingDataGovernance adds it (unless the value is null). If a payload value is an array of strings: ▪ Given a header that supports multiple values, such as `Accept`, PingDataGovernance repeats the header for each string in the array. ▪ Given a header that does not support multiple values, such as `Content-Type`, PingDataGovernance sends the last string in the array.

Modify Query

Use `modify-query` to modify the query string of the request sent to the API server.

Applicable to	All.

Additional information	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`, you can set a policy 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`.

Regex Replace Attributes

Use `regex-replace-attributes` to specify a regex to search for attributes in a request or response body and replace their values with a regex replacement string.

Applicable to	All, although you cannot apply the Regex Replace Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.

Additional information

The payload for this advice is either a JSON object or an array of JSON objects. Each object represents a single replacement operation and has up to four keys.

Key	**Description**
`"regex"`	Required. Represents the regular expression to use to find the attribute values to replace.
`"replace"`	Required. Represents the regex replacement string to use to replace the attribute values with a new value.
`"path"`	Optional. Is a JSONPath expression that represents the nodes to start searching under.
`"flags"`	Optional. Is a string that contains the regex flags to use. Recognized flags are: ▪ `"i"` Performs case-insensitive matching. ▪ `"l"` Treats the `"regex"` value as a literal string. ▪ `"c"` Performs "canonical equivalence" matching. You can combine flags. For example: `"il"`

PingDataGovernance replaces any portion of the attribute value that matches the regular expression in the `"regex"` value in accordance with the `"replace"` replacement string. If multiple substrings within the attribute value match the regular expression, PingDataGovernance replaces all occurrences.

The regular expression and replacement string must be valid as described in the API documentation for the java.util.regex.Pattern class, including support for capture groups.

For example, consider the following body.

```
{
   "id":5,
   "username":"jsmith",
   "description":"Has a registered ID number of '123-45-6789'.",
   "secrets":{
      "description":"Has an SSN of '987-65-4321."
   }
}
```

Also, consider the following payload.

```
{
   "path":"$.secrets",
   "regex":"(\\\\d{3}-\\\\d{2})-\\\\d{4}",
   "replace":"$1-XXXX"
}
```

Applying the advice produces the following body with a changed `"secrets.description"` value.

```
{
   "id":5,
   "username":"jsmith",
   "description":"Has a registered ID number of '123-45-6789'.",
```

REST API documentation

The PingDataGovernance Policy Administration GUI provides a set of REST APIs for managing policies, snapshots, and deployment packages. Swagger documentation for these APIs is available through the PingDataGovernance Policy Administration GUI if it was installed in demo mode.

For more information, click **API Reference** in the Policy Administration GUI.

Legal Information

Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Trademarks

Ping Identity, the Ping Identity logo, PingAccess, PingFederate, PingID, PingDirectory, PingDataGovernance, PingIntelligence, 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 Ping Identity product documentation 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.