PingAuthorize

Contents

PingAuthorize....7

Release Notes....7

PingAuthorize 9.1.0.5 (May 2024)....7
PingAuthorize 9.1.0.4 (November 2023)....8
PingAuthorize 9.1.0.3 (August 2023)....8
PingAuthorize 9.1.0.2 (March 2023)....8
PingAuthorize 9.1.0.1 (December 2022)....8
PingAuthorize 9.1 (June 2022)....8
PingAuthorize 9.0.0.6 (August 2023)....11
PingAuthorize 9.0.0.5 (April 2023)....11
PingAuthorize 9.0.0.4 (January 2023)....11
PingAuthorize 9.0.0.2 (July 2022)....11
PingAuthorize 9.0.0.1 (February 2022)....11
PingAuthorize 9.0 (December 2021)....11
Previous Releases....16

Introduction to PingAuthorize....16

Getting started with PingAuthorize (tutorials)....17

Using the tutorials....17
Setting up your environment....18
Starting PingAuthorize....18
Verifying proper startup....19
Accessing the GUIs....19
Stopping PingAuthorize....20
About the tutorial configuration....20
Tutorial 1: Importing default policies....21
Introduction to the Trust Framework and default policies....24
Tutorial 2: Configuring fine-grained access control for an API....26
Configuring a reverse proxy for the Meme Game API....28
Testing the reverse proxy....29
For further consideration: The PingAuthorize API security gateway, part 1....30
Adding a policy for the Create Game endpoint....31
For further consideration: The PingAuthorize API security gateway, part 2....32
Testing the policy from the Policy Editor....34
Testing the policy by making an HTTP request....37
For further consideration: Decision Visualiser....38
Modifying the rule for the Create Game endpoint....39
For further consideration: Resolvers and value processors....43
Conclusion....43
Tutorial 3: Configuring attribute-based access control for API resources....44
Configuring the API security gateway....44
Creating a policy based on user credentials....47
Creating a policy based on the API response....52
Conclusion....58

Tutorial (optional): Creating SCIM policies ... 59
Tutorial: Creating the policy tree ... 60
Tutorial: Creating SCIM access token policies ... 62
Tutorial: Creating a policy for role-based access control ... 74
Example files ... 77
Conclusion ... 77

Installing PingAuthorize ... 77
Docker deployment ... 83
Deployment requirements when using Docker ... 83
Deploying PingAuthorize Server and Policy Editor using Docker ... 84
Manual installation ... 88
Before you install manually ... 88
Installing the server and the Policy Editor manually ... 95
Signing on to the PingAuthorize Policy Editor ... 108
Changing the PingAuthorize Policy Editor authentication mode ... 108
Configuring an OIDC provider for single sign-on requests from PingAuthorize ... 110

Upgrading PingAuthorize ... 126
Upgrade considerations ... 127
Upgrade considerations introduced in PingAuthorize 8.x ... 128
Docker upgrades ... 131
Upgrading PingAuthorize Server using Docker ... 131
Upgrading the PingAuthorize Policy Editor using Docker ... 132
Manual upgrades ... 133
Upgrading PingAuthorize Server manually ... 133
Reverting an update ... 134
Upgrading the PingAuthorize Policy Editor manually ... 134
Policy-related upgrades ... 136
Backing up policies ... 136
Upgrading the Trust Framework and policies ... 136
Upgrading a PostgreSQL policy database ... 137

Uninstalling PingAuthorize ... 138

PingAuthorize Integrations ... 139
Kong API gateway integration ... 139
Preparing PingAuthorize for Kong Gateway integration ... 141
Setting up Kong Gateway ... 143
Troubleshooting the Kong Gateway integration ... 148
MuleSoft API gateway integration ... 151
Deploying the custom MuleSoft policy for PingAuthorize ... 152
Applying the custom MuleSoft policy for PingAuthorize ... 153

PingAuthorize Server Administration Guide ... 158
Running PingAuthorize ... 159
Starting PingAuthorize Server ... 159
Running PingAuthorize Server as a foreground process ... 159
Starting PingAuthorize Server at boot time (Unix/Linux) ... 159
Starting PingAuthorize Server at boot time (Windows) ... 160
Starting PingAuthorize Policy Editor ... 162

Stopping PingAuthorize Server ... 165
Stopping PingAuthorize Policy Editor ... 166
Restarting PingAuthorize Server ... 166
About the API security gateway ... 166
API gateway request and response flow ... 166
Gateway configuration basics ... 167
API security gateway authentication ... 168
API security gateway policy requests ... 169
API security gateway HTTP 1.1 support ... 175
Gateway error templates ... 176
About the Sideband API ... 178
API gateway integration ... 179
Sideband API configuration basics ... 180
Authenticating to the Sideband API ... 181
Authenticating API server requests ... 183
Sideband API policy requests ... 183
Request context configuration ... 189
Sideband access token validation ... 190
Sideband error templates ... 191
About the SCIM service ... 192
SCIM API request and response flow ... 192
SCIM configuration basics ... 193
SCIM endpoints ... 196
SCIM authentication ... 197
SCIM policy requests ... 197
Lookthrough limit for SCIM searches ... 207
Disabling the SCIM REST API ... 207
About the SCIM user store ... 208
Defining the LDAP user store ... 210
Location management for load balancing ... 213
Automatic backend LDAP server discovery ... 213
LDAP health checks ... 218
Connecting non-LDAP data stores ... 221
About the Authorization Policy Decision APIs ... 222
JSON PDP API request and response flow ... 222
Authenticating to the JSON PDP API ... 227
XACML-JSON PDP API request and response flow ... 228
Policy Editor configuration ... 238
Specifying custom configuration with an options file ... 238
Manage policy database credentials ... 250
Configuring SpEL Java classes for value processing ... 255
Setting the request list length for Decision Visualizer ... 256
HTTP caching ... 257
Policy administration ... 258
About the Trust Framework ... 258
Create policies in a development environment ... 260
Using the Deployment Manager ... 267
Use policies in a production environment ... 273
Policy database backups ... 277
Restoring a policy database from a backup ... 278
Policy application management with signed deployment packages ... 280
Environment-specific Trust Framework attributes ... 283
User profile availability in policies ... 288
Access token validators ... 290
Access token validator types ... 292
Token resource lookup methods ... 299

Server configuration ... 300
Administration accounts ... 300
About the dsconfig tool ... 301
PingAuthorize administrative console ... 302
About the configuration audit log ... 302
About the config-diff tool ... 302
Certificates ... 303
Configure the Policy Decision Service ... 354
User store configuration ... 354
Configure access token validation ... 355
Configure PingOne to use SSO for the administrative console ... 355
Configure traffic through a load balancer ... 357
PingAuthorize Server configuration with dsconfig ... 358
Deployment automation and server profiles ... 361
Variable substitution using manage-profile ... 362
Layout of a server profile ... 363
About the manage-profile tool ... 365
Common manage-profile workflows ... 366
Server status ... 370
Server availability ... 371
User Store Availability gauge ... 371
Endpoint Average Response Time (Milliseconds) gauge ... 372
HTTP Processing (Percent) gauge ... 373
Policy Decision Service Availability gauge ... 374
Auto-healing for unavailable servers ... 375
Available gauges ... 375
Common server alarms ... 379
Managing monitoring ... 381
Profiling server performance using the Stats Logger ... 381
Logging HTTP performance statistics using the Periodic Stats Logger ... 383
StatsD monitoring endpoint ... 383
Sending metrics to Splunk ... 384
Managing HTTP correlation IDs ... 385
About HTTP correlation IDs ... 385
Enabling or disabling correlation ID support ... 386
Configuring the correlation ID response header ... 386
How the server manages correlation IDs ... 386
Command-line tools ... 388
Saving command options in a file ... 393
Sample dsconfig batch files ... 395
Running task-based tools ... 395
Diagnostic and decision data ... 397
Exporting policy data ... 397
Enable detailed logging ... 397
About the Decision Response View ... 399
Visualizing a policy decision response ... 400
Capture debugging data with the collect-support-data tool ... 402
About the layout of the PingAuthorize Server folders ... 402
About the layout of the PingAuthorize Policy Editor folders ... 403

PingAuthorize Policy Administration Guide ... 404
Getting started ... 404
Version control (Branch Manager) ... 405
Creating a new top-level branch ... 405
Creating a subbranch from a commit ... 406

Importing a branch 406
Deleting a branch 407
Merging branches 407
Reverting branch changes 408
Committing changes 408
Generating snapshots 409
Partial snapshot export and merging 409
Creating a deployment package 410
Deleting a deployment package 411
Trust Framework 411
Domains (Authorization Policy Decision APIs only) 411
Services 411
Attributes 417
Actions 426
Identity classifications and IdP support 426
Named conditions 427
Value processing 427
Chained value processors 431
Trust Framework testing 431
Viewing Trust Framework entity dependencies 432
Policy management 433
Policy sets, policies, and rules 434
Policies and policy sets 434
Policy testing 444
Repeating policies and attributes 446
Policy solutions 448
Use case: Using consent to determine access to a resource 448
Use case: Using consent to change a response 463
Use case: Using a SCIM resource type or a policy request action to control behavior 471
Restricting the modification of attributes 487
Test Suite 489
Advice types 492
Add Filter 492
Combine SCIM Search Authorizations 492
Denied Reason 493
Exclude Attributes 493
Filter Response 494
Include Attributes 495
Modify Attributes 496
Modify Headers 496
Modify Query 497
Modify SCIM Patch 497
Regex Replace Attributes 499
REST API documentation 500

PingAuthorize

PingAuthorize software provides fine-grained, attribute-based access control and dynamic authorization management, enabling you to protect resources and filter data for databases, applications, and APIs.


Release Notes

- *Current*


Get Started with PingAuthorize

- *Introduction to PingAuthorize* on page 16
- *Installing PingAuthorize* on page 77
- *Uninstalling PingAuthorize* on page 138
- *PingAuthorize Tutorials*


Use PingAuthorize

- *Use cases*
- *Server admin guide*
- *Policy admin guide*
- *Policy development and promotion*
- *API gateway integrations*


Troubleshoot PingAuthorize

- *Enable detailed logging* on page 397
- *Capture debugging data*
- *Monitor server availability*
- *Troubleshoot TLS-related issues*
- *Configure LDAP health checks*
- *Visualize a policy decision response*


Learn More

- *API reference guide*
- *PingAuthorize Server Docker image*
- *PingAuthorize Policy Editor Docker image*
- *PingAuthorize Community*
- *Ping Identity Support Portal*
- *PingAuthorize customer training (existing customers only)*
- *Partner Portal (partners)*

Release Notes

New features and improvements in PingAuthorize. Updated May 31, 2024.

PingAuthorize 9.1.0.5 (May 2024)

Fixed SCIM case-sensitivity error PAZ-8473 **Fixed**

We fixed an issue where requests to create SCIM entries were not always observing the `case-exact=false` property, leading to incorrect case-sensitivity errors. Now, requests featuring this property will not be case-sensitive.

Fixed a `NullPointerException` caused by an unconfigured alert handler DS-47455 **Fixed**

We fixed an issue where a `NullPointerException` was thrown when an alert or alarm was raised, and one or more of the alert handlers were not configured. This most commonly happened when the server was being stopped.

Now, instead of throwing a `NullPointerException`, the server logs this message: `Alert notification '<notification>' will not be processed by alert handler '<alert handler>' since that alert handler does not have configuration.`

PingAuthorize 9.1.0.4 (November 2023)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.1.0.3 (August 2023)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.1.0.2 (March 2023)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.1.0.1 (December 2022)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

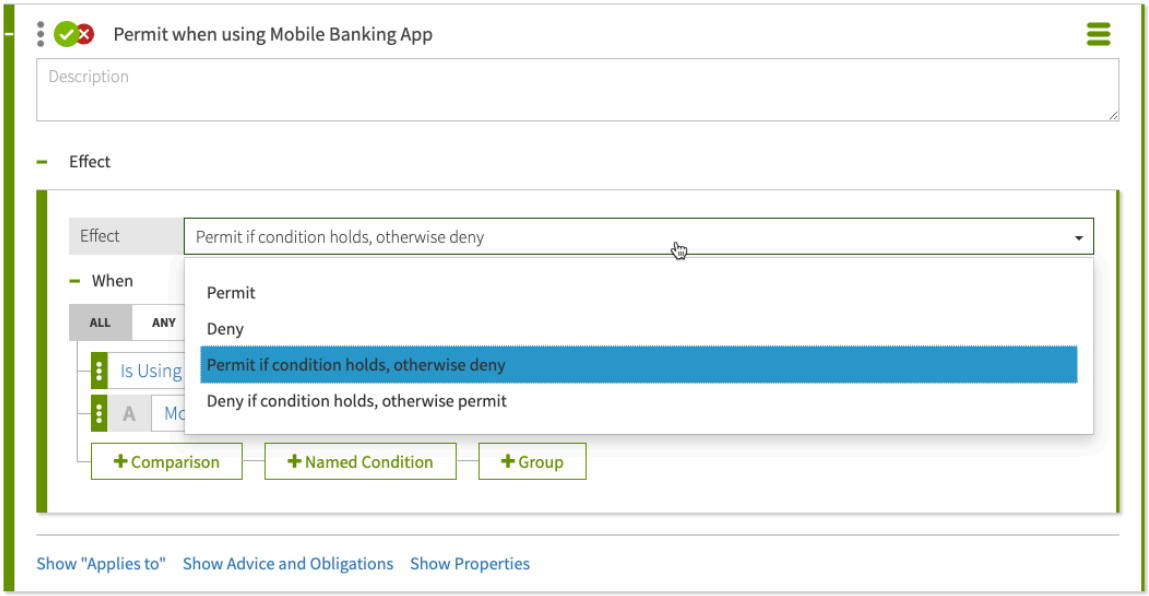
PingAuthorize 9.1 (June 2022)

Updated `commons-codec` to address a security issue DS-45898 **Security**
Updated the `commons-codec` library to version 1.13 to address a security issue.

Updated Jackson Databind to address a security vulnerability DS-45806 **Security**
Updated Jackson Databind to 2.13.3 to address the *CVE-2020-36518* security vulnerability.

Updated Google Guava to address a security vulnerability DS-45903 **Security**
Updated the Google Guava dependency in common libraries to address the *CVE-2020-8908* security vulnerability.

Added conditional effects for policy rules **New**
Rules now include *conditional effects*, allowing policy builders to write one rule with two possible effects. The effect produced depends on whether the effect condition evaluates to `true` or `false`.



Note:

Previous rule conditions are now set as targeting conditions in the **Applies To** section.

Added the ability to configure attribute logging for the Policy Decision Service **New**

Added the option to *configure logging for Trust Framework attributes*. The Policy Decision Service logs the designated attributes when they are evaluated as part of a request. This option is only available in embedded mode.

Added the ability to sanitize error logging to protect sensitive data **New**

Added the ability to sanitize error log messages as they are generated. This can help prevent sensitive information from being leaked through log messages, although the resulting log messages can potentially be less useful for troubleshooting purposes. See *Log Sanitization* for more information.

Updated the administrative console browser support **Info**

The administrative console now supports Microsoft Edge. Administrative console support for Microsoft Internet Explorer 11 has been deprecated.

Deprecated Apache Camel for PIP connections **Info**

Using Apache Camel to connect policy information points (PIPs) to PingAuthorize has been deprecated, and the feature will be removed in a future release of the product. We recommend using HTTP services instead, where applicable.

Made it easier to present a custom SSL certificate to the Policy Editor **Improved**

We added a new *environment variable* named **`KEYSTORE_PIN_FILE`** to the Policy Editor **`setup`** and **`start-server`** tools. This variable takes precedence over **`PING_KEYSTORE_PASSWORD`** when validating and presenting the server certificate.

Improved UI performance in the Policy Editor **Improved**

The Policy Editor now supports API *HTTP caching*, which is enabled by default to improve UI performance. Disable this feature and restore the legacy behavior by providing the `--disableApiHttpCache` option to the **`setup`** tool. Alternatively, set the environment variable **`PING_ENABLE_API_HTTP_CACHE`** to `false` when running **`start-server`** to disable it for a particular server runtime instance.

Added a command-line configuration tool for PingAuthorize Docker containers **Improved**

Added a **`docker-pre-start-config`** command-line tool for PingAuthorize Docker containers. Use the tool before the server is started to make configuration changes to the server that depend on the running container's environment.

Added and updated PingAuthorize Server profile command-line tools **Improved**

Added a `--skipValidation` argument for the `manage-profile replace-profile` command. This argument allows skipping the final server validation step when running on an offline server.

Added an `--excludeSetupArguments` argument for the `manage-profile generate-profile` command. This argument allows generating a server profile that does not include a `setup-arguments.txt` file.

Updated the **`setup`** and **`replace-profile`** subcommands to fail when a server profile includes an `encryption-settings-db` file in the profile's `server-root/pre-setup/` directory.

Enhanced advice logging **Improved**
During advice processing, the File Based Error Log Publisher publishes additional helpful messages to the configured output file.

Removed the OIDC `offline_access` scope requirement for the Policy Editor PAZ-3061 **Fixed**
The Policy Editor no longer requires the `offline_access` scope when configured in OpenID Connect mode using the `Authorization Code with PKCE` grant type.

Fixed the Policy Editor issue rejecting bearer tokens with array-type `aud` claims PAZ-1088 **Fixed**
Fixed an issue that prevented the Policy Editor REST APIs from accepting a bearer token when the `aud` claim was an array of strings.

Enabled the Policy Editor to decode JWTs with underscores PAZ-4325 **Fixed**
The Policy Editor is now able to decode JWTs that contain underscore characters.

Enhanced HTTP performance PAZ-3238, PAZ-2291 **Fixed**
This release includes general HTTP performance improvements and bug fixes.

Fixed alert consistency for cleared alarms DS-45578 **Fixed**
Fixed issues where gauges could raise an alarm and create an alert, but not create an alert when that same alarm was later cleared, making it unclear when the reported condition had abated.

Updated the API gateway behavior for handling trailing zeros PAZ-2705 **Fixed**
When operating as an API gateway, PingAuthorize will no longer remove trailing zeros from numbers in non-SCIM response bodies and advice payloads.

Fixed the Policy Editor UI tab switching error PAZ-2110 **Fixed**
Fixed an issue where the Policy Editor threw an error when rapidly switching between Trust Framework tabs under slow network conditions.

Fixed the Policy Editor error that occurs when updating entities concurrently PAZ-3667 **Fixed**

Fixed an issue where concurrent updates to the same entities in the Policy Editor could sometimes produce an error.

Fixed an issue resolving JSONPath expressions that contain the `keys()` function PAZ-4501 **Fixed**
Fixed an issue where calling `keys()` in a JSONPath expression did not return the object's keys.

Fixed the PIN retrieval issues with third-party passphrase providers DS-45336 **Fixed**
Fixed issues that prevented obtaining key and trust store PINs with the Amazon Secrets Manager, CyberArk Conjur, and HashiCorp Vault passphrase providers.

Fixed erroneous certificate expiration warnings DS-41468 **Fixed**
Fixed an issue that prevented the server from refreshing the monitor data used to detect and warn about an upcoming certificate expiration. This could cause the server to continue to warn about an expiring certificate even after that *certificate had been replaced*.

Fixed the PingAuthorize name and version in `collect-support-data` DS-45280 **Fixed**
The **`collect-support-data`** (CSD) tool now correctly displays the name and version of PingAuthorize.

Updated the incorrect version information for `collect-support-data` DS-44481 **Fixed**
The **`status`** tool now shows the current **`collect-support-data`** version.

Updated to LDAP SDK version 6.0.5 DS-45746 **Fixed**
Updated to LDAP SDK for Java version 6.0.5 for bug fixes and new functionality.

Recovering from a failed `setup` on Windows DS-45941 **Issue**
The **`setup`** command might fail on Windows operating systems due to the presence of Bouncy Castle JAR files that begin with `bc` in the `lib` directory. The JAR files are mentioned in an error message similar to the following:

```
An unexpected error occurred while attempting to copy the non-FIPS Bouncy
 Castle jar file into the server's classpath:
FileSystemException:
lib\bcprov-jdk15to18-1.71.jar:
```

```
The process cannot access the file because it is being used by another
 process.
```

A temporary workaround is to delete the JAR files that begin with `bc` from the `lib` directory before attempting to run **`setup`** again.

PingAuthorize 9.0.0.6 (August 2023)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.0.0.5 (April 2023)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.0.0.4 (January 2023)

Fixed erroneous certificate expiration warnings DS-41468 **Fixed**
Fixed an issue that prevented the server from refreshing the monitor data used to detect and warn about an upcoming certificate expiration. This could cause the server to continue to warn about an expiring certificate even after that *certificate had been replaced*.

PingAuthorize 9.0.0.2 (July 2022)

Updated to LDAP SDK version 6.0.5 DS-45746 **Fixed**
Updated to LDAP SDK for Java version 6.0.5 for bug fixes and new functionality.

Fixed an incorrect SCIM `POST` error response DS-45647 **Fixed**
Fixed an issue where SCIM **`POST`** requests that violated a unique attribute constraint received an internal error instead of the expected SCIM error response.

Fixed an incorrect SCIM `POST` error code DS-45863 **Fixed**
Fixed an issue where SCIM **`POST`** requests that violated a unique attribute constraint received an error response with status `400 Bad Request` instead of `409 Conflict`.

Fixed the PingAuthorize name and version in `collect-support-data` DS-45280 **Fixed**
The **`collect-support-data`** (CSD) tool now correctly displays the name and version of PingAuthorize.

PingAuthorize 9.0.0.1 (February 2022)

Version incremented for administrative purposes **Info**
The PingAuthorize version number was incremented due to changes released for PingDirectory. There are no release notes for this version of PingAuthorize.

PingAuthorize 9.0 (December 2021)

Added support for policy deployment from Microsoft Azure blob storage **New**
The PingAuthorize Server can now consume deployment packages published to Microsoft Azure blob storage. This enables policy writers to deploy new policies to a central Azure deployment package store read by the PingAuthorize Server running in embedded mode. For more information, see *Adding an Azure*

deployment package store on page 271, *Configuring the Policy Editor to publish policies to a deployment package store* on page 248, and *Using the Deployment Manager* on page 267.

Enabled configuration of the SpEL allow list in PDP mode **New**

Now you can configure the SpEL allow list when the Policy Decision Service is running in embedded policy decision point (PDP) mode. An out-of-the-box PingAuthorize installation adds the following classes to the default allow list: `String`, `Date`, `Random`, `UUID`, `Integer`, `Long`, `Double`, `Byte`, `Math`, `Boolean`, `LocalDate`, `DayOfWeek`, `Instant`, `ChronoUnit`, and `SimpleDateFormat`. When configuring a policy deployment package containing SpEL expressions that reference additional Java classes, administrators must use **`dsconfig`** or the administrative console to add spel-allowlisted-class attributes to the Policy Decision Service. The class must also be available on the server classpath at server start. For non-standard Java classes, place the `.jar` file in the server `lib` folder.

Expanded Policy Editor database support to include PostgreSQL **New**

The PingAuthorize Policy Editor can now persist its policies, Trust Framework, and versioning data in a PostgreSQL policy database instead of the default H2 file-based database. To initialize the database, use the instructions at *https://github.com/pingidentity/pingauthorize-contrib/tree/main/sql/postgresql*. To configure the Policy Editor for PostgreSQL, use the following setup options:

- `--dbConnectionString`
  - The JDBC connection string (for example, "`jdbc:postgresql://localhost:5432/policy_db`")
- `--dbAppUsername`
  - The PostgreSQL user
- `--dbAppPassword`
  - The user's password

Added support for the MuleSoft API Gateway in a sideband architecture **New**

Now you can deploy PingAuthorize in a sideband configuration with the MuleSoft API Gateway. With a sideband deployment, your organization can quickly set up an environment for fine-grained, dynamic authorization that integrates with existing identity management infrastructure and requires minimal changes to your network configuration. For more information about our custom MuleSoft policy, see *MuleSoft API gateway integration* on page 151.

OpenID Connect (OIDC) Authorization Code with Proof Key for Code Exchange (PKCE) **New**

Policy Editor setup in OpenID Connect (OIDC) authentication mode now uses the Authorization Code with Proof Key for Code Exchange (PKCE) grant type by default, instead of the implicit grant type. For information about configuring the Policy Editor in OIDC authentication mode, see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.

Upgrading from early access to general availability **Info**

If you are upgrading from PingAuthorize 9.0.0.0 Early Access to 9.0.0.0 General Availability, you must upgrade both the PingAuthorize Server and the Policy Editor before you use the Policy Decision Service in external mode. Upgrading only one component results in this error: `Please upgrade to PingAuthorize Policy Editor version '9.0.0.0'`.

Server profiles replace peer setup **Info**

Peer server setup and clustered configuration have been removed from **`setup`**. To manage server configuration, use server profiles instead of peer setup. Server profiles support deployment best practices such as automation and Infrastructure-as-Code (IaC). For more information about server profiles, see *Deployment automation and server profiles* on page 361.

Upgrading from earlier versions of PingAuthorize **Info**

For more considerations, see *Upgrade considerations* on page 127.

Added support for password storage schemes **Improved**

Added support for password storage schemes that allow users to authenticate with passwords stored in the Amazon AWS Secrets Manager service, the Microsoft Azure Key Vault service, a CyberArk Conjur instance, or a HashiCorp Vault instance.

Added redaction capability for `dsconfig` **Improved**

Added a global configuration property that can be used to indicate that the values of sensitive configuration properties should be redacted when constructing the **`dsconfig`** representation for a configuration change, given that these values might be included in the server's configuration audit log or administrative alerts whenever a configuration change is applied. By default, the values of configuration properties that are defined as sensitive get obscured rather than redacted, which allows the change to be replayed without revealing the actual value of the property. However, it is now possible to redact such values rather than obscuring them, which provides stronger protection against exposing those values but might interfere with the ability to replay the configuration audit log if it contains changes involving sensitive properties.

Mirrored configuration change logging **Improved**

Updated the server to record the original requester's DN and IP address in access log and configuration audit log messages for mirrored configuration changes.

Added support for obtaining secrets from CyberArk Conjur **Improved**

The Conjur cipher stream provider can use a retrieved secret to generate the encryption key used to protect the contents of the encryption settings database. The Conjur passphrase provider can be used in other cases where the server might need a clear-text secret, including PINs for accessing certificate key stores or credentials for authenticating to external services. The server can authenticate to Conjur with a username and password or an API key.

Added support for obtaining secrets from Azure Key Vault **Improved**

The Azure Key Vault cipher stream provider can use a retrieved secret to generate the encryption key used to protect the contents of the encryption settings database. The Azure Key Vault passphrase provider can be used in other cases where the server might need a clear-text secret, including PINs for accessing certificate key stores or credentials for authenticating to external services.

Added a PKCS #11 cipher stream provider **Improved**

Added a PKCS #11 cipher stream provider that can require access to a certificate in a PKCS #11 token to unlock the server's encryption settings database. Only certificates with RSA key pairs can be used because JVMs do not currently provide adequate key wrapping support for elliptic curve key pairs.

Runtime server problem-status handling **Improved**

When the Policy Decision Service is unable to handle requests due to misconfiguration or problems with the runtime environment, the PingAuthorize Server status is now DEGRADED instead of UNAVAILABLE. Orchestration systems like Kubernetes now remove such servers from pools instead of restarting them, allowing server administrators to investigate and correct the issue.

Added administrative console PIN support **Improved**

The administrative console can now be configured to supply PINs to its trust stores through the `oidc-trust-store-pin-passphrase-provider` and `trust-store-pin-passphrase-provider` settings. This means trust store types that require passphrases (for example, PKCS12 or BCFKS) are now properly supported.

Administrative console file retrieval with SSO **Improved**

The administrative console can now retrieve files created from **`collect-support-data`** or **`server-profile`** tasks when using single sign-on (SSO) to authenticate with the managed server.

Added file servlet support for OIDC and OAuth 2.0 **Improved**

Updated the file servlet to add support for token-based authentication using an OAuth 2.0 access token or an OpenID Connect ID token. The servlet previously only supported basic authentication.

`manage-profile generate-profile` argument validation **Improved**

Improved `includePath` argument validation performed by the **`manage-profile generate-profile`** tool. The tool will only use relative paths that exist below the server root, and it previously silently ignored absolute paths or relative paths that referenced files outside of the server root. It will now exit with an error if the `includePath` argument is used to provide an absolute path or a path outside the server root. It will accept—but warn about—paths that reference files that do not exist.

Expanded `ldap-diff` capabilities **Improved**

Made several improvements to the **`ldap-diff`** tool:

- Added the ability to perform a byte-for-byte comparison of attribute values rather than using schema-based logical equivalence.
- Added the ability to use a properties file to obtain default values for command-line arguments.
- Improved the ability to use different TLS-related settings for the source and target servers.
- Improved support for SASL authentication.

Added TLS protocol configuration to the crypto manager **Improved**
Updated the crypto manager configuration to add properties for controlling the set of TLS protocols and cipher suites that will be used for outbound connections, as well as properties for controlling whether to enable TLS cipher suites that rely on the SHA-1 digest algorithm or the RSA key exchange algorithm.

Added JDK support **Improved**
Added support for the use of JDKs obtained through Eclipse Foundation and BellSoft.

Added certificate management support **Improved**
Added support for new extended operations that can be used to help manage the server's *listener* and inter-server certificates. Updated the **`replace-certificate`** tool to add support for replacing and purging certificates in a remote instance, and to allow skipping validation for the new *certificate chain*.

Secret key loss when removing a server from the topology DS-44591 **Fixed**

Fixed an issue introduced in version 7.0.0.0 where secret keys under `cn=Topology,cn=config` could be lost when removing a server from the topology. When a server is removed via the **`dsreplication disable`** or **`remove-defunct-server`** tools, its secret keys will now be distributed among the remaining members of the topology. The keys from the rest of the topology will also be copied to the server being removed.

The cipher secret keys in the topology that are affected by this change are used by reversible password storage schemes (except for AES256, which uses the encryption settings database). If you are using a reversible password storage scheme other than AES256, prior to this fix, you could lose access to keys that had been used for reversible password encryption when removing servers from the topology.

Note:

Since this change only applies to the most recent version of **`remove-defunct-server`** and **`dsreplication disable`**, if you are removing a server from a multi-version topology, you should run that tool from the most recent version. In the past, **`dsreplication disable`** and **`remove-defunct-server`** could only be run from an older version. Now, when removing a server from the topology, they should be run from the most recent version in the topology. If you run the tool from an older server, it will not include this fix, and you might lose access to secret keys from servers that are removed from the topology.

Shutting down PingAuthorize Server with an invalid package store DS-44770 **Fixed**
An invalid deployment package store no longer prevents the PingAuthorize Server from shutting down.

`remove-defunct-server` attribute removal DS-44793 **Fixed**
Fixed an issue in which **`remove-defunct-server`** would remove attributes from **`config.ldif`** if they were identical apart from case.

Policy Editor batch scripts refer to non-existent Java files DS-45105 **Fixed**
The PingAuthorize Policy Editor **`start-server.bat`** and **`stop-server.bat`** scripts no longer output messages referring to non-existent `java.properties` or `dsjavaproperties` files.

JVM segmentation faults during `start-server` DS-45124 **Fixed**
Removed `-XX:RefDiscoveryPolicy=1` from the default **`start-server`** Java arguments. In rare cases, this argument was related to segmentation faults in the Java virtual machine, especially when used with the G1 garbage collector.

Configuration keys and values in the Policy Editor Test Suite PAZ-1481 **Fixed**
The Policy Editor now uses policy configuration keys and values correctly in Test Suite tests. For details about configuring policy configuration keys, see *Environment-specific Trust Framework attributes* on page 283.

OIDC authentication to the Policy Editor for PingOne users with TLS 1.3 might malfunction PAZ-5212 **Fixed**
When PingOne users authenticate with OIDC to the Policy Editor, environments using OpenJDK versions older than 11.0.3 might run into an intermittent TLS 1.3 issue preventing them from loading test scenarios. The issue appears in the logs as `com.symphonicsoft.authentication.OidcAuthenticator: Could not retrieve jwks information from '<ping-one-url>/as/jwks'` and includes the following message: `javax.net.ssl.SSLException: No PSK available. Unable to resume.` This is an *OpenJDK bug* that has been fixed in version 11.0.3. To circumvent this issue, you can upgrade to OpenJDK 11.0.3 or newer. Disabling TLS 1.3 also prevents this issue.

Deployment package store detection DS-44549 **Issue**
If the configured deployment package store is not available when the PingAuthorize Server starts, it will not be able to detect when the store becomes available again. To ensure that the PingAuthorize Server begins using the deployment package store when the store is available again, you must restart the server or change the Policy Decision Service configuration.

Can't use an existing persistent database with Docker volumes DS-44206 **Issue**
The `pingdatagovernancepap` and `pingauthorizepap` Docker images now run as unprivileged (non-root) users by default. If you have existing `pingdatagovernancepap` policy databases, configure the containers to run as `root`. For more information, see *Deploying PingAuthorize Policy Editor using Docker* on page 86.

Can't persist the database in `/opt/db` with Docker volumes DS-44206 **Issue**
To persist a policy database in a Docker volume, create a new Docker volume with a mount target of `/opt/out` instead of `/opt/db`. For more information, see *Deploying PingAuthorize Policy Editor using Docker* on page 86.

Reconfiguring the Policy Editor in a Docker volume DS-44207 **Issue**
When you use the Policy Editor in a Docker volume, changing the configuration using an `options.yml` file also requires that you create an empty file such as `/opt/out/instance/delete-after-setup` before you restart `pingauthorizepap`. Consider this example:

1. You start the container with a command like the following:

```
$ docker run --network=<network_name> --name pap -p 8443:1443 \
 --env-file ~/.pingidentity/config \
 --volume /home/developer/pap/server-profile:/opt/in/ \
 --env PING_OPTIONS_FILE=custom-options.yml \
 --volume /home/developer/pap/Symphonic.mv.db:/opt/out/Symphonic.mv.db \
 --env PING_H2_FILE=/opt/out/Symphonic \
 pingidentity/pingauthorizepap:<TAG>
```

> **Note:**
>
> This example command bind mounts a customized `options.yml` file named `custom-options.yml` to the server root using the server profile capability. The host system `server-profile` folder must contain `instance/custom-options.yml` for this example to work correctly. The Docker image *<TAG>* is only a placeholder. See *https://devops.pingidentity.com/reference/config/*.

2. You decide to change the configuration, so you edit the `custom-options.yml` file.
3. You create the empty file with a command like this:

```
docker exec -it pap /bin/sh -c "touch /opt/out/instance/delete-after-
setup"
```

4. With that file in place, you can now restart the Policy Editor with the following commands:

```
$ docker stop pap
$ docker start --attach pap
```

Upgrading multi-server topologies from earlier versions DS-44165 **Issue**
Upgrading multi-server topologies that contain PingDataGovernance 6.x or 7.x to PingAuthorize is not supported.

Using the Periodic Stats Logger DS-43622 **Issue**
Published throughput and latency stats for SCIM, sideband, and gateway requests for the Periodic Stats Logger are not recorded until the requests are made and the logger is reset.

Policy Editor snapshot import error DS-41741 **Issue**
The Policy Editor produces an error when a user attempts to import an exported snapshot that contains references to named value processors.

Using the administrative console with Tomcat 9.0.31 DS-41836 **Issue**

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.

Harmless failure message when stopping the PingAuthorize service DS-42365 **Issue**

If you use the **`create-systemd-script`** tool to create a forking **`systemd`** service, the service is stopped by the **`systemctl stop ping-authorize.service`** command. At that time, you can see the status using the **`systemctl status ping-authorize.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.

Previous Releases

For information about enhancements and issues resolved in previous major and minor releases of PingAuthorize, follow these links to their release notes:

- *PingAuthorize 8.3 (June 2021)*

Introduction to PingAuthorize

PingAuthorize is a solution for fine-grained, attribute-based access control and dynamic authorization management.

Digital transactions worldwide are increasing at exponential rates. At the heart of every transaction are questions of authorization:

- Can a given user perform this action or access this resource?
- How much data can a given partner access?

With more sophisticated use cases and more regulations for sensitive data, the rules that guide these questions of authorization get more complex. For example, a user can only transfer funds if their account is in good standing and they've agreed to the terms of service, or a partner can only access user data for those users who have given explicit consent.

Using traditional, static authorization solutions, like role-based access control (RBAC), to address complex authorization requirements lacks the full transaction context available only with dynamic, runtime authorization. PingAuthorize dynamic authorization can evaluate any identity attribute, consent, entitlement, resource, or context to make attribute-based access control (ABAC) decisions in real time. PingAuthorize gives you centralized control over your digital transactions and application access to data.

The following components provide the main capabilities for PingAuthorize.

PingAuthorize Policy Editor

Policy Administration and Delegation

PingAuthorize Policy Editor enables nontechnical stakeholders to collaborate with IT and application developers to *build and test authorization policies* with a drag-and-drop UI. The editor supports fine-grained permissions and workflows to enable the right operational processes and delegated administration scenarios.

Attribute Resolution and Orchestration

Authorization policies depend on any combination of *attribute expressions* that are evaluated at runtime by PingAuthorize Server. These attribute values might be present in the transaction itself, like an identifier of the authenticated user.

PingAuthorize Policy Editor enables additional attribute values to be determined at runtime by configuring *attribute sources* and *attribute processing* without writing any code.

PingAuthorize Server

PingAuthorize Server includes the runtime policy decision service and multiple integration capabilities:

Authorization Policy Decision APIs

Applications or services obtain policy decisions at runtime using a policy decision point (PDP) API. Applications then enforce the decision in their own application or service code. This integration configuration is the most flexible, supporting any application or service use case.

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, PingAuthorize 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 data access control and protection for structured data stores like LDAP and RDBMS, customers can deploy the *SCIM Service* in front of their *data stores*.

In this configuration, PingAuthorize Server provides SCIM-based APIs through which clients create, read, update, and delete (CRUD) data. The SCIM Service enforces policy by blocking, filtering, obfuscating, or otherwise *modifying data and attributes*.

Important:

The available enforcement features described above vary depending on your subscription. For more information, check your PingAuthorize license key or contact your Ping Identity account representative.

Get started

To quickly see PingAuthorize in action, see *Getting started with PingAuthorize (tutorials)* on page 17.

Getting started with PingAuthorize (tutorials)

This section provides tutorials for installing and configuring PingAuthorize Server with different *fine-grained access control policies*.

As you complete this section, you will quickly get up and running with PingAuthorize Server and its *Policy Editor*. You will also learn how to implement data access policies for REST APIs and *System for Cross-domain Identity Management (SCIM)*.

Using the tutorials

Use the tutorials to familiarize yourself with the capabilities of PingAuthorize dynamic authorization management by walking through the provided configuration exercises.

Before you begin

To complete these tutorials, you must:

- Complete the instructions at *https://devops.pingidentity.com/get-started/introduction/*.
- Have access to Git.

- 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

About this task

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

Steps

1. Clone the GitHub repository that contains the supporting source files.

   Replace the variable <*X.X*> with the first two digits of the PingAuthorize release you want to clone.

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

   This command places the files in the `pingauthorize-tutorials` directory and changes to that directory. The directory contains a `docker-compose.yml` file that defines the containers used in the tutorial.

   You shouldn't need to modify this file or understand its contents to follow the tutorial steps. However, you might 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.
2. Copy the template to a new `.env` file 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.

   Note:

   You must still have a `.env` file in place for the environment to start.

Starting PingAuthorize

About this task

To start the Docker Compose environment:

Steps

1. Go to the `pingauthorize-tutorials` directory you cloned in *Setting up your environment* on page 18.
2. Run the following command.

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

Verifying proper startup

About this task

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`. All four containers should reach an equilibrium state of `healthy`.

Steps

- To verify that both PingAuthorize Server and Policy Editor started properly and are running, run the following command.

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

> **Note:**
> It could take up to 15 minutes for all four containers to reach this equilibrium state.

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

Accessing the GUIs

About this task

PingAuthorize has two GUIs:

- Administrative console
- Policy Editor

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

Steps

- Access either the administrative console or the Policy Editor.
  Choose from:
  - To make configuration changes to PingAuthorize Server, access the administrative console.

Description	Details
URL	https://localhost:5443/console/login

Description	Details
Details to enter at sign-on	▪ **Server**: pingauthorize`:1636` ▪ **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 19.

- To make and test policy changes, access the Policy Editor.

  This GUI calculates decision responses when you configure PingAuthorize to use the GUI as an external policy decision point.

Description	Details
URL	https://localhost:8443
Details to enter at sign-on	▪ **User ID**: admin ▪ **Password**: password123

Stopping PingAuthorize

About this task

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.

Steps

1. Go to the `pingauthorize-tutorials` directory you cloned in *Setting up your environment* on page 18.
2. Run the following command.

```
docker-compose down
```

About the tutorial configuration

The provided Docker containers are pre-configured so that you can develop policies immediately.

The following Docker containers are provided through the Docker Compose environment.

Container	Description
pingauthorize	PingAuthorize Server The server enforces the policies you define.
pingauthorizepap	PingAuthorize Policy Editor Use this GUI to define the policies that determine access control and data protection.
pingdirectory	PingDirectory A directory of user information. **Note:** PingAuthorize doesn't 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 PingAuthorize.

Tutorial 1: Importing default policies

This tutorial describes how to use the PingAuthorize Policy Editor to import default attribute-based access control policies. It also 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 PingAuthorize Server and the Policy Editor. You must base all policies that you create for use with PingAuthorize on the policies and Trust Framework entities defined in this file.

To use the default policies that are distributed with PingAuthorize Server:

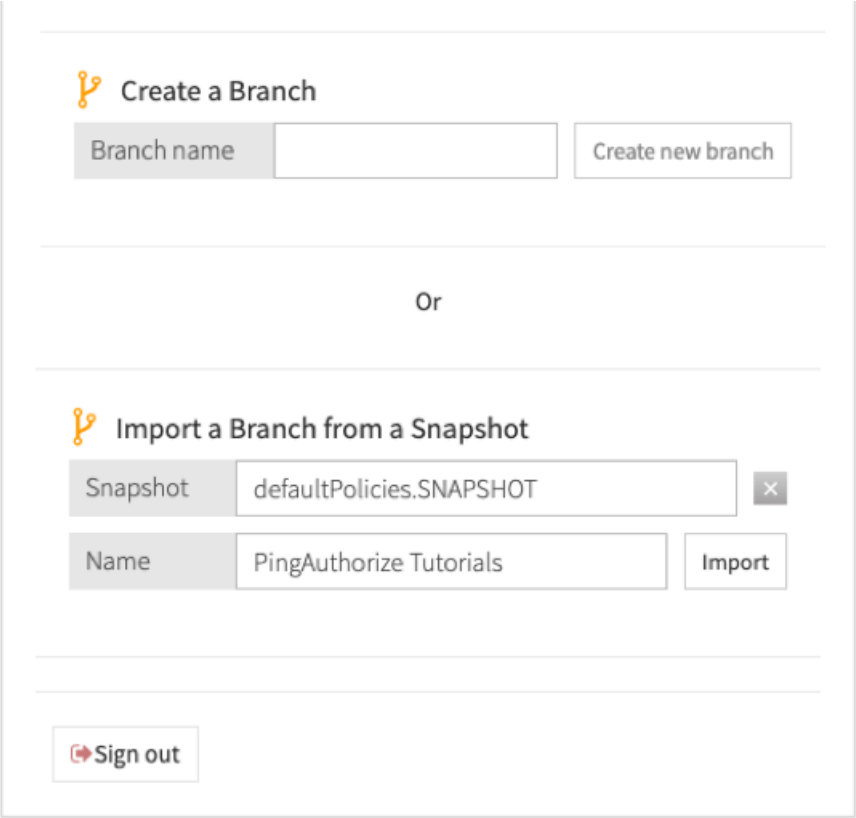
Steps

1. Copy `defaultPolicies.SNAPSHOT` from the PingAuthorize Policy Editor container to the current directory on your computer using the following command.

   > **Note:**
   >
   > Be sure to include the trailing `.` character.

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

2. Sign on to the Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
3. In the **Import a Branch from a Snapshot** section, click **Snapshot** and select the file that you just copied to your computer.
4. In the **Name** field, enter `PingAuthorize Tutorials`.



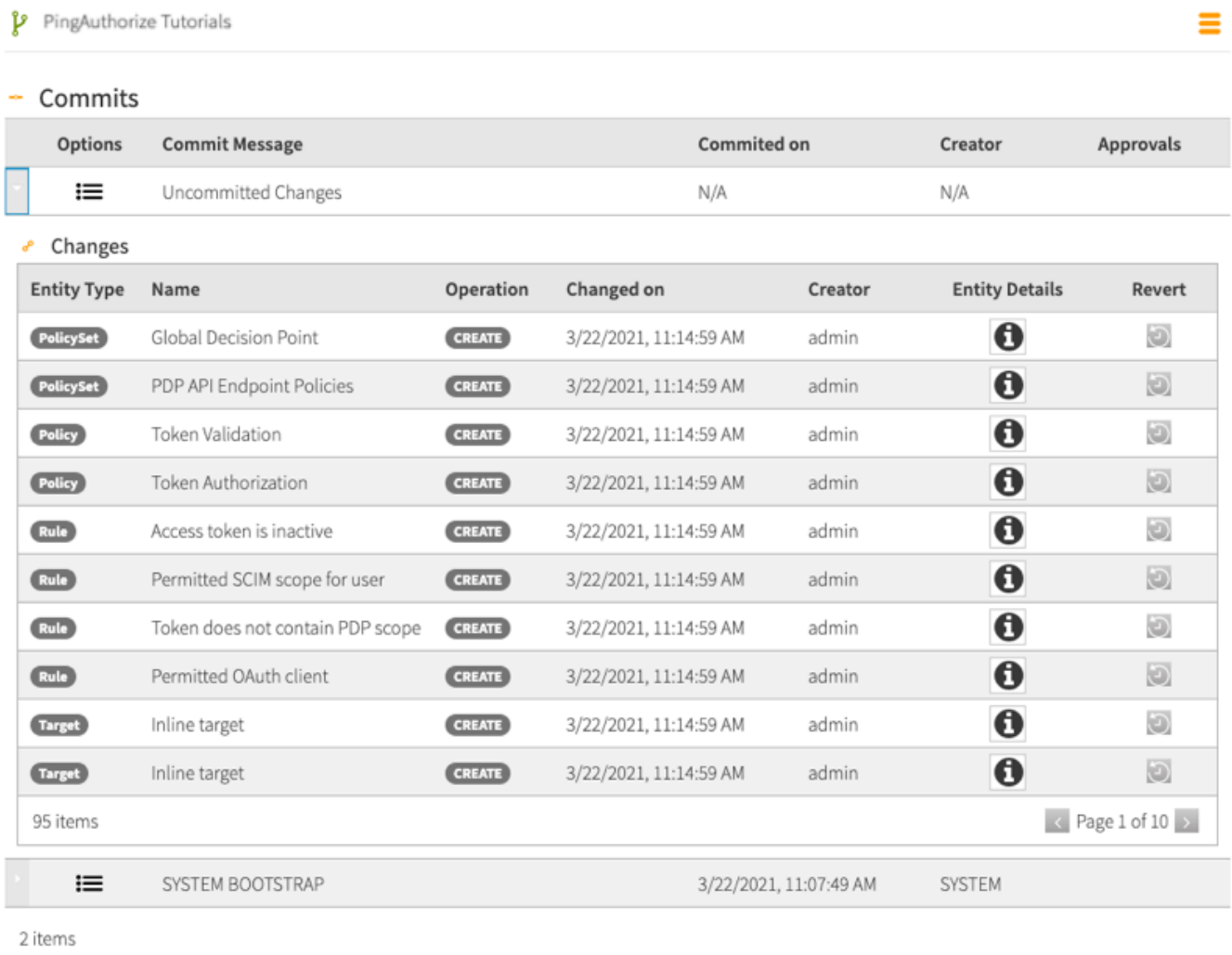
5. Click **Import**.

   Result: The Policy Editor 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. To select the policy branch that you just created, click **PingAuthorize Tutorials**.

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

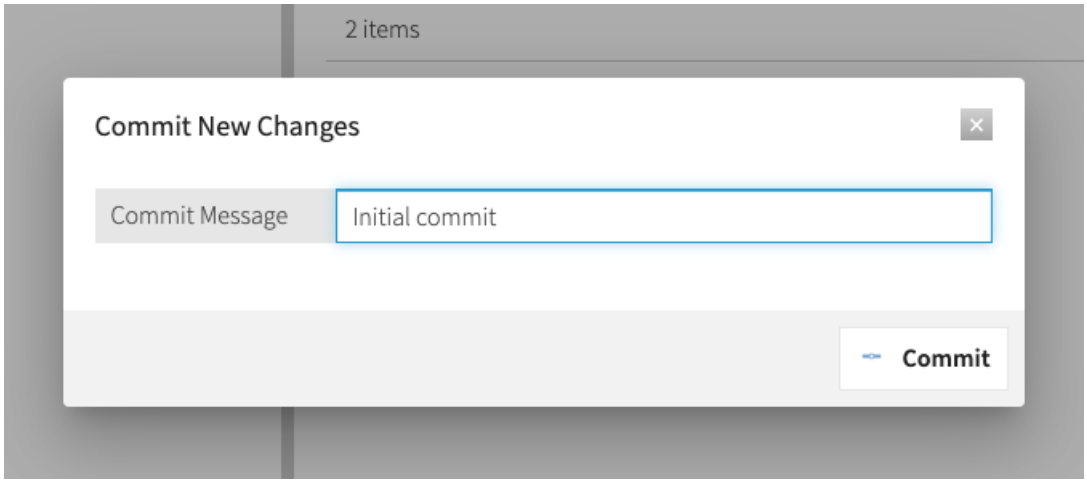
7. Click the expand arrow at the left of the top line for **Uncommitted Changes**.

   Result: This opens 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.



8. Click **Commit New Changes**.

9. In the **Commit Message** field, enter `Initial commit`. Click **Commit**.



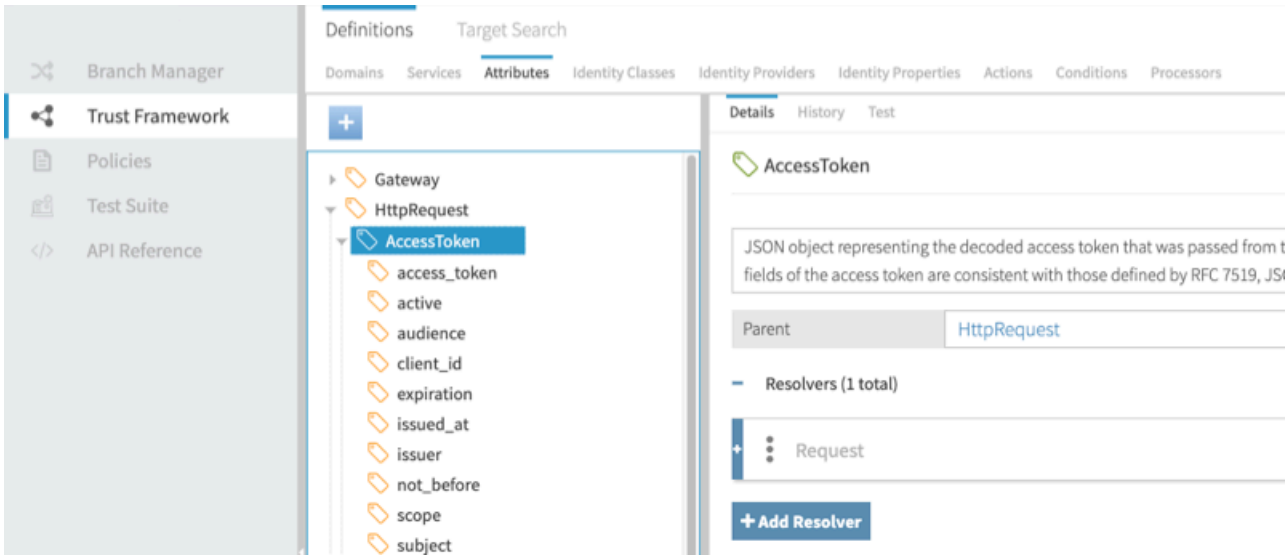
> **Tip:**
>
> As you work with your own policies, you can use the Policy Editor's version control feature to manage your changes. As you develop policies, a good practice is to set a checkpoint every time you achieve a satisfactory working state by *committing your changes*.

Introduction to the Trust Framework and default policies

You can now use the Policy Editor with PingAuthorize 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.



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

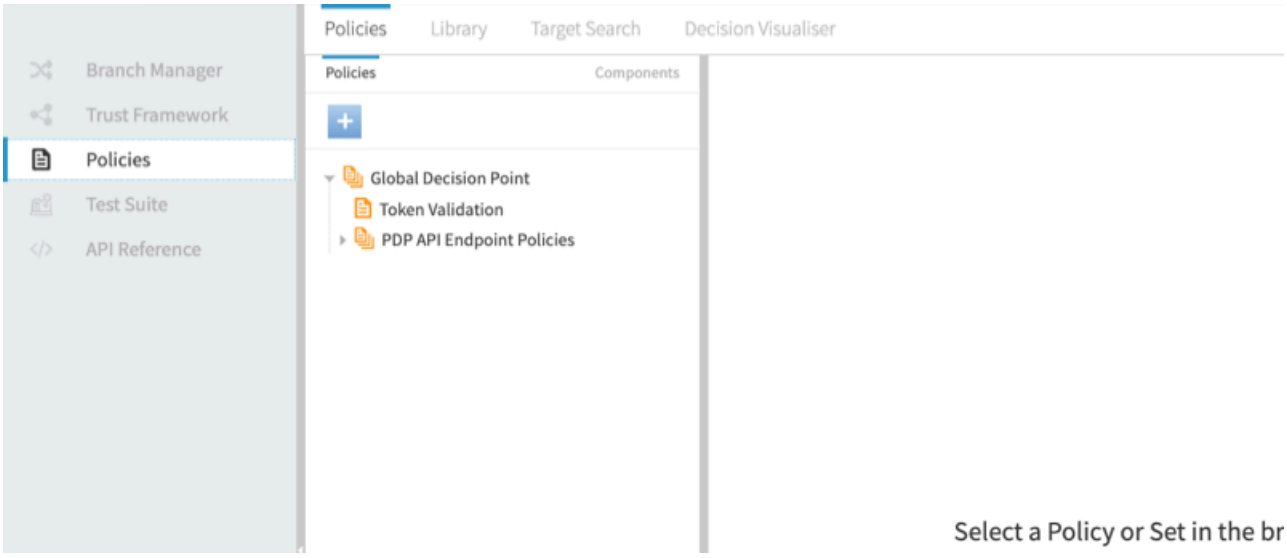
Entity	Description
Services	Services perform two functions. Most often, they represent a specific API service or *API resource type* to be protected by your policies. They can also define *[[policy information points]]* , external data sources (such as APIs or LDAP directory servers) that PingAuthorize can use to make policy decisions.
Attributes	*Attributes* on page 417 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 PingAuthorize 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 PingAuthorize Server *XACML-JSON 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 2: Configuring fine-grained access control for an API

This tutorial shows you how to set up PingAuthorize for attribute-based access control of a JSON REST 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.

Access control granularity type	Description
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*.

Note:

The Meme Game API is publicly available and does not need to be installed for the PingAuthorize tutorials.

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 Editor.
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.

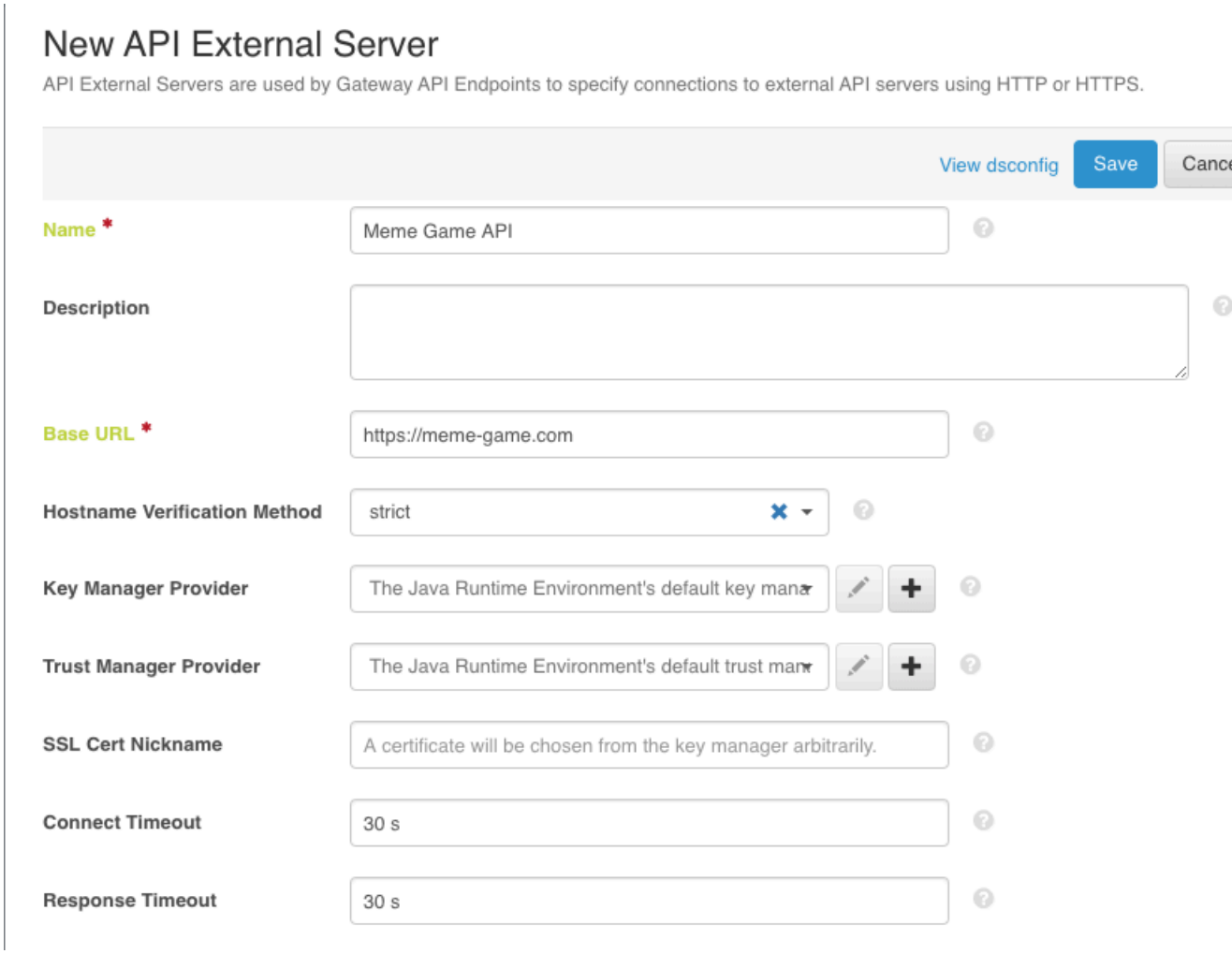
Configuring a reverse proxy for the Meme Game API

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.

Steps

1. Configure an API External Server for the Meme Game API. An API External Server controls how PingAuthorize 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 19.
   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.

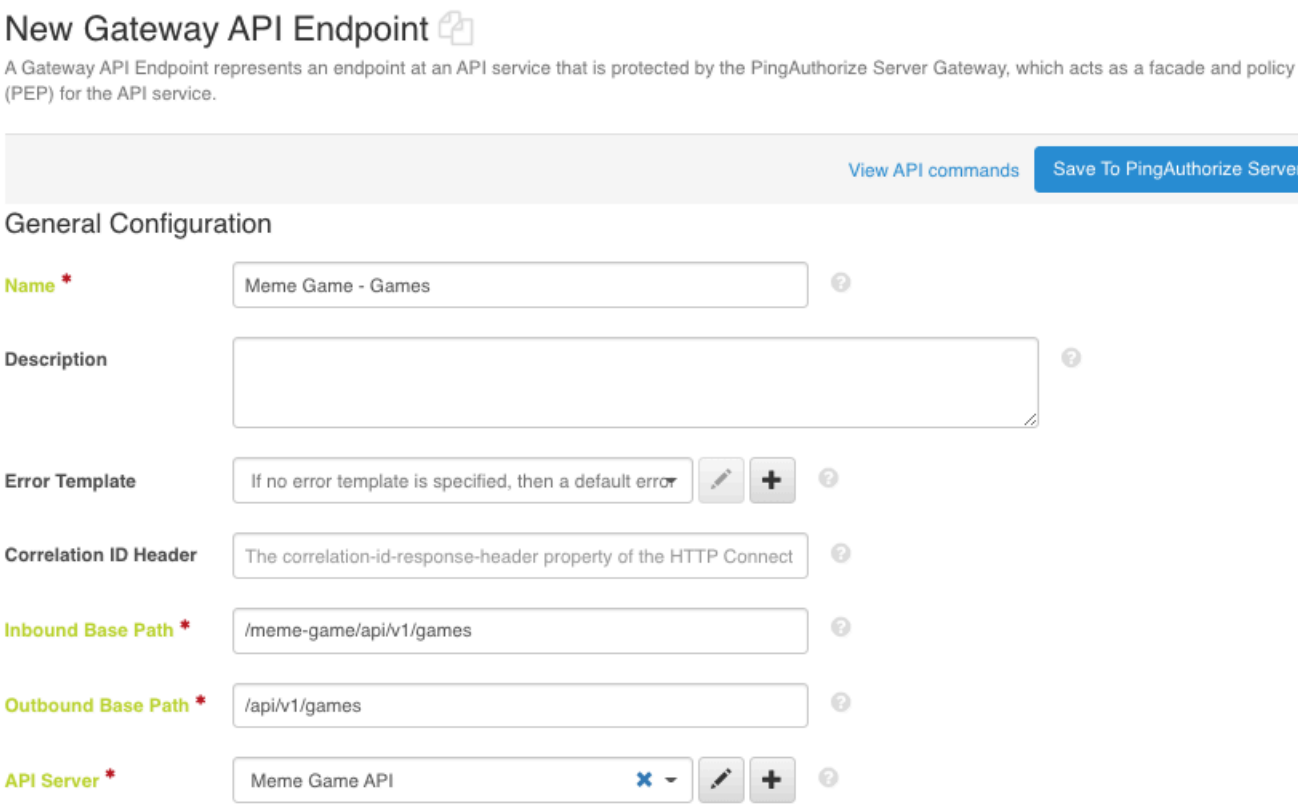


   f. Click **Save**.

2. Configure a Gateway API Endpoint. A *Gateway API Endpoint* controls how PingAuthorize 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 PingAuthorize Server.
   e. For **Outbound Base Path**, specify `/api/v1/games`.

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



   g. Save your changes.

Testing the reverse proxy

PingAuthorize 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 PingAuthorize 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 PingAuthorize API security gateway, part 1* on page 30.

Result:

If the PingAuthorize 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 PingAuthorize 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 PingAuthorize Server configuration. Each Gateway API Endpoint determines which incoming HTTP requests are proxied to an API server and *how PingAuthorize Server translates the HTTP request into a policy decision request*.

Bearer token authorization

The testing in *Testing the reverse proxy* on page 29 uses this authorization. The token itself is a *[[mock access token]]* , which is a special kind of Bearer token that a PingAuthorize 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 PingAuthorize 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.

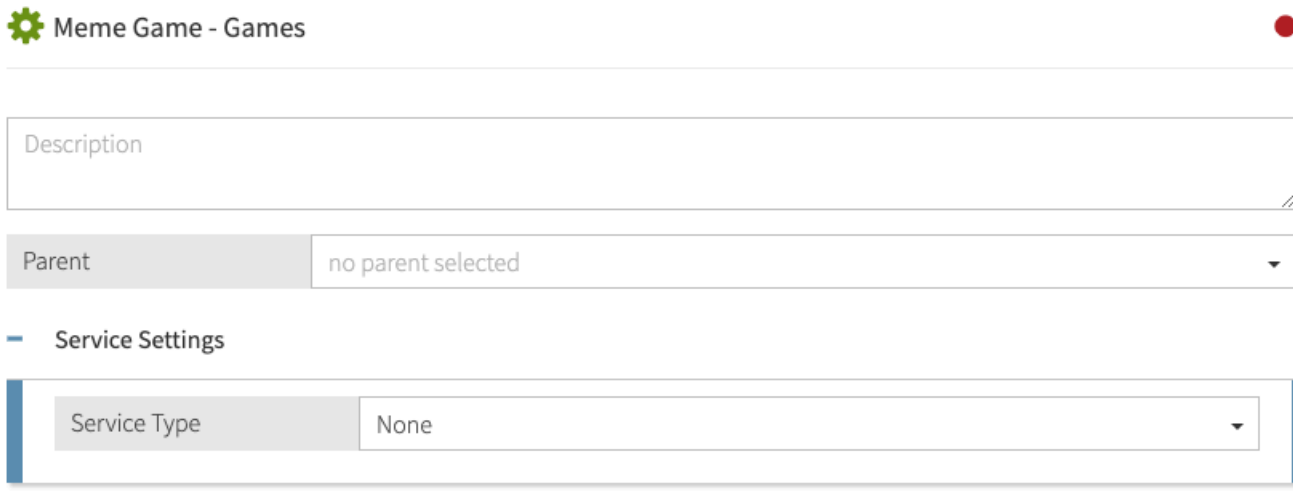
Then, we define a policy. This policy will reject any requests to start a new meme game if the user's identifier ends with `@example.com`. We will identify users using the subject of the request's access token.

Steps

1. Define the service.
   a. Sign on to the Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
   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 PingAuthorize API security gateway, part 2* on page 32.
   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.



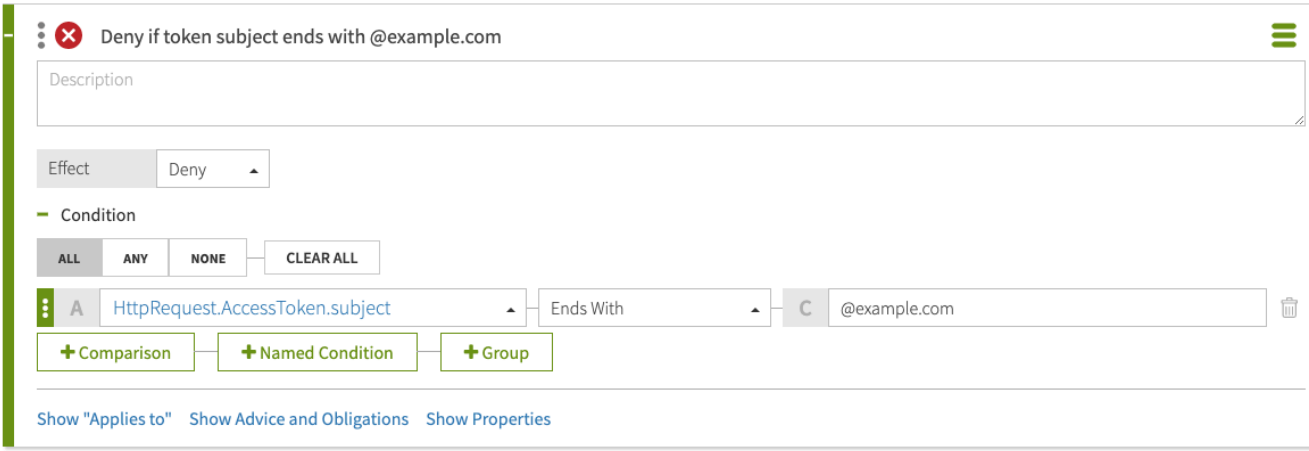
   f. Click **Save changes**.

2. Define the policy.
   a. In the Policy Editor, 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 **Add definitions and targets, or drag from Components** box.
   h. From the **Services** list, drag **Meme Games - Games** to the **Add definitions and targets, or drag from Components** 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** list, select **HttpRequest.AccessToken.subject**.
         c. In the second field, select **Ends With**.
         d. In the third field, type `@example.com`.

      The following screen shows the rule.



   k. Click **Save changes**.

      For more information about API security gateway processing, see *For further consideration: The PingAuthorize API security gateway, part 2* on page 32.

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

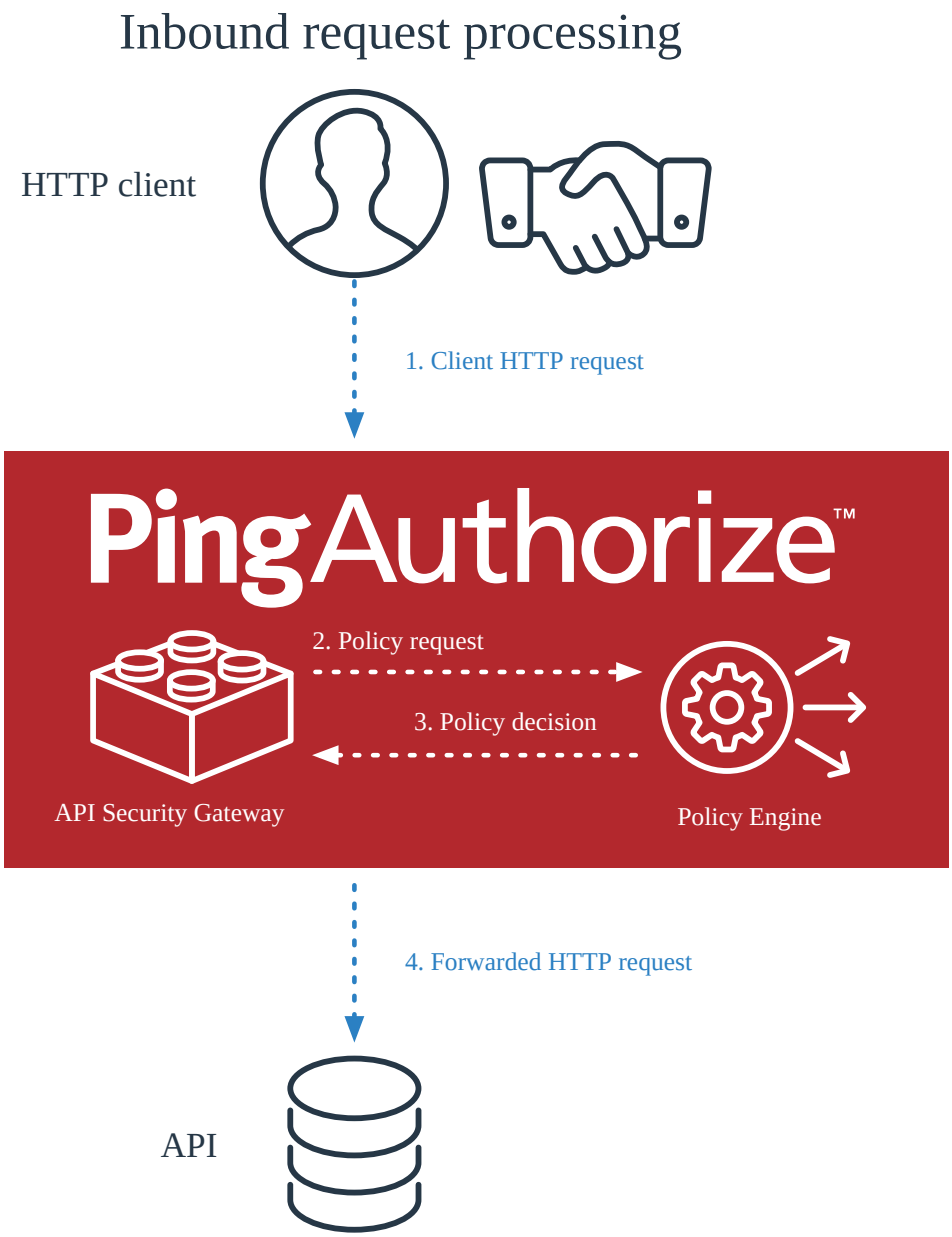
Additional concepts to consider include the phases of API security gateway processing and the need for the service name to match the Gateway API Endpoint name.

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.

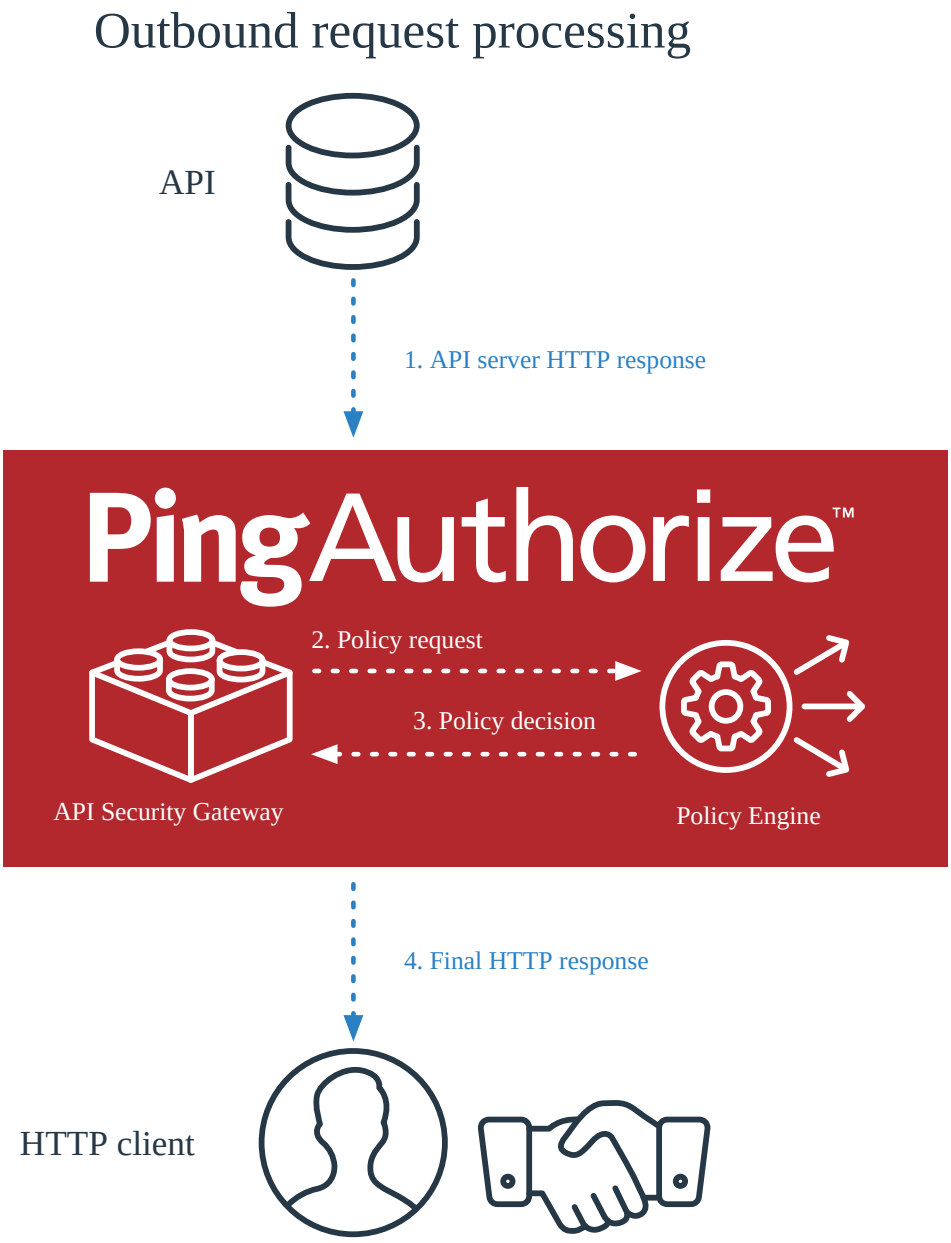
The following diagram illustrates the inbound request processing.



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 31, we named the service to match the name of the Gateway API Endpoint in the PingAuthorize configuration. This is important. When PingAuthorize 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 Editor

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

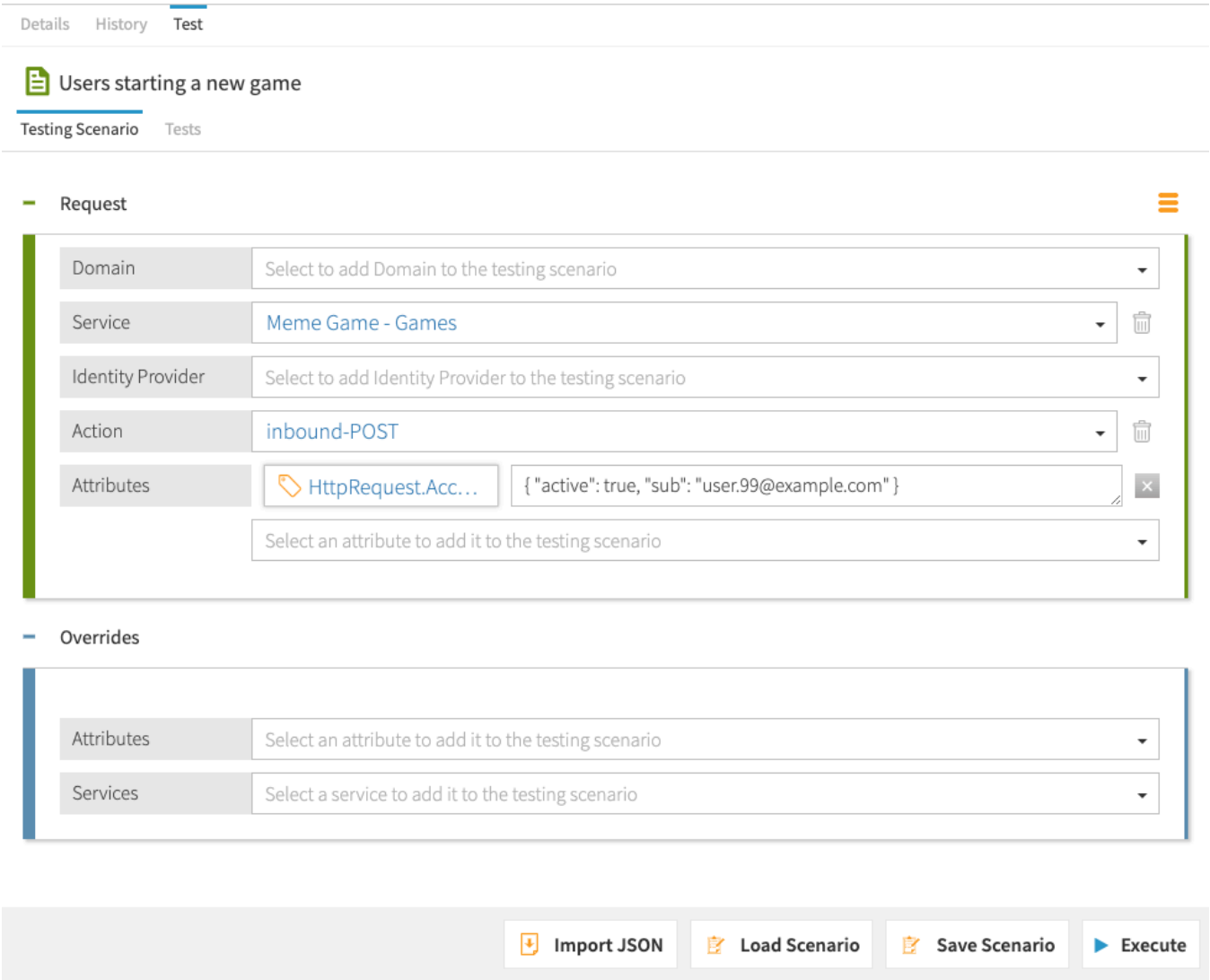
Steps

1. In the Policy Editor, 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 PingAuthorize Server makes when it receives an HTTP request.

Description	Details
Service	**Meme Games - Games**
Action	**inbound-POST**
Attributes	**HttpRequest.AccessToken** `{ "active": true, "sub": "user.99@example.com" }`

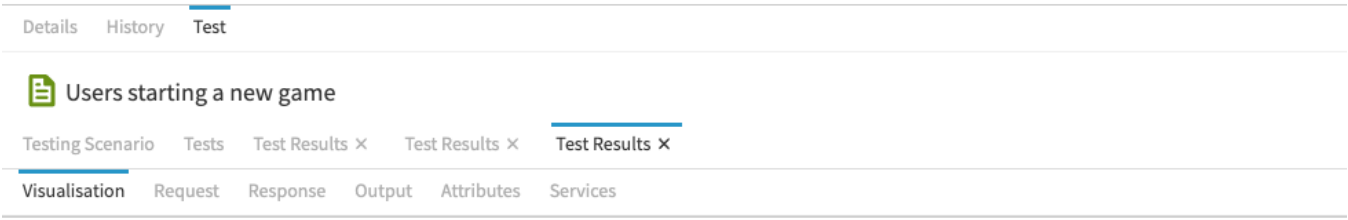
The following image shows the test.



3. Click **Execute**.

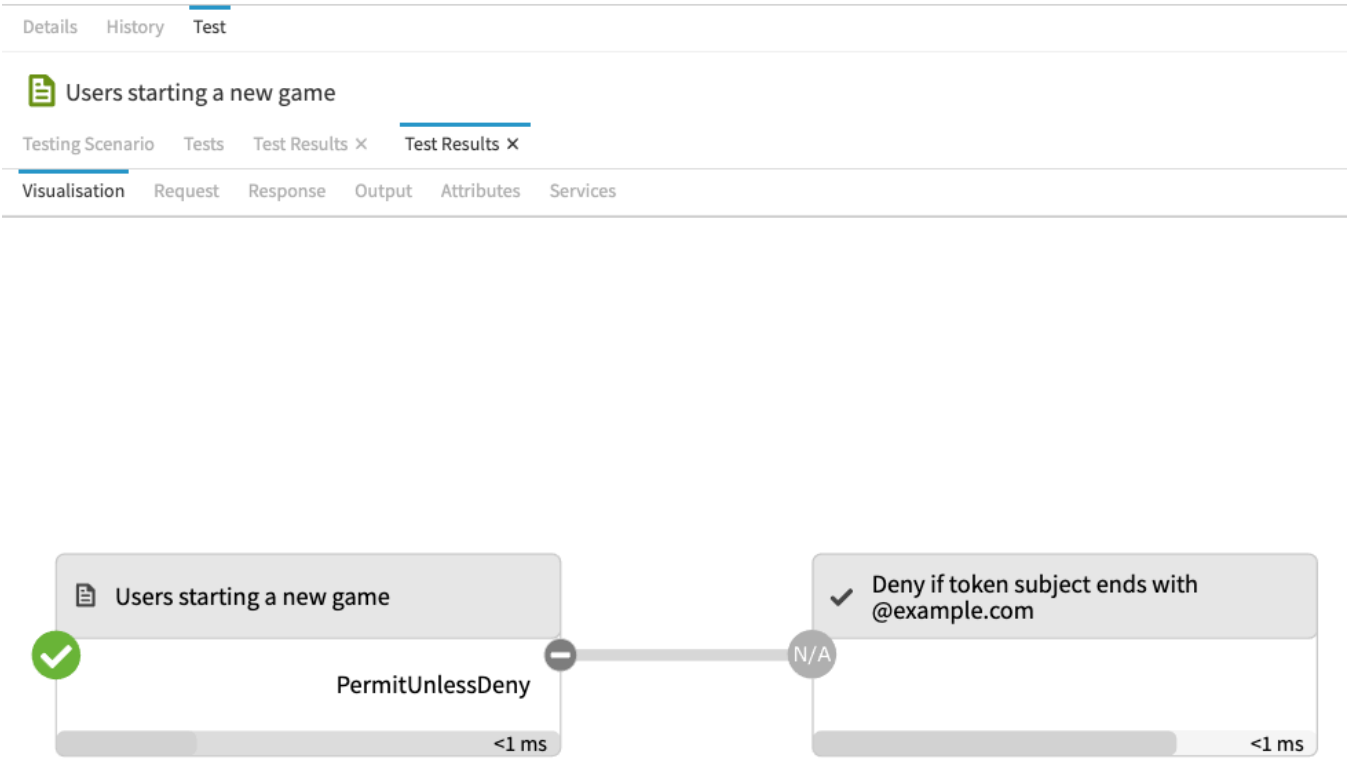
   Result:

   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 Editor to prove the policy works as intended, we can confirm that policy enforcement from end-to-end by sending an HTTP request through the PingAuthorize Server reverse proxy.

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"
            ]
        }
    }
```

```
}'
```

Result:

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"
            ]
        }
    }
}'
```

   Result:

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

   To better understand how policy decisions work, see *For further consideration: Decision Visualiser* on page 38.

For further consideration: Decision Visualiser

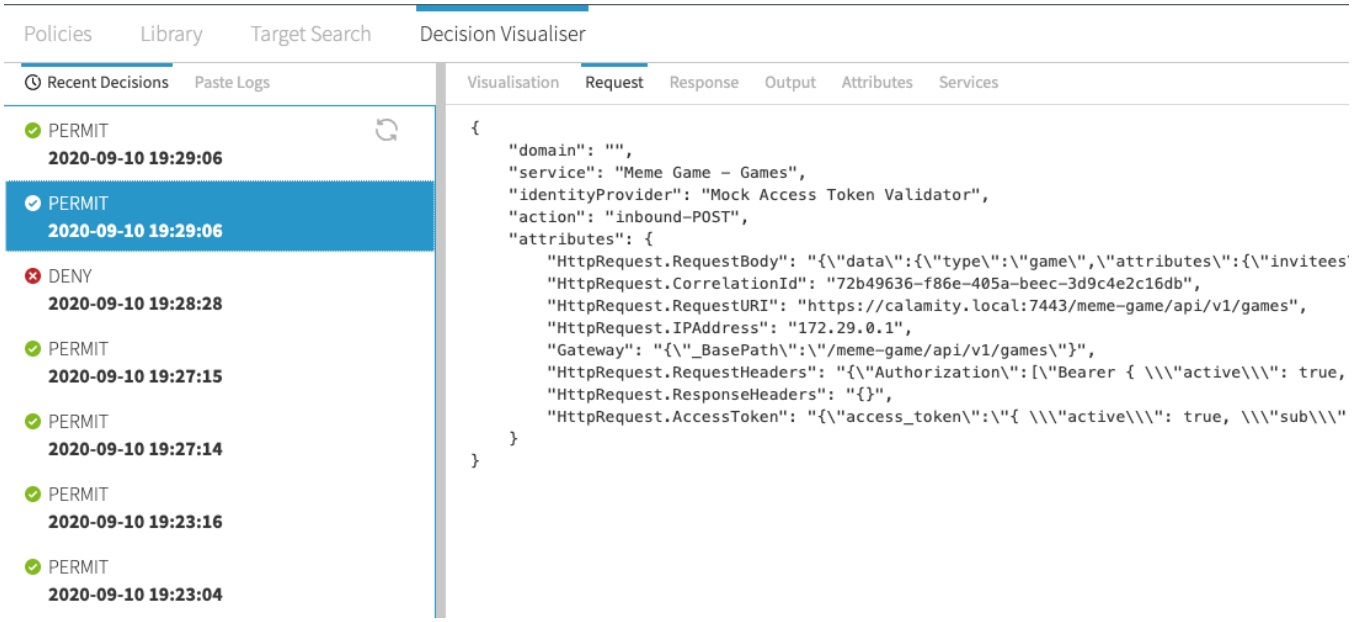
Returning to the Policy Editor, we can view a log of how the policy engine handled the HTTP request.

Steps

1. In the Policy Editor, 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.

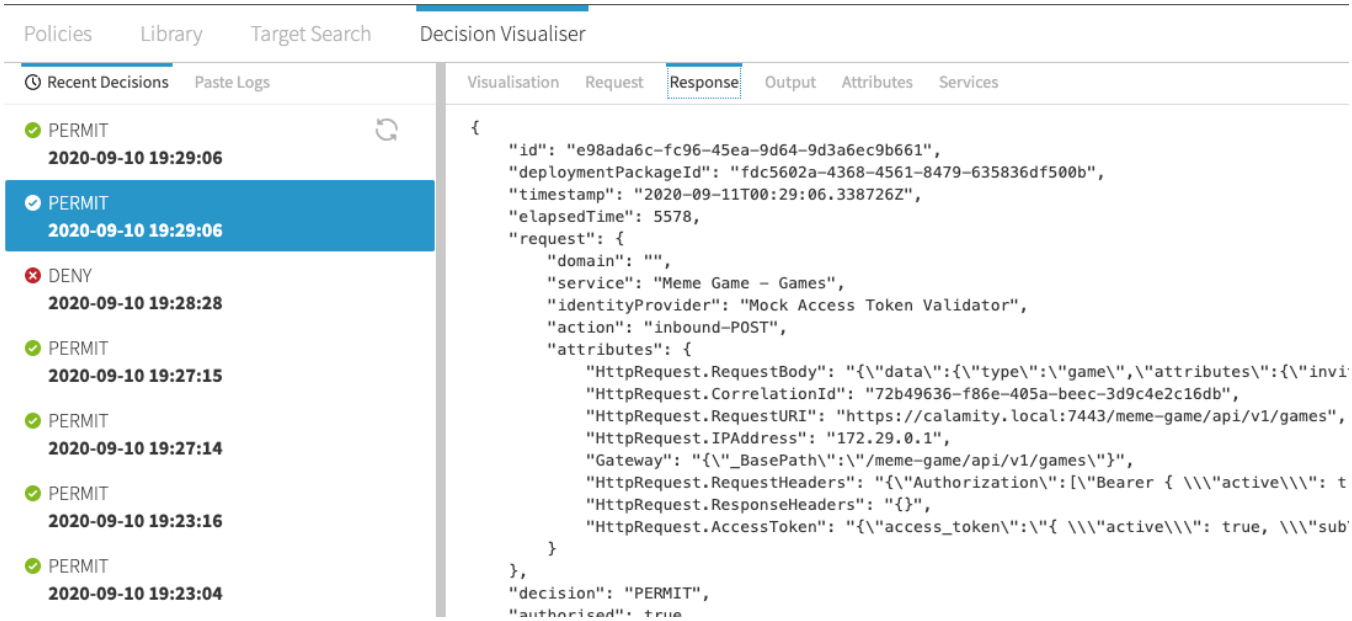
4. Click the **Request** tab. This displays a JSON representation of the policy request that PingAuthorize generated to represent your HTTP request.

   Here is a request example.



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.



   Both the policy request and the policy response might be hard to understand at the moment, but as you become familiar with PingAuthorize 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"]
        }
    }
}
```

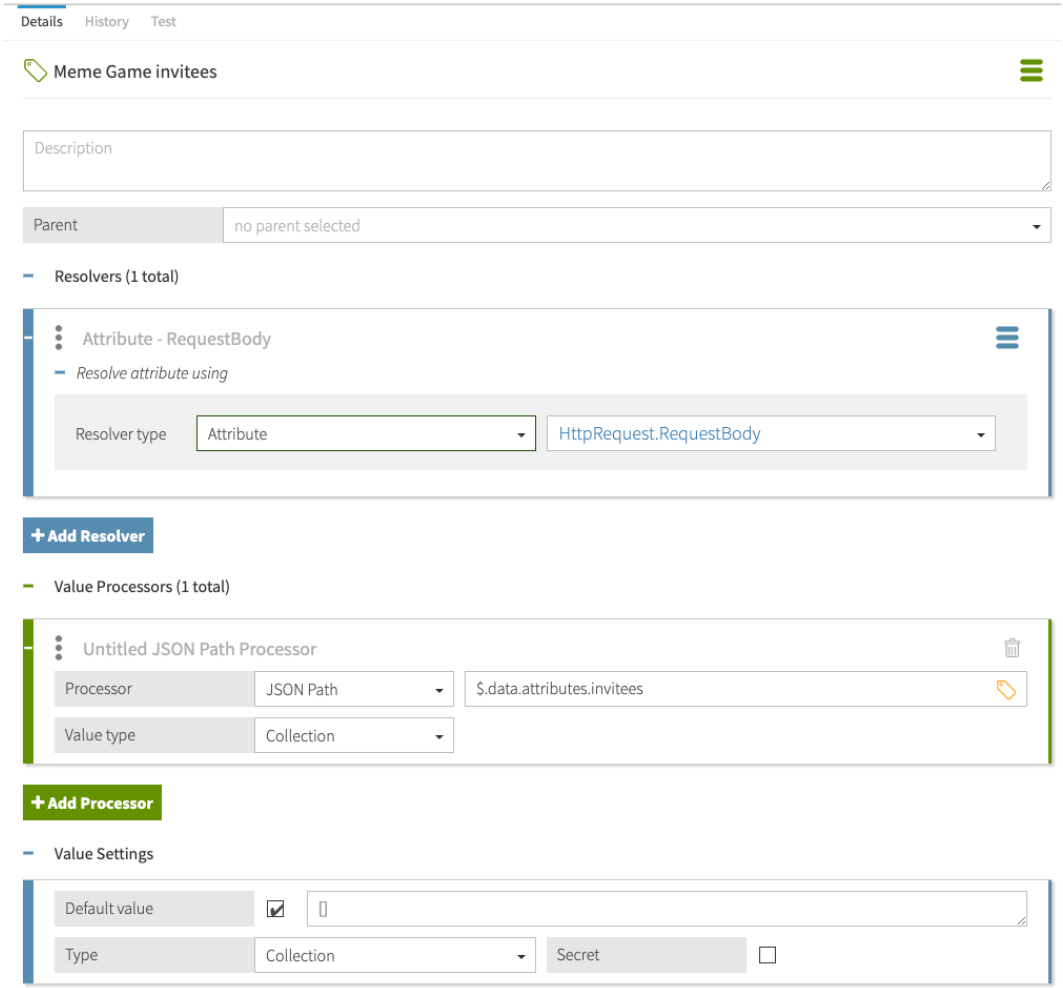
The requester specifies one or more invitees using the `data.attributes.invitees` field. We will update our policy with a second rule that disallows a new game if anybody else is invited to it.

Steps

1. Define a Trust Framework attribute to represent the `data.attributes.invitees` field.
   a. In the Policy Editor, 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**.

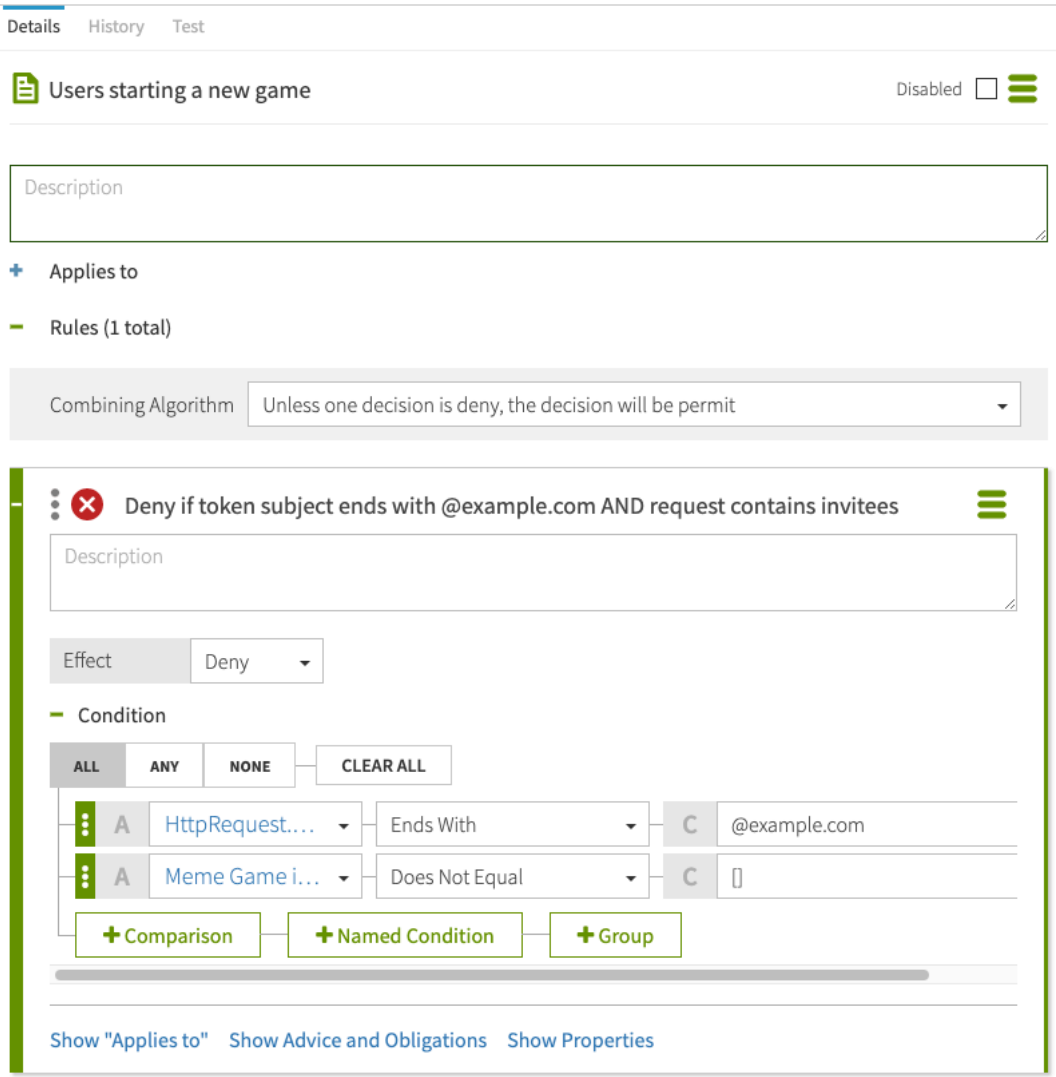
      The following image shows the new attribute.



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 43.

2. Modify a rule to use the **Meme Game invitees** attribute we just created.
   a. In the Policy Editor, 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** list, select **Meme Game invitees**.
      3. In the second field, select **Does Not Equal**.
      4. In the third field, type `[]`.
   g. Click **Save changes**.

   The following image shows the rule.



3. Test the policy.

   As before, you can test your policy from the Policy Editor 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 39 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 PingAuthorize 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 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 PingAuthorize Server to act as a reverse proxy to a JSON API. You then learned how to use the PingAuthorize Policy Editor 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 Editor.

You also learned:

- Gateway API Endpoint names in the PingAuthorize Server configuration must match Trust Framework Service names in the Policy Editor.
- 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.
- PingAuthorize Server supplies Attributes for HTTP metadata, request data, and OAuth 2 access token attributes.
- You can test policies directly from the Policy Editor.
- The Policy Editor's Decision Visualiser gives you a detailed view of recent policy decisions.

Tutorial 3: Configuring attribute-based access control for API resources

This tutorial describes how to build and test fine-grained access control (FGAC) 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 ages 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.

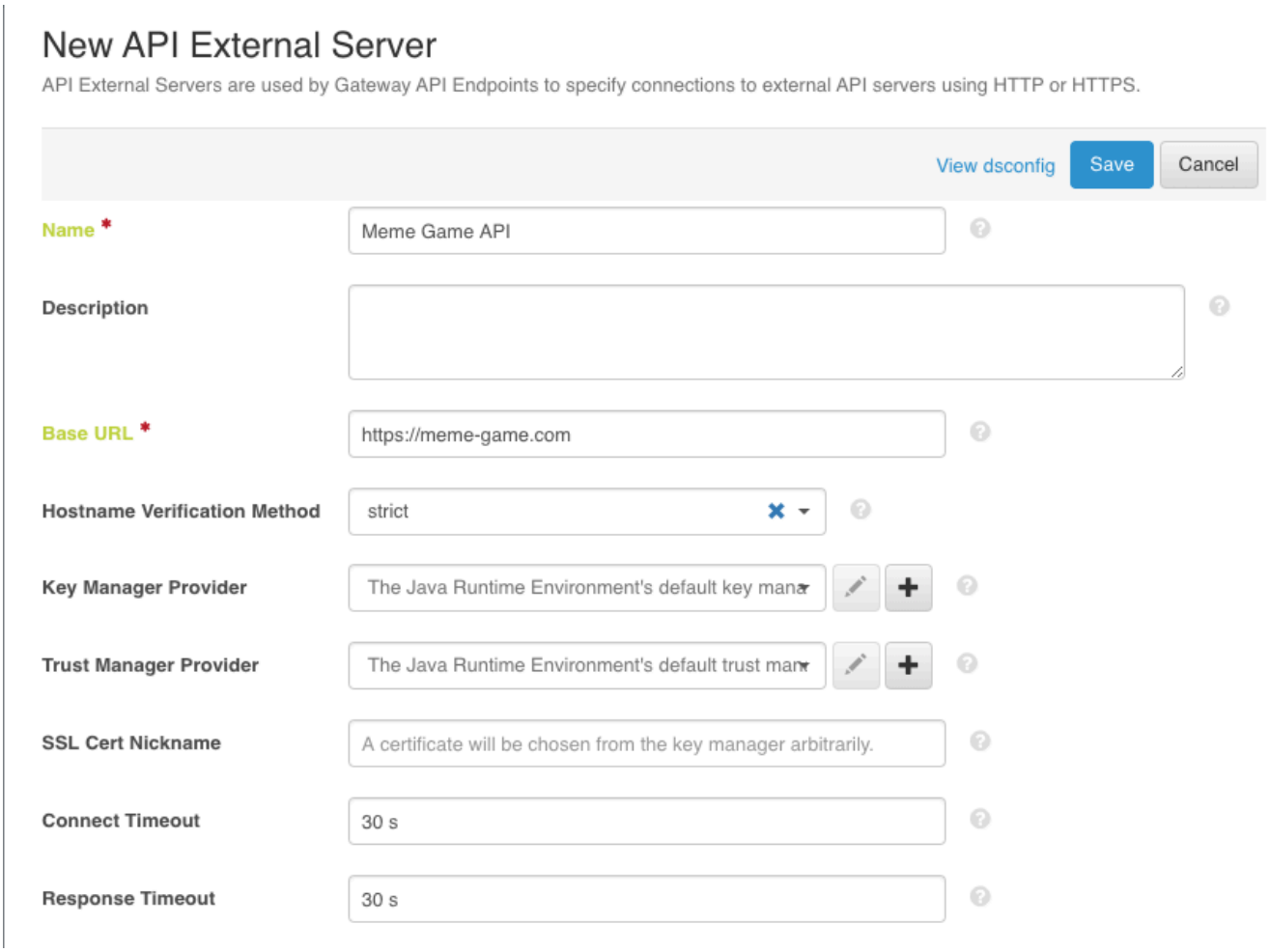
Steps

1. (Optional.) Configure an API External Server for the Meme Game API. An API External Server controls how PingAuthorize 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 2: Configuring fine-grained access control for an API* on page 26, 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 19.
   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.



   f. Click **Save**.

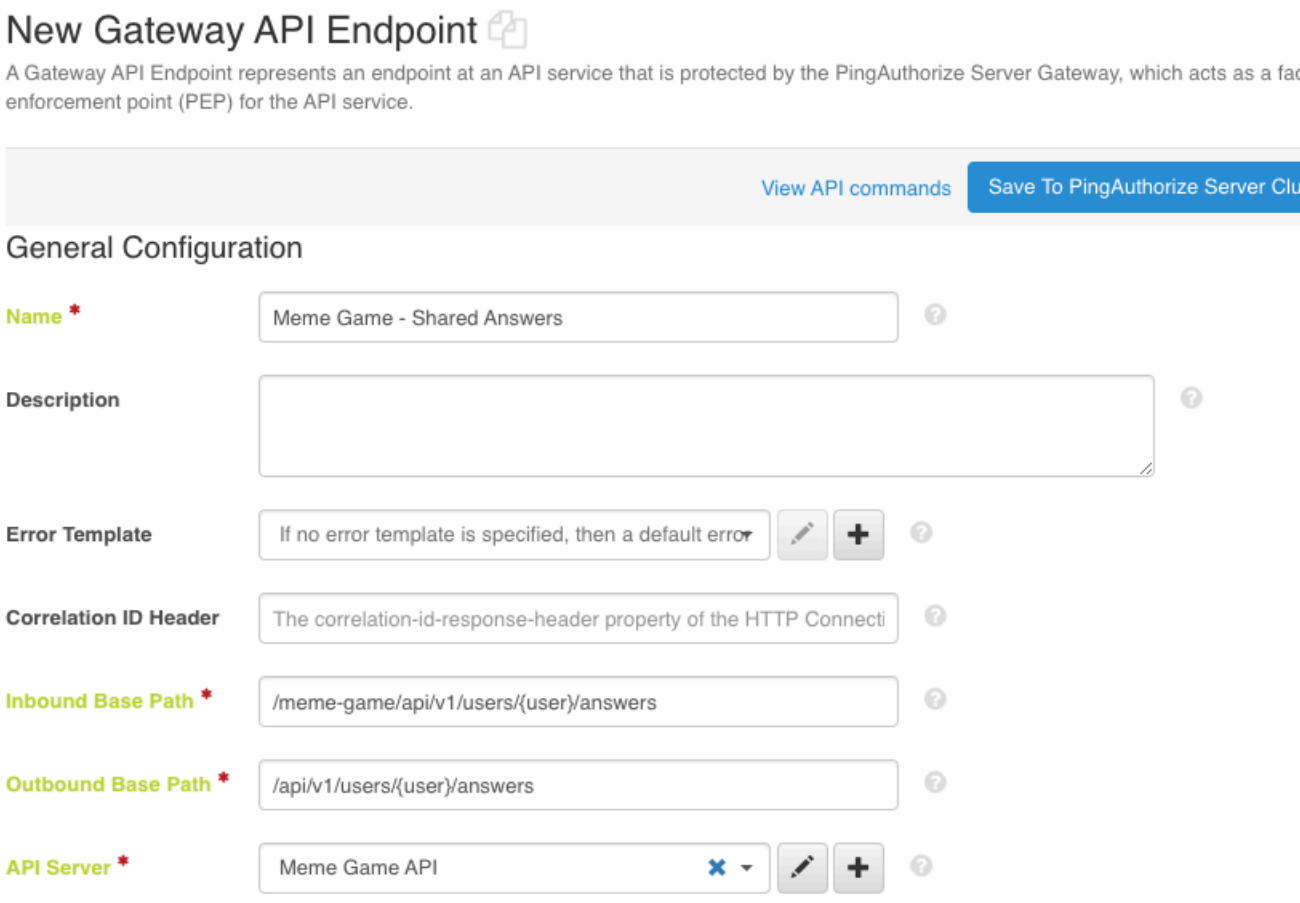
2. Configure a Gateway API Endpoint. A Gateway API Endpoint controls how PingAuthorize Server proxies incoming HTTP client requests to an upstream API server.
   a. In the administrative console, click **Configuration** and then **Gateway API Endpoints**.
   b. Click **New Gateway API Endpoint**.
   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 PingAuthorize 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 PingAuthorize 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 28.

      Your screen should look like the following one.



   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"}'
```

  Result: 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

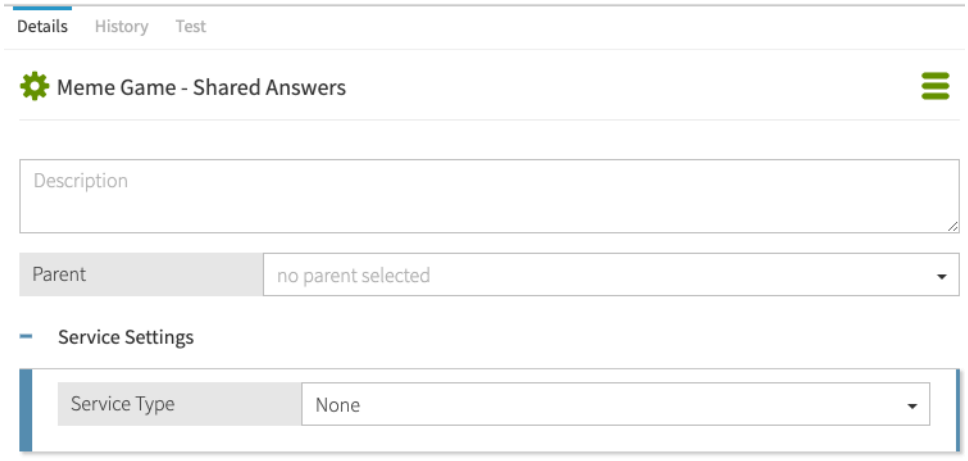
This task passes the name of the Gateway API Endpoint configured in PingAuthorize Server as the service to the PingAuthorize policy decision point (PDP).

Steps

1. From the PingAuthorize Policy Editor, 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 delete icon to the right of the Parent field.

   Your service should look like the example in the following image:



5. Click **Save changes**.

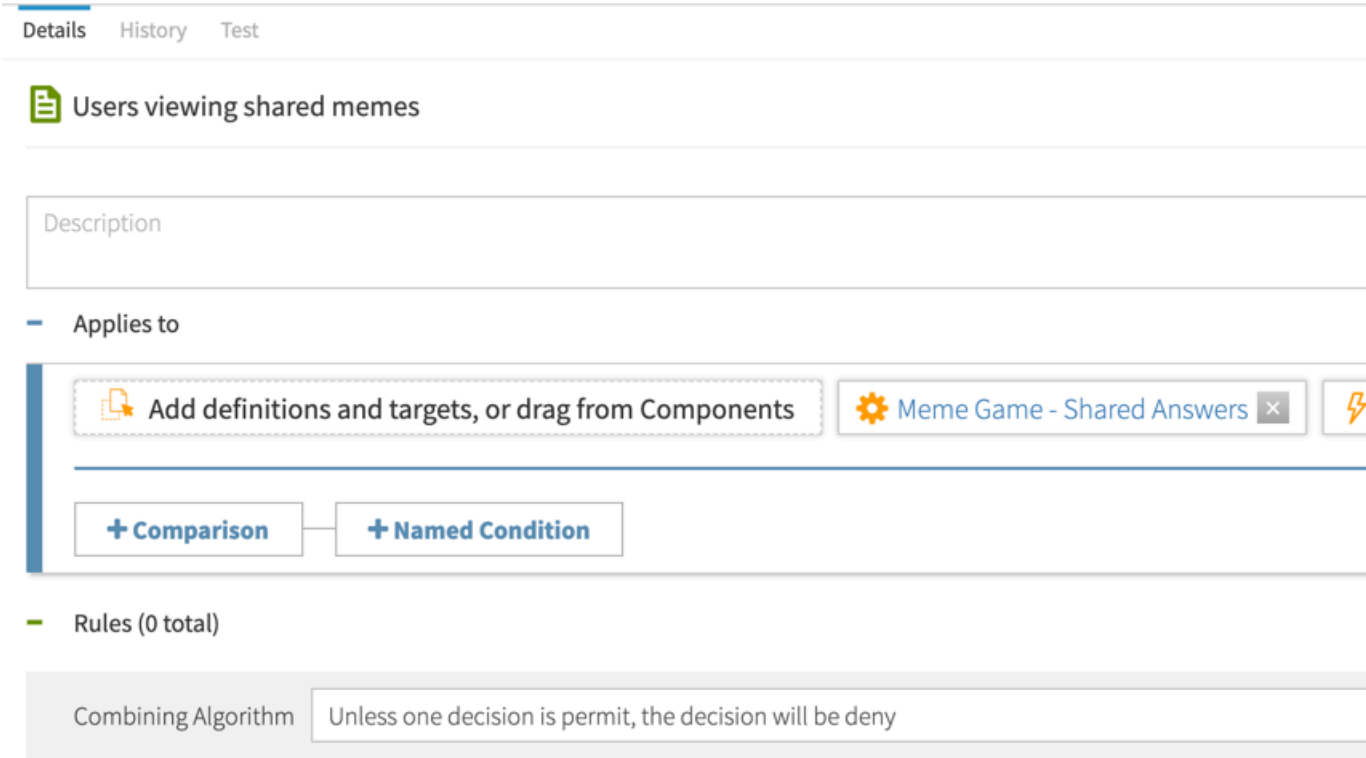
Creating a policy for the Shared Answers endpoint

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

Steps

1. In the PingAuthorize Policy Editor, 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 **Add definitions and targets, or drag from Components** box.
8. From the **Services** list, drag **Meme Game - Shared Answers** to the **Add definitions and targets, or drag from Components** 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.



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"}'
```

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

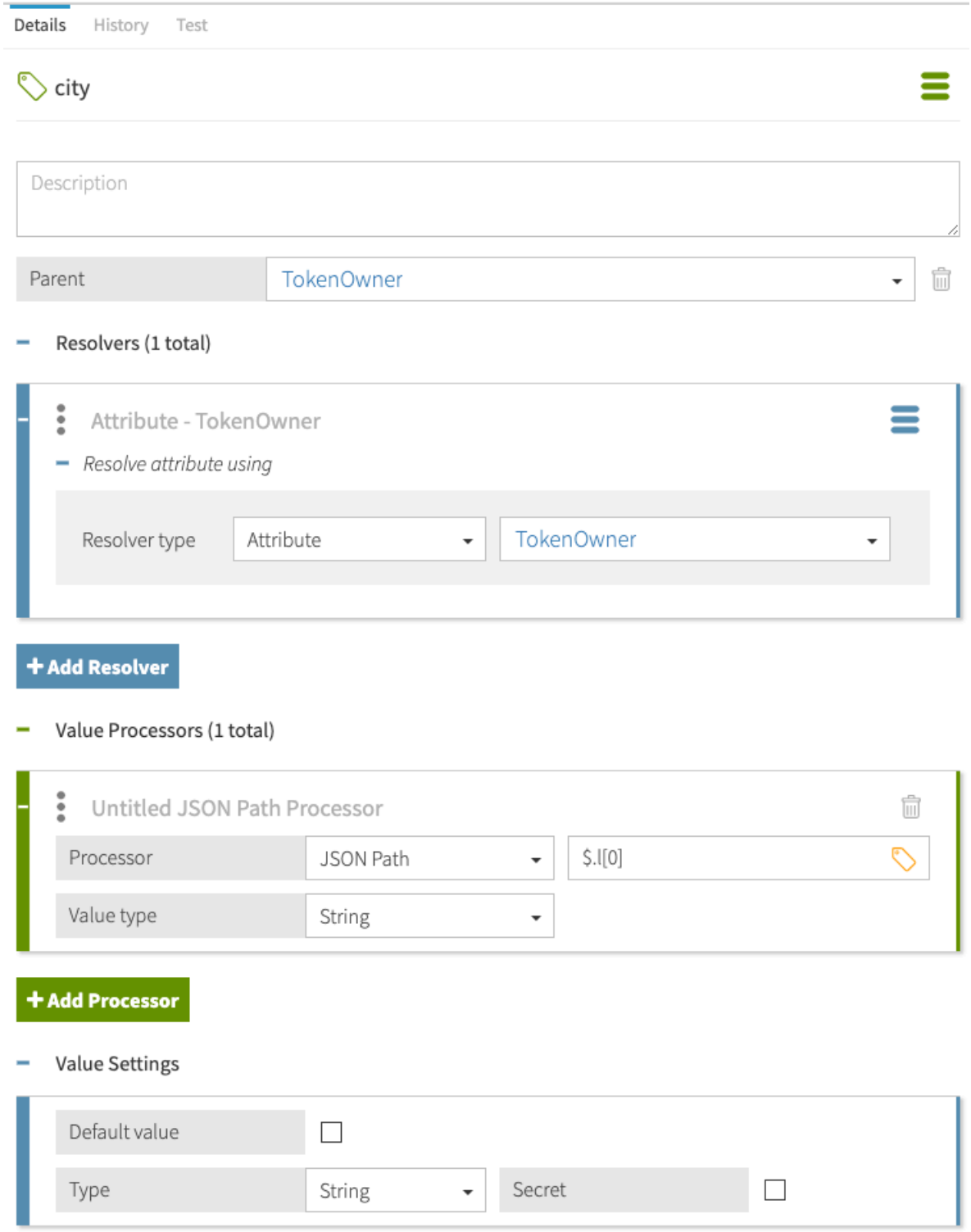
```
{
 "errorMessage": "Access Denied",
 "status": 403
}
```

Creating an attribute from user data
Create an attribute to represent the city the user lives in.

Steps

1. In the PingAuthorize Policy Editor, 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**.
    Result: You have an attribute for the user's city, as shown in the following image.

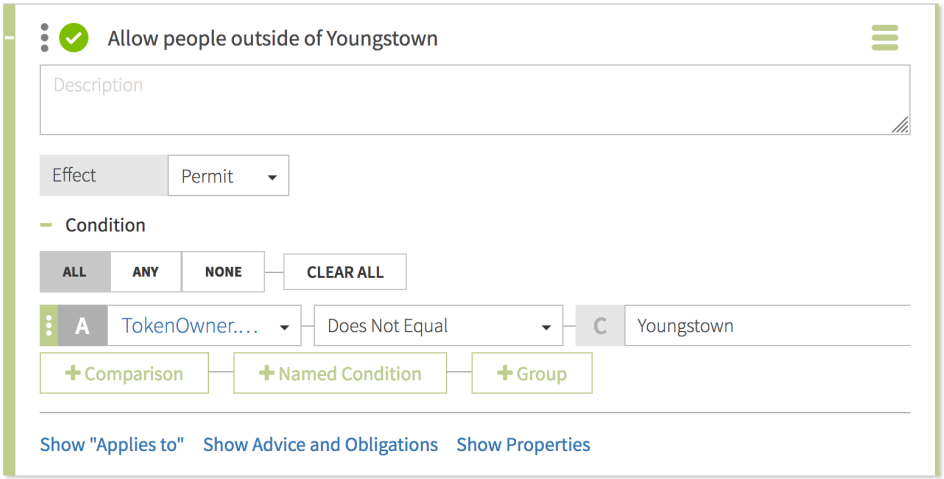


Adding logic to allow non-Youngstown users

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 PingAuthorize Policy Editor, 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** list, select **TokenOwner.city**.
   c. In the second field, select **Does Not Equal**.
   d. In the third field, type `Youngstown`.
7. Click **Save changes**.
   Result: You have a rule that allows users from outside Youngstown.



Testing that the policy blocks Youngstown users

You can test the new rule with cURL or Postman.

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"}'
```

   Result: 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"}'
```

Result: 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.

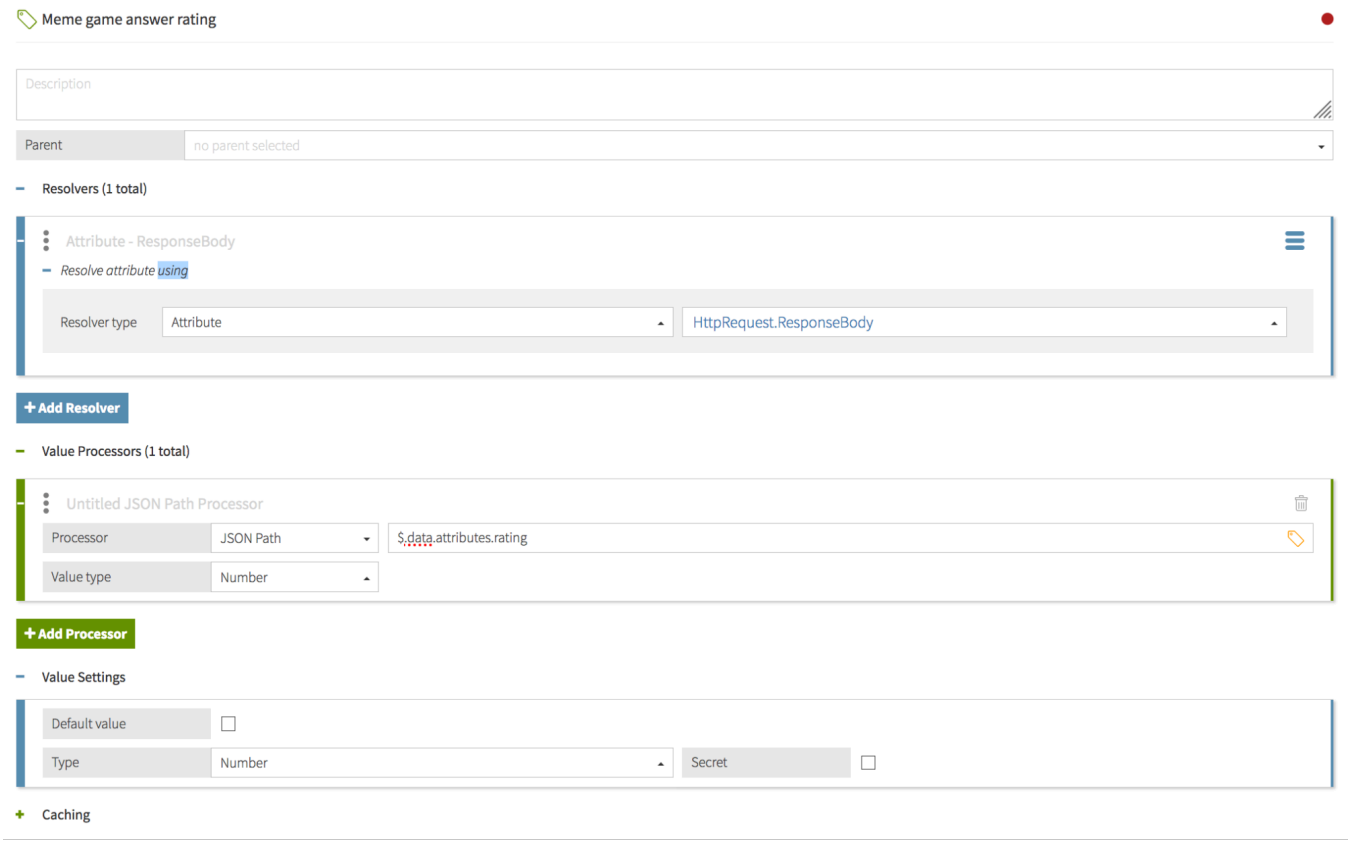
Creating an attribute from response data

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

Steps

1. From the PingAuthorize Policy Editor, 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**.
    Result: You have a new attribute for the answer's age rating.

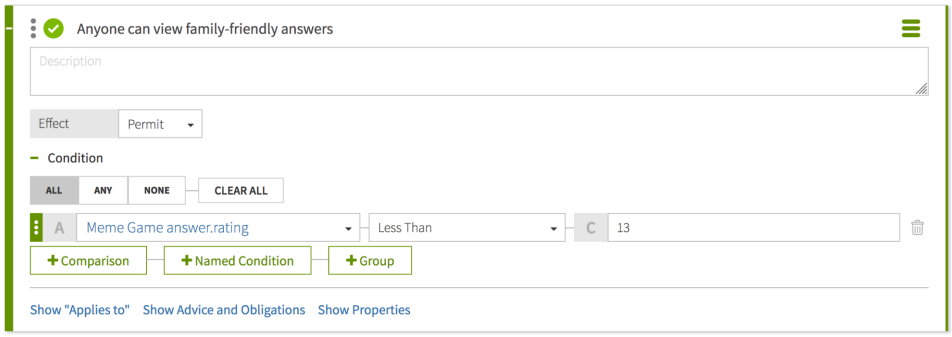


Adding logic to allow family-friendly memes
Add a rule to the **Users viewing shared memes** API policy to allow users to view answers that are rated for ages under 13.

Steps

1. From the PingAuthorize Policy Editor, 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** list, select **Meme game answer rating**.
   c. In the second field, select **Less Than**.
   d. In the third field, type `13`.

7. Click **Save changes**.
   Result: You have a rule to allow family-friendly memes that looks like the following image.



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

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"}'
```

   Result:

   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/2 \
```

```
  -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
```

Result:

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"}'
```

Result:

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/1 \
  -H 'Authorization: Bearer {"active": true, "sub": "user.660"}'
```

Result:

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

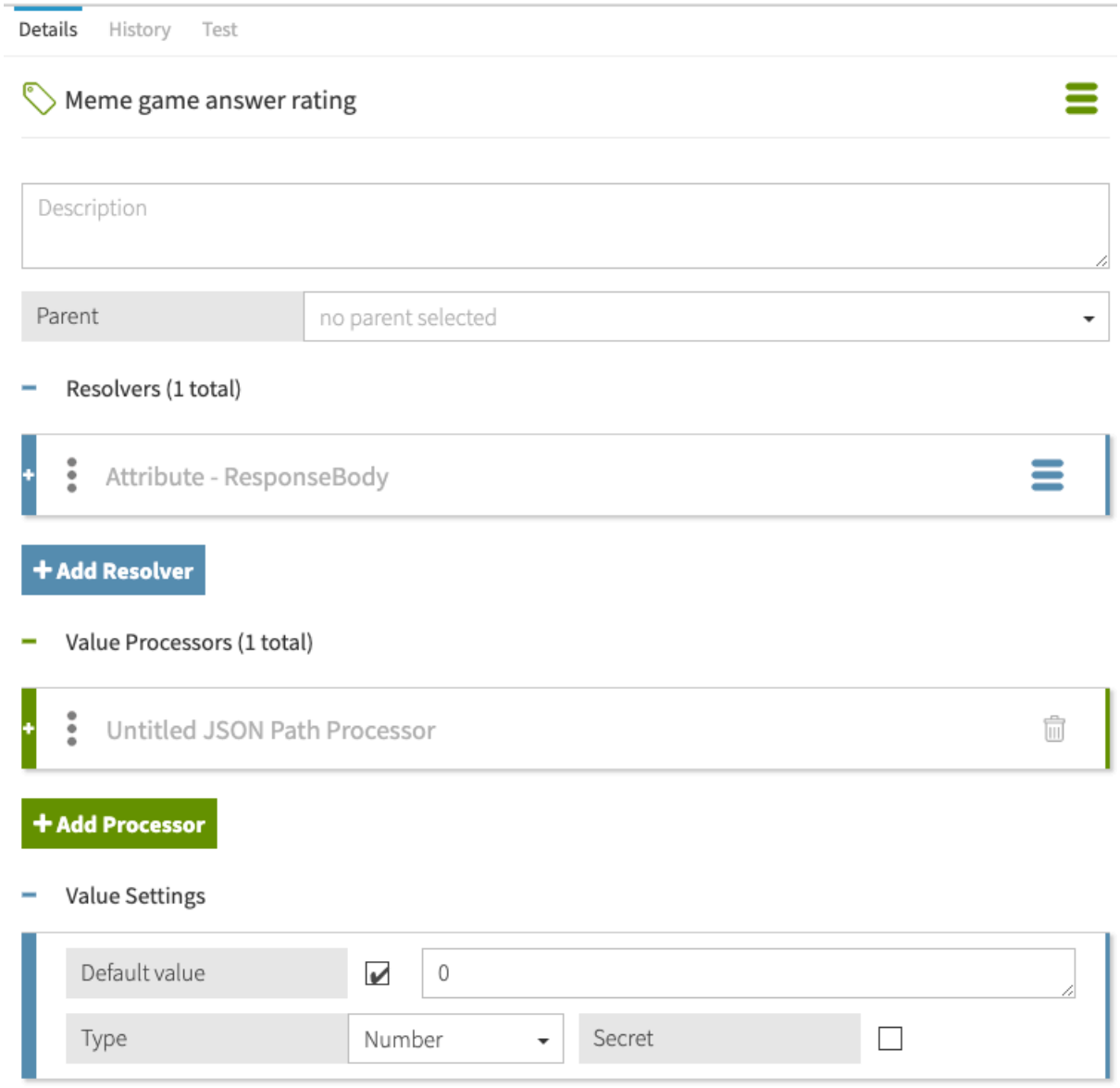
Answer 1 is not being served to `user.660`, even though it has not been rated as 13+. In this scenario, an unrated answer should be considered friendly to all users. Consider why an unrated meme is being blocked for this user. To resolve this, you can add a default value to the age rating.

Steps

1. In the PingAuthorize Policy Editor, 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**.

   Result:

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



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"}'
```

Result: 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 PingAuthorize to set the HTTP response code and provide a more useful error message when rejecting the outbound response.

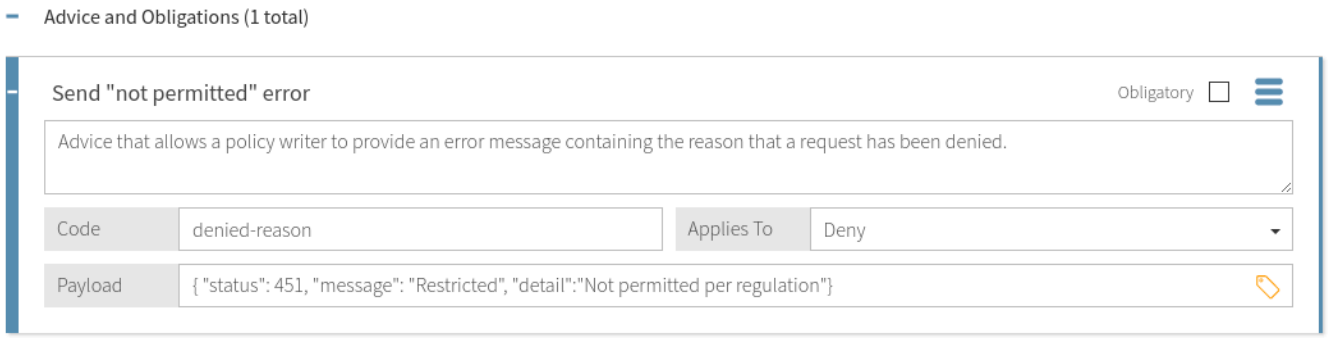
About this task

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 PingAuthorize Policy Editor, 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**.
   Result: You have a new advice, which looks something like the following image.



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"}'
```

  Result:

  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 PingAuthorize. 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 PingAuthorize 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 to 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.
- PingAuthorize supplies the user profile of the access token subject as the Trust Framework attribute `TokenOwner`.
- You must populate the child attributes of the `TokenOwner` that you want to use in a policy.
- Many attributes in LDAP are multivalued.
- Advice is used 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 (optional): Creating SCIM policies

This tutorial demonstrates how to develop fine-grained access control (FGAC) policies for the System for Cross-domain Identity Management (SCIM) REST API built into PingAuthorize Server.

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

While PingAuthorize Server's API security gateway protects existing REST APIs, PingAuthorize 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.

PingAuthorize 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 using only the default policies. As in the previous section, send a mock access token in the request.

```
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 limit the requester's access to profile data, returning only specific attributes of the profile that granted the access token. This is achieved using the OIDC-like scopes `email` and `profile`.

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 PingAuthorize Policy Editor, you can find this policy under **Global Decision Point**. This policy denies any request using an access token if the token's `active` flag is set to `false`. This policy is augmented with a set of scope-based access control policies.

Steps

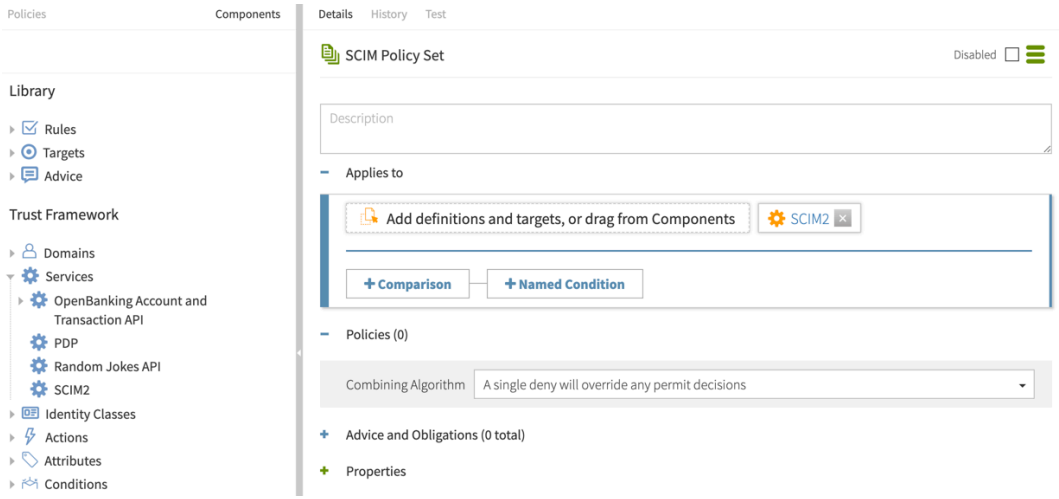
1. To create the tree structure, perform the following steps:
   a. Sign on to the PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
   b. Click **Policies**.
   c. Highlight **Global Decision Point**.
   d. From the **+** menu, select **Add Policy Set**.
   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 **Add definitions and targets, or drag from Components** box.

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

   Result:

   You should have a screen like the following.

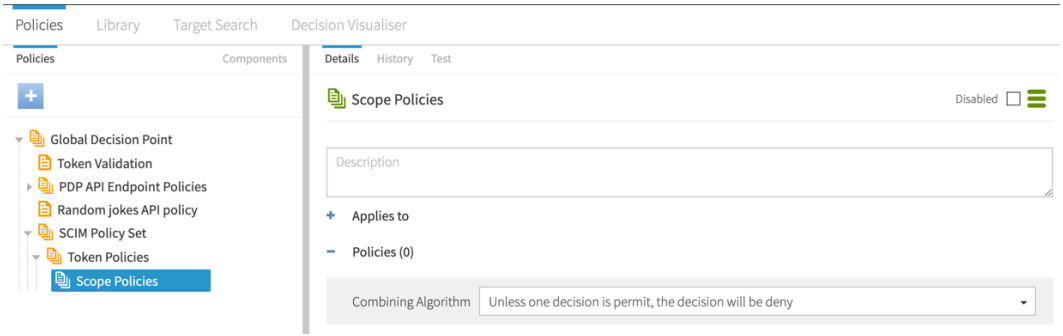


2. To add a branch under the SCIM policy set to hold SCIM-specific access token policies, go from **Components** to **Policies** and perform the following steps:
   a. Highlight **SCIM Policy Set**.
   b. From the **+** menu, select **Add Policy Set**.
   c. For the name, replace **Untitled** with `Token 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**.

   Result:

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



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 PingAuthorize 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 PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
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.
  Example:

```
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 PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
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 hamburger 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 **Add definitions and targets, or drag from Components** box.

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

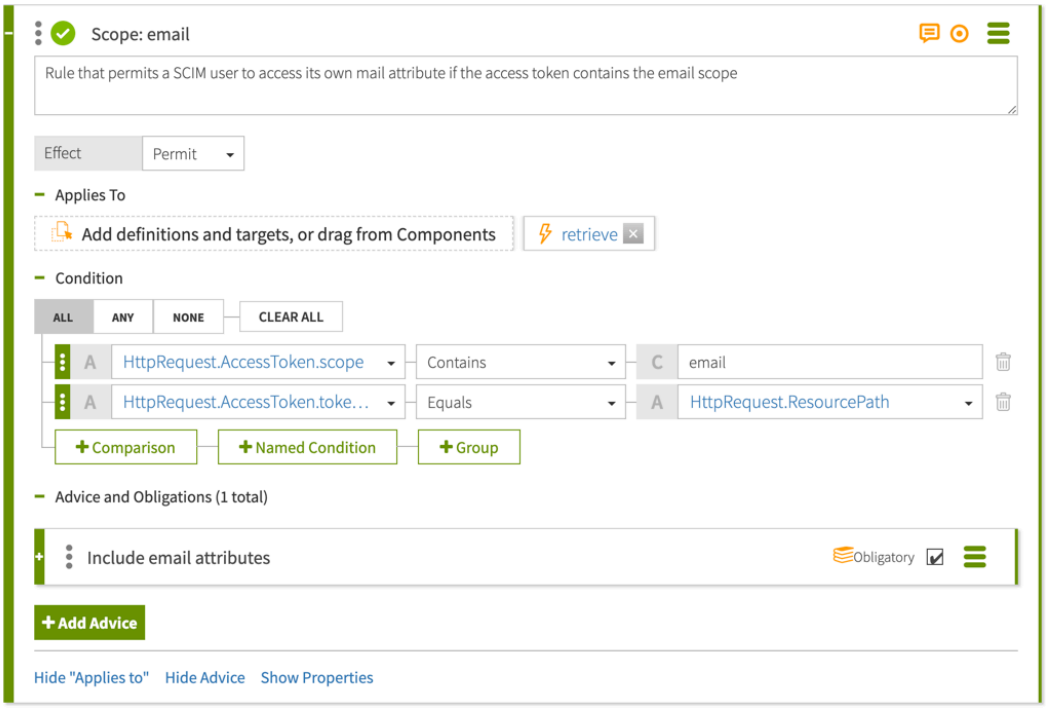
14. Within the rule, click **Show Advice and Obligations**.
15. Click **+** next to **Advice and Obligations**.
16. From the **Advice** section, drag **Include email attributes** to the **Advice and Obligations** section.

> **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**.

Result

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



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.
  Example:

```
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"]}
```

  Result: 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

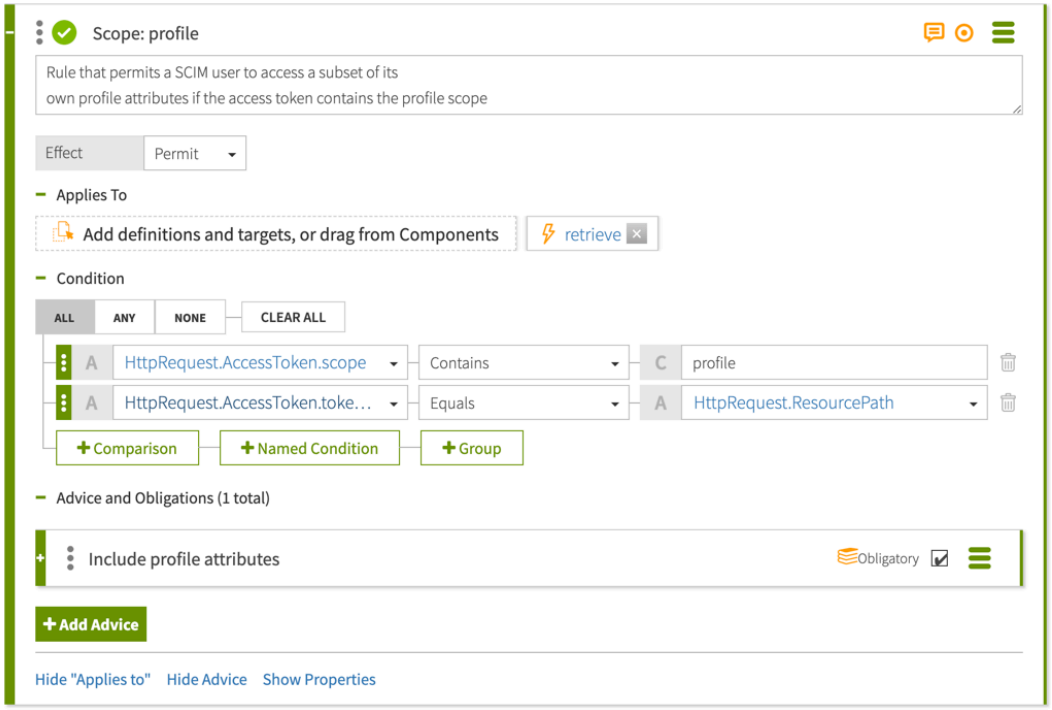
1. Sign on to the PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, **Token Policies**, and **Scope Policies**.
4. Highlight **Permitted Scopes**.
5. Click **Components**.
6. From the **Rules** list, drag **Permitted SCIM scope for user** to the **Rules** section.
7. To the right of the copied rule, click the hamburger 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 **Add definitions and targets, or drag from Components** 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**.

Result

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



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`.

  Example:

  ```
  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"]}
  ```

  Result: 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.

  Result:

  ```
  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

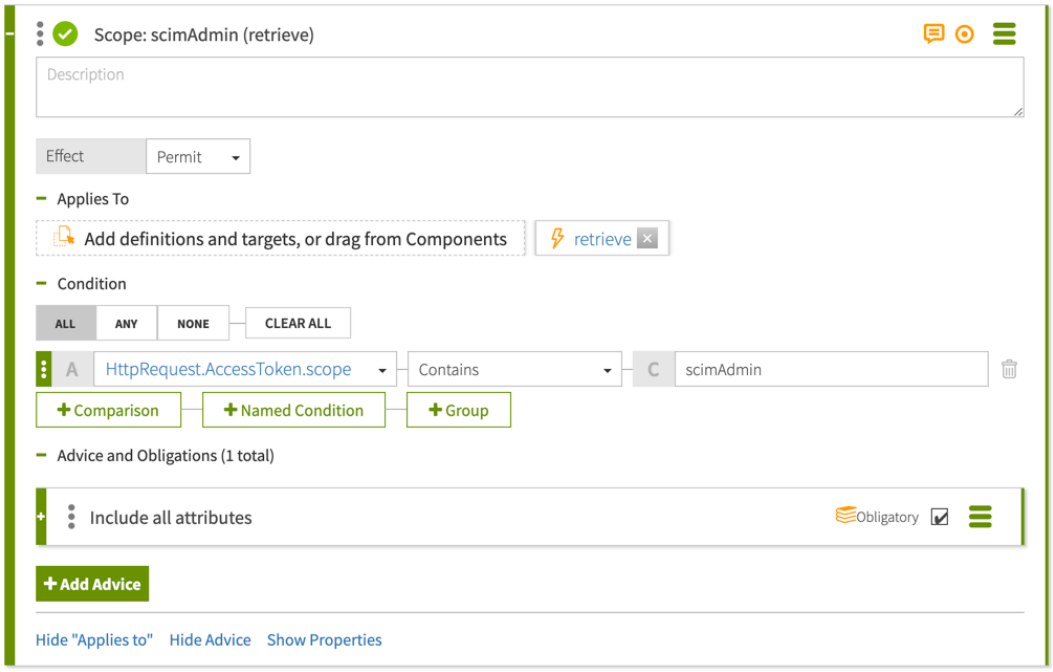
Add the scimAdmin retrieve rule to the Permitted Scopes policy.

Steps

1. Sign on to the PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
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 **Add definitions and targets, or drag from Components** 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**.

Result

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



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

Steps

1. Sign on to the PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
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 **Add definitions and targets, or drag from Components** box.
11. From the **Actions** sections, drag **modify** to the **Add definitions and targets, or drag from Components** 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 PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.

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 **Add definitions and targets, or drag from Components** 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 PingAuthorize Policy Editor using the URL and credentials from *Accessing the GUIs* on page 19.
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 **Add definitions and targets, or drag from Components** 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 an allow list of OAuth2 clients.

About this task

In the PingAuthorize Policy Editor, define a policy in which each rule specifies an allowed client.

Steps

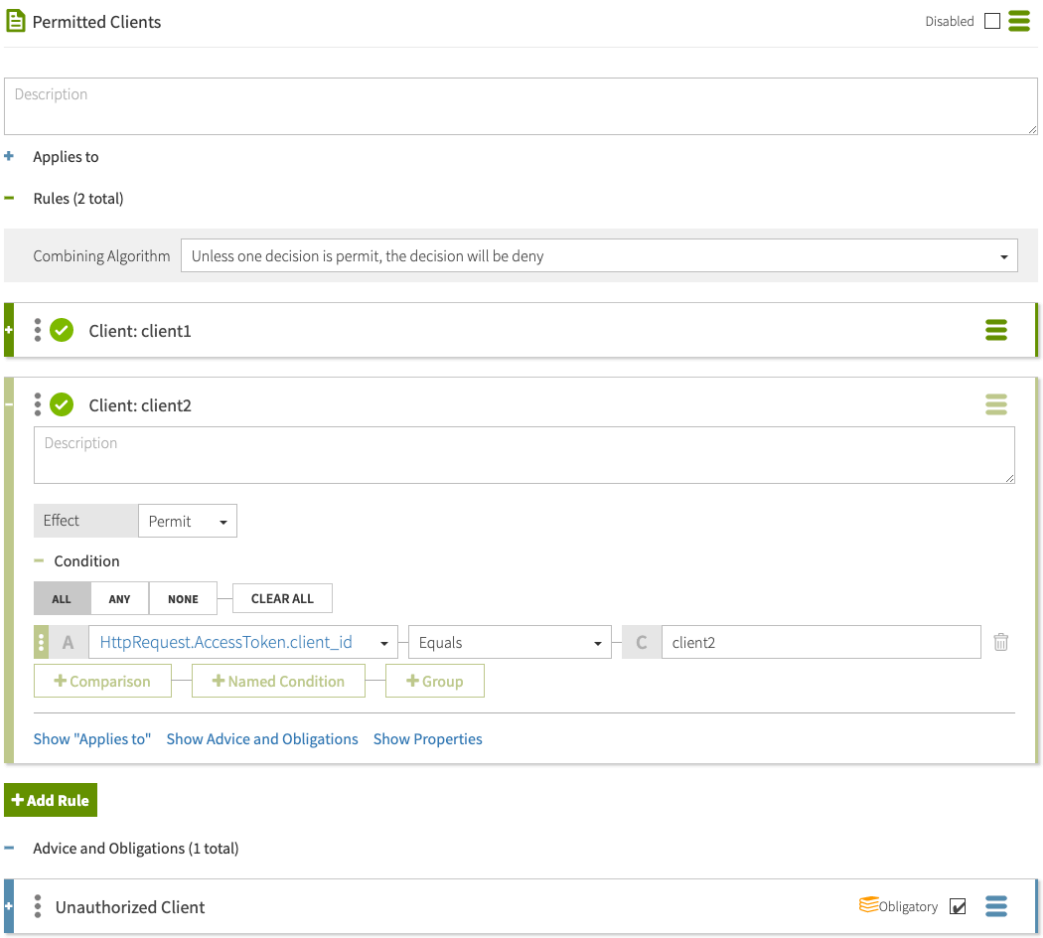
1. Go to **Policies** # **Policies**.
2. Expand **Global Decision Point** and **SCIM Policy Set**.

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, enter `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. 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, 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**.

Result
The completed configuration should resemble the following image.



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"}'
```

  Result: 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"}'
```

Result: 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.

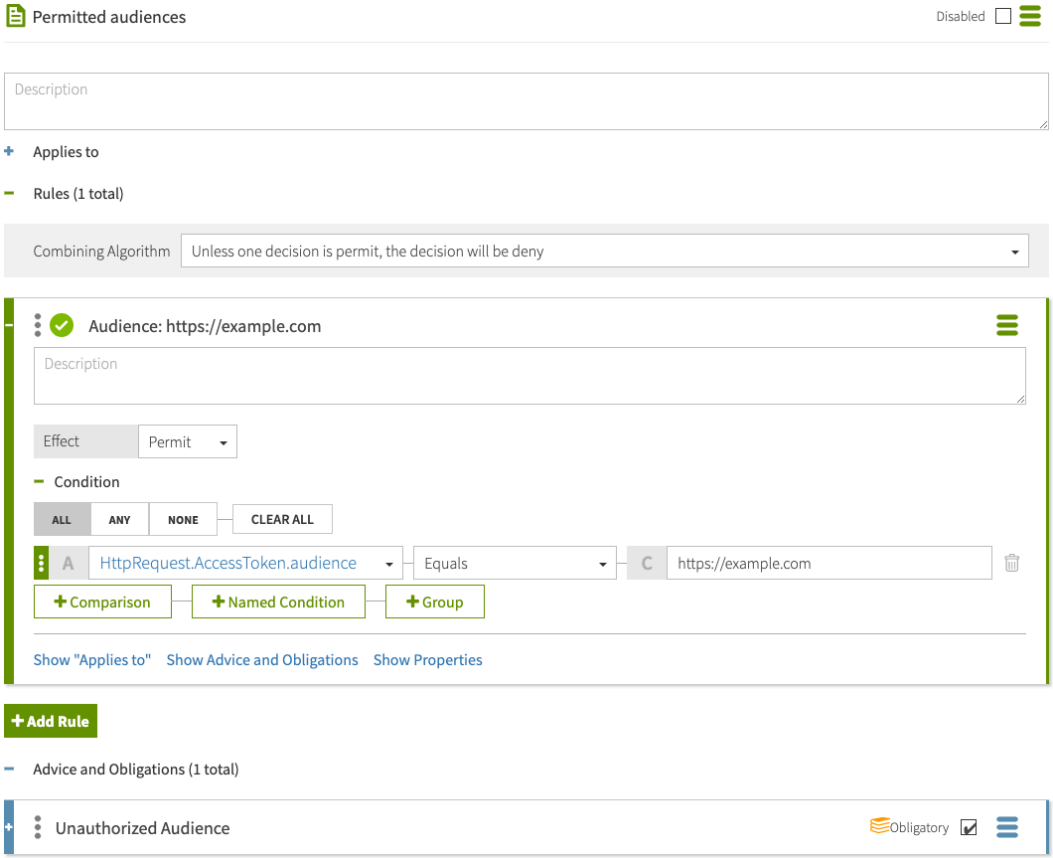
Steps

1. Go to **Policies** # **Policies**.
2. Expand **Global Decision Point** and **SCIM Policy Set**.
3. Highlight **Token Policies** and click **+** and then **Add Policy**.
4. For the name, replace **Untitled** with `Permitted Audiences`.
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 `Audience: https://example.com`.
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.audience`.
   c. From the middle, comparison-type list, select `Equals`.
   d. In the final field, enter `https://example.com`.
10. Expand **+ Advice and Obligations**.
11. 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.

12. Click **Save changes**.

Result
The final configuration should resemble the following image.



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"}'
```

   Result: 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"}'
```

   Result: 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 PingAuthorize 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

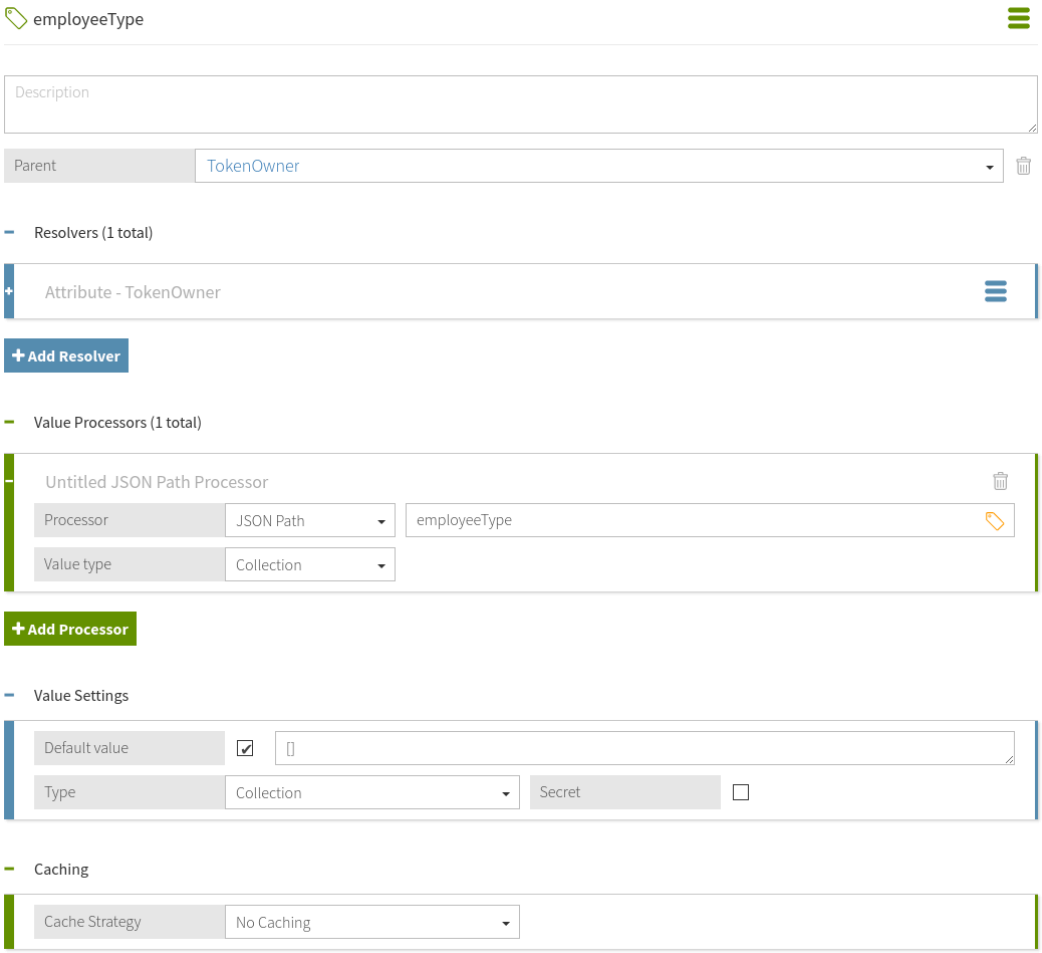
1. Go to **Trust Framework** and click the **Attributes** tab. Click **TokenOwner**.
2. Click **+** and then **Add new Attribute**.
3. For the name, replace **Untitled** with `employeeType`.
4. From the **Parent** list, select `TokenOwner`.
5. In the **Resolvers** section:
   a. Click **+ Add Resolver**.
   b. From the **Resolver type** list, select `Attribute` and in the **Select an Attribute** list, specify a value of `TokenOwner`.
6. Click **+** next to **Value Processors** and then **+ Add Processor**.
7. From the **Processor** list, select `JSON Path` and enter the value `employeeType`.
8. Set the **Value type** to `Collection`.
9. 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`.
10. Click **Save changes**.

Result

The final attribute configuration should resemble the following image.

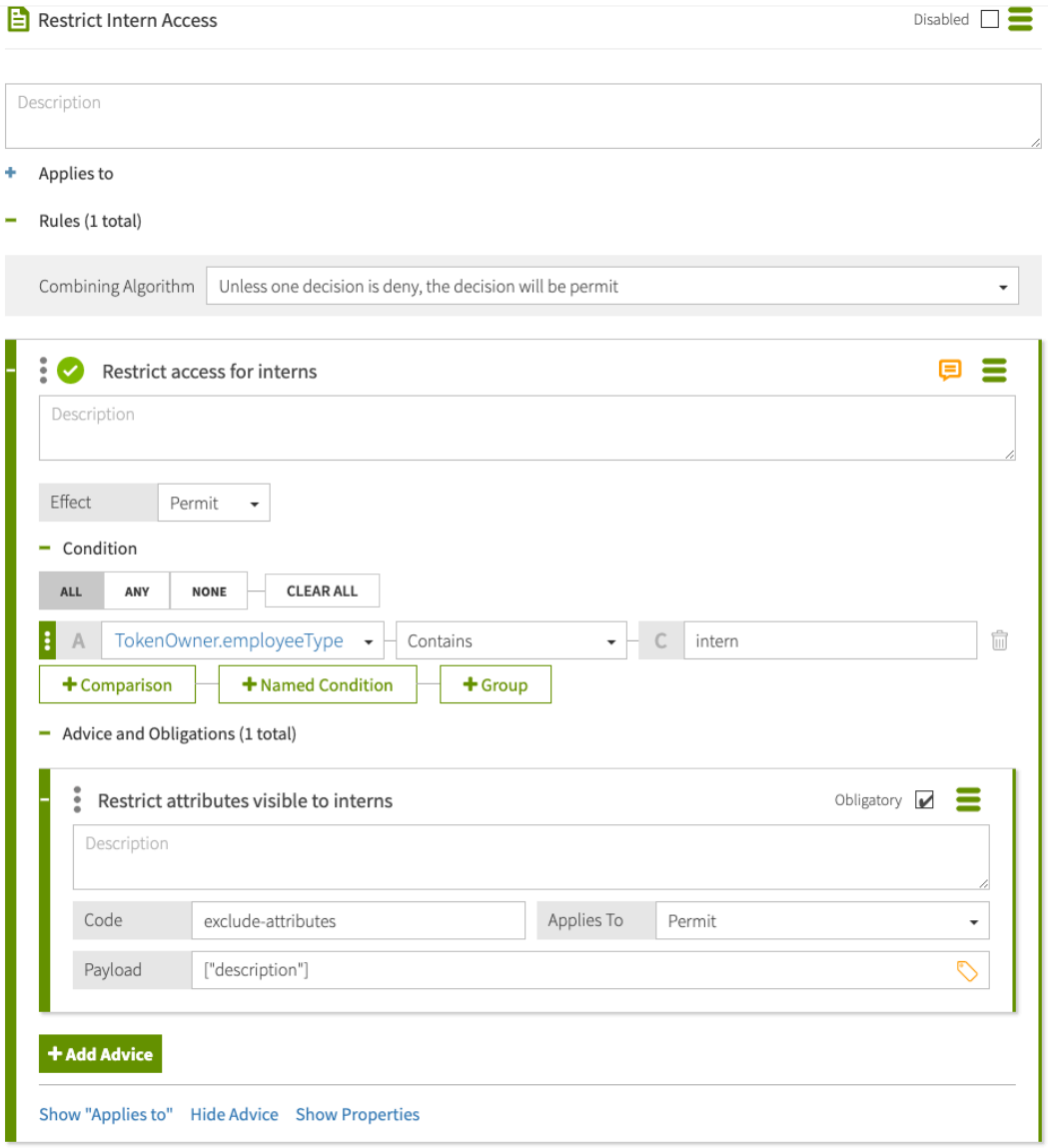


Next steps

Add a policy that uses the `employeeType` attribute.

1. Go to **Policies** # **Policies**.
2. Highlight **SCIM Policy Set** and click **+** and then **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** list, 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**.



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

About this task

The PingAuthorize 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 PingAuthorize Server file at `PingAuthorize/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 PingAuthorize to `include` before `exclude`.

Installing PingAuthorize

As you plan your PingAuthorize dynamic authorization software deployment, review the components to install as well as the potential deployment methods, architectures, and environments.

Seeing PingAuthorize in action

To quickly see PingAuthorize in action, see *Getting started with PingAuthorize (tutorials)* on page 17.

Components

Policy Editor

The PingAuthorize Policy Editor gives policy administrators the ability to develop and *test data-access policies*.

PingAuthorize Server

Enforces policies to control fine-grained access to data.

REST APIs access data through PingAuthorize Server, which applies the data-access policies to allow, block, filter, or modify data resources and *data attributes*.

Deployment methods

You can deploy PingAuthorize in either of the following ways.

Deployment method	Recommended for
Docker	Server administrators familiar with Docker who want to use orchestration to manage their environments. For more information, see *Docker deployment* on page 83.

Deployment method	Recommended for
Manual	Server administrators familiar with their operating systems who want to tweak and maintain their environments themselves. For more information, see *Manual installation* on page 88.

Implementation architectures

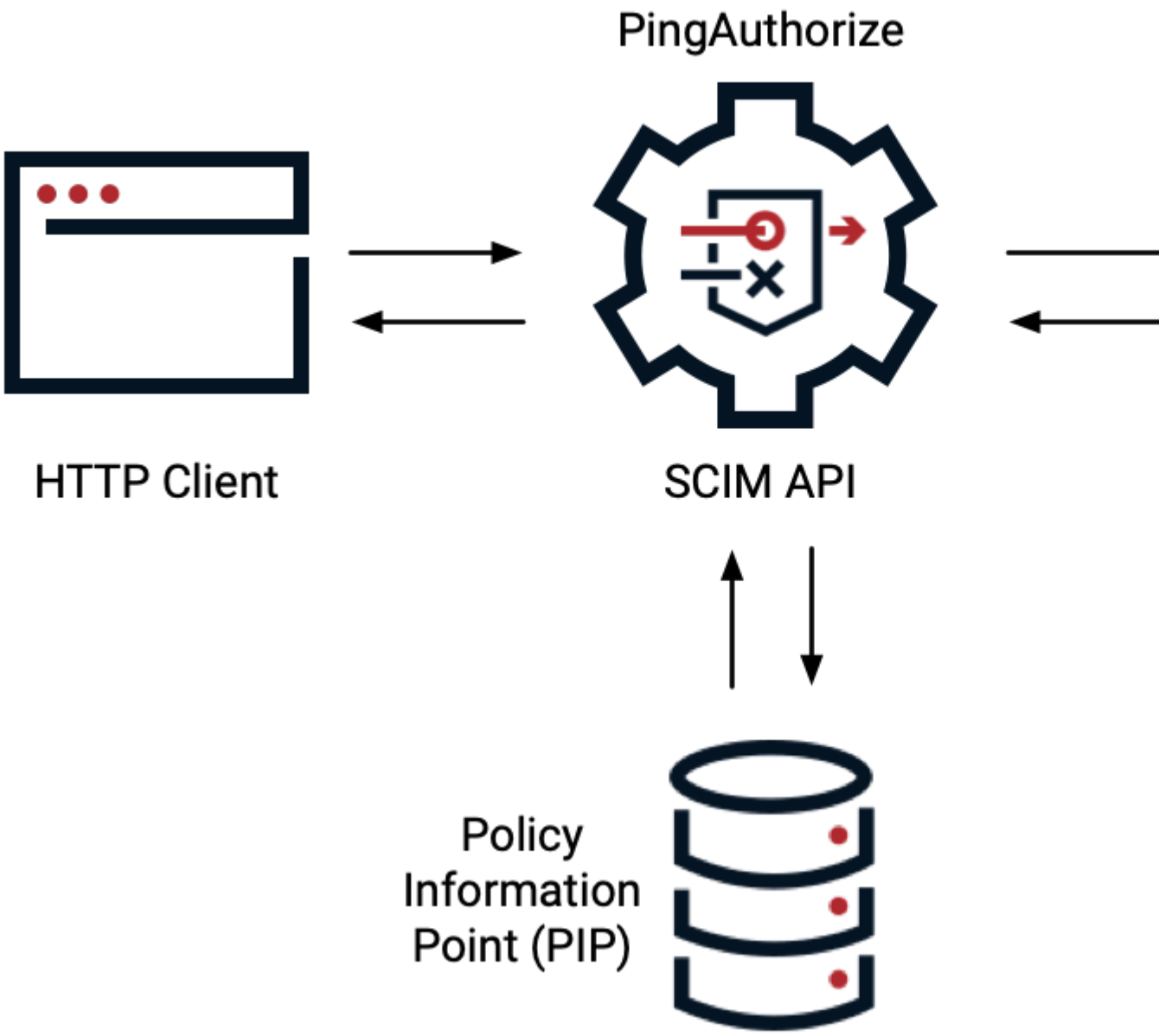
PingAuthorize Server supports the following implementation and data flow architectures for enforcing fine-grained access to data:

- *System for Cross-domain Identity Management (SCIM) API to datastores*
- *API security gateway as reverse proxy* on page 79
- *API security gateway in sideband configuration* on page 80
- *Policy Decision Point (PDP) APIs, for non-API use cases*

The following sections describe these architectures in more detail.

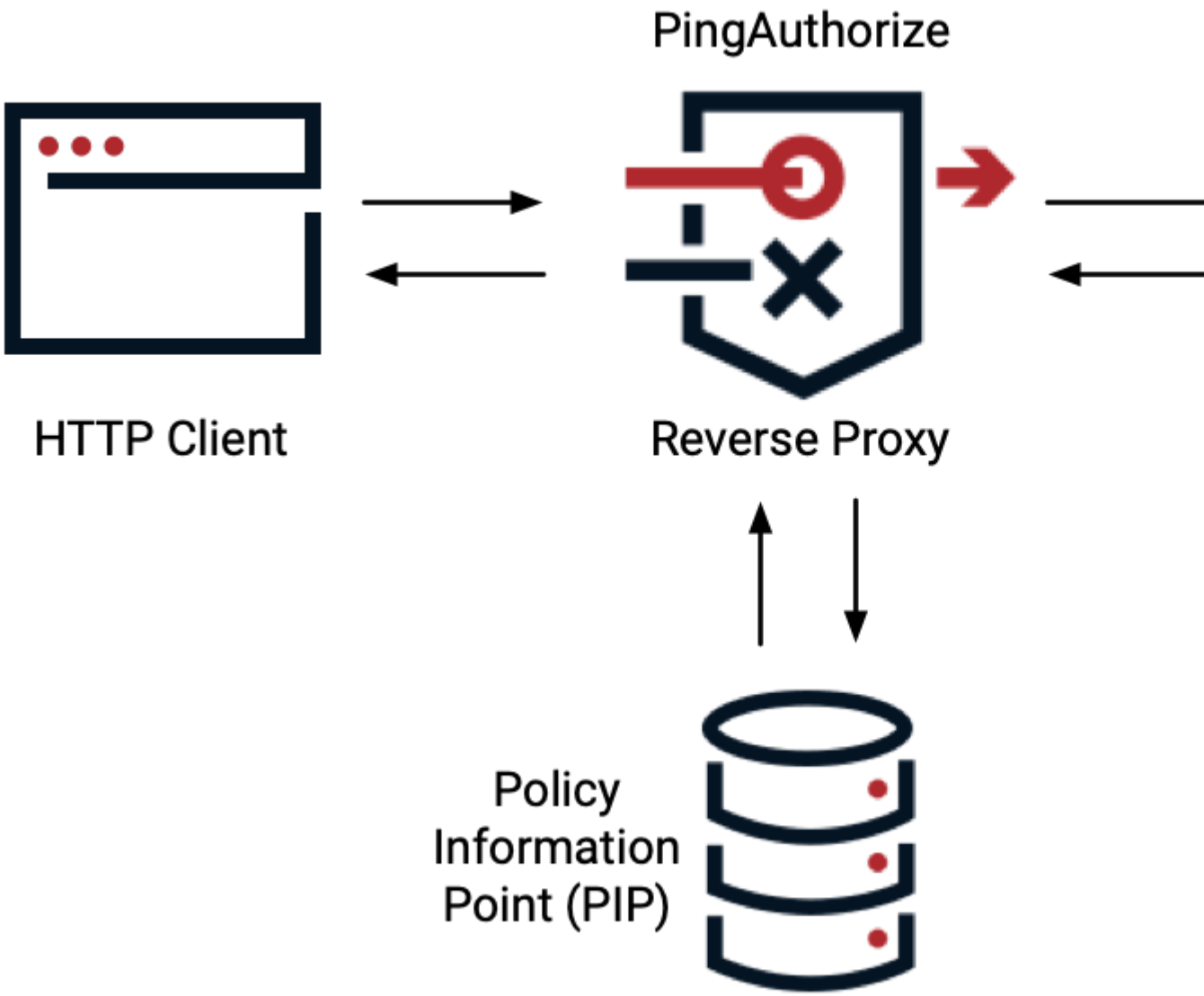
SCIM API to datastores

The PingAuthorize 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. See *SCIM API request and response flow* on page 192 for more information.



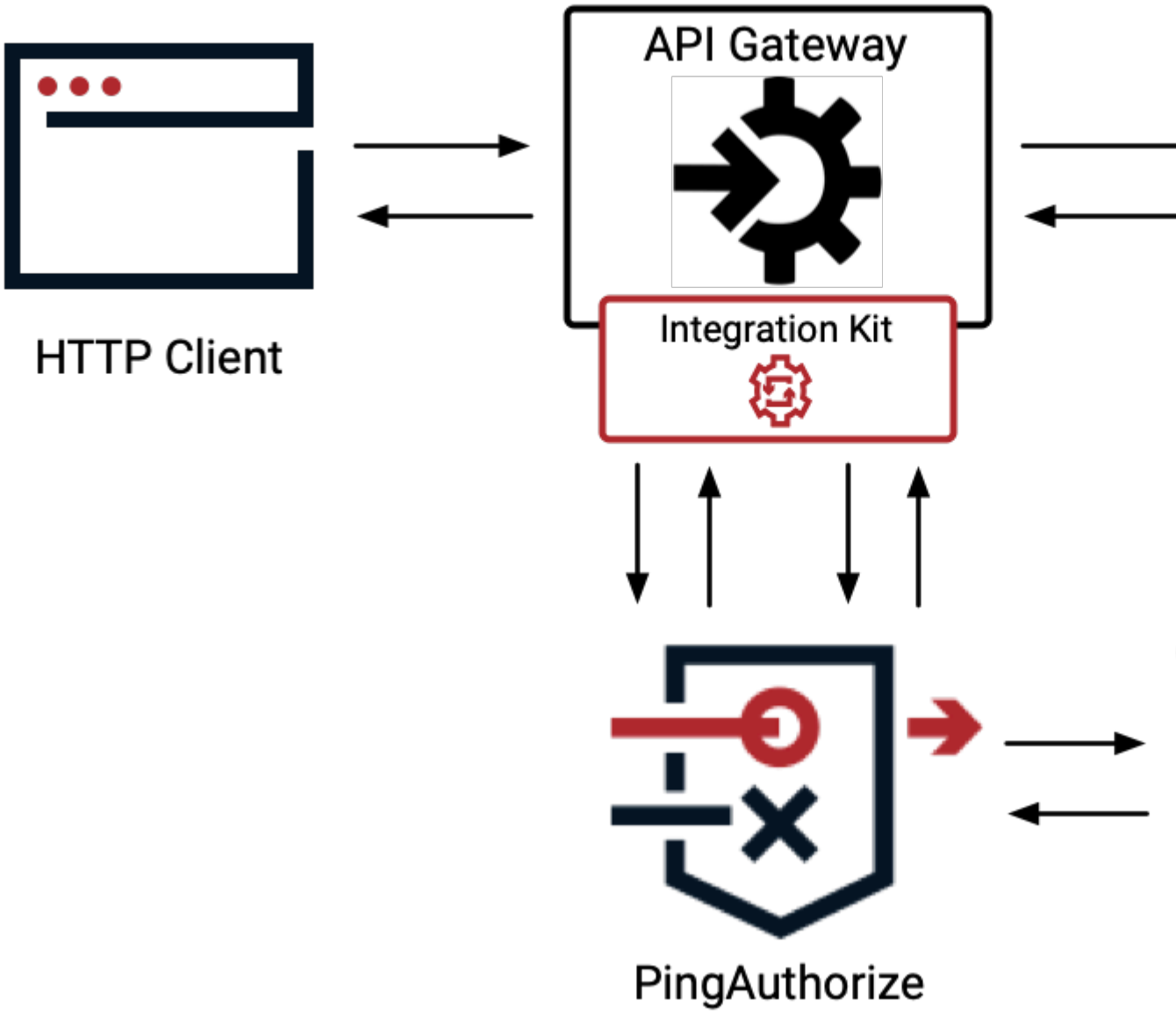
API security gateway as reverse proxy

You can configure PingAuthorize Server's API security gateway as a reverse proxy to an existing JSON-based REST API. In this architecture, PingAuthorize Server acts as an intermediary between clients and existing API services. The policy is enforced by the API security gateway. See *API gateway request and response flow* on page 166 for more information.



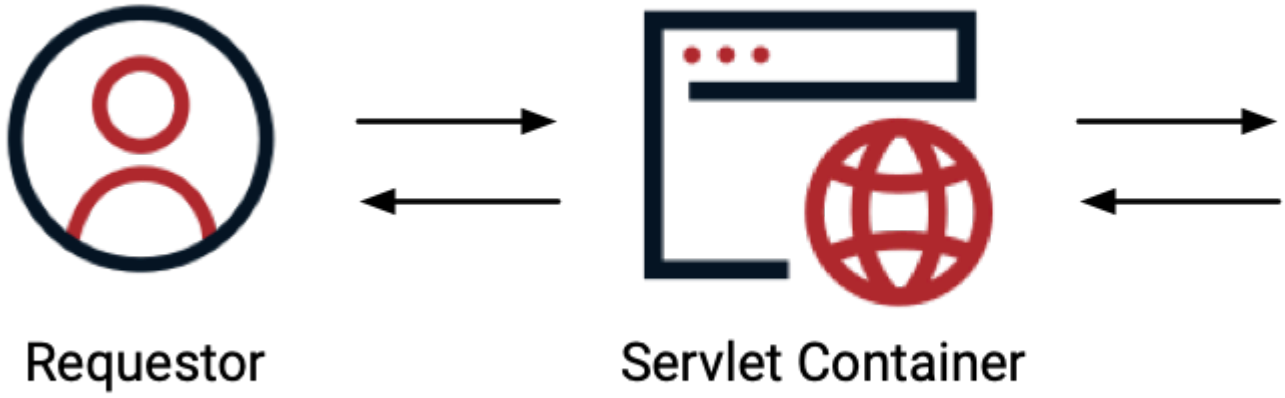
API security gateway in sideband configuration

You can configure PingAuthorize Server's API security gateway as an extension to an existing API lifecycle management gateway, which is commonly known as a sideband configuration. In this architecture, the API lifecycle management gateway functions as the intermediary between clients and existing API services. However, API request and response data still flows through PingAuthorize Server to enforce policy. See *API gateway integration* on page 179 for more information.



PDP APIs for non-API use cases

You can implement either of the PingAuthorize Server's PDP APIs to support policy decisions in cases where you don't need to protect an API resource. See *About the Authorization Policy Decision APIs* on page 222 for more information.



Policy decision environments

You can configure PingAuthorize Server for either of the following policy decision environments:

Development environment (external)

PingAuthorize Server and the Policy Editor are used together during the development of policies, with the PingAuthorize Server enforcing policy decisions, and the Policy Editor serving as the external PDP.

Other pre-production and production environments (embedded)

Policies are developed and deployed to the PingAuthorize Server, which both enforces policy decisions and serves as the PDP. This configuration supports policy testing in pre-production environments and live policy decisions in production.

The following sections describe these policy decision environments in more detail.

Development environment

To allow teams to test data-access policies during their development, PingAuthorize Server is configured to obtain policy decisions from the Policy Editor. To enable this configuration, set the Policy Decision Service **PDP Mode** to **external**.

Note:

The following image shows PingAuthorize Server configured in the Reverse Proxy architecture. The *development environment* supports all PingAuthorize implementation and data flow architectures.

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

Other pre-production and production environments

The Policy Editor is not a part of so-called "higher" environments. Rather, for these environments, the policy administrator *bundles policies into a deployment package* and then directly integrates them with the PDP. Embedding the policies in the PDP helps to reduce latency in the decision request-response flow. To enable this configuration, set the Policy Decision Service **PDP Mode** to **embedded**.

The Policy Editor can deploy policy deployment packages for integration with the PingAuthorize Server using one of the following methods:

- Exporting the deployment package as a static file
- Publishing via the *Deployment Manager* to a cloud package store (*AWS S3*, *Azure*)
- Publishing via the *Deployment Manager* to a *file system package store*

Note:

The following image shows PingAuthorize Server configured in the Reverse Proxy architecture. Pre-production and production environments support all PingAuthorize implementation and data flow architectures.

Docker deployment

Running PingAuthorize Docker containers standardizes your deployments and helps support devops principles.

For information about deployment methods and architectures, see *Installing PingAuthorize* on page 77.

Deployment requirements when using Docker

For a PingAuthorize software deployment using Docker devops, you need a supported version of Docker, the Docker images, and a compatible browser.

Docker

This following version of Docker is supported:

- Docker 20.10.9

Important:

Increase your Docker memory limit to at least 4 GB. To change this setting, go to **Docker Dashboard** # **Settings** # **Resources** # **Advanced**.

Containers

Docker images for Ping Identity's on-premise server products are available on *Ping Identity Docker Hub*. For information about Docker deployments, visit the *Ping Identity DevOps* documentation. To start deploying images, see *Get Started*.

The following Docker containers are available.

Container	Description	Image
pingdataconsole	administrative console Use the administrative console to configure PingAuthorize.	*DockerHub: PingDataConsole*
pingauthorize	PingAuthorize Server The server enforces the policies you define.	*DockerHub: PingAuthorize*
pingauthorizepap	PingAuthorize Policy Editor Use the Policy Editor to define the policies that determine access control and data protection.	*DockerHub: PingAuthorizePAP*
pingdirectory	PingDirectory A directory of user information. **Note:** PingAuthorize does not require PingDirectory.	*DockerHub: PingDirectory*

Note:

Only the PingDataConsole, PingAuthorize, PingAuthorize PAP, and PingDirectory software is licensed under Ping Identity's end user license agreement. Any other software components contained in the image are licensed solely under the terms of the applicable open source/third party license.

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.

Browsers

The PingAuthorize administrative console is compatible with several different web browsers, including:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Deploying PingAuthorize Server and Policy Editor using Docker

Instead of manual software installation, you can run Docker images of the PingAuthorize Server and Policy Editor.

About this task

To start the setup process after you obtain the Docker images:

Steps

1. Run the PingAuthorize Server container, `pingauthorize`.
2. Run the PingAuthorize Policy Editor container, `pingauthorizepap`.

3. Optional: To configure PingAuthorize with a GUI, run the PingAuthorize administrative console container, `pingdataconsole`.
4. Optional: If you need user-level control of the data, set up a user store.

   If you use PingDirectory, run the `pingdirectory` container.

Next steps

Perform additional configuration steps.

Deploying PingAuthorize Server using Docker
Perform a PingAuthorize Server deployment by running a Docker image.

About this task
The following command uses the `~/.pingidentity/config` environment file to configure common environment variables. See *https://devops.pingidentity.com/get-started/introduction*.

Steps

- Run the following command.

```
docker run --network=<network_name> \
   --env-file ~/.pingidentity/config \
   --name pingauthorize \
   --publish 1389:1389 \
   --publish 8443:1443 \
   --detach \
   --env SERVER_PROFILE_URL=https://github.com/pingidentity/pingidentity-server-profiles.git \
   --env SERVER_PROFILE_PATH=getting-started/pingauthorize \
   --tmpfs /run/secrets \
  pingidentity/pingauthorize:<TAG>
```

The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see *Docker Hub*.

> **Note:**
> - For proper communication between containers, create a Docker network using a command, such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.
> - You can use server profiles to automate deployment of PingAuthorize Server. For more information, see *Deployment automation and server profiles* on page 361.

Signing on to the administrative console (Docker deployment)
After you deploy the server by running the Docker image, access the administrative console to verify the server configuration and manage the server settings.

About this task

When using Docker containers, the containers must be on the same Docker network to communicate properly.

Steps

1. Start the PingDataConsole.

   The following command uses the `~/.pingidentity/config` environment file to configure common environment variables. See *https://devops.pingidentity.com/get-started/introduction*.

```
docker run \
   --env-file ~/.pingidentity/config \
```

```
    --name pingdataconsole \
    --detach \
    --publish 5443:8443 \
    --tmpfs /run/secrets \
    pingidentity/pingdataconsole:<TAG>
```

The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see Docker Hub (*https://hub.docker.com/r/pingidentity/pingdataconsole*).

2. Sign on using the information in the following table.

Description	Details
URL	`https://localhost:${HTTPS_PORT}/console/login`
Details to enter at login	Server: pingauthorize:1636 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 19.

Deploying PingAuthorize Policy Editor using Docker

Deploy PingAuthorize Policy Editor by running its Docker image. Using Docker devops enables the automated policy database update feature with mounted volumes.

About this task

When running the Ping Identity DevOps `pingauthorizepap` Docker container, you can use the following commands to ensure that the policy database is on the mounted volume in preparation for future versions of the image. The commands:

- Run a `pingauthorizepap` Docker container named `pap` on host port 8443.
- Use the `~/.pingidentity/config` environment file to configure common environment variables. See *https://devops.pingidentity.com/get-started/introduction*.
- Bind mount a customized `options.yml` file named `custom-options.yml` to the server root using the server profile capability. The host system `server-profile` folder must contain `instance/custom-options.yml` for this example to work correctly. See *https://devops.pingidentity.com/reference/config/*.
- Set the **`Ping_Options_File`** environment variable to tell **`setup`** to use `custom-options.yml`.

For an H2 database, the command:

- Bind-mounts a volume that maps a policy database to `/opt/out/Symphonic.mv.db`.
- Sets the **`PING_H2_FILE`** environment variable to tell **`setup`** to use `/opt/out/Symphonic.mv.db` for the policy database. The environment variable must exclude the `.mv.db` extension.

For a PostgreSQL database, the command sets environment variables to provide setup with username, password, host, and port database credentials.

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/*.

Steps

- Run the `pingauthorizepap` Docker container.
  Choose from:
  - If you are using an H2 database, run the following command.

```
$ docker run --network=<network_name> --name pap -p 8443:1443 \
  --env-file ~/.pingidentity/config \
  --volume /home/developer/pap/server-profile:/opt/in/ \
  --env PING_OPTIONS_FILE=custom-options.yml \
  --volume /home/developer/pap/Symphonic.mv.db:/opt/out/
Symphonic.mv.db \
  --env PING_H2_FILE=/opt/out/Symphonic \
  pingidentity/pingauthorizepap:<TAG>
```

> **Note:**
>
> For proper communication between containers, create a Docker network using a command such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

  - If you are using a PostgreSQL database, run the following command.

```
$ docker run --network=<network_name> --name pap -p 8443:1443 \
  --env-file ~/.pingidentity/config \
  --volume /home/developer/pap/server-profile:/opt/in/ \
  --env PING_OPTIONS_FILE=custom-options.yml \
  --env PING_DB_APP_USERNAME="<username>" \
  --env PING_DB_APP_PASSWORD="<password>" \
  --env
 PING_DB_CONNECTION_STRING="jdbc:postgresql://<host>:<port>/<database>"
 \
  pingidentity/pingauthorizepap:<TAG>
```

> **Note:**
>
> The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see *Docker Hub*.

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

> **Note:**
>
> The containers must be on the same Docker network to communicate properly.

Sign on to the Policy Editor. For more information, see *Signing on to the PingAuthorize Policy Editor* on page 108 and import a policy snapshot. You can find a set of default policies in the `resource/policies/defaultPolicies.SNAPSHOT` file.

To configure PingAuthorize Server to use the Policy Editor, use **`dsconfig`** or the administrative console to create a Policy External Server to represent the Policy Editor, 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. Assume a container named pingauthorize and that no files are needed from the file system. The following commands run **`dsconfig`** from within the container.

```
docker exec pingauthorize /opt/out/instance/bin/dsconfig create-external-server \
  --server-name "Policy Editor" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:2FederateM0re" \
  --set "branch:Default Policies"

docker exec pingauthorize /opt/out/instance/bin/dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:Policy Editor" \
  --set trust-framework-version:v2
```

In the example, the base URL consists of the host name and port chosen for the Policy Editor during setup. The shared secret value is `2FederateM0re` by default. The branch name corresponds to the branch name that you chose when importing your policy snapshot.

Next steps
After installed, complete some configuration steps and then start developing policies to enforce fine-grained access to data.

Consider performing the following next steps.

- Configure access token validation.

  For more information, see *Configure access token validation* on page 355.
- Configure a user store.

  For more information, see *User store configuration* on page 354
- Sign on to the administrative console to configure endpoints for existing JSON APIs.

  For more information, see *About the API security gateway* on page 166.
- Sign on to the administrative console to define SCIM APIs for data in databases

  For more information, see *About the SCIM service* on page 192.
- Sign on to the PingAuthorize Policy Editor to create policies.

  For more information, see the PingAuthorize Policy Administration Guide.

Manual installation

Instead of running Docker images, you can deploy the PingAuthorize software in a manual install mode using `.zip` files.

For information about deployment methods and architectures, see *Installing PingAuthorize* on page 77.

Before you install manually

You must prepare your computing environment by installing certain system requirements before a manual PingAuthorize software installation.

The following components are required to install PingAuthorize:

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

The following sections describe these prerequisites in more detail.

System requirements
Ensure that your computing environment meets the system requirements for the PingAuthorize dynamic authorization management software.

Ping Identity has qualified the configurations in this section and has certified that they are compatible with the product. PingAuthorize 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 PingAuthorize on a variety of different platforms and operating systems, including:

- Amazon Linux 2
- Canonical Ubuntu 18.04 LTS and 20.04 LTS
- CentOS Linux 7.7 and 8.1
- Microsoft Windows Server 2016 and 2019 (Policy Editor not supported)
- Oracle Linux 7.9, 8.2, and 8.4
- Red Hat Enterprise Linux ES 7.9, 8.1, 8.2, and 8.4
- SUSE Linux Enterprise 12 SP5 and 15 SP1

Note:

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

Java Runtime Environment

Make sure your Java Runtime Environment (JRE) meets the system requirements for PingAuthorize:

- Amazon Corretto 8
- OpenJDK 8 and 11, obtained from *AdoptOpenJDK*
- Oracle Java SE Development Kit 8 and 11 LTS

Note:

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

Browsers

The PingAuthorize administrative console is compatible with several different web browsers, including:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Databases

The Policy Editor persists its policies, trust framework, and versioning data in a policy database. By default, this is an embedded H2 file-based database. Optionally, you can configure the Policy Editor to use a PostgreSQL database.

For more information, see *Setting up a PostgreSQL database* on page 99.

Supported databases:

- H2
- PostgreSQL 11.2 and 12.1

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 PingAuthorize 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 PingAuthorize Server root directory does not work in this case.

For information about how to specify the license with the Policy Editor, see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.

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 PingAuthorize Server.

However, PingAuthorize Policy Editor fails to start if the license has expired.

Creating a Java installation dedicated to PingAuthorize

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

About this task

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

For Linux computing environments, complete the required tasks described in this section before initiating a PingAuthorize Server installation.

About this task

Complete the following tasks before you install PingAuthorize 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

PingAuthorize 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, PingAuthorize 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 PingAuthorize, 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 PingAuthorize 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.

Example: 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
```

Obtaining the installation packages

To begin the software installation process for PingAuthorize, obtain the server component's .zip installation packages.

About this task

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

- PingAuthorize Server
- PingAuthorize Policy Editor

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 and the Policy Editor manually

Use manual install mode for the PingAuthorize Policy Editor and PingAuthorize Server installations.

About this task

After you obtain the installation files, start the setup process.

Steps

1. Install PingAuthorize Server.
2. Install PingAuthorize Policy Editor.
3. Perform additional configuration steps.

   The following sections describe these installation and configuration steps.

Installing the server manually

Choose your manual install mode for PingAuthorize Server and then perform the server installation.

Steps

1. Read about the server installation modes and decide which mode you want to use.
2. Complete the steps for your chosen mode, interactive or noninteractive.

About the server installation modes

There are several different installation modes for PingAuthorize Server.

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

- The **`setup`** tool performs the initial tasks needed to start PingAuthorize Server, including configuring Java virtual machine (JVM) runtime settings and assigning listener ports for the PingAuthorize Server's HTTP services.
- The ***`create-initial-config`*** tool configures connectivity between a System for Cross-domain Identity Management (SCIM) 2 user store and PingAuthorize Server. During the process, the **`prepare-external-store`** tool prepares each PingDirectory Server to serve as a user store by PingAuthorize 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 *User profile availability in policies* on page 288.

- After the initial setup is finished, you can use the ***`dsconfig`*** *tool* and the administrative console to perform additional configuration.

> **Tip:**
> You can use server profiles to automate deployment of PingAuthorize Server. For more information, see *Deployment automation and server profiles* on page 361.

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 PingAuthorize Server will run.

Installing the server interactively

Run the **`setup`** tool, which prompts you interactively for the information that it needs to install PingAuthorize 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 PingAuthorize Server to accept HTTPS requests
- An available LDAPS port for PingAuthorize 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.

   Example:

   ```
   $ ./setup
   ```

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

   For additional options, see *Starting PingAuthorize Server* on page 159.

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 PingAuthorize Server to accept HTTPS requests
- An available LDAPS port for PingAuthorize 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 PingAuthorize 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 `pingauthorize1`

- 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 pingauthorize1 \
  --location Austin
```

Signing on to the administrative console (manual installation)
After a manual software installation, access the administrative console to verify the server configuration and manage the server settings.

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 the PingAuthorize Policy Editor manually
Choose the database for your fine-grained access control use case, resources, and computing environment and install the PingAuthorize Policy Editor.

About this task

You can install the PingAuthorize Policy Editor in one of two ways: interactively or noninteractively.

Steps

1. Choose the database to use:
   Choose from:
   - H2: The default embedded database.
   - PostgreSQL: This is optional and requires additional configuration.
2. Optional: If you are using a PostgreSQL database, set up the database.

   For more information, see *Setting up a PostgreSQL database* on page 99.
3. Install the PingAuthorize Policy Editor:
   Choose from:
   - Interactive: The **`setup`** tool prompts you for information during the installation process.

     For more information, see *Installing the PingAuthorize Policy Editor interactively* on page 99.
   - Noninteractive: Automated installation allows more control over configuration. If you are using a PostgreSQL database, you must use this mode.

     For more information, see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.

Setting up a PostgreSQL database

To set up a PostgreSQL database for your attribute-based access control policies, create the database and a user role and add tables and privileges.

About this task

If you're using a PostgreSQL database instead of the default H2 database, you must set up the new database before you install the Policy Editor. If you're using the default H2 database, you don't need to complete this setup.

Steps

1. Create the database.

   Example:

   In this example, the command creates a database named `pap` using the `postgres` super user.

   ```
   [postgres] createdb pap
   ```

2. Create a user role for the application to use.

   Example:

   In this example, the command creates a user named `pap_user`.

   ```
   [postgres] createuser --pwprompt pap_user
   ```

3. Add tables and grant privileges to the application user.

   PingAuthorize provides DDL scripts to create the necessary schema. For the scripts and more details, go to *https://github.com/pingidentity/pingauthorize-contrib/tree/main/sql/postgresql*.

Next steps

Configure the Policy Editor to use the PostgreSQL database:

- To configure a Docker container, see *Installing PingAuthorize Policy Editor using Docker*.
- To configure a manual installation, see *Installing the PingAuthorize Policy Editor noninteractively*.

Installing the PingAuthorize Policy Editor interactively

You can run the PingAuthorize Policy Editor **`setup`** command interactively in CLI install mode.

Before you begin

You must have the following information:

- The location of a valid license file
- An available port for the PingAuthorize Policy Editor to accept HTTPS requests

About this task

The **`setup`** tool prompts you interactively for the information that it needs.

> **Note:**
>
> You cannot configure some setup options when installing the PingAuthorize Policy Editor interactively, such as PostgreSQL database configuration. For more information, see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.

Steps

1. Choose the authentication mode for the PingAuthorize Policy Editor:
   Choose from:
   - Demo mode: Configures the PingAuthorize Policy Editor to use form-based authentication with a fixed set of credentials. Unlike OpenID Connect (OIDC) mode, this mode doesn't require an external authentication server. However, it is inherently insecure and should only be used for demonstration purposes.
   - OIDC mode: Configures the PingAuthorize Policy Editor to delegate authentication and sign-on services to a PingFederate OIDC provider.

     In OIDC mode, you must 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

   Note:

   To use PingAuthorize Policy Editor with other OIDC providers, such as PingOne, see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.
2. Run the **`setup`** command.

   Note:

   If you don't want to use the default database credentials, see *Setting database credentials at initial setup* on page 251.
3. Copy and record any generated values needed to configure external servers.

   The Shared Secret is used in PingAuthorize, under **External Servers** # **Policy External Server** # **Shared Secret**.
4. To start the Policy Editor, or policy administration point (PAP), run `bin/start-server`.

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

Next steps

1. Complete the steps in *Post-setup steps (manual installation)* on page 107.
2. Consider additional configuration options in *Specifying custom configuration with an options file* on page 238.

Example: Installing and configuring the PingAuthorize Policy Editor
This tutorial describes how to install an instance of the PingAuthorize Policy Editor.

About this task

Note:

These installation instructions are for tutorial purposes. They will only provide a limited install.

Steps

1. Extract the contents of the compressed PingAuthorize-PAP distribution file.

2. Change the directory to `PingAuthorize-PAP`.
3. To configure the application, run the **`./bin/setup`** script.
4. Answer the on-screen questions.

   For the following questions, use the recommended answers provided.

Question	Answer
How would you like to configure the Policy Editor?	Use `Quickstart` to set up a demo server with credentials `admin`/`password123` and to use a self-signed certificate for SSL
On which port should the Policy Editor listen for HTTPS communications?	You can use any unused port here, but most of the examples in this guide assume that port 9443 is used for the PingAuthorize Policy Editor.
Enter the fully qualified host name or IP address that users' browsers will use to connect to this GUI?	Unless you are testing on `localhost`, ensure that the provided API URL uses the public DNS name of the PingAuthorize Policy Editor server as shown in the following example. `pap.example.com`

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

   The Shared Secret is used in PingAuthorize, under **External Servers** # **Policy External Server** # **Shared Secret**.
6. To start the Policy Editor, or policy administration point (PAP), run `bin/start-server`.

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

Result
Your demo configuration should resemble the following example.

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

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


PingAuthorize Policy Editor
===========================================

How would you like to configure the Policy Editor?

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

Enter option [1]: 1

On which port should the Policy Editor listen for application HTTPS communications? [9443]: 9443

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

On which port should the Policy Editor listen for administrative HTTPS communications? [9444]:
 9444

Would you like to enable periodic policy database backups? (yes / no) [yes]: yes

Enter the backup schedule as a cron expression (defaults to daily at midnight): [0 0 0 * * ?]: 0
 0 0 * * ?

Setup Summary
```

```
=========================================
Host Name:          pap.example.com
Server Port:        9443
Secure Access:      Self-signed certificate
Admin Port:         9444
Periodic Backups:   Enabled
Backup Schedule:    0 0 0 * * ?

Command-line arguments that would set up this server non-interactively:
    setup demo --hostname pap.example.com --adminPort 9444 --port 9443 --certNickname server-
cert \
         --licenseKeyFile /opt/PingAuthorize/PingAuthorize.lic \
         --backupSchedule '0 0 0 * * ?' --pkcs12KeyStorePath config/keystore.p12 \
         --generateSelfSignedCertificate

What would you like to do?

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

Enter option [1]:

Setup completed successfully

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

Please start the server by running bin/start-server
```

In this example, the PingAuthorize Policy Editor is now running and listening on port 9443.

Next steps

To sign on to the interface, go to `https://<host>:9443`. The default credentials are `admin` and `password123`.

Note:

Use the default user name and password sign on credentials for demo and testing purposes only, such as this initial walk-through. To configure the PingAuthorize Policy Editor for PingFederate OpenID Connect (OIDC) single sign-on (SSO), see *Installing the PingAuthorize Policy Editor noninteractively* on page 102.

Installing the PingAuthorize Policy Editor noninteractively

For an automated software installation, run PingAuthorize Policy Editor **`setup`** in the noninteractive CLI install 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 Editor 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 Editor's logging behavior.
- Configure the Policy Editor for a PostgreSQL database.
- Configure the Policy Editor to present an existing SSL certificate instead of generating a self-signed certificate.

For more information, see *Specifying custom configuration with an options file* on page 238.

Steps

1. Optional: If you choose to use a PostgreSQL policy database, you must *set up the database* before you install the Policy Editor.

   After you set up your PostgreSQL policy database, be prepared to provide the following information when installing the Policy Editor:

   - PostgreSQL Java Database Connectivity (JDBC) connection string, with the host, port, and database name
   - The PostgreSQL user and password for the application to use when accessing the database

2. Choose the authentication mode for the PingAuthorize Policy Editor:
   Choose from:

   - Demo mode: Configures the PingAuthorize Policy Editor to use form-based authentication with a fixed set of credentials. Unlike OpenID Connect (OIDC) mode, this mode doesn't require an external authentication server. However, it's inherently insecure and should only be used for demonstration purposes.
   - OIDC mode: Configures the PingAuthorize Policy Editor to delegate authentication and sign-on services to an OIDC provider, such as PingFederate.

     If you choose OIDC mode, you must provide the host name and port of an OIDC provider or its base URL.

   Note:

   If you don't use the **`setup`** tool to generate a self-signed certificate, you must also provide information related to the PingAuthorize Policy Editor's connection security, including the location of a keystore that contains the server certificate and the nickname of that server certificate.

   If the OIDC provider presents a certificate that is not trusted by the Policy Editor's JRE, do one of the following:

   - Add the certificate to the JRE trust store. For details, see *Configuring PingFederate as an OIDC provider for PingAuthorize* on page 114.
   - Disable certificate validation by starting the Policy Editor with the PING_OIDC_TLS_VALIDATION=NONE environment variable. See the tabs below for examples.

Tip:

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

3. Run the correct command based on your needs (see the tabs below for examples of the **`setup`** command in different authentication modes):

   > **Note:**
   >
   > If you don't want to use the default database credentials for your H2 policy database, see *Setting database credentials at initial setup* on page 251.

   Choose from:

   - 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
     ```

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

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

5. To start the Policy Editor, or policy administration point (PAP), run `bin/start-server`.

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

Next steps

Click the following tabs for examples of the **`setup`** command in different authentication modes.

1. After you complete setup, see *Post-setup steps (manual installation)* on page 107.
2. Consider additional configuration options in *Specifying custom configuration with an options file* on page 238.

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

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

The Policy Editor 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 Editor and must be configured in PingFederate.

The Policy Editor can skip hostname verification and accept self-signed SSL certificates from the OIDC provider.

This example uses the PING_OIDC_TLS_VALIDATION environment variable to set up the Policy Editor to handle sign-ons for a provider using a self-signed certificate.

```
$ env PING_OIDC_TLS_VALIDATION=NONE bin/setup oidc \
  --oidcHostname <ping-federate-hostname> \
  --oidcPort <ping-federate-port> \
  --clientId pingauthorizepolicyeditor \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

For more information about configuring PingFederate, see *Configuring an OIDC provider for single sign-on requests from PingAuthorize* on page 110.

Example: Set up the PingAuthorize Policy Editor in OIDC mode (generic OIDC provider)
This example sets up the PingAuthorize Policy Editor for sign-ons using an arbitrary OpenID Connect (OIDC) provider.

This example departs from the PingFederate 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 pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

The Policy Editor 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 Editor and must be configured in the OIDC provider as well.

The Policy Editor can skip hostname verification and accept self-signed SSL certificates from the OIDC provider.

This example uses the PING_OIDC_TLS_VALIDATION environment variable to set up the Policy Editor to handle sign-ons for a provider using a self-signed certificate.

```
$ env PING_OIDC_TLS_VALIDATION=NONE bin/setup oidc \
  --oidcBaseUrl https://auth.example.com/9595f417-a117-3f24-a255-5736ab01f543/auth/ \
  --clientId 7cb9f2c9-c366-57e0-9560-db2132b2d813 \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

For more information about configuring an OIDC provider, see *Configuring an OIDC provider for single sign-on requests from PingAuthorize* on page 110.

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

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

```
$ bin/setup demo \
```

```
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

The decision point shared secret is a credential that the PingAuthorize Server uses to authenticate to the Policy Editor when it uses the Policy Editor as an external policy decision point (PDP).

For information about how to configure PingAuthorize Server to use the decision point shared secret, see *Post-setup steps (manual installation)* on page 107.

Example: Set up the PingAuthorize Policy Editor with a PostgreSQL policy database
This example sets up the PingAuthorize Policy Editor in demo mode with an automatically generated self-signed server certificate and a PostgreSQL policy database.

```
$ bin/setup demo \
  --adminUsername admin \
  --dbConnectionString "jdbc:postgresql://<host>:<port>/<database>" \
  --dbAppUsername "<postgresql-user>" \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

Note:

Using the `--dbAppPassword` option to provide the PostgreSQL database password to the setup tool persists the password to a configuration file.

Instead, run the following command to populate the **`PING_DB_APP_PASSWORD`** environment variable at server start.

```
$ env PING_DB_APP_PASSWORD=<password> bin/start-server
```

Example: Set up the PingAuthorize Policy Editor to use a custom SSL certificate
This example sets up the PingAuthorize Policy Editor in demo mode with a provided SSL server certificate in PKCS12 format.

```
$ env KEYSTORE_PIN_FILE=<path-to-keystore.pin> bin/setup demo
  --adminUsername admin \
  --pkcs12KeyStorePath <path-to-keystore.p12> \
  --certNickname <certificate-nickname> \
  --decisionPointSharedSecret <shared-secret> \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license>
```

Note:

If you don't use the **`KEYSTORE_PIN_FILE`** during **`setup`**, you can supply the **`--keystorePassword`** argument.

The following information describes the previous example code block:

- The **`KEYSTORE_PIN_FILE`** environment variable, along with the **`--pkcs12KeyStorePath`** and **`--certNickname`** command-line options, affect the server's SSL certificate configuration.
- **`KEYSTORE_PIN_FILE`** contains the path to a file containing a valid key store PIN value.
- The **`--pkcs12KeyStorePath`** value is a path to a valid PKCS12 key store file.
- The **`--certNickname`** value is the certificate nickname or alias.

Warning:

- The PingAuthorize Policy Editor only supports lowercase certificate nicknames.
- Because the **`KEYSTORE_PIN_FILE`** is not persisted, it must also be available in the environment of **`start-server`**.

Post-setup steps (manual installation)
After you set up the PingAuthorize Policy Editor, you must start the server from the CLI and then change the PingAuthorize Server configuration to use the Policy Editor as its policy decision point (PDP).

To start the Policy Editor, run the following command.

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

Then, sign on to the Policy Editor. For more information, see *Signing on to the PingAuthorize Policy Editor* on page 108 and import a policy snapshot. You can find a set of default policies in the `resource/policies/defaultPolicies.SNAPSHOT` file.

To configure PingAuthorize Server to use the Policy Editor, use **`dsconfig`** or the administrative console to create a Policy External Server to represent the Policy Editor, 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 Editor" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:pingauthorize" \
  --set "branch:Default Policies" \

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

In the example, the base URL consists of the host name and port chosen for the Policy Editor 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.

Clustering and scaling
PingAuthorize 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 PingAuthorize 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 PingAuthorize Server, see *Deployment automation and server profiles* on page 361.

Next steps
After installed, complete some configuration steps and then start developing policies to enforce fine-grained access to data.

Consider performing the following next steps.

- Configure access token validation.

  For more information, see *Configure access token validation* on page 355.
- Configure a user store.

  For more information, see *User store configuration* on page 354
- Sign on to the administrative console to configure endpoints for existing JSON APIs.

  For more information, see *About the API security gateway* on page 166.
- Sign on to the administrative console to define SCIM APIs for data in databases

  For more information, see *About the SCIM service* on page 192.
- Sign on to the PingAuthorize Policy Editor to create policies.

  For more information, see the PingAuthorize Policy Administration Guide.

Signing on to the PingAuthorize Policy Editor

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

Steps

1. After completing setup for demo mode, sign on to the PingAuthorize Policy Editor 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 PingAuthorize Policy Editor:
   - User name: `admin`
   - Password: `password123`
3. Optional: If you set up the PingAuthorize Policy Editor to use OpenID Connect (OIDC) mode, you must also configure an OIDC provider. For more information, see *Configuring an OIDC provider for single sign-on requests from PingAuthorize* on page 110.

   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 PingAuthorize Policy Editor.

Changing the PingAuthorize Policy Editor authentication mode

You can change the authentication mode after the initial setup.

Steps

- For a manually installed Policy Editor, see *Changing the Policy Editor authentication mode for manual installs* on page 109.
- For a Policy Editor Docker deployment, see *Changing the Policy Editor authentication mode for Docker deployments* on page 109.

Changing the Policy Editor authentication mode for manual installs

About this task

To change the authentication mode that a manually installed PingAuthorize Policy Editor uses, re-run the **`setup`** tool and choose a different authentication mode. This action overwrites the PingAuthorize Policy Editor's existing configuration.

Steps

1. Stop the Policy Editor.
   Example:

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

2. Run the **`setup`** command and select a different authentication mode.
   The modes are:

   - Demo mode

     Configures the PingAuthorize Policy Editor 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 PingAuthorize Policy Editor to delegate authentication and sign-on services to an OpenID Connect provider, such as PingFederate.

   Example:

   ```
   $ bin/setup
   ```

3. Start the Policy Editor.
   Example:

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

Changing the Policy Editor authentication mode for Docker deployments

About this task

To switch to OIDC authentication for a Docker deployment of the PingAuthorize Policy Editor, re-run the **`docker run`** command using the OIDC environment variables.

Steps

1. Stop the Policy Editor Docker container.

2. Run the Policy Editor Docker container in OIDC mode by using the `PING_OIDC_CONFIGURATION_ENDPOINT` and `PING_CLIENT_ID` environment variables in your **`docker run`** command, as shown in the following example.

   Example:

   > **Note:**
   >
   > For proper communication between containers, create a Docker network using a command like `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

```
docker run --network=<network_name> -p 8443:1443 -d \
--env-file ~/.pingidentity/config \
--env PING_EXTERNAL_BASE_URL=localhost:8443 \
--env PING_CLIENT_ID=c2f081c0-6a2e-4249-b07d-d60234bb5b21 \
--env PING_OIDC_CONFIGURATION_ENDPOINT=https://auth.pingone.com/3e665735-23da-40a9-
a2bb-7ccddc171aaa/as/.well-known/openid-configuration \
pingidentity/pingauthorizepap:<TAG>
```

   > **Note:**
   >
   > The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see the *PingAuthorize PAP Docker Image* on Docker Hub.

Configuring an OIDC provider for single sign-on requests from PingAuthorize

When you install the PingAuthorize software with OpenID Connect (OIDC) authentication, configure an OIDC provider to accept SSO requests from PingAuthorize.

About this task

If you chose OIDC mode when you set up the PingAuthorize Policy Editor, you must configure an OIDC provider, such as *PingFederate* or *PingOne*, to accept sign-on requests from the PingAuthorize Policy Editor.

If you're using another OIDC provider, see the provider's documentation for specific client configuration steps. The following steps show the general procedure:

Steps

1. Use the following configuration values to create an OAuth 2 client that represents the PingAuthorize Policy Editor.

OAuth 2 client configuration	Configuration value
Client ID	`pingauthorizepolicyeditor`
Redirect URI	`https://<host>:<port>/idp-callback`
Grant type	`Authorization Code with PKCE`
Response type	`code`
Scopes	▪ `openid` ▪ `email` ▪ `profile`

OAuth 2 client configuration	Configuration value
Refresh tokens	`Enable`
Client authentication on the token endpoint	`Disable` The Policy Editor doesn't have access to the client secret and doesn't send credentials to the token endpoint.
Return ID token on refresh grant	`true`
Always re-roll refresh tokens	`true`

Important:

When an authentication token expires, the Policy Editor performs a silent renewal, triggering a background process to retrieve a new token from the OIDC provider. For this process to work, you must configure your OIDC provider to issue refresh tokens in the following manner:

- Issue an `id_token` as part of the refresh grant.
- Re-roll the refresh token after each use. The Policy Editor will not use refresh tokens more than once.

Because these constraints apply to silent renewal, a misconfiguration of the previous items will still allow you to sign on. After your token expires, though, the application will eject you from your session and redirect you to the sign-on screen. This could cause you to lose unsaved changes in the Policy Editor.

2. Configure the access tokens and ID tokens issued for the OAuth 2 client with the following claims:
   - `sub`
   - `name`
   - `email`
3. Configure the OIDC provider to accept a cross-origin resource sharing (CORS) origin that matches the PingAuthorize Policy Editor's scheme, public host, and port, such as `https://<host>:<port>`.
4. Configure the OIDC provider to issue tokens to the PingAuthorize Policy Editor 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 SHA-256.

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

Configuring PingOne as an OIDC provider for PingAuthorize

To improve security and ensure a consistent authentication experience across all enterprise applications, enable single sign-on (SSO) for the PingAuthorize Policy Editor using PingOne as an OIDC provider.

Components

- PingOne
- PingAuthorize 9.0 or later

Instructions and screenshots might differ slightly from other product versions. For the latest documentation, see *PingOne documentation*.

Before you begin

- Confirm that PingOne is accessible from the subnet on which the Policy Editor is running.
- Extract the Policy Editor distribution to your specified install location, with appropriate permissions set for write access, for example `/opt/PingAuthorize-PAP`.

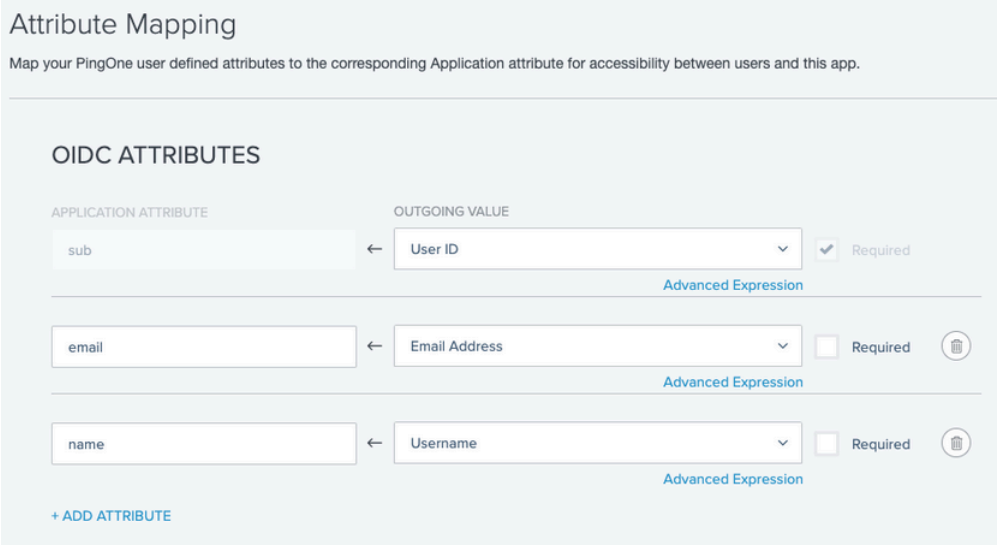
Configuring PingOne for PingAuthorize policy administration
Configure PingOne to authorize external access to the PingAuthorize Policy Editor.

About this task
The following configuration allows any authenticated user to access the Policy Editor.

Steps

1. Sign on to PingOne and click your environment.
   Choose from:
   - If you have an account, go to the URL for your environment. Each environment has a unique URL for signing in. It follows the format https://console.pingone.com/?env=<*environmentID*>.
   - If you do not already have an account, create one at Try Ping.
2. To create an application in PingOne to represent the PingAuthorize Policy Editor, go to **Connections** # **Applications** and click **+ Add Application**.
3. Go to **Connections** # **Applications** and click **+ Add Application**.
4. Click **Single Page App** and then click **Configure**.
5. Enter a name for the application, such as PingAuthorize Policy Editor.
6. Optional: Enter a description and add an icon.
7. Click **Next**.
8. Add a redirect URL that follows the format https://*pap.hostname:port*/idp-callback.
9. Click **Save and Continue**.
10. On the **Grant Access to Your Application** window, add scopes for **email** and **profile**.
11. Click **Save and Continue**.

12. On the **Attribute Mapping** window:
    a. Accept **UserID = sub**.
    b. Click **+ Add Attribute** # **PingOne Attribute** to add **Email Address = email**.
    c. Click **+ Add Attribute** # **PingOne Attribute** to add **Formatted = name** or **Username = name**.



13. Click **Save and Close**.
14. To enable the application, click the **Enable** toggle.

15. Copy the following IDs:

    Client ID

    To find the Client ID, go to the application's **Profile** tab.

    Environment ID

    To find the Environment ID, click **Environment** in the left navigation pane.

> **Note:**
> You'll need them when you configure the Policy Editor to use PingOne.

Configuring PingAuthorize policy administration to use PingOne
Configure the PingAuthorize Policy Editor to use PingOne for authentication.

About this task
The following instructions apply to a manually installed PingAuthorize Policy Editor.

Steps

1. Run the *PingAuthorize-PAP*`/bin/stop-server` command to stop the Policy Editor.
2. Using the client ID and environment ID from *Configuring PingOne for PingAuthorize policy administration* on page 112, run the following command to configure the GUI.

```
bin/setup oidc \
--licenseKeyFile </path/to/PingAuthorize.lic> \
--generateSelfSignedCertificate \
--hostname <pap-hostname> --port <pap-port> \
```

```
--adminPort <admin-port> \
--oidcBaseUrl https://auth.pingone.com/<environment-id>/as \
--clientId <client-id>
```

3. Run the **`bin/start-server`** command to start the PingAuthorize Policy Editor.
4. Verify that you can sign on to the Policy Editor using the application you created in PingOne.
   a. Go to the Policy Editor.
   b. Click **Click to Sign in**.

      Result: Your browser will redirect to the URL you set in *Configuring PingOne for PingAuthorize policy administration* on page 112.

Configuring PingFederate as an OIDC provider for PingAuthorize
To improve security and ensure a consistent authentication experience across all enterprise applications, enable single sign-on (SSO) for the PingAuthorize Policy Editor using PingFederate as an OIDC provider.

This document describes one way to configure PingFederate as an OpenID Connect provider for the PingAuthorize Policy Editor. In this example, PingFederate also acts as the identity provider and uses a PingDirectory LDAP server with sample data as the backing store.

Components

- PingFederate 10.3 or later
- PingDirectory 9.0 or later
- PingAuthorize 9.0 or later

Instructions and screenshots might differ slightly from other product versions. For the latest documentation, see the *PingFederate documentation* and *PingDirectory documentation*.

Before you begin

Make sure of the following:

- PingFederate is running and accessible from the subnet on which the Policy Editor is running.
- PingDirectory is running and accessible from the subnet on which PingFederate is running.
- PingDirectory is loaded with the identities to be used. This document uses the sample data provided when running the PingDirectory setup command line tool with option `--sampleData 1000`.
- You have extracted the Policy Editor distribution to your specified install location, with appropriate permissions set for write access. This document uses an installation directory of `/opt/PingAuthorize-PAP`.
- If using SSL, the certificate chain is available as a PKCS12 keystore to upload as the server certificate chain for PingFederate.
- The signing certificate for JWT tokens is available for upload to PingFederate.

> **Note:**
>
> If the PingFederate certificate chain contains certificates that are not trusted by the default Java truststore on the system that the Policy Editor is running on, you will need to add them. An example of how to do this is provided in the “Add Certificate to Java Trust Store” subsection below.

Configuring PingFederate for PingAuthorize
Configure PingFederate to authorize external access through tokens to the PingAuthorize Policy Editor.

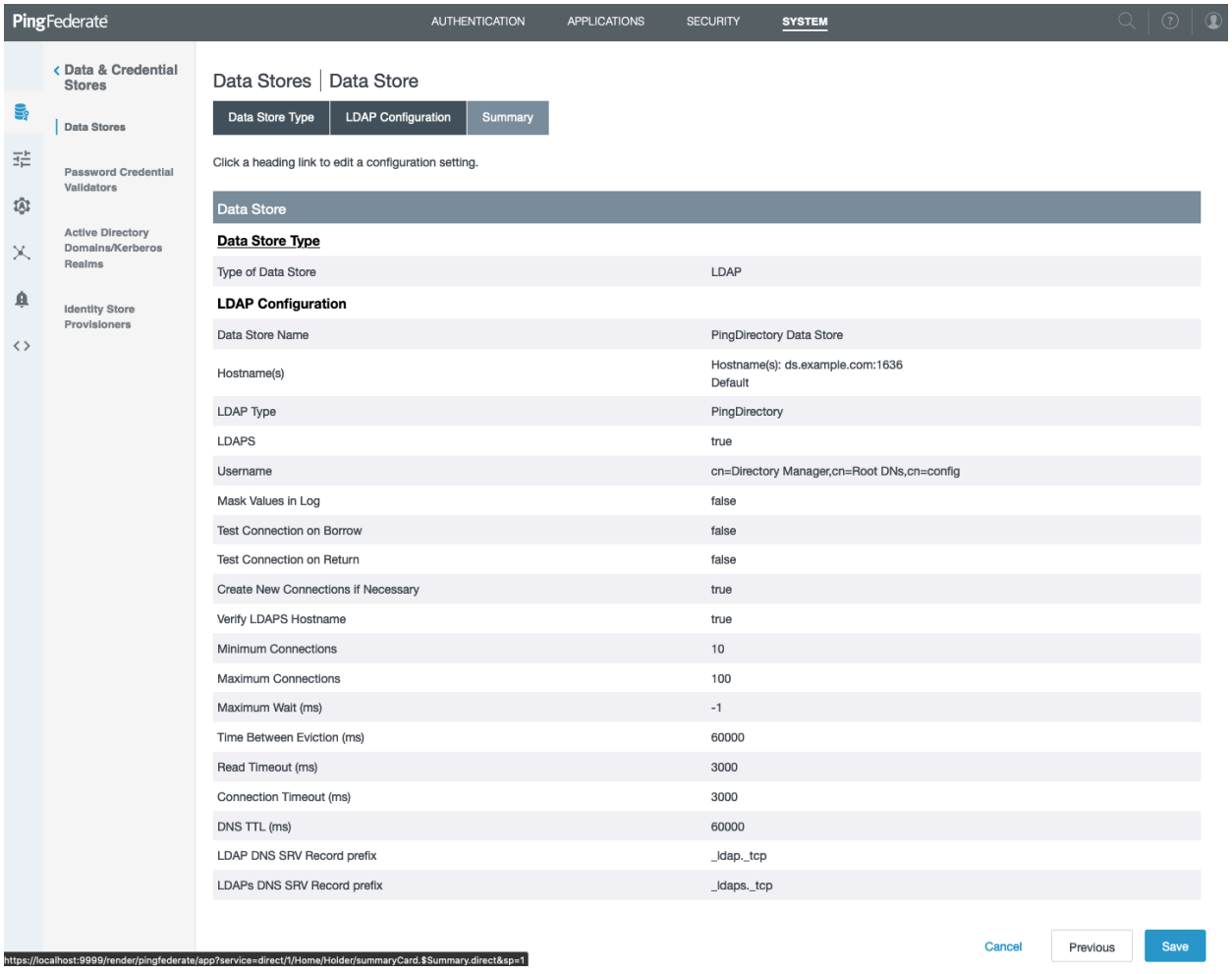
About this task

You can also use PingAccess to authorize external access through rules. See *Rule Creation in PingAccess* for information.

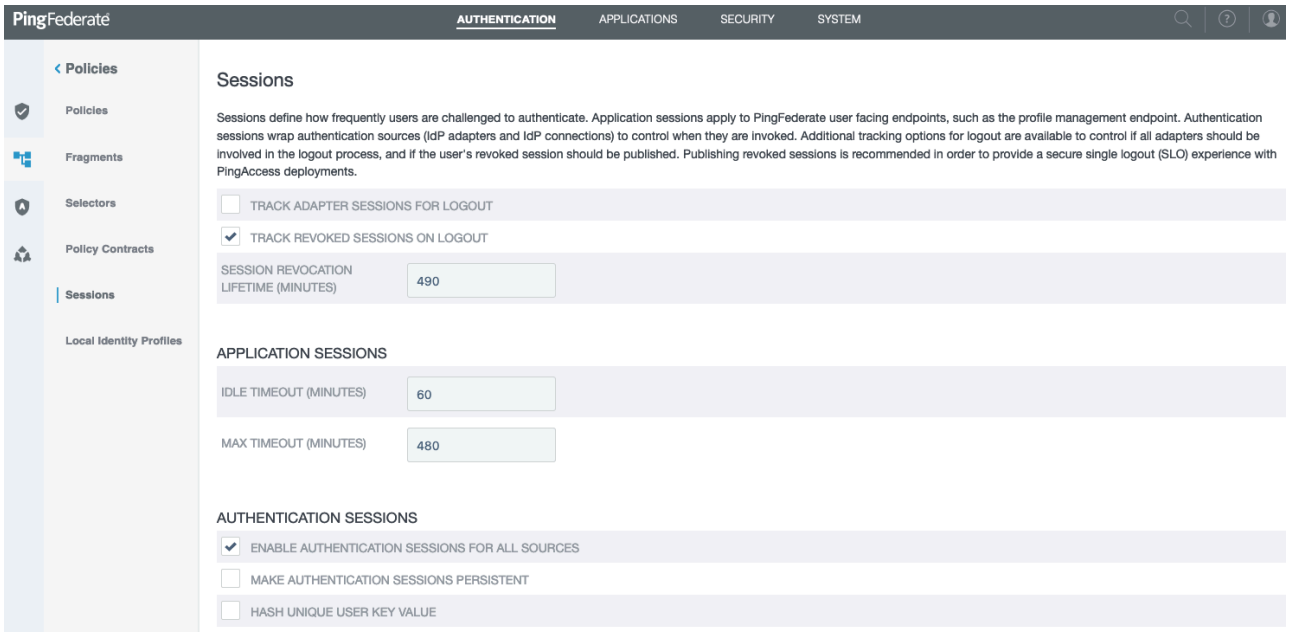
The following example configuration assumes that any authenticated user can access the PingAuthorize Policy Editor. To limit access to members of a specific group, see *Configuring PingFederate group access for PingAuthorize* on page 124.

Steps

1. In the PingFederate administration console, go to **System** # **Data & Credential Stores** # **Data Stores**.
2. Click **Add New Data Store**.
3. On the **Data Store Type** tab, in the **Name** field, enter a name for the data store.
4. From the **Type** list, select **Directory (LDAP)**, and then click **Next**.
5. On the **LDAP Configuration** tab, enter the address and authentication information for PingFederate to use when accessing PingDirectory, and then click **Next**.
6. On the **Summary** tab, review your configuration and click **Save**.



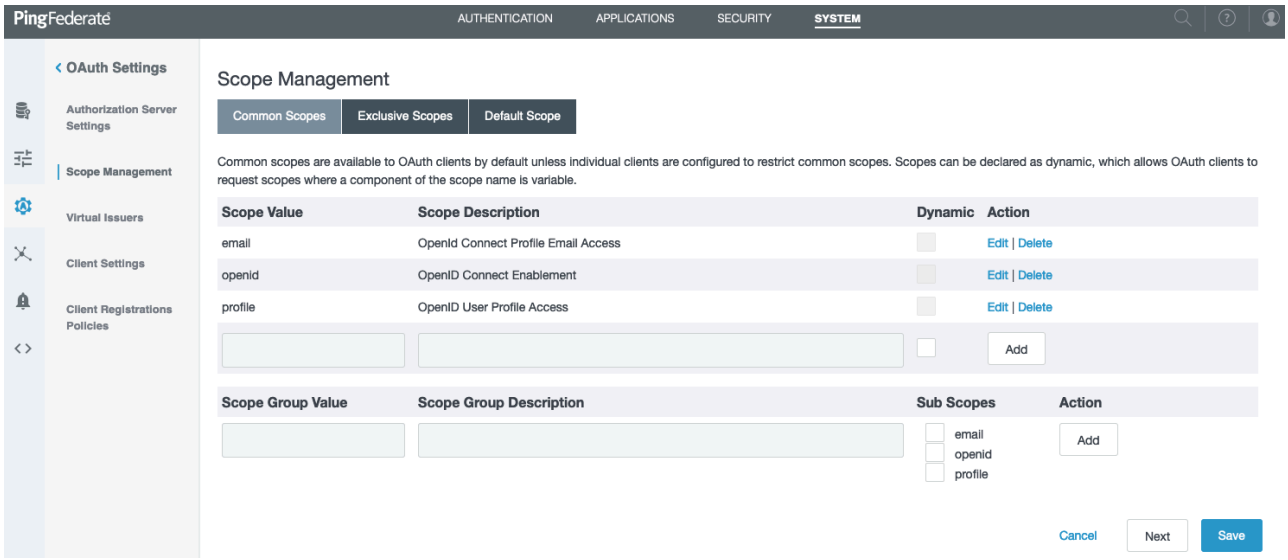
7. Go to **Authentication** # **Policies** # **Sessions** and enable authentication sessions. The following example enables authentication sessions for all sources. Make the appropriate change for your environment, and then click **Save**.



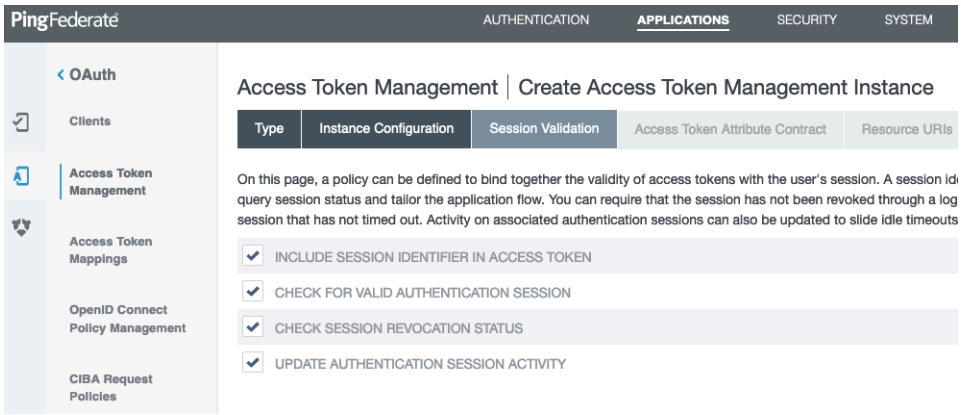
8. Go to **Security** # **Certificate & Key Management** # **SSL Client Keys & Certificates** and import your JWT signing certificate. Click **Save**.

> **Note:**
>
> PingFederate expects the certificate chain and keys to be encoded in PKCS12 format.

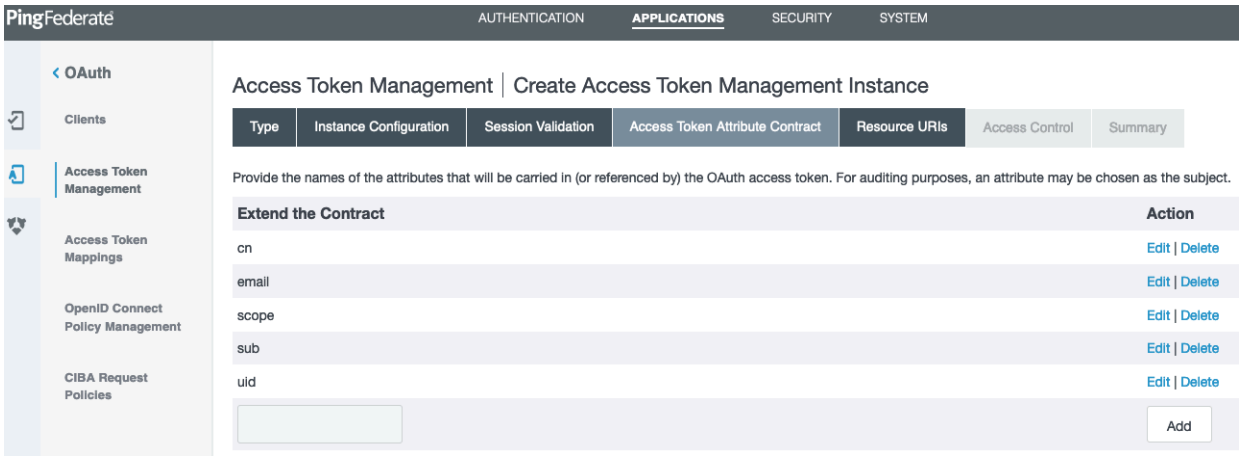
9. Configure your OAuth server using the OpenID Connect protocol.
   a. Go to **System** # **OAuth Settings** # **Scope Management** and create scopes.
   b. In the **Scope Value** field, enter the `email`, `openid`, and `profile` scopes, clicking **Add** after each entry. Click **Save**.



   c. Go to **Applications** # **OAuth** # **Access Token Management** and click **Create New Instance**.
   d. On the **Type** tab, from the **Type** list, select **JSON Web Tokens**. From the **Parent Instance** list, select **None**. Click **Next**.
   e. On the **Instance Configuration** tab, click **Add a new row to 'Certificates'** and add the previously imported signing certificate. Select the desired signing algorithm and token timeout, and then click **Next**.
   f. On the **Session Validation** tab, enable the session validation options.



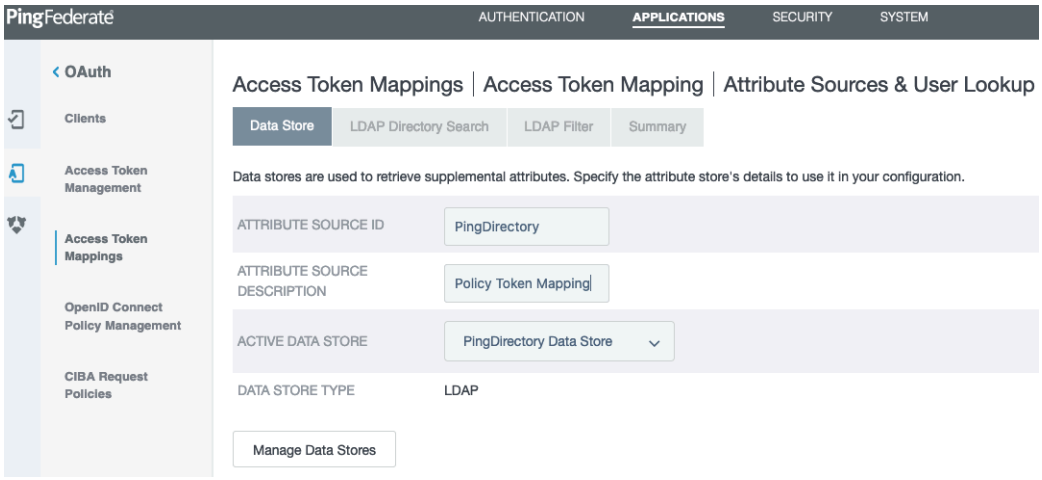
   g. On the **Access Token Attribute Contract** tab, add the attributes to be included in the OAuth access token. This example extends the contract with `cn`, `email`, `scope`, `sub`, and `uid` attributes.



h. Click **Next** until you reach the **Summary** tab, and then click **Save**. Accept the default values for the **Resources URIs** and **Access Control** settings.

i. Go to **Applications** # **OAuth** # **Access Token Mappings** to create an Access Token Mapping in the **Default** context for the Access Token Manager you just created. Click **Add Mapping**, and then click **Add Attribute Source**.

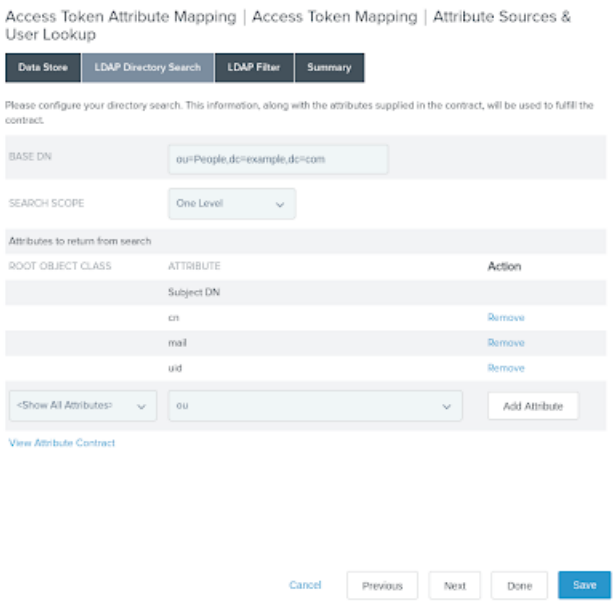
j. From the **Active Data Store** list, select the PingDirectory data store that you created in step 2. Click **Next**.



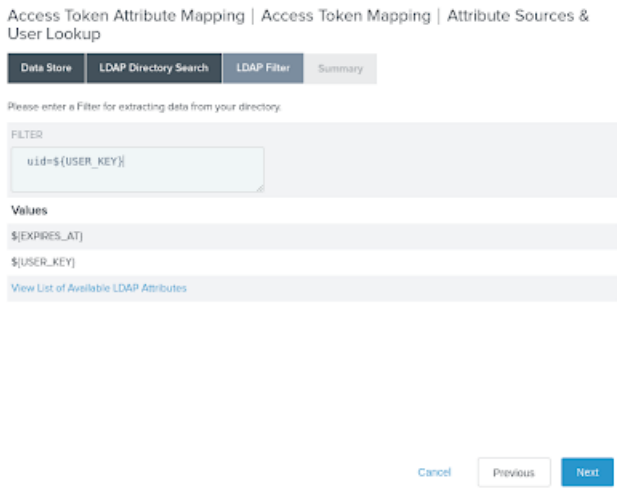
k. On the **LDAP Directory Search** tab, in the **Base DN** field, enter the base DN for the PingDirectory data that provides your identities.

l. In the **Attributes to return from search** section, click **Add Attribute** and enter the attributes to be retrieved.

   The sample data uses `ou=People,dc=example,dc=com` and the configuration shown in the following image retrieves the `cn`, `mail`, and `uid` attributes.

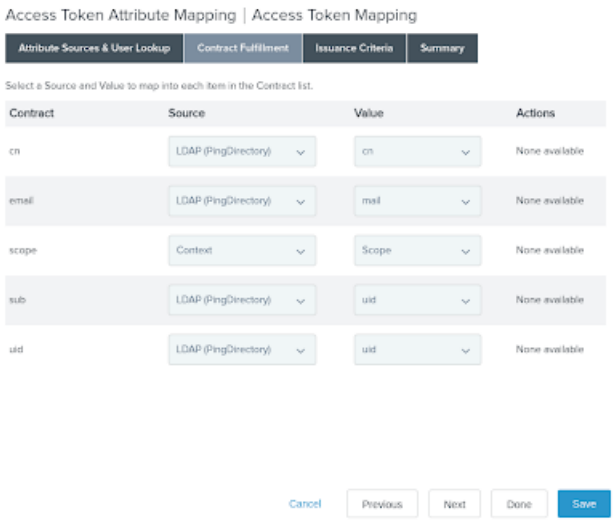


m. On the **LDAP Filter** tab, in the **Filter** field, enter `uid=${USER_KEY}` to match the PingDirectory sample data with the authenticating user information.

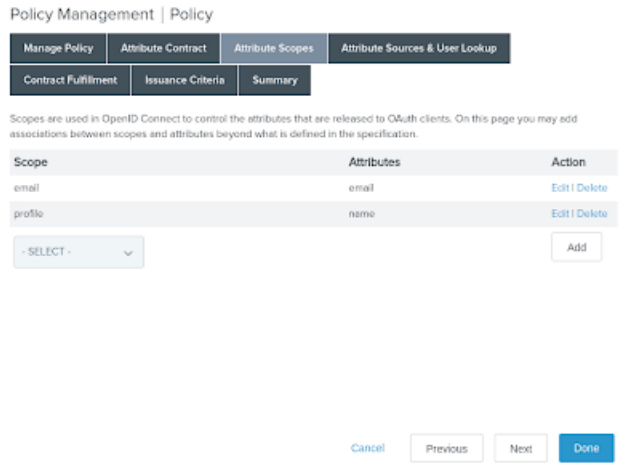


n. Click **Next** and **Save** on the **Summary** tab.
o. On the **Contract Fulfillment** tab, fulfill the contract with the LDAP attributes from the PingDirectory data store. Leave the remaining settings as their defaults and click **Save**.

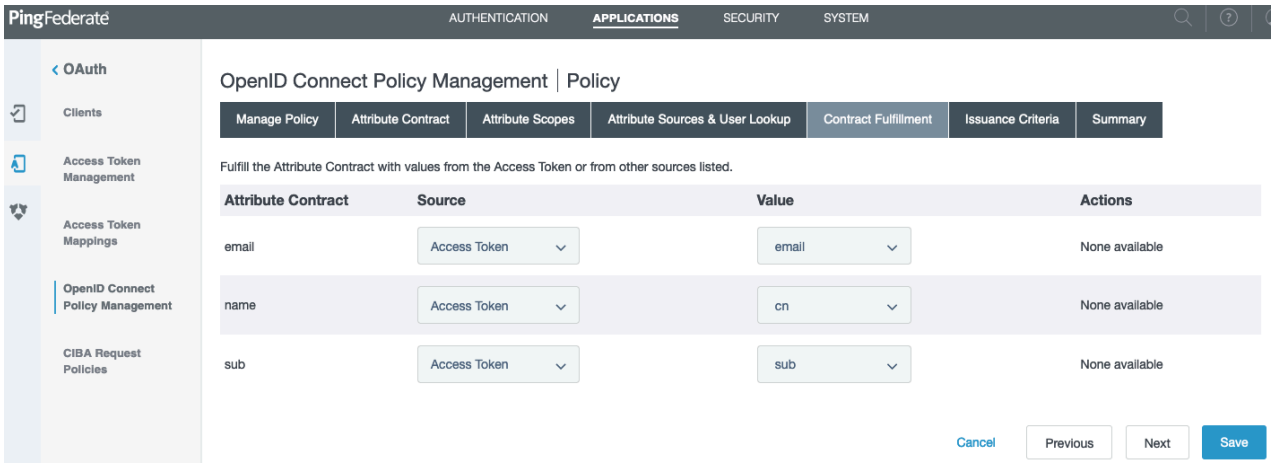
   The scope attribute is fulfilled from the OAuth context.



p. Go to **Applications** # **OAuth** # **OpenID Connect Policy Management** and click **Add Policy**.
q. In the **Manage Policy** tab, from the **Access Token Manager** list, select the access token manager you previously created.
r. Ensure that the **Include User Info in ID Token** check box is selected. Click **Next**.
s. On the **Attribute Contract** tab, extend the policy contract with the `email` and `name` attributes. Click **Next**.
t. On the **Attribute Scopes** tab, map the previously defined `email` and `profile` scopes to the `email` and `name` ID token attributes. Click **Next**.



u. On the **Contract Fulfillment** tab, fulfill the contract with the values in the access token. Click **Next** until you reach the **Summary** tab, and then click **Save**.



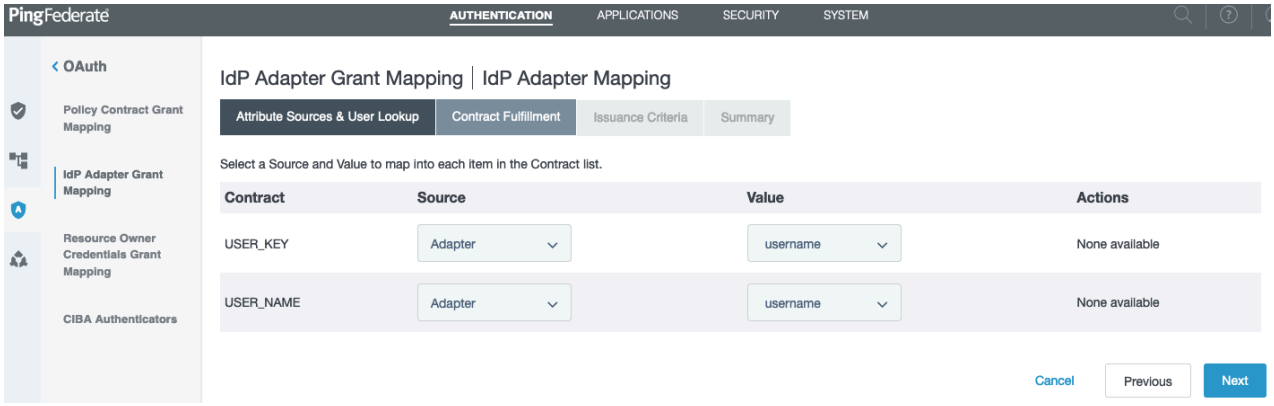
v. Click **Set as Default** to set the newly created policy as the default policy.

w. Go to **Applications** # **OAuth** # **Clients** and click **Add Client**.

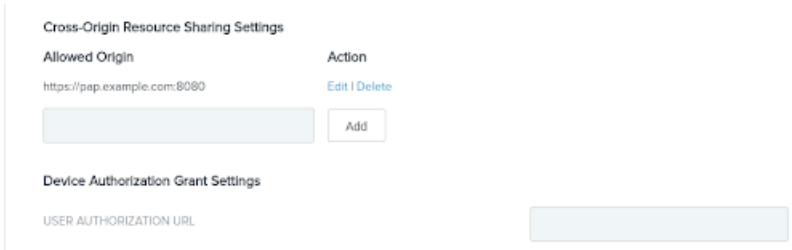
   To provide the Policy Editor with appropriate defaults, configure PingFederate with a **Client ID** of `pingauthorize-pap` and select the **Implicit** check box in the **Allowed Grant Types** section. From the **Default Access Token Manager** list, select the JWT Manager created earlier, and in the **Redirection URIs** field, add the correct callback URL for the Policy Editor, such as https://pap.example.com:9443/idp-callback.

   Click **Save**.

x. Go to **Authentication** # **OAuth** # **IdP Adapter Grant Mapping** and create a new Form Adapter Mapping, fulfilling the contracts for `USER_NAME` and `USER_KEY` with the **username** form field. Click **Next** and **Save** on the **Summary** tab.



y. Because this PingFederate instance uses a different domain from the Policy Editor, you must modify the PingFederate CORS settings. Go to **System** # **OAuth Settings** # **Authorization Server Settings**. In the **Cross-Origin Resource Sharing Settings** section, enter the domain for the Policy Editor in the **Allowed Origin** field. Click **Add** and then **Save**.



Result: PingFederate is configured to handle OpenID Connect requests.

Next steps

Configuring PingAuthorize Policy Editor to use PingFederate on page 122

Configuring PingAuthorize Policy Editor to use PingFederate

Configure the PingAuthorize Policy Editor to use PingFederate for authorization.

Before you begin

Configure PingFederate to handle OpenID Connect requests as described in *Configuring PingFederate for PingAuthorize* on page 114.

About this task

Reconfigure a manually installed PingAuthorize Policy Editor to use PingFederate for authorization.

Steps

1. Add the certificate to the Java Trust Store.

   If the certificate chain added to PingFederate uses an intermediate certificate authority that is not trusted by the default Java trust store, you must add the certificate. Use the following command (root permissions are usually required). `$JAVA_HOME` must be defined as the installation location of the JVM on which the Policy Editor will run.

   ```
   keytool -import \
   -file /path/to/IntermediateCA.cer \
   -keystore $JAVA_HOME/jre/lib/security/cacerts \
   -storepass changeit
   ```

2. Reconfigure PingAuthorize to point unauthenticated users to PingFederate.
   a. Stop the application.

      ```
      $ bin/stop-server
      The server was successfully stopped.
      ```

   b. Re-run **`bin/setup`** to reconfigure the application.
   c. Select OpenID Connect to configure the Policy Editor.

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

      There is an existing configuration file at /config/configuration.yml.
       Overwrite? (yes /
      no) [no]: yes
      Detected valid license file in server root PingAuthorize.lic

      PingAuthorize Policy Editor
      ===========================================

      How would you like to configure the Policy Editor?

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

      Enter option [1]: 2

      On which port should the Policy Editor listen for HTTPS
       communications? [9443]:
      ```

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

d. Ensure that the PingFederate discovery endpoint uses the public DNS name of the PingFederate server. In this example, the Policy Editor uses a self-signed SSL certificate.

```
Enter the port of the OpenID Connect provider [9031]:

Enter the fully qualified host name or IP address of the OpenID
 Connect provider [pap.example.com]: pf.example.com

Certificate server options:

    1)  Generate self-signed certificate (recommended for testing
 purposes only)
    2)  Use an existing certificate located on a Java Keystore (JKS)
    3)  Use an existing certificate located on a PKCS12 keystore

Enter option [1]:

There already exists a keystore at /config/keystore.p12. Do you want
 to delete it? (yes / no) [no]: yes
```

e. Follow the remaining prompts.

```
   Setup Summary
=======================================
Host Name:        pap.example.com
Server Port:      9443
Secure Access:    Self-signed certificate
Admin Port:       9444
Periodic Backups: Enabled
Backup Schedule:  0 0 0 * * ?

Command-line arguments that would set up this server non-
interactively:
    setup oidc --pkcs12KeyStorePath config/keystore.p12 --
licenseKeyFile PingAuthorize.lic \
         --oidcHostname pf.example.com --oidcPort 9031 --certNickname
 server-cert --backupSchedule '0 0 0 * * ?' \
         --hostname pap.example.com --port 9443 --
generateSelfSignedCertificate --adminPort 9444

What would you like to do?

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

Enter option [1]:

Setup completed successfully

Please configure the following values
================================================================================
PingAuthorize Server - Policy External Server
  Base URL:                                       https://
pap.example.com:9443
  Shared Secret:
 2222142a754f4838ad1e3dccb6e93940
  Trust Manager Provider:                         Blind Trust

PingFederate - OAuth Client Config
```

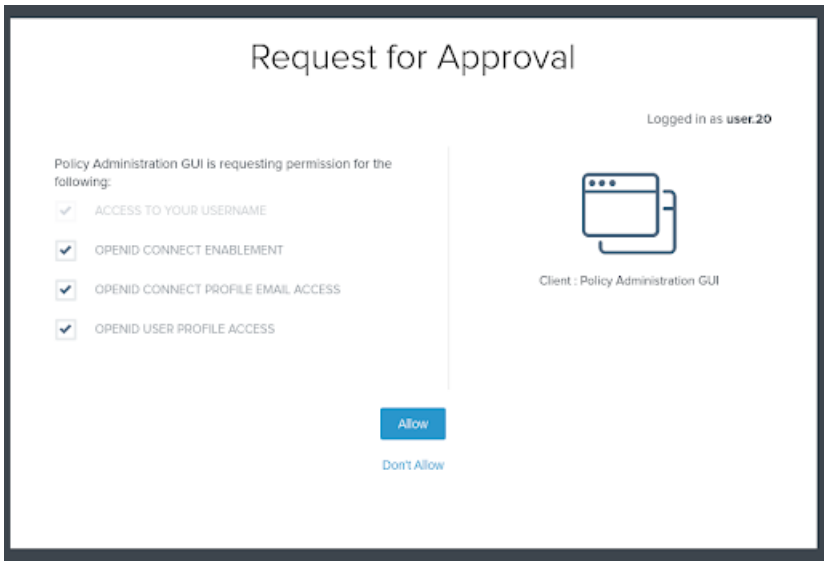
```
  Client ID:
 pingauthorizepolicyeditor
  CORS Allowed Origin:                                    https://
pap.example.com:9443
  Redirect URL:                                           https://
pap.example.com:9443/idp-callback

Please start the server by running bin/start-server
```

f. Restart the application by running **`bin/start-server`**.

3. Verify that you can log into the Policy Editor using OpenID Connect provided by PingFederate.
   a. Go to the Policy Editor, for example, https://pap.example.com:9443. Your browser should be redirected into the OAuth flow.
   b. Click **Click to Sign In**.
   c. Sign on with your user name and password.

      The sample configuration in this document creates an identity with the user name `user.20` and password `password`.
   d. Once authenticated, PingFederate will prompt the user with the scopes associated with the OAuth client. Check all of them to continue.



Result: You are now authenticated and authorized to view the Policy Editor.

Configuring PingFederate group access for PingAuthorize

Configure PingFederate so that only members of a specific LDAP group are authorized to access the application.

About this task

Configuring PingFederate for PingAuthorize on page 114 and *Configuring PingAuthorize Policy Editor to use PingFederate* on page 122 explain how to configure the PingAuthorize Policy Editor and PingFederate so that any authenticated user can access the PingAuthorize Policy Editor. This task describes how to configure PingFederate to limit access to a specific LDAP group.

Steps

1. Create an LDAP group in PingDirectory and add the desired user (`user.20`) to the group.
   a. Create a file named `create-policy-writer-group.ldif` and add the following.

```
dn: ou=groups,dc=example,dc=com
objectclass: top
```

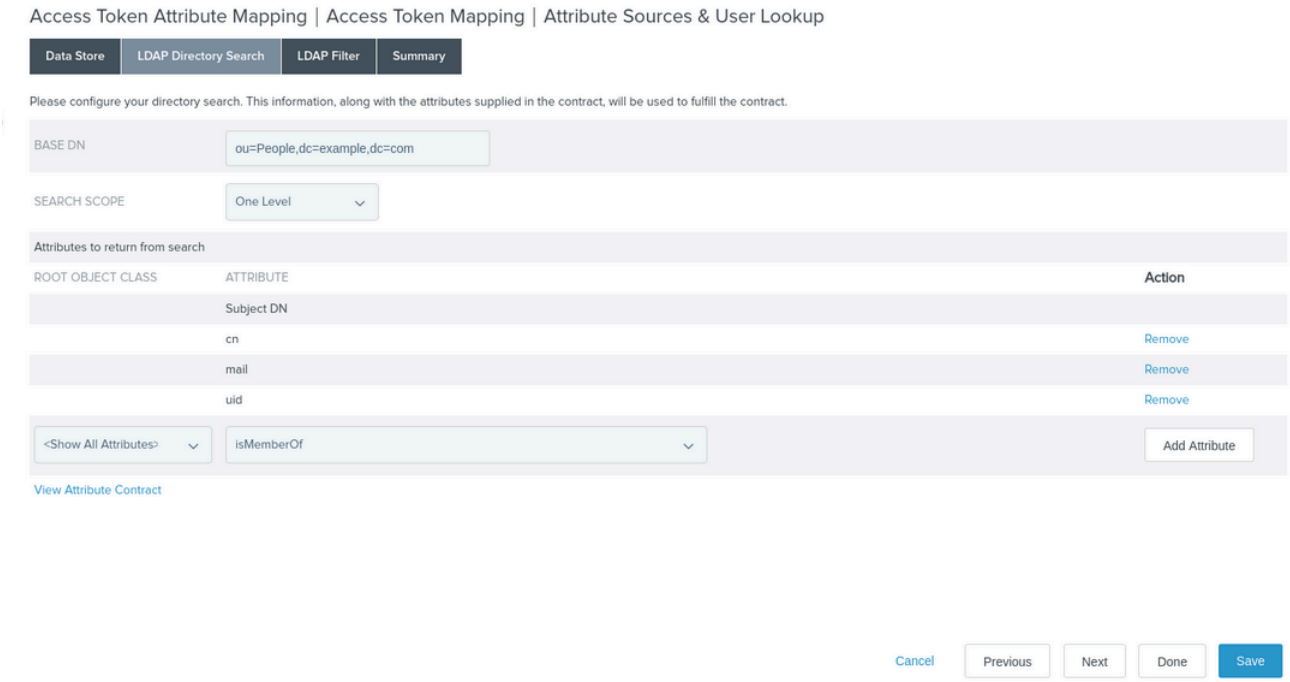
```
objectclass: organizationalunit
ou: groups

dn: cn=PolicyWriter,ou=groups,dc=example,dc=com
objectclass: top
objectclass: groupOfUniqueNames
cn: PolicyWriter
ou: groups
uniquemember: uid=user.20,ou=People,dc=example,dc=com
```

b. Use the PingDirectory **`ldapmodify`** tool to load the newly created `ldif` file.

```
/opt/PingDirectory/bin/ldapmodify -a -f create-policy-writer-
group.ldif
```

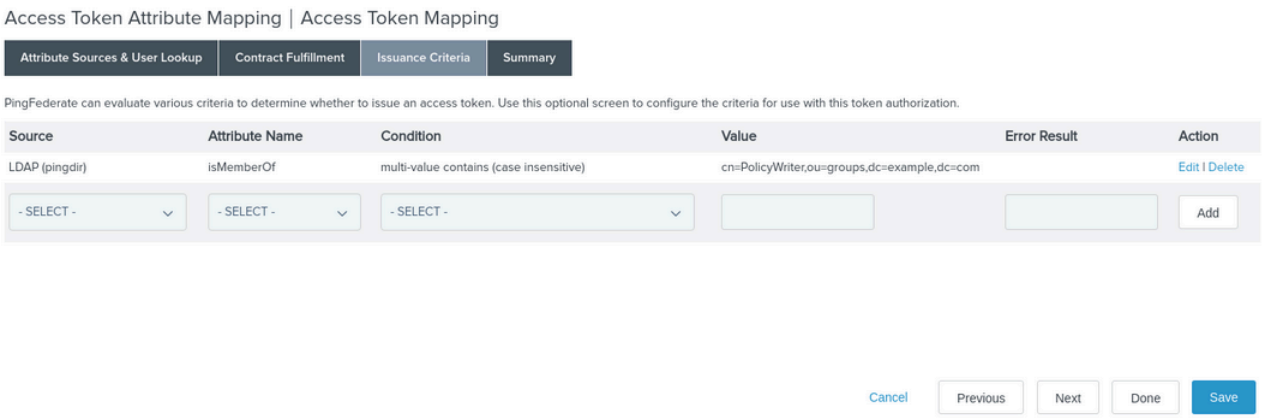
2. Add the group membership claim requirement in PingFederate.
   a. In the PingFederate console, go to **Applications** # **OAuth** # **Access Token Mappings**.
   b. Select the PingDirectory mapping from the list, and then on the **Attribute Sources & User Lookup** tab, select the PingDirectory source.
   c. Click the **LDAP Directory Search** tab, and in the **Root Object Class** list, select **Show All Attributes**.
   d. Add the **isMemberOf** attribute, and then click **Done** to return to **Access Token Attribute Mapping**.



   e. Go to the **Issuance Criteria** tab and add a new row with the following values:

Column	Value
Source	**LDAP (pingdir)**
Attribute Name	**isMemberOf**
Condition	**multi-value contains (case sensitive)**

Column	Value
Value	**cn=PolicyWriter,ou=groups,dc=example,dc=com**



f. Click **Save**.

Result: Only `user.20` can access the PingAuthorize Policy Editor.

3. Verify that only specified users can access the PingAuthorize Policy Editor.

> **Note:**
>
> Clear any active SSO sessions before you sign on as each user.

a. Sign on as `user.0`.
   Result: Access is denied.
b. Sign on as `user.20`.
   Result: Access is granted.

Upgrading PingAuthorize

PingAuthorize includes two server applications you must upgrade in tandem—the main PingAuthorize Server and the Policy Editor.

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

> **Note:**
>
> PingAuthorize Server used in external PDP mode requires a Policy Editor with the same version. When upgrading PingAuthorize Server, you must also upgrade the Policy Editor.

Upgrade considerations

When upgrading, you must consider factors such as the scope of the update, the PingAuthorize or PingDataGovernance version from which you are upgrading, and if you are not using Docker, your installed version of Java.

Note:

The 8.3.0.0 release is the first release of PingAuthorize. Previously, the product was known as PingDataGovernance.

General considerations

For Docker deployments, the upgrade process involves downloading and deploying the latest containers.

For manual installations, the upgrade process involves downloading and extracting a new version of the PingAuthorize 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:

- If you are upgrading from a PingAuthorize Early Access release to a PingAuthorize General Availability release, you must upgrade both the PingAuthorize Server and the Policy Editor before you use the Policy Decision Service in external mode. Upgrading only one component results in this error: `Please upgrade to PingAuthorize Policy Editor version <X.X.X.X>`.
- 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*.

Note:

For information about important fixes made over several releases, see *Critical Fixes*.

Considerations introduced in PingAuthorize 9.0.0.0

Keep in mind the following important upgrade considerations introduced in this version of PingAuthorize Server.

General

Peer server setup has been removed. To manage server configuration, use server profiles instead of peer setup. Server profiles support deployment best practices such as automation and Infrastructure-as-Code (IaC). For more information about server profiles, see *Deployment automation and server profiles* on page 361.

Spring compatibility

Spring configuration properties in PingAuthorize administrative console configuration files prior to version 9.0.0.0 are not compatible with the administrative console bundled with PingAuthorize 9.0.0.0 and later. This incompatibility is caused by major updates to Spring dependencies. Attempting to use these older configuration files will result in the administrative console failing to start.

If you are using older PingAuthorize administrative console configuration files, these should be updated. Replace the following excerpt in the old `application.yml` file:

```
spring:
  profiles.active: default
  main.show-banner: false
  thymeleaf.cache: true
  thymeleaf.prefix: classpath:/public/app/
```

with the following:

```
spring:
  profiles.active: default
  web.resources:
    # 1 year. Update the corresponding value in MvcConfig if this
 changes.
    cache.period: 31536000
    add-mappings: false # use our custom mappings instead of the
 defaults
  main:
    banner-mode: "OFF"
  thymeleaf:
    prefix: classpath:/public/app/
```

Upgrade considerations introduced in PingAuthorize 8.x

Considerations introduced in PingAuthorize 8.3.0.0

Keep in mind the following important considerations introduced in this version of PingAuthorize Server.

General

- If you are upgrading to PingAuthorize 8.3.0.0, you must also upgrade to PingAuthorize Policy Editor 8.3.0.0.
- The policy decision service configuration has changed. When using embedded pdp mode, you must specify a `deployment-package-source-type` for the policy decision service in your configuration. You might need to update the **`dsconfig`** files in your server profile when upgrading to version 8.3 to set a `deployment-package-source-type`. If you want to maintain the existing behavior from previous releases, use `"static-file"` as your `deployment-package-source-type`.

Deployments with multi-server topologies

- Upgrading from PingDataGovernance 6.x or 7.x

  Upgrading multi-server topologies that contain PingDataGovernance 6.x or 7.x is not supported.
- Upgrading from PingDataGovernance 8.0.0.0 or later

  You can upgrade multi-server topology deployments that contain PingDataGovernance 8.0.0.0 or later to PingAuthorize. When updating a PingDataGovernance multi-server topology to

PingAuthorize, you must remove all servers from the topology, update each server individually, then add all the servers back to the topology, as explained below.

> **Note:**
>
> The known issues and workarounds in this section apply only to deployments with multi-server PingDataGovernance topologies. Deployments with single-server topologies can upgrade without these issues.

For each server to be upgraded:

1. Remove the server from the topology by running a command like this one.

   ```
   manage-topology remove-server <connection args>
   ```

2. Update the server.

After you have successfully upgraded every server, you can then join each server to the topology by running a command like this one.

```
manage-topology add-server <connection args> <remote server
 connection args>
```

If you do not follow these steps, adding a PingAuthorize server to a PingDataGovernance topology could result in the following error message:

```
Entry cn=License,cn=Topology,cn=config cannot be modified because
 one of the
configuration change listeners registered for that entry rejected
 the change: The provided
license key was generated for PingAuthorize but this is
 PingDataGovernance with Symphonic
```

Another consequence of not following these steps is that restarting any server in the topology that is not updated fails. To use the server again, you must remove the server from the topology and reset its license to a PingDataGovernance license.

Upgrading from a version earlier than 7.3.0.0

If you are upgrading from a PingDataGovernance version earlier than 7.3.0.0, PingAuthorize creates the `deleted-oauth2-scopes.txt` file to capture data that can simplify the upgrade. For information about what to do with this file, contact your Ping Identity account representative.

Considerations introduced in PingDataGovernance 8.2.0.0

Keep in mind the following important considerations introduced in this version of PingDataGovernance Server.

General

- If you are upgrading to PingDataGovernance 8.2.0.0, you must also upgrade to Policy Administration GUI 8.2.0.0.

- Changes to SpEL expressions using collection projection might cause policy errors with the following form.

  ```
  EL1004E: Method call: Method <Symphonic Value method>() cannot be
   found on type <native Java type>
  ```

  If your policies rely on SpEL collection projection and methods like intValue(), stringValue(), jsonRepresentation(), or pojoRepresentation(), you must update these expressions. It is recommended that you update the policies to use collection transforms instead of SpEL collection projection. For information about collection transforms, see the *PingDataGovernance Policy Administration Guide*.
- 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.

Gateway API Endpoint and Sideband API Endpoint configurations

- PingDataGovernance now strictly validates path parameters used in Gateway API Endpoint and Sideband API Endpoint configurations. The `inbound-base-path` value (for Gateway API Endpoints) and the `base-path` value (for Sideband API Endpoints) no longer allow duplicate path parameters. For example, "/Users/{userId}/Manager/{userId}" defines the "userId" path parameter twice and is invalid. In addition, other configuration properties cannot refer to a path parameter that is not defined by `inbound-base-path` or `base-path`.

  Previously, the server would allow such invalid configuration changes to be saved, but now the server rejects them. Upgrades or server profile deployments including invalid configuration of this kind will now fail. If this happens, correct the invalid configuration values.

Considerations introduced in PingDataGovernance 8.1.0.0

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 PingAuthorize 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 *PingAuthorize 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.

Docker upgrades

Upgrading PingAuthorize Server using Docker

When using Docker, instead of upgrading PingAuthorize Server, you deploy a container with the new PingAuthorize version and use the same server profile.

About this task

If you deployed a container using a server profile, when you want to deploy a newer PingAuthorize Server version, you deploy a container with that version using the same server profile.

Steps

- For more information, see *https://devops.pingidentity.com/reference/config/*.

  (The server profiles for Docker deployments differ from those discussed in *Deployment automation and server profiles* on page 361.)

Upgrading the PingAuthorize Policy Editor using Docker

If you originally installed the Policy Editor with Docker per *Deploying PingAuthorize Policy Editor using Docker* on page 86, use this procedure to upgrade the PingAuthorize Policy Editor when a new version is released.

Steps

1. In your current Policy Editor, complete the steps in *Backing up policies* on page 136.
2. Stop the old Docker container and start the new one.

   When a new Docker image for the PingAuthorize Policy Editor is available, you stop the existing Docker container and start the new container from the new image while mounting the same volumes.

   Warning:

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

   The following commands stop the running container and run a new image named <*pap_new*>. This image uses the volumes from <*pap_old*> to house the policy database. Also, the command uses the same **`PING_H2_FILE`** location from *Example: Override the configured policy database location* on page 165.

   Note:

   - The Ping Identity DevOps Docker images use the PingAuthorize **`setup`** tool to update the H2 policy database on the mounted volume. If you store your policies in a PostgreSQL database, follow the instructions and use the scripts provided in *this GitHub repository* to update your policy database.
   - For proper communication between containers, create a Docker network using a command such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

   ```
   $ docker container stop <pap_old>
   $ docker run --network=<network_name> --name <pap_new> \
       -p 443:1443 -d --env-file ~/.pingidentity/config \
       --volumes-from <pap_old> \
       --env PING_H2_FILE=/opt/out/Symphonic \
       pingidentity/pingauthorizepap:<TAG>
   ```

   The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see Docker Hub (*https://hub.docker.com/r/pingidentity/pingauthorizepap*).

   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 <pap_old>
$ docker run --network=<network_name> --name <pap_new> \
-p 443:1443 -d --env-file ~/.pingidentity/config \
--env PING_H2_FILE=/opt/out/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/pingauthorizepap:<TAG>
```

The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see Docker Hub (*https://hub.docker.com/r/pingidentity/pingauthorizepap*).

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 Policy Editor, complete the steps in *Upgrading the Trust Framework and policies* on page 136.

Manual upgrades

Upgrading PingAuthorize Server manually

Perform the following steps to upgrade a PingAuthorize server.

Steps

1. Download and unzip the new version of PingAuthorize Server in a location outside the existing server's installation.

   For these steps, assume the existing server installation is in `/opt/pingauthorize/PingAuthorize` and the new server version is extracted into `/home/stage/PingAuthorize`.
2. Provide a copy of the PingAuthorize license file for the version to which you are upgrading in the `/home/stage/PingAuthorize` 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 PingAuthorize Server.

   The **update** tool might prompt for confirmation on server configuration changes if it detects customization.

   Example:

   ```
   /home/stage/PingAuthorize/update --serverRoot /opt/pingauthorize/PingAuthorize
   ```

Reverting an update

After you've updated PingAuthorize 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 PingAuthorize 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/pingauthorize/PingAuthorize/revert-update
```

Upgrading the PingAuthorize Policy Editor manually

If you originally installed the PingAuthorize Policy Editor using .zip files, use this procedure to upgrade the Policy Editor when a new version is released.

Steps

1. In your current Policy Editor, complete the steps in *Backing up policies* on page 136.
2. Stop the Policy Editor.

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

3. Obtain and unzip the new version of the PingAuthorize Policy Editor in a location outside the existing Policy Editor's installation.

4. Prepare the existing policy database.

   > **Note:**
   >
   > The new server installation might require changes to the policy database structure.

   Choose from:

   - If you store your policies in the H2 policy database, copy the existing database. The server **`setup`** tool performs these upgrades and generates a new `configuration.xml` file.

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

To upgrade a database from	Run this command
8.1 and later versions	`$ cp /opt/pingauthorize/PingAuthorize-PAP-previous/Symphonic.mv.db opt/pingauthorize/PingAuthorize-PAP`
8.0 and earlier versions	`$ cp /opt/pingauthorize/PingAuthorize-PAP-previous/admin-point-application/db/Symphonic.mv.db opt/pingauthorize/PingAuthorize-PAP`

   - If you store your policies in a PostgreSQL database, follow the steps for *Upgrading a PostgreSQL policy database* on page 137.

5. Run **`setup`**.

   > **Note:**
   >
   > Updating PingAuthorize Server uses an **`update`** tool. PingAuthorize Policy Editor does not have this tool though. Instead of updating the Policy Editor in-place, you install the new Policy Editor.

   > **Warning:**
   >
   > The **`setup`** tool uses the default credentials to upgrade the H2 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 H2 policy database, or it might reset credentials to their default values. For more information, see *Manage policy database credentials* on page 250.

   Follow the instructions in one of the following topics:

   - *Installing the PingAuthorize Policy Editor interactively* on page 99
   - *Installing the PingAuthorize Policy Editor noninteractively* on page 102

6. Start the new Policy Editor.

   Follow the instructions in *Post-setup steps (manual installation)* on page 107.

7. In the new Policy Editor, complete the steps in *Upgrading the Trust Framework and policies* on page 136.

Policy-related upgrades

As part of the PingAuthorize upgrade process, you must upgrade specific Policy Editor components and dependencies, including policies, policy databases, and the Trust Framework.

See the following topics for instructions on upgrading Policy Editor components and dependencies:

- *Backing up policies* on page 136
- *Upgrading the Trust Framework and policies* on page 136
- *Upgrading a PostgreSQL policy database* on page 137

Backing up policies

Back up existing policies before upgrading the Policy Editor. Do this by exporting policy snapshots.

About this task

Back up policies manually as described below or rely on the automatic backups covered in *Policy database backups* on page 277.

Steps

1. Sign on to the Policy Editor 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.

   Result: 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**.

   Result: Your browser downloads the file.
6. Repeat for any additional branches that you want to back up.

Upgrading the Trust Framework and policies

PingAuthorize 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 Editor 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 Editor 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`.

   Example: 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 PingAuthorize Server installation.
7. Run the following **`dsconfig`** command to configure PingAuthorize Server to use the latest Trust Framework version.

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

Upgrading a PostgreSQL policy database

To upgrade an existing PostgreSQL policy database from version 9.0.X.X to version 9.1.X.X, use the provided upgrade SQL scripts as directed.

About this task

These instructions apply only to non-Docker deployments of PingAuthorize versions 9.0.X.X (EA and GA) and 9.1.X.X (EA and GA). Earlier versions don't support PostgreSQL, and later versions require the **`db-cli`** tool to upgrade PostgreSQL databases.

Steps

1. Prepare the Policy Editor by selecting one of the following options:
   Choose from:
   - If you haven't upgraded the Policy Editor, follow the steps for *Installing the PingAuthorize Policy Editor noninteractively* on page 102 to create and start up a new instance of the server at the target version.
   - If you have already upgraded the Policy Editor to the target version, run the **`start-server`** command, as follows:

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

   Result:

   After you execute the **`start-server`** command, the application checks the PostgreSQL database schema version against the version of the Policy Editor and provides the *locations of any necessary upgrade scripts*, as illustrated in the following example:

   ```
   The policy database at
    'jdbc:postgresql://<postgresql_host>:<postgresql_port>/<postgresql_db_name>'
    is older than this version of PingAuthorize (9.1.0.0). Please use the
    following scripts to upgrade the policy database before running start-
   server again:
   A) https://github.com/pingidentity/pingauthorize-contrib/blob/main/sql/
   postgresql/9.1-EA.sql
   B) https://github.com/pingidentity/pingauthorize-contrib/blob/main/sql/
   postgresql/9.1-GA.sql
   ```

2. Download and apply the upgrade scripts for the policy database schema versions between your current version and the target version, as indicated in the previous step.

   > **Important:**
   >
   > You must apply the scripts incrementally, not concurrently, and in sequence from oldest to newest.

   Example:

   For example, to upgrade from 9.0.X.X-GA to 9.1.X.X-GA, you must apply both the 9.1-EA and 9.1-GA upgrade scripts:

   ```
   $ wget https://raw.githubusercontent.com/pingidentity/pingauthorize-
   contrib/main/sql/postgresql/9.1-EA.sql
   $ psql --dbname=<postgresql_db_name> --file=9.1-EA.sql >/dev/null
   $ wget https://raw.githubusercontent.com/pingidentity/pingauthorize-
   contrib/main/sql/postgresql/9.1-GA.sql
   $ psql --dbname=<postgresql_db_name> --file=9.1-GA.sql >/dev/null
   ```

Result

After you apply the specified SQL scripts, the PostgreSQL database upgrade is complete. The Policy Editor should now start successfully using the target version of the PostgreSQL database. You can verify this by running the **`start-server`** command.

Uninstalling PingAuthorize

For manual installations, PingAuthorize Server provides an **`uninstall`** tool to remove its components from the system.

Steps

1. Go to the PingAuthorize 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.

   Example:

   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.

Next steps

To remove PingAuthorize Policy Editor, run **`stop-server`** and remove its installation directory.

PingAuthorize Integrations

Ping Identity provides the following API gateway integrations to enable you to use PingAuthorize for attribute-based access control and policy decisions with your API gateway:

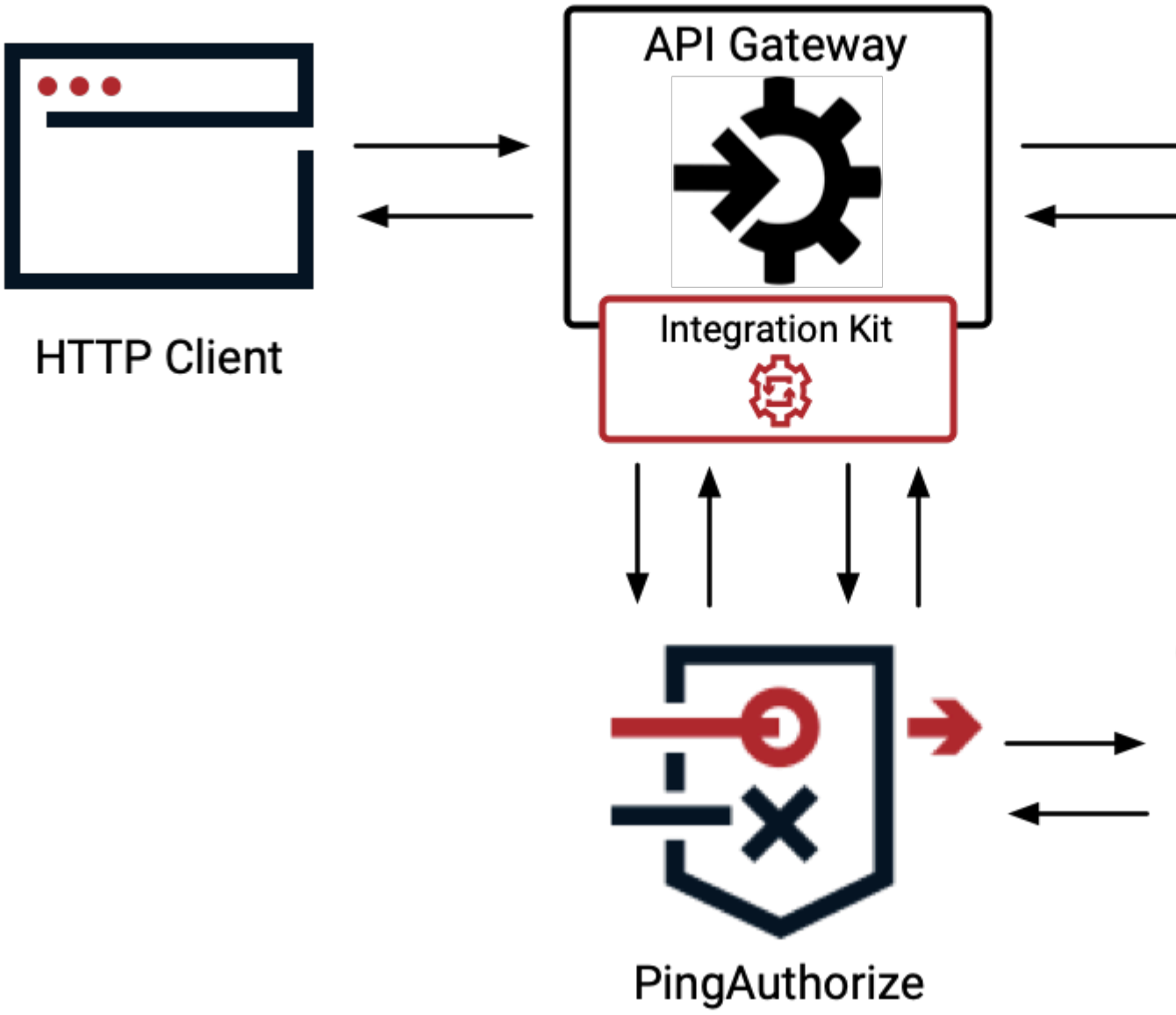
- *Kong API gateway integration* on page 139
- *MuleSoft API gateway integration* on page 151

Kong API gateway integration

Ping Identity provides the `ping-auth` Kong Gateway integration plugin, which enables PingAuthorize to be used for attribute-based access control and policy decisions.

Integration with Kong Gateway allows PingAuthorize to handle the complexities of attribute-based access control and dynamic authorization, making it easier for you to control access to your API resources. Instead of configuring policies multiple times, deploy the Kong Gateway integration once and manage your policy rules in PingAuthorize.

The following diagram explains how traffic flows through Kong Gateway and PingAuthorize.



1. The HTTP client sends an inbound request to Kong Gateway.
2. Kong Gateway sends a sideband request to PingAuthorize.
3. PingAuthorize Server evaluates the request against policies defined in the PingAuthorize Policy Editor and sends a permit or deny response to Kong Gateway.
4. Kong Gateway analyzes the response from PingAuthorize to determine whether the request should be allowed to the resource server—and if so, whether there should be any modification to the request. Should the request be denied, then PingAuthorize includes directives to influence how Kong Gateway responds to the HTTP client.
5. If the request is permitted, the resource server sends an outbound response to Kong Gateway.
6. Kong Gateway passes the response to PingAuthorize for processing.
7. PingAuthorize sends a response to Kong Gateway.
8. Kong Gateway processes the response from PingAuthorize. This includes directives for how to modify the response to the HTTP client, if any modifications should be made.

Note:

The following notes are important to consider when using the `ping-auth` Kong Gateway integration plugin for PingAuthorize:

Mutual TLS (mTLS)

This plugin supports client certificate authentication using mTLS; however this feature requires using the `mtls-auth` plugin (only available in the enterprise edition of Kong) in conjunction with `ping-auth`. For more information, see the *Kong mTLS documentation*. When configured, `mtls-auth` uses the mTLS process to retrieve the client certificate, which allows `ping-auth` to provide the certificate in the `client_certificate` field of the sideband requests.

Transfer-Encoding

Because of an outstanding defect in Kong, `ping-auth` is unable to support the Transfer-Encoding header, regardless of the value.

Logging limit

Because of OpenResty's log level limit, log messages are limited to 2048 bytes by default, which is less than the size of many requests and responses. For more information, see the *OpenResty reference documentation*.

Preparing PingAuthorize for Kong Gateway integration

For Kong Gateway to use PingAuthorize as an external authorization policy runtime service, you must prepare PingAuthorize to receive authorization requests from Kong Gateway.

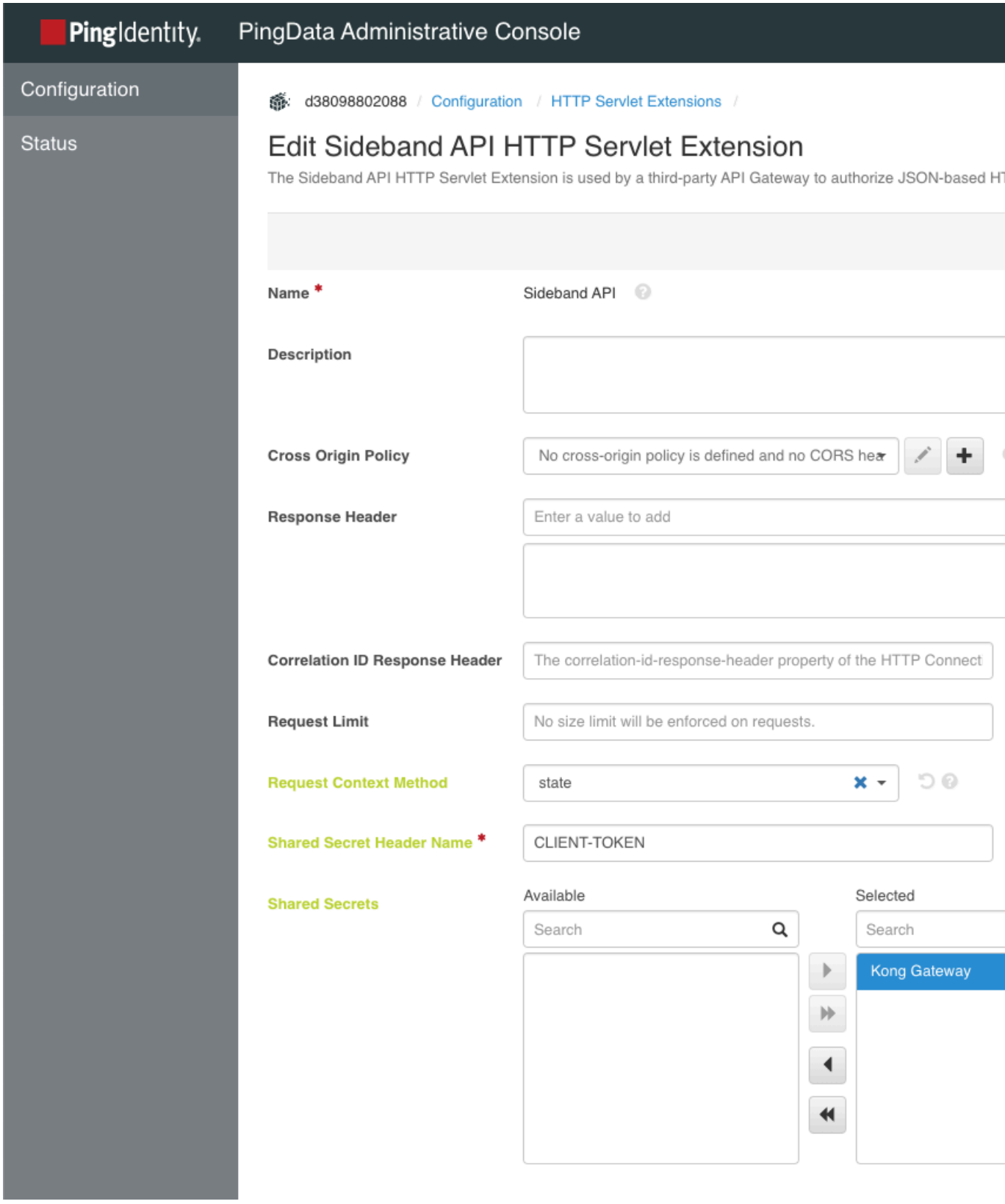
Before you begin

- Install and start Kong Gateway. For more information, see the *Kong Gateway* documentation.
- Install and start PingAuthorize. For more information, see *Installing PingAuthorize* on page 77.

Steps

1. In the PingAuthorize admin console, go to **Configuration** # **HTTP Servlet Extensions** # **Sideband API**.
2. In the **Request Context Method** list, select **State**.
3. In the **Shared Secret Header Name** field, modify the value to `CLIENT-TOKEN`.
4. Next to the **Selected** table for **Shared Secrets**, click the **+** icon to create a new shared secret.
5. In the modal dialog, create a suitably long shared secret value and click **Save To PingAuthorize Server Cluster**.

6. At the top of the **Edit Sideband API HTTP Servlet Extension** page, click **Save**.



Setting up Kong Gateway

Download, install, and configure the `ping-auth` plugin to set up Kong Gateway with PingAuthorize.

About this task

To configure the `ping-auth` plugin in Kong to set up a connection between PingAuthorize and Kong Gateway:

Steps

1. Install the plugin by running the `luarocks install kong-plugin-ping-auth` command.

   See the *Kong Gateway plugin installation guide* for more information.
2. After installation, load the plugin into Kong by editing the `plugins = bundled,ping-auth` property in the `kong.conf` file.
3. Restart Kong Gateway.
4. To confirm loading, look for the debug-level message `Loading plugin: ping-auth` in Kong's `error.log`.

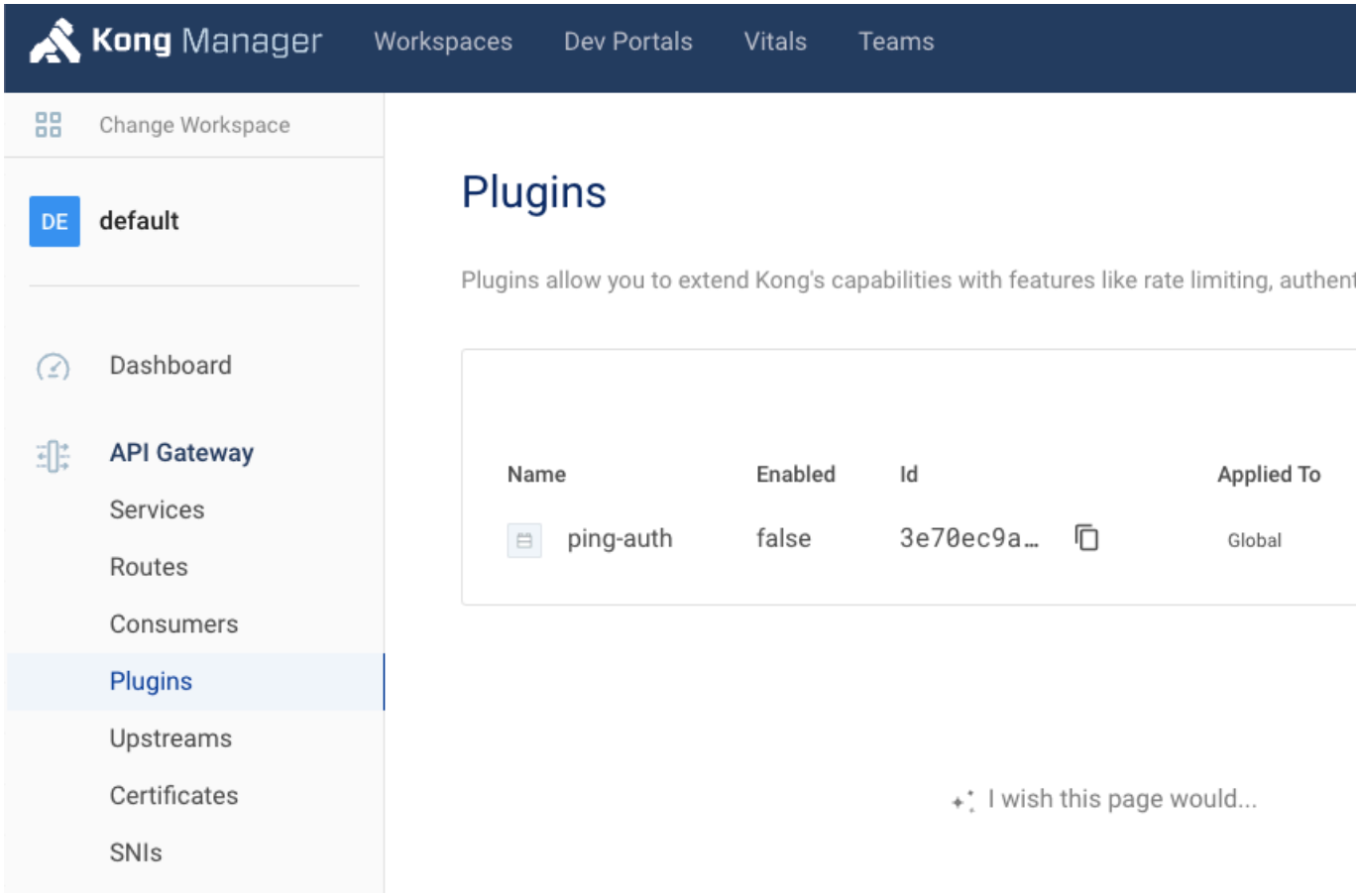
Next steps

- To complete Kong Gateway setup using Kong Manager, proceed to **Using the GUI**.
- To complete Kong Gateway setup using API requests, proceed to **Using the API**.

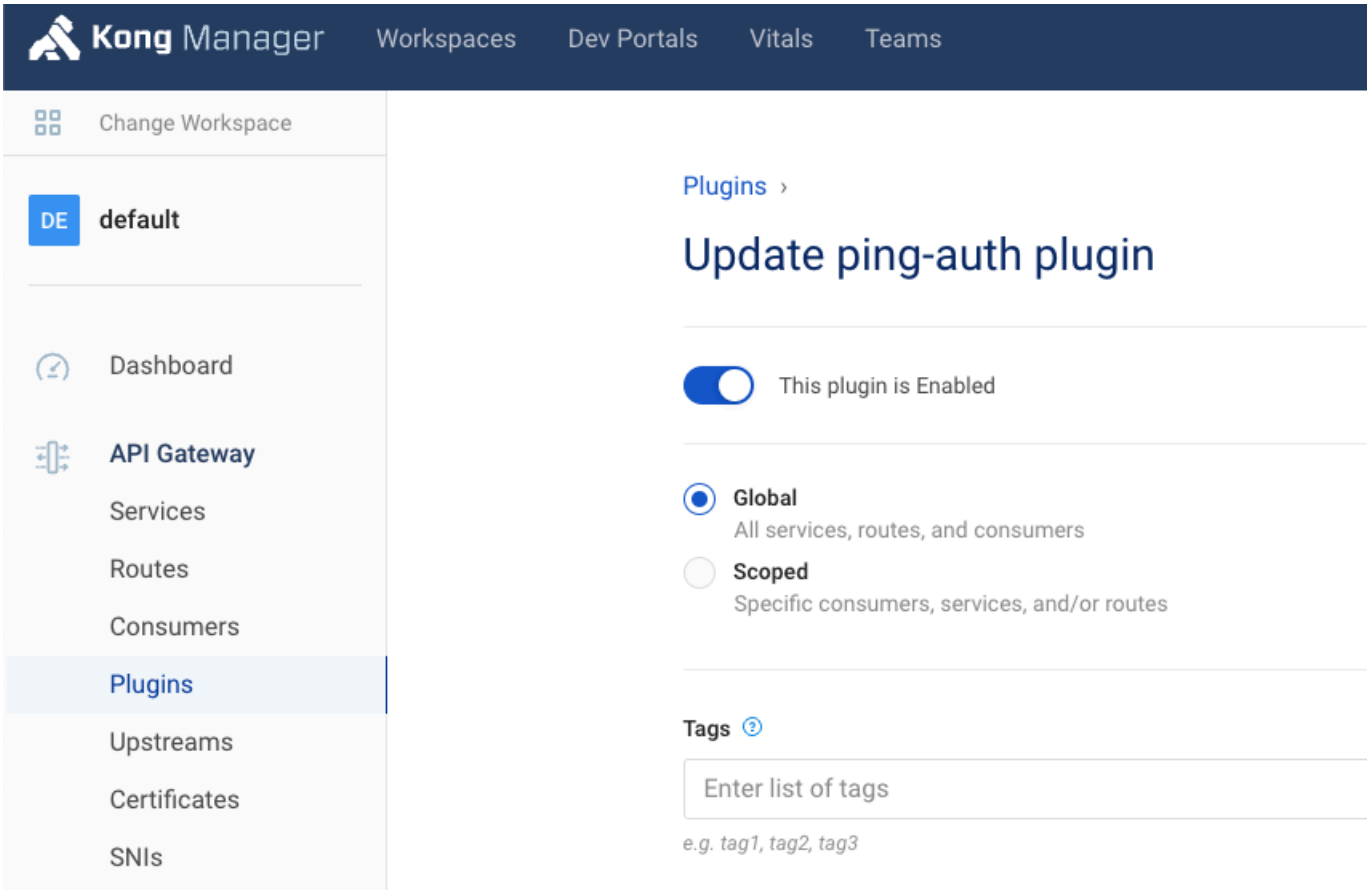
Setting up Kong Gateway using the GUI

Steps

1. In Kong Manager, select the **default** workspace and then click **Plugins**.



2. For the `ping-auth` plugin, click **Edit**, and then click the toggle to enable the plugin.



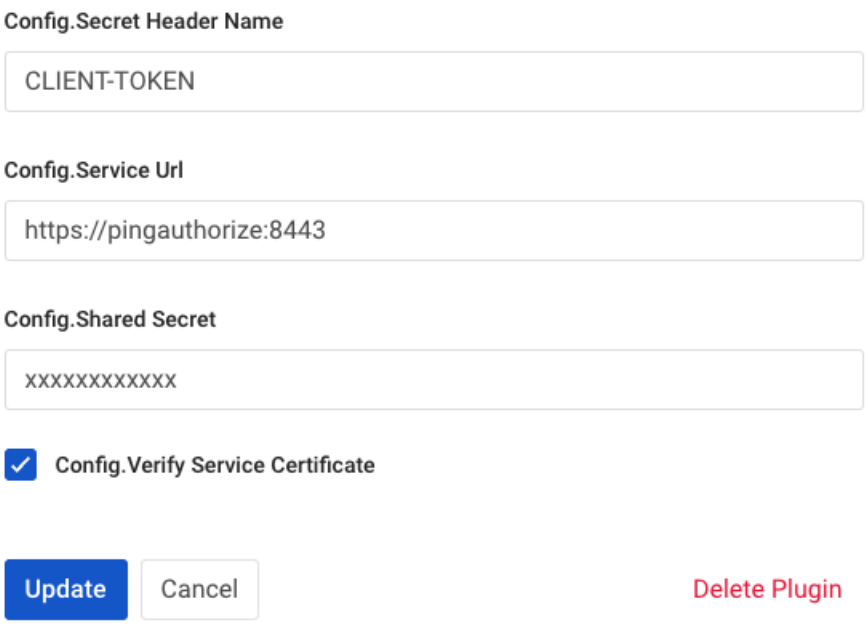
3. Optional: If you want to enable the plugin for specific consumers, services, or routes, click **Scoped**, and then enter **Service**, **Route**, and **Consumer** information as needed.

4. Connect Kong Gateway to PingAuthorize:
   a. Copy the PingAuthorize sideband client's shared secret.
   b. Enter the hostname of your PingAuthorize server and the port of the **HTTPS Connection Handler** into the **Config.Service URL** field.

      You can find this port number in the PingAuthorize Admin Console by going to **Configuration** # **System** # **Connection Handlers**.

      Example:

      For example, this field's value could be `https://pingauthorize:8443`.
   c. Paste the shared secret into the **Config.Shared Secret** field in Kong Manager.
   d. Ensure the **Config.Secret Header Name** value in Kong Manager matches the secret header name configured for the Sideband API Servlet Extension in PingAuthorize.



5. Optional: Configure the rest of the optional fields in Kong Manager or the API.

Option	API Field Name	Description
Config.Connection KeepAlive Ms	**`connection_keepAlive_ms`**	The duration to keep the connection alive for reuse. The default is `60000`.
Config.Connection Timeout Ms	**`connection_timeout_ms`**	The duration to wait before the connection times out. The default is `10000`.
Config.Enable Debug Logging	**`enable_debug_logging`**	Controls if requests and responses are logged at the debug level. The default is `false`. For log messages to show in `error.log`, you must set `log_level = debug` in `kong.conf`.

Option	API Field Name	Description
Config.Verify Service Certificate	**`verify_service_certificate`**	Controls whether the service certificate is verified. This is intended for testing purposes and the default is `true`.

6. Click **Update**, and then click **Update Plugin**.

Result

Kong Gateway is now configured to work with PingAuthorize.

Setting up Kong Gateway using the API

Steps

1. Send the following in a `POST` request to https://*<KONG_URL>*/plugins:

```
{
  "name": "ping-auth",
  "enabled": true,
  "config": {
    "service_url": "https://<PingAuthorize Server hostname>:<HTTPS
 Connection Handler port>/",
    "shared_secret": "<shared secret>",
    "secret_header_name": "<shared secret header name>"
  }
}
```

Note:

See the following list for more information about the required fields for the previous API request:

`service_url`

The full URL of the Ping policy provider. This should not contain `/sideband` in the path.

`shared_secret`

The shared secret value to authenticate this plugin to the policy provider.

`secret_header_name`

The header name in which the shared secret is provided.

You can provide additional configuration in accordance with the Kong API specification. For more information, see the *Kong documentation*.

2. Optional: Configure the rest of the optional fields through the API.

Option	API Field Name	Description
Config.Connection KeepAlive Ms	**`connection_keepAlive_ms`**	The duration to keep the connection alive for reuse. The default is `60000`.
Config.Connection Timeout Ms	**`connection_timeout_ms`**	The duration to wait before the connection times out. The default is `10000`.

Option	API Field Name	Description
Config.Enable Debug Logging	**`enable_debug_logging`**	Controls if requests and responses are logged at the debug level. The default is `false`. For log messages to show in `error.log`, you must set `log_level = debug` in `kong.conf`.
Config.Verify Service Certificate	**`verify_service_certificate`**	Controls whether the service certificate is verified. This is intended for testing purposes and the default is `true`.

Result

Kong Gateway is now configured to work with PingAuthorize.

Troubleshooting the Kong Gateway integration

Consult the following sections to troubleshoot issues with the Kong Gateway integration with PingAuthorize:

- *Troubleshooting API client HTTP 5xx errors* on page 148
- *API client HTTP 4xx errors* on page 149
- *Enabling error logging in Kong Gateway* on page 150
- *Enabling debug logging for the Kong Gateway plugin* on page 150

Troubleshooting API client HTTP 5xx errors

About this task

Kong Gateway might return `HTTP 502` when there is misconfiguration or miscommunication between the Ping Identity plugin for Kong Gateway and PingAuthorize Server.

Trouble:

The plugin for Kong Gateway logs warning messages to the Kong Gateway error log when it encounters problems communicating with PingAuthorize.

For more information, see *Enabling error logging in Kong Gateway* on page 150.

Steps

1. Check the `ping-auth` shared secret value in Kong Gateway to confirm it matches your PingAuthorize environment.

   Example:

   If the `ping-auth` **Config.Shared Secret** value doesn't match the PingAuthorize sideband client's shared secret value, the Kong error log message might indicate that the plugin received an `HTTP 401` error from PingAuthorize, which gets translated to a 5xx error sent to the API client. For example:

   ```
   2022/03/28 16:19:49 [warn] 78#0: *85187 [lua] network_handler.lua:145:
    is_failed_request(): [ping-auth] Sideband request denied with status
    code 401: The Gateway Token is invalid
   ```

   a. If there is a shared secret mismatch, go to **Configuration** # **Web Services and Applications** # **Sideband API Shared Secrets** in the PingAuthorize Admin Console.
   b. Update the shared secret value for PingAuthorize.
   c. Copy the value to the **Config.Shared Secret** field in the Kong Gateway `ping-auth` plugin configuration.
2. Check the `ping-auth` **Config.Service URL** value in Kong Gateway to confirm that it matches your PingAuthorize environment.

   Example:

   If the **Config.Service URL** value doesn't contain the hostname and HTTPS Connection Handler port configured for your PingAuthorize server, the Kong error log message might indicate that the plugin received an invalid response from the server. For example:

   ```
   2022/03/28 16:19:49 [error] 78#0: *90929 [lua] access.lua:114:
    handle_response(): [ping-auth] Unable to parse JSON body returned from
    policy provider. Error: Expected value but found T_END at character 1
   ```

   a. If necessary, confirm that the values entered in the **Config.Service Url** field of the `ping-auth` plugin in Kong Gateway correspond to the hostname and HTTPS Connection Handler port of your PingAuthorize server.

      You can find this port number in the PingAuthorize Admin Console by going to **Configuration** # **System** # **Connection Handlers**.
   b. Update any mismatched values in **Config.Service Url**.

API client HTTP 4xx errors

The API gateway could return 4xx errors to API clients in these situations:

- PingAuthorize cannot match an API client's request to any of the base paths configured for a sideband API endpoint.
- The API client's request cannot be authenticated for a sideband API endpoint.

Tip:

For more information, see *Diagnostic and decision data* on page 397.

Enabling error logging in Kong Gateway

Steps

1. To view error log messages, configure Kong Gateway error logging.

   For more information on log levels, see the Kong Gateway *Logging Reference* documentation.

   Example:

   For example, in a Docker environment, you can set the environment variable **`KONG_PROXY_ERROR_LOG`** to `/dev/stderr` to send the error log to the container console.
2. View the Kong Gateway error log.

   Example:

   For example, in a Docker deployment, you can use the `docker-compose logs kong --follow` command.

Enabling debug logging for the Kong Gateway plugin

About this task

Ping Identity Support might ask you to enable debug logging for the Kong Gateway integration kit. Changing these settings logs the full authorization request and response between the `ping-auth` plugin in Kong Gateway and PingAuthorize.

> **CAUTION:**
>
> This could log sensitive and personally identifiable information (PII). Enable debug logging only when troubleshooting and disable it afterward.

Steps

1. *Enable error logging* in Kong Gateway.
2. To view debug messages, configure Kong error log verbosity.

   For more information, see the Kong Gateway *Logging Reference* documentation.

   Example:

   For example, in a Docker environment, you can set the environment variable **`KONG_LOG_LEVEL`** to `debug` to set the verbosity.
3. To enable debug logging, edit settings for the `ping-auth` plugin and select the **Config.Enable Debug Logging** check box.
4. View the Kong Gateway error log.

   Example:

   For example, when depoloying Docker, you can use the `docker-compose logs kong --follow` command.
5. Look for messages containing `ping-auth`.

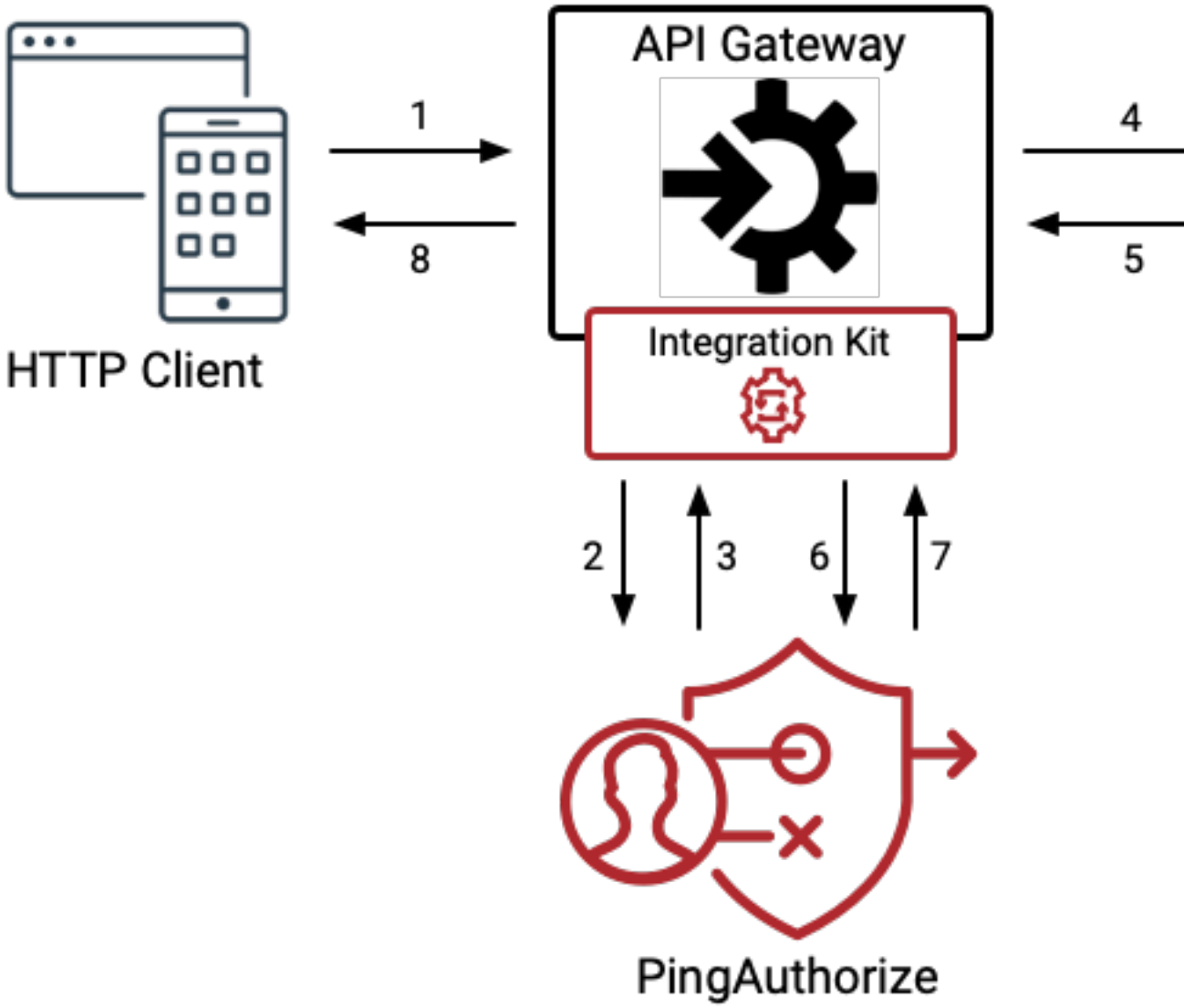
   Example:

   A typical log message looks like: `[ping-auth] Sending sideband request to policy provider`.

MuleSoft API gateway integration

Learn how to enable fine-grained access control through the MuleSoft API Gateway by deploying the PingAuthorize API integration kit and connecting to the Sideband API.

Ping Identity provides a custom MuleSoft policy to enable this configuration.



The custom MuleSoft policy acts as the sideband adapter, allowing MuleSoft to be used as the API gateway as follows:

1. The client sends an incoming request to MuleSoft.
2. The custom MuleSoft policy passes the incoming request to PingAuthorize Server.
3. PingAuthorize Server determines whether to permit or deny the request based on policies defined in the PingAuthorize Policy Editor (not to be confused with MuleSoft policies). The server also performs any desired request modifications.
4. If the request is permitted, MuleSoft makes the request to the backend resource.

5. MuleSoft receives a response from the backend resource.
6. The custom MuleSoft policy makes a second API call to pass response information to PingAuthorize Server.
7. PingAuthorize Server determines whether to permit or deny based on the backend response. Before the server returns the request to MuleSoft, it also modifies the request based on policies defined in PingAuthorize.
8. MuleSoft sends the response to the client.

Deploying the custom MuleSoft policy for PingAuthorize

Before you begin

You must:

- Have the correct MuleSoft version.

  The custom policy supports MuleSoft 4.3.0. If you are using any other version, contact Ping Identity support.
- Install and configure PingAuthorize software.

  See the *PingAuthorize installation information* for your environment.
- Download the *MuleSoft Integration Kit for PingAuthorize*, which contains the custom MuleSoft policy.
- Create a sideband adapter shared secret.

  Sideband adapters like the custom MuleSoft policy use a shared secret header to authorize against PingAuthorize. For information, see *Creating a shared secret* on page 181.

  > **Note:**
  >
  > Make sure you record the shared secret value. You need it to configure the MuleSoft policy.

- Configure the sideband adapter request context.

  For more information, see *Request context configuration* on page 189. Complete the section titled Request context using the state field.
- Install Apache Maven.

About this task

To begin integrating PingAuthorize with MuleSoft 4.3.0, deploy the custom MuleSoft policy. The MuleSoft policy package has a `.zip` archive that contains the policy files.

Steps

1. Extract the policy files to create a project folder.
2. Edit the `pom.xml` file to enter your organization's `groupID`.

   Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee</groupId>

    <artifactId>PingAuthorize</artifactId>
    <version>0.4.0</version>
```

```
    <name>PingAuthorize</name>
    <description>PingAuthorize sideband policy for Mule 4.X APIs deployed
 on Mule Cloudhub from Ping Identity</description>
```

3. From the command line in your project folder, run the following command to package the PingAuthorize policy and create a deployable `.jar` file.

   ```
   > mvn clean install
   ```

   > **Note:**
   >
   > You must have a MuleSoft Enterprise Repository license to compile the policy. For more information, see **Configure Maven to Access MuleSoft Enterprise Repository** in *Maven Reference*.

4. Upload the PingAuthorize policy to **Exchange**.

   For more information, see the instructions in *Deploying a Policy Created Using the Maven Archetype*.

Result

The custom MuleSoft policy is now available to your APIs. For more information, see *Applying the custom MuleSoft policy for PingAuthorize* on page 153.

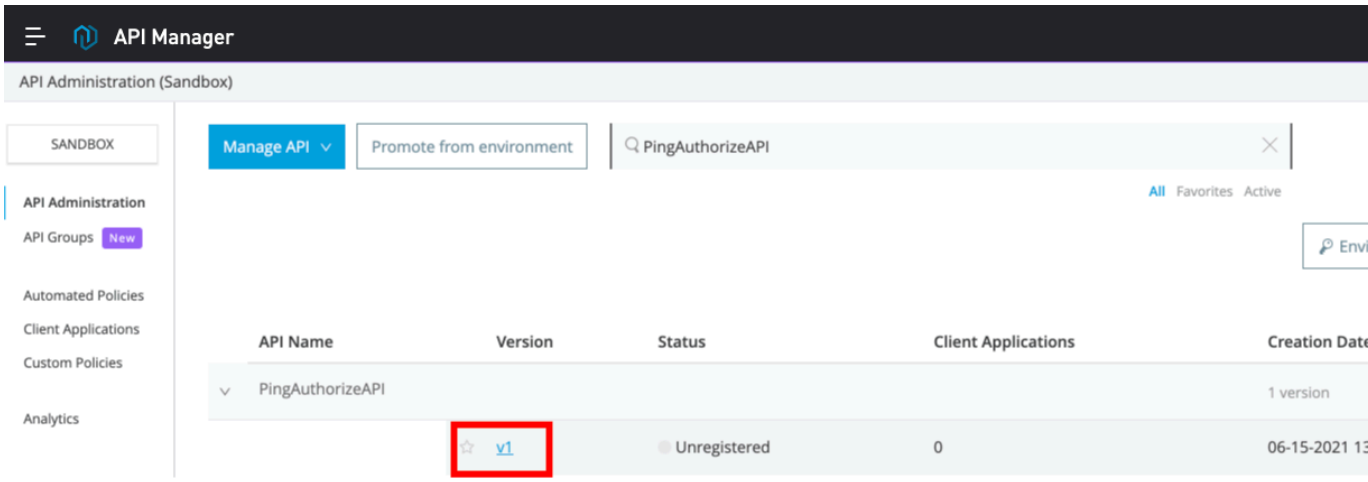
Applying the custom MuleSoft policy for PingAuthorize

About this task

The PingAuthorize policy supports HTTP APIs configured with the `Endpoint with proxy` or `Basic Endpoint` options.

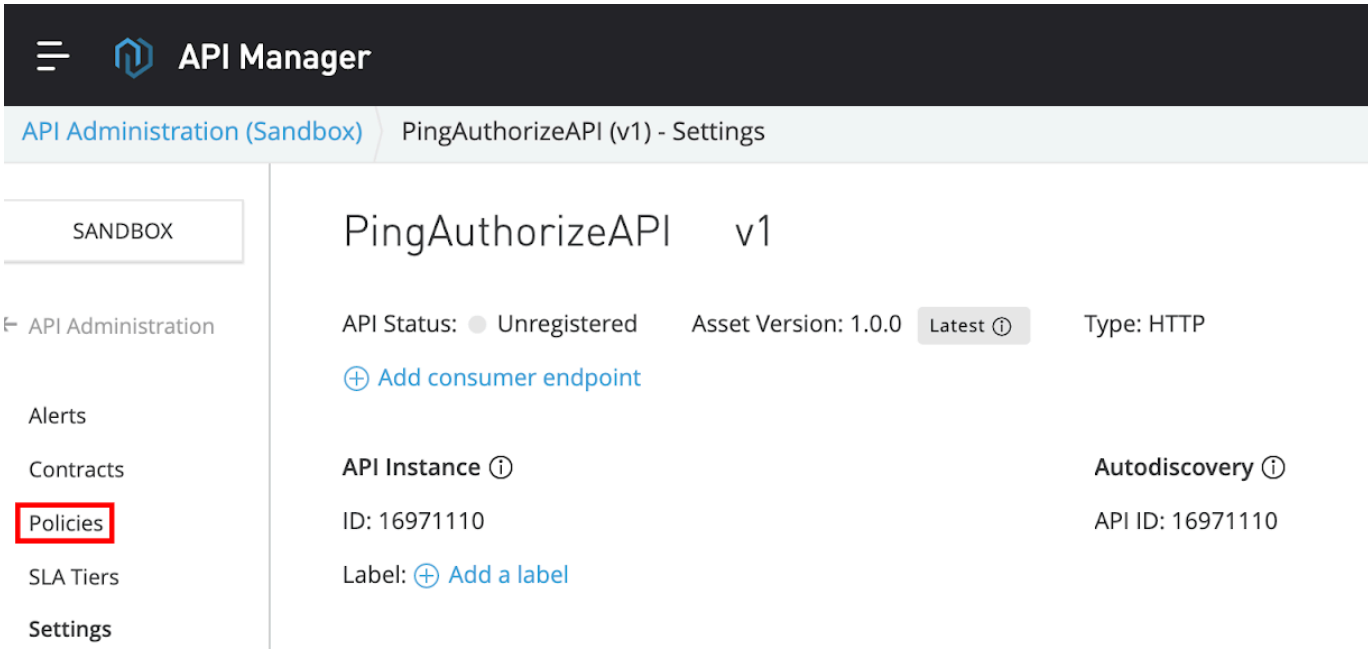
Steps

1. Sign on to your MuleSoft Anypoint account.
2. Go to the API manager, expand the API to which you want to attach the PingAuthorize policy, and click **Version**.

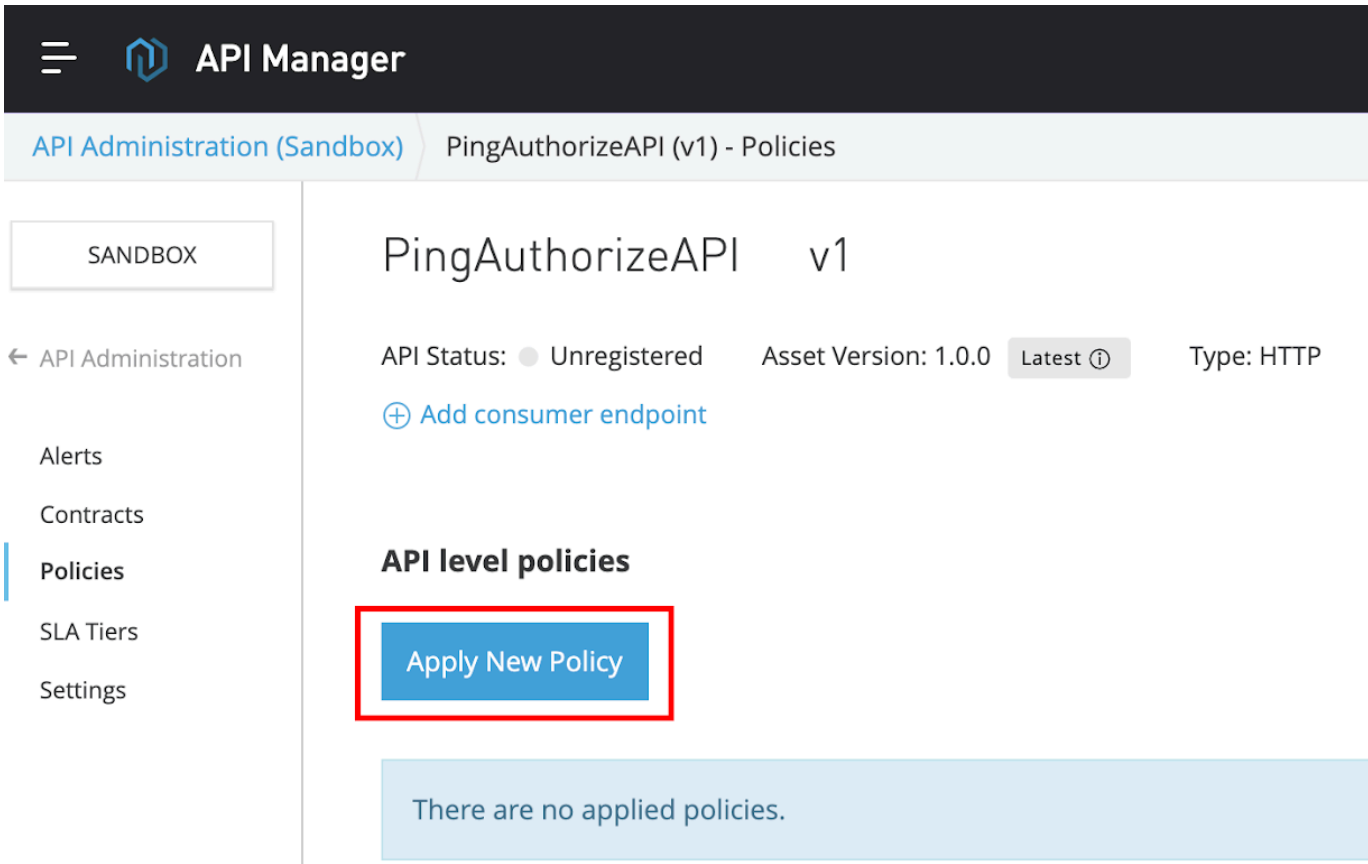


3. In the left navigation pane, click **Policies**.

   The **Policies** page supports applying the PingAuthorize policy to the API.

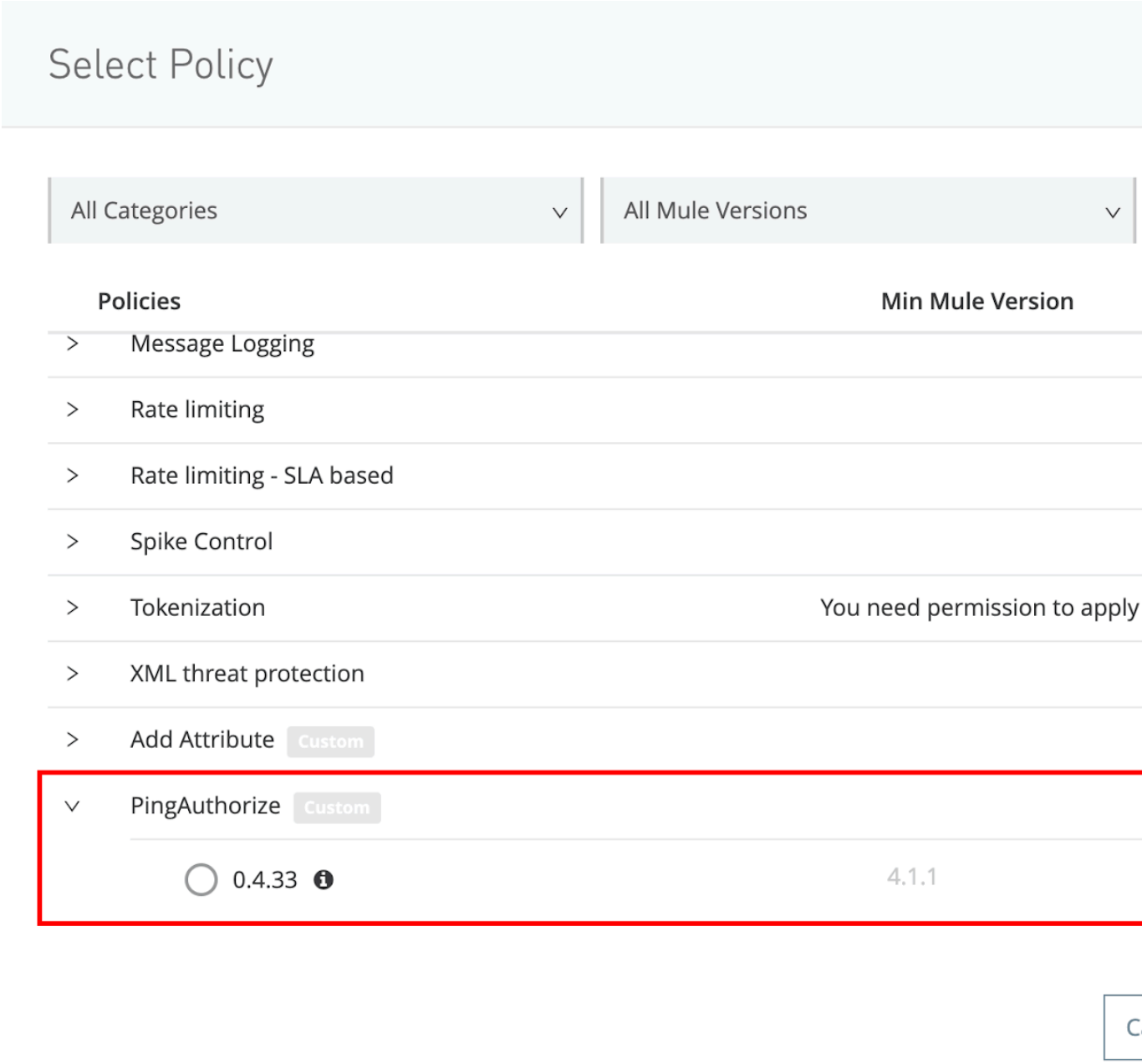


4. Click **Apply New Policy**.



Result: The **Select Policy** window opens.

5. In the **Select Policy** window, select the PingAuthorize policy and current version. Click **Configure Policy**.



6. On the **Apply Policy** page, enter the following values:
   a. In the **PAZ Token** field, enter the sideband adapter shared secret generated as part of the prerequisites in *Deploying the custom MuleSoft policy for PingAuthorize* on page 152
   b. In the **PAZ Host** field, enter the PingAuthorize host and port.

      > **Note:**
      >
      > Do not include the connection scheme (http:// or https://).

   c. Select the **Enable SSL** check box for a secure HTTPS connection between MuleSoft and PingAuthorize.
   d. Select the **Allow self-signed certificate** check box to enable MuleSoft to accept a self-signed certificate from PingAuthorize.

      For information about configuring PingAuthorize to use trusted certificates, see *Importing signed and trusted certificates* on page 337.
   e. Select an access token type:

      Choose from:

      - Use Authorization Header.

        Indicates that the authorization header of an incoming request should be passed to PingAuthorize and used to authorize the client.
      - Use hard-coded parsed access token.

        Allows configuration of an access token that will be used for every request. Use this only for testing purposes.
      - Use parsed access token.

        Allows configuration of a DataWeave expression for retrieving a parsed access token from the Mule message. When you use MuleSoft's OAuth 2.0 Token Enforcement policies to obtain a parsed access token, use the expression `#[authentication.properties.userProperties]`. For more information, see *DataWeave Language*.
   f. Optional: Configure the **Connection Timeout** and **Read Timeout**.

      Timeouts govern the behavior of the API gateway when it cannot connect to PingAuthorize or the response from PingAuthorize is delayed.

Timeout parameter	Description
Connection Timeout	Governs the time the API gateway waits to establish a connection with PingAuthorize, following which it sends the client request to the backend server.

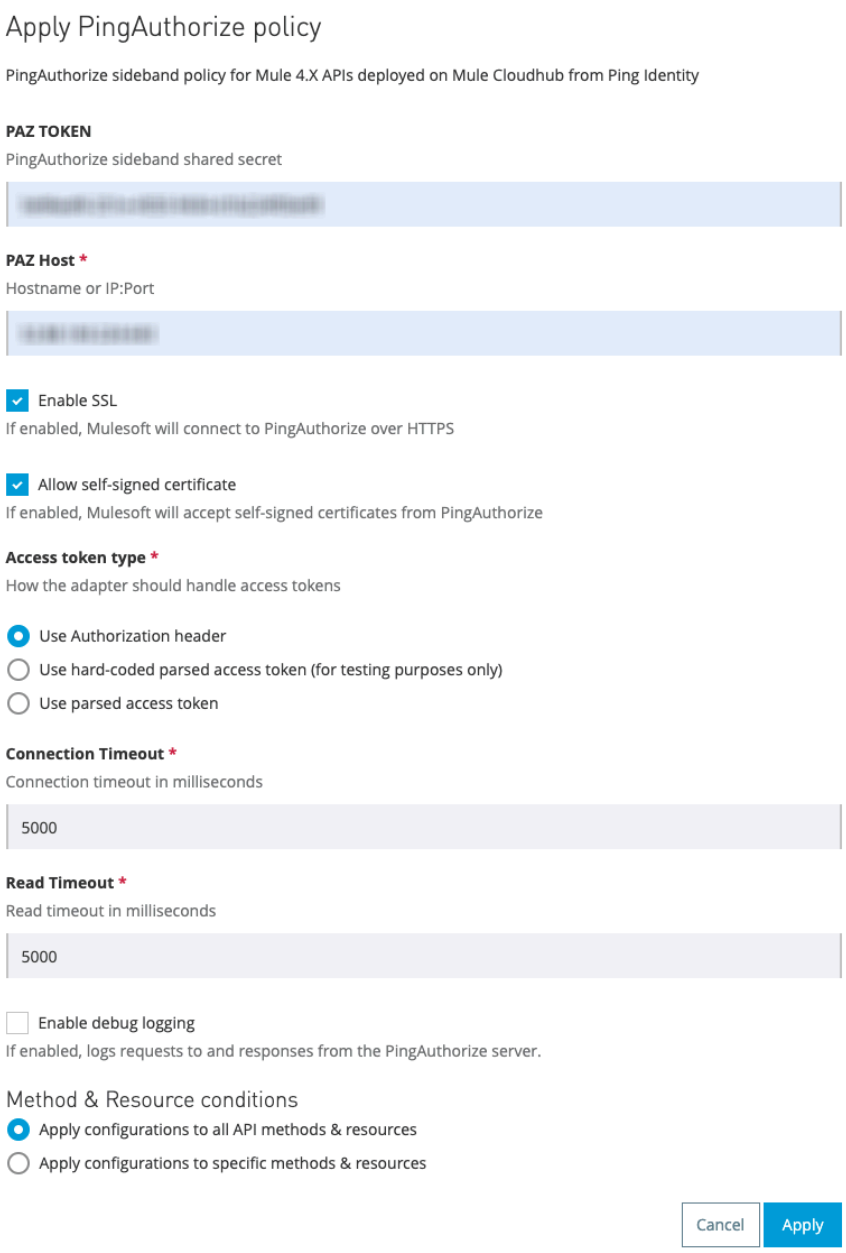
Timeout parameter	Description
Read Timeout	Governs the time the API Gateway waits for PingAuthorize's response before sending the request to the backend server.

Note:

The default value is 5000 milliseconds (5 seconds). It's good practice to configure a small value to limit the delay in case PingAuthorize isn't reachable or is unresponsive.

g. Optional: Select the **Enable debug logging** check box to see requests sent to PingAuthorize Server along with responses.
h. Optional: Configure **Methods & Resource Conditions**.

   See *Resource-Level Policies* for more information.



Next steps

If there are any changes to PingAuthorize endpoints, repeat the process explained in step 6 and re-deploy the configuration.

PingAuthorize Server Administration Guide

PingAuthorize Server includes the runtime policy decision service and multiple integration capabilities:

- Authorization policy decision APIs
- API security gateway and sideband API
- SCIM service

Running PingAuthorize

For manual software installations of PingAuthorize Server and the PingAuthorize Policy Editor, you can launch the applications from the CLI.

Starting PingAuthorize Server

To start PingAuthorize Server in a Unix/Linux computing environment, use the **`bin/start-server`** CLI command. On Windows, use the **`bat/start-server.bat`** command.

Steps

1. In a terminal window, enter go to the directory where you have installed PingAuthorize Server.
2. Run the command for your operating system.

Operating System	Command
Unix/Linux	**`bin/start-server`**
Windows	**`bat/start-server.bat`**

Running PingAuthorize Server as a foreground process

Run or stop PingAuthorize Server as a foreground process in Unix/Linux computing environments through the CLI.

Steps

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

Starting PingAuthorize Server at boot time (Unix/Linux)

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

About this task

PingAuthorize 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 PingAuthorize Server to start automatically when the system boots, complete one of the following tasks:
  Choose from:
  - 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-authorize.service \
```

```
  --userName pingauthorize
```

In this example, `pingauthorize` represents the username assigned to PingAuthorize 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-authorize.service` configuration file to the `/etc/systemd/` system directory as shown.

```
cp ping-authorize.service /etc/systemd/
```

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

```
$ systemctl daemon-reload
```

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

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

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

```
$ systemctl enable ping-authorize.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 PingAuthorize Server at boot time (Windows)

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

About this task

PingAuthorize 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 PingAuthorize Server as a Windows service

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

About this task

Note:

The following options are not supported when PingAuthorize 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 PingAuthorize 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 PingAuthorize Server:
   Choose from:
   - The **Windows Services Control Panel**
   - The **`bat/ start-server.bat`** command

Running multiple service instances
You can run multiple instances of PingAuthorize 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 `pingauthorize1`.
3. **Save** the `wrapper-product.conf` file.
4. Register PingAuthorize Server as a service. For more information, see *Registering PingAuthorize Server as a Windows service* on page 160.
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 PingAuthorize Server, run the `PingAuthorize/uninstall.bat` script. For more information, see *Uninstalling PingAuthorize* on page 138.

Log files for Windows services
You can configure the log files generated by PingAuthorize 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 PingAuthorize Policy Editor

For a manual software installation, use the **`start-server`** CLI command to start the Policy Editor. Also, you can use environment variables to override configuration variables at startup.

To start PingAuthorize Policy Editor, 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 Editor 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 Editor, define one or more of the environment variables, and restart the Policy Editor.

Environment variables you can use to override configuration variables

The following table lists the environment variables that you can define, sorted based on expected frequency of use with related variables grouped together.

Environment variable	Example value	Description
`PING_EXTERNAL_BASE_URL`	https://pa.example.com:9443	The Policy Editor hostname and port. PingAuthorize 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 Editor HTTPS port. The server binds to this listen port.
`PING_KEYSTORE_TYPE`	JKS	The Policy Editor's key store type. Valid values include `JKS` and `PKCS12`.
`PING_KEYSTORE_PATH`	/path/to/keystore.jks	The path to the Policy Editor's key store.

Environment variable	Example value	Description
KEYSTORE_PIN_FILE	/path/to/keystore.pin	The path to the Policy Editor's key store PIN file. When present, this environment variable takes precedence over **PING_KEYSTORE_PASSWORD** when validating and presenting the server certificate. The key store PIN value itself does not persist to the `configuration.yml` file and is not visible on the command-line. For a more complete example, see the **Demo mode (custom SSL certificate)** tab of *Installing the PingAuthorize Policy Editor noninteractively* on page 102.
PING_KEYSTORE_PASSWORD	password1234	The Policy Editor's key store password.
PING_CERT_ALIAS	server-cert	The alias for the Policy Editor's server certificate.
PING_SHARED_SECRET	pingauthorize	The Policy Editor's shared secret, which PingAuthorize Server needs to make policy requests to the Policy Editor.
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 Editor is set up in OIDC mode.
PING_OIDC_TLS_VALIDATION	NONE	The OpenID Connect (OIDC) Transport Layer Security (TLS) validation setting. Set to `NONE` to configure the Policy Editor to accept self-signed SSL certificates from the OpenID Connect provider and skip hostname verification. Used when the Policy Editor is set up in OIDC mode. For non-production use only.
PING_CLIENT_ID	8cb9f2c9-c366-47e0-9560-db2132b2d813	The Policy Editor's client ID with the OpenID Connect provider. Used when the Policy Editor 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 Editor 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.
PING_DB_ADMIN_USERNAME	db_admin	The username the **setup** tool uses when upgrading the policy database.

Environment variable	Example value	Description
PING_DB_ADMIN_PASSWORD	$3cr3T	The password the **setup** tool uses when upgrading the policy database.
PING_OPTIONS_FILE	/path/to/options.yml	The path to an `options.yml` file to use with the Policy Editor's setup tool.
PING_ADMIN_PORT	9444	The admin port where the H2 database backup endpoint is available. The policy administration point, or PAP, uses this endpoint to back up the H2 database, which stores your Trust Framework, policies, commit history, and other data. Related environment variables: **PING_BACKUP_SCHEDULE**, **PING_H2_BACKUP_DIR**
PING_BACKUP_SCHEDULE	0 0 0 * * ?	The periodic database backup schedule for the Policy Editor (also known as the PAP) in the form of a **cron** expression. **Note:** The PAP evaluates the expression against the system timezone. For the PingAuthorize Docker images, the default timezone is UTC. The default is `0 0 0 * * ?`, which is midnight every day. For more information, see *Quartz 2.3.0 cron format*. Related environment variables: **PING_ADMIN_PORT**, **PING_H2_BACKUP_DIR**
PING_H2_BACKUP_DIR	`/opt/out/backup`	The directory in which to place the H2 database backup files. The default is **SERVER_ROOT**`/policy-backup`. Related environment variables: **PING_ADMIN_PORT**, **PING_BACKUP_SCHEDULE**
PING_ENABLE_API_HTTP_CACHE	false	Controls the API *HTTP caching* on page 257 feature for the run-time instance of the server. APIs are cached by default. Provide this environment variable at run time and set it to `false` to disable API HTTP caching for that server instance.

Example: Use an existing SSL certificate for HTTPS connections

This example shows how to provide the environment variables necessary for the Policy Editor to present a different SSL certificate than the one configured during **`setup`**:

```
env PING_CERT_ALIAS=<certificate-nickname> \
PING_KEYSTORE_PATH=<path-to-keystore-file> \
PING_KEYSTORE_TYPE=<PKCS12-or-JKS> \
KEYSTORE_PIN_FILE=<path-to-keystore-pin-file> \
bin/start-server
```

Example: Override the configured HTTPS port

In this example, the Policy Editor 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 PingAuthorize 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/authorize-pe.log` and `logs/start-server.log` files.

Stopping PingAuthorize Server

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

Steps

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

  Note:

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

Stopping PingAuthorize Policy Editor

PingAuthorize Policy Editor provides a simple shutdown script to stop the system.

Steps

- To stop the PingAuthorize Policy Editor, run the **`bin/stop-server`** command.

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

Restarting PingAuthorize Server

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

About this task

Running this command is equivalent to shutting down PingAuthorize Server, exiting the Java virtual machine (JVM) session, and starting the server again.

Steps

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

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

About the API security gateway

When you configure PingAuthorize Server for the API gateway pattern, the server and gateway provide dynamic authorization management between a client and a REST API.

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

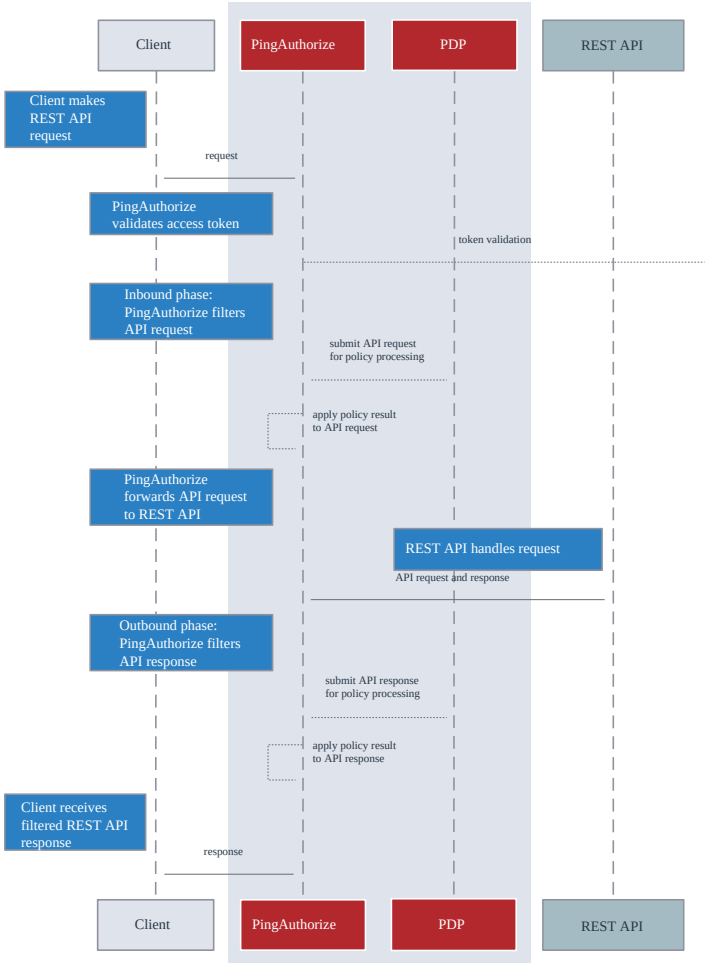
- *API gateway request and response flow* on page 166
- *Gateway configuration basics* on page 167
- *API security gateway authentication* on page 168
- *API security gateway policy requests* on page 169
- *API security gateway HTTP 1.1 support* on page 175
- *Gateway error templates* on page 176

API gateway request and response flow

Using the API gateway pattern, PingAuthorize 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 PingAuthorize 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, PingAuthorize Server forwards the request to the API server.
- Outbound phase – After PingAuthorize 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, PingAuthorize Server forwards the response to the client.



The API gateway supports only JSON requests and responses.

Gateway configuration basics

You can configure the API gateway architecture 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 PingAuthorize administrative console, or with the following example command.

```
PingAuthorize/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 PingAuthorize 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.

```
PingAuthorize/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

PingAuthorize/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*, located in the server's `docs/config-guide` directory.

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.

Property	Description
`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 397.

API gateway 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

Policy request attributes	Description	Type
`attributes`	Identifies additional attributes that do not correspond to a specific entity type in the PingAuthorize 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
`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

Attribute	Description	Type
`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
`active`	Indicates whether this access token is currently active, as determined by the access token validator.	Boolean
`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
`scope`	Identifies the list of scopes granted to this token.	Collection
`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

Attribute	Description	Type
`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
`subjectRegex`	Regular expression that must be matched by the subject field of the certificate to ensure that the certificate belongs to the requesting client.	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`.

API gateway 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 173 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 an allow list 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.

Gateway 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 PingAuthorize 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, PingAuthorize 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, PingAuthorize 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 *Sideband error templates* on page 191.

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.

Example: A policy `deny` results in a response like the following example.

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

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

About the Sideband API

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

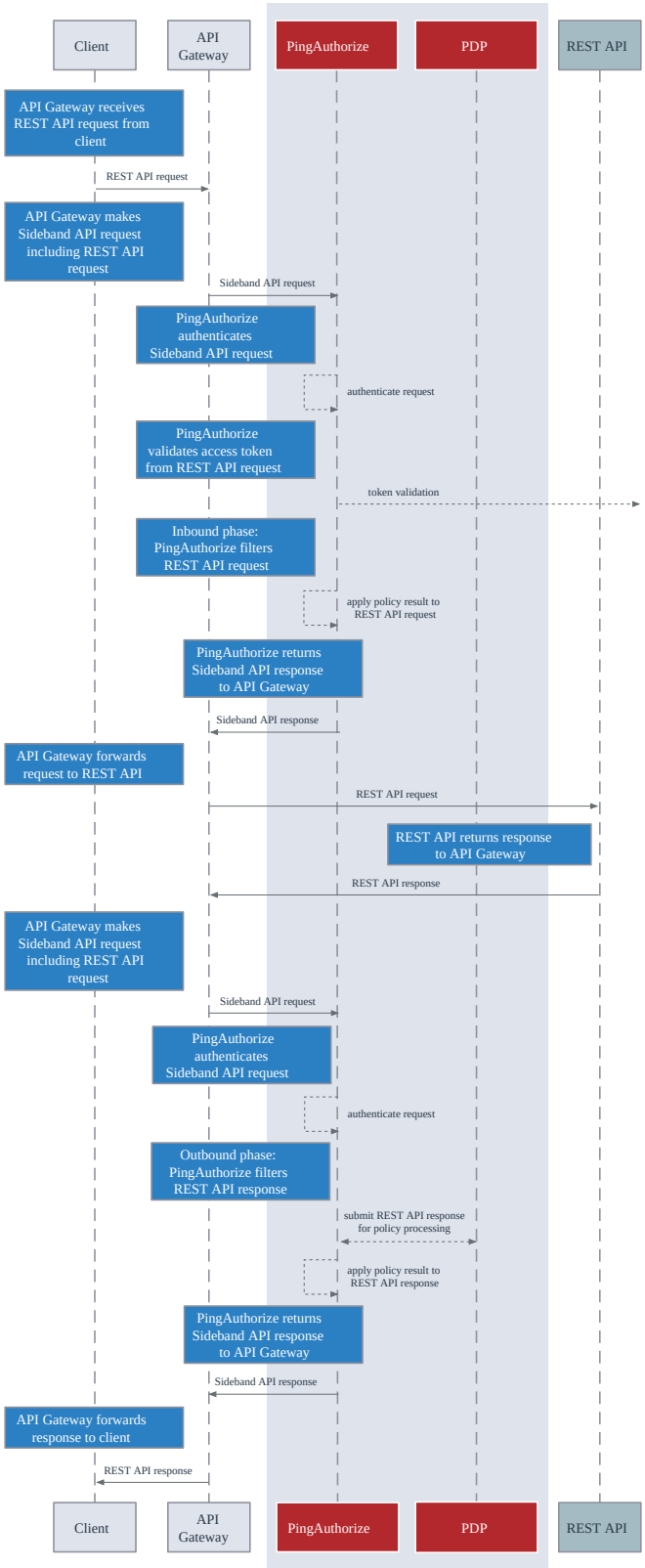
As a gateway, PingAuthorize Server functions as a reverse proxy that performs the following steps:

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

Using the Sideband API, you can configure the PingAuthorize 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 PingAuthorize Sideband API.

API gateway integration

Enable attribute-based access control through your API gateway by installing the PingAuthorize API integration plugin (where supported) and connecting to the Sideband API.



Processing steps

1. When 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.

   The API gateway forwards the response 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.

   The API gateway forwards the response to the client.

Sideband API configuration basics

The Sideband API provides fine-grained access control to supported third-party API gateways through an API integration.

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 181.

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 181.

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

PingAuthorize 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 183.

Note:

Changes to these components do not typically require a server restart to take effect. For more information, see the *Configuration Reference*, located in the server's `docs/config-guide` directory.

Example

Example

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

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

PingAuthorize/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.

   Example:

   ```
   PingAuthorize/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.

   Example:

   ```
   PingAuthorize/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.

  Example:

  ```
  PingAuthorize/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.

  Example:

  ```
  PingAuthorize/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 181.
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 182.

Customizing the shared secret header

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

Steps

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

  Example:

  The following command changes the shared secret header to `x-shared-secret`.

  ```
  PingAuthorize/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.

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

Authenticating API server requests

As with the PingAuthorize 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

Sideband API 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
`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

Attribute	Description	Type
`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
`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
`active`	Indicates whether this access token is currently active, as determined by the access token validator.	Boolean

Attribute	Description	Type
`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
`scope`	Identifies the list of scopes granted to this token.	Collection
`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
`subjectRegex`	Regular expression that must be matched by the subject field of the certificate to ensure that the certificate belongs to the requesting client.	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
`_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`.
`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`.

Sideband API 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 188.

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

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.

```
PingAuthorize/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.

```
PingAuthorize/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.

```
PingAuthorize/bin/dsconfig \
  set-http-servlet-extension-prop \
  --extension-name "Sideband API" \
  --reset request-context-method
```

Sideband 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

Example configuration

The following example shows how to configure an external API gateway access token validator with a token resource lookup method and 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"
```

Sideband 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 PingAuthorize 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, PingAuthorize 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, PingAuthorize 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

PingAuthorize 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:

- *SCIM API request and response flow* on page 192
- *SCIM configuration basics* on page 193
- *SCIM endpoints* on page 196
- *SCIM authentication* on page 197
- *SCIM policy requests* on page 197
- *Lookthrough limit for SCIM searches* on page 207
- *Disabling the SCIM REST API* on page 207

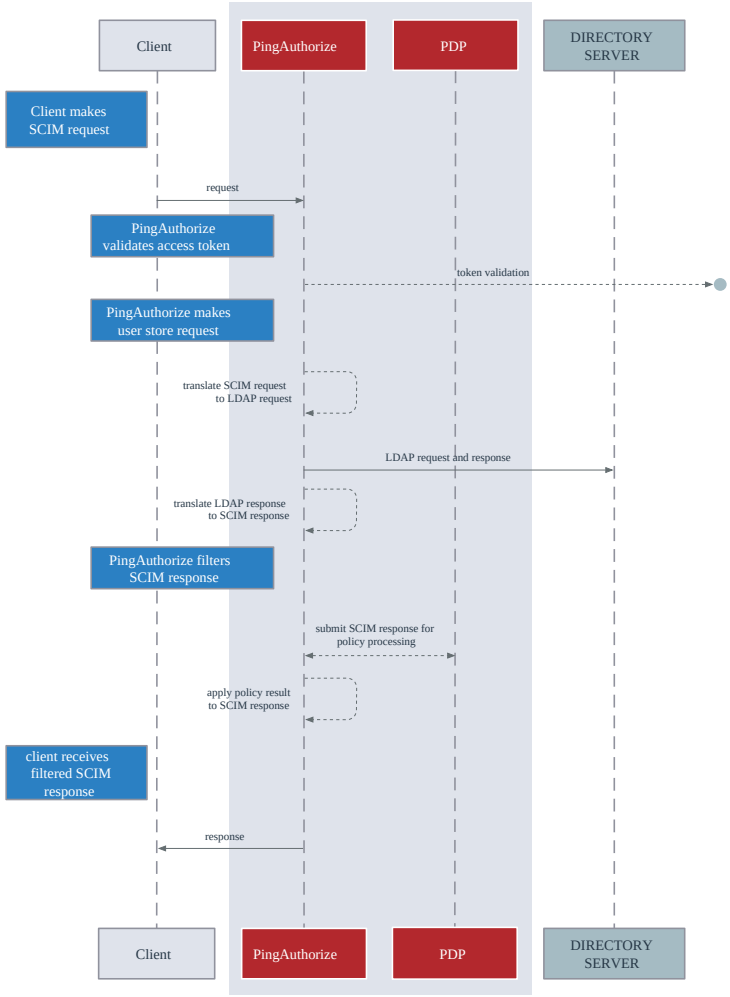
SCIM API request and response flow

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.



SCIM configuration basics

PingAuthorize 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 PingAuthorize Server's SCIM system and an external datastore. PingAuthorize 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 PingAuthorize 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 PingAuthorize user store* on page 359.

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 PingAuthorize 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 `PingAuthorize/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 PingAuthorize 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 PingAuthorize 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

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

SCIM authentication

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`

`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 202.

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 397.

SCIM 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

Policy request attribute	Description	Type
`attributes`	Additional attributes that do not correspond to a specific entity type in the PingAuthorize 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

Attribute	Description	Type
`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
`active`	Indicates whether this access token is currently active, as determined by the access token validator.	Boolean
`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
`scope`	Identifies the list of scopes granted to this token.	Collection
`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
`subjectRegex`	Regular expression that must be matched by the subject field of the certificate to ensure that the certificate belongs to the requesting client.	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 202.	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 202.
2. Policies filter the search result set. For more information, see *Search response authorization* on page 203.

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 203.

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.

Using paged SCIM searches

When searching large data sets, the results can be numerous and produce errors about a request matching too many results relative to the lookthrough limit. Paged searches avoid these errors and also reduce memory utilization.

Before you begin

The paged SCIM searches feature is not available for entry-balanced data sets.

To use paged SCIM searches, your SCIM service's backend servers must be LDAP directory servers and you must use the LDAP store adapter.

Complete the following one-time operations. For either command, you only need to run the command one time per backend server. If you are not sure whether you have run the command, you can run it again safely.

- Set the service account's permissions by running the `prepare-external-store` command on the PingAuthorize server for each backend server.

  > **Note:**
  >
  > If you have run this command with PingDataGovernance 8.1.0.0 or earlier, run it again using the command from a PingDataGovernance 8.2.0.0 or a PingAuthorize 8.3.0.0 or later release.

  For example:

  ```
  $ prepare-external-store --hostname server.example.com --port 1389 \
  --bindDN "cn=Directory Manager" --bindPassword <password1> \
  --governanceBindDN "cn=Authorize User,cn=Root DNs,cn=config" \
  --governanceBindPassword <password2> \
  --userStoreBaseDN ou=people,dc=example,dc=com
  ```

- If your LDAP store adapter points to a PingDirectoryProxy server, run the following command on that server.

  ```
  $ dsconfig set-request-processor-prop \
  --processor-name <proxying-request-processor> \
  --set supported-control-oid:2.16.840.1.113730.3.4.9 \
  --set supported-control-oid:1.2.840.113556.1.4.473
  ```

  where *<proxying-request-processor>* is the request processor handling the entries targeted by the search.

About this task

PingAuthorize does SCIM searches using LDAP requests. After you complete the steps below, PingAuthorize creates LDAP requests that include request controls that ask the backend servers to sort and page the search results before returning the results. These request controls are marked noncritical, meaning that if the backend server cannot page the results, the backend server still returns the results. In this case, PingAuthorize handles the sorting and paging itself.

If your SCIM searches result in an error because the request matched too many results, as discussed in *Lookthrough limit for SCIM searches* on page 207, you can avoid the error by using paged searches.

Complete the following steps for each search.

Steps

1. Decide your SCIM search.

   Note:
   To get paged results, your search must include at least one of these parameters: startIndex, count, or sortBy.

   For example, your search might look like the following search.

   ```
   https://<pingauthorize-hostname>:<pingauthorize-port>/scim/v2/Users/?
   filter=st eq "TX"&sortBy=sn&sortOrder=ascending
   ```

   Here is the corresponding encoded version.

   ```
   https://<pingauthorize-hostname>:<pingauthorize-port>/scim/v2/Users/?
   filter=st%20eq%20%22TX%22&sortBy=sn&sortOrder=ascending
   ```

   On your PingAuthorize Server, collect some information to use later.

   a. Given a SCIM resource type that you want to search for, find the primary LDAP store adapter that the SCIM resource type uses by looking at its `primary-store-adapter` property.
   b. Find the corresponding adapter by running the following command.

      ```
      $ dsconfig list-store-adapters
      ```

   c. Find the `structural-ldap-objectclass`, `include-base-dn`, and `include-filter` values for the adapter by running this command.

      ```
      $ dsconfig get-store-adapter-prop --adapter-name <name-of-store-adapter> \
      --property structural-ldap-objectclass \
      --property include-base-dn \
      --property include-filter
      ```

2. On each backend server, complete the following steps.

   a. Create a Virtual List View (VLV) index for your search.

      Each SCIM search that you want to produce paged results must have its own VLV index.

      Create this index using `dsconfig create-local-db-vlv-index` with the following options.

Option	Description
`--index-name`	Names the index.
`--backend-name`	Specifies the name of the local database backend in which to place the index. The default database backend for PingDirectory is userRoot.
`--set base-dn`	Specifies the desired base dn. This value must match the value of the `include-base-dn` property that you found in the previous step.
`--set scope`	Is always `whole-subtree`.

Option	Description
`--set filter`	Specifies the filter. Specify "`(objectclass=`*`<name-of-store-adapter-objectclass>`*`)`" where *`<name-of-store-adapter-objectclass>`* is the name of the objectclass used by the adapter, which you found in the previous step. If the primary LDAP store adapter has the `include-filter` property set, also specify that property value in the filter. For example, if the filter for the adapter objectclass is `(objectclass=inetorgperson)` and the `include-filter` value is `(st=CA)`, specify the `--set filter` argument as "`(&(objectclass=inetorgperson)(st=CA))`". Specify the LDAP attributes for all the components of your SCIM search filter. For example, if a mapping SCIM resource type maps the LDAP attribute `st` to the SCIM attribute `address.region` and the SCIM search filter requires that `address.region eq TX`, then this filter must include `(st = TX)` instead of `(address.region = TX)`.
`--set sort-order`	Specifies whether to sort ascending (+) or descending (-) and the LDAP attribute to sort by. If the SCIM search does not specify the `sortBy` parameter, specify the sort order as `+entryUUID`.

Recall the original, decoded SCIM search, shown here.

```
https://<pingauthorize-hostname>:<pingauthorize-port>/scim/v2/Users/?
filter=st eq "TX"&sortBy=sn&sortOrder=ascending
```

For example, to create a VLV index for that search, run the following command.

```
$ dsconfig create-local-db-vlv-index --index-name sn \
--backend-name userRoot --set base-dn:ou=people,dc=example,dc=com \
--set scope:whole-subtree \
--set filter:"(&(objectclass=inetorgperson)(st=TX))" --set sort-order:
+sn
```

b. Stop the server. Rebuild the index. Start the server. Run the `rebuild-index` command specifying the baseDN and the name of the index.

```
$ rebuild-index --baseDN <baseDN-value> --index <name-of-index>
```

For example, run these commands.

```
$ stop-server
$ rebuild-index --baseDN dc=example,dc=com --index vlv.sn
$ start-server
```

3. Run your SCIM search filter.

> **Note:**
>
> The search can include only the filter you specified with `--set filter` in the earlier step without the "`(objectclass=<name-of-store-adapter-objectclass>)`" portion.

In addition to the Virtual List View request control, PingAuthorize adds a Server Side request control to the LDAP request. These request controls require certain parameters be set. To satisfy this requirement, PingAuthorize uses the following parameters. If the client does not provide values for one of the parameters, the search uses the corresponding default value shown in the following table.

Parameter	Default
startIndex	1
count	The value of the `lookthrough-limit` property of the SCIM resource type being searched. That default is 500.
sortBy	entryUUID With this default, the results appear unsorted.
sortOrder	ascending

Lookthrough limit for SCIM searches

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, you have these options:

- The client must refine its search filter to return fewer matches.
- Configure paged searches as explained in *Using paged SCIM searches* on page 204.

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:

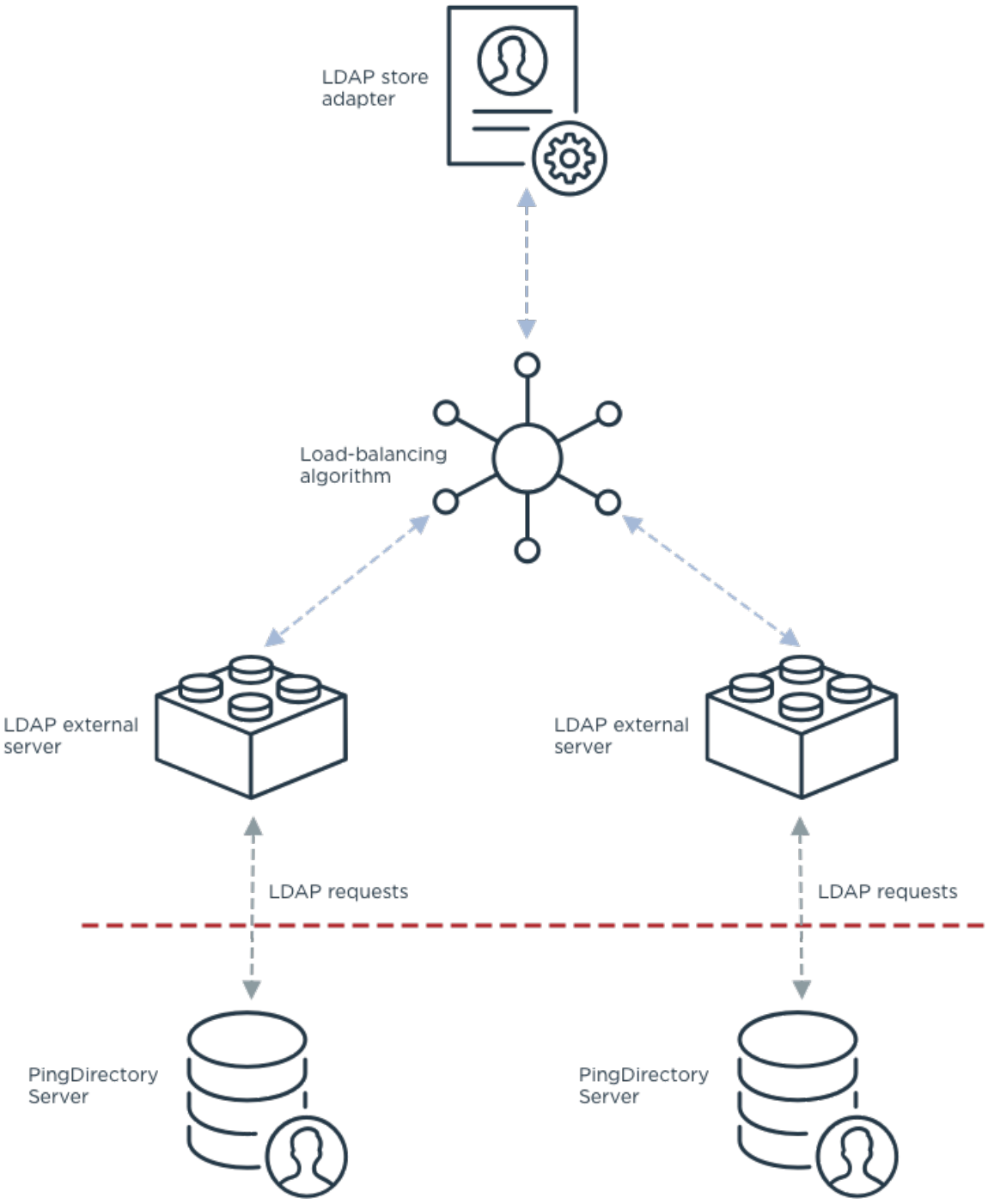
When the SCIM REST API is disabled, access token validators still use PingAuthorize Server's SCIM system to look up token owners.

About the SCIM user store

This topic focuses on the relationship between the PingAuthorize Server SCIM subsystem and its backend data stores, particularly LDAP directory servers.

For general information about SCIM configuration, see *SCIM configuration basics* on page 193.

The PingAuthorize 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.



PingAuthorize 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*. PingAuthorize 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.

Load-balancing algorithm type	Description
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 LDAP server discovery* on page 213.

LDAP external servers monitor and report the availability of backend LDAP servers using LDAP health checks. See *LDAP health checks* on page 218.

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 210
- *Defining the LDAP user store manually* on page 211

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 PingAuthorize user store* on page 359.

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 PingAuthorize Server will use to access backend LDAP servers. Create this account using the **`prepare-external-store`** tool.
`--governanceBindPassword`	The password for the above account.

Option	Description
`--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 PingAuthorize 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 PingAuthorize 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 PingAuthorize Server.

   Example: For example:

   ```
   prepare-external-store \
     --hostname ds1.example.com \
     --port 636 \
     --useSSL \
     --trustAll \
     --bindDN "cn=directory manager" \
     --bindPassword password \
     --governanceBindDN 'cn=Authorize 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 PingAuthorize Server connects to each LDAP server.

   Example: 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=Authorize 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=Authorize 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.

   Example: 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.

   Example: 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 PingAuthorize 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 208, take server location into account when routing requests. By default, they always prefer LDAP backend servers in the same location as the PingAuthorize 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 PingAuthorize 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 LDAP server discovery

Instead of explicitly specifying all backend LDAP servers in the configuration as LDAP external servers, you can configure PingAuthorize 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 PingAuthorize Server to the same topology as the PingDirectory Servers.
- Configure the PingAuthorize Server's load-balancing algorithm with an LDAP external server template. This template provides the connection and health check settings that PingAuthorize Server uses for all PingDirectory Servers.
- Configure the topology registry entry for each PingDirectory Server to indicate the name of the PingAuthorize Server load-balancing algorithm.

Joining a PingAuthorize Server to an existing PingDirectory Server topology
To use automatic backend discovery, the PingAuthorize Server must be a member of the same topology of each backend PingDirectory Server.

You can join a PingAuthorize 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 214
- *Joining a topology with manage-topology* on page 215

Joining a topology at setup
To join a new PingAuthorize 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.
`--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 PingAuthorize 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.
`--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 PingAuthorize 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 LDAP server 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 PingAuthorize Server connects to each LDAP server that it discovers. Typically, the default settings are sufficient, so this example only specifies the template name.

   Example: 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.

   Example: 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.

   Example: 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 PingAuthorize Server to a running PingDirectory Server.

   Example: 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 PingAuthorize 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`.

   Example: 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 PingAuthorize 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 PingAuthorize Server to prefer the server with the higher score.

PingAuthorize 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 PingAuthorize Server are made available to the load-balancing algorithms to take into account when determining where to send requests. PingAuthorize 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 PingAuthorize 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 219. 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

PingAuthorize Server provides the following LDAP health checks.

Health check	Description
Measure the response time for searches and examine the entry contents	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.
Monitor the replication backlog	If a server falls too far behind in replication, then a PingAuthorize 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.
Consume PingAuthorize Server administrative alerts	If a PingDirectory Server indicates there is a problem, it flags itself as DEGRADED or UNAVAILABLE. When a PingAuthorize Server detects this, it stops sending requests to the server. You can configure a PingAuthorize Server to detect administrative alerts as soon as they are issued by maintaining an LDAP persistent search for changes within the `cn=alerts` branch of a PingDirectory Server. When PingAuthorize Server is notified by the PingDirectory Server of a new alert, it can immediately retrieve the base `cn=monitor` entry of the PingDirectory Server.
Monitor the busyness of the server	If a server becomes too busy, the health check might mark it as DEGRADED or UNAVAILABLE so that less heavily loaded servers are preferred.

When **`cn=monitor`** entry has value for this attribute:	PingAuthorize Server should consider PingDirectory Server to be:
`unavailable-alert-type`	UNAVAILABLE
`degraded-alert-type`	DEGRADED

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.
   Example:

   ```
   $ bin/dsconfig
   ```
2. Type the host name or IP address for your PingAuthorize Server, or press **Enter** to accept the default, `localhost`.
   Example:

   ```
   PingAuthorize Server host name or IP address [localhost]:
   ```

3. Type the number corresponding to how you want to connect to PingAuthorize, or press **Enter** to accept the default, LDAP.

   Example:

   ```
   How do you want to connect?
     1) LDAP
     2) LDAP with SSL
     3) LDAP with StartTLS
   ```

4. Type the port number for your PingAuthorize Server, or press **Enter** to accept the default, 389.

   Example:

   ```
   PingAuthorize 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.

   Example:

   ```
   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.

   Example:

   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.

   Example:

   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.

   Example:

   ```
   >>>> 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.

    Example:

    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 PingAuthorize Server SCIM subsystem supports non-LDAP data stores using custom store adapter extensions. For more information, see the Server SDK.

About the Authorization Policy Decision APIs

The PingAuthorize Server provides Authorization Policy Decision APIs to support non-API use cases needing attribute-based access control.

> **Important:**
>
> The Authorization Policy Decision APIs feature requires PingAuthorize Premier. For more information, contact your Ping Identity account representative.

The PingAuthorize 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, PingAuthorize Server becomes the PDP, and the application server becomes the policy enforcement point (PEP).

The Authorization Policy Decision APIs consist of the following policy decision point (PDP) APIs:

- XACML-JSON PDP API

  This API provides a standards-based interface.

  Standards-based 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*.
- JSON PDP API

  This API provides a simpler interface.

> **Note:**
>
> The Authorization Policy Decision APIs can indicate when a request or response triggers advice, but the application server must implement the advice.

To make a PDP API available, you must:

- Configure the PingAuthorize Server with a feature-enabled license during setup.
- Configure the Policy Decision Point Service. For more information, see *Use policies in a production environment*.
- For the XACML-JSON PDP API, configure an Access Token Validator. For more information, see *Access Token Validators*.

JSON PDP API request and response flow

The JSON policy decision point (PDP) API provides an HTTP REST API for attribute-based access control based on policies configured in the PingAuthorize Server Policy Decision Service.

The JSON PDP API is implemented with both an individual decision request endpoint and a batch request endpoint that consuming application servers can access using POST requests to the `/governance-engine` or `/governance-engine/batch` paths, respectively.

The HTTP requests must include the appropriate `Content-Type` and `Accept` headers, and request bodies must be valid JSON in the expected request format.

The endpoint paths and headers are listed in the following table.

JSON PDP API Endpoint path	Action	Content-Type/Accept	Request data
`/governance-engine`	POST	application/json	JSON

JSON PDP API Endpoint path	Action	Content-Type/Accept	Request data
`/governance-engine/batch`	POST	application/json	JSON

A successful JSON PDP API request goes through the following flow:

1. The client makes the JSON request, which is received by the JSON PDP API. The API forwards the request to the PDP.
2. When the PDP returns a response, the API sends the response to the client.

Note:

The Policy Enforcement Point (PEP) must apply any obligations or advice. See the *JSON PDP API Reference* for more information about making API requests.

JSON PDP API request format

Individual requests

A valid JSON PDP API request is a simple JSON object that can be forwarded to the Policy Decision Service. Policies can match a decision request by `Service`, `Domain`, `Action`, or other attributes.

The following table describes the values contained in a valid JSON PDP API request.

Field	Type	Required	PingAuthorize Trust Framework type	Example value
`domain`	string	no	Domain	`Sales.Asia Pacific`
`action`	string	no	Action	`Retrieve`
`service`	string	no	Service	`Mobile.Landing page`
`identityProvider`	string	no	Identity Provider	`Social Networks. Spacebook`
`attributes`	map<string, string>	yes	Other Attributes	`{"Prospect name": "B. Vo"}`

Tip:

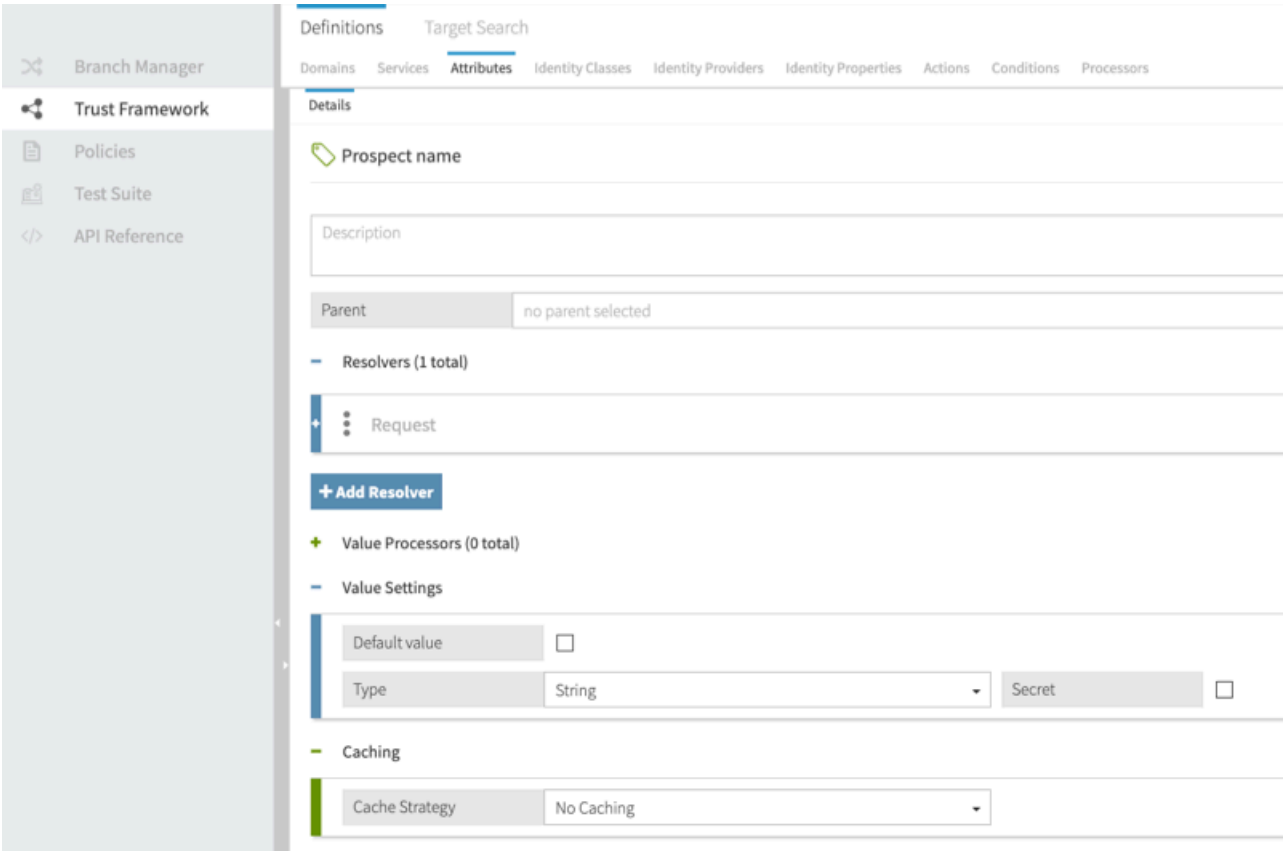
While the `attributes` value is required, you can leave it empty.

The following example shows the correct format of a JSON individual decision request.

```
{
 "domain": "Sales.Asia Pacific",
 "action": "Retrieve",
```

```
 "service": "Mobile.Landing page",
 "identityProvider": "Social Networks.Spacebook",
 "attributes": {
  "Prospect name": "B. Vo"
 }
}
```

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.



Note:

The Trust Framework attribute name must match with the key of the attributes map.

For example, if you have an attribute named `"UserID"`, an example value for the `"attributes"` object would be `{"UserID":13848}`.

Batch requests

Batch requests consist of an array named `"requests"` of JSON objects, each of which is a standard JSON PDP API single decision request.

The following example shows the correct format of a JSON batch decision request.

```
{
    "requests": [
   {
       "domain": "Sales.Asia Pacific",
       "action": "Retrieve",
       "service": "Mobile.Landing page",
       "identityProvider": "Social Networks.Spacebook",
```

```
        "attributes": {
           "Prospect name": "B. Vo"
         }
           },
           {
        "domain": "Sales.EMEA",
        "action": "Search",
        "service": "Mobile.Users search",
        "identityProvider": "Social Networks.Chirper",
        "attributes": {
           "Prospect name": "A. Mann"
     }
           }
    ]
}
```

JSON PDP API response format

After the Policy Decision Service determines a decision response, it hands the response back to the JSON PDP API to provide to the client. JSON PDP API responses include decisions, such as `Permit` or `Deny`, and any obligations or advice that matched during policy processing.

Individual response

The following example shows the correct JSON individual response format.

```
{
 "id": "12345678-90ab-cdef-1234-567890abcdef",
 "deploymentPackageId": "12345678-90ab-cdef-1234-567890abcdef",
 "timestamp": "2021-06-11T03:12:19.720485Z",
 "elapsedTime": 184024,
 "decision": "PERMIT",
 "authorized": true,
 "statements": [
  {
         "id": "12345678-90ab-cdef-1234-567890abcdef",
         "name": "Advice Name",
         "code": "advice-code",
         "payload": "{\"data\": \"some data\"}",
         "obligatory": true,
         "fulfilled": false,
         "attributes": { }
     }
],
 "status": {
  "code": "OKAY",
  "messages": [ ],
  "errors": [ ],
 }
}
```

Note:

The `decision` and `authorized` values identify whether the policies authorize the request, and the `"statements"` array contains advice to be applied by the Policy Enforcement Point.

Batch response

Batch decision responses consist of an array, named `"responses"`, of JSON objects, each of which is a standard JSON PDP API single decision response. The decision responses are guaranteed to be returned

in the same order as the received responses. For example, the first response in the batch responses corresponds to a decision on the first request in the batch requests.

The following example shows the correct JSON batch decision response format.

```
{
    "responses": [
   {
       "id": "12345678-90ab-cdef-1234-567890abcdef",
       "deploymentPackageId": "12345678-90ab-cdef-1234-567890abcdef",
       "timestamp": "2021-06-11T04:18:32.820482Z",
       "elapsedTime": 830492,
       "decision": "PERMIT",
       "authorized": true,
       "statements": [
    {
               "id": "12345678-90ab-cdef-1234-567890abcdef",
               "name": "Advice Name",
               "code": "advice-code",
               "payload": "{\"data\": \"some data\"}",
               "obligatory": true,
               "fulfilled": false,
               "attributes": {}
               }
              ],
       "status": {
        "code": "OKAY",
        "messages": [ ],
         "errors": [ ],
           }
         },
         {
       "id": "fedcba09-8765-4321-fedcba098765",
       "deploymentPackageId": "fedcba09-8765-4321-fedcba098765",
       "timestamp": "2021-06-11T04:18:33.650974Z",
       "elapsedTime": 492048,
       "decision": "PERMIT",
       "authorized": true,
       "statements": [
     {
               "id": "fedcba09-8765-4321-fedcba098765",
               "name": "Different Advice",
               "code": "advice-code",
               "payload": "{\"data\": \"other data\"}",
               "obligatory": false,
               "fulfilled": false,
               "attributes": { }
                 }
        ],
 "status": {
  "code": "OKAY",
  "messages": [ ],
  "errors": [ ],
   }
      }
    ]
}
```

Authenticating to the JSON PDP API

The JSON PDP API can require a client to authenticate to it by using a shared secret.

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

Creating a shared secret

Define the authentication credentials that the JSON PDP API might require a client to present.

Steps

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

   Example:

   ```
   PingAuthorize/bin/dsconfig create-authorization-policy-decision-shared-
   secret \
     --secret-name "Shared Secret A" \
     --set "shared-secret:secret123"
   ```

   Note:

   - The `shared-secret` property sets the value that the JSON PDP API requires the client 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 JSON PDP API Shared Secret from another.

2. To update the `shared-secrets` property, run the following example **`dsconfig`** command.

   Example:

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

   A new JSON PDP API Shared Secret is not used until the `shared-secrets` property of the JSON PDP 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 JSON PDP API Shared Secret from use, run the following example **`dsconfig`** command, substituting values of your choosing.

  Example:

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

- To delete a JSON PDP API Shared Secret, run the following example **`dsconfig`** command.

  Example:

  ```
  PingAuthorize/bin/dsconfig delete-authorization-policy-decision-shared-
  secret \
    --secret-name "Shared Secret A"
  ```

Rotating shared secrets
To avoid service interruptions, the JSON PDP 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 JSON PDP API accepts alongside an existing shared secret. This allows time to update the client to use the new shared secret.

Steps

1. Create a new JSON PDP API Shared Secret and assign it to the JSON PDP API HTTP Servlet Extension. For more information, see *Creating a shared secret* on page 227.
2. Update the client to use the new shared secret.
3. Remove the previous JSON PDP API Shared Secret. For more information, see *Deleting a shared secret* on page 227.

Customizing the shared secret header
By default, the JSON PDP API accepts a shared secret from a client through the CLIENT-TOKEN header.

Steps

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

  Example:

  The following command changes the shared secret header to `x-shared-secret`.

  ```
  PingAuthorize/bin/dsconfig set-http-servlet-extension-prop \
    --extension-name "JSON PDP API" \
    --set shared-secret-header-name:x-shared-secret
  ```

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

  ```
  PingAuthorize/bin/dsconfig set-http-servlet-extension-prop \
    --extension-name "JSON PDP API" \
    --reset shared-secret-header-name
  ```

XACML-JSON PDP API request and response flow

The XACML-JSON policy decision point (PDP) API provides a standards-based HTTP API for decisions determined based on the policies configured within the PingAuthorize Server Policy Decision Service.

The XACML-JSON 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 229.

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

The XACML-JSON 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. See the *XACML-JSON PDP API Reference* for more information about making API requests.

A successful XACML-JSON PDP API request goes through the following two-phase flow:

1. The client makes the XACML-JSON request, which is received by the XACML-JSON PDP API. The API converts the request to a PingAuthorize 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 XACML-JSON PDP API. The 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 XACML-JSON 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 233.

```
{
  "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"
      }]
    }]
  }
}
```

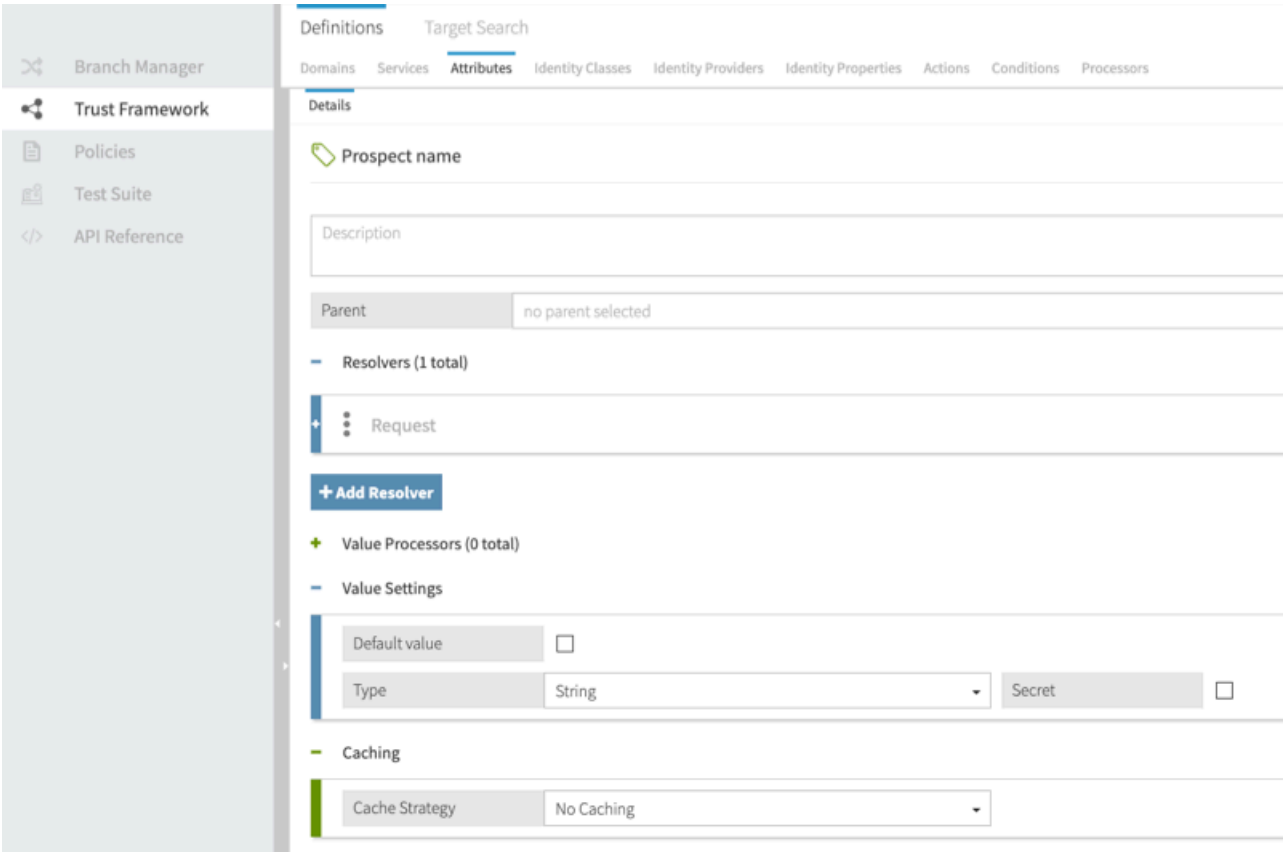
The previous example shows a single decision request with the following attributes:

- 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)	PingAuthorize 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 Editor. In this example, the `Prospect name` attribute has a Request resolver and a **Value Settings Type** of `String`.



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 XACML-JSON PDP API attempts to authorize the client making the XACML-JSON 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 PingAuthorize Server perform this authorization by only permitting requests with active access tokens that contain the `urn:pingauthorize: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:pingauthorize:pdp", "sub":"<valid-subject>"}' \
  -H 'Content-Type: application/xacml+json' \
  -d '{"Request":{}}' "https://<your-pingauthorize-host>:<your-pingauthorize-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 XACML-JSON PDP API invokes the Policy Decision Service with the batch decision requests converted from the XACML-JSON request.

When writing policy for the XACML-JSON PDP API endpoint, you should note the mapping between the XACML-JSON schema and the PingAuthorize Server decision request. For more information, see *Requests* on page 229. After the Policy Decision Service determines a decision response, it hands the response back to the XACML-JSON PDP API to provide to the client.

Responses
The XACML-JSON 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)	PingAuthorize 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 229.

```
{
  "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 XACML-JSON 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)	PingAuthorize Trust Framework type	Example value
`$.Request.AccessSubject[*]`	`$.Attribute[?(@.AttributeId == "domain")].Value`	Domain	`AnyCompany.Management`

Parent (JSON Path)	Field (JSON Path)	PingAuthorize Trust Framework type	Example value
`$.Request.Action[*]`	`$.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)	PingAuthorize 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)	PingAuthorize 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`
`$.Request.Category[*]`	`$.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 Editor configuration

You can configure the PingAuthorize Policy Editor 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
- HTTP caching status

Specifying custom configuration with an options file

You can configure the Policy Editor 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 Editor 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 Editor's logging behavior.
- Configure private JSON Web Token (JWT) claims. This allows an organization to convey specific claims about an identity.

Note:

If the server detects existing configuration files when running the **`setup`** tool, the setup process terminates. To re-configure the server, you must either:

- Delete the existing configuration files and run **`setup`** again.
- Use the `--ignoreWarnings` option with the **`setup`** tool to overwrite the existing `configuration.yml` file, delete the administrator key store, and, if you also use the `--generateSelfSignedCertificate` option, overwrite the server certificate file.

To reconfigure the server while preserving the values in `configuration.yml` or any certificate key stores, back up the `configuration.yml` and key stores before re-running **`setup`**.

Steps

1. Make a copy of the default options file provided at `config/options.yml` and 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 Identity Support*.

2. Configure the Policy Editor with an options file:
   a. Stop the Policy Editor:

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

   b. Run the **`setup`** tool.
   c. Provide the options file using the `--optionsFile` argument.

      For example, the following **`setup`** command configures a Policy Editor in demo mode using an options file named `my-options.yml`:

      ```
      $ bin/setup demo \
        --adminUsername admin \
        --generateSelfSignedCertificate \
        --decisionPointSharedSecret pingauthorize \
        --hostname <pap-hostname> \
        --port <pap-port> \
        --adminPort <admin-port> \
        --licenseKeyFile <path-to-license> \
        --optionsFile my-options.yml
      ```

3. Start the Policy Editor:

   ```
   $ 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 PingAuthorize Policy Editor 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 Editor'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 Editor.

```
$ 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 pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

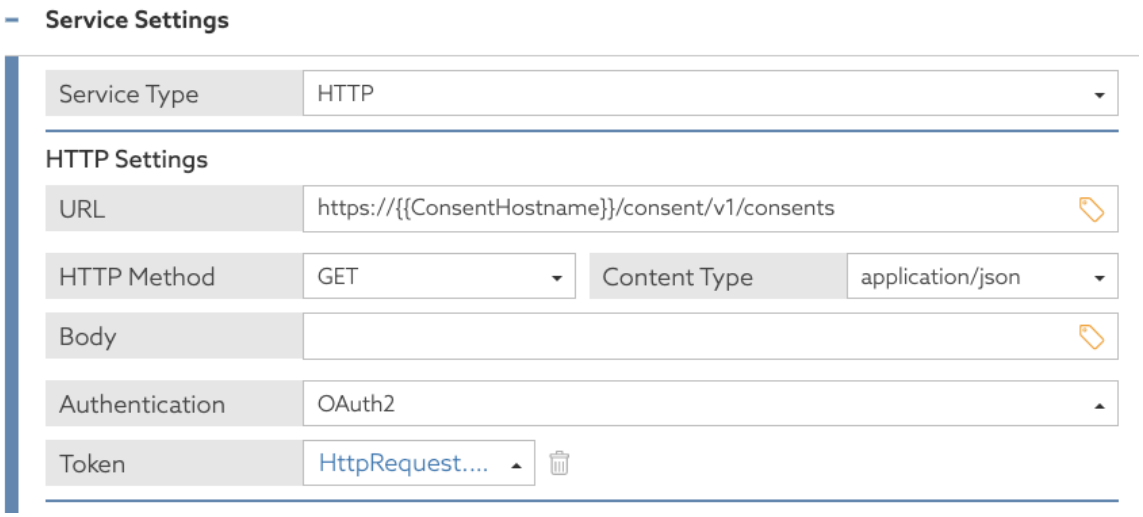
5. Start the Policy Editor.

```
$ 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 Editor 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.



Key store configuration for policy information providers

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 PingAuthorize.

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 Editor.

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

4. Run **`setup`** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

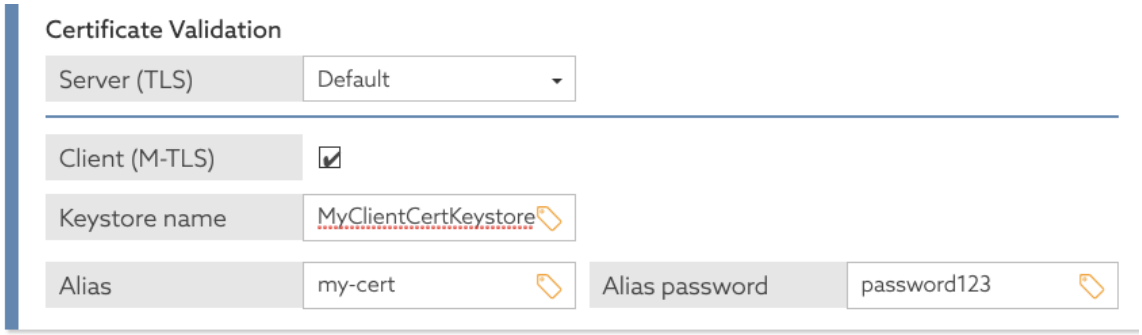
```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
```

```
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

5. Start the Policy Editor.

```
$ 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`.



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 PingAuthorize.

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.

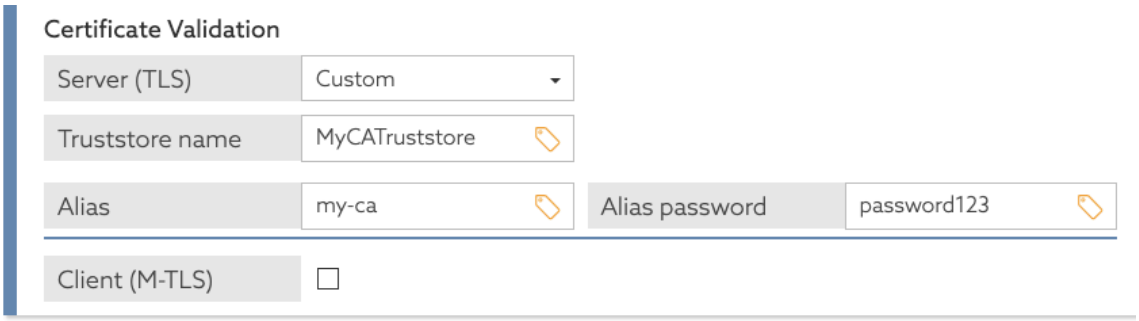
```
truststores:
  - name: MyCATruststore
    resource: /path/to/my-ca-truststore.jks
    password: password123
# Other options omitted for brevity...
```

3. Run **`setup`** using the `--optionsFile` argument. Customize all other options as needed.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
```

```
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-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`.



Policy Editor configuration with runtime environment variables

You do not have to hard-code values for policy configuration keys in an options file in the Policy Editor 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 Editor.

Example: Set policy information provider host name using an environment variable

This example takes the scenario in *Example: Configure policy configuration keys* on page 239 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 pingauthorize \
     --hostname <pap-hostname> \
     --port <pap-port> \
     --adminPort <admin-port> \
     --licenseKeyFile <path-to-license> \
   ```

```
  --optionsFile my-options.yml
```

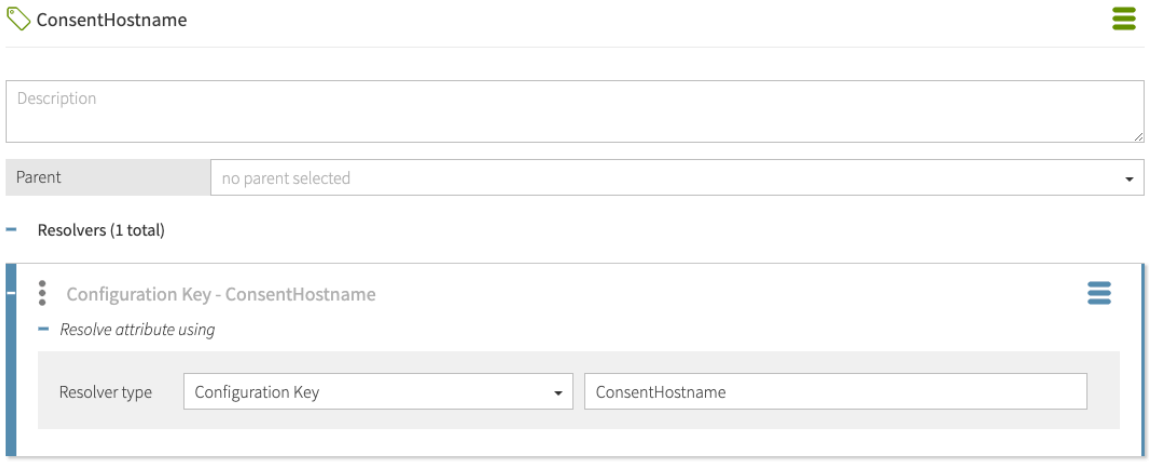
5. Set the value of the **`CONSENT_HOSTNAME`** environment variable and then start the server.

```
$ export CONSENT_HOSTNAME=consent-us-east.example.com; bin/start-server
```

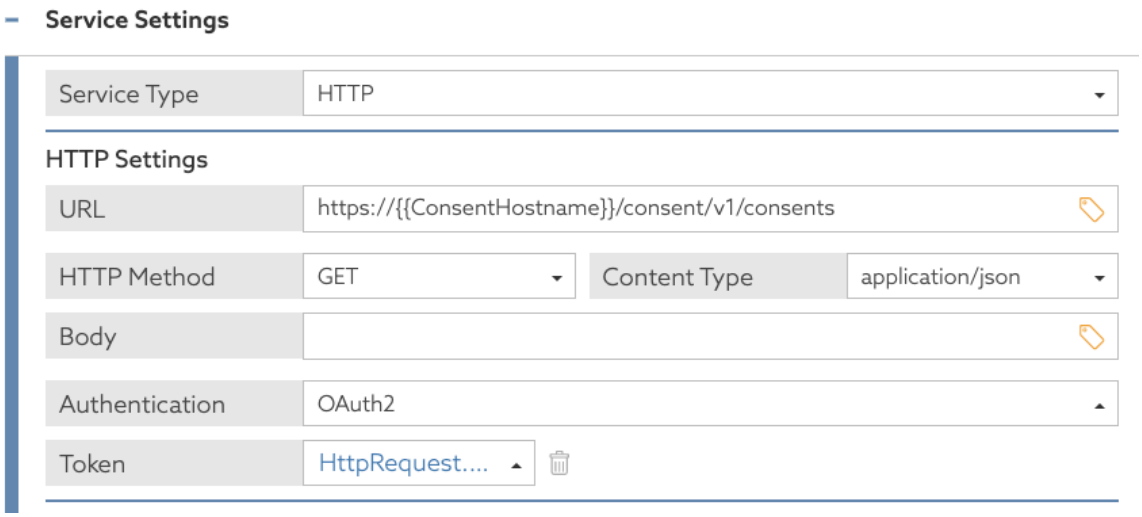
After you define the Consent API service in the Trust Framework, you can refer to the policy configuration key that you defined in the Policy Editor configuration (ConsentHostName), which will use the environment variable that you also defined. You must first create an attribute in the Trust Framework 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 Editor.



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.



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 242 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 Editor.

   ```
   $ 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 pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

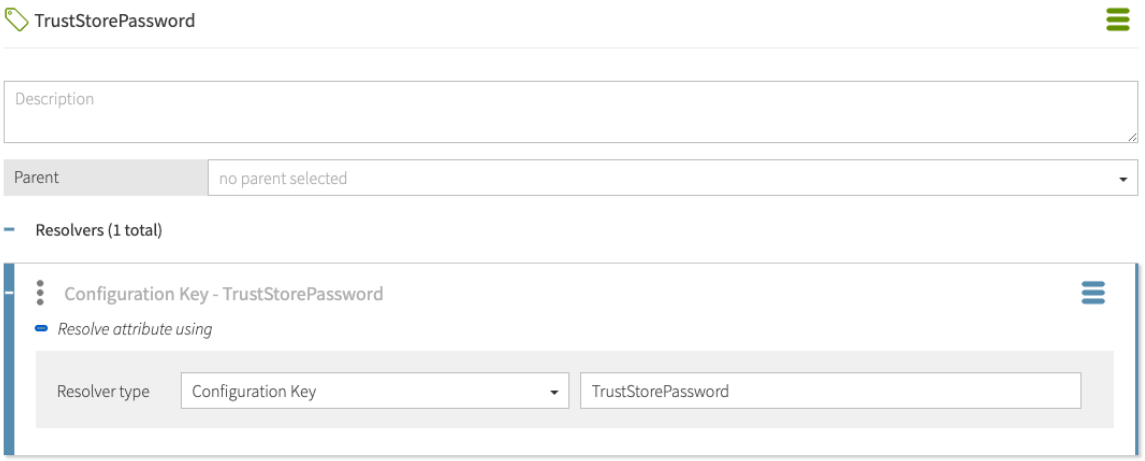
5. Set the value of the **`TRUST_STORE_PASSWORD`** environment variable and start the server.

```
$ export TRUST_STORE_PASSWORD=password123; bin/start-server
```

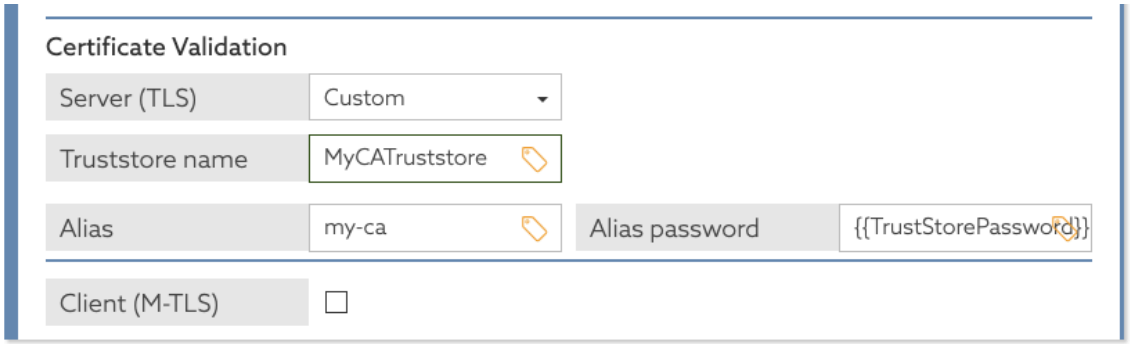
The policy configuration key that you defined can be used in the Trust Framework. You must first create an attribute to hold the policy configuration key value. Add an attribute with the following settings.

Property	Value
Name	TrustStorePassword
Resolver Type	Configuration Key
Resolver Value	TrustStorePassword

The following image shows the attribute in the Policy Editor.



After you define the policy information provider in the Trust Framework, you can refer to the trust store password using the TrustStorePassword attribute.



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
```

Example: Configure JWT claims
You can configure private JSON Web Token (JWT) claims for your organization under the option file's `core` section.

The JWT specification defines registered claims and also allows for public and private claims to be included in the token. The seven optional, registered claims are:

- `iss`
- `sub`
- `aud`
- `exp`
- `nbf`
- `iat`
- `jti`

Note:

When you configure private claims for your organization, make sure you avoid name collisions because private claim names are not registered.

Example

When a user signs on with OpenID Connect (OIDC), the Policy Editor uses the JWT `sub` claim in the user profile as the default OIDC user ID. Changes committed by policy editors are recorded under this user ID. If your organization wants to record changes under the email address instead, you can define a different claim, such as `email`, for the OIDC user ID.

To define this claim:

1. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

2. In the `core` section of the new options file, uncomment the example `Authentication.oidcUserIdField` field that uses the `email` claim.

   ```
   core:
   # Use a JWT claim other than "sub" for the OIDC User ID.
   #
   # Authentication.oidcUserIdField: jwt_claim
   #
   Authentication.oidcUserIdField: "email"
   ```

3. Stop the Policy Editor.

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

4. Run **`setup`** using the `--optionsFile` argument and customize all other options as appropriate for your needs.

   ```
   $ bin/setup demo \
     --adminUsername admin \
     --generateSelfSignedCertificate \
   ```

```
  --decisionPointSharedSecret pingauthorize \
  --hostname <pap-hostname> \
  --port <pap-port> \
  --adminPort <admin-port> \
  --licenseKeyFile <path-to-license> \
  --optionsFile my-options.yml
```

5. Start the Policy Editor.

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

6. To verify that your claim is used, commit a policy change in the Policy Editor at **Branch Manager** # **Version Control** and ensure that your claim appears in the **Creator** column.

Configuring the Policy Editor to publish policies to a deployment package store
Use an options file to configure the Policy Editor.

About this task

To use the Deployment Manager feature, you must configure the Policy Editor to publish policies to a deployment package store in the options file's `deploymentPackageStores` section.

For more information, see *Using the Deployment Manager* on page 267.

Steps

1. Make a copy of the *default options file*.

```
$ cp config/options.yml my-options.yml
```

2. To define a deployment package store or stores for the Policy Editor to publish policies to, edit the `deploymentPackageStores` section of the new options file.

   The file contains commented out examples of different deployment package store types.

   a. Duplicate the desired deployment package store type, uncomment, and modify its values according to your deployment.

> **Important:**
>
> - The use of indentation in the `options.yml` file is important. When removing comment hashes, ensure that you retain valid YAML file indentation structure.
> - For an Azure deployment package store, record the prefix you define for the deployment package store. You will need the prefix for PingAuthorize Server configuration.

```
deploymentPackageStores:
  # Define deployment package store publishing targets here.
  #
  # - name: Filesystem store
  #   description: File system directory store
  #   type: filesystem
  #   path: /path/to/deployment-package-store/
  # - name: Signed filesystem store
  #   description: Signed file system directory store
  #   type: filesystem
  #   path: /path/to/signed-deployment-package-store/
  #   securityLevel: signed
  #   keystore:
  #     resource: /path/to/deployment-package-signing-keystore.jks
  #     password: keystore-password
  #   signingKey:
```

```
  #     alias: signing-cert-alias
  #     password: private-key-password
  # - name: S3 bucket store
  #   description: AWS S3 bucket store
  #   type: s3bucket
  #   securityLevel: unsigned-or-signed
  #   config:
  #     bucket: store-bucket-name
  #     prefix: store-prefix
  #     endpoint: https://s3-bucket-endpoint.aws-region.amazonaws.com
  #     region: aws-s3-bucket-region
  #     accessKey: aws-access-key
  #     secretKey: aws-secret-key
  #  Other deployment package store types omitted for brevity...
```

3. Stop the Policy Editor.

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

4. Run `setup` using the `--optionsFile` argument.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname  <pap-hostname> \
  --port  <pap-port> \
  --adminPort  <admin-port> \
  --licenseKeyFile  <path-to-license> \
  --optionsFile my-options.yml
```

5. Start the Policy Editor.

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

6. To verify that your deployment package store or stores are available in the Policy Editor, go to **Branch Manager** # **Deployment Manager**.

Configuring Policy Editor security headers
Use an options file to configure the Policy Editor.

About this task

You can configure the Policy Editor to add certain security headers to responses for calls to the UI resources in the options file's `securityHeaders` section. Supported headers include X-Frame-Options, Content-Security-Policy, and Access-Control-Allow-Origin. By default, X-Frame-Options will be set to `deny` and the other headers will remain unset.

Steps

1. Make a copy of the default options file.

```
$ cp config/options.yml my-options.yml
```

2. To configure Policy Editor security headers, edit the `securityHeaders` section of the new options file.

   The file contains commented out examples of different security headers.

   a. Duplicate the desired security header, uncomment, and modify its value according to your deployment.

> **Note:**
>
> The use of indentation in the `options.yml` file is important. When removing comment hashes, ensure that you retain valid YAML file indentation structure.

   The following example illustrates the X-Frame-Options header duplicated and modified.

```
securityHeaders:
  # Configure the values that the Policy Editor will set in its
  # responses for the X-Frame-Options, Content-Security-Policy, and/or
  # Access-Control-Allow-Origin HTTP security headers here.
  #
  # X-Frame-Options: "deny"
  # Content-Security-Policy: "default-src https:"
  # Access-Control-Allow-Origin: "*"
  X-Frame-Options: "sameorigin"
```

3. Stop the Policy Editor.

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

4. Run `setup` using the `--optionsFile` argument.

```
$ bin/setup demo \
  --adminUsername admin \
  --generateSelfSignedCertificate \
  --decisionPointSharedSecret pingauthorize \
  --hostname  <pap-hostname> \
  --port  <pap-port> \
  --adminPort  <admin-port> \
  --licenseKeyFile  <path-to-license> \
  --optionsFile my-options.yml
```

5. Start the Policy Editor.

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

Manage policy database credentials

By default, the PingAuthorize Policy Editor stores policies in an H2 database file on the server. You can set the initial credentials and change them later.

> **Note:**
>
> These instructions don't apply if you are using a managed RDBMS, such as PostgreSQL, instead of the default H2 database.

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 PingAuthorize* on page 126.

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 PingAuthorize Policy Editor.

About this task

Using **`setup`** or environment variables, you can set credentials for both the admin user and the application user.

Because this setup is an initial setup, the Policy Editor is not running.

Steps

- Set credentials for both the admin user and the application user.
  Choose from:
  - Setting credentials 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 policy database application credentials to `appuser` / `S3cret`.

    ```
    bin/setup --dbAdminUsername adminuser \
      --dbAdminPassword Passw0rd \
      --dbAppUsername appuser \
      --dbAppPassword S3cret \
      --interactive
    ```

  - Setting credentials with environment variables.

    Using environment variables, you can avoid credentials showing up in process lists and command-line history.

    The following example sets the policy database admin credentials to `adminuser` / `Passw0rd` and the application user credentials to `app` / `S3cret`.

    ```
    env PING_DB_ADMIN_USERNAME=adminuser \
      PING_DB_ADMIN_PASSWORD=Passw0rd \
      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 `adminuser` / `Passw0rd` credentials and the `app` / `S3cret` credentials instead of the default credentials.

For more information about these and other UNIX environment variables you can use to override configuration settings, see *Starting PingAuthorize Policy Editor* on page 162.

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 Editor.

   ```
   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 Editor.

```
bin/start-server
```

Specifying database credentials when you start the GUI
You can override database credentials for the admin account and application account in the `configuration.yml` file when you start the GUI by using the UNIX environment variables **`PING_DB_ADMIN_USER`**, **`PING_DB_ADMIN_PASSWORD`**, **`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 PingAuthorize Policy Editor* on page 162.

Steps

1. Stop the Policy Editor.

```
bin/stop-server
```

2. Set the environment variables and start the Policy Editor.

Example
The following example starts the server with the overridden policy database admin credentials `adminuser` / `Passw0rd` and the overridden policy database application credentials `app` / `S3cret`. These environment variables override any values in `configuration.yml`.

```
env PING_DB_ADMIN_USERNAME=adminuser \
  PING_DB_ADMIN_PASSWORD=Passw0rd \
  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 *Deploying PingAuthorize Policy Editor using Docker* on page 86.

Note:
For proper communication between containers, create a Docker network using a command such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

```
$ docker run --network=<network_name> \
  --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/pingauthorizepap
```

Docker: Changing database credentials
When your Docker container uses `/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 `pap` with a mounted volume as shown in the example in *Deploying PingAuthorize Policy Editor using Docker* on page 86, the following command changes the credentials for the admin and application accounts from their default values to `admin` / `Passw0rd` and `app` / `S3cret`, respectively.

Note:
For proper communication between containers, create a Docker network using a command such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

```
docker run --network=<network_name> -p 443:1443 -d \
  --env-file ~/.pingidentity/config \
  --volumes-from pap \
  --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/out/Symphonic \
```

```
pingidentity/pingauthorizepap:<TAG>
```

The Docker image <*TAG*> used in the example is only a placeholder. For actual tag values, see Docker Hub (*https://hub.docker.com/r/pingidentity/pingauthorize*).

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.

> **Note:**
>
> These instructions are for configuring SpEL Java classes for use in the Policy Editor. When using embedded PDP mode, you must add Java classes to the **SpEL Allowed Class** list to use them in deployment packages. See *Adding SpEL Java classes to the allowed list*.

Steps

1. Make a copy of the default options file.

   Example:

   ```
   $ 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 classes in these packages:

   ```
   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 Editor.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.

   Example:

```
$ bin/setup demo \
 --adminUsername admin \
 --generateSelfSignedCertificate \
 --decisionPointSharedSecret <shared-secret> \
 --hostname <pap-hostname> \
 --port <pap-port> \
 --adminPort <admin-port> \
 --licenseKeyFile <path-to-license> \
 --optionsFile my-options.yml
```

5. Start the Policy Editor.

   Example:

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

Setting the request list length for Decision Visualizer

In the PingAuthorize Policy Editor, you can select **Policies**, **Decision Visualizer**, and then **Recent Decisions** to view graphs of recent decisions, the times the requests were made, and the decision outcomes. The requests do not include test requests.

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 Editor 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.
   Example:

   ```
   $ 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.

   > **Warning:**
   > Setting a buffer size greater than 20 can cause serious performance degradation.

   To disable the feature, set the value to 0.

   Example:

   ```
   core:
     RecentRequest.buffer.size: 10
   # Other options omitted for brevity...
   ```

3. Stop the Policy Editor.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.
   Example:

   ```
   $ bin/setup demo \
    --adminUsername admin \
    --generateSelfSignedCertificate \
    --decisionPointSharedSecret <shared-secret> \
    --hostname <pap-hostname> \
    --port <pap-port> \
    --adminPort <admin-port> \
    --licenseKeyFile <path-to-license> \
    --optionsFile my-options.yml
   ```

5. Start the Policy Editor.
   Example:

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

HTTP caching

The Policy Editor transfers data through HTTP APIs.

To improve page loading speeds, the Policy Editor uses HTTP headers to cache the API responses for the following URLs:

- `/app/trust-framework/*`
- `/app/policy-manager/*`
- `/app/test-suite/*`

HTTP caching is enabled by default.

> **Note:**

When hosting the Policy Editor using a self-signed SSL certificate, browsers like Google Chrome and Microsoft Edge don't include the caching HTTP headers. Customers with such deployments can use a different browser, such as Mozilla Firefox, to observe the performance benefits.

You can disable HTTP caching persistently by providing the `--disableApiHttpCache` option when running **`setup`**. Caching remains disabled for future server starts and stops. The following example illustrates this option:

```
bin/setup demo \
--disableApiHttpCache \
--adminUsername admin \
--generateSelfSignedCertificate \
--decisionPointSharedSecret 2FederateM0re \
--hostname example.com \
--port 9443 \
--adminPort 9444
```

To disable HTTP caching for a single server start, provide the **`PING_ENABLE_API_HTTP_CACHE=`**`false` environment variable when running **`start-server`**, as illustrated in the following example:

```
env PING_ENABLE_API_HTTP_CACHE=false bin/start-server
```

Note:

To temporarily re-enable HTTP caching after using the `--disableApiHttpCache`option, provide the **`PING_ENABLE_API_HTTP_CACHE=`**`true` environment variable when running **`start-server`**.

To persistently re-enable HTTP caching after using the `--disableApiHttpCache`option, delete the **`setup`** output (the `configuration.yml` file and key stores generated during **`setup`**). Then, reconfigure the server by running **`setup`**.

Policy administration

You define policies for access-control using the PingAuthorize Policy Editor.

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 PingAuthorize Server and the services that identify the REST APIs themselves.

To understand how PingAuthorize Server uses the Trust Framework, you must understand how PingAuthorize Server interacts with its policy engine, also called the policy decision point (PDP). In general, the flow is:

1. PingAuthorize Server receives a SCIM 2.0 or API request and translates it to a *policy request*.
2. PingAuthorize 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. PingAuthorize 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 PingAuthorize Server to reject a request because it originates from an untrusted IP address.
- A policy decision with the Exclude Attributes advice could cause PingAuthorize Server to remove specific attributes from an API response because the requesting user lacks a necessary entitlement.

Each policy request that PingAuthorize 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 169
- *Sideband API policy requests* on page 183
- *SCIM policy requests* on page 197

Policy request structure is tightly coupled to the Trust Framework. If the Trust Framework entity definitions do not match the policy requests generated by PingAuthorize Server, then PingAuthorize 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 411.

Trust Framework versions

The policy request structure used by PingAuthorize 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. PingAuthorize 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 PingAuthorize 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 PingAuthorize 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, PingAuthorize 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 136.

Create policies in a development environment

During policy development, configure PingAuthorize Server in external PDP mode where PingAuthorize Server forwards all policy requests to the Policy Editor, which acts as PingAuthorize Server's policy decision point (PDP).

Any policy changes made while using external PDP mode immediately take effect, allowing for rapid development and troubleshooting.

Develop policies in the PingAuthorize Policy Editor. To get started, see *Getting started with PingAuthorize (tutorials)* on page 17 or *Loading a policy snapshot* on page 264.

> **Note:**
>
> PingAuthorize Server does not function as expected without many of the Trust Framework entities defined by the `defaultPolicies.SNAPSHOT` file bundled with PingAuthorize Server. When developing new policies, begin by importing this snapshot and using it as the basis for your own customizations.

Configuring external PDP mode

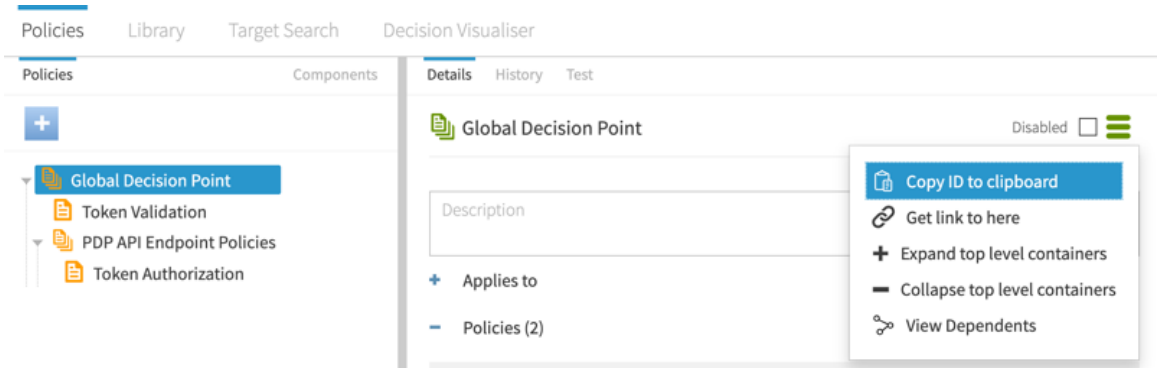
To configure PingAuthorize Server to use external PDP mode, use the administrative console or **`dsconfig`** to create a Policy External Server to represent the Policy Editor, then assign the Policy External Server to the Policy Decision Service and set the PDP mode.

Before you begin

You need the following values to configure PingAuthorizeServer to use external PDP mode:

- The shared secret, which is chosen or generated when you install the Policy Editor.
- The branch name, which corresponds to the policy branch you want to evaluate requests against in the Policy Editor.
- The decision node, which is the ID of a node in the policy tree that will be considered first during policy processing. To find the decision node value:
  1. In the Policy Editor, 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 hamburger menu and select **Copy ID to clipboard**.

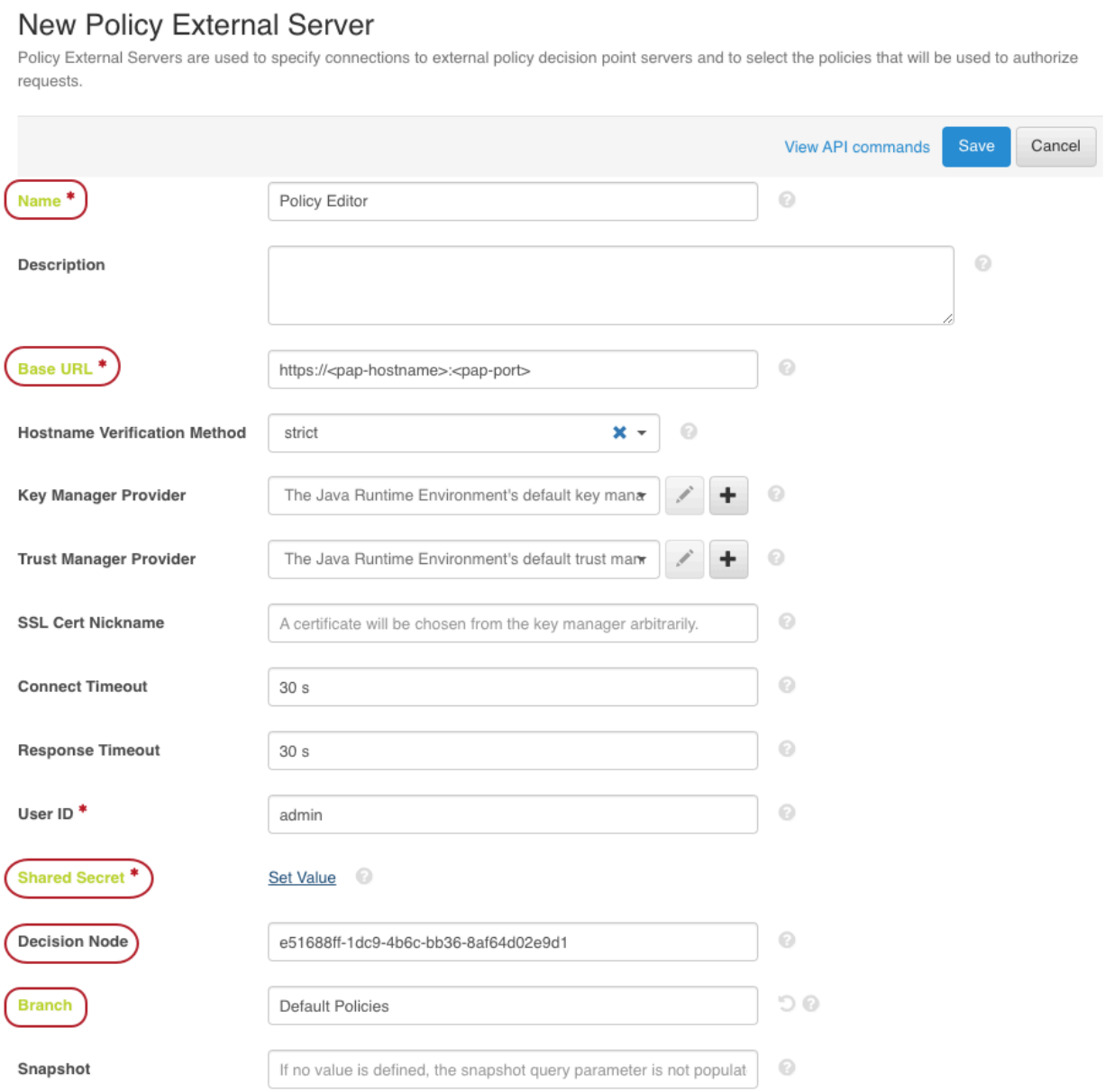


Configuring external PDP mode using the administrative console

Steps

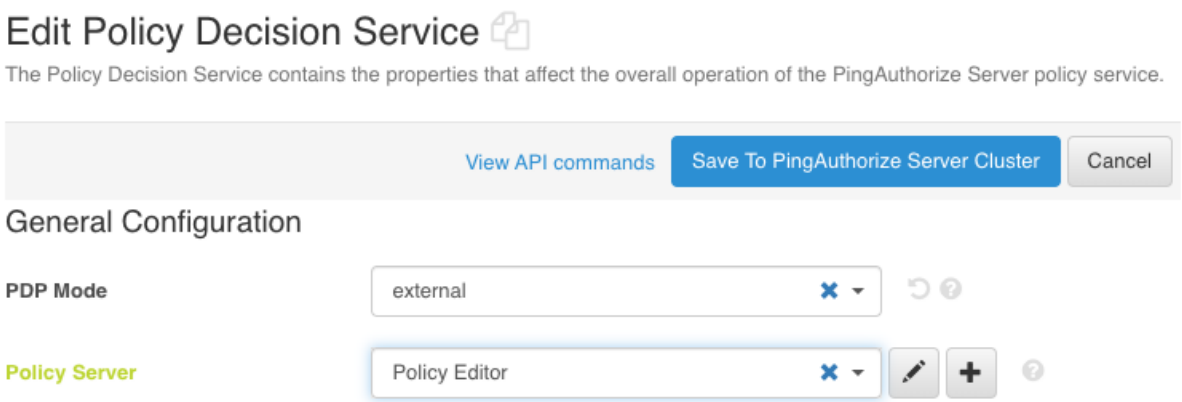
1. In the PingAuthorize administrative console, go to **Configuration** # **Data Sources** # **External Servers**.
2. Click **New External Server** and select **Policy External Server** from the drop-down menu.

3. In the **New Policy External Server** window, specify the following information:
   - **Name**
   - **Base URL**
   - **Shared Secret**
   - **Decision Node**
   - **Branch**



4. Click **Save**.
5. Go to **Authorization and Policies** # **Policy Decision Service**.
6. In the **PDP Mode** list, select **external**.

7. In the **Policy Server** list, select the name you gave to the policy external server in step 3.



8. Click **Save To PingAuthorize Server Cluster**.

Configuring external PDP mode using dsconfig

Steps

- Use the **`dsconfig`** commands in the following code block to configure external PDP mode:

```
dsconfig create-external-server \
  --server-name "Policy Editor" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:pingauthorize" \
  --set "branch:Default Policies" \
  --set "decision-node:<your decision node ID value>"

dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:Policy Editor"
```

Changing the active policy branch
The PingAuthorize Policy Editor can manage multiple sets of Trust Framework attributes and policies by storing data sets in different branches.

About this task

In a development environment, you might need to quickly reconfigure PingAuthorize Server between policy branches.

Steps

1. To set up branch changes, you must first *define a Policy External Server configuration* for each branch.
2. Change the Policy Decision Service's `policy-server` property as needed.

   Example:

   Assume that you have two policy branches in the Policy Editor: `Stable Policies` and `Experimental Policies`. Each branch is represented in the PingAuthorize 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 PingAuthorize Policy Editor. PingAuthorize includes a number of default and example policy snapshot files in the `resource/policies` directory. The following table describes the available snapshot files.

Snapshot filename	Description
`defaultPolicies.SNAPSHOT`	The default Trust Framework for PingAuthorize 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 PingAuthorize Server as an API security gateway. Based on *Getting started with PingAuthorize (tutorials)* on page 17.
`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 PingAuthorize (tutorials)* on page 17.

Importing and exporting policies
PingAuthorize supports two import and export file formats for Trust Framework and policy data.

About this task

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 Editor. A snapshot is used to load data into the Policy Editor 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 PingAuthorize Server when using embedded PDP mode.

The following sections describe how to import and export these files from the Policy Editor.

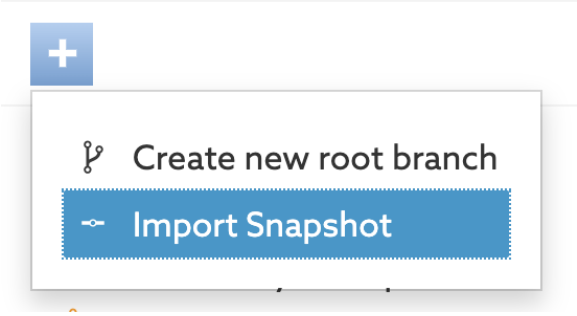
Loading a policy snapshot
To import a policy snapshot into the Policy Editor for policy development, complete the following steps.

About this task

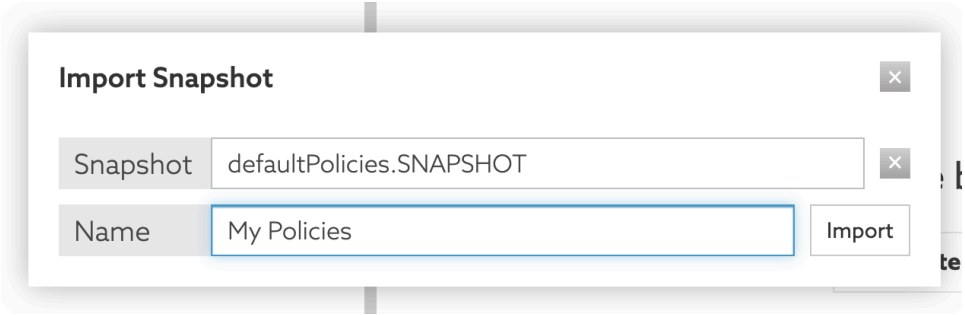
To create a new policy branch with the Trust Framework and policies of the provided snapshot:

Steps

1. Go to the **Branch Manager** section.
2. Click the **Version Control** tab.
3. In the **+** menu, select **Import Snapshot**.



4. Select a snapshot file and provide a name for your policy branch.



5. Optional: Click **Commit New Changes** to commit the initial state of the policy branch.

Exporting a policy snapshot

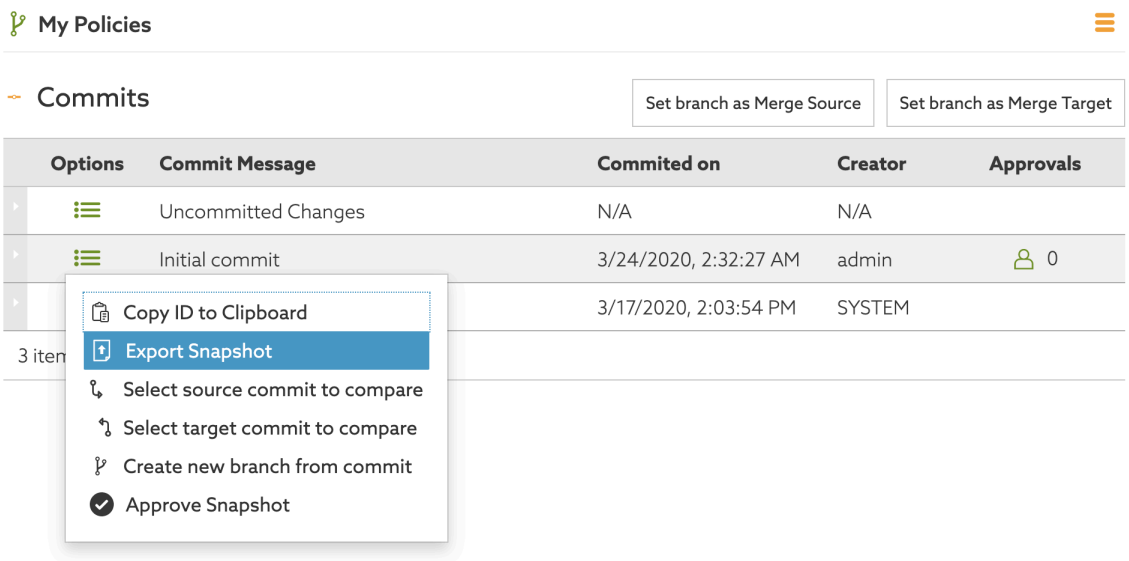
About this task

To import a policy snapshot into a different Policy Editor or use it as the basis to create a deployment package to be loaded in the PingAuthorize 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 icon in the **Options** column to the left of the commit message.

4. Select **Export Snapshot**.



5. Provide a snapshot filename and click **Export**.

Result

The snapshot file is downloaded to your computer.

Publishing a deployment package to a deployment package store
To use the Deployment Manager feature, create a deployment package and publish it to a deployment package store.

Before you begin

You must configure the Policy Editor to publish policies to your deployment package store using an options file.

For more information, see *Configuring the Policy Editor to publish policies to a deployment package store* on page 248.

About this task

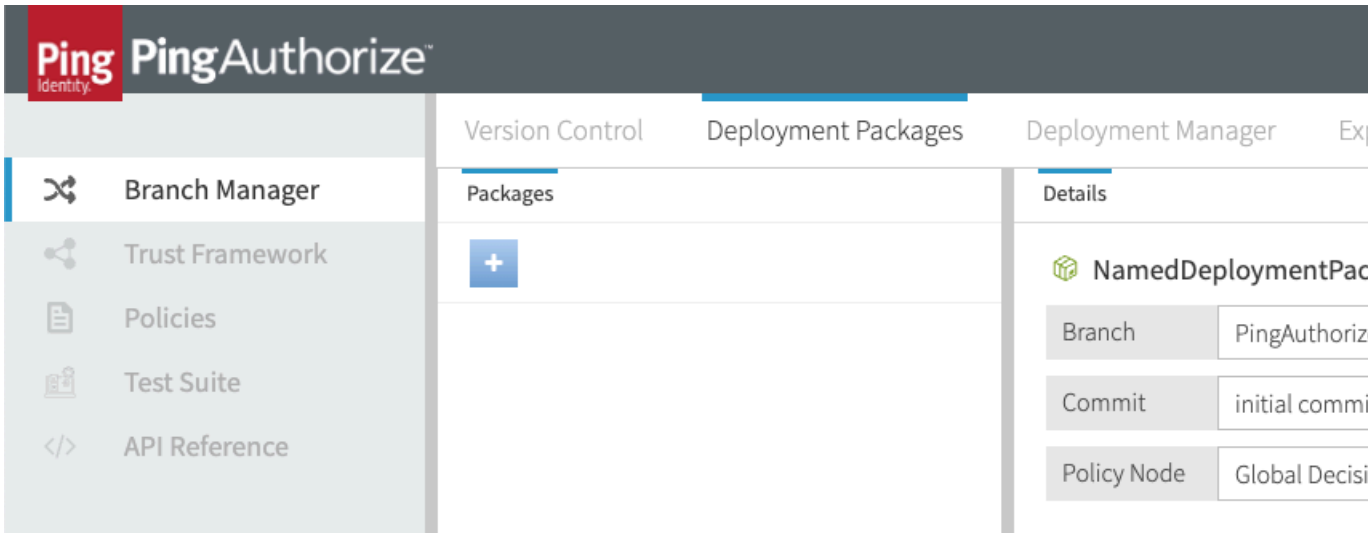
To publish deployment packages to a deployment package store:

Steps

1. Export a snapshot.

   For more information, see *Exporting a policy snapshot* on page 264.
2. Go to **Branch Manager** # **Deployment Packages**.
3. Click the **+** icon.
4. Enter a meaningful name for your deployment package.
5. In the **Branch** list, select a policy branch.
6. In the **Commit** list, select a commit.

7. In the **Policy Node** list, select a policy node.
   Example:



8. Click **Create Package**.
9. Click the **Deployment Manager** tab.
10. In the **Deployments** pane, select the deployment package store you want to publish the policies to.
11. In the **Available Packages** list, select the deployment package you want to publish to the deployment package store.
12. Click **Deploy**.

Next steps

Add the deployment package store to the PingAuthorize Server for read access. Based on your deployment package store configuration, add one of the following:

- *Add a filesystem deployment package store*.
- *Add an Amazon S3 deployment package store*.
- *Add an Azure deployment package store*.

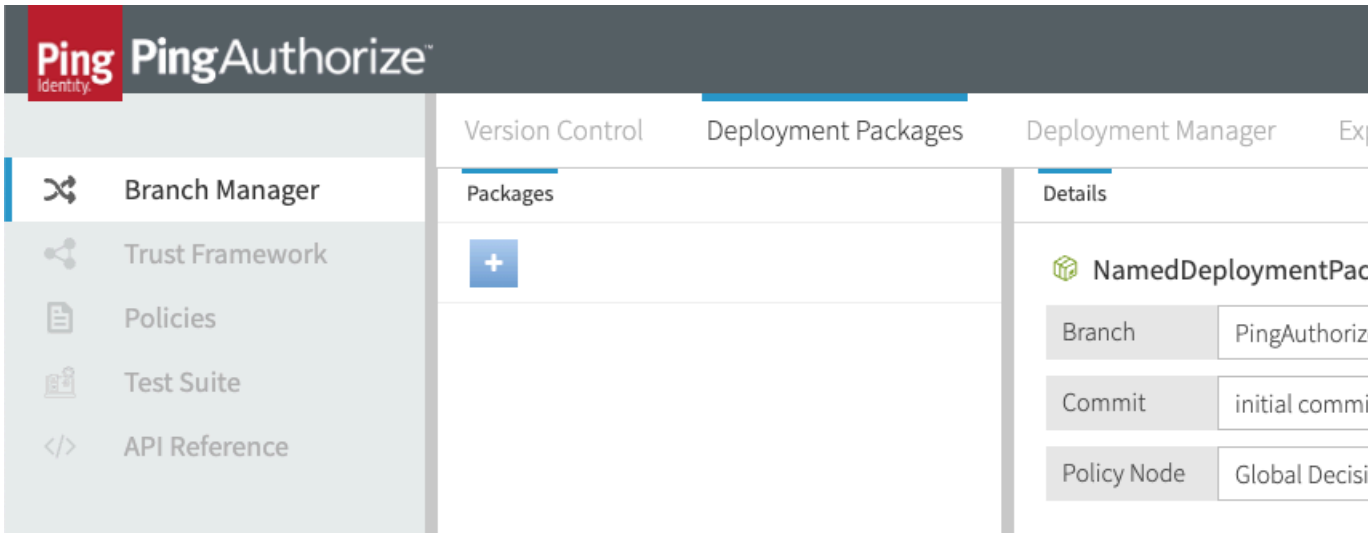
Exporting a deployment package
When you have completed development and testing of your policies, you can export your Trust Framework and policies to a deployment package for use in embedded PDP mode.

Steps

1. Export a snapshot.

   See *Exporting a policy snapshot* on page 264.
2. Go to the **Branch Manager** section.
3. Click the **Deployment Packages** tab.
4. Click the **+** icon.
5. Enter a meaningful name for your deployment package.
6. In the **Branch** list, select a policy branch.
7. In the **Commit** list, select a commit.

8. In the **Policy Node** list, select a policy node.
   Example:



9. Click **Create Package**.
10. Click **Export Package**.

Result

The deployment package is downloaded to your computer.

Using the Deployment Manager

The Deployment Manager simplifies policy updates by enabling policy writers to deploy new policies to a central deployment package store to be read by the PingAuthorize server running in embedded mode.

About this task

This process is two-fold:

- Policy writers use the Policy Editor to publish policies in a deployment package to a deployment package store.
- Updated deployment packages are picked up by the PingAuthorize Policy Decision Service from the deployment package store.

> **Note:**
> You configure the interval that the server checks for updates in the store during setup.

This allows a policy writer to deploy new policies without the manual process of exporting a deployment package that is then uploaded into the server through the administrative console.

The Deployment Manager can use deployment package stores that are based on:

- A directory in the filesystem
- An Amazon Simple Storage Service (Amazon S3) bucket
- Azure Blob storage

Package stores hold deployment packages in a central location that the Policy Editor publishes to and the PingAuthorize server reads from.

To use the Deployment Manager:

Steps

1. Define a deployment package store.

> **Note:**
>
> - For a filesystem store, you must have a directory on the filesystem that the Policy Editor has read-write access to.
> - Amazon S3 buckets must be configured with a secret key and an access key for use.
> - For Azure storage, you must set up an Azure storage account and a container. For later use, record the Connection string value found in your account's Access key settings.

2. *Use an options file to configure the Policy Editor to publish policies to a store*.
3. *Create and deploy deployment packages to the deployment package store*.
4. Add the deployment package store for read access to the PingAuthorize Server:
   a. *Add a filesystem deployment package store*.
   b. *Add an Amazon S3 deployment package store*.
   c. *Add an Azure deployment package store*,
5. *Configure the Policy Decision Service to read from your deployment package store*.

Adding a filesystem deployment package store
To use the Deployment Manager, add a deployment package store for read access to the PingAuthorize server.

About this task

Use the administrative console or **`dsconfig`** to add the deployment package store.

Adding a new filesystem deployment package store using the administrative console

Steps

1. In the administrative console, go to **Configuration** # **Authorization and Policies** # **Deployment Package Stores**.
2. Click **New Deployment Package Store**.
3. In the **New Deployment Package Store** list, select **Filesystem Deployment Package Store**.
4. Complete the **General Configuration** fields:
   a. In the **Name** field, enter a name for the deployment package store.
   b. In the **Poll Interval** field, enter a value in seconds for how often the directory should be polled for changes.

   > **Note:**
   >
   > A value of `0` only updates on start-up.

   c. In the **Poll Directory** field, enter the directory where the deployment package is stored locally.
5. Optional: Complete the **Policy Security** fields.

> **Note:**
>
> If you select **signed** in the **Deployment Package Security Level** field, you must complete the **Deployment Package Trust Store** field.

6. Click **Save To PingAuthorize Server Cluster**.
   Result:

   Your filesystem deployment package store is displayed on the **Deployment Package Stores** page.

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Adding a new filesystem deployment package store using dsconfig

Steps

- Run **`dsconfig`** with the `create-deployment-package-store` option:
  Choose from:
  - Create a store with an unsigned deployment package.

```
dsconfig create-deployment-package-store \
   --store-name "<store-name>"  \
   --type filesystem  \
   --set "poll-interval:<poll-interval>"  \
   --set "poll-directory:<filesystem-directory>"
```

  - Create a store with `deployment-package-security-level` set to `signed`.

```
dsconfig create-deployment-package-store \
  --store-name "<store-name>" \
  --type filesystem \
  --set "poll-interval:<poll-interval>" \
  --set deployment-package-security-level:signed \
  --set "deployment-package-trust-store:<trust-store-provider-name>" \
  --set "deployment-package-verification-key-nickname:<key-nickname>"
 \
  --set "poll-directory:<filesystem-directory>"
```

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Adding an Amazon S3 deployment package store
To use the Deployment Manager, add a deployment package store for read access to the PingAuthorize server.

About this task

Use the administrative console or **`dsconfig`** to add the deployment package store.

Adding an Amazon S3 deployment package store using the administrative console

Before you begin
You must set up an access key and accompanying secret key with your Amazon S3 bucket.

For information on setting up an access key and secret key, see your Amazon Web Services (AWS) documentation.

Steps

1. In the administrative console, go to **Configuration** # **Authorization and Policies** # **Deployment Package Stores**.
2. Click **New Deployment Package Store**.

3. In the **New Deployment Package Store** menu, select **S3 Deployment Package Store**.
4. Complete the **General Configuration** fields:
   a. In the **Name** field, enter a name for the deployment package store.
   b. In the **Poll Interval** field, enter a value in seconds for how often the Amazon S3 bucket should be polled for changes.

      Note:

      A value of `0` only updates on restart.

   c. In the **S3 Bucket Name** field, enter the name of your Amazon S3 bucket as shown on your AWS services page.
   d. In the **S3 Bucket Prefix** field, enter your Amazon S3 bucket prefix.
   e. In the **S3 Server Endpoint** field, enter your Amazon S3 bucket AWS endpoint.
   f. In the **S3 Region Name** field, enter the AWS region for your S3 bucket.
   g. Next to the **S3 Access Key ID** field, click **Set Value** and enter the S3 Access Key ID for your S3 bucket.
   h. Enter the S3 Access Key ID value again to confirm and click **OK**.

      Note:

      Your access key value is not displayed after you enter it. The page still displays **Set Value**.

   i. Next to the **S3 Secret Key** field, click **Set Value** and enter the S3 Secret Key for your S3 bucket.
   j. Enter the value again to confirm and click **OK**.

      Note:

      Your secret key value is not displayed after you enter it. The page still displays **Set Value**.

5. Optional: Complete the **Policy Security** fields.

   Note:

   If you select **signed** in the **Deployment Package Security Level** field, you must complete the **Deployment Package Trust Store** field.

6. Click **Save To PingAuthorize Server Cluster**.
   Result:

   Your Amazon S3 deployment package store is displayed on the **Deployment Package Stores** page.

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Adding an Amazon S3 deployment package store using dsconfig

Steps

- Run **`dsconfig`** with the `create-deployment-package-store` option:
  Choose from:
  - Create a store with an unsigned deployment package.

    ```
    dsconfig create-deployment-package-store \
    ```

```
  --store-name "<store-name>" \
  --type s3  \
  --set "poll-interval: <poll-interval>" \
  --set "s3-bucket-name:<bucket-name>"  \
  --set "s3-bucket-prefix:<bucket-prefix>"  \
  --set "s3-server-endpoint:<server-endpoint>"  \
  --set "s3-region-name:<region-name>"  \
  --set "s3-access-key-id:<access-key-id>"  \
  --set "s3-secret-key:<secret-key>"
```

- Create a store with `deployment-package-security-level` set to `signed`.

```
dsconfig create-deployment-package-store \
  --store-name "<store-name>"  \
  --type s3  \
  --set "poll-interval: <poll-interval>" \
  --set deployment-package-security-level:signed  \
  --set "deployment-package-trust-store:<trust-store-provider-name>"
 \
  --set "deployment-package-verification-key-nickname:<key-nickname>"
 \
  --set "s3-bucket-name:<bucket-name>"  \
  --set "s3-bucket-prefix:<bucket-prefix>"  \
  --set "s3-server-endpoint:<server-endpoint>"  \
  --set "s3-region-name:<region-name>"  \
  --set "s3-access-key-id:<access-key-id>"  \
  --set "s3-secret-key:<secret-key>"
```

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Adding an Azure deployment package store

To use the Deployment Manager, add a deployment package store for read access to the PingAuthorize server.

About this task

Use the administrative console or **`dsconfig`** to add the deployment package store.

Adding an Azure deployment package store using the administrative console

Before you begin

Set up your Azure storage account:

- If you don't already have an Azure storage account, create one.
- Add a container to your storage account.
- Record the Connection string value found in your account's Access key settings.

For information on setting up an Azure storage account, see your Azure Blob Storage documentation.

Steps

1. In the administrative console, go to **Configuration** # **Authorization and Policies** # **Deployment Package Stores**.
2. Click **New Deployment Package Store**.
3. In the **New Deployment Package Store** menu, select **Azure Deployment Package Store**.

4. Complete the **General Configuration** fields.
   a. In the **Name** field, enter a name for the deployment package store.
   b. In the **Poll Interval** field, enter a value in seconds for how often the Azure store should be polled for changes.

      Note:

      A value of `0` only updates on restart.

   c. In the **Azure Blob Connection String** field, enter the connection string shown in your Azure storage account's Access key settings.

      Note:

      Your connection string value is not displayed after you enter it. The page still displays **Set Value**.

   d. In the **Azure Blob Container** field, enter the name of your container.
   e. In the **Azure Blob Prefix** field, enter the *prefix you defined* for the deployment package store.
5. Optional: Complete the **Policy Security** fields.

   Note:

   If you select **signed** in the **Deployment Package Security Level** field, you must complete the **Deployment Package Trust Store** field.

6. Click **Save To PingAuthorize Server Cluster**.

   Result:

   Your Azure deployment package store is displayed on the **Deployment Package Stores** page.

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Adding an Azure deployment package store using dsconfig

Steps

- Run **`dsconfig`** with the `create-deployment-package-store` option:
  Choose from:
  - Create a store with an unsigned deployment package.

```
dsconfig create-deployment-package-store \
  --store-name "<store-name>" \
  --type azure  \
  --set "poll-interval:<poll-interval>" \
  --set "azure-blob-connection-string:<blob-connection-string>"  \
  --set "azure-blob-container:<blob-container>"  \
  --set "azure-blob-prefix:<blob-prefix>"
```

  - Create a store with `deployment-package-security-level` set to `signed`.

```
dsconfig create-deployment-package-store \
  --store-name "<store-name>"  \
  --type azure  \
  --set "poll-interval:<poll-interval>" \
  --set "azure-blob-connection-string:<blob-connection-string>"  \
  --set "azure-blob-container:<blob-container>"  \
```

```
  --set "azure-blob-prefix:<blob-prefix>"
  --set deployment-package-security-level:signed  \
  --set "deployment-package-trust-store:<trust-store-provider-name>"
\
  --set "deployment-package-verification-key-nickname:<key-nickname>"
```

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Use policies in a production environment

You can configure PingAuthorize 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 used in PingAuthorize Server's internal policy engine, which then handles all policy requests. The deployment package can be loaded into the server in two ways:

- The deployment package is deployed to a deployment package store, which is read by the internal policy engine for updates at a configurable interval.
- The deployment package is exported from the Policy Editor and loaded into the internal policy engine by an administrator.

Because embedded PDP mode does not require PingAuthorize Server to call out to an external server, it is considerably more performant than external PDP mode. To facilitate rapid policy development, you should use the Deployment Manager functionality that uses a deployment package store instead of the exported deployment package method.

Configure embedded PDP mode

To configure PingAuthorize Server to use embedded PDP mode, set the PDP mode and assign to the Policy Decision Service either:

- A deployment package store using the Deployment Manager functionality

> **Note:**
>
> For more information on the deployment package store option and the requirements for the Deployment Manager feature, see *Using the Deployment Manager* on page 267.

- An exported deployment package

> **Note:**
>
> For more information, see *Exporting a deployment package* on page 266.

Configuring embedded PDP mode with a deployment package store

About this task

To assign a deployment package store to the Policy Decision Service and set the policy decision point (PDP) mode to embedded:

Steps

- Use **dsconfig** or the administrative console:
  Choose from:
  - Run **dsconfig** with the `set-policy-decision-service-prop` option.

    ```
    dsconfig set-policy-decision-service-prop \
    --set pdp-mode:embedded \
    --set deployment-package-source-type:store \
    --set deployment-package-store:<name of the store>
    ```
  - Use the administrative console.
    1. In the administrative console, go to **Configuration** # **Authorization and Policies** # **Policy Decision Service** .
    2. On the **Edit Policy Decision Service** page, complete the **General Configuration**fields.
    3. In the **Deployment Package Store Configuration** section, in the **Deployment Package Store** field, select your deployment package store.
    4. In the **Policy Request Configuration** section, select a **Trust Framework Version**.
    5. Click **Save To PingAuthorize Server Cluster**.

Configuring embedded PDP mode with an exported deployment package

About this task

To assign an exported deployment package to the Policy Decision Service and set the PDP mode:

Steps

- Run **dsconfig** with the `set-policy-decision-service-prop` option.
  Example:

  In this example, the `deployment-package` value is the full path to a deployment package file. To create a deployment package for export, see *Exporting a deployment package* on page 266.

  ```
  dsconfig set-policy-decision-service-prop \
    --set pdp-mode:embedded \
    --set "deployment-package</path/to/my-deployment-
  package.deploymentpackage"
  ```

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 PingAuthorize 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 PingAuthorize 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 283.

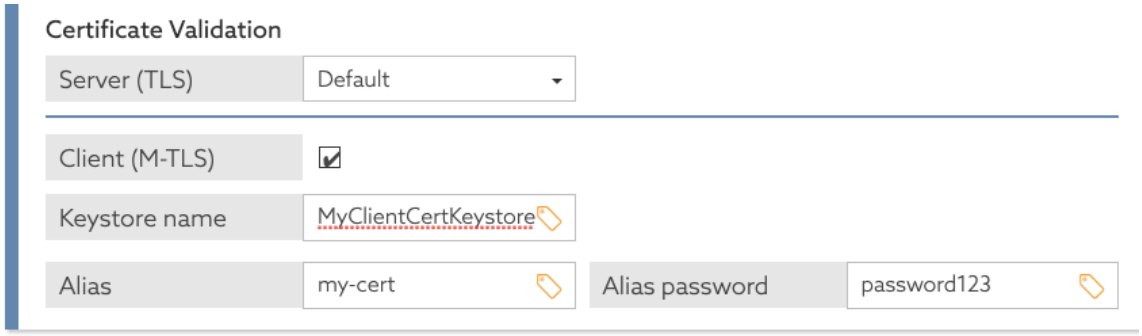
Example: Define a policy information provider key store for MTLS
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. In these cases, the policy engine can use a client certificate contained in a Java KeyStore (JKS) or PKCS12 key store.

When using embedded PDP mode, the key store containing the client certificate is represented in the PingAuthorize 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`.



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.

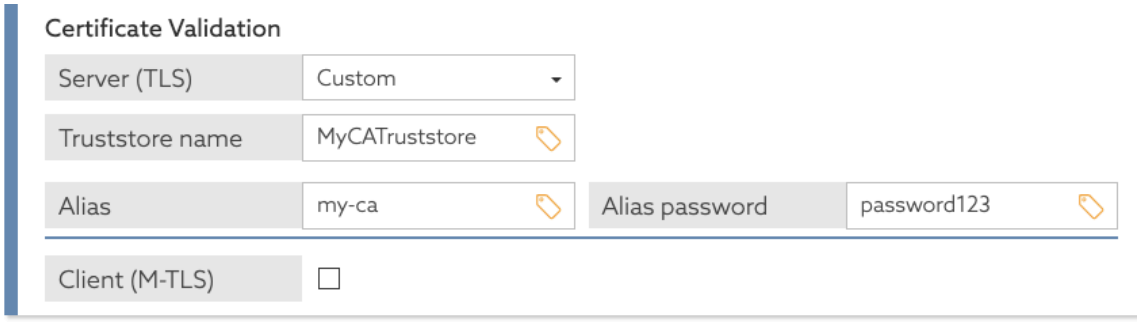
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. By default, the policy engine determines whether it should accept a PIP's server certificate using the Java Runtime Environment (JRE)'s default trust store, which contains public root certificates 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 PingAuthorize 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`.



Example: Add SpEL Java classes to the allowed list
When you develop policies, you can use SpEL expressions in your deployment packages. Configure the Java classes used during SpEL expression evaluation by adding classes to the allowed list.

When using embedded PDP mode, the policy engine allows use of the following classes by default.

```
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
```

Use **`dsconfig`** or the administrative console to add non-standard classes to the allowed list. In the administrative console, you can find SpEL allowed classes in the Policy Decision Service configuration.

Example

The following example shows how to add the `java.time.format.DateTimeFormatter` and `java.util.Base64` classes to the allowed list. Run **`dsconfig`** with the `set-policy-decision-service-prop` option.

```
dsconfig set-policy-decision-service-prop \
 --set spel-allowed-class:java.time.format.DateTimeFormatter \
 --set spel-allowed-class:java.util.Base64
```

Important:

After you add non-standard classes to the allowed list, you must make them available on the server classpath at server start.

For more information, see *Adding non-standard Java classes to the server classpath*.

Example: Add non-standard Java classes to the server classpath
After you add non-standard SpEL Java classes to the allowed list, you must make them available on the server classpath at server start.

Example

The following example shows how to add `.jar` files containing the classes to the `lib` folder and restart the server.

```
cd <paz-instance-root>
cp <jar-file-dir>/addl-spel-classes.jar lib
bin/stop-server -R
```

Policy database backups

The PingAuthorize Policy Editor uses a policy database to store its Trust Framework, policies, commit history, and other data needed for proper operation.

By default, the Policy Editor backs up the policy database to a compressed file once a day by making an HTTP request to an admin connector. You can configure the admin port, backup schedule, and output location.

Note:

If you are using a managed RDBMS, such as PostgreSQL, instead of the default H2 database, make sure you implement backup strategies in line with your organization's best practices.

Configure or disable backup

To change the backup configuration, you can:

- Set the relevant environment variables and restart the Policy Editor.
- Run the Policy Editor **`setup`** tool with the relevant command-line options.

The following table describes the relevant environment variables and command-line options.

For more information about using the environment variables, see *Starting PingAuthorize Policy Editor* on page 162.

Environment variable	Command-line option	Description
PING_ADMIN_PORT	`--adminPort <port>`	Specifies the admin port, where administrative task endpoints like periodic policy database backups are handled.
PING_BACKUP_SCHEDULE	`--backupSchedule <cron-expression>`	Specifies a **cron** expression to indicate when to perform backups. The default is `0 0 0 * * ?`, which is midnight every day. For more information, see *Quartz 2.3.0 cron format*. **Note:** The PAP evaluates the expression against the system timezone. For the PingAuthorize Docker images, the default timezone is UTC.
PING_H2_BACKUP_DIR	N / A	Specifies the directory in which to place the policy H2 database backup files. The default is **SERVER_ROOT**/`policy-backup`. **Note:** If you are using a Docker image, set this value to a directory on a volume that you mount when you start the Docker container.
N / A	`--disablePeriodicBackups`	Turns off the periodic policy database backups.

For information about how to use a backup, see *Restoring a policy database from a backup* on page 278.

Restoring a policy database from a backup

The policy database stores PingAuthorize Policy Editor items such as the Trust Framework, policies, and commit history. If someone accidentally deletes or changes those items or the database gets corrupted, restore the database from a backup.

For information about how to configure backups, see *Policy database backups* on page 277.

Note:

If you are using a managed RDBMS, such as PostgreSQL, instead of the default H2 database, make sure you implement backup strategies in line with your organization's best practices.

Restoring a database when not using Docker

About this task

To restore a policy database when not in a Docker environment:

Steps

1. Ensure the Policy Editor server is no longer running by either using `bin/stop-server` or killing the process.
2. Locate the backup `.zip` file that you want to restore.

   The default location is **`SERVER_ROOT`**`/policy-backup`. However, the location might have been changed using the **`PING_H2_BACKUP_DIR`** environment variable.
3. Extract the `.zip` file to the configured database location overwriting the previous policy database file, if present.

   The default location is the root of the Policy Editor server installation directory. If it's not there, check the location specified by the **`PING_H2_FILE`** environment variable.
4. Start the Policy Editor server.

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

Restoring a database when using Docker

About this task

To restore a policy database in a Docker environment:

Steps

1. Locate the backup `.zip` file that you want to restore.

   The location should be a directory specified using the **`PING_H2_BACKUP_DIR`** environment variable, as mentioned in *Policy database backups* on page 277.
2. Extract the `.zip` file to the database location that you will specify using the **`PING_H2_FILE`** environment variable when you start the Docker container.
3. Start the Docker container with a mounted volume that has the extracted backup file and use **`PING_H2_FILE`** to specify that backup file in the container file system.

   For example, the following command assumes the uncompressed database file is named `Symphonic.mv.db` in the host file system. The **`PING_H2_FILE`** environment variable specifies the file name without the `.mv.db` extension.

   ```
   $ docker run --network=<network_name> --env-file ~/.pingidentity/config \
     --env PING_H2_FILE=/opt/out/Symphonic \
   ```

```
--volume <HOST_BACKUP_DIR>:/opt/out pingidentity/pingauthorizepap:<TAG>
```

Tip:

For proper communication between containers, create a Docker network using a command such as `docker network create --driver <network_type> <network_name>`, and then connect to that network with the `--network=<network_name>` option.

Note:

The Docker image <*TAG*> used in the example is only a placeholder.For actual tag values, see *Docker Hub*.

Policy application management with signed deployment packages

Signed deployment packages ensure a PingAuthorize Server uses only deployment packages from a certain PingAuthorize Policy Editor, allowing you to avoid the use of packages intended for a different context or to use packages from only a designated source.

Use case: Distinct PingAuthorize deployments

Consider an organization with two distinct PingAuthorize deployments: healthcare and banking. Each deployment has a unique set of policies. Using the healthcare policies for the banking deployment, or vice versa, would make the deployment ineffective. Signed deployment packages avoid this issue. To set up signed deployment packages for these two deployments, the steps are outlined next.

1. Set up the healthcare configuration.
    a. Create a signing key pair with a private key and a public key for healthcare.
    b. Set up a Policy Editor to create all healthcare policies. Configure that GUI to sign its deployment packages with the healthcare private key.
    c. Configure the healthcare PingAuthorize Server to use the healthcare public key to verify deployment packages. Now the healthcare deployment only accepts healthcare policies and does not accept banking policies.
2. Set up the banking configuration.
    a. Create a signing key pair with a private key and a public key for banking.
    b. Set up a Policy Editor to create all banking policies. Configure that GUI to sign its deployment packages with the banking private key.
    c. Configure the banking PingAuthorize Server to use the banking public key to verify deployment packages. Now the banking deployment only accepts banking policies and does not accept healthcare policies.

Use case: Designated source for deployment packages

An organization has several people who write policies. Each policy writer has their own Policy Editor to develop and test policies. However, to ensure the organization fully verifies each deployment package before it goes into preproduction or production, only one Policy Editor can actually sign deployment packages with the key accepted by the PingAuthorize Server.

Example: Configure signed deployment packages for healthcare
In this example, you configure a PingAuthorize Policy Editor to sign its deployment packages for a PingAuthorize Server dedicated to healthcare policies.

Note:

This example uses the **`manage-certificates`** tool that comes with PingAuthorize. The tool provides many of the same features as the Java **`keytool`** utility but can be easier to use. If you prefer to use **`keytool`**, use **`manage-certificates --display-keytool-command`** to show a command you can use to obtain a similar result with **`keytool`**.

1. Generate a signing key pair for the Policy Editor.

   Create a key pair consisting of a private key and the corresponding public key. Put the key pair in a key store so that the Policy Editor can use it. The following command accomplishes both of these goals by generating a key store with a self-signed certificate.

   ```
   $ manage-certificates generate-self-signed-certificate \
     --keystore "healthcare-pap-signing.jks" \
     --keystore-type jks \
     --keystore-password "<keystore-password>" \
     --private-key-password "<private-key-password>" \
     --alias "healthcare-pap" \
     --subject-dn "cn=Healthcare PAP,dc=example,dc=com" \
     --days-valid 90
   ```

   - This command creates a key store with the filename `healthcare-pap-signing.jks`. The Policy Editor uses this to sign deployment packages.
   - The key store contains the Policy Editor's private signing key and the corresponding public key.
   - The key store itself has the password *<keystore-password>*.
   - The private key itself also has a password, *<private-key-password>*.
   - The signing key pair has the nickname/alias `healthcare-pap`.
   - The subject DN is arbitrary.
   - The keys are valid for 90 days.
   - This key store is a sensitive asset that you should carefully protect.
2. Export a public certificate from the Policy Editor's key store.

   ```
   $ manage-certificates export-certificate \
     --keystore "healthcare-pap-signing.jks" \
     --keystore-password "<keystore-password>" \
     --alias "healthcare-pap" \
     --export-certificate-chain \
     --output-format pem \
     --output-file "healthcare-pap.pem"
   ```

   - This command creates a public certificate file with the filename `healthcare-pap.pem`.
   - The public certificate file is an input during the next step. It is not used directly by either the Policy Editor or PingAuthorize Server.
   - This public certificate represents the public key created in the previous step.

     Note:

     The alias is used to specify the key.

   - This public certificate is not a sensitive asset.
3. Create a trust store for PingAuthorize Server for the public certificate from the previous step.

   ```
   $ manage-certificates import-certificate \
     --keystore "healthcare-pap-verification.jks" \
     --keystore-password "<keystore-password>" \
     --keystore-type jks \
     --alias "healthcare-pap" \
     --certificate-file "healthcare-pap.pem" \
   ```

```
--no-prompt
```

- This command creates a trust store with the filename `healthcare-pap-verification.jks`. PingAuthorize Server uses this to verify that deployment packages created by the Policy Editor were actually created by that GUI.
- The trust store contains the Policy Editor's public certificate.
- The trust store itself has the password *<truststore-password>*.
- This trust store is not a sensitive asset.

4. Configure the Policy Editor to use the key store to sign the deployment packages it creates.
   a. Make a copy of the default options file.

   ```
   $ cp config/options.yml my-options.yml
   ```

   b. Edit the new options file to include a configuration block like the following one, substituting your passwords and other values. Place this new block at the top level, parallel to the `core` block, either before or after it.

   ```
   deploymentPackageData:
     contentType: json
     keystore:
       resource: /path/to/healthcare-pap-signing.jks
       password: keystore-password
     securityLevel: signed
     signingKey:
       alias: healthcare-pap
       password: private-key-password
   ```

   c. Stop the Policy Editor.

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

   d. Run **`setup`** using the `--optionsFile my-options.yml` argument. Customize all other options as appropriate for your needs.
   e. Start the Policy Editor.

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

5. Configure the PingAuthorize Server to use the trust store for verification so that it accepts only deployment packages created by this Policy Editor.
   a. Create a trust manager provider, which is how the PingAuthorize Server configuration refers to a trust store file. Include the path to the trust store file and the trust store's password.

   ```
   $ dsconfig create-trust-manager-provider \
     --provider-name "Healthcare PAP Verification Store" \
     --type file-based \
     --set enabled:true \
     --set "trust-store-file:/path/to/healthcare-pap-verification.jks" \
     --set trust-store-type:JKS \
     --set "trust-store-pin:<truststore-password>"
   ```

   b. Configure the policy decision service.

   ```
   $ dsconfig set-policy-decision-service-prop \
     --set pdp-mode:embedded \
     --set "deployment-package</path/to/deployment-package.deploymentpackage" \
     --set deployment-package-security-level:signed \
     --set "deployment-package-trust-store:Healthcare PAP Verification Store" \
   ```

```
  --set "deployment-package-verification-key-nickname:healthcare-pap"
```

Deployment packages are only for the embedded PDP mode, so this command sets the `pdp-mode` property accordingly. The other properties are described in the following table.

Property	Description
`deployment-package-security-level`	Determines whether PingAuthorize Server require a deployment package to be signed. Valid values are: ▪ `unsigned` (the default) PingAuthorize Server does not check a deployment package for a trusted signature. ▪ `signed` PingAuthorize Server checks a deployment package for a trusted signature and rejects a deployment package that fails that check. Whenever a deployment package fails a check, PingAuthorize Server continues to use the last accepted deployment package.
`deployment-package-trust-store`	Specifies a trust manager provider, which specifies in turn a trust store containing a Policy Editor's public certificate. This property is required if `deployment-package-security-level` is `signed`.
`deployment-package-verification-key-nickname`	Specifies the nickname or alias of the Policy Editor's public certificate. This property is required if `deployment-package-security-level` is `signed`.

Note:

For more information about the properties, see the *Configuration Reference* located in the server's `docs/config-guide` directory.

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.

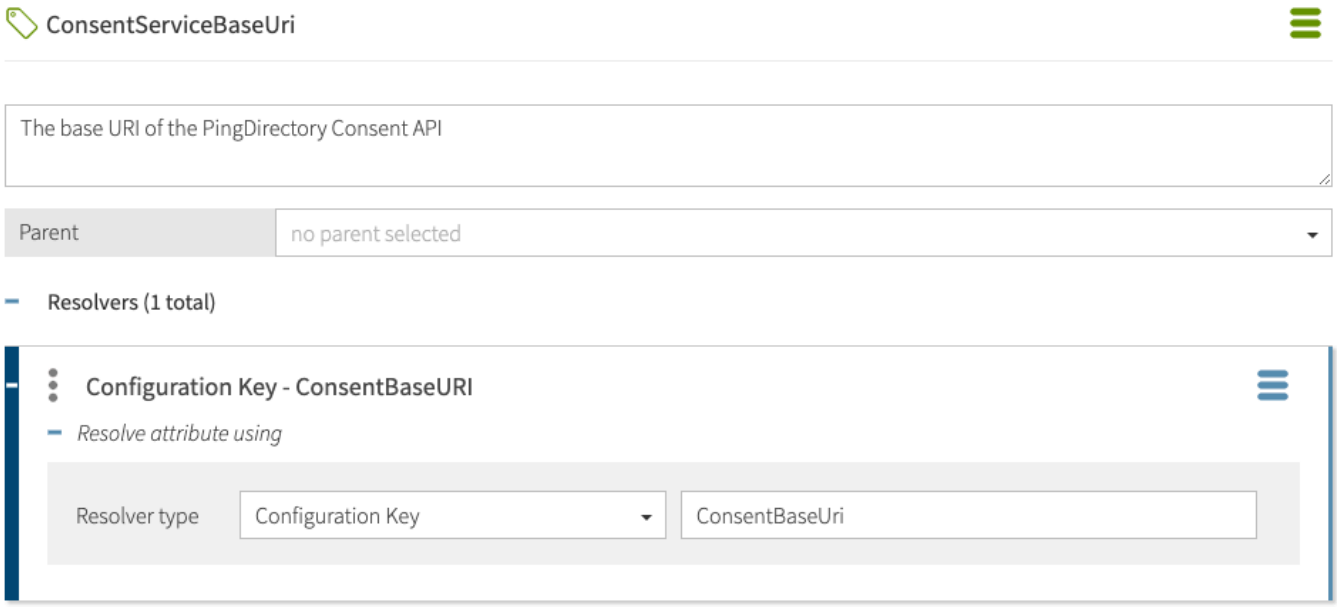
For example, you might use a Trust Framework service to retrieve a user's consent from the PingDirectory Consent API. This service depends on the URL of the Consent API, the username and password that are used for authentication, and other items that vary between development, preproduction, and production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



The means by which policy configuration keys are provided to the policy engine differ based on whether the PingAuthorize 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 Editor's **`setup`** tool. See *Example: Configure policy configuration keys* on page 239.
Embedded PDP mode	The PingAuthorize Server configuration. See *Example: Define policy configuration keys* on page 274.

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:

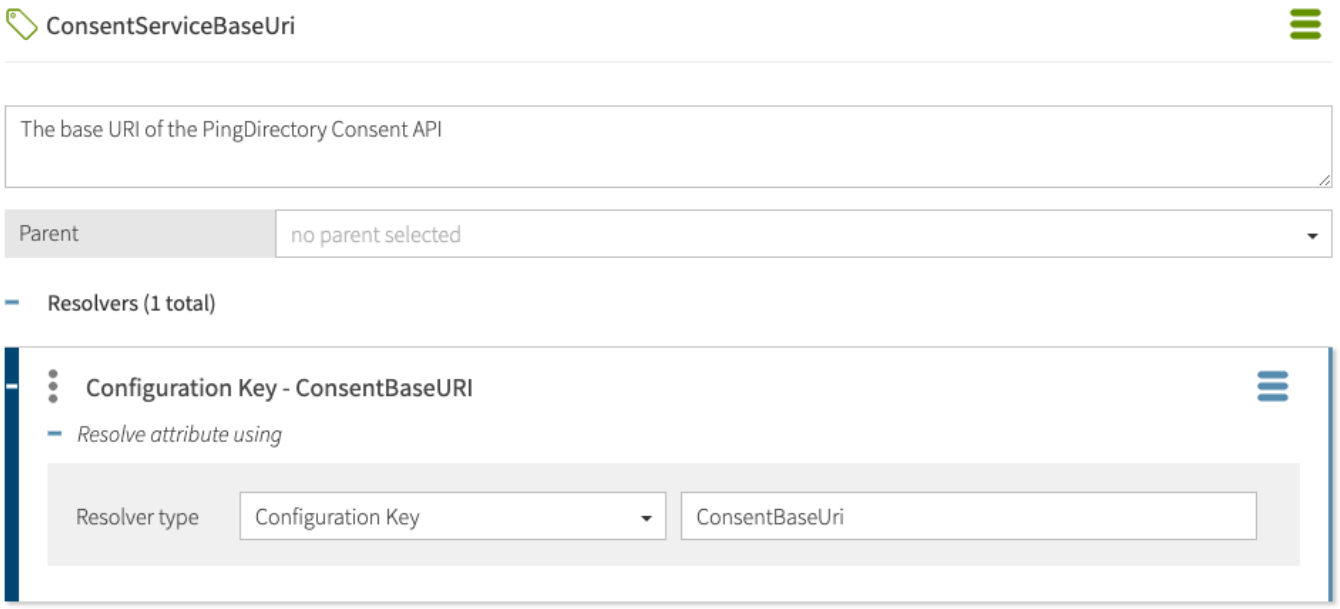
Policy configuration key	Description
`ConsentBaseUri`	The base URL to use when making requests to the Consent API.
`ConsentUsername`	The username for a privileged Consent API account.
`ConsentPassword`	The password for a privileged Consent API account.

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.



2. Repeat the previous steps for `ConsentUsername` and `ConsentPassword`.
   Result:

   When complete, you should have defined the following attributes.

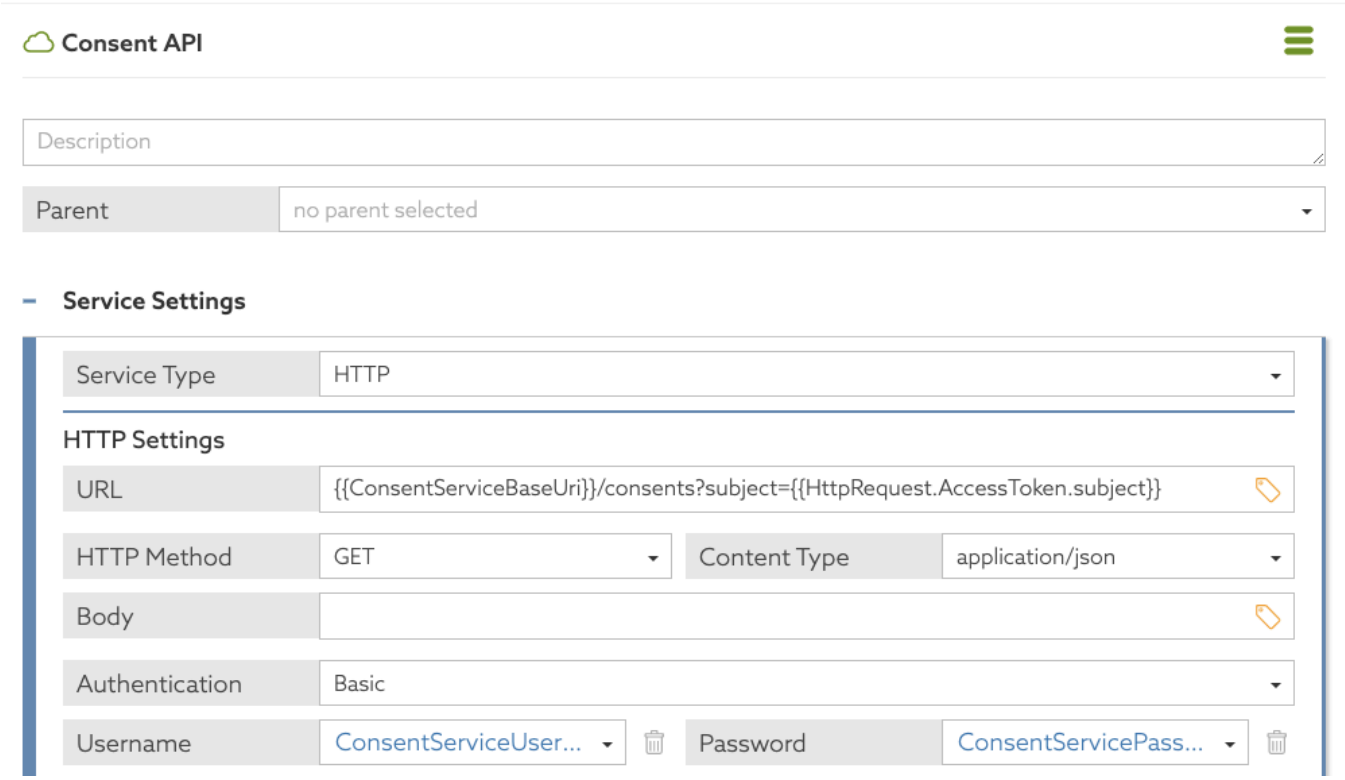
Attribute name	Policy configuration key name
`ConsentServiceBaseUri`	`ConsentBaseUri`
`ConsentServiceUsername`	`ConsentUsername`
`ConsentServicePassword`	`ConsentPassword`

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.



   Result:

   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 Editor to provide values for the PIP's base URL, username, and password.

About this task

To configure the Policy Editor 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.
   Example:

   ```
   $ cp config/options.yml my-options.yml
   ```

2. Edit the new options file and define the policy configuration keys in the `core` section.
   Example:

   ```
   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 Editor.

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

4. Run **`setup`** using the `--optionsFile` argument, and then customize all other options as appropriate for your needs.
   Example:

   ```
   $ bin/setup demo \
    --adminUsername admin \
    --generateSelfSignedCertificate \
    --decisionPointSharedSecret pingauthorize \
    --hostname <pap-hostname> \
    --port <pap-port> \
    --adminPort <admin-port> \
    --licenseKeyFile <path-to-license> \
    --optionsFile my-options.yml
   ```

5. Start the Policy Editor.
   Example:

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

Define policy configuration keys in a preproduction environment
Do not use the Policy Editor in a pre-production or production environment. Define policy configuration keys in the PingAuthorize Server configuration.

About this task

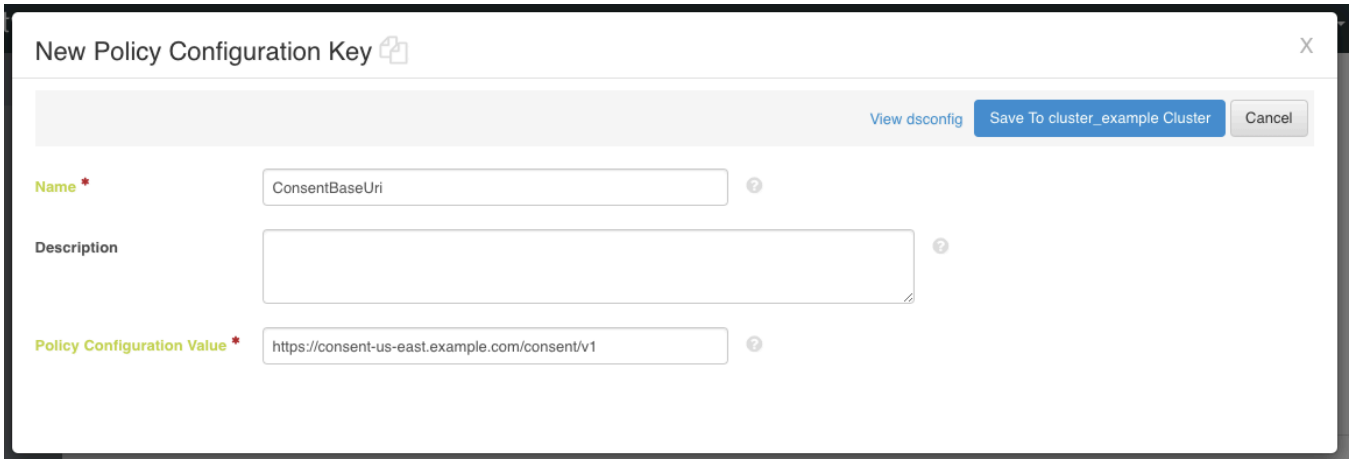
To define policy configuration keys, use either **`dsconfig`** or the administrative console, as shown in the following steps.

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.



3. Save the policy configuration key.
4. Repeat the previous steps for the policy configuration keys `ConsentUsername` and `ConsentPassword`.

User profile availability 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. PingAuthorize can automatically look up the token owner's attributes and provide them in the policy request using a token resource lookup method.

Token resource lookup methods

PingAuthorize provides built-in support for retrieving token owner data using *SCIM token resource lookup methods* on page 299. 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 193.

For examples that show how to set up a token resource lookup method, see:

- *Configuring the PingAuthorize OAuth subject search* on page 360
- *Sideband access token validation* on page 190
- *SCIM token resource lookup methods* on page 299

User profile data from access tokens

When processing an incoming HTTP request, PingAuthorize 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 PingAuthorize Server includes a `TokenOwner` attribute with the policy request. The contents of the `TokenOwner` attribute are a JSON object containing the user profile.

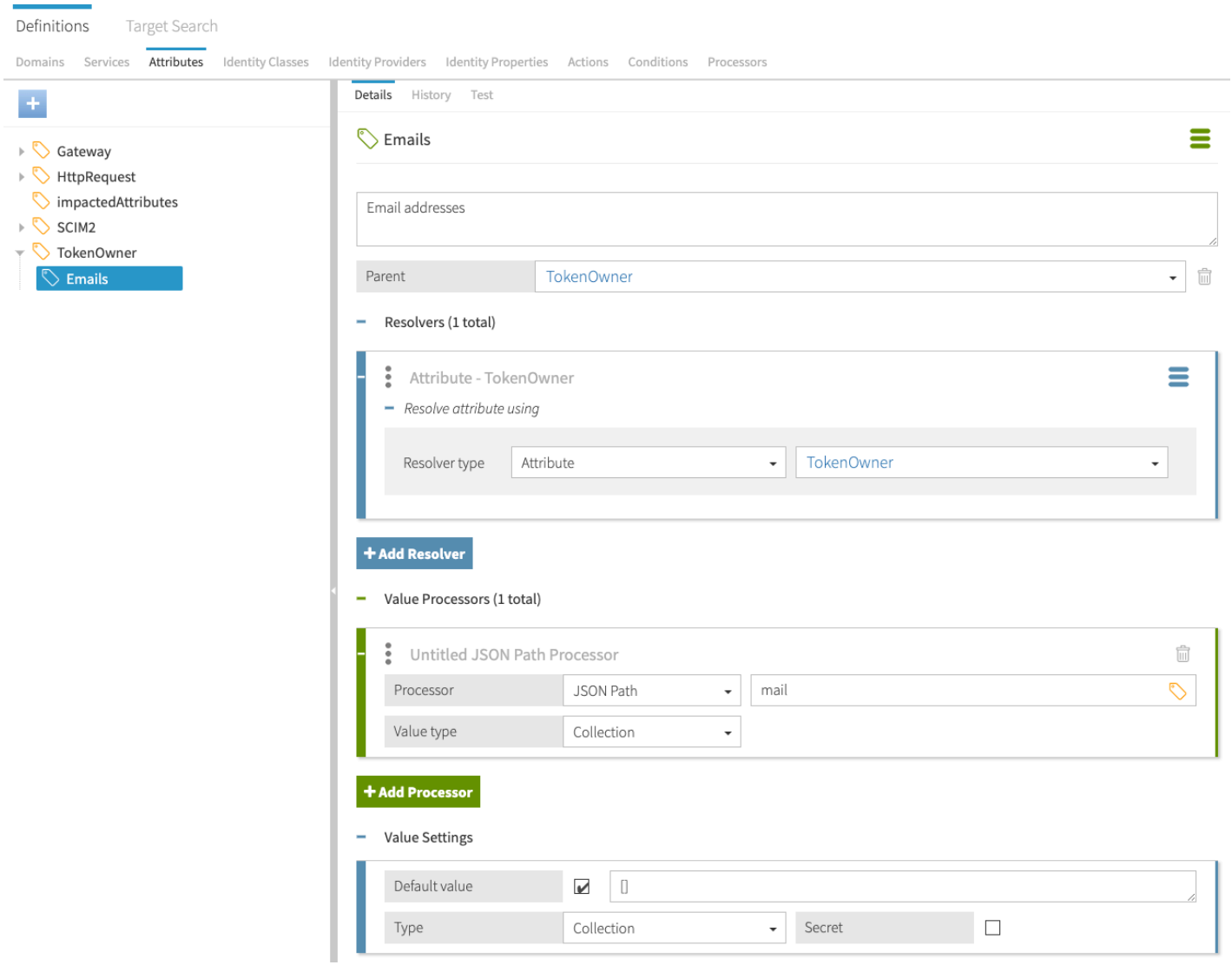
The exact structure of the `TokenOwner` attribute varies from deployment to deployment. When using a SCIM token resource lookup method, the contents of the `TokenOwner` attribute are a SCIM resource using the schema of the SCIM resource type configured for the token resource lookup method, exactly as 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 PingAuthorize Policy Editor, as shown in the following Trust Framework image.



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 PingAuthorize 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, PingAuthorize Server uses access token validators, which determine whether to accept an access token and translate it into a set of properties, called claims.

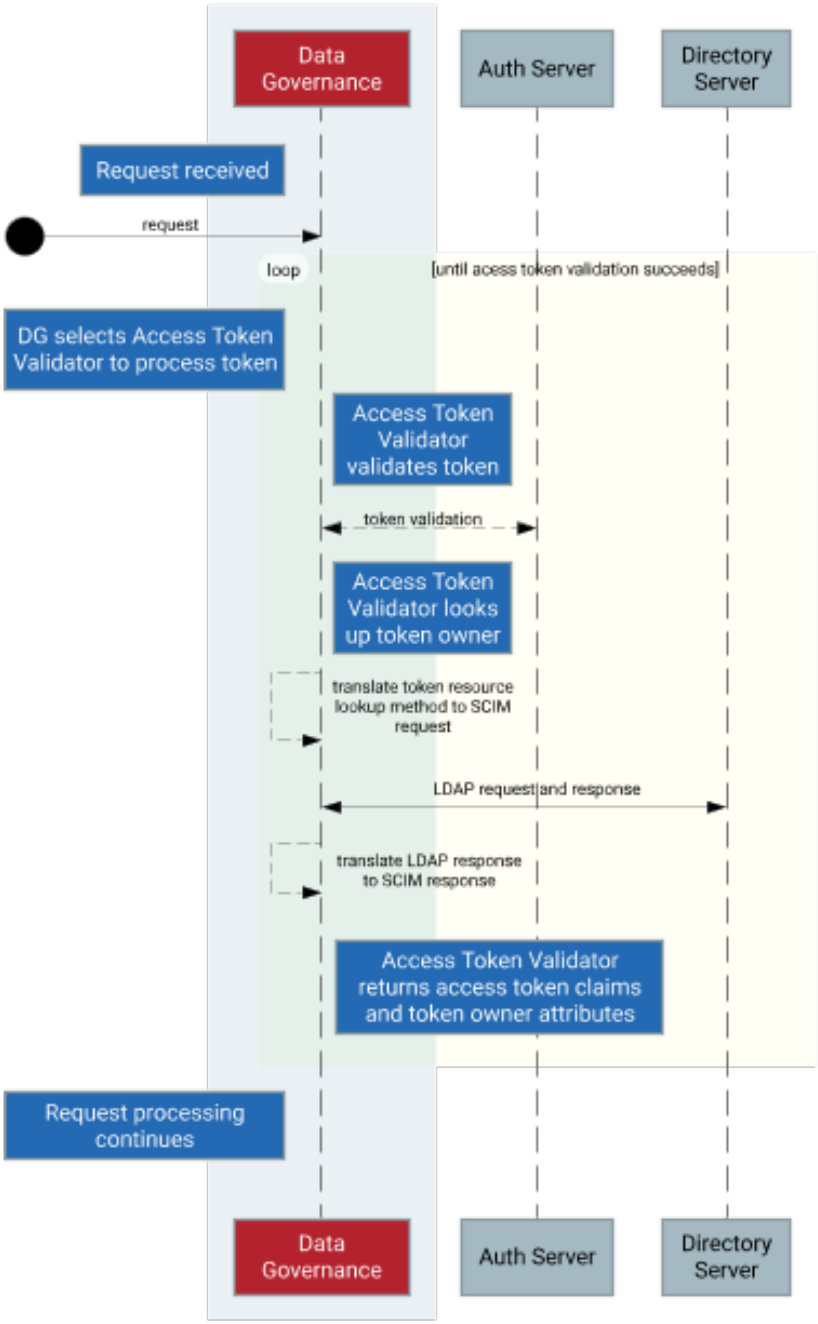
Most access tokens identify a user, also called the token owner, as its subject. Access token validators can retrieve the token owner's attributes from the user store using a related component called a token resource lookup method. The user data obtained by a token resource lookup method is sent to the policy decision point (PDP) so that policies can determine whether to authorize the request.

For more information about the types of access tokens PingAuthorize can validate, see *Access token validator types* on page 292.

About access token validator processing

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 System for Cross-domain Identity Management (SCIM) token resource lookup method.



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

2. The access token validator validates the access token.

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

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

   If the token is valid, its `active` flag is set to `true`. The flag and other access token claims are added to the `HttpRequest.AccessToken` attribute of the policy request.
3. If the access token contains a subject, the access token validator sets the `user_token` flag to `true`, and uses a token resource lookup method to fetch the token owner through SCIM.

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

   > **Note:**
   >
   > For deployments that don't 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 *User profile availability in policies* on page 288.
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 PingAuthorize 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

PingAuthorize Server works with many different types of access token validators.

Click the tabs to learn more about the following types of access token validators:

- PingFederate
- JSON web token (JWT)
- Mock access token
- Third-party
- External API gateway

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.

> **Note:**

If you are using PingFederate 10.0 or earlier, ensure that PingFederate is configured to respond to OAuth and OpenID Connect (OIDC) requests by selecting the **Enable OAuth 2.0 Authorization Server (AS) role** and **OpenID Connect** check boxes, as explained in *Enabling the OAuth AS role*. Starting with PingFederate 10.1, these items are always enabled.

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

Example PingFederate access token validator configuration

In PingFederate, create a client with the following properties:

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

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

```
# 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:PingAuthorize \
  --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 JWT access token validator verifies access tokens that are encoded in JSON web token (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. Because the JWT access token validator doesn't 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
- ES256
- ES384
- ES512

For encrypted tokens, the JWT access token validator supports the following key-encryption algorithms:

- RSA-OAEP
- ECDH-ES
- ECDH-ES+A128KW
- ECDH-ES+A192KW
- ECDH-ES+A256KW

For encrypted tokens, the JWT access token validator supports the following content-encryption algorithms:

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

The JWT access token validator configuration defines three *[[allow lists]]* for the JWS/JWE signing and encryption algorithms that it will accept. You should customize these allow lists to reflect only the signing and encryption algorithms used by your access token issuer and no others. Doing so minimizes the access token validator's security threat surface.

Configure these allow lists using the following configuration properties.

Property	Description
`allowed-signing-algorithm`	Specifies the signing algorithms that the access token validator accepts.
`allowed-key-encryption-algorithm`	Specifies the key-encryption algorithms that the access token validator accepts.
`allowed-content-encryption-algorithm`	Specifies the content-encryption algorithms that the access token validator accepts.

Handling signed tokens

About this task

All access tokens the JWT access token validator handles must be cryptographically signed by the token issuer. The JWT access token validator validates a token's signature using a public signing key provided by the issuer.

Steps

- Configure the JWT access token validator with the issuer's public signing key in one of the following ways:
  Choose from:
  - Store the public key as a trusted certificate in PingAuthorize Server's local configuration using the `trusted-certificate` property.
  - Provide the issuer's JSON Web Key Set (JWKS) endpoint using the `jwks-endpoint-path` property. The JWT access token validator then retrieves the issuer's public keys when it initializes. This method ensures that the JWT access token validator uses updated copies of the issuer's public keys.

Example: Use a locally configured trusted certificate

Example

The following example configures a JWT access token validator to use a locally stored public signing certificate to validate access token signatures. The signing certificate is assumed to have been obtained out of band and must be a PEM-encoded X.509v3 certificate.

```
# Add the public signing certificate to the server configuration
dsconfig create-trusted-certificate \
  --certificate-name "JWT Signing Certificate" \
  --set "certificate</path/to/signing-certificate.pem"

# 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 allowed-signing-algorithm:RS256 \
  --set "trusted-certificate:JWT Signing Certificate"

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

Example: Use the issuer's JWKS endpoint

Example

The following example configures a JWT access token validator 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 allowed-signing-algorithm:RS256 \
  --set "authorization-server:PingFederate External Server" \
  --set jwks-endpoint-path:/ext/oauth/jwks

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

Handling encrypted tokens

About this task

The JWT access token validator can accept encrypted access tokens. To enable this functionality, you must configure the access token validator with a private/public key pair and provide the public key to the token issuer.

The examples in each step configure a JWT access token validator to handle access tokens signed and encrypted using elliptic curve algorithms. For RSA signing and encryption algorithms, the configuration is similar, but you would choose different values for the `allowed-signing-algorithm` and `allowed-encryption-algorithm` properties.

Steps

1. Create an encryption key pair.

   Example:

   ```
   # Create an encryption key pair
   dsconfig create-key-pair \
     --pair-name "JWT Elliptic Curve Encryption Key Pair" \
     --set key-algorithm:EC_256
   ```

2. Create the JWT access token validator.

   Example:

   ```
   # 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 allowed-signing-algorithm:ES256 \
     --set "authorization-server:PingFederate External Server" \
     --set jwks-endpoint-path:/ext/oauth/jwks \
     --set "encryption-key-pair:JWT Elliptic Curve Encryption Key Pair" \
     --set allowed-key-encryption-algorithm:ECDH_ES

   # Match the token's subject (sub) claim to the uid attribute
   ```

```
# of a SCIM resource
dsconfig create-token-resource-lookup-method \
  --validator-name "JWT Access Token Validator" \
  --method-name "User by uid" \
  --type scim \
  --set scim-resource-type:Users \
  --set 'match-filter:uid eq "%sub%"' \
  --set evaluation-order-index:1000
```

3. Export the public encryption key from PingAuthorize Server and provide it to your token issuer.

   Without this public encryption key, the issuer cannot encrypt tokens that can be decrypted by the JWT access token validator.

   Example:

   You can run **`dsconfig`** to copy the public key to a file, or you can copy the value of the key pair's `certificate-chain` property in the administrative console.

   ```
   dsconfig get-key-pair-prop \
     --pair-name "JWT Elliptic Curve Encryption Key Pair" \
     --property certificate-chain \
     --no-prompt \
     --script-friendly > jwt-public-encryption-key.pem
   ```

Mock access token validator

A mock access token validator is a special access token validator type used for development or testing purposes; it 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

Example mock access token validator 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. This type of 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 *Sideband access token validation* on page 190.

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

Example configuration

The following example shows how to configure an external API gateway access token validator with a token resource lookup method and 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

Access token validators can use token resource lookup methods to search a datastore and retrieve the subject's profile data for use in policy decisions.

Most access tokens include a subject, which identifies the user who granted access to the application using the token. Token resource lookup methods use the access token subject value, which is usually a string identifier such as a GUID or username, to perform a search in an external datastore, such as a PingDirectory Server or an API providing user data. For this reason, the datastore or API must be accessible to PingAuthorize Server; and in most cases, it should be the same datastore or API used by the authorization server that issues the access tokens. After the lookup completes, the token subject's user attributes get passed into the policy request's `TokenOwner` attribute, allowing policies to make decisions based on some aspect of the user.

Note:

Using a token resource lookup method is optional. If your policies don't need user profile information, you don't need to configure token resource lookup methods.

PingAuthorize Server provides the following types of token resource lookup methods:

- *SCIM token resource lookup methods* on page 299
- *Third-party token resource lookup methods* on page 300

SCIM token resource lookup methods

System for Cross-domain Identity Management (SCIM) token resource lookup methods use PingAuthorize 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 193.

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 performs a SCIM search using the filter `id eq "8ac3d8b5-4f17-33fa-a4b4-854599ed9a89"`.

Example

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 PingAuthorize Server deployments where SCIM is not otherwise needed. For example, you could use a third-party token resource lookup method to connect a PingAuthorize 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 PingAuthorize Server *Configuration Reference*, located in the server's `docs/config-guide` directory.

This section covers basic server configuration.

PingAuthorize 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 PingAuthorize 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 PingAuthorize 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.

Example

This example `dsconfig` script configures the PingAuthorize Server policy service.

```
# Define an external PingAuthorize PAP
dsconfig create-external-server \
  --server-name "PingAuthorize Policy Editor" \
  --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:PingAuthorize PAP" \
  --set "decision-response-view:request" \
  --set "decision-response-view:decision-tree"
```

Example

To load a `dsconfig` batch file, run `dsconfig` with the `--batch-file` argument.

```
$ PingAuthorize/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 "PingAuthorize 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:PingAuthorize PAP" --set
decision-response-view:request --set decision-response-view:decision-tree

The Policy Decision Service was modified successfully.
```

PingAuthorize administrative console

The PingAuthorize 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 PingAuthorize 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`.

Example

```
$ tail -n 8 PingAuthorize/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 "PingAuthorize
 PAP"
dsconfig create-external-server --server-name "PingAuthorize 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:PingAuthorize (Gateway Policy Example)"
dsconfig set-policy-decision-service-prop --set "policy-server:PingAuthorize
 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.

Example

```
$ PingAuthorize/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.

PingAuthorize 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.

   Result: 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`.
   Example:

   ```
   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`.
   Example:

   ```
   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`.
   Example:

   ```
   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
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"
```

Listener certificates
When a client initiates TLS negotiation with the server, the server presents a certificate chain to the client and the certificate at the head of the chain functions as a listener certificate.

Because the client decides whether to trust the certificate chain, it is recommended that the chain be signed by an issuer whom the client is likely to trust or that the client can be easily configured to trust.

You can create self-signed certificates with long lifespans, but a certificate that a certification authority signs is likely to have a relatively short lifespan. Commercial authorities typically issue certificates that are valid for only one or two years, but some authorities use shorter validity windows.

Short certificate lifespans offer some security benefits. In particular, because most clients do not verify whether a certificate has been revoked, a shorter validity window minimizes the timeframe that a compromised certificate can be used. If the process for replacing certificates is streamlined or automated, administrative inconvenience can be kept to a minimum.

Listener certificates are stored in key stores that are referenced by key manager providers, which in turn provide the logic and configuration for accessing the key stores. If a server component, like a connection handler, requires access to a certificate that it presents to a peer during the TLS negotiation process, that component must reference the key manager provider that points to the key store containing the appropriate certificate. If the key store contains multiple certificates, and if the component referencing the key store includes a property specifying the certificate's nickname, the certificate with that alias is selected. Otherwise, the server lets the Java virtual machine (JVM) select a certificate that might not be well-defined.

The server also provides trust manager providers, which determine whether to trust the certificate chains with which it is presented. A trust manager provider can reference a specified trust store file, but other options include the JVM default trust store, which uses the Java installation's default set of trusted issuers, and the blind trust manager provider, which automatically trusts every certificate chain that is presented to it.

Note:

Never use a blind trust manager in a production environment because it leaves the server vulnerable to impersonation and man-in-the-middle attacks. However, a blind trust manager can be convenient in test environments when troubleshooting certain types of problems.

Replacing listener certificates

Certificate authorities typically restrict the lifespans of the certificates that they sign. If you use a certification authority to issue listener certificates, you are likely replacing the certificates on a regular basis.

About this task

The **`replace-certificate`** tool performs the following steps:

1. Obtain a new certificate chain.
2. Make necessary updates to the key manager provider and the connection handler configurations
3. Update the server instance listener configuration with the new certificate.

The **`replace-certificate`** tool offers the following modes of operation:

Interactive mode

Walks you through the process of obtaining a new certificate and installing it in the server. Interactive mode also displays the non-interactive commands that are required to achieve the same result.

Non-interactive mode

Useful when scripting the process of replacing a certificate.

Steps

- To replace a listener certificate, run the **`replace-listener-certificate`** subcommand of the **`replace-certificate`** tool.

  Note:

  You can replace certificates manually, but the **`replace-certificate`** tool automates the process. The **`replace-certificate`** tool provides information about multiple listener certificates during the transitional phase that occurs when you install them.

  The **`replace-listener-certificate`** subcommand takes arguments that provide the following information:

  - Arguments required to authenticate to , such as **`--bindDN`** and **`--bindPasswordFile`**
  - Details about the key store that contains the new certificate
  - Updates that must be made to the key and trust manager providers
  - Whether to signal the HTTP connection handler to reload its certificates after the update is complete

  The following arguments are available:

Argument	Description
`--source-key-store-file {path}`	Path to the Java KeyStore (JKS) or PKCS #12 file that contains the private key entry with the new certificate chain. This argument is required.
`--source-key-store-password {password}`	Clear-text password that is needed to access the contents of the source key store.

Argument	Description
--source-key-store-password-file {path}	Path to the file that contains the password necessary to access the contents of the source key store. The file can contain the password in the clear or can be encrypted with a definition from the server's encryption settings database.
--source-certificate-alias {alias}	Password that is required to access the appropriate private key in the source key store. If neither the **--source-private-key-password** nor the **--source-private-key-password-file** argument is provided, the key store password is used as the private key password.
--source-private-key-password-file {path}	Path to the file that contains the password needed to access the appropriate private key in the source key store. The file can contain the password in the clear or can be encrypted with a definition from the server's encryption settings database. If neither the **--source-private-key-password** nor the **--source-private-key-password-file** argument is provided, the key store password is used as the private key password.
--key-manager-provider {name}	Name of the key manager provider that is updated to use the new certificate chain. The value must identify a file-based key manager provider, and the new certificate chain must be enabled. Defaults to **JKS** if a value is not specified.
--trust-manager-provider {name}	Name of the trust manager provider that is updated with the information required to trust the new certificate chain. The value must identify a file-based trust manager provider, and the new certificate chain must be enabled. If neither this argument nor the **--use-jvm-default-trust-manager-provider** argument is provided, the tool assumes that the name of the trust manager provider is identical to the name of the key manager provider.
--use-jvm-default-trust-manager-provider	Indicates that the server must be configured to use the JVM-default trust manager provider, which trusts certificates signed by issuers in the `cacerts` trust store provided with the Java virtual machine (JVM), rather than updating an existing trust manager provider.
--target-certificate-alias {alias}	Alias to use for the new certificate in the key manager provider's key store, and for appropriate updates in the trust manager provider's trust store. Defaults to an alias of **server-cert** if a value is not specified.

Argument	Description
	Note: If the key manager provider's key store, or the trust manager provider's trust store, already contains an entry with the given alias, the existing entry is renamed.
`--reload-http-connection-handler-certificates`	Indicates that the tool is requesting that the server cause any HTTPS-based connection handlers to reload their certificates, so that the connection handlers can use the updated certificate. LDAP connection handlers react to the change immediately and start presenting the new certificate chain during subsequent TLS negotiations. HTTPS connection handlers continue using the former certificate until the connection handler is restarted or until the connection handler is asked specifically to reload its certificates. **Note:** This option might prevent clients with existing TLS sessions that were negotiated with the former certificate from being resumed.

- To remove earlier certificates from the server instance listener configuration, run the **`purge-retired-listener-certificates`** subcommand.

> **Note:**
>
> The **`purge-retired-listener-certificates`** subcommand does not take arguments other than the ones that are required to authenticate to the server.
>
> By default, the **`replace-certificate`** tool updates the server instance listener configuration object to include the new listener certificate, and it merges the old and new certificates residing in the configuration object.

X.509 certificates

The server supports X.509 certificates, the most common type of certificates. *RFC 5280* describes X.509v3, which provides the current version of the specification.

An X.509v3 certificate includes the following components:

X.509 encoding version

: Enables the differentiation between an X.509v3 certificate and one that conforms to an earlier or later version of the specification.

Serial number of the certificate

: Integer value that uniquely identifies a certificate as issued by a certification authority.

Subject DN

Distinguished name for the certificate, which often provides details about the context in which the certificate is to be used. For more information, see *Certificate subject DNs* on page 311.

Issuer DN

Distinguished name for the issuer certificate, which is the certificate used to sign the certificate. For a self-signed certificate, this value matches the subject DN.

Validity window

Indicates the timeframe during which the certificate is considered valid. This component includes the following elements:

- **`notBefore`**

  Specifies the earliest time at which the certificate is considered valid.
- **`notAfter`**

  Specifies the latest time at which the certificate is considered valid.

Public key

Public portion of a pair of cryptographically linked keys. For more information, see *Certificate key pairs* on page 312.

Signature

A type of cryptographic proof that the certificate truly was sent from the issuer and has remained unaltered. A self-signed certificate is signed with its own private key. Otherwise, it is signed with the issuer's private key.

An X.509v3 certificate might also include the following optional components:

Subject unique ID

Uniquely identifies the certificate. This component has been deprecated in favor of the subject key identifier extension, so it is generally omitted from X.509v3 certificates.

Issuer unique ID

Subject unique ID of the issuer certificate, if available. This component has been deprecated in favor of the authority key identifier extension.

Set of extensions

Provides additional context for the certificate and the manner in which it is used. For more information, see *Certificate extensions* on page 313.

Certificate subject DNs

A certificate's subject distinguished name (DN) provides information about how the certificate should be used.

Like an LDAP DN, a certificate's subject DN consists of a comma-delimited series of attribute-value pairs. However, unlike an LDAP DN, the attribute names in a certificate subject DN are typically written in all uppercase characters.

A certificate's subject DN is also referred to as its subject. The following attributes commonly appear in a certificate subject.

Attribute name	Attribute description
`CN`	Common name **Note:** For a listener certificate, the `CN` attribute typically identifies the host name that clients use to access the certificate. However, the subject alternative name extension is recommended more highly for accomplishing the same task. Most certificate subject DNs include at least the `CN` attribute.
`E`	Email address
`OU`	Name of the organizational unit, such as the relevant department
`O`	Name of the organization or company
`L`	Name of the locality, such as the appropriate city
`ST`	Full name of the state or province
`C`	ISO 3166 country code

A certificate subject includes at least one attribute-value pair, and the `CN` attribute is typically present. Other attributes can be omitted, although the `O` and `C` attributes are also common. For example, a listener certificate for a server with an address of `ldap.example.com`, which is run by the US-based company Example Corp, might have a subject of `CN=ldap.example.com,O=Example Corp,C=US`.

Certificate key pairs
Each certificate contains a key pair that consists of two keys that are linked cryptographically. If you encrypt data with one key, the data can be only decrypted with the other key.

Although a key pair can be created easily when both keys are generated simultaneously, the process of deriving one key from the other is extremely difficult, a process categorized in cryptographic terms as computationally infeasible.

When generating a key pair, one key is designated as the public key, and the other key is designated the private key. The public key can be made widely available, but the private key must be kept secret and not shared with anyone.

As long as the secrecy of the private key is maintained, the key pair can be used to perform the following functions:

- Encryption, sometimes referred to as confidentiality

  If someone wants to send you a secret message without anyone else viewing it, the message can be encrypted with your public key. Only you possess the private key, so only you can decrypt the message.
- Digital signatures

  If you encrypt data with your private key, it can be decrypted only with your public key. Because your public key can be made widely available, this encryption method does not actually protect the content.

However, digital signatures prove that a message came from you because only your private key could have generated it.

> **Note:**
>
> When generating a digital signature, the entire message is generally not encrypted. Only a hash of the message is encrypted, typically by using a digest algorithm like SHA-256.
>
> This approach protects the integrity of a message. A decrypted signature that matches the digest of the original message guarantees that the message came from you and that it has remained unaltered since you signed it.

The following public key algorithms are used primarily in certificates that facilitate TLS communication:

- RSA, which is based on the multiplication of large prime numbers
- EC, which is based on computations that involve special types of elliptical curves

Although RSA is supported more widely than EC, it is slower and requires larger keys to achieve the same level of security. To support legacy clients, you should use an RSA certificate and choose a key size of at least 2,048 bits.

If all of your clients support EC certificates, you should use an EC certificate with a key size of at least 256 bits.

Certificate extensions
Extensions provide additional context for a certificate.

Some of the more common extension types include the following:

Subject key identifier

: Holds a unique identifier for the certificate, which is generally derived from the certificate's public key.

Authority key identifier

: Holds the subject key identifier for the issuer certificate. This extension type helps to identify the issuer certificate, especially when presented with an incomplete certificate chain.

Subject alternative name

: Holds a list of ways that clients are expected to reference a server when establishing a connection to it.

> **Note:**
>
> Clients must take this information into account when deciding whether to trust a server's certificate.
>
> The most common types of values include DNS names, IP addresses, and URIs. DNS names must be fully qualified, but can optionally use an asterisk in the leftmost component to match any single name in that component. For example, `*.example.com` could match `www.example.com` or `ldap.example.com`, but would not match `ldap.east.example.com` or `example.com`.

Key usage

: Provides information about the manner in which the certificate is expected to be used. The following key usages are allowed:

  `digitalSignature`

  : Indicates that the certificate can be used for digitally signing data, excluding certificates and certificate revocation lists (CRL).

`nonRepudiation`

Indicates that the certificate can be used to prevent denying the authenticity of a message. `nonRepudiation` is also known as `contentCommitment`.

`keyEncipherment`

Indicates that the certificate can be used to protect encryption keys, such as symmetric keys that are derived during TLS key agreement.

`dataEncipherment`

Indicates that the certificate can be used for encrypting data directly.

`keyAgreement`

Indicates that the certificate's public key can be used for key agreement, such as deriving the symmetric key that protects TLS communication.

`keyCertSign`

Indicates that the certificate can act as a certification authority and be used for signing other certificates.

`cRLSign`

Indicates that the certificate can be used to sign CRLs.

`encipherOnly`

When used in conjunction with `keyEncipherment`, indicates that the public key can be used only for encrypting data during key agreement.

`decipherOnly`

When used in conjunction with `keyEncipherment`, indicates that the public key can be used only for decrypting data during key agreement.

Extended key usage

Acts as an alternative to the key usage extension and provides additional high-level functionality. The following extended key usages are allowed:

`serverAuth`

Indicates that the server can present the certificate to the client during TLS negotiation.

`clientAuth`

Indicates that the client can present the certificate to the server during TLS negotiation.

`codeSigning`

Indicates that the certificate can be used to sign source and compiled code.

`emailProtection`

Indicates that the certificate can be used to sign or encrypt email messages.

`timeStamping`

Indicates that the certificate can be used to assert the time that an event occurred.

`ocspSigning`

Indicates that the certificate can be used to sign an online certificate status protocol (OCSP) response.

Basic constraints

Indicates whether the certificate can act as a certification authority and, if so, the maximum number of intermediate certificates that can follow it in a certificate chain.

Certificate chains

A certificate chain is an ordered list of one or more certificates. In such a chain, each subsequent certificate is the issuer of the previous certificate.

During TLS negotiation, the server presents a certificate chain to the client, which determines whether to trust the chain and continue with the negotiation. The client can also present its own certificate chain to the server.

If a certificate is self-signed, its chain contains only that single certificate. If a certificate is signed by a self-signed certificate authority (CA) certificate, such as a root CA, the chain contains two certificates: the server certificate and the CA certificate that follows it. If a single intermediate CA (a CA certificate that is signed by a root CA) is present, the chain contains the server certificate, followed by the intermediate CA, and then the root CA.

Intermediate certificate authorities are useful for security purposes, especially in commercial authorities. If a client trusts a root CA certificate, it is likely to trust anything with that root CA certificate at the base of its chain. Consequently, the root CA certificate must be kept secure.

Note:

If the root CA certificate is compromised, any certificate that is directly or indirectly signed by it can no longer be trusted.

With intermediate CA certificates, the root certificate can be kept offline in secure storage and used only when a new intermediate CA certificate must be signed. The intermediate CA certificates can be used to sign end-entity certificates, but must be protected to avoid compromising any of the certificates. A compromised certificate must be revoked along with all of the certificates that it signed. In such a scenario, the root CA can be used to sign a new certificate.

Note:

The certificate chain that the server presents to the client, or that the client presents to the server, during TLS negotiation does not always need to be the complete chain. If the root CA at the end of the chain is widely trusted, the server can assume that the client already has that root CA in its default set of trusted certificates. The server can leave that root CA off the chain with the assumption that the client will retrieve it from its default trust store. While the same assumption could theoretically be true for intermediate CA certificates, only the root CA certificate is commonly omitted. When a client receives an incomplete chain, the client looks in its default trust store to determine whether the trust store contains the issuer certificate, which it can identify by using properties like the issuer distinguished name (DN) or an authority key identifier extension.

The certificate at the head of a certificate chain, which appears as the first one in the list, is often called the end-entity certificate. If this certificate appears at the head of the chain that a server presents during TLS negotiation, it is referred to as the server certificate. If the certificate appears at the head of a chain that a client presents, it is referred to as a client certificate. The certificate at the end of a complete chain must be a root CA certificate. In the case of a self-signed certificate, the chain contains only a single certificate that serves both roles.

About representing certificates, private keys, and certificate signing requests
X.509 is an encoding format that uses the ASN.1 distinguished encoding rules (DER), which exist in binary format. When writing a certificate to a file, either a raw DER format or a plaintext format called PEM can be used.

PEM encoding consists of a line that contains the text `-----BEGIN CERTIFICATE-----`, followed by a set of lines that contains the base64-encoded representation of the raw DER bytes (typically with no more than 64 characters per line), followed by a line that contains the text `-----END CERTIFICATE-----`.

The X.509 encoding contains a certificate's public key, but not its private key. The PKCS #8 specification in *RFC 5958* describes the encoding for private keys. This approach uses a DER encoding with a PEM variant that instead uses the following header and footer, respectively.

```
-----BEGIN PRIVATE KEY-----
-----END PRIVATE KEY-----
```

RFC 5958 also describes an encrypted representation of the private key, but that format is currently unsupported.

The PKCS #10 specification in *RFC 2986* describes the CSR format. This format uses a DER encoding with a PEM variant that uses the following header and footer, respectively.

```
-----BEGIN CERTIFICATE REQUEST-----
-----END CERTIFICATE REQUEST-----
```

Some implementations use the following alternate, nonstandard forms.

```
-----BEGIN NEW CERTIFICATE REQUEST-----
-----END NEW CERTIFICATE REQUEST-----
```

Certificate trust
When a server presents its certificate chain to a client during TLS negotiation, the client decides whether to trust the certificate chain and concludes whether it is communicating with a legitimate server instead of an impostor.

If a client is tricked through DNS hijacking into communicating with a rogue application instead of with a legitimate server, the application can steal the client's credentials, or can fool the client into concluding that it has performed an action that it has not performed. If a rogue application acts as a broker between the client and the legitimate server, the client might be unable to detect the change, and the malicious application might be capable of stealing data or altering the communication. Consequently, you should avoid `trust all` or `blind trust` options in a production environment.

When determining whether to trust a server certificate chain, a client performs the following steps.

Processing steps

1. Verifies that it has received the complete certificate chain.

   If a server presents an incomplete chain, the client must ensure that it can complete the chain with information in an explicitly provided trust store or default trust store. If the client cannot complete the certificate chain, the chain is not trusted.

2. Verifies that each subsequent certificate in the chain is the issuer certificate for, and that its private key was used to sign, the certificate that precedes it.

   Note:

   If a certificate chain contains extraneous certificates, or if a subsequent certificate did not issue the certificate that precedes it, the chain is not trusted.

3. Confirms that it has a reason to trust the certificate at the root of the chain.

   Note:

   This step is generally performed by ensuring that the root certificate authority (CA) certificate can be found in either a default trust store or a trust store that is configured for use by the client. If the client has no prior knowledge of the root CA certificate, the chain is not trusted.

4. Verifies that the current time lies within the validity window for each certificate in the chain.

   Note:

   The chain is not trusted under the following conditions:

   - When the `notBefore` value of any certificate in the chain is later than the current time.
   - When the `notAfter` value of any certificate in the chain is earlier than the current time.

5. Verifies that the server certificate at the head of the chain is suitable for the server with which the client thinks it is communicating.

   Note:

   The client must verify that the address used to connect to the server matches one of the following values:

   - The `CN` attribute of the certificate's subject.
   - One of the values of any subject alternative name extension.

These steps represent a starting point. If necessary, the client can perform additional types of validation. For example, if a root or intermediate certification authority maintains a certificate revocation list (CRL) or supports the online certificate status protocol (OCSP), the client must verify that none of the certificates in the chain has been revoked. The client can also verify that the CA certificates include the basic constraints extension, and that the server certificate does not contain too many levels. Other checks, like those that use certificate policy extensions, can also be performed.

Keystores and truststores

A keystore is a type of database that holds certificates.

The following examples represent the most common forms of keystores:

- File that uses the Java-specific Java KeyStore (JKS) format
- File that uses the standard PKCS #12 format
- Collection of files that holds certificates and private keys, typically in PEM or DER format
- Hardware security module (HSM) that makes the certificate information available through an interface like PKCS #11

The server supports file-based keystores by using the JKS and PKCS #12 formats and by using hardware security modules that are accessible through PKCS #11. The server does not currently support a keystore

format that consists of individual certificate and private key files. To import these files into a JKS or PKCS #12 keystore, use the **`manage-certificates`** tool.

A keystore also represents a collection of entries, each of which is identified by a name that an alias calls. Keystores can have the following entry types:

Private key entries

: Contain a certificate chain and a private key. When a server accepts a TLS-based connection, it uses a private key entry to obtain the certificate chain that it presents to the client. The server can also use the private key from the same entry to process its key agreement. Similarly, a client uses a private key entry when presenting its own certificate chain to a server.

Trusted certificate entries

: Contain a single certificate without a private key. As the name implies, a trusted certificate entry is intended primarily for use when determining whether to trust a certificate chain that is presented during TLS negotiation.

Secret key entries

: Contain a secret key only, without an associated certificate. These types of entries are not used for TLS processing. Instead, they hold symmetric encryption keys or other types of secrets.

A password, sometimes called a PIN, protects the contents of a keystore. In some cases, like with JKS keystores, some content might be accessible without a password, and a password might be required only when trying to access private keys or secret keys. In other cases, like with PKCS #12 keystores, you might need a password for any interaction with the keystore.

Additional passwords can further protect private keys. This approach is often the same as with the keystore password, but the password can be different. This tactic is useful when a single keystore is shared for multiple purposes, for example, and when merely having access to the keystore does not guarantee access to all of the data that it contains.

> **Note:**
>
> A truststore is another name for a keystore that is intended primarily for use when determining whether to trust a certificate chain that has been presented. Truststores generally contain primarily trusted certificate entries, but that case is not required.

Java runtime environments typically include a default truststore, often `jre/lib/security/cacerts` or `lib/security/cacerts`, that is prepopulated with several widely trusted certification authority certificates. When presented with a certificate that one of these authorities has signed, the default truststore can allow the certificate to be trusted without any additional configuration. When presented with a self-signed certificate, or when presented with a certificate that is signed by an issuer not in the default truststore, such as a private corporate certification authority, a separate truststore is required.

Transport Layer Security (TLS)
TLS describes a mechanism for securely communicating between two parties that might have no prior knowledge of each other.

TLS is the successor to SSL, and the two terms are often used interchangeably, even though such usage might not technically be correct.

> **Note:**
>
> SSL remains the more widely recognized term. The abbreviation TLS occasionally generates confusion with the StartTLS extended operation, particularly in LDAP.

TLS provides security in the form of the following main components:

Certificate trust

Is about reassuring a connection-initiating client that it is communicating with the server to which it intended to connect. To ensure that the server shares the same degree of confidence in the identity and legitimacy of the client, it can ask the client to present its own certificate chain. For more information, see *Certificate Trust*.

Cipher selection

Involves choosing the cipher and the key to protect the bulk of the communication. Although a client can use a server certificate's public key to encrypt data before sending it, this approach can lead to the following issues:

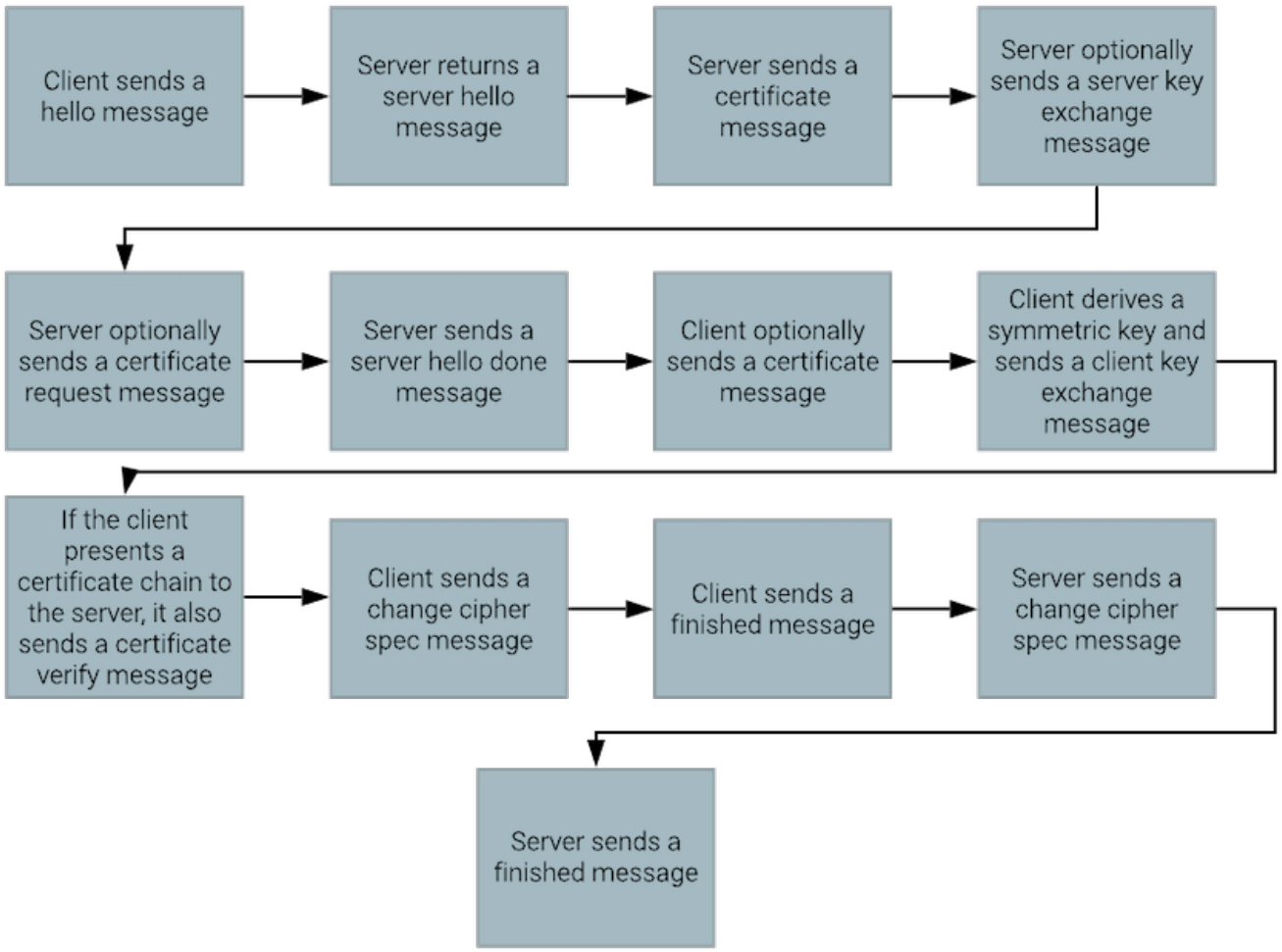
- Unless the client presents its own certificate chain to the server, the server cannot encrypt the data that it sends back to the client.
- Public key encryption is considerably slower than symmetric encryption, in which the same key is used for both encryption and decryption. Public key encryption is also called asymmetric encryption because different keys are used to encrypt and decrypt data.
- If you rely entirely on the security of a private key to ensure the secrecy of a communication, and if the private key becomes compromised, data that has been encrypted with the private key must also be considered compromised.

Rather than relying solely on public key encryption to protect communication between a client and server, the TLS negotiation process allows a client and server to agree on the type of encryption and the secret key to use after completing the negotiation process.

TLS handshakes

The process of negotiating the TLS is referred to as the handshake.

Although the exact process depends on the TLS version that is ultimately chosen, the following steps represent the basic components of a TLS 1.2 handshake:



TLS processing steps

1. The client sends a `hello` message that provides the server with the following information:
   - Highest supported version of the TLS protocol
   - The cipher suites that the client uses
   - Set of extensions with additional information:
     - The address that the client uses to communicate with the server
     - The signature algorithms and elliptic curves that the client supports
     - Whether the client supports secure renegotiation
2. The server returns a `server hello` message that provides the client with the following information:
   - The TLS protocol version that the server uses
   - The cipher suite that the server selects

   The server can also provide its own extensions to the client.
3. The server sends a `certificate` message that provides its certificate chain to the client.
4. The server can optionally send a `server key exchange` message with additional information that the client might need to securely derive the same symmetric encryption key that the server generates.
5. The server can optionally send a `certificate request` message that asks the client to present its own certificate chain to the server.

6. The server sends a `server hello done` message to inform the client that it has completed its `hello` sequence.
7. The client can optionally send a `certificate` message to the server with its own certificate chain.

> **Note:**
>
> The client sends a `certificate` message only when the server initially sends a certificate request. If the client receives such a request, it can refuse to, and probably will not, send a certificate chain. The server decides whether to require a client certificate chain. In LDAP, the server commonly asks the client to present a certificate, but continues with TLS negotiation even if the client does not present one. This approach supports authentication methods like SASL EXTERNAL, in which a client uses the certificate chain that it presents during TLS negotiation as proof of its identity.

8. The client derives a symmetric key to use for the remainder of the encrypted processing, and sends a `client key exchange` message to the server. The `client key exchange` message includes the information that the server needs to generate the same key. Only the client and server know the value of the key, even if another entity can observe the communication that passes between the client and the server.
9. If the client presents a certificate chain to the server, it also sends a `certificate verify` message to prove that the private key for the certificate is included at the head of the chain.
10. The client sends a `change cipher spec` message to the server, which informs the server that the client will use the agreed-upon symmetric key to encrypt everything else that it sends to the server.
11. The client sends a `finished` message to the server to indicate that it has completed its portion of the handshake.
12. The server sends a `change cipher spec` message to the client, which informs the client that the server will use the agreed-upon symmetric key to encrypt everything else that it sends to the client.
13. The server sends a `finished` message to the client to indicate that it has completed its portion of the handshake.

TLS 1.3 uses a different handshake sequence that can require only a single round-trip to exchange the necessary information between the client and the server. TLS 1.2 requires two round-trips. To accomplish this task, TLS 1.3 tries to guess the type of key agreement that the server wants to use, and sends the relevant information to the server up front instead of waiting to hear from the server.

Because an extra round of communication between the client and server is eliminated, the server finishes its portion of the negotiation before the client. The server must assume that the client trusts its certificate chain. Because the server might log a successful negotiation only to discover late, through a TLS alert, that the client rejected the certificate, this approach might complicate certain types of troubleshooting.

Key agreement

Key agreement processing provides a critical component of TLS negotiation.

It allows the client and server to select the symmetric key that encrypts the remainder of the communication, but does not reveal the key to anyone who can access the communication. Although several key agreement algorithms are available, the following types are the most common:

RSA key exchange

: The client generates random data, uses the server's public key to encrypt it, and provides it to the server, which uses its private key to decrypt it. The client and server alike derive the encryption key from the randomly generated data.

Diffie-Hellman (DH) key exchange

: The client and server agree publicly on a pair of mathematically linked numbers, and each participant chooses its own secret value. Through a special computation, they generate a key that can be discovered only by someone who knows one of the secret values. Although several variants of the Diffie-Hellman algorithm can be used in key exchange, we recommend the ECHDE and DHE

versions because they use ephemeral keys with no relation to the server's certificate. Of those two versions, ECDHE is faster and uses smaller keys.

When possible, use ECHDE over DHE, and either of those options over RSA. The DH algorithms provide a substantial benefit over RSA in the form of forward secrecy. Because RSA key exchange uses the server certificate's public key to encrypt data, the encryption can be broken if the certificate's private key is compromised. This warning applies to previously captured data as well as to communication on new TLS connections. The use of ephemeral keys in ECDHE and DHE ensures that, even if the certificate's private key is compromised, the encrypted communication remains indecipherable to anyone but the client and server, although anyone with the private key can still impersonate the legitimate server.

LDAP StartTLS extended operation
In most scenarios, a client that uses TLS establishes a connection to a port that is dedicated to its use, like 636 (LDAPS) or 443 (HTTPS).

The client begins the TLS-negotiation process by sending a `client hello` message over the connection. In some scenarios, the client establishes a non-secure connection and later converts it to a secure one. In LDAP, this task is accomplished by using the **`StartTLS`** extended operation.

The **`StartTLS`** extended operation provides the following advantages over a dedicated LDAPS connection:

- To enable secure as well as insecure communication, only one port needs to be opened through a firewall.
- A client can use opportunistic encryption, in which the client performs the following steps:
  1. Queries the root DSE to determine whether the server supports StartTLS.
  2. Secures the connection, if possible.

  Opportunistic encryption is useful in scenarios like following referrals because LDAP URLs do not officially support LDAPS as a scheme.

To ensure that a communication is always secure, use LDAPS instead of establishing an insecure connection that you secure later with the **`StartTLS`** extended operation. If you enable support for unencrypted LDAP communication, as **`StartTLS`** requires, a client might send a password-containing bind request or other sensitive data over an unencrypted connection. A server can be configured to reject unencrypted communication, but it cannot prevent a client from sending an unencrypted request.

Note:

Although you can use **`StartTLS`** to temporarily secure a connection before falling back on an unencrypted LDAP communication, the server does not support this strategy.

About the manage-certificates tool
PingAuthorize Server offers a **`manage-certificates`** tool that enables interaction with Java KeyStore (JKS) and PKCS #12 key stores.

Although it behaves similarly to the **`keytool`** utility that accompanies most Java distributions, **`manage-certificates`** is easier to use, provides improved usage information, and offers additional functionality.

Available manage-certificates subcommands

The **`manage-certificates`** tool uses the following subcommands to indicate which function to invoke:

Subcommand	Function
`list-certificates`	Lists the certificates in a keystore.

Subcommand	Function
import-certificate	Imports a certificate into a trusted certificate entry or imports a certificate chain and private key into a private key entry.
export-certificate	Exports a certificate from a keystore.
export-private-key	Exports a private key from a keystore.
generate-self-signed-certificate	Generates a self-signed certificate.
generate-certificate-signing-request	Generates a certificate-signing request that can be provided to a certification authority.
sign-certificate-signing-request	Signs a certificate-signing request with a specified issuer certificate.
check-certificate-usability	Checks a specified certificate in a keystore to verify whether it is suitable for use as a listener certificate.
trust-server-certificate	Initiates the TLS-negotiation process with a specified server to obtain its certificate chain so that a truststore can be updated with the necessary information to trust the chain.
display-certificate-file	Displays the contents of a file that contains one or more PEM-encoded or DER-encoded X.509 certificates.
display-certificate-signing-request-file	Displays the contents of a file that contains a PEM-encoded or DER-encoded PKCS #10 certificate-signing request (CSR).
change-certificate-alias	Changes the alias for an entry in a keystore.
change-keystore-password	Changes the password for a keystore.
change-private-key-password	Changes the password that protects the private key for a specified entry in a keystore.

Using manage-certificates as a simple certification authority

If your server instances need to support an arbitrary or unknown set of clients, configure them with certificates from a trusted issuer, such as a commercial authority or the free Let's Encrypt service.

About this task

If you control every client that accesses the servers, you might want to create your own internal certification authority so that you have a common issuer for all servers. In such a scenario, the clients need to trust only the certificates that the issuer signs. Commercial and open-source software packages provide full-featured certification authority functionality, but you can use the **manage-certificates** tool to create a certificate authority (CA) certificate that you can use to sign certificate-signing requests.

Steps

1. Create a CA certificate.

   A CA certificate is a self-signed certificate that possesses the following extensions:

   - A key usage extension that includes at least the keyCertSign usage
   - A basic constraints extension that identifies the certificate as a CA certificate

   If you do not plan to use an intermediate CA certificate, the basic constraints extension must have a path length constraint of **0**. If you plan to use an intermediate CA certificate, the path length constraint must be **1**. Because certificates that the CA certificate signs are valid only for as long as all certificates in the chain remain valid, we recommend that you specify a long lifespan for the CA certificate.

   Example:

   The following example creates a new root CA certificate.

```
$ bin/manage-certificates generate-self-signed-certificate \
     --keystore /ca/root-ca-keystore \
     --keystore-password-file /ca/root-ca-keystore.pin \
     --keystore-type JKS \
     --alias root-ca-cert \
     --subject-dn "CN=Example Root CA,O=Example Corp,C=US" \
     --days-valid 7300 \
     --key-algorithm RSA \
     --key-size-bits 4096 \
     --signature-algorithm SHA256withRSA \
     --basic-constraints-is-ca true \
     --basic-constraints-maximum-path-length 1 \
     --key-usage key-cert-sign \
     --key-usage crl-sign

Successfully created a new JKS keystore.

Successfully generated the following self-signed certificate:
Subject DN:  CN=Example Root CA,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Monday, January 27, 2020 at 03:47:29 PM CST (0
 seconds ago)
Validity End Time: Sunday, January 22, 2040 at 03:47:29 PM CST
                   (7299 days, 23 hours, 59 minutes, 59 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm: SHA-256 with RSA
Public Key Algorithm: RSA (4096-bit)
SHA-1 Fingerprint:
 bc:8e:5b:30:52:ec:03:63:b4:9a:aa:1a:45:a0:fc:84:49:dd:e8:64
SHA-256 Fingerprint:

 d5:47:06:cd:a2:95:42:61:1f:c7:aa:04:16:1e:c1:70:41:c4:44:48:bf:74:20:5f:1c:
   61:e2:aa:40:08:3a:ff
```

2. Export the public portion of the root CA certificate for future reference.

   When you import a signed certificate, you can import the public portion of the root CA certificate as a standalone certificate into trust stores as well as into part of a certificate chain.

3. Create a new certificate signing request to create an intermediate CA certificate,

   The certificate signing request uses essentially the same settings as the root CA. If you anticipate only a single intermediate CA, its basic constraints extension must have a path length constraint of

`0`, rather than `1`, to indicate that it is used only to sign end-entity certificates and that it cannot create subordinate CA certificates by itself.

Example:

The following example command creates a certificate signing request.

```
$ bin/manage-certificates generate-certificate-signing-request \
     --keystore /ca/intermediate-ca-keystore \
     --keystore-password-file /ca/intermediate-ca-keystore.pin \
     --keystore-type JKS \
     --alias intermediate-ca-cert \
     --subject-dn "CN=Example Intermediate CA,O=Example Corp,C=US" \
     --key-algorithm RSA \
     --key-size-bits 4096 \
     --signature-algorithm SHA256withRSA \
     --basic-constraints-is-ca true \
     --basic-constraints-maximum-path-length 0 \
     --key-usage key-cert-sign \
     --key-usage crl-sign \
     --output-file /ca/intermediate-ca-cert.csr \
     --output-format PEM

Successfully created a new JKS keystore.

Successfully generated the key pair to use for the certificate signing
request.

Successfully wrote the certificate signing request to file
'/ca/intermediate-ca-cert.csr'.
```

4. Use the root CA certificate to sign the certificate signing request for the intermediate CA certificate with the **`sign-certificate-signing-request`** subcommand.

   The **`sign-certificate-signing-request`** subcommand takes most of the same arguments as generating a self-signed certificate. The primary differences between the argument sets are as follows:

   - The key store that contains the certificate uses the provided key store arguments to sign the request. To specify the name of the certificate to use when signing the request, use the `--signing-certificate-alias` argument.
   - To specify the path to the file that contains the certificate signing request file to generate, provide a `--request-input-file` argument.
   - To specify the path to the file to which the signed certificate is written, provide a `--certificate-output-file` argument. If this argument is omitted, the PEM representation of the certificate is written to standard output.
   - To specify the format, **`PEM`** or **`DER`**, in which the certificate is written to the output file, provide an `--output-format` argument.
   - To specify the subject to use for the signed certificate, use the `--subject-dn` argument. To use the subject DN from the certificate signing request, omit this argument.
   - To specify the name of the signature algorithm, use the `--signature-algorithm` argument.

     > **Note:**
     >
     > Because the requester generated the key, you cannot specify the key algorithm or the key length.

   - To indicate that the signed certificate includes every extension that is listed in the certificate signing request, use the `--include-requested-extensions` argument. If this argument is not provided, explicitly specify the set of extensions to include.

   Example:

   The following example command signs the certificate signing request for an intermediate CA certificate.

   ```
   $ bin/manage-certificates sign-certificate-signing-request \
        --keystore /ca/root-ca-keystore \
        --keystore-password-file /ca/root-ca-keystore.pin \
        --signing-certificate-alias root-ca-cert \
        --days-valid 7300 \
        --include-requested-extensions \
        --request-input-file /ca/intermediate-ca-cert.csr \
        --certificate-output-file /ca/intermediate-ca-cert.pem \
        --output-format PEM

   Read the following certificate signing request:

   PKCS #10 Certificate Signing Request Version:  v1
   Subject DN:  CN=Example Intermediate CA,O=Example Corp,C=US
   Signature Algorithm:  SHA-256 with RSA
   Public Key Algorithm:  RSA (4096-bit)

   Do you really want to sign this request? yes

   Successfully wrote the signed certificate to file
   '/ca/intermediate-ca-cert.pem'.
   ```

5. After you obtain the intermediate CA certificate, create secure, offline backups of the root CA certificate.

6. Remove the root CA certificate, or at least its private key, from all systems.

> **Note:**
>
> Make certain that all end-entity certificates are signed with the intermediate CA certificate, and that the process is identical to the previous example. Restore the root CA certificate only if you need to sign another intermediate CA certificate.

Common manage-certificates arguments

Most of the **`manage-certificates`** subcommands require access to a Java KeyStore (JKS) or PKCS #12 keystore. In such instances, use the **`--keystore`** argument to specify the path to the keystore.

If the keystore already exists, the tool detects automatically whether it is a JKS or PKCS #12 keystore. If the operation creates a new keystore, you can specify the type explicitly by using the **`--keystore-type`** argument, followed by a value of **`JKS`** or **`PKCS12`**. If you do not specify the keystore type, a default value of **`JKS`** is used.

Some situations require you to provide the password that is needed to access the keystore. For a JKS keystore, you might need to provide a keystore password only for operations that involve creating a keystore or accessing a private key. However, you will likely need to provide the password for all operations that involve a PKCS #12 keystore.

To provide a keystore password, use one of the following arguments:

- **`--keystore-password`**, followed by the clear-text password for the keystore.
- **`--keystore-password-file`**, followed by the path to a file that contains the password for the keystore. The file might contain the password in the clear, or it might be encrypted with a definition from the server's encryption-settings database.
- **`--prompt-for-keystore-password`**. If this argument is provided, the tool prompts you interactively to provide the password.

If a private key is protected with a different password than the keystore itself, specify one of the following arguments to provide the private key password:

- **`--private-key-password`**, followed by the plaintext password.
- **`--private-key-password-file`**, followed by the path to a file that contains the clear-text or encrypted password.
- **`--prompt-for-private-key-password`**, which causes the tool to prompt interactively for the password.

Several operations require you to specify the keystore entry to target. In such scenarios, provide the **`--alias`** argument, followed by the name of the alias for that entry.

Listing the certificates in a keystore

List the certificates available in a keystore.

Steps

- To list the certificates in a keystore, use the **`list-certificates`** subcommand.

  This subcommand requires you to specify the path to the keystore file, and possibly the password that is needed to access the keystore. The following options are also available:

Option	Description
`--alias {alias}`	Specifies the alias of the certificate to display. If this value is not provided, all certificates are displayed. To list more than one specific certificate, specify this value multiple times.

Option	Description
--display-pem-certificate	Includes a PEM-encoded representation of each certificate as part of the output.
--verbose	Includes details about each certificate.

Example:

The following command demonstrates the basic listing of a keystore that contains a single certificate chain.

```
$ bin/manage-certificates list-certificates \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin

Alias:  server-cert (Certificate 1 of 2 in a chain)
Subject DN:  CN=ds1.example.com,O=Example Corp,C=US
Issuer DN:  CN=Example Certification Authority,O=Example Corp,C=US
Validity Start Time:  Saturday, November 9, 2019 at 11:26:09 AM CST
                      (8 minutes, 15 seconds ago)
Validity End Time:  Sunday, November 8, 2020 at 11:26:09 AM CST
(364 days, 23 hours, 51 minutes, 44 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP256r1)
SHA-1 Fingerprint: 42:f8:85:97:bf:88:bc:74:4b:5b:ce:0c:54:43:9b:44:6b:
                   81:23:a3
SHA-256 Fingerprint: 4f:be:47:ed:36:68:13:38:ba:e8:c0:c5:6c:85:51:97:
                     8b:40:1b:76:10:c0:be:80:15:62:06:96:c5:71:30:df
Private Key Available:  Yes
The certificate has a valid signature.

Alias:  server-cert (Certificate 2 of 2 in a chain)
Subject DN:  CN=Example Certification Authority,O=Example Corp,C=US
Issuer DN:  CN=Example Certification Authority,O=Example Corp,C=US
Validity Start Time: Saturday, November 9, 2019 at 11:26:08 AM CST
                     (8 minutes, 16 seconds ago)
Validity End Time: Friday, November 4, 2039 at 12:26:08 PM CDT
                   (7299 days, 23 hours, 51 minutes, 43 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP256r1)
SHA-1 Fingerprint: b8:d0:16:9b:5d:f2:e7:a1:80:79:95:a2:64:b5:aa:ad:80:
                   23:64:16
SHA-256 Fingerprint: cf:98:2a:66:35:6e:6d:f9:5d:25:c6:68:68:04:5a:a8:
                     88:43:ca:b5:c8:e5:c9:95:09:e9:fc:ab:b9:41:ec:71
The certificate has a valid signature.
```

Example:

The following sample represents the verbose version of the previous command.

```
$ bin/manage-certificates list-certificates \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --verbose

Alias:  server-cert (Certificate 1 of 2 in a chain)
X.509 Certificate Version:  v3
Subject DN:  CN=ds1.example.com,O=Example Corp,C=US
Issuer DN:  CN=Example Certification Authority,O=Example Corp,C=US
Serial Number:  7b:2d:91:6a:ff:51:4f:7a:19:16:26:4f:ce:cb:cb:31
```

```
Validity Start Time:  Saturday, November 9, 2019 at 11:26:09 AM CST
(9 minutes, 48 seconds ago)
Validity End Time: Sunday, November 8, 2020 at 11:26:09 AM CST
                   (364 days, 23 hours, 50 minutes, 11 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Signature Value:
     30:46:02:21:00:cb:d5:5e:45:b2:8a:33:5e:2d:85:23:39:49:d1:3f:8f:dc:
     f8:9e:2f:f3:44:2f:41:0d:69:95:ec:f0:f5:c0:80:02:21:00:ef:8f:32:35:
     3c:88:f4:89:ed:f3:a6:76:
     bb:92:6c:eb:c6:17:ac:61:dc:67:26:f0:ec:67:90:51:28:a1:d0:d5
Public Key Algorithm:  EC (secP256r1)
Elliptic Curve Public Key Is Compressed:  false
Elliptic Curve X-Coordinate:
   -24253153720011259408467676608081666342358203254369897642016197975874105796326
Elliptic Curve Y-Coordinate:
   48722714538591494552787288916186748185323678082126843165293664643134352536146
Certificate Extensions:
     Subject Key Identifier Extension:
          OID:  2.5.29.14
          Is Critical:  false
          Key Identifier:

 21:ad:b9:7a:15:e4:08:13:05:e1:c2:64:0c:86:aa:9b:f0:4c:fb:a0
     Authority Key Identifier Extension:
          OID:  2.5.29.35
          Is Critical:  false
          Key Identifier:

 01:4b:69:99:93:5f:76:51:39:95:61:cc:a9:a8:cb:16:f2:0f:8c:c8
     Subject Alternative Name Extension:
          OID:  2.5.29.17
          Is Critical:  false
          DNS Name:  ds1.example.com
          DNS Name:  ds.example.com
          DNS Name:  ldap.example.com
          DNS Name:  localhost
          IP Address:  127.0.0.1
          IP Address:  0:0:0:0:0:0:0:1
     Key Usage Extension:
          OID:  2.5.29.15
          Is Critical:  false
          Key Usages:
               Digital Signature
               Key Encipherment
               Key Agreement
     Extended Key Usage Extension:
          OID:  2.5.29.37
          Is Critical:  false
          Key Purpose ID:  TLS Server Authentication
          Key Purpose ID:  TLS Client Authentication
SHA-1 Fingerprint:
   42:f8:85:97:bf:88:bc:74:4b:5b:ce:0c:54:43:9b:44:6b:81:23:a3
SHA-256 Fingerprint:
   4f:be:47:ed:36:68:13:38:ba:e8:c0:c5:6c:85:51:97:8b:40:1b:76:
   10:c0:be:80:15:62:06:96:c5:71:30:df
Private Key Available:  Yes
The certificate has a valid signature.

Alias:  server-cert (Certificate 2 of 2 in a chain)
X.509 Certificate Version:  v3
Subject DN:  CN=Example Certification Authority,O=Example Corp,C=US
```

```
Issuer DN:  CN=Example Certification Authority,O=Example Corp,C=US
Serial Number:  43:b7:bb:0c:82:58:42:d8:06:fc:2a:f6:04:e8:2e:8c
Validity Start Time: Saturday, November 9, 2019 at 11:26:08 AM CST
                     (9 minutes, 49 seconds ago)
Validity End Time: Friday, November 4, 2039 at 12:26:08 PM CDT
                   (7299 days, 23 hours, 50 minutes, 10 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Signature Value:
     30:45:02:21:00:b9:87:50:5d:b7:6a:19:82:99:9b:aa:f1:5d:25:a1:90:3c:
     17:9d:7f:f5:7f:8d:06:b4:57:41:9e:15:c6:5a:af:02:20:0c:00:5e:17:bf:
     ca:bf:0b:ff:db:9f:dc:55:ad:35:eb:df:f6:37:4e:23:83:36:88:d2:cc:
     7d:9e:23:da:78:28
Public Key Algorithm:  EC (secP256r1)
Elliptic Curve Public Key Is Compressed:  false
Elliptic Curve X-Coordinate:

 -207531030019209390598003353674157617387647003537725397654050699787263240396 4
Elliptic Curve Y-Coordinate:

 670793565039084272923789184408894120026594857316835707373651279535545085537 3
Certificate Extensions:
     Subject Key Identifier Extension:
          OID:  2.5.29.14
          Is Critical:  false
          Key Identifier:

 01:4b:69:99:93:5f:76:51:39:95:61:cc:a9:a8:cb:16:f2:0f:8c:c8
     Basic Constraints Extension:
          OID:  2.5.29.19
          Is Critical:  false
          Is CA:  true
          Path Length Constraint:  0
     Key Usage Extension:
          OID:  2.5.29.15
          Is Critical:  false
          Key Usages:
               Key Cert Sign
               CRL Sign
SHA-1 Fingerprint:
 b8:d0:16:9b:5d:f2:e7:a1:80:79:95:a2:64:b5:aa:ad:80:23:64:16
SHA-256 Fingerprint:

 cf:98:2a:66:35:6e:6d:f9:5d:25:c6:68:68:04:5a:a8:88:43:ca:b5:c8:e5:c9:95:09:
   e9:fc:ab:b9:41:ec:71
The certificate has a valid signature.
```

Generating self-signed certificates

The process of creating a self-signed certificate is straightforward because a self-signed certificate claims itself as its own issuer.

Although self-signed certificates are convenient for testing environments, clients do not trust them by default. Consequently, you should not use them as listener certificates in production environments.

The **`manage-certificates`** tool offers a **`generate-self-signed-certificate`** subcommand that can create a self-signed certificate. In addition to the arguments that provide information about the keystore, certificate alias, and optional private key password, the following arguments are available.

Argument	Description
`--subject-dn {subject}`	Subject DN for the certificate to create. This value is required.

Argument	Description
`--days-valid {number}`	Number of days that the certificate remains valid. Defaults to **`365`** if no value is specified.
`--validity-start-time {timestamp}`	Indicates the time at which the certificate begins its validity window. This value is assumed to reflect the local time zone, and must be expressed in the form **`YYYYMMDDhhmmss`**, where a value of **`20190102030405`** indicates January 2, 2019, at 3:04:05 AM. Defaults to the current time if no value is specified.
`--key-algorithm {name}`	Name of the algorithm to use when generating the key pair. For a listener certificate, this value is typically **`RSA`** or **`EC`**. Defaults to **`RSA`** if no value is specified. **Note:** This argument cannot be used in conjunction with the `--replace-existing-certificate` argument.
`--key-size-bits {number}`	Length of the key, in bits, to generate. If the `--key-algorithm` argument is given, then **`--key-size-bits {number}`** must also be specified. Conversely, if the `--replace-existing-certificate` argument is given, then `--key-size-bits {number}` must not be specified. Typical key sizes are: ▪ RSA key – 2048 or 4096 bits If a default RSA key is used but this argument is not provided, a default key size of 2048 bits is used. ▪ Elliptic curve key – 256 or 384 bits

Argument	Description
`--signature-algorithm {name}`	Name of the algorithm to use to sign the certificate. If the `--key-algorithm` argument is used to specify an algorithm other than **`RSA`**, then `--signature-algorithm {name}` must also be specified. If the `--replace-existing-certificate` argument is used, then `--signature-algorithm {name}` must not be specified. Typical signature algorithms include **`SHA256withRSA`** for certificates with RSA keys, and **`SHA256withECDSA`** for certificates with elliptic curve keys. If a default key algorithm is used but the `--signature-algorithm {name}` argument is not provided, a default value of **`SHA256withRSA`** is used.
`--replace-existing-certificate`	Uses the new certificate to replace an existing certificate in the key store (within the same alias), and reuses the key for that certificate.
`--inherit-extensions`	Indicates that, when replacing an existing certificate, the new certificate contains the same set of extensions as the existing certificate. If the `--replace-existing-certificate` argument is provided, but the `--inherit-extensions` argument is omitted, the new certificate contains only arguments that are provided explicitly.
`--subject-alternative-name-dns {name}`	Indicates that the certificate is expected to have a subject alternative name extension with the provided DNS name. The given name must be fully qualified, although it can contain an asterisk (**`*`**) as a wildcard in the leftmost component. To include multiple DNS names in the subject alternative name extension, specify the `--subject-alternative-name-dns {name}` argument multiple times.
`--subject-alternative-name-ip-address {address}`	Indicates that the certificate is expected to have a subject alternative name extension with the provided IP address. The given address must be a valid IPv4 or IPv6 address. No wildcards are allowed. To include multiple IP addresses in the subject alternative name extension, specify the `--subject-alternative-name-ip-address {address}` argument multiple times.

Argument	Description
`--subject-alternative-name-email-address {address}`	Indicates that the certificate is expected to have a subject alternative name extension with the provided email address. To include multiple email addresses in the subject alternative name extension, specify the `--subject-alternative-name-email-address {address}` argument multiple times.
`--subject-alternative-name-uri {uri}`	Indicates that the certificate is expected to have a subject alternative name extension with the provided URI. To include multiple URIs in the subject alternative name extension, specify the `--subject-alternative-name-uri {uri}` argument multiple times.
`--subject-alternative-name-oid {oid}`	Indicates that the certificate is expected to have a subject alternative name extension with the provided object identifier (OID). The given value must be a valid OID. To include multiple OIDs in the subject alternative name extension, specify the `--subject-alternative-name-oid {oid}` argument multiple times.
`--basic-constraints-is-ca {value}`	Indicates that the certificate is expected to have a basic constraints extension, with a specified value of **`true`** or **`false`**, for the flag indicating whether to consider the certificate a certification authority that can be used to sign other certificates. ▪ For root and intermediate certificate authority (CA) certificates, the `--basic-constraints-is-ca {value}` argument must be present with a value of **`true`**. ▪ For end-entity certificates, the `--basic-constraints-is-ca {value}` argument can optionally be present with a value of **`false`**. ▪ For a self-signed certificate, specify the `--basic-constraints-is-ca {value}` argument with a value of **`false`** to indicate that the certificate is not considered a CA certificate.

Argument	Description
`--basic-constraints-maximum-path-length {number}`	Indicates that the basic constraints extension is expected to include a path length constraint element with the specified value. Use this argument only if `--basic-constraints-is-ca` is provided with a value of **`true`**. A path length constraint value of **`0`** indicates that the certificate can be used to issue only end-entity certificates. A path length constraint value of **`1`** indicates that the certificate can be used to sign end-entity certificates or intermediate CA certificates, the latter of which can be used to sign only end-entity certificates. A value greater than **`1`** indicates the presence of several intermediate CA certificates between it and the end-entity certificate at the head of the chain.
`--key-usage {value}`	Indicates that the certificate is expected to have a key usage extension with the specified value. The following values are allowed: ▪ **`digital-signature`** ▪ **`non-repudiation`** ▪ **`key-encipherment`** ▪ **`data-encipherment`** ▪ **`key-agreement`** ▪ **`key-cert-sign`** ▪ **`crl-sign`** ▪ **`encipher-only`** ▪ **`decipher-only`** To include multiple key usages, specify the `--key-usage {value}` argument multiple times.
`--extended-key-usage {value}`	Indicates that the certificate is expected to have an extended key usage extension with the specified value. The following values are allowed: ▪ **`server-auth`** ▪ **`client-auth`** ▪ **`code-signing`** ▪ **`email-protection`** ▪ **`time-stamping`** ▪ **`ocsp-signing`**

Example

For example, the following command can be used to generate a self-signed server certificate.

```
bin/manage-certificates generate-self-signed-certificate \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --keystore-type JKS \
     --alias server-cert \
```

```
     --subject-dn "CN=ds.example.com,O=Example Corp,C=US" \
     --key-algorithm EC \
     --key-length-bits 256 \
     --signature-algorithm SHA256withECDSA \
     --subject-alternative-name-dns ds.example.com \
     --subject-alternative-name-dns ds1.example.com \
     --subject-alternative-name-dns localhost \
     --subject-alternative-name-ip-address 1.2.3.4 \
     --subject-alternative-name-ip-address 127.0.0.1 \
     --subject-alternative-name-ip-address 0:0:0:0:0:0:0:1 \
     --key-usage digital-signature \
     --key-usage key-encipherment \
     --key-usage key-agreement \
     --extended-key-usage server-auth \
     --extended-key-usage client-auth

Successfully created a new JKS keystore.

Successfully generated the following self-signed certificate:
Subject DN:  CN=ds.example.com,O=Example Corp,C=US
Issuer DN:  CN=ds.example.com,O=Example Corp,C=US
Validity Start Time: Monday, January 27, 2020 at 03:40:13 PM CST
                     (0 seconds ago)
Validity End Time: Tuesday, January 26, 2021 at 03:40:13 PM CST
                   (364 days, 23 hours, 59 minutes, 59 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP256r1)
SHA-1 Fingerprint:
 4f:41:82:7f:08:e9:d8:05:8c:19:8b:3e:5b:bc:59:98:d3:15:71:3a
SHA-256 Fingerprint:

 76:e6:8e:c5:c8:8d:27:ce:2b:85:b9:8c:9d:49:3c:06:f4:40:f1:d0:70:67:39:24:fc:
   31:bc:f8:51:83:f2:42
```

Generating certificate signing requests

A certificate signing request (CSR) contains all of the information that a certification authority requires to issue a certificate.

RFC 2986 defines the request format, also known as PKCS #10, and includes the following elements:

- Certificate signing request version
- Requested subject distinguished name (DN) for the certificate
- Public key for the requested certificate
- Requested set of extensions for the certificate
- Signature that proves the requester has the private key for the given public key

To create a certificate signing request, use the **`manage-certificates generate-certificate-signing-request`** command, which performs the following steps:

1. Generated a public and private key pair.
2. Stores the key pair in a key store with a given alias.
3. Outputs the certificate signing request to the terminal.
4. Optionally writes the certificate signing request to a file.

Because a certificate signing request contains many of the same elements as a certificate, the command to generate one takes most of the same arguments as for generating a self-signed certificate. The following arguments are unavailable when generating a CSR:

- `--replace-existing-certificate`
- `--days-valid {number}`
- `--validity-start-time {timestamp}`

The following arguments are available when generating a certificate signing request but not when generating a self-signed certificate:

`--output-file {path}`

Path to a file to which the certificate signing request is written. If this value is not provided, the request is written only to the terminal in PEM form.

`--output-format {value}`

Format to use when writing the certificate signing request. This value can be **`PEM`** or **`DER`**, but the DER format is used only in conjunction with the `--output-file` argument. Defaults to **`PEM`** if the `--output-format {value}` argument is not provided.

`--use-existing-key-pair`

Indicates that the CSR uses a key pair that already exists in the key store with the given alias, rather than generating a new key pair, in which case the specified alias must not already be in use in the key store.

The following example command creates a CSR.

```
bin/manage-certificates generate-certificate-signing-request \
     --output-file ds1-cert.csr \
     --output-format PEM \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --keystore-type JKS \
     --alias server-cert \
     --subject-dn "CN=ds.example.com,O=Example Corp,C=US" \
     --key-algorithm EC \
     --key-length-bits 256 \
     --signature-algorithm SHA256withECDSA \
     --subject-alternative-name-dns ds.example.com \
     --subject-alternative-name-dns ds1.example.com \
     --subject-alternative-name-dns localhost \
     --subject-alternative-name-ip-address 1.2.3.4 \
     --subject-alternative-name-ip-address 127.0.0.1 \
     --subject-alternative-name-ip-address 0:0:0:0:0:0:0:1 \
     --key-usage digital-signature \
     --key-usage key-encipherment \
     --key-usage key-agreement \
     --extended-key-usage server-auth \
     --extended-key-usage client-auth
```

If the contents of the resulting CSR file are made available to a certification authority to be signed, the resulting signed certificate can be imported into the key store.

To print the contents of a certificate signing request file, use the **`display-certificate-signing-request-file`** subcommand, which supports the following arguments:

`--certificate-signing-request-file {path}`

Path to the file that contains the certificate signing request to display.

`--verbose`

Indicates that the command is expected to display verbose information about the request, rather than a basic information set.

The following example demonstrates the basic output from the command.

```
$ bin/manage-certificates display-certificate-signing-request-file \
```

```
    --certificate-signing-request-file ds1-cert.csr

    PKCS #10 Certificate Signing Request Version:  v1
    Subject DN:  CN=ds.example.com,O=Example Corp,C=US
    Signature Algorithm:  SHA-256 with ECDSA
    Public Key Algorithm:  EC (secP256r1)
```

The following example demonstrates the verbose output.

```
$ bin/manage-certificates display-certificate-signing-request-file \
     --certificate-signing-request-file ds1-cert.csr \
     --verbose

PKCS #10 Certificate Signing Request Version:  v1
Subject DN:  CN=ds.example.com,O=Example Corp,C=US
Signature Algorithm:  SHA-256 with ECDSA
Signature Value:

 30:45:02:20:46:31:be:9e:6d:2f:0e:e3:d0:80:5c:88:ef:da:86:07:fd:15:b7:62:
     83:45:39:0a:c9:f2:f9:17:eb:08:94:ff:02:21:00:c8:bd:df:57:fa:ea:8c:04:
     df:c5:27:76:e5:b3:3b:4f:df:ec:d3:e4:09:5b:c0:6c:7b:86:39:ec:d0:0e:c1:64
Public Key Algorithm:  EC (secP256r1)
Elliptic Curve Public Key Is Compressed:  false
Elliptic Curve X-Coordinate:
   208628537904757963197889471667098239762296638799662436502070112279302 4
   3221133
Elliptic Curve Y-Coordinate:
   479697739226644990505743464941788269420922508654777168408919906254139
   60212095
Certificate Extensions:
     Subject Key Identifier Extension:
          OID:  2.5.29.14
          Is Critical:  false
          Key Identifier:
               f2:de:fd:bf:d3:2f:96:ef:01:70:2d:0e:85:f5:fb:17:d5:a0:9e:67
     Subject Alternative Name Extension:
          OID:  2.5.29.17
          Is Critical:  false
          DNS Name:  ds.example.com
          DNS Name:  ds1.example.com
          DNS Name:  localhost
          IP Address:  1.2.3.4
          IP Address:  127.0.0.1
          IP Address:  0:0:0:0:0:0:0:1
     Key Usage Extension:
          OID:  2.5.29.15
          Is Critical:  false
          Key Usages:
               Digital Signature
               Key Encipherment
               Key Agreement
     Extended Key Usage Extension:
          OID:  2.5.29.37
          Is Critical:  false
          Key Purpose ID:  TLS Server Authentication
          Key Purpose ID:  TLS Client Authentication
```

Importing signed and trusted certificates

Use the **`manage-certificates import-certificate`** command to import certificates into a keystore.

This command is used to accomplish the following tasks:

- Import a certificate that a certification authority has signed into the keystore in which the key pair was generated. In this scenario, the certificate is imported into a private key entry and must be imported as a certificate chain rather than an end-entity certificate.
- Import a trusted issuer certificate into a trust store. In this scenario, the certificate is imported into a trusted certificate entry as a single certificate instead of as a chain.
- Import a certificate chain, along with the private key for the end-entity certificate. This approach imports certificates that were generated through another library, like OpenSSL.

In addition to the arguments that provide information about the key store and the alias into which the certificate or certificate chain is imported, the **`manage-certificates import-certificate`** command accepts the following arguments:

`--certificate-file {path}`

Path to the file that contains the certificate to import. The certificate can be in PEM or DER format and can be a single certificate or a certificate chain. If the certificates in the chain reside in separate files, specify the **`--certificate-file {path}`** argument multiple times when you import a certificate chain.

`--private-key-file {path}`

Path to the file containing the private key that corresponds to the certificate at the head of the imported chain. The private key can be in PEM or DER format.

`--no-prompt`

Indicates that the certificate is to be imported without prompting for confirmation. By default, a summary of the certificate is displayed, and you must confirm that you want to import it.

The following example command imports a signed certificate into the key store that generates the certificate signing request.

```
$ bin/manage-certificates import-certificate \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --alias server-cert \
     --certificate-file ds1-cert.pem \
     --certificate-file ca-cert.pem

The following certificate chain will be imported into the keystore into
 alias
'server-cert', preserving the existing private key associated with that
 alias:

Subject DN:  CN=ds.example.com,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Sunday, November 10, 2019 at 09:09:23 PM CST
                     (4 minutes, 16 seconds ago)
Validity End Time: Monday, November 9, 2020 at 09:09:23 PM CST
                   (364 days, 23 hours, 55 minutes, 43 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP256r1)
SHA-1 Fingerprint:
 02:51:25:43:3e:68:f5:71:36:e3:5d:df:74:de:f6:a1:5a:db:0f:eb
SHA-256 Fingerprint: 1d:b5:eb:3c:f5:ff:bf:79:a2:a5:86:b8:e4:33:76:4d:d7:
                     50:dc:a4:34:95:37:be:89:45:86:1f:5d:79:c3:93

Subject DN:  CN=Example Root CA,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Sunday, November 10, 2019 at 09:00:07 PM CST
                     (13 minutes, 32 seconds ago)
Validity End Time: Saturday, November 5, 2039 at 10:00:07 PM CDT
```

```
                    (7299 days, 23 hours, 46 minutes, 27 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP384r1)
SHA-1 Fingerprint:
 0e:5c:21:c9:a5:36:0a:24:eb:aa:55:b6:a5:94:0e:e0:56:03:22:e6
SHA-256 Fingerprint:
 77:cf:66:d7:3c:8a:fd:67:2d:b7:36:fd:60:1d:ca:eb:1b:03:b1:
                     12:7b:10:1f:26:05:b7:b9:0d:02:e0:38:3e

Do you want to import this certificate chain into the keystore? yes

Successfully imported the certificate chain.
```

If you do not provide the `--no-prompt` argument, the **`manage-certificates import-certificate`** tool still displays information about the certificates to import. To view additional information about a certificate before you import it, use the **`display-certificate-file`** subcommand, which supports the following arguments:

`--certificate-file {path}`

Path to the file that contains the certificate to view.

`--verbose`

Displays verbose information about the certificate.

The output of the **`display-certificate-file`** subcommand has the same format and content as the **`list-certificates`** subcommand.

Exporting certificates

Use the **`export-certificate`** subcommand to export a single certificate or a certificate chain from a key store to a file in PEM or DER format.

The **`export-certificate`** subcommand supports the normal arguments about the key store and certificate alias, in addition to the following arguments:

`--output-file {path}`

Path to the file to which exported certificates are written. If this value is not provided, the certificates are written to standard output rather than a file.

`--output-format {format}`

Format in which exported certificates are written. The value can be **`PEM`** or **`DER`**, but the DER format can be used only if the output is written to a file. Defaults to **`PEM`** if no value is specified.

`--export-certificate-chain`

Indicates that a certificate chain, rather than the end-entity certificate only, is to be exported.

`--separate-file-per-certificate`

Indicates the use of separate output files for each exported certificate, rather than placing all of the certificates in a single file. If this argument is provided and multiple certificates are to be exported, then **`.1`** is appended to the path for the indicated output file for the first certificate in the chain, **`.2`** is appended for the second certificate, and so on.

The following example exports a certificate chain.

```
$ bin/manage-certificates export-certificate \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --alias server-cert \
     --output-file server-cert.pem \
```

```
     --output-format PEM \
     --export-certificate-chain \
     --separate-file-per-certificate

Successfully exported the following certificate to '/ds/server-cert.pem.1':
Subject DN:  CN=ds.example.com,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Sunday, November 10, 2019 at 09:09:23 PM CST
                     (3 hours, 26 minutes, 23 seconds ago)
Validity End Time: Monday, November 9, 2020 at 09:09:23 PM CST
                           (364 days, 20 hours, 33 minutes, 36 seconds from
 now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP256r1)
SHA-1 Fingerprint:
 02:51:25:43:3e:68:f5:71:36:e3:5d:df:74:de:f6:a1:5a:db:0f:eb
SHA-256 Fingerprint:
1d:b5:eb:3c:f5:ff:bf:79:a2:a5:86:b8:e4:33:76:4d:d7:50:dc:a4:34:95:37:be:89:45:
86:1f:5d:79:c3:93

Successfully exported the following certificate to '/ds/server-cert.pem.2':
Subject DN:  CN=Example Root CA,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Sunday, November 10, 2019 at 09:00:07 PM CST
                     (3 hours, 35 minutes, 39 seconds ago)
Validity End Time: Saturday, November 5, 2039 at 10:00:07 PM CDT
                   (7299 days, 20 hours, 24 minutes, 20 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with ECDSA
Public Key Algorithm:  EC (secP384r1)
SHA-1 Fingerprint:
 0e:5c:21:c9:a5:36:0a:24:eb:aa:55:b6:a5:94:0e:e0:56:03:22:e6
SHA-256 Fingerprint:
   77:cf:66:d7:3c:8a:fd:67:2d:b7:36:fd:60:1d:ca:eb:1b:03:b1:12:7b:10:1f:26:
   05:b7:b9:0d:02:e0:38:3e
```

The **`export-certificate`** subcommand exports only the public portion of a certificate. Its private key is not included. To export the private key, use the **`export-private-key`** subcommand, which supports the following arguments, in addition to the usual key store and alias arguments:

`--output-file {path}`

Path to the file to which the exported private key is written. If this value is not provided, the key is written to standard output rather than a file.

`--output-format {format}`

Format in which the exported private key is written. The value can be **`PEM`** or **`DER`**, but the DER format is used only if the output is written to a file. Defaults to **`PEM`** if no value is specified.

The following code provides an example of the **`export-private-key`** subcommand .

```
$ bin/manage-certificates export-private-key \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --alias server-cert \
     --output-file server-cert-key.pem \
     --output-format PEM

Successfully exported the private key.
```

Enabling TLS support during server setup
Enable TLS support in the server.

To enable TLS support in the server, you should complete one of the following tasks during the setup procedure:

- Provide a key store that contains the certificate to use.
- Make the installer generate a self-signed certificate.

When using the **`setup`** tool in interactive mode, it prompts you for the information that it needs to configure secure communication.

When using **`setup`** in non-interactive mode, use the following arguments to configure TLS support.

Argument	Description
`--ldapsPort {port}`	Server enables support for LDAPS (LDAP over TLS) on the specified TCP port.
`--httpsPort {port}`	Server enables support for HTTPS for SCIM, the Directory REST API, and the web-based administration console on the specified TCP port.
`--enableStartTLS`	LDAP connection handler enables support for the StartTLS extended operation.
`--generateSelfSignedCertificate`	**`setup`** generates a self-signed certificate that is presented to clients that use LDAPS, HTTPS, and the StartTLS extended operation.
`--useJavaKeyStore {path}`	Server uses the specified Java KeyStore (JKS) key store to obtain the certificate chain that it presents to clients that use LDAPS, HTTPS, and the StartTLS extended operation.
`--usePKCS12KeyStore {path}`	Server uses the specified PKCS #12 key store to obtain the certificate chain that it presents to clients that use LDAPS, HTTPS, and the StartTLS extended operation.
`--usePKCS11KeyStore`	Server uses a PKCS #11 key store, like a hardware security module, to obtain the certificate chain that it presents to clients that use LDAPS, HTTPS, and the StartTLS extended operation. The Java Virtual Machine (JVM) must already be configured to access the appropriate key store through PKCS #11.
`--keyStorePassword {password}`	Password that is needed to interact with the specified JKS, PKCS #12, or PKCS #11 key store. The **`setup`** tool assumes that the private key password matches the key store password.
`--keyStorePasswordFile {path}`	Path to the file that contains the password needed to interact with the specified JKS, PKCS #12, or PKCS #11 key store.

Argument	Description
`--certNickname {alias}`	Alias of the private key entry in the specified key store that contains the certificate chain to present to clients during TLS negotiation. This argument is optional but recommended if the key store contains multiple certificates.
`--useJavaTrustStore {path}`	Server uses the specified JKS trust store to determine whether to trust certificate chains that are presented to it during TLS negotiation.
`--usePKCS12TrustStore {path}`	Server uses the specified PKCS #12 trust store to determine whether to trust certificate chains that are presented to it during TLS negotiation
`--trustStorePassword {password}`	Password that is needed to interact with the specified JKS or PKCS #11 trust store.
`--trustStorePasswordFile {path}`	Path to the file that contains the password needed to interact with the specified JKS or PKCS #11 trust store.

The following example command sets up PingAuthorize in non-interactive mode with an existing certificate.

```
$ ./setup \
  --no-prompt \
  --acceptLicense \
  --ldapPort 8389 \
  --ldapsPort 8636 \
  --httpsPort 8443 \
  --enableStartTLS \
  --useJavaKeyStore config/keystore \
  --keyStorePasswordFile config/keystore.pin \
  --certNickname server-cert \
  --useJavaTrustStore config/truststore \
  --trustStorePasswordFile config/truststore.pin  \
  --rootUserDN "cn=Directory Manager" \
  --rootUserPasswordFile root-pw.txt \
  --maxHeapSize 1g \
  --location Austin \
  --instanceName paz1
  .
  .
  .

Initializing ..... Done
Configuring PingAuthorize Server ..... Done
Configuring Certificates ..... Done
Creating Encryption Settings ..... Done
Starting PingAuthorize Server ..... Done

The server is now ready for configuration. You may either run the
create-initial-config tool to continue configuration or import an
existing configuration using dsconfig.

Access product documentation from https://myhostname:8443/docs/index.html
```

Enabling TLS support after setup

If the server has been set up without support for TLS, enable TLS support later by completing the following tasks.

Steps

1. Obtain a certificate chain.

   For more information about obtaining a certificate chain, see *Certificate chains* on page 315. To prepare a Java KeyStore JKS or PKCS #12 key store with an appropriate certificate chain and private key, use the **`manage-certificates`** tool. We also recommend that you create a trust store that the server can use.
2. Configure the key and trust manager providers.

   For more information, see *Configuring key and trust manager providers* on page 343.
3. Configure connection handlers.

   For more information, see *Configuring TLS connection handlers* on page 343.

Configuring key and trust manager providers

After you have a key store, configure a key manager provider to access it.

The server is preconfigured with key manager providers, `JKS` and `PKCS12`, that you can use with JKS or PKCS #12 key stores, respectively. You can update the appropriate key manager provider in most cases to reference the key store that you plan to use. The following code provides an example.

```
dsconfig set-key-manager-provider-prop \
     --provider-name JKS \
     --set enabled:true \
     --set key-store-file:config/keystore \
     --set key-store-pin-file:config/keystore.pin
```

A similar change configures a trust manager provider to reference the appropriate trust store. The following code provides an example.

```
dsconfig set-trust-manager-provider-prop \
     --provider-name JKS \
     --set enabled:true \
     --set include-jvm-default-issuers:true \
     --set trust-store-file:config/truststore \
     --set trust-store-pin-file:config/truststore.pin
```

Note:

If all clients and servers use certificates that are signed by issuers and are included in the JVM's default trust store, you can use the `JVM-Default` trust manager provider to accomplish this task.

Configuring TLS connection handlers

After you configure the key and trust manager providers, update the connection handlers to use the key and trust manager providers.

Steps

- For the LDAP connection handler, use the following command to enable StartTLS with a configuration change. By default, the LDAP connection handler accepts non-secure connections.

  Example:

  ```
  dsconfig set-connection-handler-prop \
    --handler-name "LDAP Connection Handler" \
  ```

```
  --set allow-start-tls:true \
  --set key-manager-provider:JKS \
  --set trust-manager-provider:JKS \
  --set ssl-cert-nickname:server-cert \
  --set ssl-client-auth-policy:optional
```

- If you did not configure secure communication during setup, the LDAPS connection handler is disabled. To configure LDAPS support in this scenario, enable the connection handler and configure most of the same settings. You must set **`allow-start-tls`** to `false` and **`use-ssl`** to `true`. See the following code for an example configuration.

  Example:

  ```
  dsconfig set-connection-handler-prop \
    --handler-name "LDAPS Connection Handler" \
    --set enabled:true \
    --set key-manager-provider:JKS \
    --set trust-manager-provider:JKS \
    --set ssl-cert-nickname:server-cert \
    --set ssl-client-auth-policy:optional
  ```

  Example: The following example uses a similar configuration change to enable the HTTPS connection handler.

  ```
  dsconfig set-connection-handler-prop \
    --handler-name "HTTPS Connection Handler" \
    --set enabled:true \
    --set listen-port:443 \
    --set key-manager-provider:JKS \
    --set trust-manager-provider:JKS \
    --set ssl-cert-nickname:server-cert
  ```

Updating the topology registry

After the server connection handlers are updated to enable TLS, update the topology registry to provide information about the new configuration.

The topology registry holds information about server instances that are part of the environment, and it helps to facilitate inter-server communication, such as replication, mirroring portions of the configuration, and the PingAuthorize automatic backend server-discovery functionality.

The following table details the two types of entries that require updating.

Configuration types and their update descriptions

Configuration Type	Update description

Server instance listener configuration	▪ Provides information that is needed to trust the TLS certificates that instances in the topology present.
	▪ The server instance listener configuration must include the server certificate, which is defined as the certificate at the head of the chain. This version must be the multi-line, PEM-formatted representation of the certificate. You can use **`dsconfig`** to import the certificate from a file, as shown in the following example.
	`bin/dsconfig set-server-instance-listener-prop \`
	`  --instance-name ds1 \`
	`  --listener-name ldap-listener-mirrored-config \`
	`  --set server-ldap-port:636 \`
	`  --set connection-security:ssl \`
	`  --set 'listener-certificate>/ca/ds1-cert.pem'`
	Note:
	The less-than operator > in the final line indicates that the value is read from a file rather than provided directly. In addition, you might not need to enclose the property name and path within single straight quotes to prevent the shell from interpreting the less-than symbol as an attempt to redirect input.

Configuration Type	Update description

Server instance configuration	▪ Provides information about options for communicating with those instances.
	▪ Update the server instance configuration object to reflect the new methods that are available for communication with the instance. For example, the `preferred-security` property identifies the mechanism by which other instances in the topology attempt to communicate with the instance.
	
	The following example code sets the LDAPS and HTTPS ports, indicates that StartTLS support is enabled, and instructs other instances to use SSL (LDAPS) when communicating with the instance.
	
	`dsconfig set-server-instance-prop \`
	`  --instance-name ds1 \`
	`  --set ldaps-port:636 \`
	`  --set https-port:443 \`
	`  --set preferred-security:ssl \`
	`  --set start-tls-enabled:true`

Troubleshooting TLS-related issues
Use this section for troubleshooting problems that might arise during TLS configuration, including communication and security issues that affect clients as well as PingAuthorize.

- *Log messages*
- *manage-certificates check-certificate-usability*
- *ldapsearch*
- *Using low-level TLS debugging*

Log messages
The following describes how to use the server's log messages to troubleshoot TLS-related issues.

To troubleshoot TLS-related issues, start by checking the server's access log. If the client can establish a TCP connection to the server, which must occur before TLS negotiation can start, the access log shows a `CONNECT` message with the following information:

- Source and destination address and port for the connection
- Protocol
- Selected client connection policy

The `CONNECT` message does not appear

If the `CONNECT` does not appear, the client might be unable to communicate with the server. The culprit can be a network problem, a firewall that is blocking attempts to communicate, or the client is trying to use an incorrect address or port.

The `CONNECT` message does appear

If the `CONNECT` message appears in the access log, it typically includes a `conn` element that specifies the connection ID. To view additional log messages for the client connection, use the **`search-logs`** tool. For example, if the connection ID is `12345`, the following command displays the complete set of associated log messages.

```
$ bin/search-logs --logFile logs/access conn=12345
```

If you are using LDAPS

If you are attempting to use LDAPS, one of the following log messages appears next:

- `SECURITY-NEGOTIATION` message – Indicates that the client and server successfully completed the negotiation process and that the issue likely occurred after the TLS session was established. This message also includes details about the negotiation, including the TLS protocol and the selected cipher suite.
- `DISCONNECT` message – The issue might involve a failure in the TLS-negotiation process. In such scenarios, the message usually includes a `reason` element that provides additional information about the reason for the disconnect.

  If the failure occurred during TLS negotiation, the usefulness of the `DISCONNECT` message depends in part on whether the failure occurred on the client or the server. For example, if the server decided to abort the negotiation, the message ideally contains the specific reason. If the problem occurred on the client, the log message likely contains only the general category for the failure.

> **Note:**
>
> The TLS protocol does not provide a mechanism for conveying detailed error messages. Instead, it offers only a basic alert mechanism with a fixed set of alert types. For example, if a client does not trust the certificate chain that the server presents to it, the server might receive a generic alert like `certificate_unknown`, even if the client knows the precise reason for rejecting the chain. In such instances, you might need to determine whether the client can provide additional details about the issue.

If the access log does not provide useful information

If the access log does not provide useful information, check the server error log. Although the error log does not normally include information about issues that relate to client communication, it provides helpful information in certain circumstances, like when an internal error within the server interferes with communication attempts.

About manage-certificates check-certificate-usability

The **`manage-certificates`** tool offers a **`check-certificate-usability`** subcommand to examine a specified entry in a key store and to identify potential issues that might interfere with secure communication.

The **`check-certificate-usability`** tool completes the following tasks:

- Ensures that a specified entry in the key store includes a private key and a complete certificate chain
- Checks whether the certificate at the root of the chain is found in the Java virtual machine's (JVM's) default set of trusted certificates
- Ensures that the current time lies is within the validity window for all certificates in the chain
- Validates the signatures for all certificates in the chain
- Warns if the end-entity certificate is self-signed
- Warns if the end-entity certificate does not contain an extended key usage extension with the `serverAuth` usage
- Warns if the issuer certificates do not have a key usage extension with the `keyCertSign` usage
- Warns if the issuer certificates do not have a basic constraints extension indicating that it can operate as a certification authority

  If the chain violates a path length constraint, the **`check-certificate-usability`** tool reports an error.
- Ensures that the signature algorithm uses a strong message digest algorithm, like SHA-256

  The **`check-certificate-usability`** tool reports an error for weak digest algorithms like MD5 or SHA-1, and reports a warning for unrecognized digest algorithms.

- Ensures that none of the certificates that use an RSA key pair have a key size less than 2048 bits

The following example demonstrates the usage for the **`manage-certificates check-certificate-usability`** command and its output when no problems are identified.

```
$ bin/manage-certificates check-certificate-usability \
     --keystore config/keystore \
     --keystore-password-file config/keystore.pin \
     --alias server-cert

Successfully retrieved the certificate chain for alias 'server-cert':

Subject DN:  CN=ds1.example.com,O=Example Corp,C=US
Issuer DN:  CN=Example Intermediate CA,O=Example Corp,C=US
Validity Start Time: Tuesday, November 12, 2019 at 03:52:44 PM CST
                     (5 minutes, 45 seconds ago)
Validity End Time: Wednesday, November 11, 2020 at 03:52:44 PM CST
                   (364 days, 23 hours, 54 minutes, 14 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with RSA
Public Key Algorithm:  RSA (2048-bit)
SHA-1 Fingerprint:
 84:e4:00:b9:f0:6b:58:bb:ac:67:79:28:2f:43:9f:e3:ac:24:ee:98
SHA-256 Fingerprint:
 63:85:4d:2c:50:ea:a8:84:54:e0:73:9a:e7:5b:e7:1b:06:85:0e:
                     28:2b:76:a9:8b:57:fc:27:f7:60:81:48:41

Subject DN:  CN=Example Intermediate CA,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Tuesday, November 12, 2019 at 03:52:42 PM CST
                     (5 minutes, 47 seconds ago)
Validity End Time: Monday, November 7, 2039 at 03:52:42 PM CST
                   (7299 days, 23 hours, 54 minutes, 12 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with RSA
Public Key Algorithm:  RSA (4096-bit)
SHA-1 Fingerprint:
 de:da:3d:fc:d4:1f:67:79:0a:a1:5a:cd:ca:4a:7e:a5:d3:46:88:27
SHA-256 Fingerprint:

 02:3c:af:ad:b7:07:81:89:45:48:d0:09:31:a8:90:c4:17:11:1c:00:11:fd:49:b2:2c:
   ba:ac:dd:c4:9f:03:36

Subject DN:  CN=Example Root CA,O=Example Corp,C=US
Issuer DN:  CN=Example Root CA,O=Example Corp,C=US
Validity Start Time: Tuesday, November 12, 2019 at 03:52:38 PM CST
                     (5 minutes, 51 seconds ago)
Validity End Time: Monday, November 7, 2039 at 03:52:38 PM CST
                   (7299 days, 23 hours, 54 minutes, 8 seconds from now)
Validity State:  The certificate is currently within the validity window.
Signature Algorithm:  SHA-256 with RSA
Public Key Algorithm:  RSA (4096-bit)
SHA-1 Fingerprint:
 8e:03:e4:58:e6:e3:59:9a:55:77:c0:88:3c:fa:d7:29:f4:ff:de:6c
SHA-256 Fingerprint:
 95:54:0d:e2:aa:48:29:c1:25:7c:20:69:c0:27:33:31:81:07:02:
                     2e:00:24:ae:49:5e:98:bd:a3:72:a5:05:26

OK:  The certificate chain is complete.  Each subsequent certificate is
the issuer for the previous certificate in the chain, and the chain ends
with a self-signed certificate.

OK:  Certificate 'CN=ds1.example.com,O=Example Corp,C=US' has a valid
```

```
signature.

OK:  Certificate 'CN=Example Intermediate CA,O=Example Corp,C=US' has a
valid signature.

OK:  Certificate 'CN=Example Root CA,O=Example Corp,C=US' has a valid
signature.

OK:  Certificate 'CN=ds1.example.com,O=Example Corp,C=US' will expire at
Wednesday, November 11, 2020 at 03:52:44 PM CST (364 days, 23 hours, 54
minutes, 14 seconds from now), which is not in the near future.

OK:  Issuer certificate 'CN=Example Intermediate CA,O=Example Corp,C=US'
will expire at Monday, November 7, 2039 at 03:52:42 PM CST (7299 days, 23
hours, 54 minutes, 12 seconds from now), which is not in the near future.

OK:  Issuer certificate 'CN=Example Root CA,O=Example Corp,C=US' will
expire at Monday, November 7, 2039 at 03:52:38 PM CST (7299 days, 23
hours, 54 minutes, 8 seconds from now), which is not in the near future.

OK:  Certificate 'CN=ds1.example.com,O=Example Corp,C=US' at the head of
the chain includes an extended key usage extension, and that extension
includes the serverAuth usage.

OK:  Issuer certificate 'CN=Example Intermediate CA,O=Example Corp,C=US'
includes a basic constraints extension, and the certificate chain
satisfies those constraints.

OK:  Issuer certificate 'CN=Example Intermediate CA,O=Example Corp,C=US'
includes a key usage extension with the keyCertSign usage flag set to
true.

OK:  Issuer certificate 'CN=Example Root CA,O=Example Corp,C=US' includes
a basic constraints extension, and the certificate chain satisfies those
constraints.

OK:  Issuer certificate 'CN=Example Root CA,O=Example Corp,C=US' includes
a key usage extension with the keyCertSign usage flag set to true.

OK:  Certificate 'CN=ds1.example.com,O=Example Corp,C=US' uses a signature
algorithm of 'SHA-256 with RSA', which is is considered strong.

OK:  Certificate 'CN=Example Intermediate CA,O=Example Corp,C=US' uses a
signature algorithm of 'SHA-256 with RSA', which is is considered strong.

OK:  Certificate 'CN=Example Root CA,O=Example Corp,C=US' uses a signature
algorithm of 'SHA-256 with RSA', which is is considered strong.

OK:  Certificate 'CN=ds1.example.com,O=Example Corp,C=US' has a 2048-bit
RSA public key, which is considered strong.

OK:  Certificate 'CN=Example Intermediate CA,O=Example Corp,C=US' has a
4096-bit RSA public key, which is considered strong.

OK:  Certificate 'CN=Example Root CA,O=Example Corp,C=US' has a 4096-bit
RSA public key, which is considered strong.

No usability errors or warnings were identified while validating the
certificate chain.
```

If any usability issues are identified, they might be responsible for communication problems.

ldapsearch for TLS-related arguments

The **ldapsearch** command-line utility is a powerful tool for issuing searches against an LDAP directory server. It also provides a convenient method for troubleshooting a variety of issues, including problems that are relevant to TLS communication.

The following table details arguments that are the most useful for TLS-related communication.

TLS-related communication arguments and their descriptions

Argument	Description
--hostname {address}	Address of the server to which the connection is established
--port {port}	TCP port of the server to which the connection is established. The standard port for non-secure LDAP, or LDAP to be secured with StartTLS, is **389**, and the standard port for secure LDAPS is **636**. Many deployments use alternate ports, especially non-privileged ports above **1024**.
--useSSL	The tool establishes an initially insecure LDAP connection, which is secured later with the StartTLS extended operation.
--trustStorePath {path}	Path to the trust store that is used when determining whether to trust the certificate chain that the server presents during TLS negotiation. If neither this argument nor the **--trustAll** argument is provided, the tool prompts you interactively whether to trust server certificates that are not signed by an issuer in the Java virtual machine's (JVM's) default trust store.
--trustStoreFormat {format}	Format for the trust store, which is typically **JKS** or **PKCS12**.
--trustStorePassword {password}	Password that is required to access the contents of the trust store.
--trustStorePasswordFile {path}	Path to the file that contains the password that is required to access the contents of the trust store.
--trustAll	The tool blindly trusts all TLS certificate chains that are presented to it. Although this argument can prove useful for troubleshooting purposes, it is not recommended for general use.

Argument	Description
--keyStorePath {path}	Path to the key store to use if a client certificate chain is presented to the server. **Note:** Use this argument only when one of the following conditions is satisfied: ▪ The server is configured to require clients to present a certificate. ▪ You intend to use the certificate to authenticate through SASL EXTERNAL.
--keyStoreFormat {format}	Format for the key store, which is typically **JKS** or **PKCS12**.
--keyStorePassword {password}	Password to access the key store.
--keyStorePasswordFile {path}	Path to the file that contains the password necessary to access the key store.
--certNickname {alias}	Alias of the private key entry in the key store. Use when obtaining the certificate chain to present to the server.
--useSASLExternal	The client authenticates with the EXTERNAL SASL mechanism, which typically identifies the client using the certificate chain that is presented during TLS negotiation.
--enableSSLDebugging	The tool activates the low-level TLS-debugging feature that the JVM provides.

The following command provides an example of the simplest method for testing TLS communication with PingAuthorize Server.

Example

```
$ bin/ldapsearch \
     --hostname ds1.example.com \
     --port 636 \
     --useSSL \
     --baseDN "" \
     --scope base \
     "(objectClass=*)"
The server presented the following certificate chain:

     Subject: CN=ds1.example.com,O=Example Corp,C=US
     Valid From: Tuesday, November 12, 2019 at 08:28:08 PM CST
     Valid Until: Wednesday, November 11, 2020 at 08:28:08 PM CST
     SHA-1 Fingerprint:
        6a:22:2a:bd:0b:1b:09:35:63:bc:12:3e:2c:9e:e7:70:bc:a4:73:de
     256-bit SHA-2 Fingerprint:
        7a:8c:e4:76:d4:47:15:fd:65:f5:26:0e:d2:55:77:d7:03:7a:e6:79:9f:bc:
        ae:93:2c:76:9c:01:fc:ef:15:38
```

```
     -
     Issuer 1 Subject: CN=Example Intermediate CA,O=Example Corp,C=US
     Valid From: Tuesday, November 12, 2019 at 08:28:06 PM CST
     Valid Until: Monday, November 7, 2039 at 08:28:06 PM CST
     SHA-1 Fingerprint:
 01:b3:70:8b:6c:11:43:87:3b:e9:bb:73:27:99:ea:fd:08:c4:db:ec
     256-bit SHA-2 Fingerprint:
 49:60:69:df:33:9d:26:d0:66:c9:6d:7b:0b:cb:3b:96:

 40:22:dc:6d:11:32:b7:c0:30:47:d6:7c:6a:19:cd:60
     -
     Issuer 2 Subject: CN=Example Root CA,O=Example Corp,C=US
     Valid From: Tuesday, November 12, 2019 at 08:28:03 PM CST
     Valid Until: Monday, November 7, 2039 at 08:28:03 PM CST
     SHA-1 Fingerprint:
 b4:83:55:db:82:e4:63:5c:3a:44:13:8f:88:44:e3:60:f2:53:80:48
     256-bit SHA-2 Fingerprint:

 e8:af:6f:ed:b9:0e:df:94:9c:20:29:53:a9:74:44:a9:17:b4:08:65:c8:19:c1:fb:
        34:34:a1:90:83:8a:d5:12

Do you wish to trust this certificate?  Enter 'y' or 'n': y
dn:
objectClass: top
objectClass: ds-root-dse
startupUUID: 8d574122-4584-4522-96d9-0cdcb9d2e339
startTime: 201911130611492

# Result Code:  0 (success)
# Number of Entries Returned:  1
```

Trust stores and trust-related arguments

If no trust-related arguments are provided, the tool uses the JVM's default trust store to verify whether to trust the certificate chain, based on the information that it contains. If a trusted authority has signed the server certificate, the negotiation process continues without further interaction.

If the chain cannot be trusted, based on the information in the JVM-default trust store, **`ldapsearch`** prompts you interactively about whether to trust the certificate. If you accept the chain, the client and server complete the negotiation process, and the client sends the search request to the server. If the search succeeds, the server can communicate over TLS.

To test with a trust store instead of being prompted interactively, use the **`--trustStorePath`** argument that points to the appropriate trust store. If you are using a Java Keystore (JKS) trust store, you might not need to provide the trust store password. If you are using a PKCS #12 trust store, you need to provide the trust store password. The following code provides an example.

Example

```
$ bin/ldapsearch \
     --hostname ds1.example.com \
     --port 636 \
     --useSSL \
     --trustStorePath config/truststore.p12 \
     --trustStorePasswordFile config/truststore.pin \
     --trustStoreFormat PKCS12 \
     --baseDN "" \
     --scope base \
     "(objectClass=*)"
dn:
objectClass: top
objectClass: ds-root-dse
```

```
startupUUID: c8724159-8c37-45eb-b210-879bfcf74ad6
startTime: 20191113154023Z

# Result Code:  0 (success)
# Number of Entries Returned:  1
```

Client certificate chains and key stores

To present a client certificate chain to the server, either because the server's connection handler is configured with an **`ssl-client-auth-policy`** value of **`required`** or because you plan to use the certificate to authenticate by way of the SASL EXTERNAL mechanism, provide at least the key store and its corresponding password. You can also specify the alias of the certificate chain to present, which is recommended if your client key store contains multiple certificates. The following code provides an example.

Example

```
$ bin/ldapsearch \
     --hostname ds1.example.com \
     --port 636 \
     --useSSL \
     --trustStorePath config/truststore.p12 \
     --trustStorePasswordFile config/truststore.pin \
     --trustStoreFormat PKCS12 \
     --keyStorePath client-keystore \
     --keyStorePasswordFile client-keystore.pin \
     --certNickname client-cert \
     --useSASLExternal \
     --baseDN "" \
     --scope base \
     "(objectClass=*)"
dn:
objectClass: top
objectClass: ds-root-dse
startupUUID: c8724159-8c37-45eb-b210-879bfcf74ad6
startTime: 20191113154023Z

# Result Code:  0 (success)
# Number of Entries Returned:  1
```

If you need to further troubleshoot a TLS-related issue

If you encounter a TLS-related issue that you cannot resolve by examining the **`ldapsearch`** output or the server logs, use the **`--enableSSLDebugging`** option to enable the JVM's support for low-level debugging of TLS processing. For more information, see *Using low-level TLS debugging*.

Using low-level TLS debugging
Use tools other than the command-line tools that are provided with PingDirectory Server for performing low-level TLS debugging.

Before you begin

Note:

If you need to use low-level debugging options, enable the Java Virtual Machine (JVM)'s support for TLS debugging. Many of the command-line tools that are provided with PingDirectory Server, such as **`ldapsearch`**, offer an **`--enableSSLDebugging`** argument that simplifies this process.

Steps

1. In the `config/java.properties` file, add the following line to the set of properties for the appropriate tool.

   ```
   -Djavax.net.debug=all
   ```

2. For the changes to take effect, run the **`bin/dsjavaproperties`** command.

Next steps

The next time the tool is run, an output is generated detailing the TLS-related processing that the JVM is performing. You and the support team can use the output to identify the issue.

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 258.

User store configuration

If you want to control data access at the user level, configure PingAuthorize Server to use a user store so you can 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 PingAuthorize Server assumes that PingDirectory Server is the default user store, other LDAPv3-compliant directories are also supported.

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

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

create-initial-config

The **`create-initial-config`** command configures connectivity between PingAuthorize Server and the user store. It also creates a System for Cross-domain Identity Management (SCIM) resource type through which PingAuthorize 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 *User profile availability in policies* on page 288.

For more information about configuring SCIM, see *About the SCIM service* on page 192.

Example

For an example, see *Configuring the PingAuthorize user store* on page 359.

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. PingAuthorize 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 290.

Configure PingOne to use SSO for the administrative console

The steps below explain how to configure PingOne so that you can use SSO in PingOne to access the PingAuthorize administration console.

Before you begin
You should have already set up the PingAuthorize server that will be administered. This server will host the PingAuthorize administration console that is being configured for SSO.

Tip:

You can use groups to organize user identities as explained in *Groups*. Also, you can set access to applications as explained in *Application access control*.

Steps

1. In the PingOne administration console, add a PingAuthorize Server service to one of the existing environments. Alternatively, add a custom environment solely for a PingAuthorize Server service.
   a. When prompted, select the `It's already been deployed` option.
   b. Provide `https://<hostname>:<port>/console/login` as the value for the Admin URL, filling in the bracketed values with the PingAuthorize server's hostname and HTTP port.

   Tip:

   By binding to the LDAP server, you can use a single console instance to administer multiple PingAuthorize servers. Note that an LDAPS scheme is always assumed because an encrypted connection is always required for SSO.

   You can specify the LDAP server to bind to using the query parameters `ldap-hostname` and `ldaps-port` when the administrative console is configured for SSO. Using these parameters, you can specify the URL as follows:

   ```
   https://<hostname>:<port>/console?ldap-hostname=<my-ldap-host>&ldaps-port=<my-ldaps-port>
   ```

2. Configure the matching administrator accounts for PingOne and the PingAuthorize server. Go to the PingOne dashboard for the environment that will be used with the PingAuthorize server. Repeat the following steps for each PingOne user for which you wish to enable SSO.
   a. Locate the desired user under the **Identities** tab. For the example purposes, we will assume the desired PingOne user has the following properties.

Description	Details
Given Name	Jane
Family Name	Smith
Username	jsmith

   b. Run the following **`dsconfig`** command against the PingAuthorize server, filling in the bracketed field with the previously located PingOne user's **Username** value.

```
dsconfig create-root-dn-user --user-name jsmith \
  --set first-name:Jane \
  --set last-name:Smith
```

3. Register the administrative console with PingOne. Follow the instructions for *Adding an application* and select **OIDC Web App** for **Application Type**. Configure the application properties as shown in the following table.

Property	Value
Application Name	PingAuthorize administrative console
Description	Application for the PingAuthorize administrative console
Redirect URLs	`https://<hostname>:<port>/console/oidc/cb`
Attribute Mapping	`Username` = `sub`

> **Note:**
>
> Fill in the bracketed values in redirect URLs with the PingAuthorize server's hostname and HTTP port, similar to Step 2.

4. Edit the listed properties for the newly created application so that the properties have the values show in the following table, following the instructions in *Edit an application - OIDC* in the PingOne Administration Guide.

Property	Value
Response Type	Code
Grant Type	Authorization Code
Token Endpoint Authentication Method	Client Secret Basic

5. Note the values for the following application properties to use in later steps:
   - **Issuer**
   - **Client ID**
   - **Client Secret**
6. Locate the `enable-pingone-admin-console-sso.dsconfig` file in the `PingAuthorize/config/sample-dsconfig-batch-files/` directory. Make a copy of it, and edit the copy rather than the source file.
7. Replace all the bracketed values in the batch file with the corresponding values from step 5. Then run the file using the following command.

   ```
   dsconfig --batch-file \
       enable-pingone-admin-console-sso-copy.dsconfig \
       --no-prompt
   ```
8. Click the link to the PingAuthorize server from the PingOne solutions home page. A PingOne login page should appear. After you provide credentials, you should see the administrative console index page.

Configure traffic through a load balancer

Use **`dsconfig`** or the administrative console to configure PingAuthorize Server to get traffic through a load balancer and to record the actual client's IP address.

To record the actual client's IP address to the trace log, enable `X-Forwarded-*` handling in both the intermediate HTTP server and the PingAuthorize Server.

By default, when a PingAuthorize Server is sitting behind an intermediate HTTP server, such as a load balancer, a reverse proxy, or a cache, it logs incoming requests as originating with the intermediate HTTP server instead of the client that sent the request.

When you set the `use-forwarded-headers` property and enable an HTTP connection handler to use `Forwarded` or `X-Forwarded-*` headers, many intermediate HTTP servers add information about the original request that would otherwise be lost.

If `use-forwarded-headers` is set to `true`, the server uses the client IP address and port information in the `Forwarded` or `X-Forwarded-*` headers instead of the address and port of the entity that's sending the request (the load balancer). This client address information shows up in logs, such as in the `from` field of the `HTTP REQUEST` and `HTTP RESPONSE` messages.

> **Note:**
>
> If both the `Forwarded` and `X-Forwarded-*` headers are included in the request, the `Forwarded` header takes precedence. The `X-Forwarded-Prefix` header only overrides the context path for HTTP servlet extensions, not for web application extensions.

Configuring traffic through a load balancer using dsconfig

Steps

1. Edit the HTTP or HTTPS connection handler object and set `use-forwarded-headers` to `true` by running **`dsconfig`**.

   Example:

   ```
   dsconfig set-connection-handler-prop \
       --handler-name "HTTPS Connection Handler" \
       --set use-forwarded-headers:true
   ```

2. To finalize the changes to the HTTP or HTTPS connection handler, use **`dsconfig`** to restart the connection handler.

   Example:

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

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

3. To provide the `X-Forwarded-*` information to your load balancer, consult your provider's guide on configuring load balancer settings.

Configuring traffic through a load balancer using the administrative console

Steps

1. On the PingAuthorize administrative console **Configuration** page, click **Connection Handlers**.
2. To edit your HTTP or HTTPS connection handler, in the **Connection Handlers** list, select the connection handler you want to edit.
3. To enable `Forwarded` headers, go to **Use Forwarded Headers** and select the **Enabled** check box.
4. Click **Save**.
5. To finalize the changes to the HTTP or HTTPS connection handler, use **`dsconfig`** to restart the connection handler.

   Example:

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

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

   Note:

   Because disabling the connection handler brings down the administrative console, you must complete this step in the command line instead of the administrative console.

6. To provide the `X-Forwarded-*` information to your load balancer, consult your provider's guide on configuring load balancer settings.

PingAuthorize Server configuration with dsconfig

These examples show how to configure PingAuthorize Server using **`dsconfig`**.

The examples cover the following topics.

- *Configuring the PingAuthorize user store* on page 359
- *Configuring the PingAuthorize OAuth subject search* on page 360
- *Configuring PingAuthorize logging* on page 360

Configuring the PingAuthorize user store

Configure PingAuthorize Server to use PingDirectory Server as its user store.

Steps

1. To make a set of changes to PingDirectory Server that PingAuthorize Server needs, including the creation of a service account, run the **`prepare-external-store`** command.

   Example:

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

2. To configure PingAuthorize Server with a store adapter that allows it to communicate with PingDirectory Server to retrieve identity attributes, run the **`create-initial-config`** command.

   > **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 *User profile availability in policies* on page 288.

   Example:

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

   This command also sets up a System for Cross-domain Identity Management (SCIM) resource type that defines a `Users` type with a SCIM schema that is automatically mapped to an LDAP type, `inetOrgPerson`, on PingDirectory Server.

Configuring the PingAuthorize OAuth subject search

Configure PingAuthorize Server to search the user store for OAuth token subjects.

Steps

- To configure the PingAuthorize Server to mock OAuth access token validation, run the **`dsconfig create-access-token-validator`** command.

  Example:

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

  The Mock Access Token Validator accepts tokens without authenticating them and is used only for demonstration and testing purposes. To use an authorization server like PingFederate, see *Access token validators* on page 290.
- To configure PingAuthorize Server to search the user store and retrieve the identity attributes of the OAuth token subject so the attributes can be evaluated in a policy, run the **`dsconfig create-token-resource-lookup-method`** command.

  Example:

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

  A token resource lookup method defines the expression that is used to search System for Cross-domain Identity Management (SCIM) resources by the access token subject or additional claims. In this example, the value of the access token subject claim is used to search the `uid` attribute value of the SCIM user resource.

Configuring PingAuthorize logging

Increase the default logging value to include details that will aid in debugging.

Steps

- To enable more detailed logging to understand how policy decisions are being made, including the comparison values and results of the various expressions that comprise a policy decision tree, run the **`dsconfig set-policy-decision-service-prop`** command.

  Example:

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

```
  --add decision-response-view:evaluated-entities
```

Warning:

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

Note:

Policy Decision views affect the decision response payload of the request. You can remove added views by using the `--remove decision-response-view:<view_name>` argument. See *About the Decision Response View* on page 399 for more information.

- To enable Trace (detailed) logging, including complete HTTP requests and responses, run the **`dsconfig set-log-publisher-prop`** command .

  Example:

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

Note:

Complete HTTP requests and responses might contain sensitive data.

For information about enabling detailed debug logging for troubleshooting purposes, see *Enable detailed logging* on page 397.

Deployment automation and server profiles

Administrators can export the configuration of a PingAuthorize Server instance to a directory of mostly text files, called a server profile. An administrator can then use that server profile to configure another deployment.

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 PingAuthorize 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 PingAuthorize 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 using manage-profile* on page 362.
- 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 using manage-profile

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 **Note:** Use **`PING_PROFILE_ROOT`** only with files that are not needed after initial setup, such as password files in `setup-arguments.txt`. Do not use the **`PING_PROFILE_ROOT`** variable for files needed

Built-in variable	Description
	while the server is running. The **manage-profile** tool creates a temporary copy of the server profile that is deleted after the tool completes, so files are not accessible under **PING_PROFILE_ROOT** when the server is running. For files you need while the server is running, such as keystore and truststore files, copy the files into the server root using the profile's `server-root/pre-setup` directory, and then refer to the files using with the **PING_SERVER_ROOT** variable.

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-paz.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 PingAuthorize 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`.

Note:

Use **`PING_PROFILE_ROOT`** only with files that are not needed after initial setup, such as password files in `setup-arguments.txt`. Do not use the **`PING_PROFILE_ROOT`** variable for files needed while the server is running. The **`manage-profile`** tool creates a temporary copy of the server profile that is deleted after the tool completes, so files are not accessible under **`PING_PROFILE_ROOT`** when the server is running.

For files you need while the server is running, such as keystore and truststore files, copy the files into the server root using the profile's `server-root/pre-setup` directory, and then refer to the files using with the **`PING_SERVER_ROOT`** variable.

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`.

About the manage-profile tool

The **`manage-profile`** tool is provided with the server to work with server profiles. It includes subcommands for creating, applying, and replacing server profiles, all of which significantly reduce the effort required by an administrator to configure a server appropriately.

The following sections describe these subcommands 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`**.

manage-profile generate-profile

To create a server profile from a configured server, use the **`generate-profile`** subcommand. The generated profile contains the following information, which provides a base for completing a profile:

- Command-line arguments that were used to set up the server
- **`dsconfig`** commands necessary to configure the server
- Installed server SDK extensions
- Files that are added to the server root

To produce a complete profile, some parts of the generated profile might require modifications, such as adding password files that `setup-arguments.txt` uses. The **`--instanceName`** and **`--localHostName`** arguments in `setup-arguments.txt` are made variables by **`generate-profile`**, and must be provided values when other **`manage-profile`** subcommands use the generated profile.

LDIF files must also be added manually to the generated profile.

The `--excludeSetupArguments` option generates a server profile without a setup-arguments.txt file. This is useful when generating server profiles for use with Docker images.

manage-profile setup

To apply a server profile to a fresh, unconfigured server, use the **`setup`** subcommand, which replaces the normal setup tool when using a server profile. Run **`manage-profile setup`** to complete the following tasks:

- Copies the pre-setup files to the server root
- Runs the setup tool
- Copies the post-setup files to the server root
- Installs any server SDK extensions
- Runs any dsconfig batch files
- Imports any LDIF files
- Starts the server

While manage-profile setup is running, a copy of the profile is created in a temporary directory that can be specified by using the **`--tempProfileDirectory`** argument. The command leaves the server in a complete and running state when finished, unless the **`--doNotStart`** argument is specified.

manage-profile replace-profile

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 tool applies a specified server profile to an existing server while preserving its data, topology configuration, and replication configuration. New LDIF files from the replacement server profile are not imported.

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

Run **manage-profile replace-profile** from a second unzipped server install package on the same host as the existing server, similar to the **update** tool. Use the **--serverRoot** argument to specify the root of the existing server that will have its profile replaced.

If files have been added or modified in the server root since the most recent **manage-profile setup** or **manage-profile replace-profile** was run, they are included in the final server with the replaced profile. Otherwise, files specifically added from the `server-root` directory of the previous server profile are absent from the final server with the replaced profile. If errors occur during the subcommand, such as the new profile having an invalid `setup-arguments.txt` file, the existing server returns to its original state from before **manage-profile replace-profile** was run.

The `--skipValidation` option skips the validation step when running on an offline server

> **Note:**
>
> The **manage-profile replace-profile** tool can update the server version when needed. This tool can also directly apply configuration changes when there are no other changes in the new profile. This is a shorter process when making small changes to **dsconfig**.

Common manage-profile workflows

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

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 366
- *Installing a new environment* on page 368
- *Scaling up your environment* on page 369
- *Rolling out an update* on page 369

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 PingAuthorize Server.

About this task

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

Steps

1. Create a profile directory.
   Example:

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

2. Run **`generate-profile`**.
   Example:

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

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

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

        ```
        $ cp PingAuthorize.lic /opt/server-profiles/pingauthorize/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.

     > **Note:**
     >
     > Use **`PING_PROFILE_ROOT`** only with files that are not needed after initial setup, such as password files in `setup-arguments.txt`. Do not use the **`PING_PROFILE_ROOT`** variable for files needed while the server is running. The **`manage-profile`** tool creates a temporary copy of the server profile that is deleted after the tool completes, so files are not accessible under **`PING_PROFILE_ROOT`** when the server is running. For files you need while the server is running, such as keystore and truststore files, copy the files into the server root using

the profile's `server-root/pre-setup` directory, and then refer to the files using with the **`PING_SERVER_ROOT`** variable.

```
${PING_PROFILE_ROOT}/misc-files/PingAuthorize.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/pingauthorize/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 PingAuthorize 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/pingauthorize`.
- Your development environment's variables file is saved at the path `~/pingauthorize-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.
   Example:

   ```
   $ mkdir /opt/pingauthorize
   $ cd /opt/pingauthorize
   $ unzip PingAuthorize-<version>.zip
   ```

2. Change directories.
   Example:

   ```
   $ cd PingAuthorize
   ```

3. Run **`setup`**.
   Example:

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

Scaling up your environment
You can scale up the environment in your PingAuthorize 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 PingAuthorize 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 PingAuthorize 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/pingauthorize`.
- The server's server profile variables file is saved at the path `/opt/pingauthorize/pingauthorize-variables.env`.
- The existing server with the earlier configuration is installed at `/opt/pingauthorize/PingAuthorize`.

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 PingAuthorize Server to a location outside the existing server's installation.

   Example:

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

2. Change directories.

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

   Example:

   ```
   $ cd PingAuthorize
   ```

3. Run **`replace-profile`**.

   Example:

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

Server status

You can check server status using the PingAuthorize 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 *PingAuthorize administrative console* on page 302.

`status` command

The PingAuthorize 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

PingAuthorize 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 PingAuthorize Server and set up load balancing or auto-healing for it.

Use the following gauges to monitor PingAuthorize Server availability:

- User Store Availability gauge
- Endpoint Average Response Time (Milliseconds) gauge
- HTTP Processing (Percent) gauge
- Policy Decision Service Availability 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 370. 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 PingAuthorize.

If PingAuthorize 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 PingAuthorize 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 370.

When PingAuthorize has a status of UNAVAILABLE, a load balancer can try to route traffic to a different PingAuthorize server or take some other action. See *Auto-healing for unavailable servers* on page 375.

If you followed the standard setup and configuration given in *Getting started with PingAuthorize (tutorials)* on page 17, the `User Store Availability` gauge should automatically work.

Important:

The gauge assumes the PingAuthorize LDAP Store Adapter name is UserStoreAdapter. If your PingAuthorize 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 PingAuthorize 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 PingAuthorize 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.

Value	Description
`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 370.

The following example shows how to activate the gauge.

> **Note:**
>
> You might need to experiment to find values that work for your environment.

Example

```
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 370.

The following example shows how to activate the gauge.

Note:

You might need to experiment to find values that work for your environment.

Example

```
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
```

Policy Decision Service Availability gauge

The `Policy Decision Service Availability` gauge monitors the ability of the Policy Decision Service to respond to requests using the configured policies.

If the Policy Decision Service is misconfigured, or the configured deployment package store is not reachable, PingAuthorize can't handle requests for the following services:

- API Security Gateway
- Sideband API
- SCIM 2
- Authorization Policy Decision APIs

In this case, this gauge marks the status of PingAuthorize as DEGRADED.

Possible causes of the Policy Decision Service being unavailable include:

- The `pdp-mode` is set to `disabled`
- The `trust-framework-version` is set to `undefined`
- The configured deployment package store can't be reached

Auto-healing for unavailable servers

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 371.
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 372 and *HTTP Processing (Percent) gauge* on page 373, 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 370.

Available gauges

PingAuthorize 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.

Gauge name	Enabled by default	Description
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.
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.

Gauge name	Enabled by default	Description
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 PingAuthorize 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 372.
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 373.
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%, open a case with *Ping Identity 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`.

Gauge name	Enabled by default	Description
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.
Policy Decision Service Availability	true	Monitors availability of the Policy Decision Service. If the Policy Decision Service is misconfigured or cannot reach the deployment package store, PingAuthorize services will be unavailable. Ensure that the `pdp-mode` and `trust-framework-version` are correctly set, and that the deployment package store is reachable. For more information, see *Policy Decision Service Availability gauge* on page 374.
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.

Gauge name	Enabled by default	Description
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 371.

Common server alarms

The server uses alarms and alerts to notify administrators of situations that might require intervention.

Policy Decision Service unavailable

PingAuthorize Server raises this alarm if it cannot process policy decisions because the Policy Decision Service requires further configuration. When this alarm is present, PingAuthorize Server cannot handle requests for the following services:

- API Security Gateway
- Sideband API
- SCIM 2
- Authorization Policy Decision APIs

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 and that configured deployment package stores are reachable.

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, PingAuthorize 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 136.
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, PingAuthorize 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 PingAuthorize 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 PingAuthorize 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
```

Managing monitoring

PingAuthorize provides several monitoring options.

The following sections describe the options.

- *Profiling server performance using the Stats Logger* on page 381
- *Logging HTTP performance statistics using the Periodic Stats Logger* on page 383
- *StatsD monitoring endpoint* on page 383
- *Sending metrics to Splunk* on page 384

Profiling server performance using the Stats Logger

PingAuthorize provides a Stats Logger plugin you can use to profile server performance for a given configuration.

At a specified interval, the Stats Logger can write server statistics to a JSON file or to a log file in a comma-separated value (`.csv`) format.

The logger has a negligible impact on server performance unless the `log-interval` property is set to a very small value (less than 1 second). You can customize the statistics logged and their verbosity.

You can also use the Stats Logger to view historical information about server statistics including LDAP operations, host information, and gauges. Your options include:

- Update the configuration of the existing Stats Logger Plugin to set the advanced `gauge-info` property to `basic/extended` to include this information.
- Create a dedicated Periodic Stats Logger for information about statistics of interest.

Enabling the Stats Logger

By default, the Stats Logger plugin is disabled. Enable it using the **`dsconfig`** tool (and its Advanced Objects menu and Plugin option) or the administrative console (and its Advanced Configuration menu and Plugin Root option).

About this task

The steps below show how to use **`dsconfig`** to enable the plugin.

Steps

1. Run **dsconfig** in interactive mode. Enter the LDAP or LDAPS connection parameters when prompted.

   Example:

   ```
   $ bin/dsconfig
   ```
2. Enter `o` to change to the Advanced Objects menu.
3. On the main menu, enter the number for the Plugin menu.
4. On the Plugin menu, enter the number corresponding to view and edit an existing plugin.
5. On the Plugin selection list, enter the number corresponding to the Stats Logger.
6. On the Stats Logger Plugin menu, enter the number to set the `enabled` property to `TRUE`.

   If the server is idle, nothing is logged. You can log data even when idle by setting the `suppress-if-idle` property to `FALSE` (`suppress-if-idle=false`).

   > **Note:**
   > On this menu, you can also change the format from `csv` to `json`.
7. When done changing properties, enter `f` to save and apply the configuration.

   The default logger logs information about the server every second to *<server-root>*`/logs/dsstats.csv`. You can open the file in a spreadsheet.

Configuring multiple Periodic Stats Loggers

Create multiple, Periodic Stats Loggers to log different statistics or to view historical information about gauges. Also, you might create multiple loggers to create a log at different intervals (such as logging cumulative operations statistics every hour). To create a new log, use the existing Stats Logger as a template to get reasonable settings, including rotation and retention policy.

Steps

1. Run **dsconfig** in interactive mode. Enter the LDAP or LDAPS connection parameters when prompted.

   Example:

   ```
   $ bin/dsconfig
   ```
2. Enter `o` to change to the Advanced Objects menu.
3. On the main menu, enter the number for the Plugin menu.
4. From the Plugin management menu, enter the number to create a new plugin.
5. Enter `t` to use an existing plugin as a template.
6. Enter the number corresponding to the existing stats logger as a template.
7. Enter a descriptive name for the new stats logger.
8. Enter the log file path to the file.

   For example, type `logs/dsstats2.csv`.
9. On the menu, make any desired changes to the properties for the logger.

   For information about the included-http-servlet-stat property, see *Logging HTTP performance statistics using the Periodic Stats Logger* on page 383.

   > **Note:**
   > On this menu, you can also change the format from `csv` to `json`.
10. Enter `f` to save and apply the configuration.

Logging HTTP performance statistics using the Periodic Stats Logger

To log HTTP performance statistics, set the Periodic Stats Logger property `included-http-servlet-stat`.

About this task

You can log HTTP performance statistics for any combination of the following servlet extensions:

- **gateway**
- **scim2**
- **sideband-api**

The provided statistics come in pairs:

- One statistic represents the average latency introduced by PingAuthorize during the current log interval in microseconds. The calculation is total time to respond to a request less the time spent waiting for the upstream server.
- The other statistic represents the number of requests made during the current log interval.

These throughput and latency pairs exist for every service, action combination for the **scim2** and **sideband-api** servlet extensions and for every service, HTTP method combination for the **gateway** servlet extension.

To log these statistics:

Steps

1. Enable the Periodic Stats Logger.

   For more information, see *Enabling the Stats Logger* on page 381.
2. Set the `included-http-servlet-stat` property.

   For more information, see *Configuring multiple Periodic Stats Loggers* on page 382.

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 not using Data Metrics Server to monitor your server, you can disable the generation of some metrics files that are not necessary for the StatsD Monitoring Endpoint. To do this, set the `generate-collector-files` property on the Stats Collector Plugin to `false`.

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 384.
- 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 PingAuthorize 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.

PingAuthorize, 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 PingAuthorize handles correlation IDs

- When any HTTP request is received, PingAuthorize automatically assigns the request a correlation ID.
- All related activity appears in the trace logs with this correlation ID.
- The PingAuthorize 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 PingDirectoryProxy 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.

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");
```

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 PingAuthorize Server sends with HTTP requests.

About this task

By default, PingAuthorize Server generates a correlation ID for every HTTP request and response header.

To customize this response header name:

Steps

- By default, PingAuthorize Server generates a correlation ID for every HTTP request and sends it in the response with the **`dsconfig`** command.

  Example: 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.

Example: HTTP correlation ID
This example shows a SCIM 2 request with a correlation ID assigned in the response. Then the example uses that ID to locate entries in the debug trace log and the policy decision log.

First, make a SCIM 2 GET request.

The response includes a `Correlation-Id` header with the value `c52af735-788d-4798-be3b-8d1f3c8f9d64`. The ellipsis (`...`) in the response indicates lines removed to keep the example brief. Because the request does not include a correlation ID, the server generates the header and value.

```
GET https://localhost:8443/scim/v2/Me 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: 903
Content-Type: application/scim+json
Correlation-Id: c52af735-788d-4798-be3b-8d1f3c8f9d64
Date: Mon, 15 Mar 2021 15:23:06 GMT
Request-Id: 371

{
    "mail": [
        "user.0@example.com"
    ],
    "initials": [
        "AOR"
    ],
    "homePhone": [
        "+1 295 940 2750"
    ],
    "pager": [
        "+1 604 109 3407"
    ],
    "givenName": [
        "Anett"
    ],
   ...
}
```

Use the correlation ID to search the HTTP debug trace log for matching log records.

```
$ grep 'correlationID="c52af735-788d-4798-be3b-8d1f3c8f9d64"' PingAuthorize/
logs/debug-trace
```

Also, use the correlation ID to search the policy decision log for matching log records.

```
$ grep 'correlationID="c52af735-788d-4798-be3b-8d1f3c8f9d64"' PingAuthorize/
logs/policy-decision
```

Command-line tools

PingAuthorize Server provides a full suite of command-line tools to administer the server. You can run these tools in interactive, noninteractive, or script mode.

Note:

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.

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 *PingAuthorize Command-Line Tool Reference* at `docs/cli/index.html`.

Command-line tools

Tool	Description
`backup`	Run full or incremental backups on one or more PingAuthorize Server backends. This tools supports the use of a properties file to pass command-line arguments. See *Saving command options in a file* on page 393.
`base64`	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
`collect-support-data`	Collect and package system information useful in troubleshooting problems. The information is packaged as a zip archive that you can send to a technical support representative.
`config-diff`	Compares PingAuthorize Server configurations and produces a **`dsconfig`** batch file needed to bring the source inline with the target.
`create-initial-config`	Create an initial PingAuthorize Server configuration.

Tool	Description
create-rc-script	Create a Run Control (RC) script to start, stop, and restart the PingAuthorize Server on UNIX-based systems.
create-systemd-script	Create a **systemd** script to start and stop the PingAuthorize Server on Linux-based systems.
docker-pre-start-config	Run this tool before starting PingAuthorize Server to make configuration changes to the server that depend on the running container's environment.
dsconfig	View and edit the PingAuthorize Server configuration.
dsjavaproperties	Configure the JVM options used to run PingAuthorize 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 compare operations in an LDAP directory server. Compare operations can be used to efficiently determine whether a specified entry has a given value.
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	Update the password for a user in an LDAP directory server using the password modify extended operation (as defined in RFC 3062), a standard LDAP modify operation, or an Active Directory-specific modification.
ldapsearch	Process one or more searches in an LDAP directory server.
ldif-diff	Compare the contents of two files containing LDIF entries. The output will be an LDIF file containing the add, delete, and modify change records needed to convert the data in the source LDIF file into the data in the target LDIF file.
ldifmodify	Apply a set of changes (including add, delete, modify, and modify DN operations) to a set of entries contained in an LDIF file. The changes will be read from a second file (containing change records rather than entries), and the updated entries will be written to a third LDIF file. Unlike **ldapmodify**, **ldifmodify** cannot read the changes to apply from standard input.
ldifsearch	Search one or more LDIF files to identify entries matching a given set of criteria.
list-backends	List the backends and base DNs configured in PingAuthorize Server.
manage-certificates	Manage certificates and private keys in a JKS, PKCS #12, PKCS #11, or BCFKS key store.
manage-extension	Install or update PingAuthorize 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 PingAuthorize Server.
manage-topology	Tool to manage the topology registry.

Tool	Description
prepare-external-store	Prepare a PingAuthorize 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 PingAuthorize Server backend.
remove-defunct-server	Remove a server from this server's topology.
replace-certificate	Replace the listener certificate for this PingAuthorize Server server instance.
restore	Restore a backup of a PingAuthorize 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.
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. **Note:** To sanitize error log content as it's being written, see *Log Sanitization*.

Tool	Description
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 multi-line 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 PingAuthorize Server process.
setup	Perform the initial setup for a server instance.
start-server	Start the PingAuthorize 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 PingAuthorize Server.
update	Update a deployed server so its version matches the version of this package.
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 command options in a file

PingAuthorize 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 PingAuthorize Server in multiple environments.

Each property takes the form of a name-value pair that defines predetermined values for a tool's options.

PingAuthorize 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 priority of command-line options
You can specify options for a command-line tool on the command line, in a properties file, or both.

Options you specify on a tool's command line take priority over options in a properties file.

Consider the following scenarios.

Command-line options	PingAuthorize Server uses ...
No command-line options	The options in the default *<server-root>*/config/tools.properties file
Command-line options other than the --propertiesFilePath *<my-properties-file>* option	The command-line options, which take priority if the options are also in the *<server-root>*/config/tools.properties file The file options for options that are only in the default *<server-root>*/config/tools.properties file
Only the --propertiesFilePath *<my-properties-file>* option	The options in *<my-properties-file>*
The --propertiesFilePath *<my-properties-file>* option and other command-line options	The command-line options, which take priority if the options are also in *<my-properties-file>* The file options for options that are only in *<my-properties-file>*
The --noPropertiesFile option and other command-line options	Only the options you specify on the command line, ignoring the default properties file

Example

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
```

PingAuthorize Server checks command-line options and file options to determine the options to use, as explained below.

- All options presented with the tool on the command line take precedence over any options in a properties file.

  In the following example, the command runs with the options specified on the command line (--port and --baseDN). With the port value both on the command line and in the properties file, the command-line value takes priority. 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=*)"
  ```

- If you specify the properties file using the --propertiesFilePath option and no other command-line options, PingAuthorize Server uses only the options in the specified properties file:

  ```
  $ bin/ldapsearch --propertiesFilePath bin/tools.properties \
    "(objectclass=*)"
  ```

- If do not specify any command-line options, PingAuthorize Server attempts to locate the default properties file in the following location:

  ```
  <server-root>/config/tools.properties
  ```

  By moving your `tools.properties` file from *<server-root>*`/bin` to *<server-root>*`/config`, you do not have to specify the `--propertiesFilePath` option. That change shortens the previous command to the following command.

  ```
  $ bin/ldapsearch "(objectclass=*)"
  ```

Sample dsconfig batch files

PingAuthorize provides sample **`dsconfig`** batch files that you can use to easily make a number of common or recommended changes to the server configuration.

The `config/sample-dsconfig-batch-files` directory contains **`dsconfig`** batch files that you can use to configure various aspects of the server. For example, these files can enable additional security capabilities or take advantage of features that might require customization from one environment to another.

Each file includes comments that describe the purpose and benefit of its configuration change. You can choose which of the changes you want to apply.

You need to customize some of the batch files to provide values that might vary from one environment to another. To apply a batch file that requires changes, copy it to another directory and edit the copy. Leave the files in the `config/sample-dsconfig-batch-files` directory unchanged so that they can be updated when you upgrade the server. To specify the path to the file that contains the changes to apply, use the **`dsconfig`** tool (**`bin/dsconfig`** on UNIX-based systems or **`bat\dsconfig.bat`** on Windows) with the `--batch-file` argument.

You should also provide the arguments needed to connect and authenticate to the server. The `--no-prompt` argument ensures that the tool does not block while waiting for input if any necessary arguments are missing. Consider this example.

```
bin/dsconfig --hostname localhost \
  --port 636 --useSSL --trustStorePath config/truststore \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPasswordFile admin-password.txt \
  --batch-file config/hardening-dsconfig-batch-files/reject-insecure-request.dsconfig \
  --no-prompt
```

Running task-based tools

PingAuthorize 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.

Option	Description
`--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.
`--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.

Diagnostic and decision data

For problems with PingAuthorize 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 397
- *Enable detailed logging* on page 397
- *About the Decision Response View* on page 399
- *Visualizing a policy decision response* on page 400
- *Capture debugging data with the collect-support-data tool* on page 402

Exporting policy data

Export all Trust Framework and policy data from the PingAuthorize Policy Editor to a snapshot that captures all of the policy data contained within a branch of the PingAuthorize Policy Editor.

About this task

Snapshots provide a convenient way to load policy data into a separate PingAuthorize Policy Editor 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**.
   Result: 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 PingAuthorize Server is configured to evaluate a policy in embedded or external mode, a `policy-decision` file logs every policy decision per request. The file is located at `PingAuthorize/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.

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.

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
```

Note:

Policy Decision views also affect the decision response payload of the request. You can remove added views by using the `--remove decision-response-view:<view_name>` argument. See *About the Decision Response View* on page 399 for more information.

Configurable attribute logging for embedded mode

When running the Policy Decision Service in embedded mode, you can exercise some control over which attributes get logged as part of the policy-decision response. The **`dsconfig set-policy-decision-service-prop`** command supports an attribute-logging argument. This argument allows you to log the full details of the specified attributes when they're evaluated as part of the policy-decision request.

Here's an example of how to use the attribute-logging argument for embedded mode:

```
dsconfig set-policy-decision-service-prop \
  --set embedded-mode-logged-attributes:<attribute1> \
  --set embedded-mode-logged-attributes:<attribute2>
```

Warning:

Attributes specified using this argument are only logged if they get evaluated as part of the of policy-decision request. Enabling certain decision response views could override this configuration and cause all evaluated attributes to be included in the response.

Including additional attributes could cause the Trace Log Publisher or the Policy Decision Log Publisher to record sensitive data.

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 `PingAuthorize/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 `PingAuthorize/logs/debug`.

About the Decision Response View

You can use the **Decision Response View** to increase or decrease the size of the policy decision response from the Policy Decision Point (PDP).

When a client application makes a request for API resources, the PingAuthorize server returns a decision response payload that includes, at minimum, basic information about the server instance, the API resources, and the inbound and outbound flow of data. The payload also includes any views selected in the **Decision Response View**. By default, no views are selected. PingAuthorize then passes the full response payload to the ***Policy Decision Logger***.

To configure the selected views for the **Decision Response View**, do one of the following:

- In the administrative console, go to **Configuration** # **Policy Decision Service** and change the **Selected** views included for **Decision Response View**.
- *Use CLI commands* to add or remove views.

You can change the verbosity of the response payload and the size of the `policy-decision` log files by changing the selected views in the Decision Response View by either:

- Adding views increases the size of response payloads and `policy-decision` log files.
- Removing views decreases the size of response payloads and `policy-decision` log files.

Note:

- Some views are more verbose than others.
- If you remove all views, the **Policy Decision Logger** still logs an abbreviated response. To prevent this abbreviated logging, disable **`include-pdp-response`** for the File Based Policy Decision Log Publisher.
- The Decision Response View behavior doesn't significantly change between embedded and external PDP modes.

You can select the following additional views in the **Decision Response View**.

Decision Response View	Description
attributes	Full details of attributes evaluated during policy decision evaluation.
decision-tree	Detailed output tracing the decision's policy evaluation flow.
evaluated-entities	Attribute and service resolution details. This is equivalent to specifying both **attributes** and **services**.
evaluation-log	Attribute and service resolution details. This is similar to specifying **evaluated-entities**, but the data are expressed in a flat format.
evaluation-log-with-attribute-values	Attribute and service resolution details. This is equivalent to specifying **evaluation-log** but also includes values and types for successful attribute resolutions.
request	The policy decision request. Might include sensitive data.
services	Full details of services invoked during policy decision evaluation.

Warning:

Selecting the **request** view causes the **Policy Decision Logger** to record potentially sensitive data in API requests and responses.

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 PingAuthorize Policy Editor.

2. Choose a method for visualizing a decision.
   Choose from:

   - Select a recent decision

     a. In the Policy Editor, go to **Policies**.
     b. Click the **Decision Visualiser** tab.
     c. Click **Recent Decisions** and select a decision.
     d. Click **Visualise**.

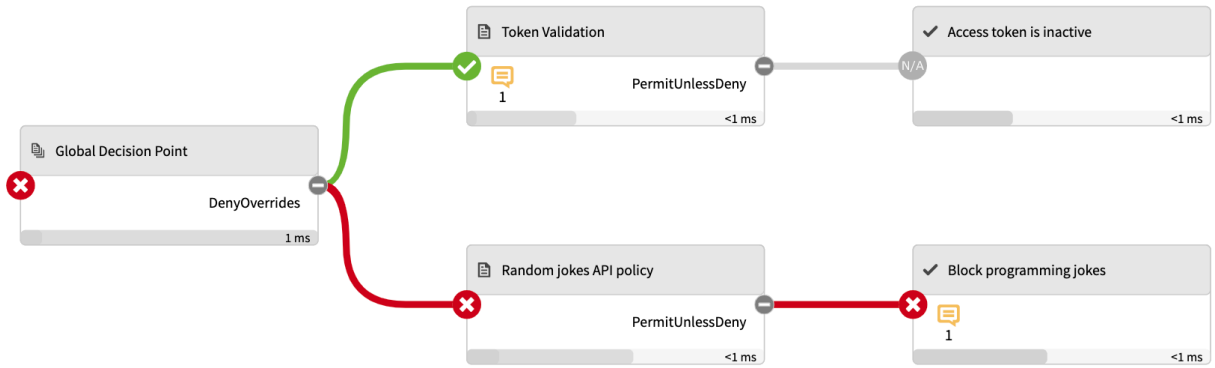
     > **Note:**
     >
     > You can control the number of recent decisions that appear in the list as explained in *Setting the request list length for Decision Visualizer* on page 256.

   - 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 PingAuthorize logging* on page 360.
     >
     > 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 Editor, 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**.

Result
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 PingAuthorize Server's configuration, server state, environment, and other information to use for troubleshooting issues.

When you run `PingAuthorize/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 397. When you use test data, send the following log files alongside `collect-support-data`'s compressed output file:

- `PingAuthorize/logs/policy-decision`
- `PingAuthorize/logs/debug-trace`
- `PingAuthorize/logs/debug`

About the layout of the PingAuthorize Server folders

The following table describes the contents of the PingAuthorize Server distribution file. In addition, the table describes items created as you use PingAuthorize Server.

PingAuthorize Server directories, files, and tools

Directories, files, and tools	Description
`README`	README file that describes the steps to set up and start PingAuthorize Server.
`bak`	Stores the physical backup files used with the backup command-line tool.
`bat`	Stores Windows-based command-line tools for PingAuthorize Server.
`bin`	Stores UNIX/Linux-based command-line tools for PingAuthorize Server.
`build-info.txt`	Contains build and version information for PingAuthorize 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.

Directories, files, and tools	Description
`legal`	Stores any legal notices for dependent software used with PingAuthorize Server.
`lib`	Stores any scripts, jar, and library files needed for the server and its extensions.
`locks`	Stores any lock files in the backends.
`logs`	Stores log files for PingAuthorize 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 PingAuthorize Policy Editor folders

The following table describes the contents of the PingAuthorize Policy Editor distribution file.

PingAuthorize Policy Editor 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 PingAuthorize Policy Editor.
`build-info.txt`	Contains build and version information for the PingAuthorize Policy Editor.

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 PingAuthorize Policy Editor.
`resource`	Stores supporting files such as policy snapshots.

PingAuthorize Policy Administration Guide

PingAuthorize Policy Editor includes policy development and testing capabilities:

- Policy administration and delegation
- Attribute resolution and orchestration

Getting started

This guide introduces the dynamic authorization features of the PingAuthorize Policy Editor. It shows you how to create attribute-based access control policies that reflect your business requirements. It also provides a tour of the various concepts involved in modeling policies in the Policy Editor.

About this task

To get started with the Policy Editor, complete the following tasks:

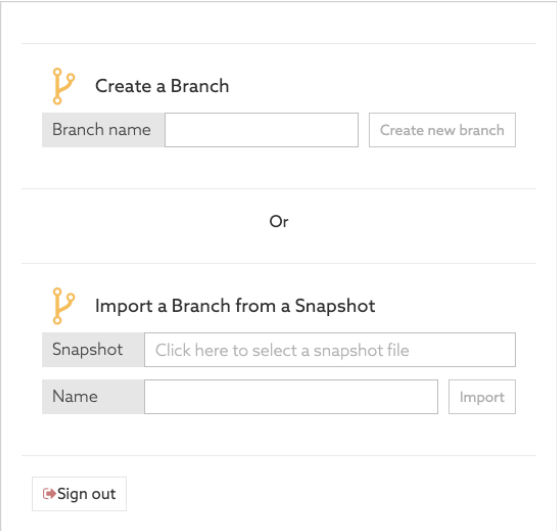
Steps

1. Sign on to the Policy Editor.
   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 PingAuthorize Server instances across multiple environments.

Next steps

After you sign on to the Policy Editor, 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 PingAuthorize Policy Editor embraces similar principles to general software source control. As such, it begins with the creation of a branch. When you first deploy the Policy Editor, 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 fine-grained authorization policy branches, commits, snapshots, and deployment packages.

Creating a new top-level branch

The PingAuthorize Policy Editor 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 Editor can share the same name.

Steps

1. Sign on to the Policy Editor.
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:

To create a new top-level branch from	Do this
	a. From the **+** menu, select **Create new root branch**. 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 408.

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**.

Result

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 409.

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.

> **Important:**
>
> If two branches each contain a Trust Framework or Test Suite definition that was created natively in that branch but has the same exact name and hierarchy in the other branch, the merge operation will not complete successfully.
>
> This happens because the duplicated items don't come from the same source, making the branches ineligible for merge conflict resolution. Rename any such duplicated items in one branch before attempting to merge.

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 PingAuthorize Policy Editor 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 both the incoming and existing branches, the Policy Editor flags a conflict. You must resolve the conflict for the merge to continue. Conflicts appear in the Merge Conflicts table.

   a. If you need all or almost all of the sections from one branch, click either the **Take All Incoming** button or the **Keep All Existing** button.
   b. To examine conflicts one at a time, click **Resolve Individual Conflicts**.

      On the resulting screen, select the **Show diff** check box to highlight differences.

      Decide which change to keep and click either **Keep Existing** or **Take Incoming**.
   c. After you resolve all conflicts, close the entity difference box.

      The **Apply Merge** button becomes available.
   d. Click **Apply Merge**.

Reverting branch 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 410.

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 snapshot 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

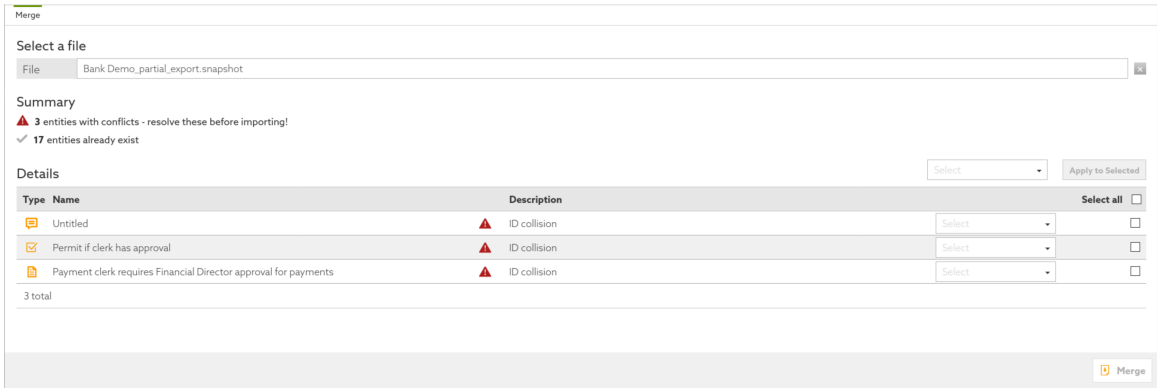
1. Click **Branch Manager**.
2. Click **Merge Snapshot**.
3. Select the appropriate snapshot file from your system.
4. Click **Merge**.

Result

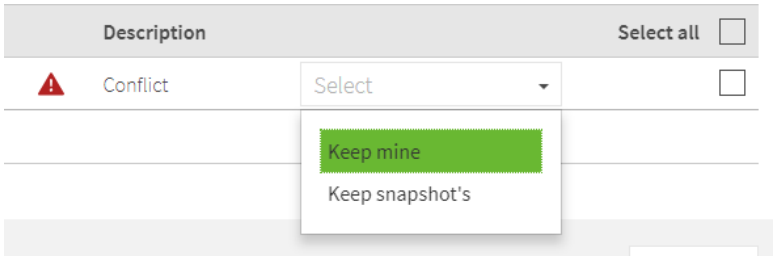
The system displays a **Summary** page that details the result of the merge.

Next steps

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



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



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

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 PingAuthorize 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.

When defining and using these items, you can identify all the places they are used as described in *Viewing Trust Framework entity dependencies* on page 432.

Domains (Authorization Policy Decision APIs 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 servers with Authorization Policy Decision APIs enabled. 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 *Value processing* on page 427.

Common settings
The settings in this section apply to all service types.

Request Timeout

The number of milliseconds that PingAuthorize 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 PingAuthorize Server attempts the request again. To try the request only once, set this value to zero.

> **Note:**
>
> If the service responds with a 4xx error, PingAuthorize Server won't make any retry attempts.

Retry Strategy

Options are:

Fixed Interval (default)

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

Exponential Backoff

: PingAuthorize 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 PingAuthorize Server waits between request attempts.

For Exponential Backoff, PingAuthorize 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, PingAuthorize 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. See *Value processing* on page 427.

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 425.

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 238.

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 273.

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 425.

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 414.

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

You can retrieve information from any endpoint that the *Apache Camel* enterprise integration platform supports. See the *list of Camel components* for a full list of supported systems.

Warning:

Using Camel to connect policy information points (PIPs) to PingAuthorize has been deprecated and is no longer supported. You should use *HTTP services* on page 413 instead, where applicable.

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 425.

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 427, 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 **+**.
4. Select **Add new Attribute**.
5. Update the attribute to include resolvers, value processing, and other changes, as discussed in the subsections after this one.
6. Click **Save changes**.

   After you create an attribute, you can modify it to be a repeating attribute. For more information, see *Repeating policies and attributes* on page 446.

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.

You can reorder collapsed resolvers by dragging the handles on the left. To reorder using the keyboard, press Tab to go to the resolver, press Enter to select the resolver, press the Up Arrow or Down Arrow to go to the desired location, press Enter to drop the resolver in the new location.

For more information, see:

- *Resolver types* on page 418
- *Conditional resolvers* on page 419
- *Value processing for a resolver* on page 420

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.

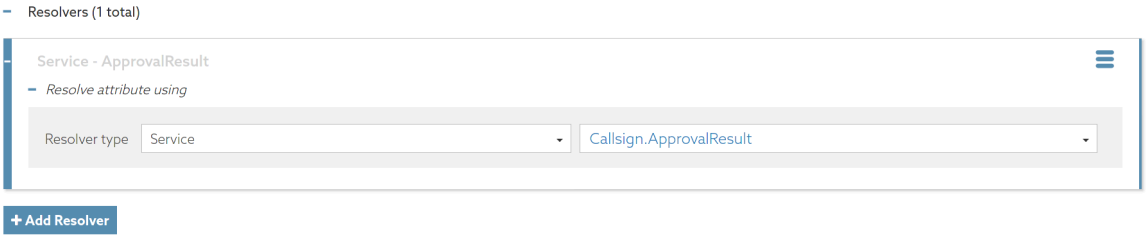
The following table describe the various resolver types.

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 425.
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.

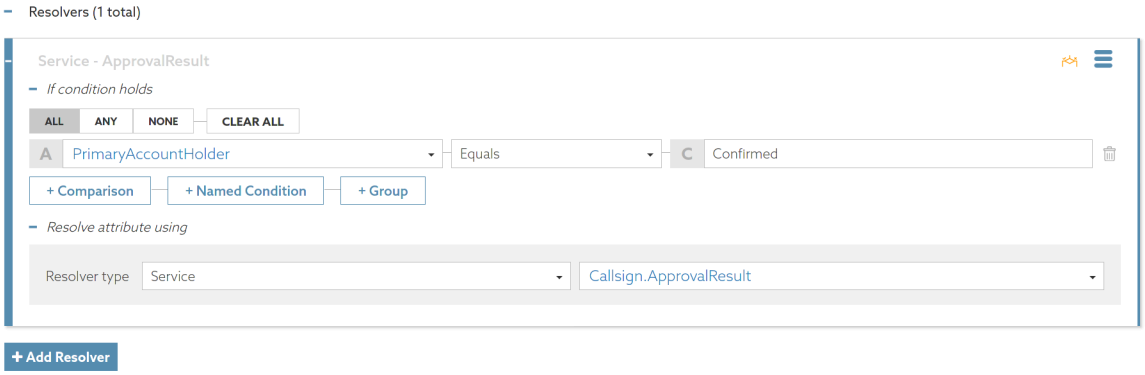
Resolver type	Description
Attribute	PingAuthorize Server can also resolve attributes from other attributes. This ability is useful when you have attributes that contain multiple pieces of information and you want to create nested or child attributes as subset extracts from them. For example, the **`Customer.Name`** 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 PingAuthorize Policy Editor 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 238. 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 273.

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.



In the following example, the service resolver `Callsign.ApprovalResult` applies only when the attribute **`PrimaryAccountHolder`** has a value of **`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 427.

The following examples show how you might use these resolvers.

Example

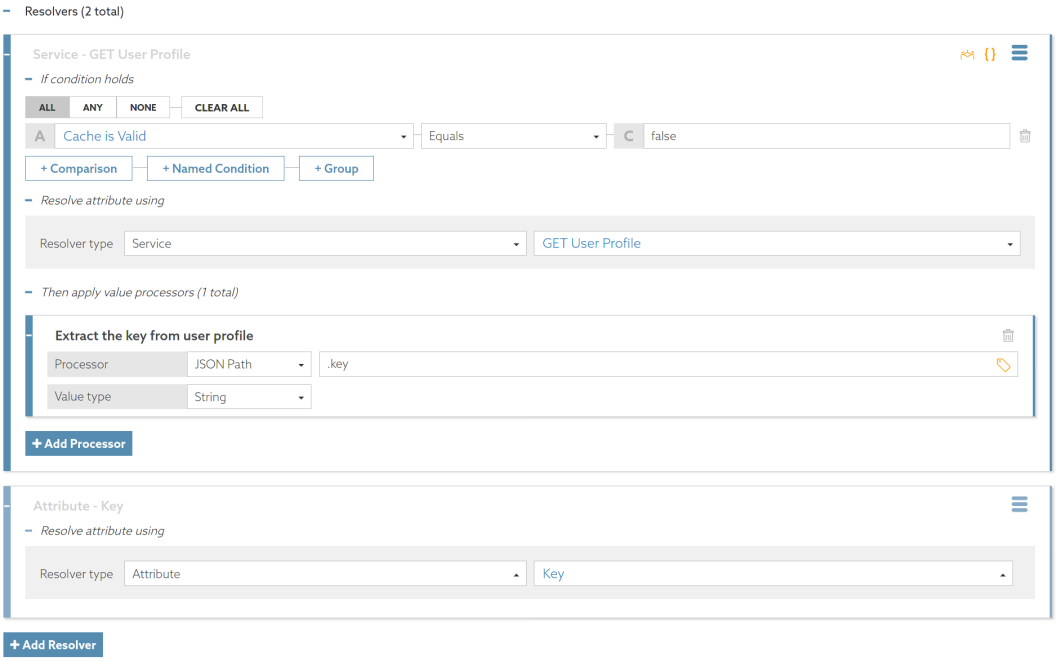
If you expect responses from different resolved sources to vary, you can add a processor to the resolvers to normalize the output. In this example, the attribute's value can come from one of the following resolvers:

- 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.

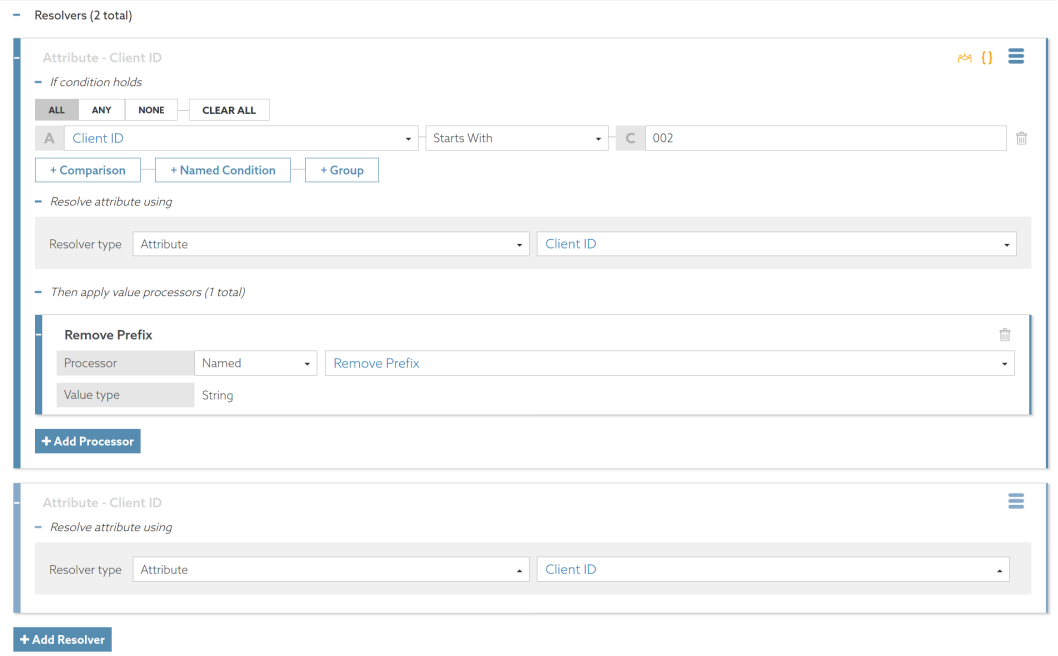


Example

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.



Attribute caching

The policy decision point (PDP) and the PingAuthorize Policy Editor support caching for attributes. The ability to cache resolved attributes can deliver significant performance gains for the PDP.

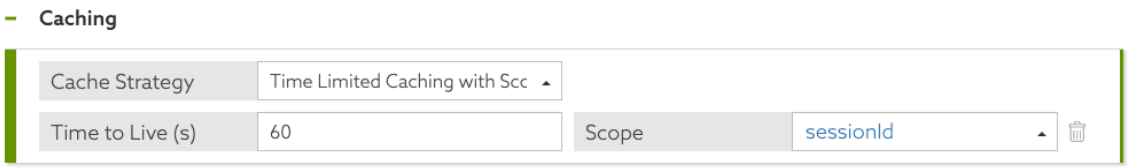
Carefully consider this concept to ensure optimum configuration.

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.



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 413 in *Services* on page 411.

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 422.

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 425. 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.

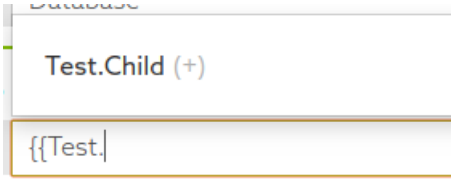
Attribute interpolation
With attribute interpolation, you reference an attribute in a field. The system resolves the value of the referenced attribute, replacing the reference with the value itself.

About this task

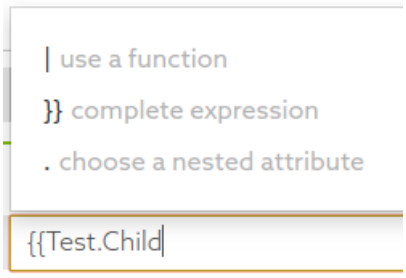
You can use attribute interpolation in any field that has the label icon, shown below.

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.



2. Complete the reference by typing two close curly brackets (`}}`) or by selecting the **}} complete expression** item from the attribute tree menu.



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 PingAuthorize Policy Editor 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 PingAuthorize Policy Editor 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**.

Value processing

Use value processing on responses returned from attributes or services to transform the resolved value.

Add a value processor when you create or edit an attribute or service. Alternatively, you can define a value processor to reference by name by going to **Trust Framework** # **Processors**.

The PingAuthorize Policy Editor supports these value processors:

- Collection filter
- Collection transform
- JSON Path
- X Path
- Spring Expression Language (SpEL)
- Named

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 reorder collapsed value processors by dragging the handles on the left. To reorder using the keyboard, press Tab to go to the processor, press Enter to select the processor, press the Up Arrow or Down Arrow to go to the desired location, press Enter to drop the processor in the new location.

Collection filter

When the data being processed is a collection, you can set a filter to examine each item in the collection and keep only the items that satisfy some condition. A collection filter uses a value processor to yield a true

or false for each item in the collection. When true, the original item goes in the resulting collection; when false, it is omitted.

Each item in the collection can optionally be preprocessed by one or more value processors before applying the condition. For example, suppose we received a JSON collection from a service invocation and we want to filter the items by the `score` field. The input data might look like the following lines.

```
[
    { "name": "Alice", "role": "Sender",   "score": 72 },
    { "name": "Bob",   "role": "Receiver", "score": 36 },
    { "name": "Carol", "role": "Observer", "score": 47 },
    { "name": "Dave",  "role": "Attacker", "score": 99 }
]
```

A collection filter processor could achieve this by using a JSON Path preprocessor to extract the `score`.

```
$.score
```

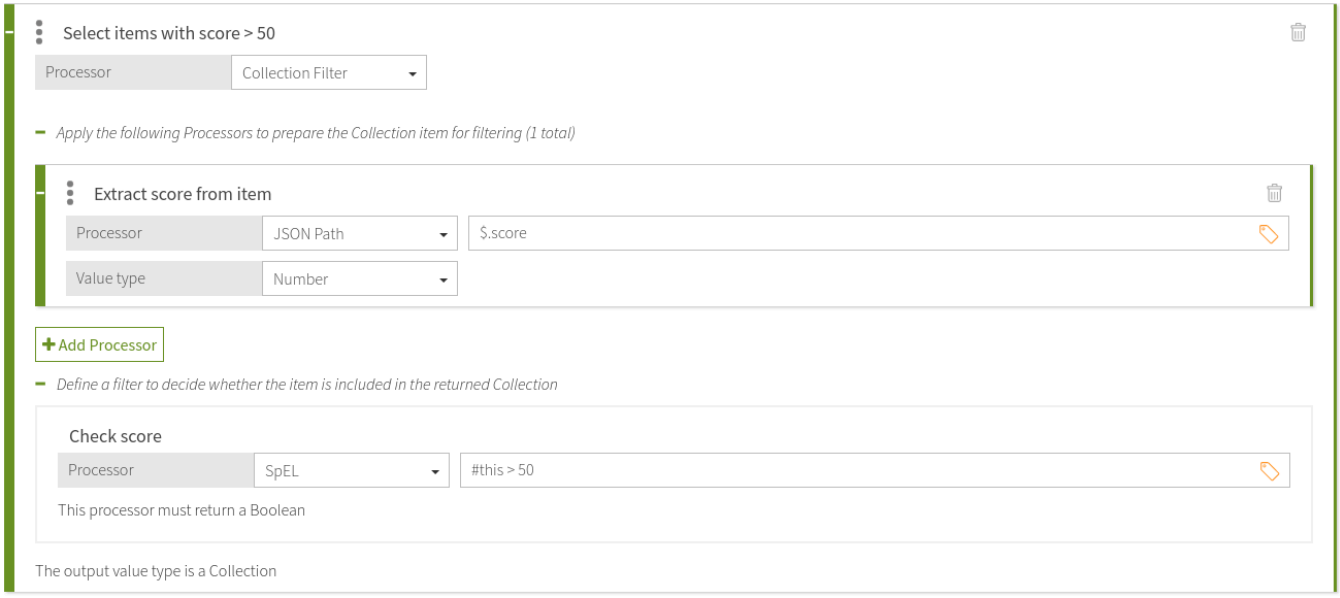
The following SpEL condition yields a true or false decision for each item.

```
#this > 50
```

Each list item is in turn passed through the preprocessing and the condition. The first item has score 72, which is greater than 50 so the condition yields true and the item is retained for the result collection. The second and third items have scores less than 50, so the condition yields false and these items are omitted. The final item also has a score higher than 50 and is retained. The result of the collection filter is:

```
[
    { "name": "Alice", "role": "Sender",   "score": 72 },
    { "name": "Dave",  "role": "Attacker", "score": 99 }
]
```

The values produced by the preprocessing and condition are only used to determine inclusion. The final result of a collection filter consists of those original collection items that satisfied the predicate after preprocessing. Here is the collection filter in the GUI.



If the condition or preprocessing produces an error for any item in the input collection (for example, if a `score` field is missing or not a number in the source data), the whole collection filter is considered to have failed.

Collection transform

When the data being processed is a collection, you can set a transform to apply a processor or a sequence of processors to each item in the collection.

Assume we have the following input collection.

```
[
    { "name": "Alice", "role": "Sender",   "score": 72 },
    { "name": "Bob",   "role": "Receiver", "score": 36 },
    { "name": "Carol", "role": "Observer", "score": 47 },
    { "name": "Dave",  "role": "Attacker", "score": 99 }
]
```

The following JSON Path processor extracts the `name` field for each item.

```
$.name
```

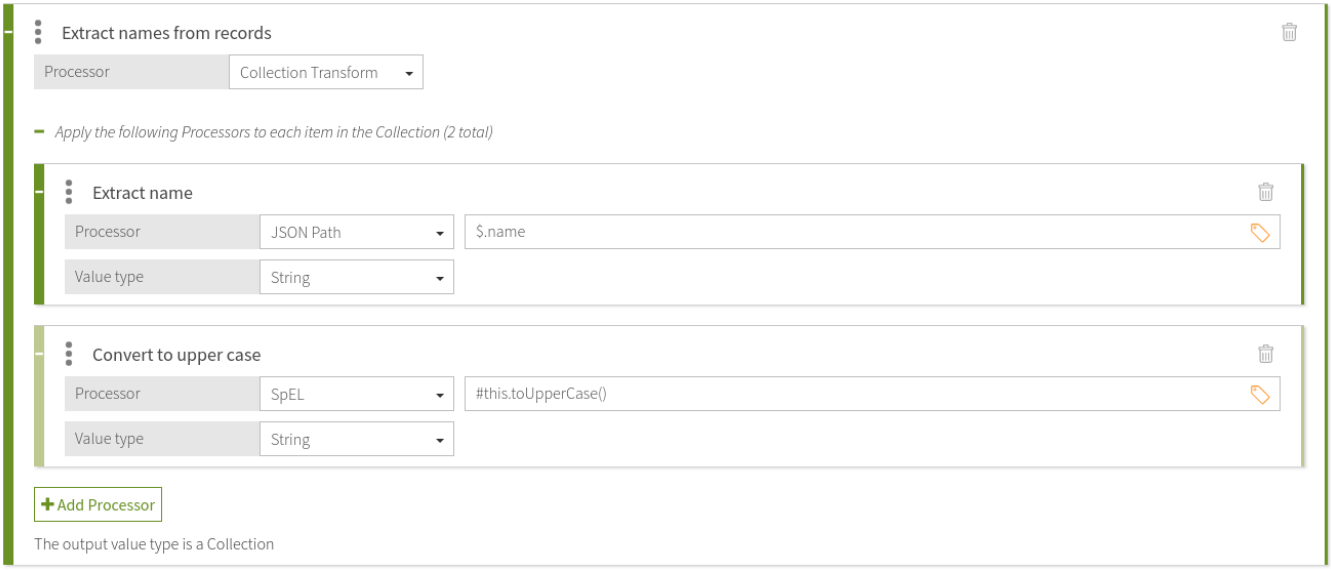
This SpEL processor converts each `name` to upper case.

```
#this.toUpperCase()
```

Then the resulting collection consists of just the extracted names converted to upper case, preserving the order of the original collection.

```
[ "ALICE", "BOB", "CAROL", "DAVE" ]
```

Here is the collection transform in the GUI.



If the item processor produces an error for any item, the overall collection transform processor produces an error.

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://github.com/json-path/JsonPath*.

X Path

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 Editor only supports the use of *XPath 1.0*. Functions added in later versions are not available.

SpEL (Spring Expression Language)

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 425):

```
{{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 255.

Named

Use named value processors to create reusable value processing logic.

Extracting this logic into reusable components helps abstract some of the complexity when you define an attribute or a service. Also, it reduces repetition.

You can still create inline value processors that co-exist with named value processors.

To define a named value processor that you can reference, go to **Trust Framework** # **Processors**.

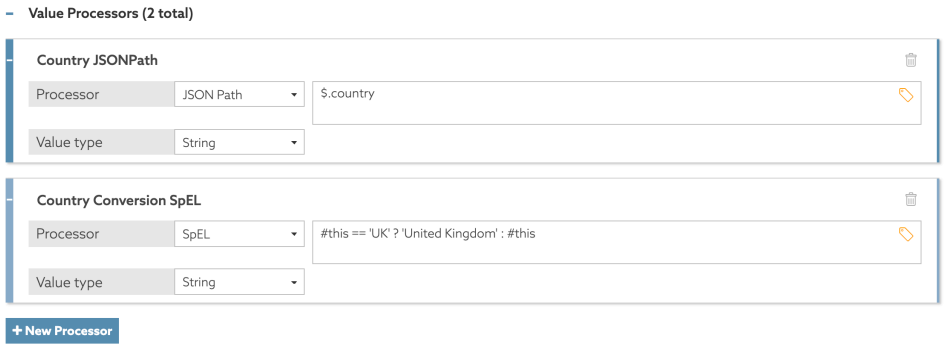
Chained value processors

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.



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.

Trust Framework testing

The PingAuthorize Policy Editor 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

Testing repeating attributes

Repeating attributes are resolved from values in a specified collection. A repeating attribute requires a repetition source that points to a collection. Also, to get its values from each repetition of the collection, the repeating attribute's resolver must be set to **Current Repetition Value**. When you properly configure a repeating attribute, you can test it the same way you test regular, nonrepeating attributes.

The **Output** tab in the test results will show results for each matching value from the collection. The results are ordered with indices that reflect the order of resolution.

For more information about these variables, see *Repeating policies and attributes* on page 446.

Viewing Trust Framework entity dependencies

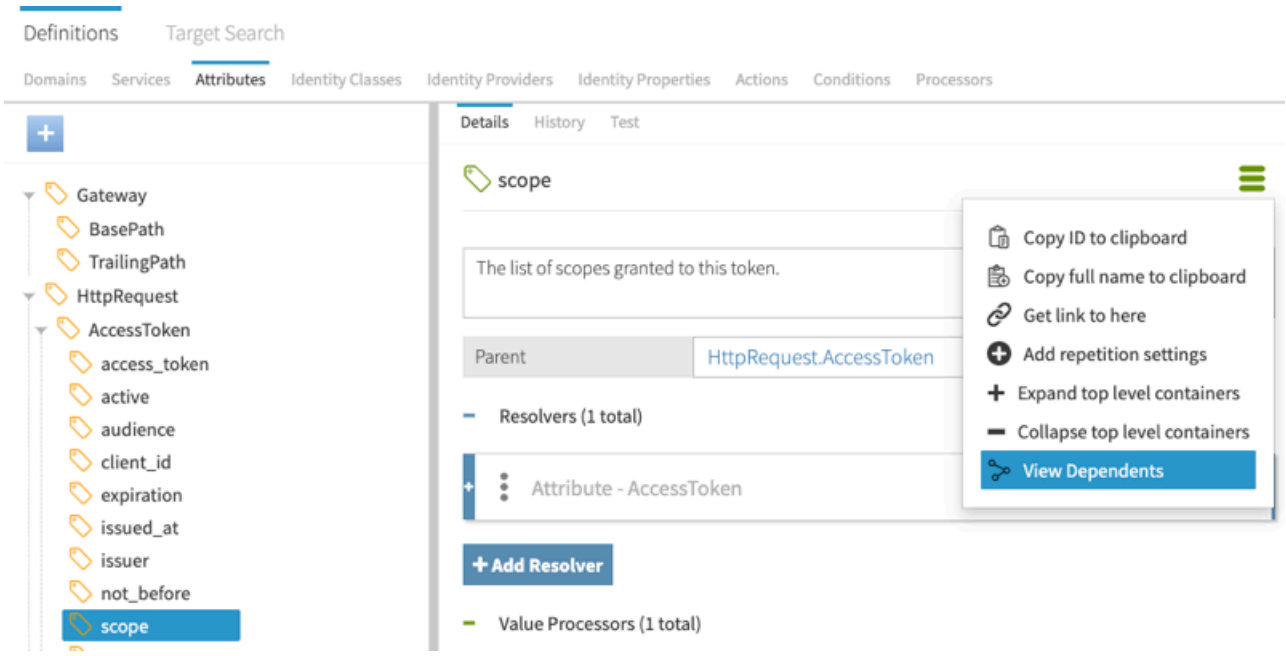
Before you change an entity, check what depends on that entity so that you do not introduce unintended consequences.

Steps

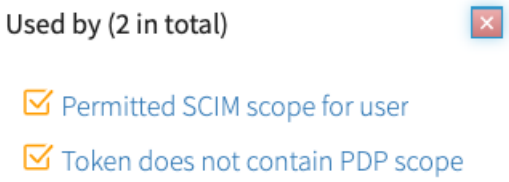
1. Go to the Trust Framework entity.
2. Click the Trust Framework entity.
3. Click the three-line icon to the right of the entity name.

4. Click **View Dependents**.

   The following image shows the `scope` attribute.



   If the entity has dependents, they are shown in a new window. For example, the following image shows the dependents for `scope`.



Policy management

The Policy Manager provides the tools to implement attribute-based access control and dynamic authorization management, 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 PingAuthorize Policy Editor 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 Editor. Fortunately, the Policy Manager makes harnessing this power straightforward.

Policy sets, policies, and rules

The PingAuthorize Policy Editor reflects the structure of grouping rules for attribute-based 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

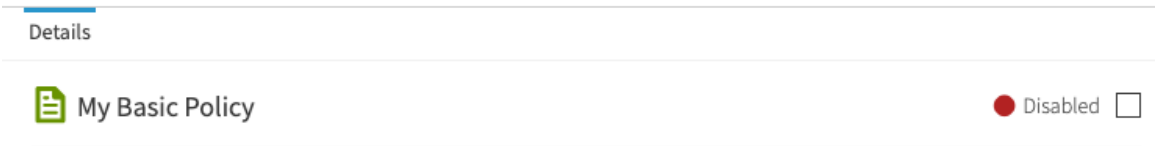
1. Click **Policies**.
2. Click **+**.
3. Select **Add Policy Set** or **Add Policy**, as appropriate.

   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. Update the policy to include targets, advice, and other changes, as discussed in the subsections after this one.
5. Click **Save changes**.

   After you create a policy, you can modify it to be a repeating policy. For more information, see *Repeating policies and attributes* on page 446.

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.



Adding targets to a policy

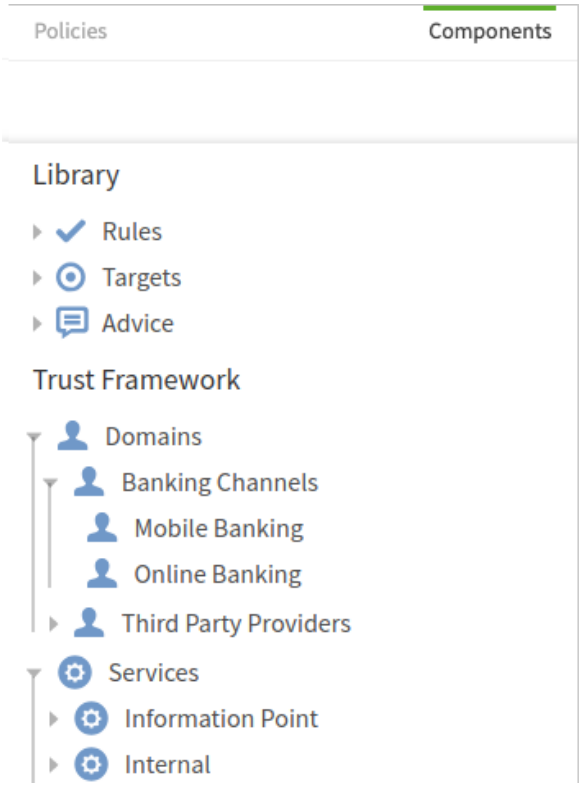
Add targets to identify the requests to which the policy applies its fine-grained authorization logic. If no targets are attached to a policy, the policy applies to all requests. To make a policy only apply for all requests to a certain database, for example, add the database domain as a target.

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.

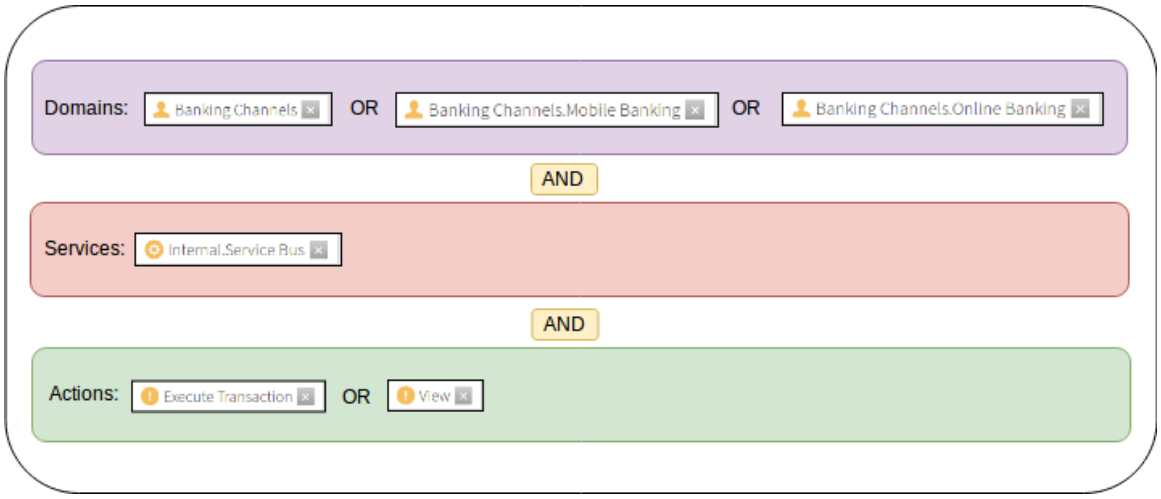


4. Click **Save changes**.

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.



Conditional targets (applies when)
You can use conditional targets to extend the capability of the "Applies to" concept when creating attribute-based access control rules and policies.

Conditional targets extend the capability of the "Applies to" concept because they:

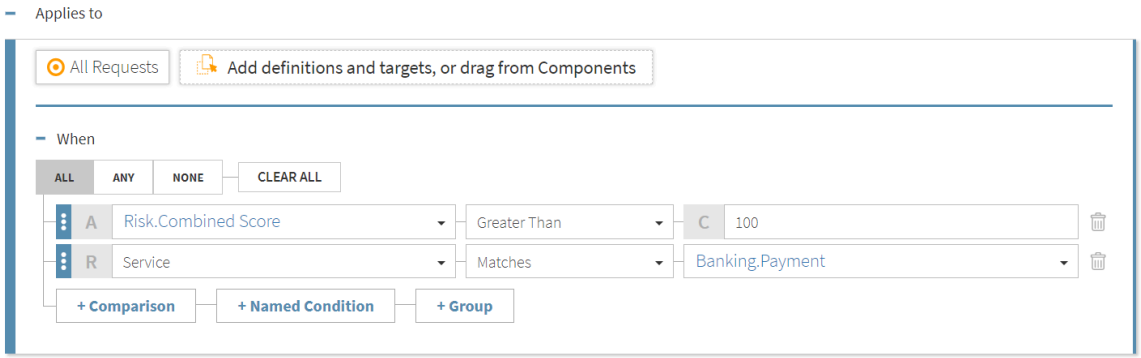
- Permit the interweaving of targets with other conditional logic.
- Allow standalone logic to determine if and when a policy or rule applies.

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.



You can navigate conditions using the Up Arrow and the Down Arrow to move between members of a group or using the Left Arrow and Right Arrow to move in and out of nested groups.

You can reorder conditions by dragging the handles on the left. To reorder using the keyboard, press Tab to go to the condition, press Enter to select the condition, press the Up Arrow or Down Arrow to go to the desired location, press Enter to drop the condition in the new location.

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

Advice
An advice is additional information you can attach to a decision response.

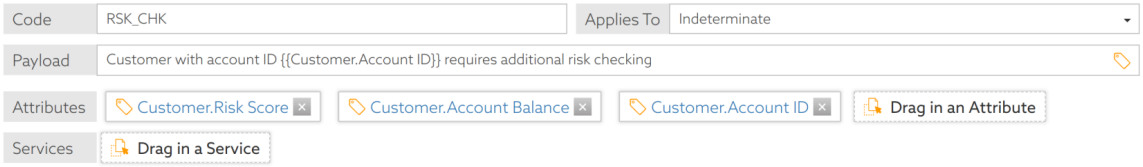
An advice returns to the governance engine so that, depending on the evaluation response from the policy, PingAuthorize can take the appropriate action. If you have a policy set up to verify the authentication level of a user, and if the policy evaluates that a user does not possess the required access privileges, then PingAuthorize can send details about the reason for denying access.

To indicate that the final decision applies only if an advice can be fulfilled, mark the advice as **Obligatory**. Typically, the service that calls PingAuthorize 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, PingAuthorize 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.



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.

You can reorder collapsed advices by dragging the handles on the left. To reorder using the keyboard, press Tab to go to the advice, press Enter to select the advice, press the Up Arrow or Down Arrow to go to the desired location, press Enter to drop the advice in the new location.

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 PingAuthorize cannot fulfill an obligatory advice, it fails the operation and returns an error to the client application. If PingAuthorize cannot fulfill a non-obligatory advice, the server logs an error, but the client's requested operation continues.
`Code`	Identifies the advice type. This value corresponds to an advice ID that the PingAuthorize configuration defines.
`Applies To`	Specifies the policy decisions, such as `permit` or `deny`, that include the advice with the policy result.

Property	Description
`Payload`	Set of parameters governing the actions that the advice performs when PingAuthorize applies the advice. The appropriate payload value depends on the advice type.

PingAuthorize Server supports the following advice types:

- *Add Filter* on page 492
- *Combine SCIM Search Authorizations* on page 492
- *Denied Reason* on page 493
- *Exclude Attributes* on page 493
- *Filter Response* on page 494
- *Include Attributes* on page 495
- *Modify Attributes* on page 496
- *Modify Headers* on page 496
- *Modify Query* on page 497
- *Modify SCIM Patch* on page 497
- *Regex Replace Attributes* on page 499

To develop custom advice types, use the Server SDK.

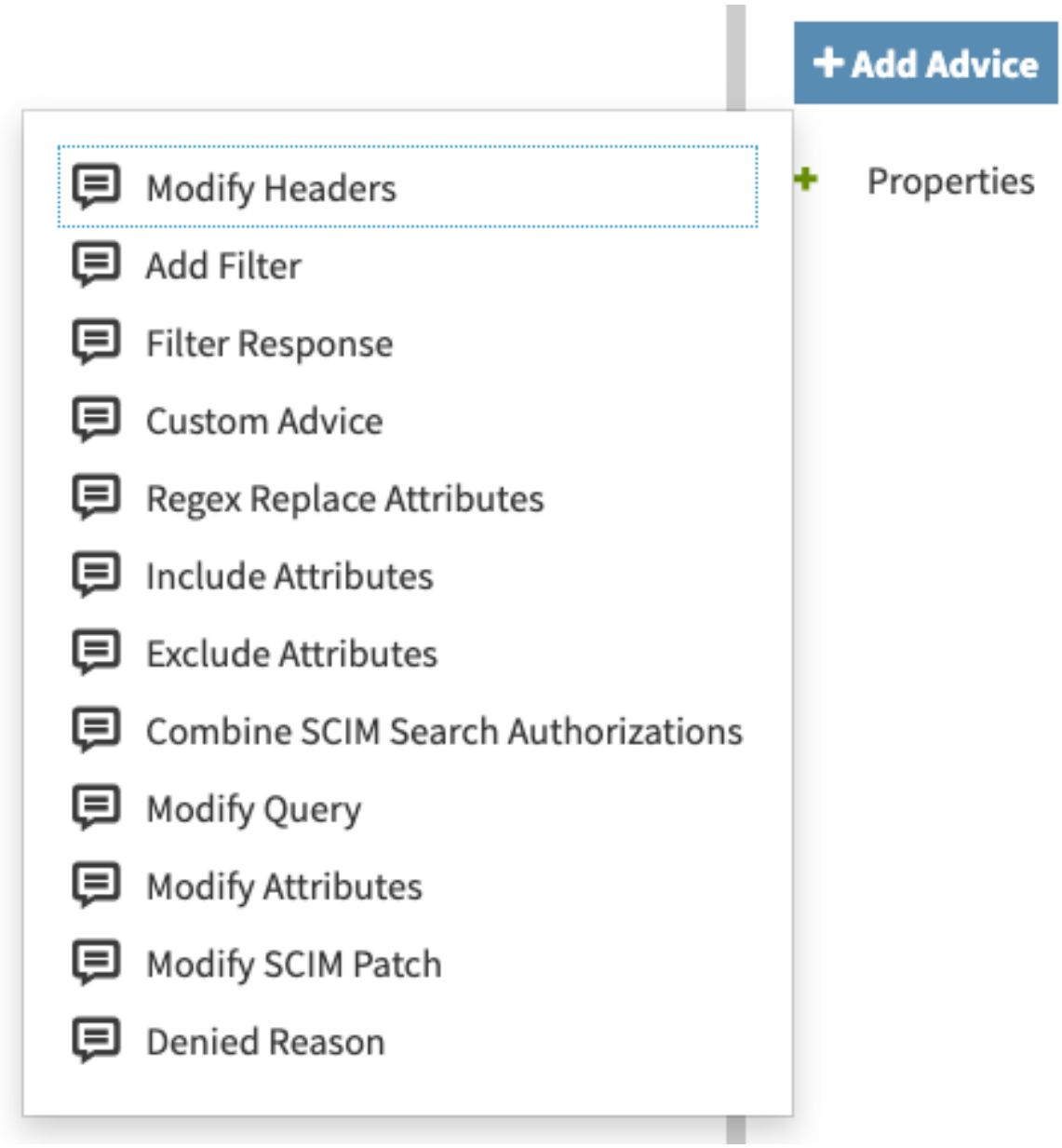
Note:

Many statement types let you use the JSONPath expression language to specify JSON field paths. To experiment with JSONPath, use this *JSONPath evaluator*.

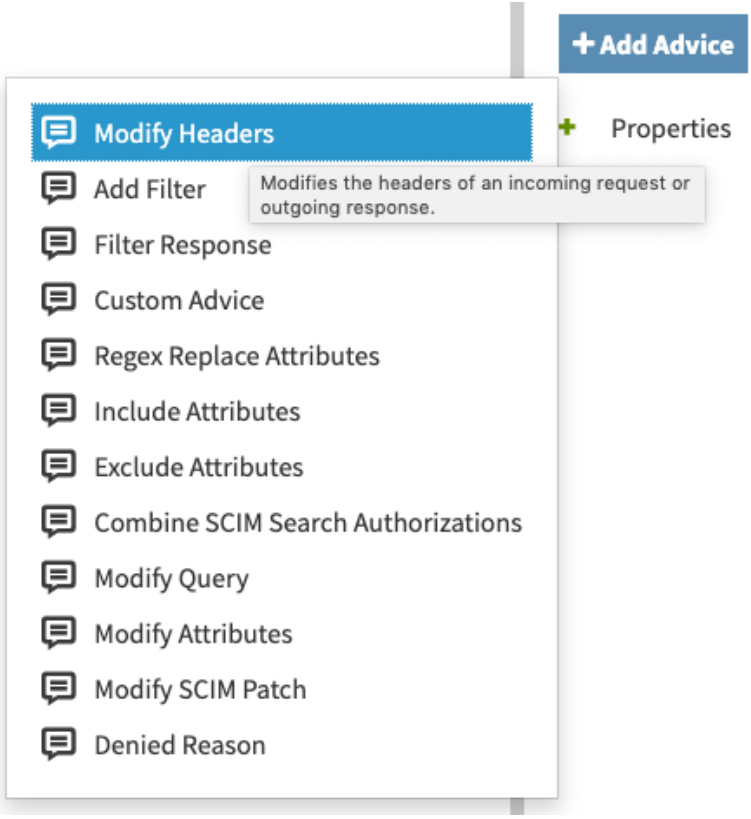
Provided advice

The PingAuthorize Policy Editor comes with preconfigured advice types that are also in PingAuthorize Server.

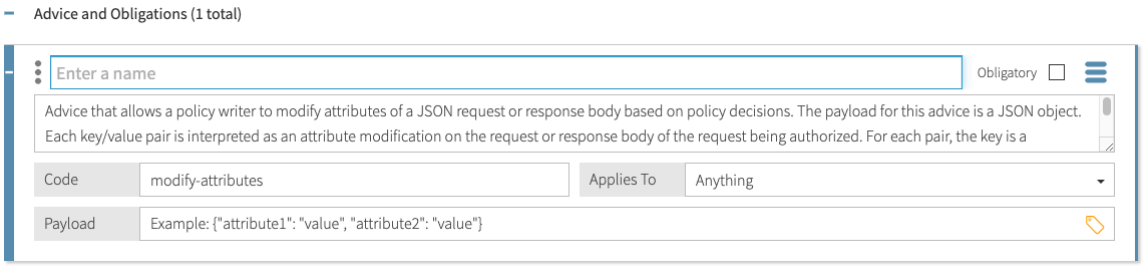
Policy writers can use this advice out of the box, and PingAuthorize Server fulfills the advice as documented. To view the full set of provided advice types, click **+ Add Advice**.



You can see the documentation for the provided advice types from within the Policy Editor. After you click **+ Add Advice**, hover over an advice type to view its description.



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



For more information, see *Advice types*.

Custom advice

In addition to the advice types that are available out of the box in the PingAuthorize Policy Editor, policy writers can use a custom advice that leverages the PingAuthorize Server SDK.

For information about the implementation and configuration of such advice, see the *PingAuthorize 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

Policies can include one or more rules to produce a fine-grained authorization decision of **`Permit`**, **`Deny`**, **`Indeterminate`**, or **`Not Applicable`**.

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	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`**.
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`**.

Combining algorithm	Summary	Details
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

Policy rules power the fine-grained access control capability in PingAuthorize. Rules contain logical conditions that evaluate to `true` or `false`.

When you create a rule, you set the conditions and criteria that dictate when the rule applies and how the rule evaluates. The rule structure begins with the **Applies to** criteria, which define the conditions under which the rule applies.

Applies to

By default, rules target all requests with no conditions. You can leave this default criteria in place, if desired. You can also *add targets*, *set a condition*, or include a group of conditions. If the **Applies to** criteria are not met, the rule effect is **`Not Applicable`**.

Note:

The **Applies to** criteria are always enabled, whether shown or hidden. When there are **Applies to** criteria that are not met, the effect is always **`Not Applicable`**, regardless of which effect type is selected.

Effect

Whether you choose to change the **Applies to** criteria or not, you must give each rule one of the following effects:

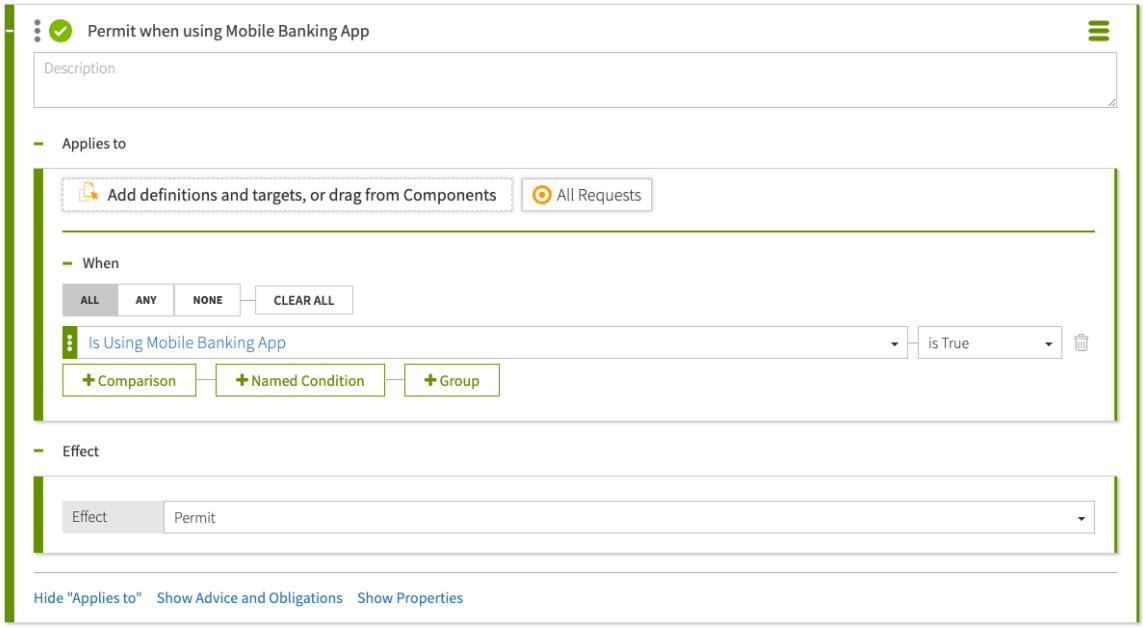
- **Permit**
- **Deny**
- **Permit if condition holds, otherwise deny**
- **Deny if condition holds, otherwise permit**

If the **Applies to** criteria evaluate to `true`, the **Permit** and **Deny** effects cause the rule to permit or deny, respectively.

The following example includes an **Applies to** condition and a **Permit** effect:

- If the condition evaluates to `true`, the rule permits.
- If it evaluates to `false`, the effect is **`Not Applicable`**.

If the example included a **Deny** effect instead, the rule would deny when the **Applies to** condition evaluated to `true`.



To configure a rule to permit or deny based on how its **Effect** conditions evaluate, choose one of the following effect types:

- The **Permit if condition holds, otherwise deny** effect causes the rule to permit if the conditions are met and to deny if the conditions are not met.
- The **Deny if condition holds, otherwise permit** effect does the opposite, causing the rule to deny if the conditions are met and to permit if the conditions are not met.

Note:

Effect conditions are hidden until you select one of the **if condition holds** effect types.

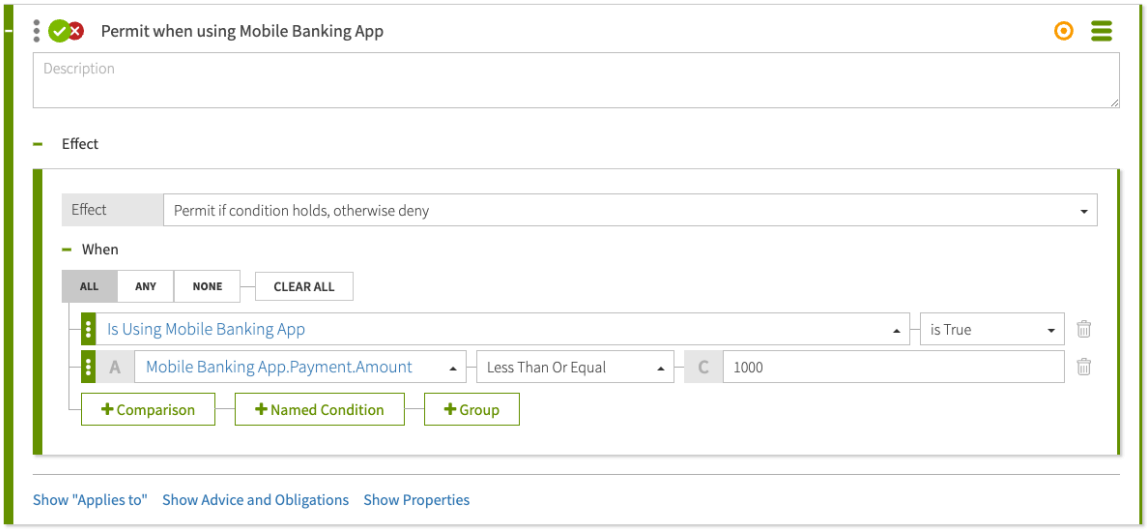
Tip:

- 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.
- Just as with *Trust Framework entities*, you can check which entities depend on a policy or policy set.

The following example includes a **Permit if condition holds, otherwise deny** effect without any **Applies to** criteria:

- If the group **Effect** condition evaluates to `true`, the rule permits.
- If the group condition evaluates to `false`, the rule denies.

When there are no **Applies to** criteria, the rule always permits or denies.



Tip:

Rules with conditional effects display two effect icons in the rule header. The icon for the **`if condition holds`** effect displays on the left and is larger than the icon for the **`otherwise`** effect.

Rule order

When a policy has multiple rules, rule order can affect the way the policy evaluates. You can reorder collapsed rules by dragging the handles on the left. To reorder rules using the keyboard, do the following:

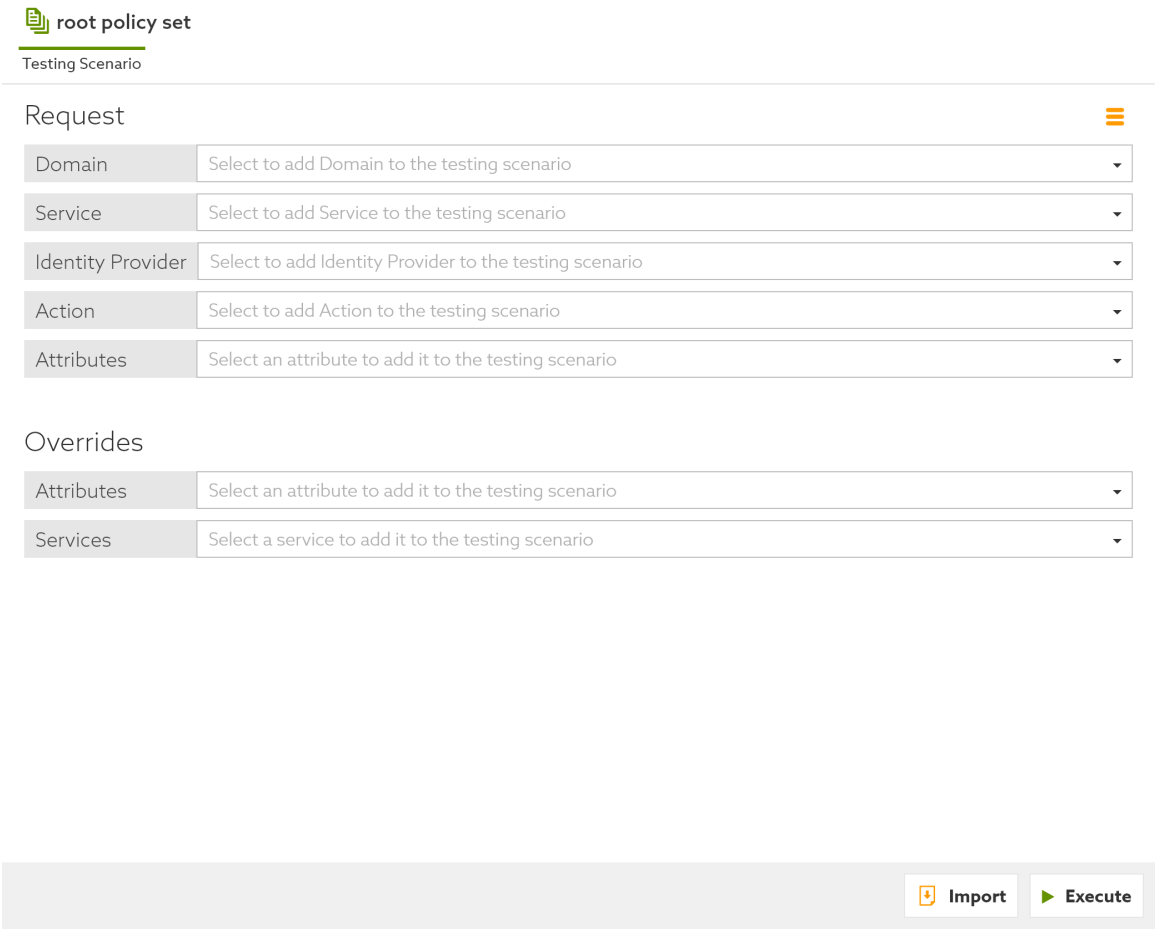
1. Press Tab to move the cursor to a rule. When the cursor is positioned on the entire rule, a blue box displays and the rule changes color to purple.
2. Press Enter to select the rule. When a rule is selected, it changes color to dark green.
3. Press the Up Arrow or Down Arrow to move the cursor to the desired location.
4. Press Enter to drop the selected rule in the new location.

Policy testing

The PingAuthorize Policy Editor 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.



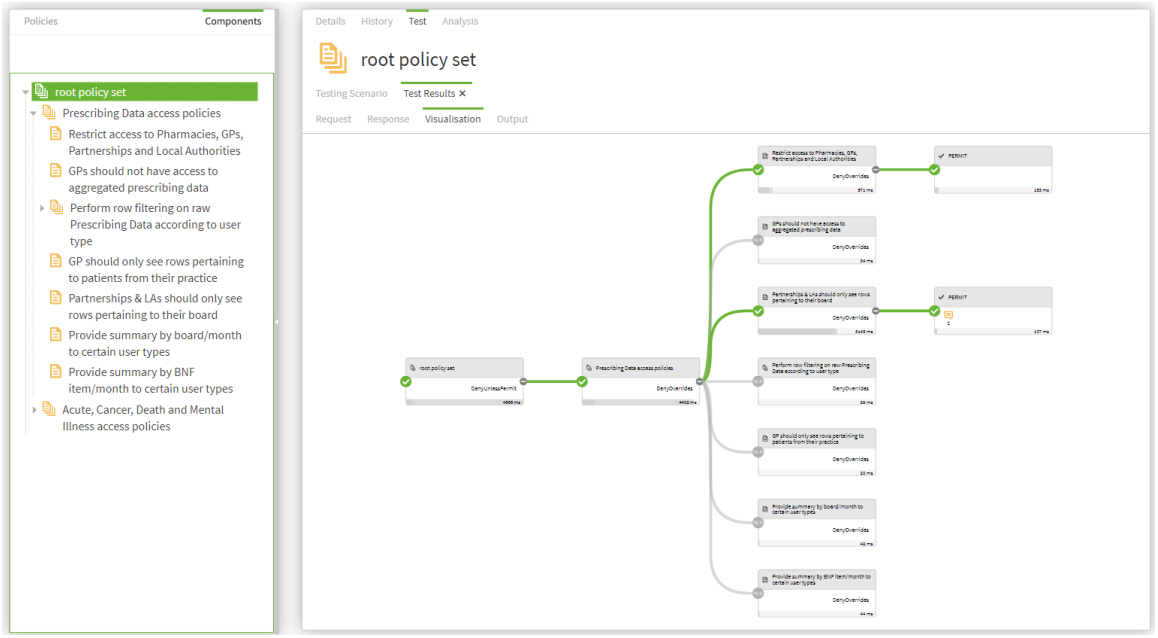
Select the following main elements to form a request:

- 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.



Repeating policies and attributes

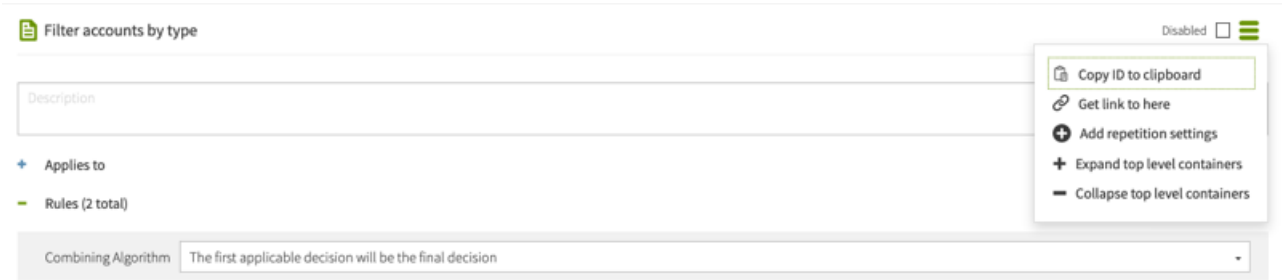
Use repeating policies and attributes to evaluate a policy multiple times, once for each item in a collection.

For example, assume the `Accounts` attribute contains a list of accounts associated with a customer. You want to filter access to the accounts based on the account type. With repeating policies, a decision is made for each item in the `Accounts` attribute, returning advice for each account that is permitted.

Repeating policies

To make a policy repeat, from the three-line menu, select **Add repetition settings**.

> **Note:**
>
> You can only add repetition settings to an existing policy. The three-line menu to add these settings does not appear when you are creating a new policy.



The policy repetition settings are described below.

- **Apply this policy to each item of**

  The collection attribute to repeat over.

  This item is referred to as the repetition source.

- **Filtering by**

  The decision and any attached advice to filter by.

The following example uses the `Accounts` attribute and `Permit` decision. In this case, the policy applies to every item in the `Accounts` collection attribute. The policy keeps each result that returns Permit.

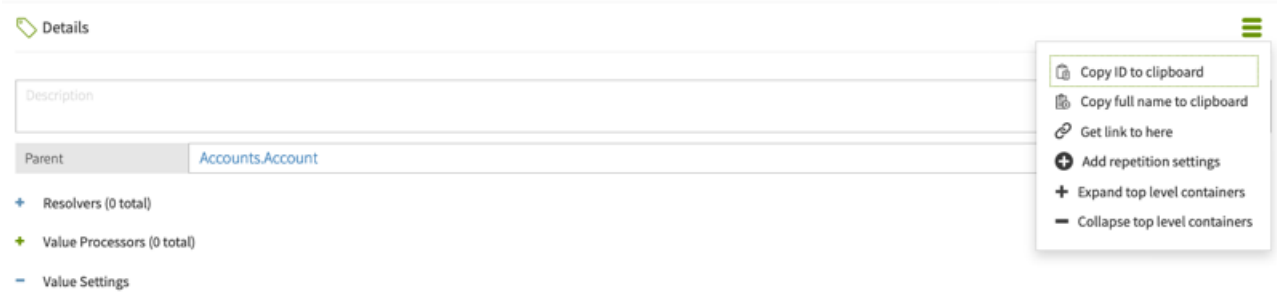
When you define rules and advice for a repeating policy, you can use:

- Attributes with no repetition source
- Attributes with the same repetition source as the policy

Repeating attributes

To make an attribute repeat, from the three-line menu, select **Add repetition settings**.

> **Note:**
>
> You can only add repetition settings to an existing attribute. The three-line menu to add these settings does not appear when you are creating a new attribute.



The policy repetition settings are described below.

- **Repeat for each item of**

  > **Note:**
  >
  > If you set this field, you can only use the attribute in repeating policies. However, the attribute can then resolve against attributes repeating against the same collection. The attribute can still resolve against attributes that do not have this field set.

  The attribute to repeat over.

  This item is referred to as the repetition source.

- **Resolvers**, **Value Processors**, **Caching**

  For a resolver, if **Resolver type** is **Current Repetition Value**, resolution is against individual items in the collection itself.

  For information about these items, see *Resolvers* on page 418, *Value processing for an attribute* on page 422, and *Attribute caching* on page 422.

You can use repeating attributes in named conditions and value processors. If an attribute uses a named condition or value processor, any repeating attributes referenced in the condition or value processor must have the same repetition source as the attribute itself. If a policy uses a named condition, any repeating attributes referenced in the condition must have the same repetition source as the policy itself.

Policy solutions

This section recommends how to implement commonly needed business rules in policy.

- *Use case: Using consent to determine access to a resource* on page 448
- *Use case: Using consent to change a response* on page 463
- *Use case: Using a SCIM resource type or a policy request action to control behavior* on page 471
- *Restricting the modification of attributes* on page 487

Use case: Using consent to determine access to a resource

PingAuthorize can provide attribute-based access control to a specific resource based on the resource owner's consent to share.

Examples of resources include:

- Health care records shared with a spouse (an individual)
- Banking records shared with a known third party, such as an asset-monitoring tool
- Purchase history shared with an anonymous third party, possibly for improved promotional offers

In this scenario, we continue using the meme games API used in *Getting started with PingAuthorize (tutorials)* on page 17. Assume my friend has crafted several funny memes that she wants to share with me. When my browser or app requests her memes, PingAuthorize enforces access based on her consent to share.

We first set up some Trust Framework attributes and services and then create a policy that uses those items to check consent and then permit or deny access. The following topics cover these tasks.

1. *Getting a path component from the request URL* on page 448
2. *Getting the requestor identifier from the access token* on page 453
3. *Searching for consent by resource owner to requestor* on page 453
4. *Getting consent status from the consent record* on page 459
5. *Creating a policy to check consent and then permit or deny access* on page 460

Getting a path component from the request URL

For this use case, the resource owner is given in the URL for the meme game API. To get the owner requires pulling the corresponding path component from the request URL.

Before you begin

This procedure assumes you have created a meme game API server named `meme-game`, similar to the one shown in the "Configure an API External Server for the Meme Game API" step in *Configuring a reverse proxy for the Meme Game API* on page 28.

About this task

In general, you can configure PingAuthorize to control access based on the path component that best suits your needs. For example, consider the /purchases/1234 path. The purchases component is a class of resources, while 1234 is a specific resource for a given purchase.

The meme game API has URLs of the form meme-game/api/v1/users/user.0/answers. The user.0 path component is a specific resource owner. The following steps explain how to get the specific resource owner from a request URL.

Steps

1. In the PingAuthorize administrative console, create a new gateway API endpoint.

   A Gateway API Endpoint controls how PingAuthorize 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 user_answers`.
   d. For **Inbound Base Path**, specify `/meme-game/api/v1/users/{UserFromUrl}/answers`.

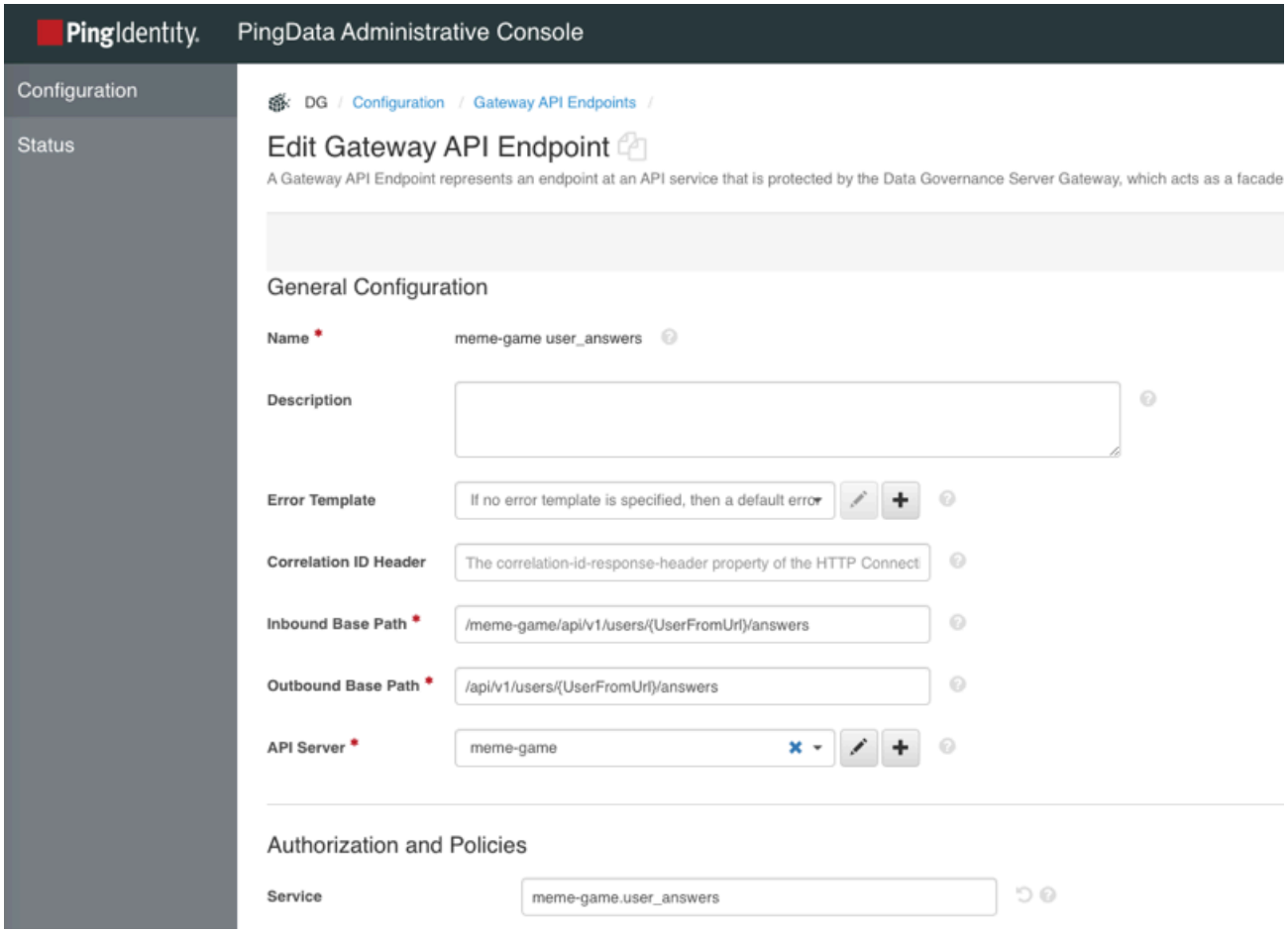
      The inbound base path defines the base request path for requests to be received by PingAuthorize Server.

      Using the curly braces (`{` and `}`) around a string creates an item with the name given by the string so that we can refer to it later. That notation also preserves the item to pass along in the next step.
   e. For **Outbound Base Path**, specify `/api/v1/users/{UserFromUrl}/answers`.

      The outbound base path defines the base request path for requests that PingAuthorize Server forwards to an API server.
   f. For **API Server**, specify `meme-game`. This is the API External Server you defined previously.
   g. For **Service**, specify `meme-game.user_answers`.

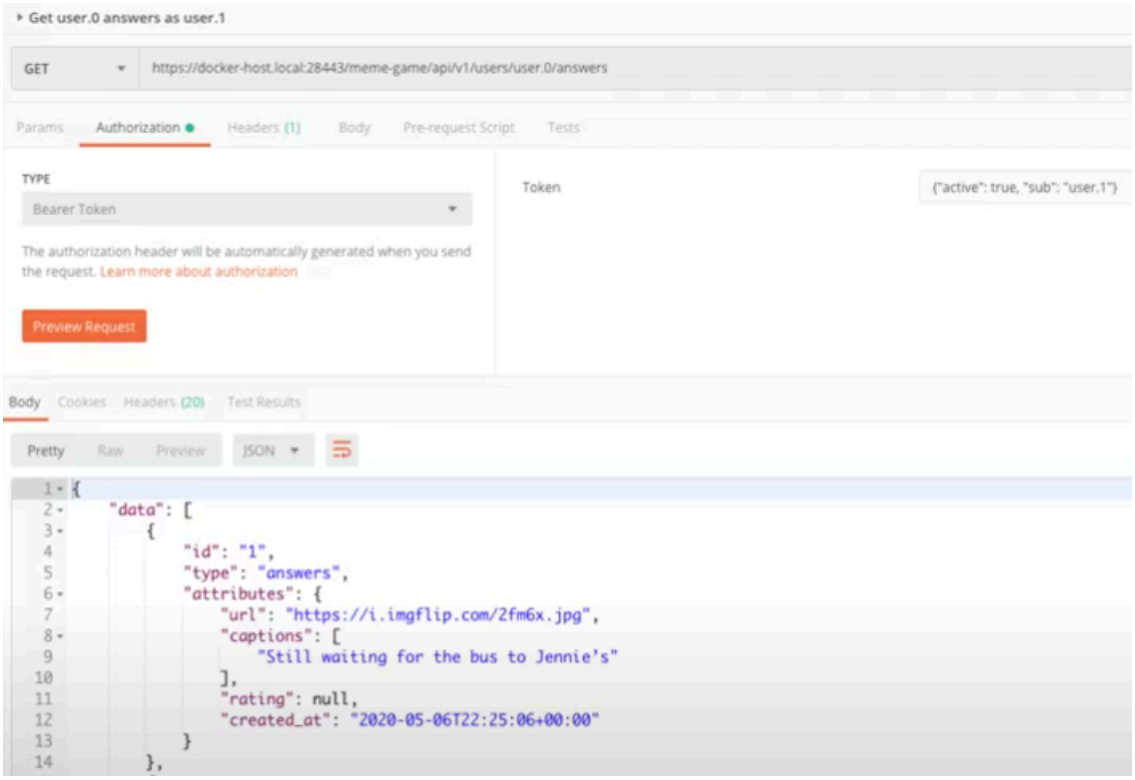
      You will use this service in the PingAuthorize Policy Editor to get a value to define an attribute.

      The following image shows this configuration.



   h. Save your changes.

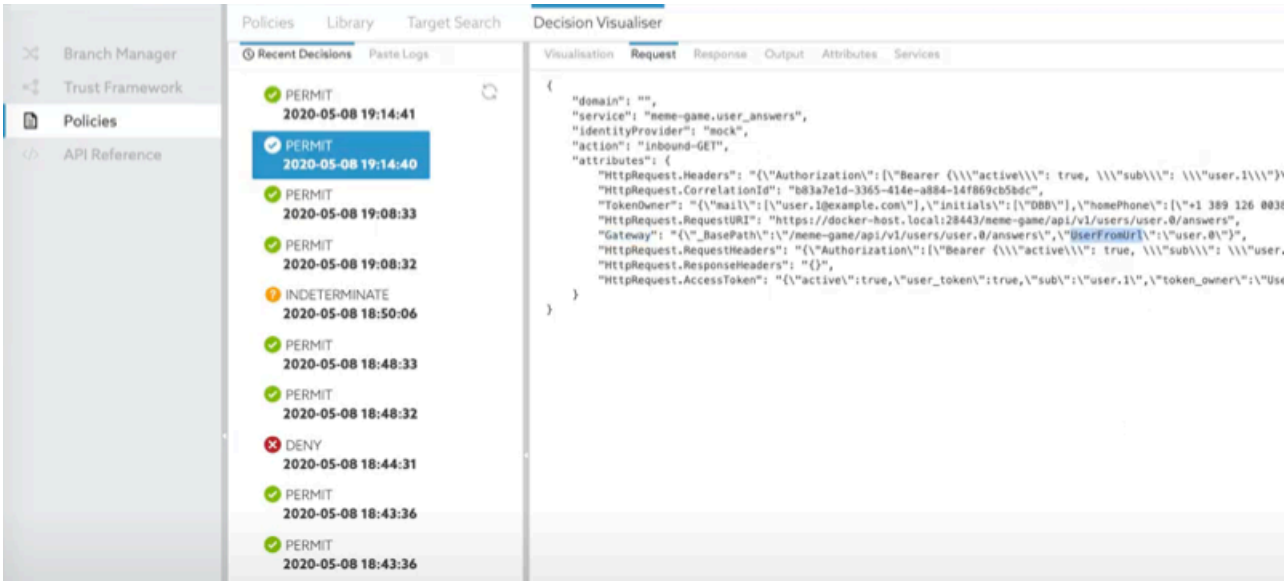
2. Send a test request to the gateway to see how PingAuthorize handles the request. The following request uses Postman.



3. Check the request in the Policy Editor.

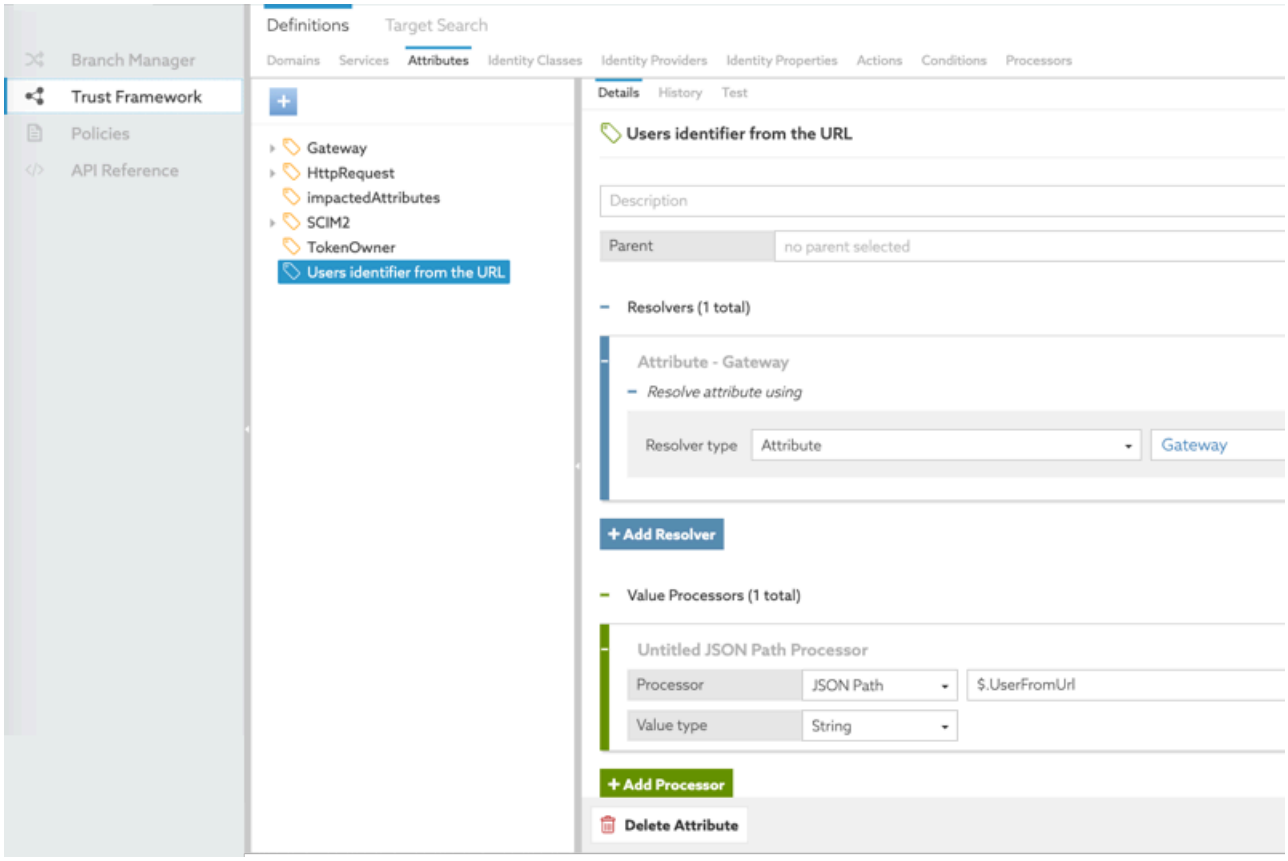
   Go to **Policies** in the left pane and then click **Decision Visualiser** along the top. Under **Recent Decisions**, click **Refresh** icon. Select the decision and click **Request**.

   In the request, the attributes include a Gateway object. Items set in the gateway API endpoint in the previous step are in this Gateway object. One of the items in the object is UserFromUrl, providing the exact path component we want. The following image shows the Gateway object.



4. Create an attribute to pull UserFromUrl from the object.
   a. Go to **Trust Framework** and then click **Attributes** along the top.
   b. From the **+** menu, select **Add new Attribute**.
   c. For the name, replace **Untitled** with `Users identifier from the URL`.
   d. Click the **+** next to **Resolvers** and click **+ Add Resolver**.
   e. Set **Resolver type** to **Attribute** and select the **Gateway** attribute.
   f. Click the **+** next to **Value Processors** and click **+ Add Processor**.
   g. Set **Processor** to **JSON Path** to pull an item from a JSON object and specify a value of `$.UserFromUrl`.

      The following image shows this configuration.

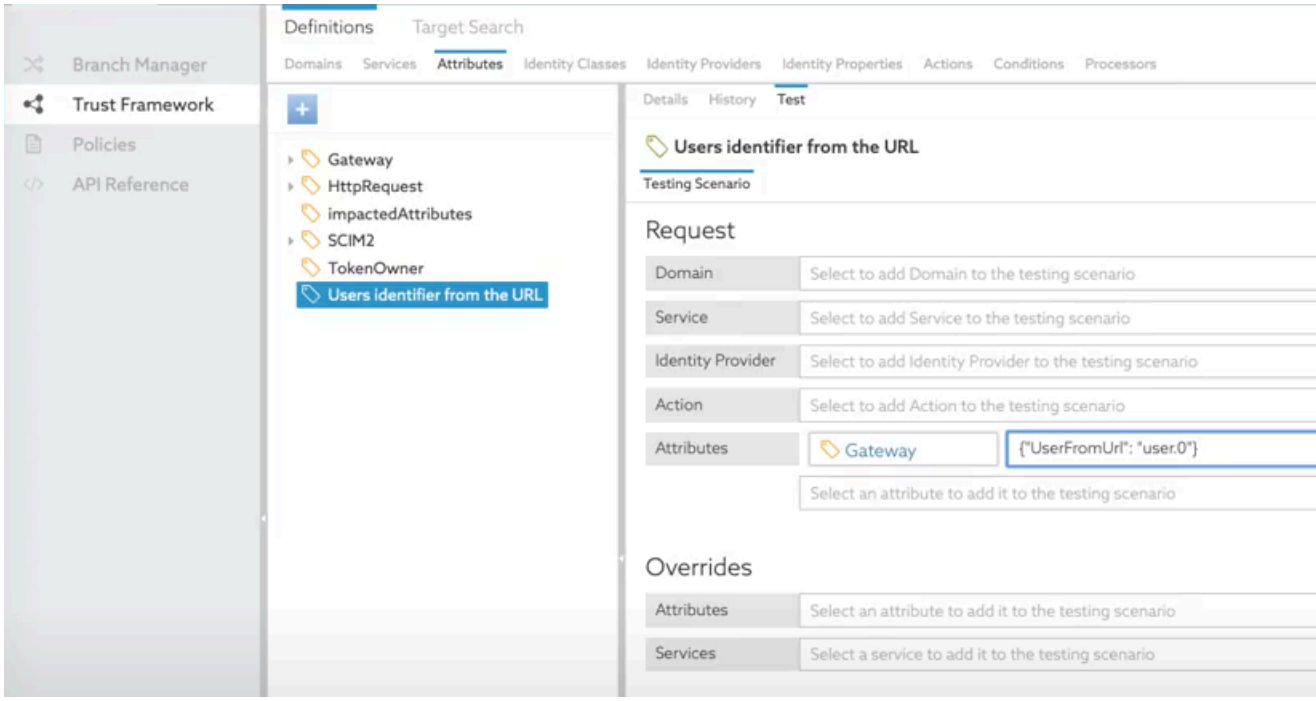


   h. Click **Save changes**.

5. Test the new attribute.
   a. Click **Test** just above the attribute name.
   b. Pass in a gateway object that uses UserFromUrl.

      In the **Request**, set **Attributes** to `Gateway` and specify a value of `{"UserFromUrl":"user.0"}`.

      The next image shows the test setup.



   c. Click **Execute**.
      The test results should be user.0.

Result
The `Users identifier from the URL` attribute is available for use in policies.

Getting the requestor identifier from the access token
We need the requestor identifier to check whether the resource owner has given the requestor access to the resource.

About this task
The PingAuthorize Policy Editor provides many attributes, including `HttpRequest.AccessToken`. The `HttpRequest.AccessToken.subject` attribute has the needed information.

Steps

- Be prepared to use the `HttpRequest.AccessToken` attribute in a later step.

Searching for consent by resource owner to requestor
Using the resource owner information from the `Users identifier from the URL` attribute, we need to determine what consent the owner has granted to a given requestor.

About this task

This task is useful for:

- Resource sharing or delegation where consent is granted to an individual (based on the `collaborator` claim)
- Data sharing where consent is granted to a third party (based on the `audience` claim)

This task uses the Trust Framework HTTP service to pull a claim from a request.

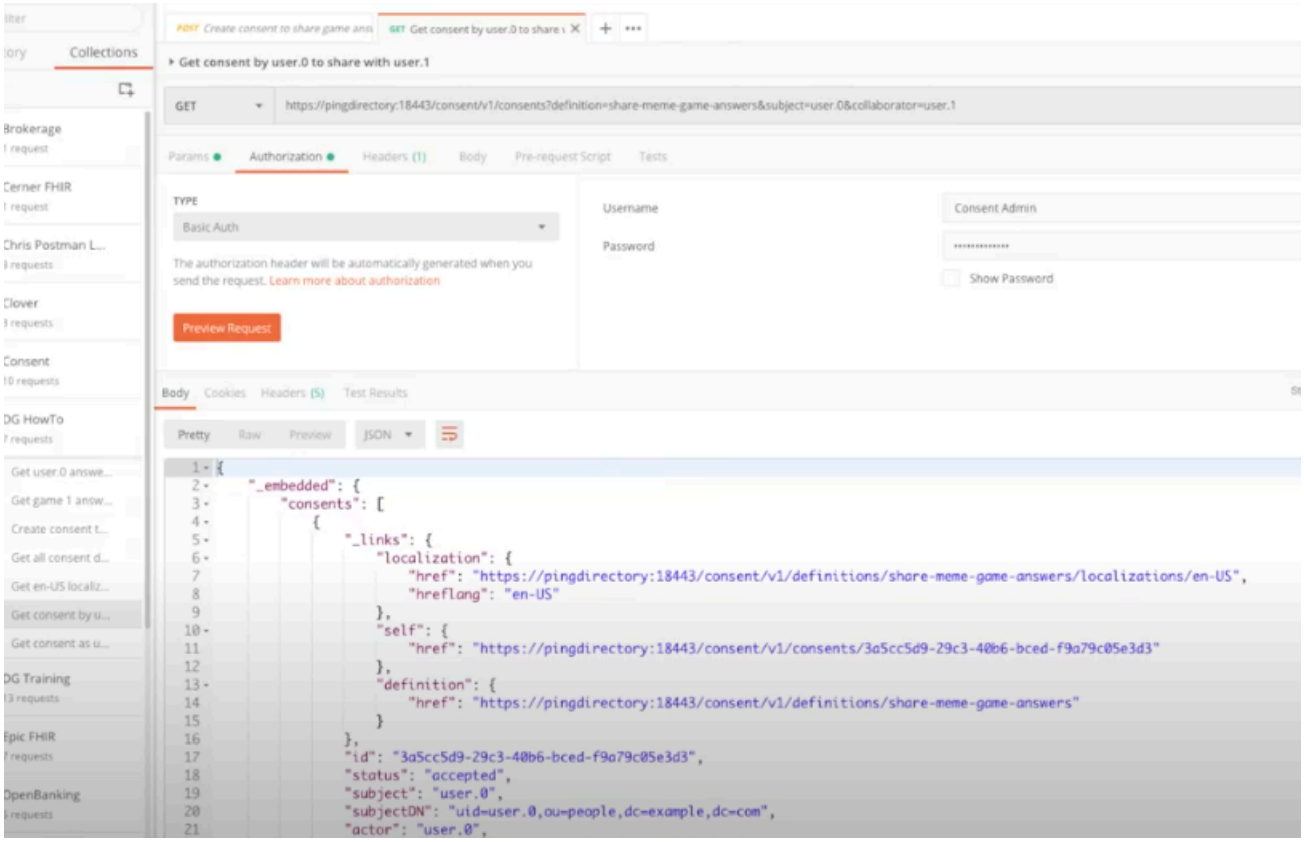
Steps

1. Make sure you understand the body of the request that you are pulling a claim from.

   The following Postman image shows a request being made to a directory server. The consent definition is in the request URL and has the form share-meme-game-answers&subject=user.0&collaborator=user.1. The resource owner is given by the subject, and the person being shared with is given by the collaborator.

   We use the Consent Admin account for the service. In Postman, for Authorization, we use BasicAuth with the username Consent Admin and its password.

   The consent record is for the PingDirectory Consent API, but you can use other consent stores. We use this consent record to determine who a resource owner has given consent to.



2. Copy the request URL to use in defining a Trust Framework service in the Policy Editor.
3. Sign on to the Policy Editor.

4. Create Trust Framework attributes for the Consent Admin account credentials.

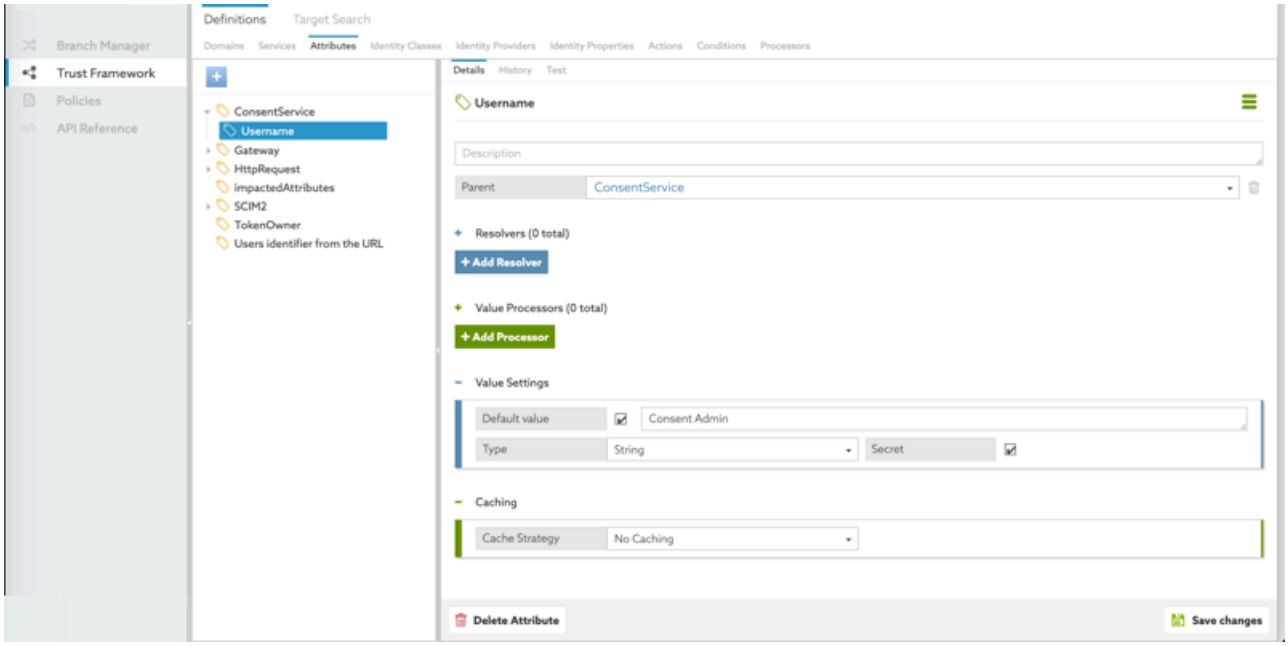
   This is the Consent Admin account we used with Postman. We will create attributes for the username and password and then use those attributes when we define the Trust Framework HTTP service.

   a. Go to **Trust Framework** and click **Attributes**.
   b. From the **+** menu, select **Add new Attribute**.
   c. For the name, replace **Untitled** with `ConsentService` and click **Save changes**.

      This attribute will serve as a parent to the username and password attribute and will help organize the attributes.
   d. From the **+** menu, select **Add new Attribute**.

      Because the **ConsentService** attribute is selected, the new attribute is a child to it.
   e. For the name, replace **Untitled** with `Username`, set **Default value** to **Consent Admin**, select the **Secret** option, and then click **Save changes**.

      The following image shows this configuration.



   f. From the **+** menu, select **Add new Attribute**.
   g. For the name, replace **Untitled** with `Password`, set **Default value** to **Consent Admin**, select the **Secret** option, and then click **Save changes**. Selecting the **Secret** option keeps the item out of logs.

5. Create the HTTP service.
   a. Click **Services** along the top.
   b. From the **+** menu, select **Add new Service**.
   c. For the name, replace **Untitled** with `Search for consent to share game answers`.
   d. Set **Service Type** to HTTP.
   e. Set **URL** to the request URL.

      In this case, the URL is https://pingdirectory:18443/consent/v1/consents?definition=share-meme-game-answers&subject=user.0&collaborator=user.1.
   f. Set **Authentication** to **Basic**.

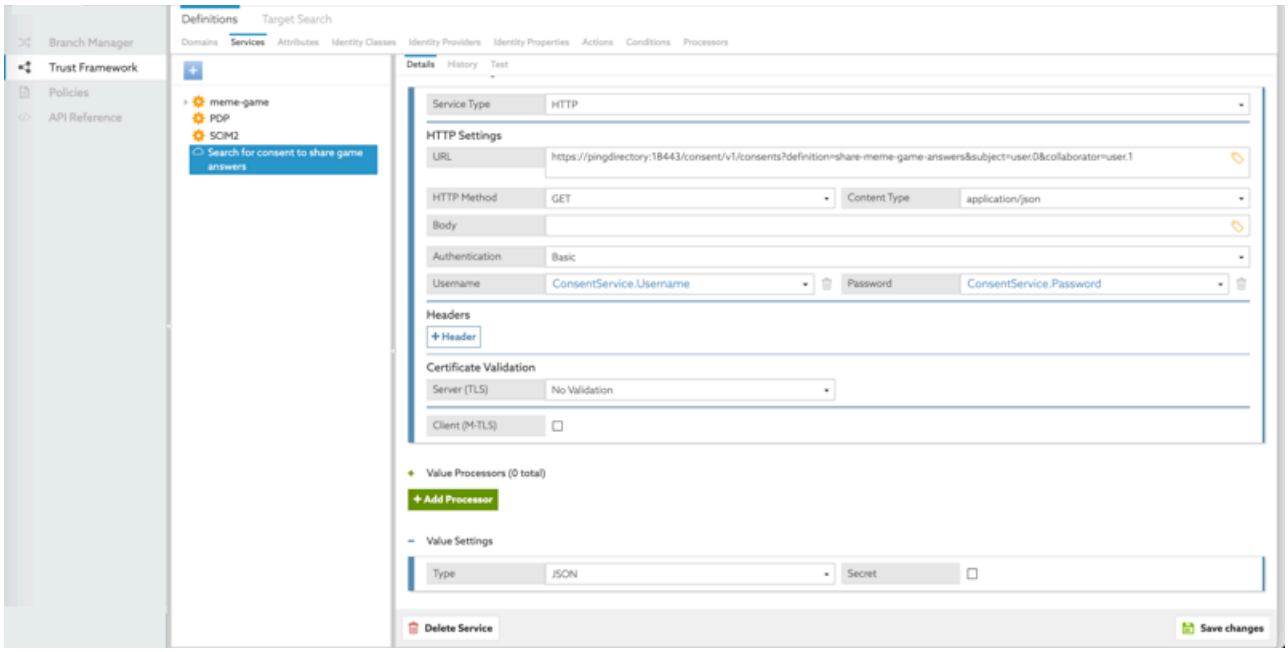
      This setting requires a username and password. We will use the attributes we just created.

      1. Set **Username** to **ConsentService.Username**.
      2. Set **Password** to **ConsentService.Password**.
   g. This setup uses a self-signed certificate, so set **Server (TLS)** to **No Validation**.

> **Note:**
> This case is for a development environment only. Do not use this setting for other environments.

   h. Under **Value Settings**, set **Type** to **JSON**.

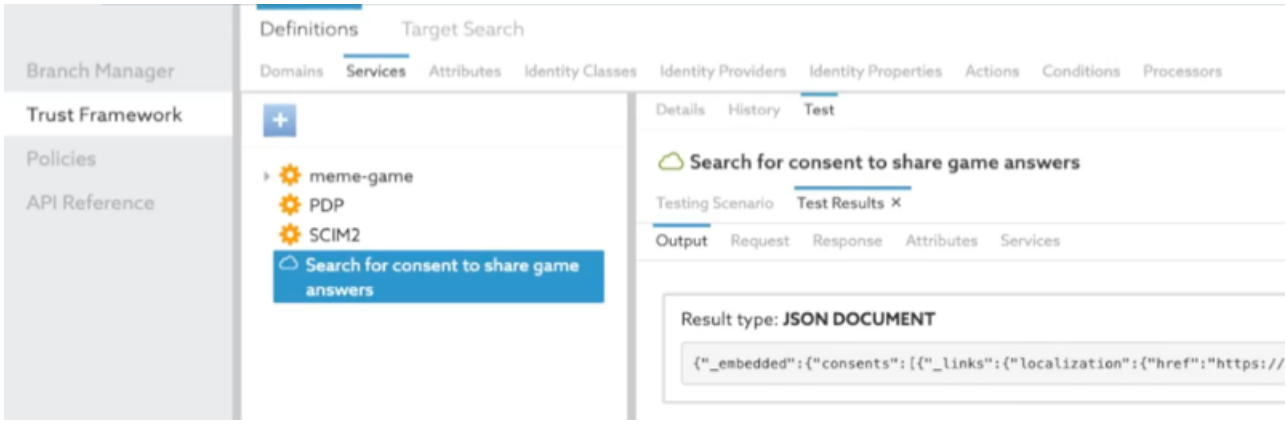
      The following image shows this configuration.



   i. Click **Save changes**.

6. Test the service.
   a. Click **Test** above the **Search for consent to share game answers** service name.
   b. Click **Execute**.

      The results should include a `consents` array.



      So the service works with hard-coded values: subject=user.0&collaborator=user.1. We need to use parameters in place of the subject and collaborator values so that the service works for anyone using the API.

7. Click **Details** above the service name to update the service definition to replace the values with parameters.
   a. In the **URL** field, replace the collaborator value, which is user.1. Delete user.1 and type two open curly braces (`{{`). Use the pop-up that appears to choose the `HttpRequest.AccessToken.subject` attribute. Recall from *Getting the requestor identifier*

from the access token on page 453 that this attribute specifies the requestor. The resource owner must have a consent record for the requestor to grant access.

With this change, the URL changes from

https://pingdirectory:18443/consent/v1/consents?definition=share-meme-game-answers&subject=user.0&collaborator=user.1

to

https://pingdirectory:18443/consent/v1/consents?definition=share-meme-game-answers&subject=user.0&collaborator={{HttpRequest.AccessToken.subject}}

b. Click **Save changes**.
c. Test the change by clicking **Test**, in the **Request** section, setting **Attributes** to `HttpRequest.AccessToken.subject`, specifying a value such as `{"sub":"user.1"}`, where user.1 has a consent record in your consent store, and clicking **Execute**.

   The result should include a consents array. Repeat the step for a user who does not have a consent record to verify that those results do not include a consents array.
d. Click **Details** to replace the subject value with a parameter.

   The subject is the resource owner. Recall from *Getting a path component from the request URL* on page 448 that we have that information in the `Users identifier from the URL` attribute. Using curly braces to interpolate that attribute, the URL becomes

   https://pingdirectory:18443/consent/v1/consents?definition=share-meme-game-answers&subject={{Users identifier from the URL}}&collaborator={{HttpRequest.AccessToken.subject}}
e. Click **Save changes**.
f. Test this change the same way you tested the previous change, using two users where one has a consent record and one does not.

   In the **Overrides** section, set **Attributes** to `Users identifier from the URL` with the value specifying the resource owner, which is user.0 in this case.

8. Update the service to pull only the first consent record from the response instead of the entire response.

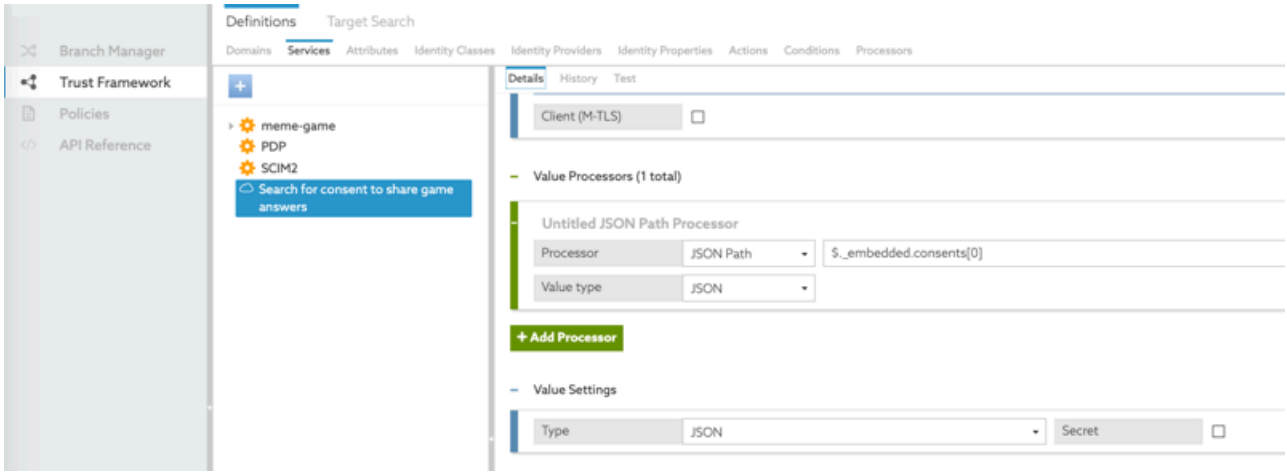
   The response starts with

   {"_embedded":{"consents":[{_links":"localization":

   We want to pull the first consent record for the user, which starts after the square bracket (`[`).

   a. Click **Details** to return to the service definition.
   b. Click the **+** next to **Value Processors** and click **+ Add Processor**.
   c. Set **Processor** to **JSON Path** with a value of `$._embedded.consents[0]`.
   d. Set **Value type** to **JSON**.

      This image shows such a screen.

   e. Click **Save changes**.
   f. Test the change by clicking **Test**, in the **Request** section, setting **Attributes** to `HttpRequest.AccessToken.subject`, and specifying a value such as `{"sub":"user.1"}`, where user.1 has a consent record in your consent store. Then in the **Overrides** section, setting **Attributes** to `Users identifier from the URL` with the value specifying user.0 again, and clicking **Execute**.

Result
The service returns only the user's first consent record. With the record isolated, you can pull the given requestor's status from the record.

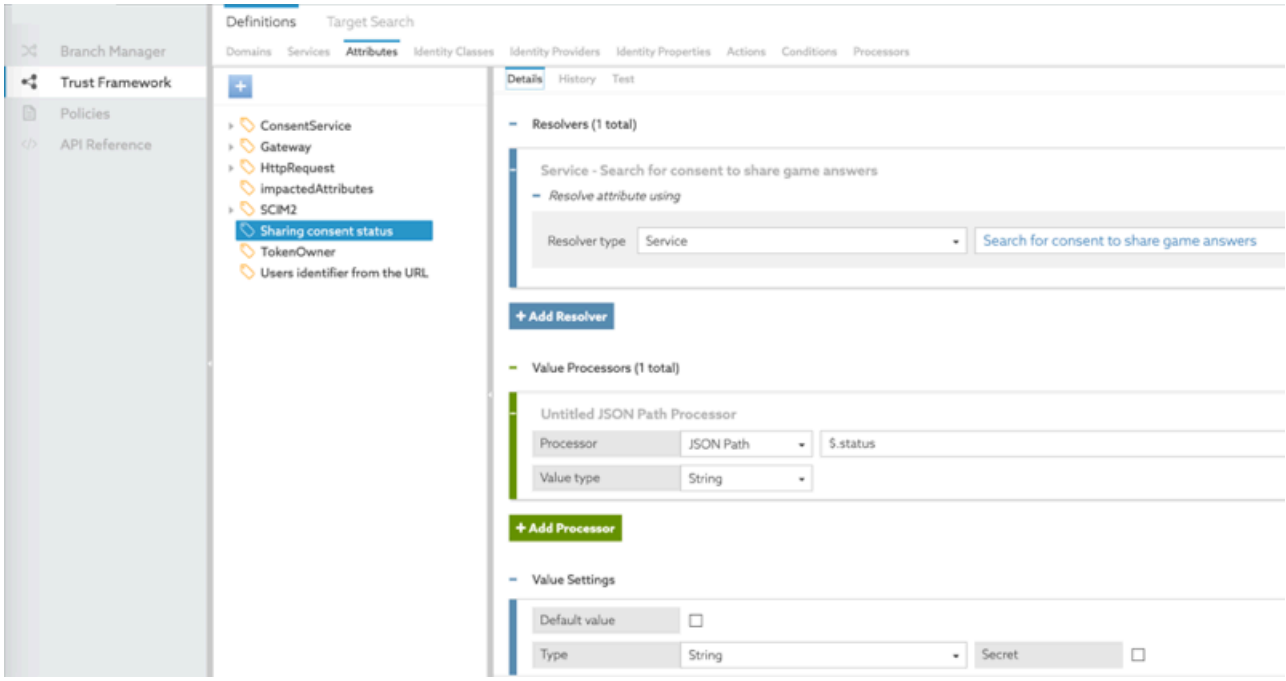
Getting consent status from the consent record
This task defines an attribute that uses a service to get a consent record and then uses a processor to pull the consent status from that record.

Steps

1. Sign on to the Policy Editor.
2. Go to **Trust Framework** and click **Attributes**.
3. From the **+** menu, select **Add new Attribute**.
4. For the name, replace **Untitled** with `Sharing consent status`.
5. Click the **+** next to **Resolvers**.
6. Click **+ Add Resolver**.
7. Set **Resolver type** to **Service** with a value of **Search for consent to share game answers**.
8. Click the **+** next to **Value Processors**.
9. Click **+ Add Processor**.

10. Set **Processor** to **JSON Path** with a value of `$.status`.
11. Set **Value type** to **String**.

    The following image shows this configuration.



12. Click **Save changes**.

Result
The `Sharing consent status` attribute is available for use in policies.

Creating a policy to check consent and then permit or deny access
Using the Trust Framework attributes and services we created, we now create a policy for the meme game API to get a user's answers. The policy permits access if consent exists and the consent status is `accepted`.

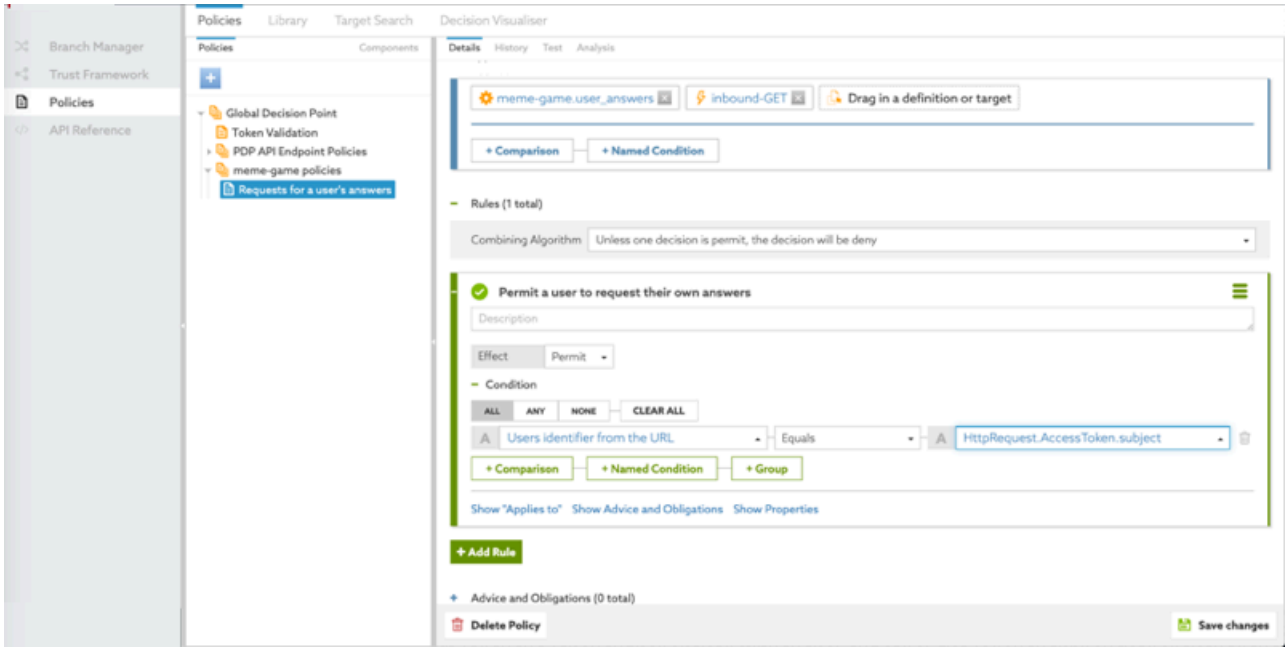
Steps

1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.

   The following steps create a policy under an existing policy called **meme-game policies**. This existing policy is for all requests to the meme game.
2. Select the existing **meme-game policies** policy.
3. From the **+** menu, select **Add Policy**.
4. For the name, replace **Untitled** with `Requests for a user's answers`.
5. Click the **+** next to **Applies to**.
6. Click **Add definitions and targets, or drag from Components** and add the **meme-game.user_answers** service, which we set up in *Getting a path component from the request URL* on page 448. Also add the **inbound-GET** action.
7. Set **Combining Algorithm** to **Unless one decision is permit, the decision will be deny**.

8. Add a rule so that a user can access their own answers.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `Permit a user to request their own answers.`
   c. Click **+ Comparison**.
   d. From the **Select an Attribute** list, select **Users identifier from the URL**, which we also set up in *Getting a path component from the request URL* on page 448.
   e. In the second field, select **Equals**.
   f. In the third field, click the **C** to toggle to an **A** (for attribute) so that you can select the **HttpRequest.AccessToken.subject** attribute.

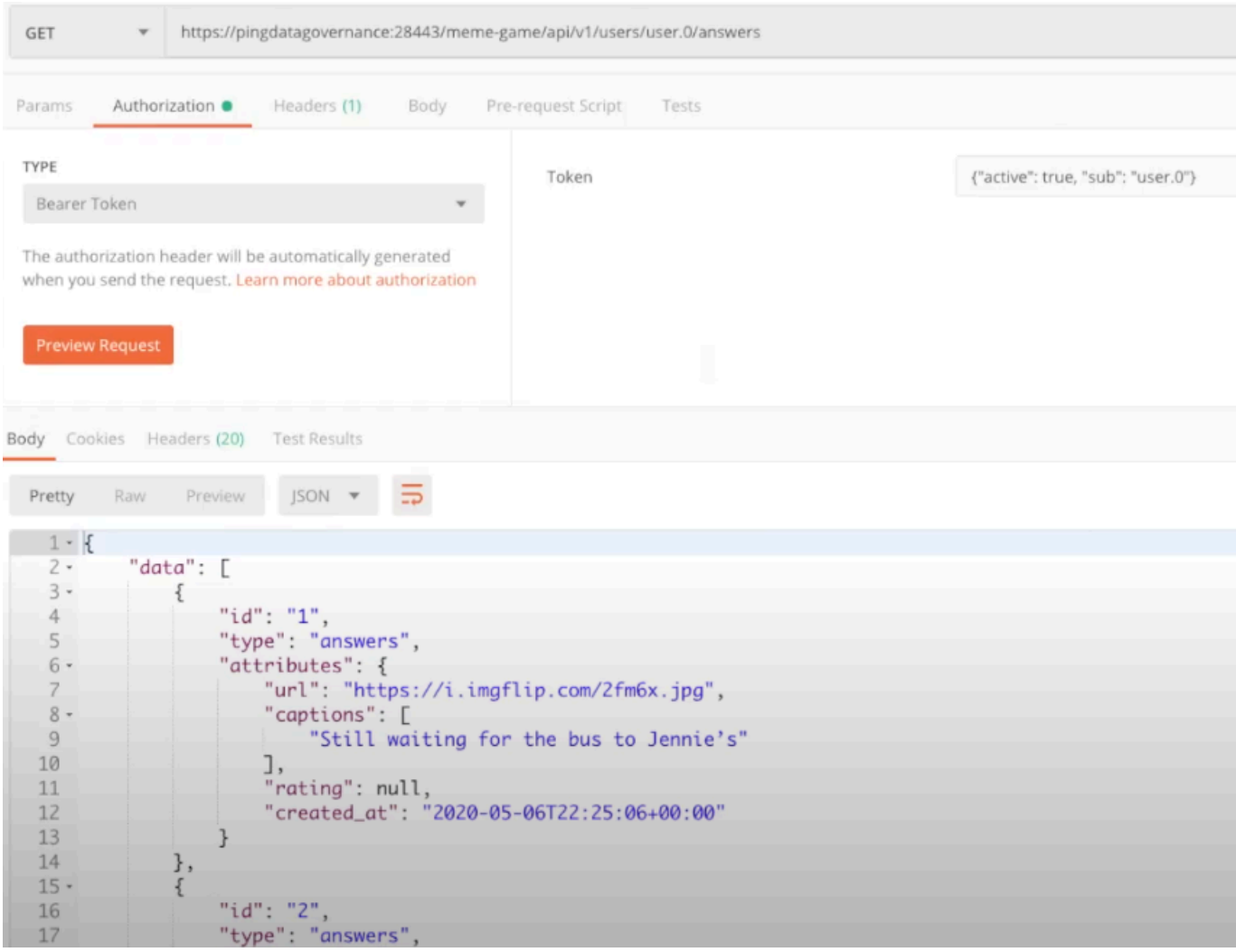
      The following image shows this configuration.



   g. Click **Save changes**.

9. Test the rule.

   The following image shows a test with Postman making a request to the user.0 answers as user.0. The response shows the rule works.

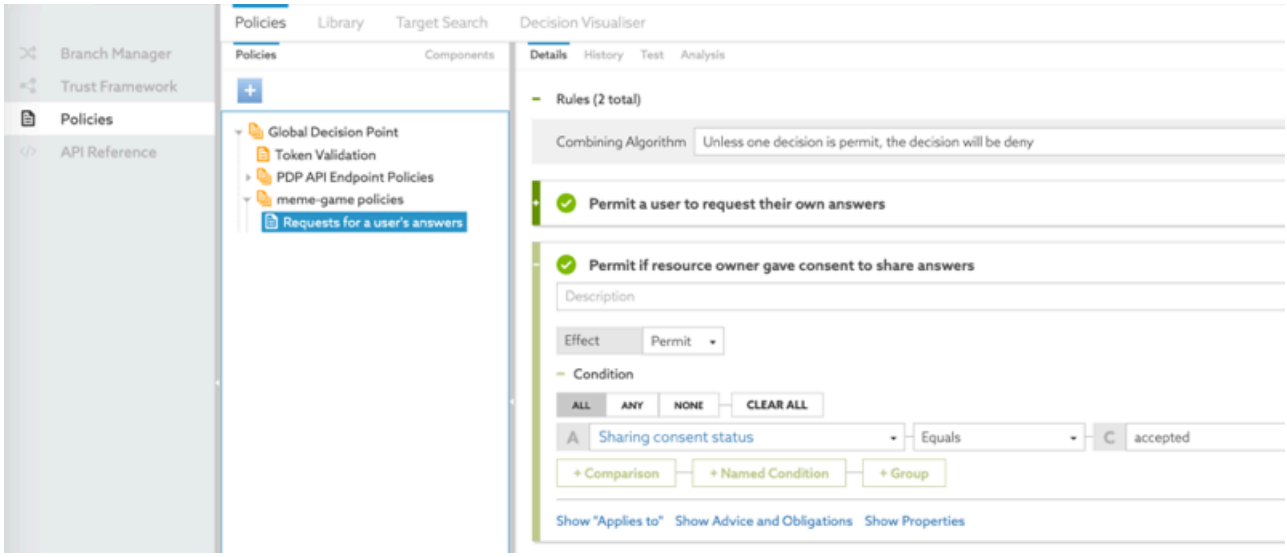


   If we try again with user.1, the request is denied. Even though user.1 does have a consent record in our consent store, the policy does not do anything with that consent record. We need another rule to look at the consent record and get the status from that record.

10. Add a rule to get status from a consent record.
    a. Click **+ Add Rule.**
    b. For the name, replace **Untitled** with `Permit if resource owner gave consent to share answers`.
    c. Click **+ Comparison**.
    d. From the **Select an Attribute** list, select **Sharing consent status**, which we set up in *Getting consent status from the consent record* on page 459.
    e. In the second field, select **Equals**.
    f. In the third field, type `accepted`.

       This value is the status to check against.

       The following image shows this rule.



    g. Click **Save changes**.
11. Test the policy with both rules in place now.

    A request to the user.0 answers as user.1 should now work.

    However, a request to the user.0 answers as a user without a consent record, say user.2, is denied.

    The user.2 request is denied because of the combining algorithm, **Unless one decision is permit, the decision will be deny**. When the policy engine evaluates the policy rules, the **Permit a user to request their own answers** rule does not produce a permit because user.2 is not requesting their own answers. The **Permit if resource owner gave consent to share answers** rule uses the `Sharing consent status` attribute. user.0 does not have a consent record for user.2. With no consent record to get status from, the policy engine cannot evaluate the rule. So this rule also does not produce a permit. Thus, the combining algorithm produces a deny for the user.2 request.

    If user.0 revokes the consent given to user.1, the status in the consent record becomes `revoked`. The rule no longer applies, so user.1 requests are then denied.

Use case: Using consent to change a response

PingAuthorize can change a server response based on the resource owner's consent to share.

This feature is useful for:

- Data control
- Information security
- Resource management

Again, we continue using the meme games API used in *Getting started with PingAuthorize (tutorials)* on page 17.

We first set up some Trust Framework attributes and services to provide consent status. Then we create a policy with rules that use the consent status to include, exclude, or modify attributes in the response. The following topics cover the Trust Framework tasks. If you completed *Use case: Using consent to determine access to a resource* on page 448, you have already finished the tasks of setting up Trust Framework attributes and services. Those tasks are the same for both use cases.

1. *Getting a path component from the request URL* on page 448
2. *Getting the requestor identifier from the access token* on page 453
3. *Searching for consent by resource owner to requestor* on page 453
4. *Getting consent status from the consent record* on page 459
5. What is different for this use case is the policy itself. The following topic explains how to add rules with advices to include, exclude, or modify attributes in the response.

   Creating a policy to check consent and then change the server response on page 464

Creating a policy to check consent and then change the server response
Using the Trust Framework attributes and services we created, we now create a policy for the meme game API to get a user's answers and change the server response with various advices based on the consent status.

About this task

Here is a snippet of an unedited response. It shows the `id`, `type`, and `attributes` attributes.

```
{
  "data": [{
                "id": "1",
                "type": "answers",
                "attributes": {
                        "url": "https: //l.imqflip.com/2fm6x.jpq",
                        "captions": ["Still waiting for the bus to
 Jennie's"],
                        "rating": null,
                        "created_at": "2020-05-e6T22:25:06-00:00"
                }
        },
```

Steps

1. Sign on to the Policy Editor, click **Policies** in the left pane and then click **Policies** along the top.
2. Select the existing **meme-game policies** policy. The new policy is created under this policy.
3. From the **+** menu, select **Add Policy**.
4. For the name, replace **Untitled** with `Control user's response to answers request`.
5. Click **+** next to **Applies to**.
6. Click **Add definitions and targets, or drag from Components** and add the **meme-game.user_answers** service, which we set up in *Getting a path component from the request URL* on page 448. Also, because we want to control the response to the client, add the **outbound-GET** action.
7. Set **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.

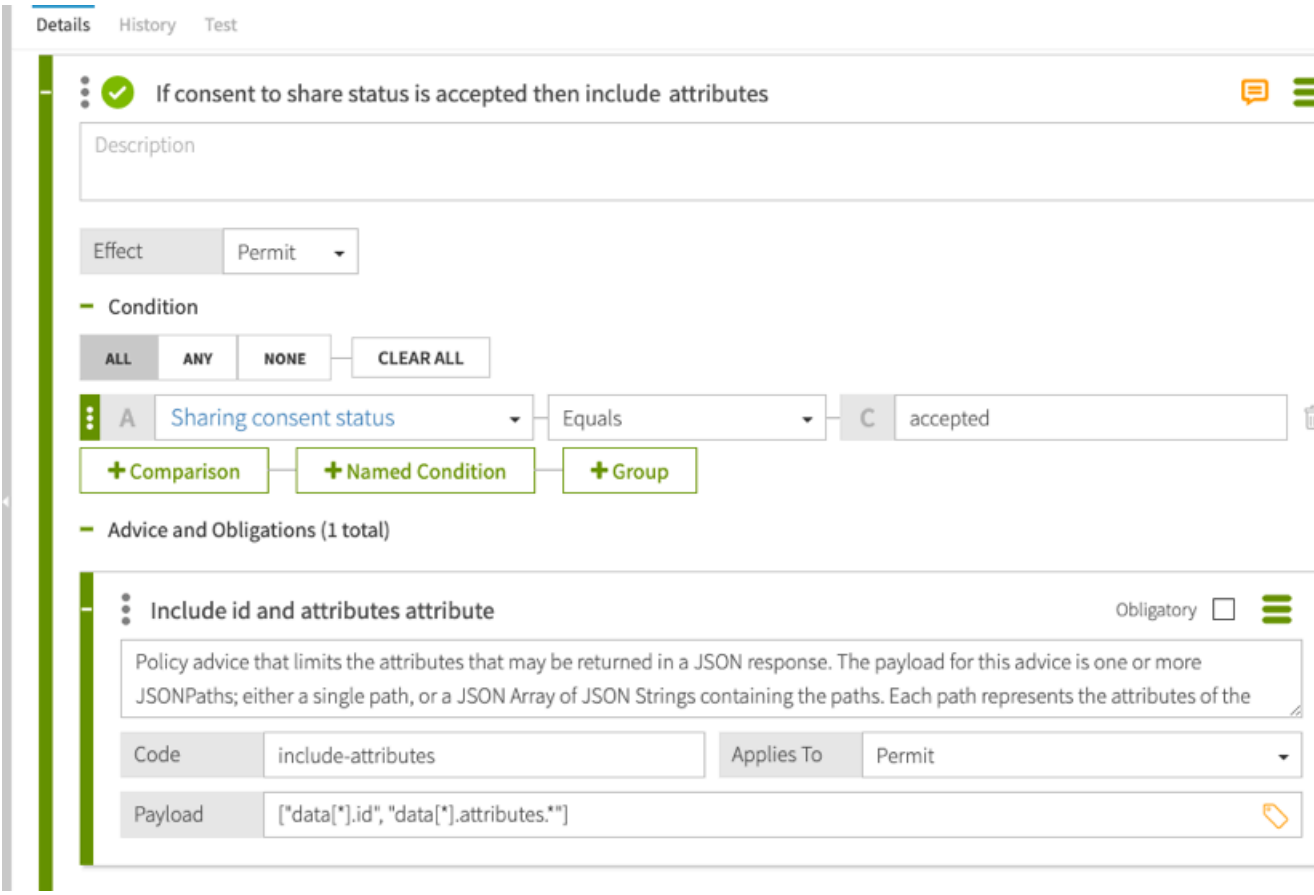
8. Add a rule to include attributes.
   a. Click **+ Add Rule**.
      1. For the name, replace **Untitled** with `If consent to share status is accepted then include attributes`.
      2. Specify the condition.
         a. Click **+ Comparison**.
         b. From the **Select an Attribute** list, select **Sharing consent status**, which we created in *Getting consent status from the consent record* on page 459.
         c. In the second field, select **Equals**.
         d. In the third field, type `accepted`.
      3. Specify the advice.
         a. Click **Show Advice and Obligations**.
         b. Click **+** next to **Advice and Obligations**.
         c. Click **+ Add Advice** # **Include Attributes**.

            Use this advice to be explicit about what attributes to keep, especially when you have a large set of attributes where you only need a small subset in the response.

            For information about this advice, see *Include Attributes* on page 495.
         d. For the name, replace **Untitled** with `Include id and attributes attribute`.
         e. In the **Code** field, enter `include-attributes`.
         f. From the **Applies To** list, select **Permit**.
         g. In the **Payload** field, enter the following text to include the `id` attribute and the `attributes` attribute but not the `type` attribute.

            ```
            ["data[*].id","data[*].attributes.*"]
            ```
         h. Click **Save changes**.

      The following screen shows the rule.



With the policy in place, trying the request again gets a response with the `type` attribute removed, as shown in the following snippet.

```
{
  "data": [{
                "attributes": {
                        "url": "https: //l.imqflip.com/2fm6x.jpq",
                        "captions": ["Still waiting for the bus to
 Jennie's"],
                        "rating": null,
                        "created_at": "2020-05-e6T22:31:06-00:00"
                "id": "1",
                }
        },
```

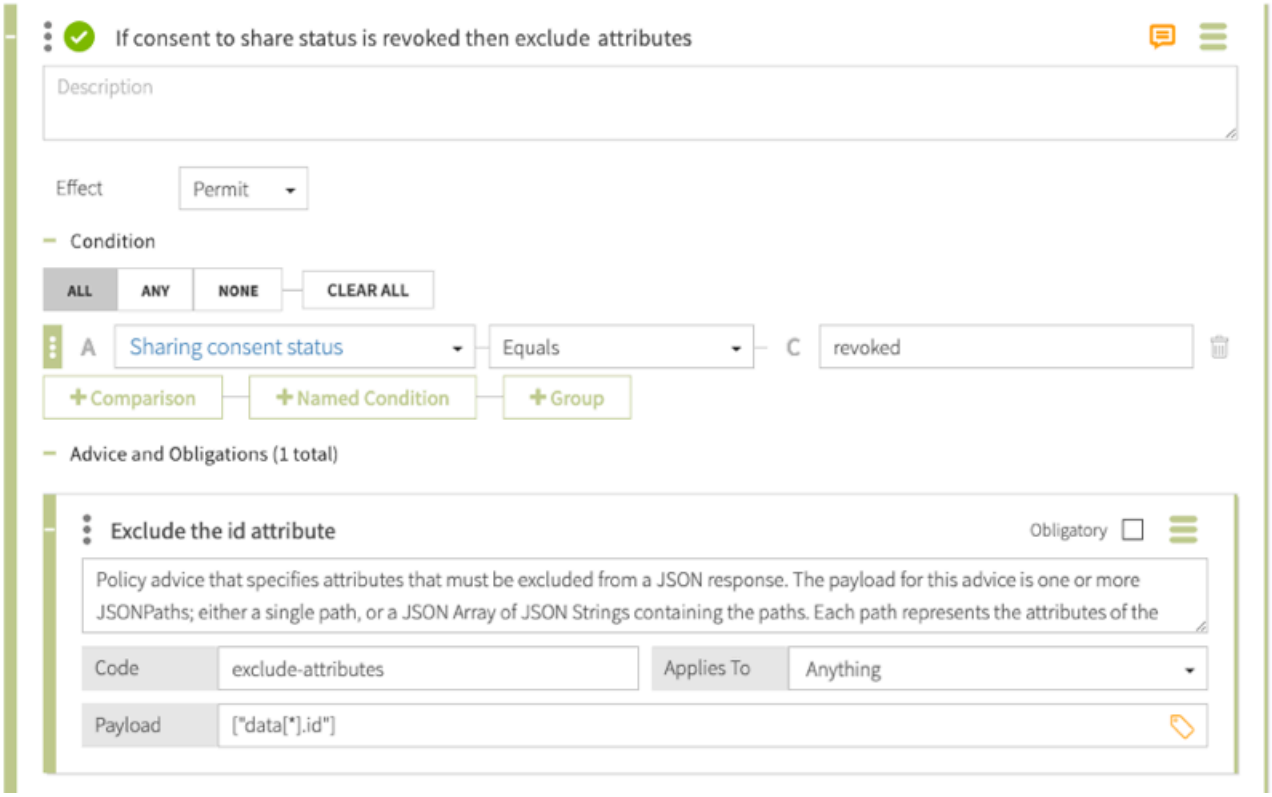
9. Add a rule to exclude attributes.
   a. Click **+ Add Rule**.
      1. For the name, replace **Untitled** with `If consent to share status is revoked then exclude attributes`.
      2. Specify the condition.
         a. Click **+ Comparison**.
         b. From the **Select an Attribute** list, select **Sharing consent status**, which we created in *Getting consent status from the consent record* on page 459.
         c. In the second field, select **Equals**.
         d. In the third field, type `revoked`.
      3. Specify the advice.
         a. Click **Show Advice and Obligations**.
         b. Click **+** next to **Advice and Obligations**.
         c. Click **+ Add Advice** # **Exclude Attributes**.

            Use this advice to be explicit about what attributes to leave out. For example, a third-party client might request banking records; the client does not need account numbers, so give them everything but the account number.

            For information about this advice, see *Exclude Attributes* on page 493.
         d. For the name, replace **Untitled** with `Exclude the id attribute`.
         e. In the **Code** field, enter `exclude-attributes`.
         f. From the **Applies To** list, select **Anything**.
         g. In the **Payload** field, enter the following text to exclude the `id` attribute.

            ```
            ["data[*].id"]
            ```
         h. Click **Save changes**.

   The following screen shows the rule.



With the policy in place, trying the request again gets a response with the `type` attribute removed, as shown in the following snippet.

```
{
  "data": [{
                "type": "answers",
                "attributes": {
                        "url": "https: //l.imqflip.com/2fm6x.jpq",
                        "captions": ["Still waiting for the bus to
 Jennie's"],
                        "rating": null,
                        "created_at": "2020-05-e6T22:35:06-00:00"
                }
        },
```

You can use the Decision Visualiser to see how the decision engine processed the decision. In the Policy Editor, click **Policies** in the left pane, then click **Decision Visualiser** along the top, and then click **Recent Decisions**. Click a decision and follow the green paths to see what polices are executed and which rules are invoked. Click **Attributes** along the top to see the names and values of attributes that are used in the decision.

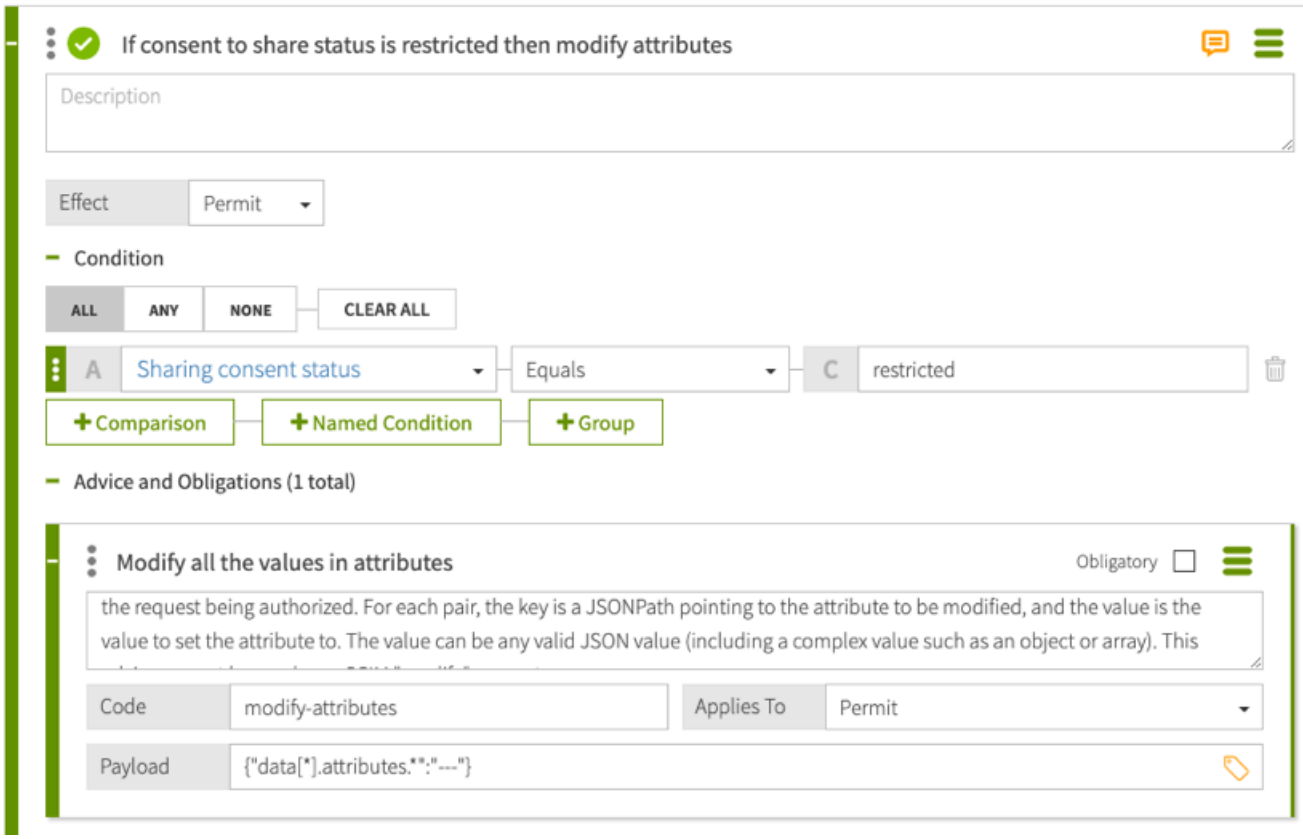
10. Add a rule to modify attributes.
    a. Click **+ Add Rule**.
        1. For the name, replace **Untitled** with `If consent to share status is restricted then modify attributes`.
        2. Specify the condition.
            a. Click **+ Comparison**.
            b. From the **Select an Attribute** list, select **Sharing consent status**, which we created in *Getting consent status from the consent record* on page 459.
            c. In the second field, select **Equals**.
            d. In the third field, type `restricted`.
        3. Specify the advice.
            a. Click **Show Advice and Obligations**.
            b. Click **+** next to **Advice and Obligations**.
            c. Click **+ Add Advice** # **Modify Attributes**.

               Use this advice to change attributes. For example, the client might request health records and require all items from a record, such as a social security number, even if partially or fully hidden.

               For information about this advice, see *Modify Attributes* on page 496.
            d. For the name, replace **Untitled** with `Modify all the values in attributes`.
            e. In the **Code** field, enter `modify-attributes`.
            f. From the **Applies To** list, select **Permit**.
            g. In the **Payload** field, enter the following text to replace all values in the `attributes` attribute with three dashes.

               ```
               {"data[*].attributes.*":"---"}
               ```
            h. Click **Save changes**.

    The following screen shows the rule.

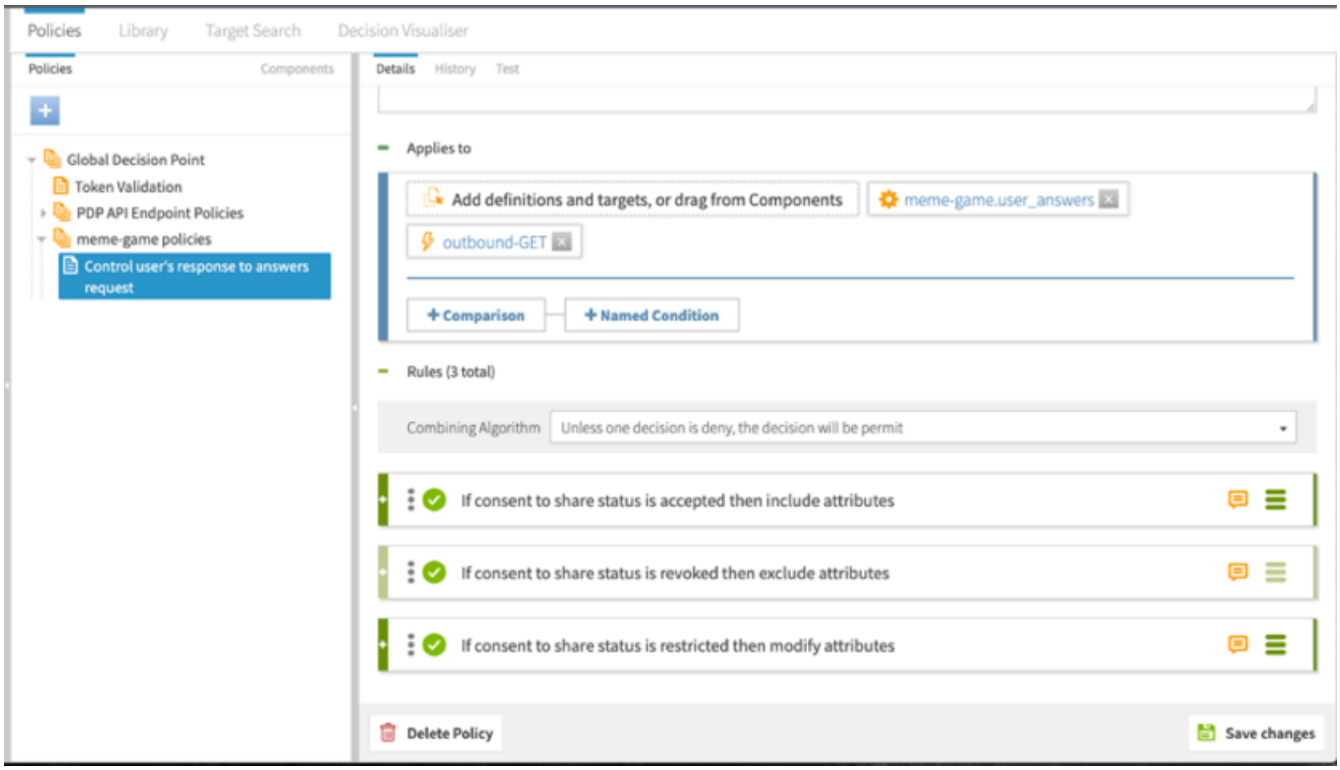


With the policy in place, trying the request now gets a response with the `id` and `type` attributes unchanged but all the `attributes` values changed to dashes, as shown in the following snippet.

```
{
  "data": [{
                "id": "168",
                "type": "answers",
                "attributes": {
                        "url": "---",
                        "captions": "---",
                        "rating": "---",
                        "created_at": "---"
                }
        },
```

Result

The following image shows the what policy applies to and the three rules.



Use case: Using a SCIM resource type or a policy request action to control behavior

SCIM (System for Cross-domain Identity Management) resource types define a class of resources, such as users or devices. The PingAuthorize Server SCIM service provides a REST API for data stored in external datastores that are based on the SCIM 2.0 standard.

The SCIM service translates each SCIM request or response into one or more policy requests to the policy decision point (PDP).

These policy requests have an `action` value that you can reference in the policies you write to deny or permit the action.

For more background information, see *About the SCIM service* on page 192.

For more information about actions, see *SCIM policy requests* on page 197.

This feature is useful for:

- Data control
- Information security
- Resource management

Example scenarios include:

- A bank that wants to prevent delete operations of their client profiles
- A health care system that should only allow the creation of new patient records and should not allow the modification of existing patient records
- A university system that only allows the retrieval of student information from the student's defined department; the system can modify the information differently based on the department

In this use case, we define services in the Trust Framework. We then create policies that use those services or policy request actions to control various operations. The following topics cover these tasks.

1. *Getting the SCIM resource type and the action being executed* on page 472
2. *Creating a policy to permit or deny the creation of resources* on page 474
3. *Creating a policy to control the set of actions for a specific resource* on page 477

4. *Creating a policy to restrict the ability to delete based on resource type* on page 481
5. *Creating a policy to dynamically modify a resource based on the SCIM resource type* on page 484

Getting the SCIM resource type and the action being executed

The SCIM resource type indicates the class of resources with which to interact. The action indicates what the user is trying to do. Here we define Trust Framework services to use in policies and locate the resource type and actions.

About this task

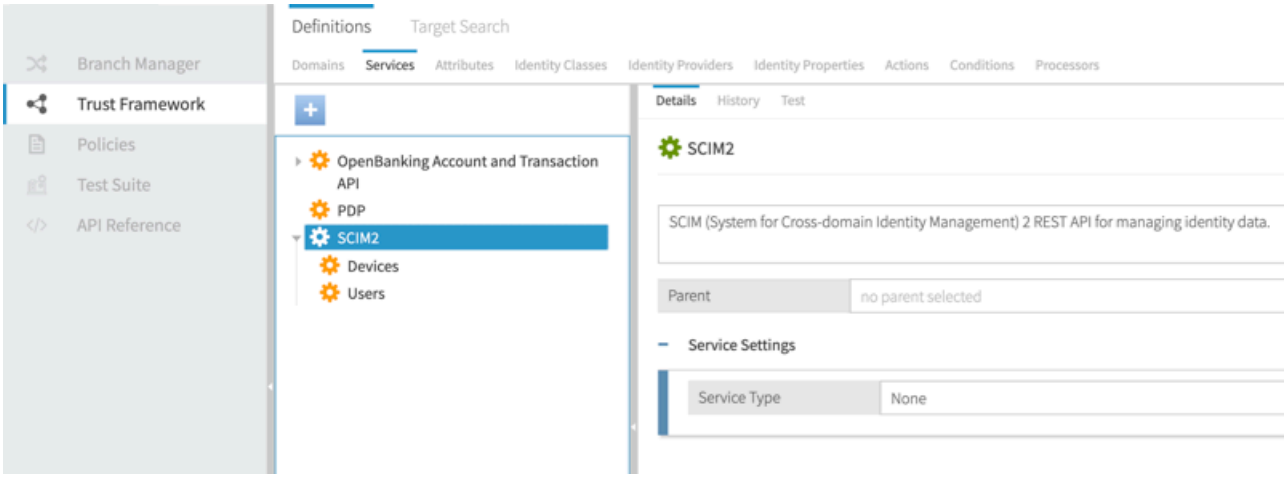
The PingAuthorize Policy Editor provides a SCIM2 service in the Trust Framework. This service is for the SCIM2 REST API and does not reference resource types. This task creates two services: Users and Devices.

Steps

1. Sign on to the Policy Editor.

2. Create the Users and Devices services.
   a. Go to **Trust Framework** and click **Services**.
   b. Click the **SCIM2** service so the service we create is listed under **SCIM2**.
   c. From the **+** menu, select **Add new Service**.
   d. For the name, replace **Untitled** with `Users`.
   e. Click **Save changes**.
   f. Click the **SCIM2** service again.
   g. From the **+** menu, select **Add new Service**.
   h. For the name, replace **Untitled** with `Devices`.
   i. Click **Save changes**.

   With the services defined, you should have a screen similar to the following one.



   We will use these services in the policies we create.

   Also, we will use the attribute `SCIM2.resource.meta.resourceType`.

   To see the attribute in the Trust Framework, click **Attributes** and navigate to it starting from **SCIM2**.

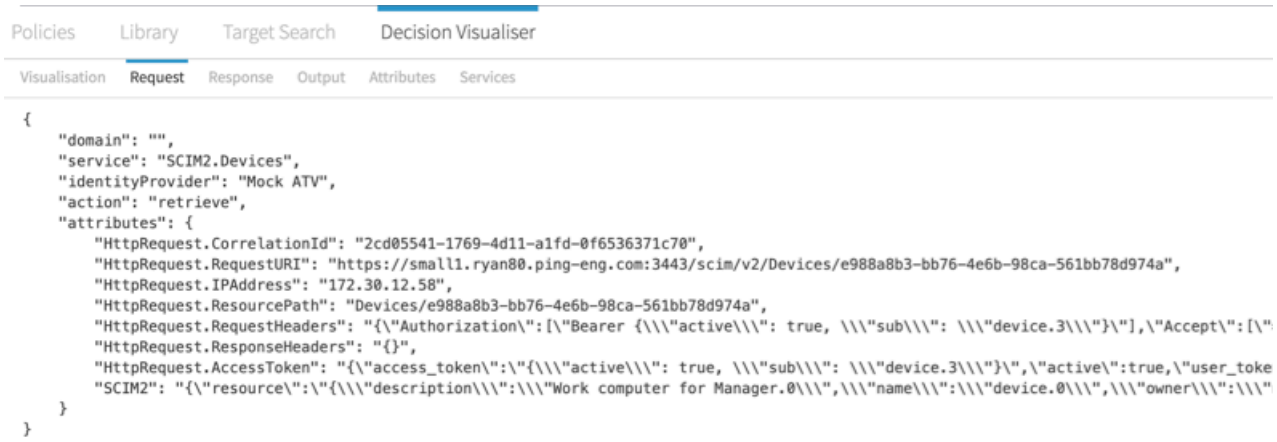
> **Note:**
> The `SCIM2.resource` attribute is only available when the SCIM resource exists. For example, the search and create actions do not have this attribute. However, the search action does have a policy request with a retrieve action that does have the attribute.

   Your policy can use a service you define or the `SCIM2.resource.meta.resourceType` attribute.

   Also, we can use these actions in our policies: create, delete, modify, retrieve, search, search-results.

   To see the actions in the Trust Framework, click **Actions**.

   When you are creating your policy, use the Policy Editor's Decision Visualiser to make sure your policy accurately reflects the policy requests. For example, consider the following screen showing the request.



We can use the following lines from the Decision Visualiser:

- `service` line

  Verify the name of the service in your Trust Framework and policy.
- `action` line

  Verify that the request produces the expected action that the policy uses.

  The PingAuthorize SCIM translates a get request in the SCIM REST API to retrieve action. For more information about actions, see *SCIM policy requests* on page 197.
- `RequestURI` line

  Verify that the endpoint belongs to the expected service.
- `SCIM2` line

  Scroll right to see the verify that the `resourceType` is as expected.

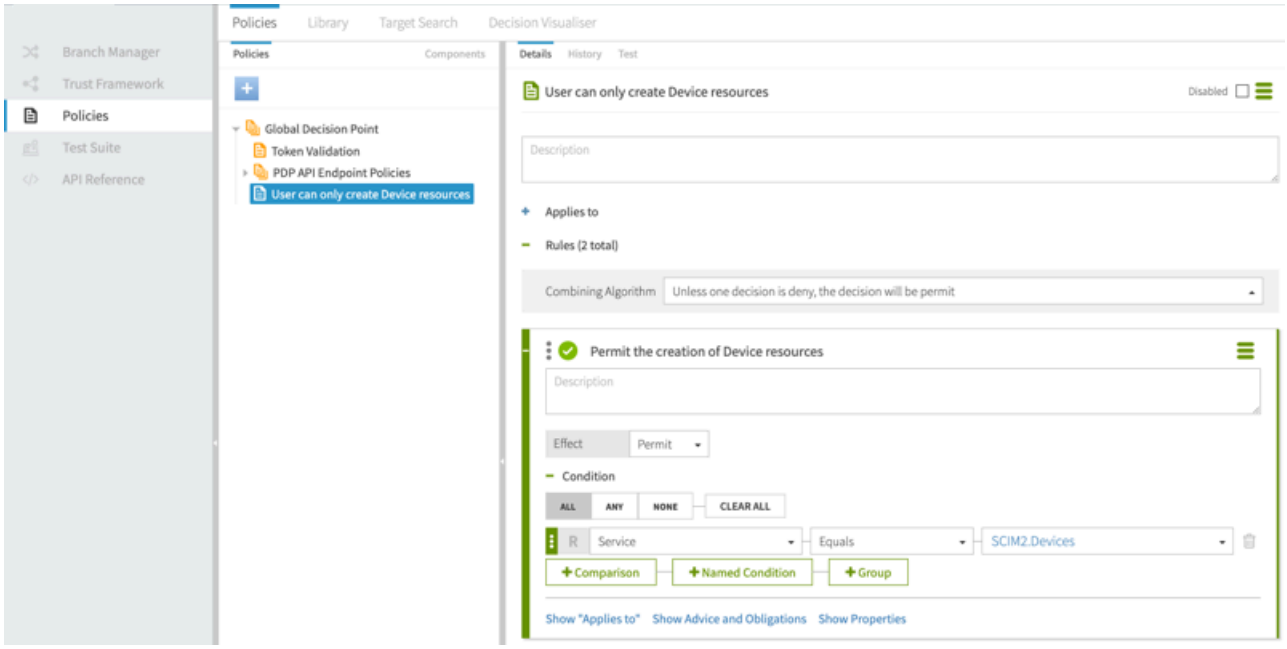
Creating a policy to permit or deny the creation of resources
This policy allows the creation of one resource type but not another. In particular, the policy focuses on the create action and then allows the creation of Device resources but denies the creation of User resources.

Steps

1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.
2. From the **+** menu, select **Add Policy**.
3. For the name, replace **Untitled** with `User can only create Device resources.`
4. Click the **+** next to **Applies to**.
5. Click **Add definitions and targets, or drag from Components** and add the **create** action.
6. Set **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.

7. Add a rule to allow the creation of Device resources.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `Permit the creation of Device resources.`
   c. Click **+ Comparison**.
   d. In the first field, click the **A** to toggle to an **R** and from that field's drop-down list, select **Service**.
   e. In the second field, select **Equals**.
   f. In the third field, select the **SCIM2.Devices** service.
   g. Click **Save changes**.

   You should have a screen similar to the following one for the policy and this rule.



8. Add a rule to deny the creation of User resources.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `Deny the creation of User resources.`
   c. Set **Effect** to **Deny**.
   d. Click **+ Comparison**.
   e. In the first field, click the **A** to toggle to an **R** and from that field's drop-down list, select **Service**.
   f. In the second field, select **Equals**.
   g. In the third field, select the **SCIM2.Users** service.
   h. Add advice to provide a custom message.
      1. Within the rule, click **Show Advice and Obligations**.
      2. Click **+** next to **Advice and Obligations**.
      3. Click **+ Add Advice** # **Denied Reason**.
      4. For the name, specify `denied-reason`.
      5. Set **Applies To** to **Deny**.
      6. In the **Payload** field:
         - Remove

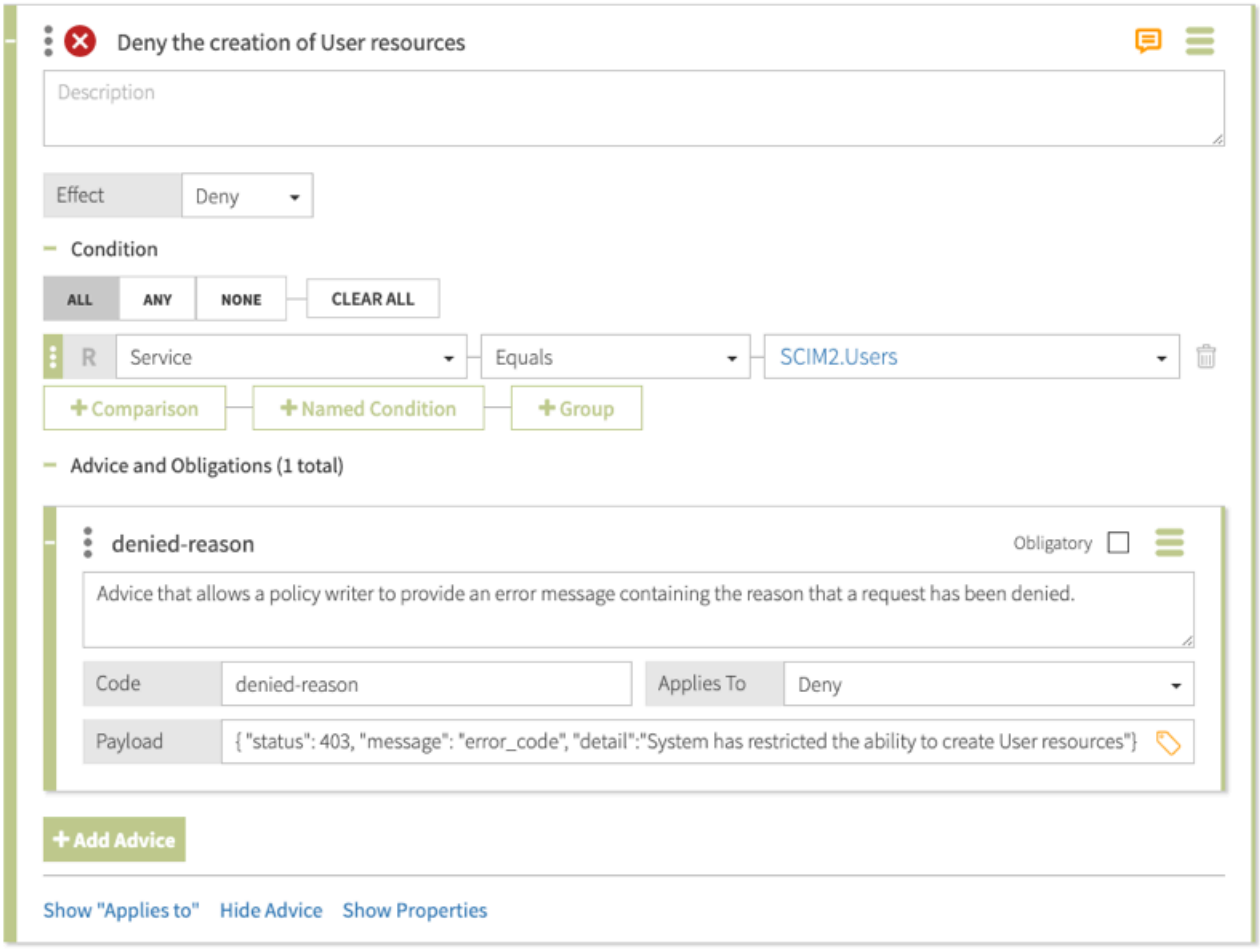
           `Example:`
         - Change

           `Human-readable error message`

           to

           `System has restricted the ability to create User resources`
   i. Click **Save changes**.

   You should have a screen similar to the following one for the second rule.



9. Send test requests to the SCIM service and verify data using the Policy Editor's Decision Visualiser.

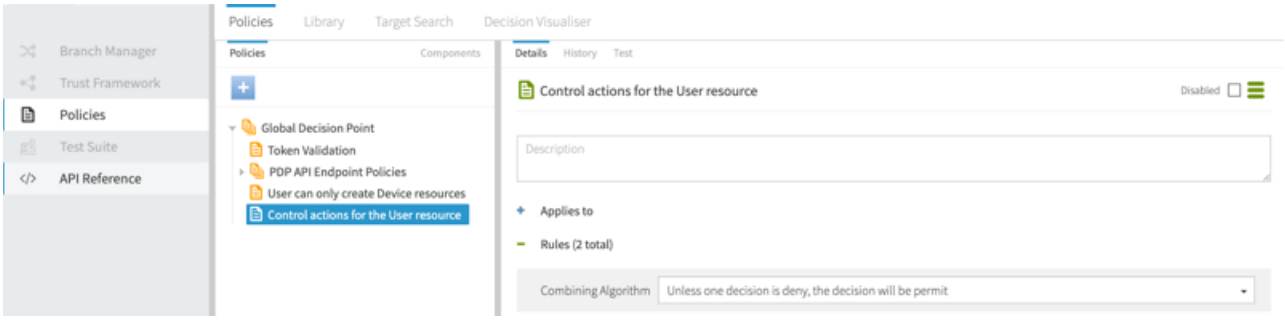
Creating a policy to control the set of actions for a specific resource
For a given resource, control the outcomes (deny or permit) of actions on the resource. In particular, the policy focuses on the Users resource, and then denies deletes but permits retrieves.

Steps

1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.
2. From the **+** menu, select **Add Policy**.
3. For the name, replace **Untitled** with `Control actions for the User resource`.
4. Click the **+** next to **Applies to**.
5. Click **Add definitions and targets, or drag from Components** and add the **SCIM2.Users** service.

6. Set **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.

   You should have a screen similar to the following one for the policy so far.



7. Add a rule to deny the deletion of User resources.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `Action: delete`.
   c. Set **Effect** to **Deny**.
   d. Click **+ Comparison**.
   e. In the first field, click the **A** to toggle to an **R** and from that field's drop-down list, select **Action**.
   f. In the second field, select **Equals**.
   g. In the third field, select the **delete** action.
   h. Add advice to provide a custom message.
      1. Within the rule, click **Show Advice and Obligations**.
      2. Click **+** next to **Advice and Obligations**.
      3. Click **+ Add Advice** # **Denied Reason**.
      4. For the name, specify `denied-reason`.
      5. Set **Applies To** to **Deny**.
      6. In the **Payload** field:
         - Remove

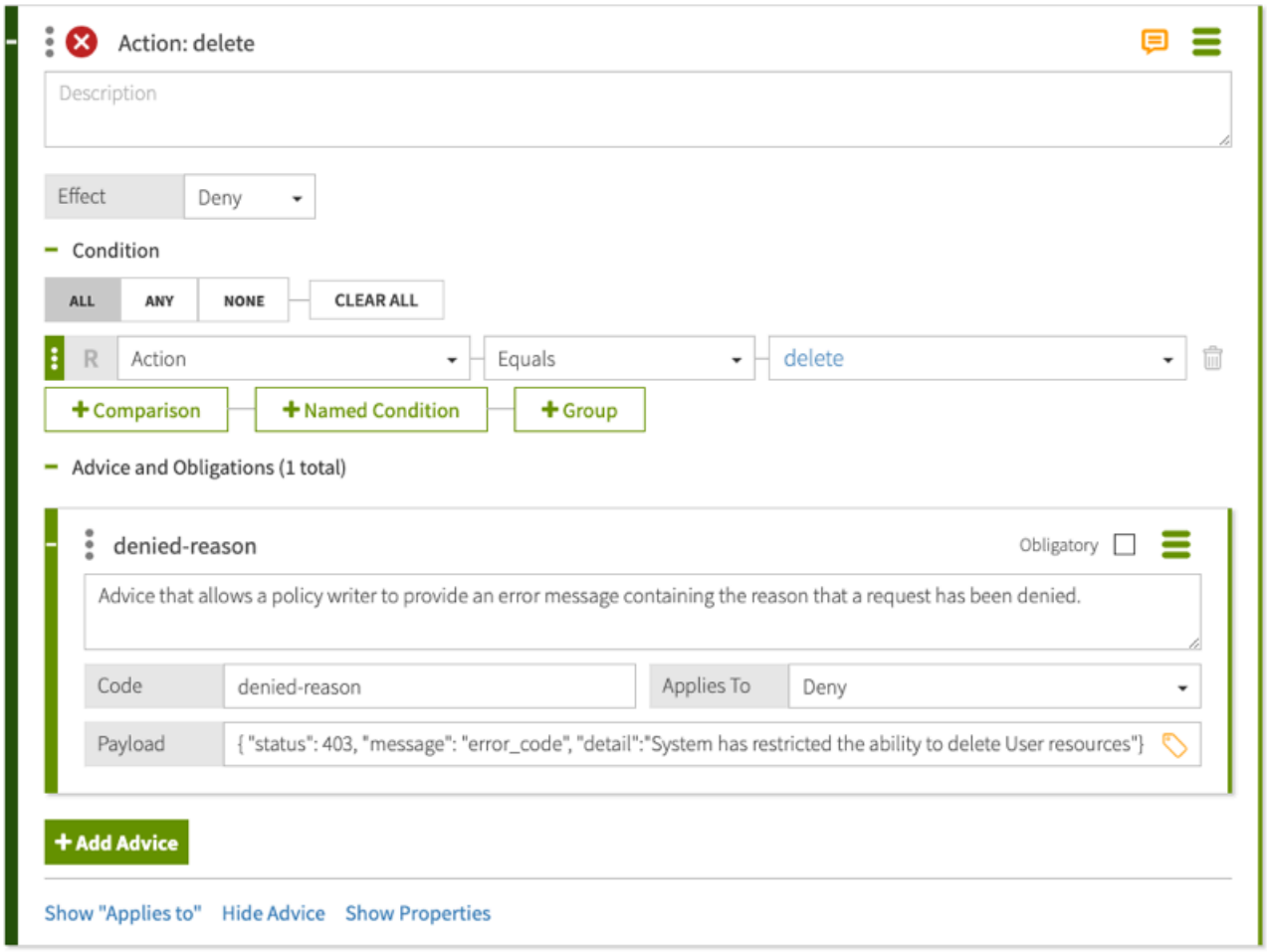
           `Example:`
         - Change

           `Human-readable error message`

           to

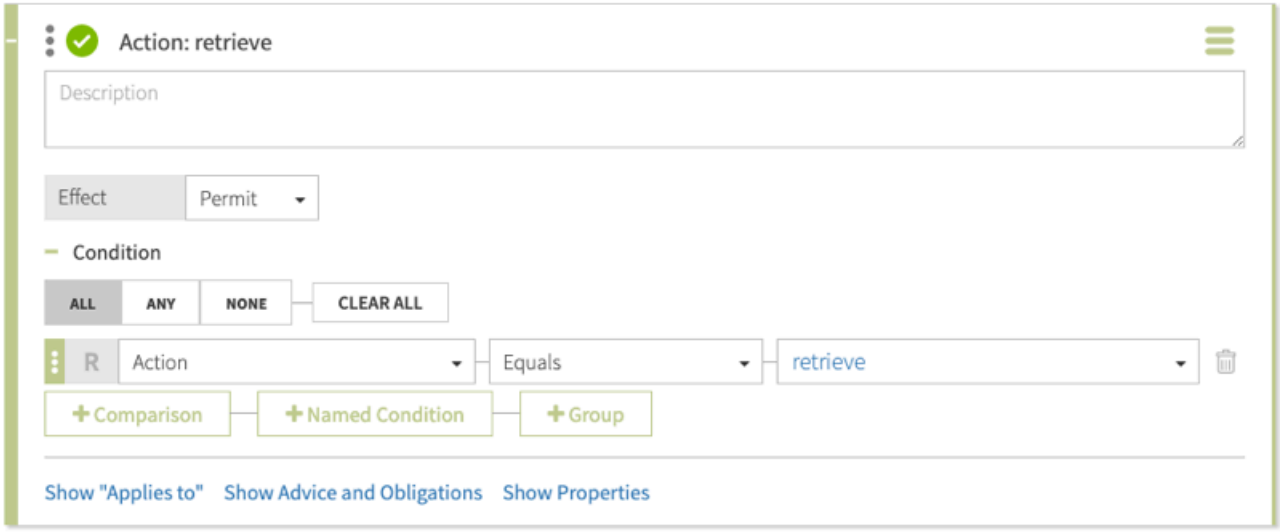
           `System has restricted the ability to delete User resources`
   i. Click **Save changes**.

      Your rule should be similar to the following one.



8. Add a rule to permit the retrieval of User resources.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `Action: retrieve`.
   c. Click **+ Comparison**.
   d. In the first field, click the **A** to toggle to an **R** and from that field's drop-down list, select **Action**.
   e. In the second field, select **Equals**.
   f. In the third field, select the **retrieve** action.
   g. Click **Save changes**.

   Your rule should be similar to the following one.



9. Send test requests to the SCIM service and verify data using the Policy Editor's Decision Visualiser.

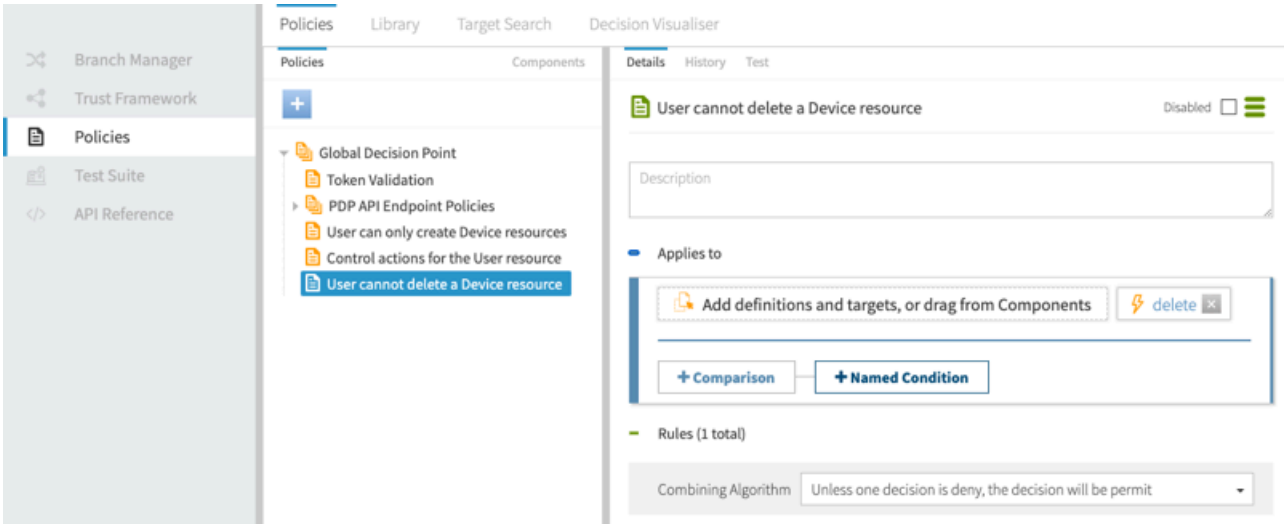
Creating a policy to restrict the ability to delete based on resource type
For a given resource type, restrict the ability to delete. In particular, the policy focuses on the delete action and then denies the action when the resource type is Devices.

Steps

1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.
2. From the **+** menu, select **Add Policy**.
3. For the name, replace **Untitled** with `User cannot delete a Device resource`.
4. Click the **+** next to **Applies to**.
5. Click **Add definitions and targets, or drag from Components** and add the **delete** action.

6. Set **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.

   You should have a screen similar to the following one for the policy so far.



7. Add a rule to deny the deletion of Device resources.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `If the SCIM resource type is Device, then deny`.
   c. Set **Effect** to **Deny**.
   d. Click **+ Comparison**.
   e. In the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
   f. In the second field, select **Equals**.
   g. In the third field, specify `Devices` as the constant.
   h. Add advice to provide a custom message.
      1. Within the rule, click **Show Advice and Obligations**.
      2. Click **+** next to **Advice and Obligations**.
      3. Click **+ Add Advice** # **Denied Reason**.
      4. For the name, specify `denied-reason`.
      5. Set **Applies To** to **Deny**.
      6. In the **Payload** field:
         - Remove

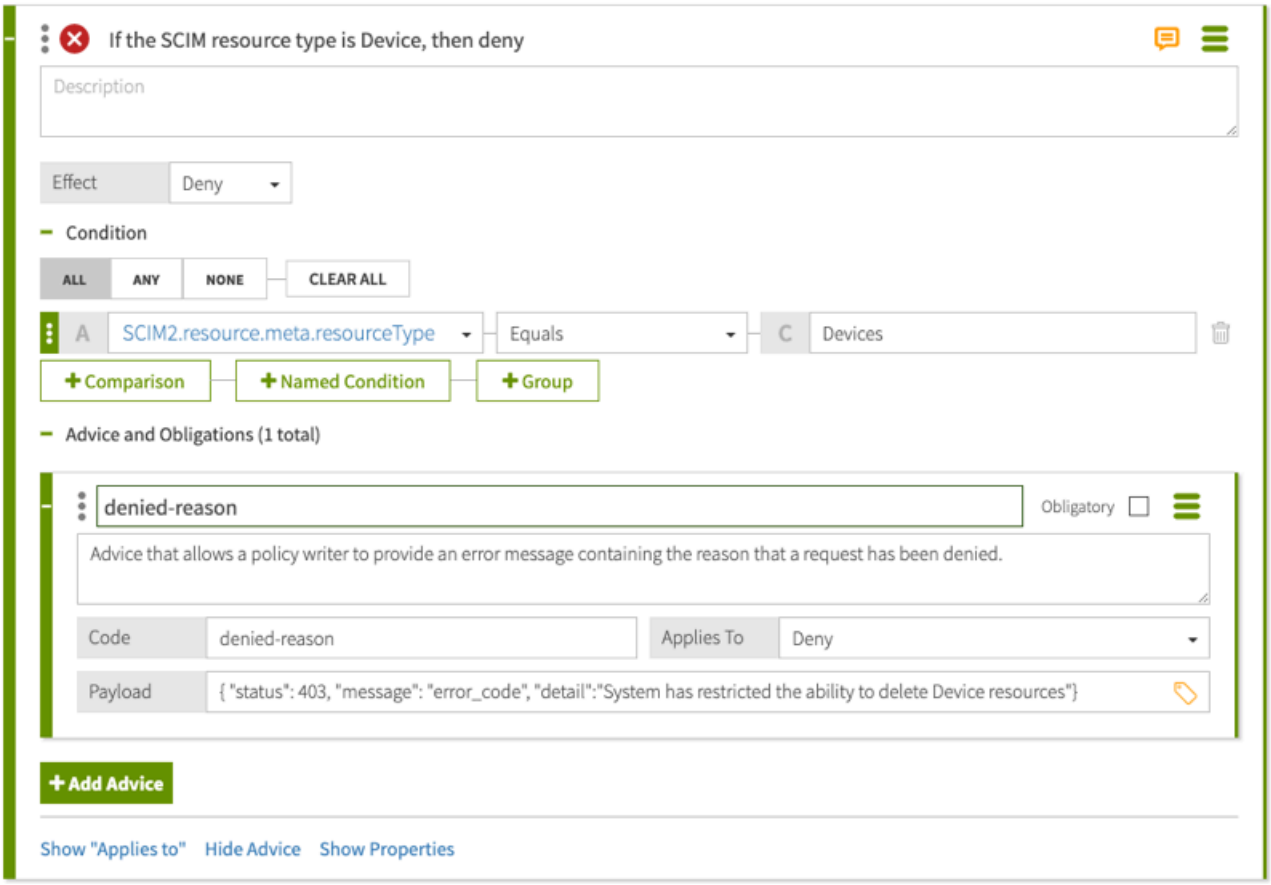
           `Example:`
         - Change

           `Human-readable error message`

           to

           `System has restricted the ability to delete Device resources`
   i. Click **Save changes**.

      Your rule should be similar to the following one.



8. Send test requests to the SCIM service and verify data using the Policy Editor's Decision Visualiser.

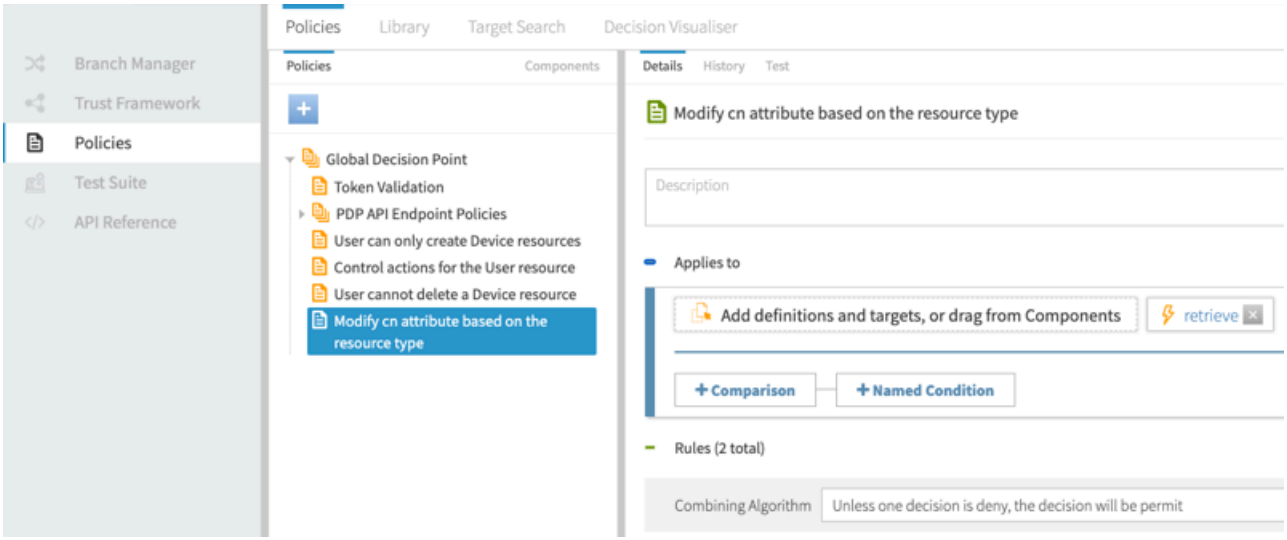
Creating a policy to dynamically modify a resource based on the SCIM resource type
Given an attribute defined in multiple resource types, modify the attribute differently depending on the resource type. In particular, this policy focuses on the retrieve action and changes the `cn` attribute to one value for the Users resource type and to another value for the Devices resource type.

Steps

1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.
2. From the **+** menu, select **Add Policy**.
3. For the name, replace **Untitled** with `Modify cn attribute based on the resource type`.
4. Click the **+** next to **Applies to**.
5. Click **Add definitions and targets, or drag from Components** and add the **retrieve** action.

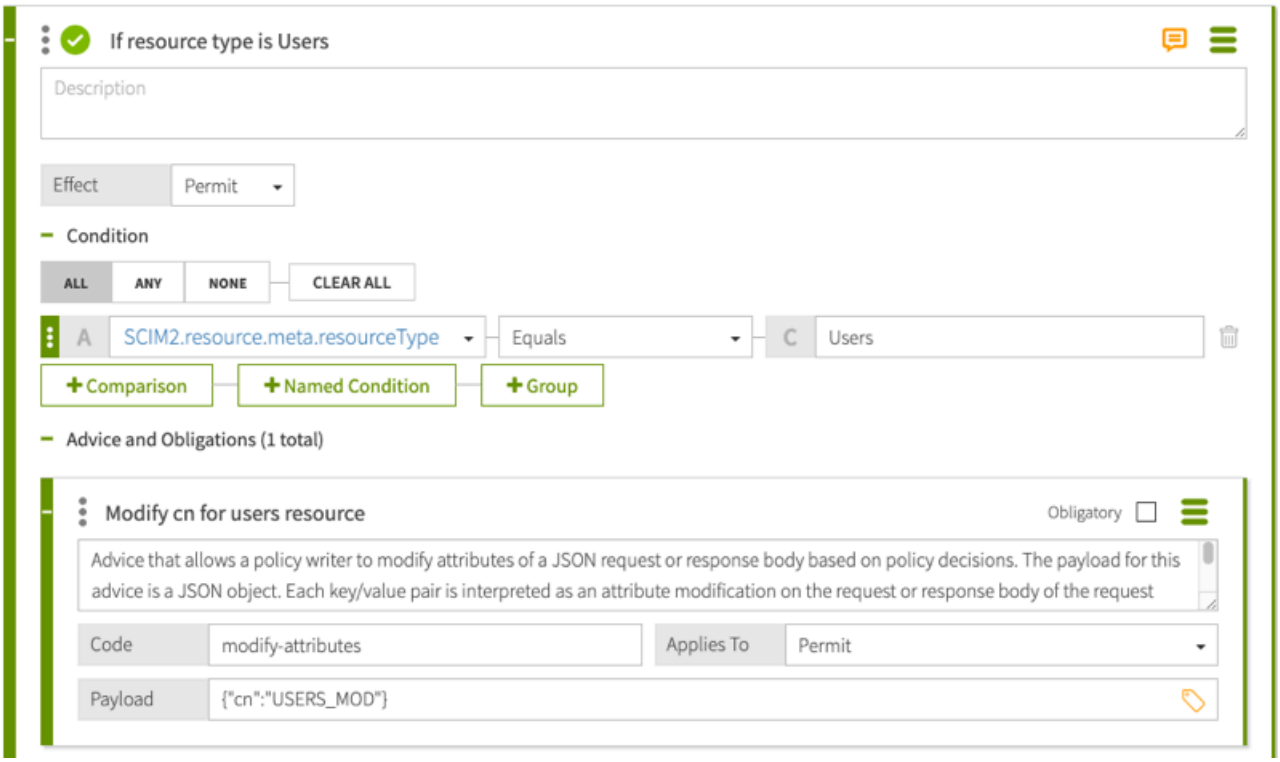
6. Set **Combining Algorithm** to **Unless one decision is deny, the decision will be permit**.

   You should have a screen similar to the following one for the policy so far.



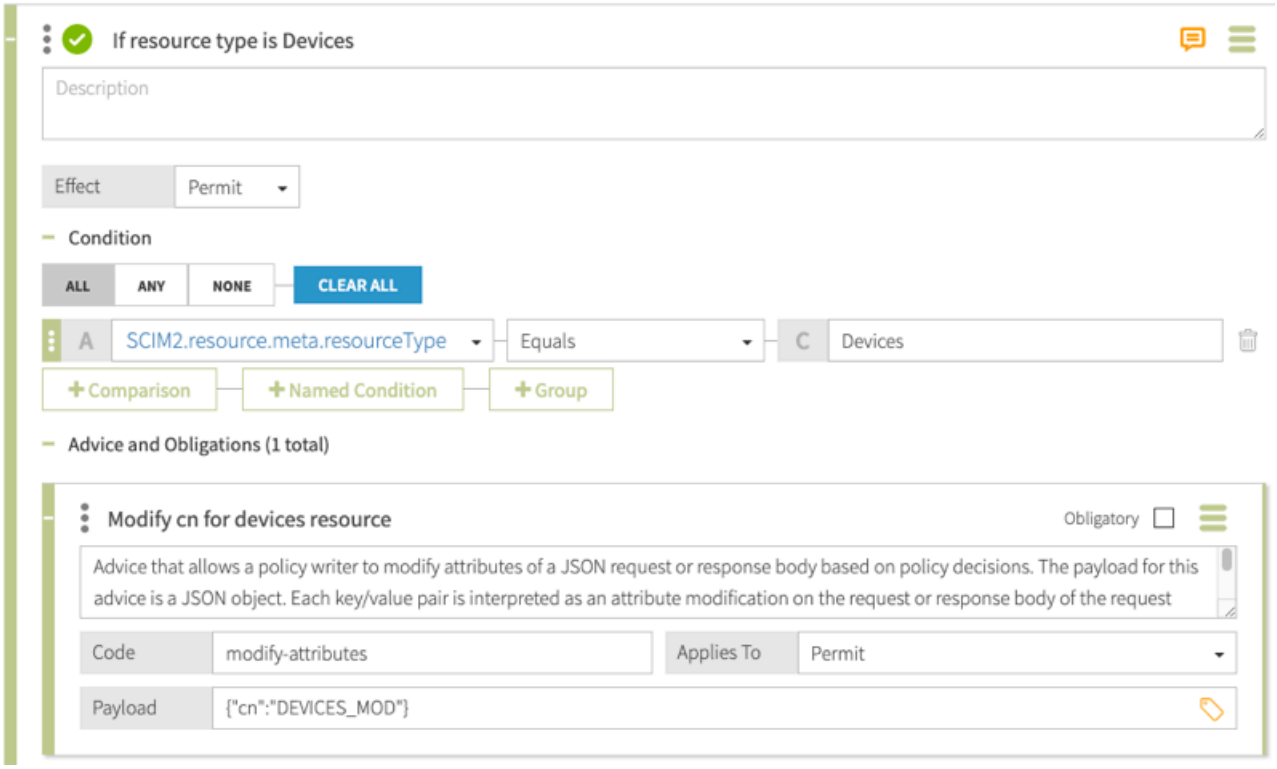
7. Add a rule for the Users resource.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `If resource type is Users`.
   c. Click **+ Comparison**.
   d. From the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
   e. In the second field, select **Equals**.
   f. In the third field, specify `Users` as the constant.
   g. Add advice to modify attributes.
      1. Within the rule, click **Show Advice and Obligations**.
      2. Click **+** next to **Advice and Obligations**.
      3. Click **+ Add Advice** # **Modify Attributes**.
      4. For the name, specify `Modify cn for users resource`.
      5. Set **Applies To** to **Permit**.
      6. Set the **Payload** field to `{"cn":"USERS_MOD"}`.
   h. Click **Save changes**.

      Your rule should be similar to the following one.



8. Add a rule for the Devices resource.
   a. Click **+ Add Rule.**
   b. For the name, replace **Untitled** with `If resource type is Devices`.
   c. Click **+ Comparison**.
   d. From the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
   e. In the second field, select **Equals**.
   f. In the third field, specify `Devices` as the constant.
   g. Add advice to modify attributes.
      1. Within the rule, click **Show Advice and Obligations**.
      2. Click **+** next to **Advice and Obligations**.
      3. Click **+ Add Advice** # **Modify Attributes**.
      4. For the name, specify `Modify cn for devices resource`.
      5. Set **Applies To** to **Permit**.
      6. Set the **Payload** field to `{"cn":"DEVICES_MOD"}`.
   h. Click **Save changes**.

   Your rule should be similar to the following one.



9. Send test requests to the SCIM service and verify data using the Policy Editor's Decision Visualiser.

Restricting the modification of attributes

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. For the name, 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 by administrators
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.

Adding attributes to an allow list
To allow the user to modify a set of attributes limited to an allow list and return an error if the user attempts to modify any attribute outside of the allow list, create a constant in the Trust Framework and then use the constant in a policy.

Steps

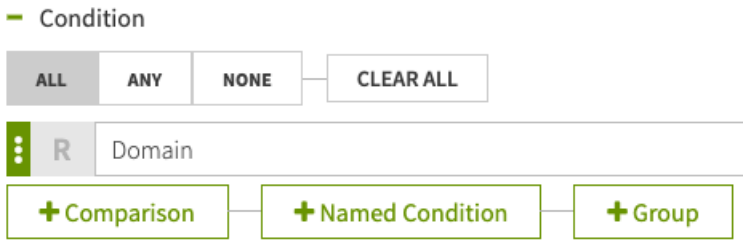
1. Create a constant in the Trust Framework.
   a. Go to **Trust Framework** and then **Attributes**.
   b. From the **+** menu, select **Add new Attribute**.
   c. For the name, replace **Untitled** with `allowlistAttributes`.
   d. Verify that in the **Parent** field, no parent is selected. To remove 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. For the name, 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 `allowlistAttributes` 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 modification of attributes* on page 487.

      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 `allowlistAttributes`.

Test Suite

Use the Test Suite to define tests, scenarios, and assertions to validate behavior for most Trust Framework and Policy Manager entities.

Policy writers can build a library of test cases to use as part of a test-driven development approach to policy and Trust Framework design. The library you develop can form a suite of regression checks that you run against each new version of policies or the Trust Framework.

The Test Suite has these components: Tests, scenarios, and assertions. The following table highlights the similarities and differences. The components are very similar. However, with test cases, you specify a Trust Framework or Policy Manager entity to test. Scenarios do not use such entities and are instead for reuse across tests.

Test cases	Scenarios
A test case definition includes: ▪ A decision request ▪ Optional overrides for attributes ▪ Optional overrides for services ▪ An entity to test ▪ Optional assertions	A scenario definition includes: ▪ A decision request ▪ Optional overrides for attributes ▪ Optional overrides for services You can reuse a scenario within a test suite.

Tests

In the Test Suite, use the **Tests** tab to view and manage tests and test groups. A test group is a collection of tests.

To add a test or test group, click **+**.

When you create a test, keep the following items in mind:

Field	Description
Name	A unique name avoiding the following characters: `{ } | .`
Description	A description for the test to clarify its intention and usage.
Tested Entity	The entity to verify with the test. After you assign an entity, you can run the test on that entity using the **Test** tab in the Trust Framework or Policy Manager pages.
Scenario Type	The type of scenario to use: ▪ Inline—you define the scenario on the same page where you define the test ▪ Referenced—you select a scenario that you already defined in the **Scenarios** tab **Tip:** You can use a referenced scenario as a template for a new inline scenario by selecting that referenced scenario and then switching to **Inline Scenario**.

When you create a test group, you need only provide a name and description.

Scenarios

In the Test Suite, use the **Scenarios** tab to view and manage scenarios and scenario groups. A scenario group is a collection of scenarios.

Scenarios define a decision request and optional attribute and service overrides to serve as input for a test. After you define a scenario, you can reference it by name in your tests. Also, on the **Test** tab in the Trust Framework or Policy Manager pages, you can load a scenario directly into the test by clicking the **Load Scenario** button in the lower, right corner.

To add a test or test group, click **+**.

You can specify request and override data by hand or by importing it in JSON format by clicking the **Import JSON** button in the lower, right corner.

When you create a scenario group, you need only provide a name and description.

Assertions

After you define a test scenario, you can create assertions to verify content in the decision response generated by the scenario. Use assertions to ensure that a particular property in the response is behaving correctly.

In the Test Suite, use the **Assertions** tab to view and manage assertions and assertion groups.

To create an assertion, your options include:

- Using the **Assertions** tab.

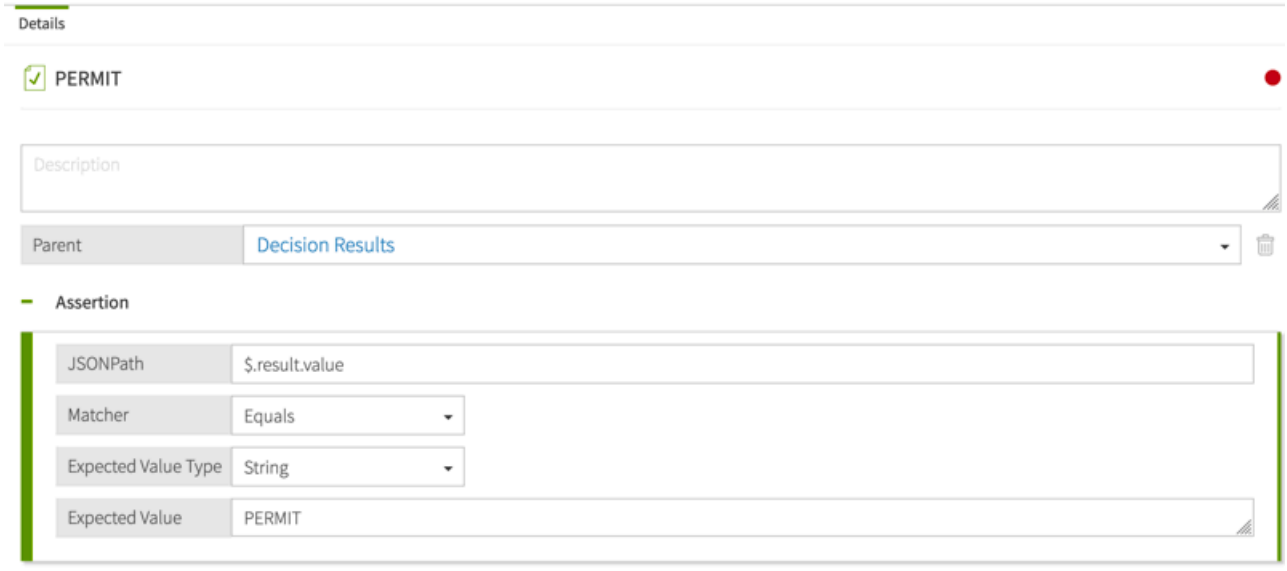
> **Tip:**
>
> For assertions you create using the **Assertions** tab, use them in a test by clicking **+ Add Assertion**, setting **Assertion Type** to **Referenced**, and then selecting the assertion in the drop-down list.

- Creating them inline when you define a test on the **Tests** tab.

When you define an assertion, you:

1. Provide a JSONPath accessor to extract information from the response.
2. Specify a matcher to indicate how to compare the extracted information against an expected value.
3. Specify the expected value type.
4. Specify the expected value.

The following image shows an assertion that checks whether result value equals PERMIT:



Test execution

After you assign a testable entity, such as a policy or attribute, to a test case, you can run the test. To view and run the test, your options are:

- In the test definition, after you add the tested entity to the test and save changes, click the name of the tested entity to view the entity. Next, click **Test** and then **Tests**.
- View the entity through a tab on the left, such as the **Policies** tab. Next, click **Test** and then **Tests**.

You see a table of the tests available for the entity. Click a test's **Execute** button to run that test. For longer running tests, you can go to other tasks in the Policy Editor and return to this page later to check progress.

If a test uses assertions, when you expand the row for the test case, an **Assertions** tab appears. Use this tab to see the results for the assertions.

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.

Advices allow the policy enforcement point—PingAuthorize Server, in this case—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, as well as modify that advice as part of a policy definition. You can also add an advice in **Components** for use with multiple policies or rules.

This section describes the advice types built into PingAuthorize Server.

Add Filter

Use `add-filter` to add administrator-required filters to System for Cross-domain Identity Management (SCIM) search queries.

Description	Details
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.

Description	Details
Applicable to	SCIM.

Description	Details
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 202.

Denied Reason

Use `denied-reason` to allow a policy writer to provide an error message that contains the reason for denying a request.

Description	Details
Applicable to	DENY decisions. **Note:** The `denied-reason` advice only applies to SCIM searches using the *optimized search response authorization* mode.
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.

Description	Details
Applicable to	PERMIT decisions, although you cannot apply Exclude Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search. Also, do not use this advice type with SCIM modifies. Instead, use the *Modify SCIM Patch* on page 497 advice type.

Description	Details
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 PingAuthorize 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 494.

Filter Response

Use `filter-response` to direct PingAuthorize Server to invoke policy iteratively over each item of a JSON array contained within an API response.

Description	Details
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.

<table>
<tr><th>Description</th><th>Details</th></tr>
<tr><td>Additional information</td><td>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.

<table>
<tr><th>Field</th><th>Required</th><th>Description</th></tr>
<tr><td><code>Path</code></td><td>Yes</td><td>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.</td></tr>
<tr><td><code>Action</code></td><td>No</td><td>Value to pass as the <code>action</code> parameter on subsequent policy requests. If no value is specified, the action from the parent policy request is used.</td></tr>
<tr><td><code>Service</code></td><td>No</td><td>Value to pass as the <code>service</code> parameter on subsequent policy requests. If no value is specified, the service value from the parent policy request is used.</td></tr>
<tr><td><code>ResourceType</code></td><td>No</td><td>Type of object contained by each JSON node in the array, selected by the <code>Path</code> field. On each subsequent policy request, the contents of a single array element 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.</td></tr>
</table>
On each policy request, if policy returns a <code>deny</code> decision, the relevant array node is removed from the response. If the policy request returns a <code>permit</code> 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.</td></tr>
</table>

Include Attributes

Use `include-attributes` to limit the attributes that a JSON response can return.

Description	Details
Applicable to	PERMIT decisions, although you cannot apply Include Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search. Also, do not use this advice type with SCIM modifies. Instead, use the *Modify SCIM Patch* on page 497 advice type.

Description	Details
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 494.

Modify Attributes

Use `modify-attributes` to modify the values of attributes in the JSON request or response.

Description	Details
Applicable to	All, although you cannot apply the Modify Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search. Also, do not use this advice type with SCIM modifies. Instead, use the *Modify SCIM Patch* on page 497 advice type.
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 PingAuthorize sends them to the upstream server or to modify the values of response headers before PingAuthorize returns them to the client.

Description	Details
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, PingAuthorize overwrites it. If the header does not exist, PingAuthorize 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`, PingAuthorize repeats the header for each string in the array. ▪ Given a header that does not support multiple values, such as `Content-Type`, PingAuthorize 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.

Description	Details
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`.

Modify SCIM Patch

Use `modify-scim-patch` to add operations to a SCIM patch in a modify request before it is submitted to the store adapter.

Description	Details
Applicable to	SCIM requests with an action of modify.

Description	Details

Additional information	The payload for this advice is either a JSON array or a JSON object.
	
	If the payload is an array, PingAuthorize treats it as a list of operations in the SCIM patch format to add to the end of the operations in the patch. For example, assume the modify has the following patch.
	
	`{ "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"], "Operations": [ {"op": "replace", "path": "name.formatted", "value": "John Doe"} ] }`
	
	Also, assume the advice payload is as follows.
	
	`[ {"op": "add", "path": "name.first", "value": "John"}, {"op": "remove": "path": "name.last"} ]`
	
	Then the resulting request to the store adapter looks like this.
	
	`{ "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"], "Operations": [ {"op": "replace", "path": "name.formatted", "value": "John Doe"}, {"op": "add", "path": "name.first", "value": "John"}, {"op": "remove", "path": "name.last"} ] }`
	
	If the payload is an object, PingAuthorize interprets it as a set of new replace operations to add to the end of the operations in the patch. In these replace operations, the keys from the object become the paths to modify, and the values from the object become the values for those paths. For example, assume the modify has the following patch.
	
	`{ "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"], "Operations": [ {"op": "replace", "path": "name.formatted", "value": "John Doe"} ] }`
	
	Also, assume the advice payload is as follows.
	
	`{"name.first": "John", "name.last": "Doe"}`
	
	Then the resulting request to the store adapter looks like this.
	
	`{ "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"], "Operations": [ {"op": "replace", "path": "name.formatted", "value": "John Doe"}, {"op": "replace", "path": "name.first", "value": "John"}, {"op": "replace", "path": "name.last", "value": "Doe"} ] }`

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.

<table>
<tr><th>Description</th><th>Details</th></tr>
<tr><td>Applicable to</td><td>All, although you cannot apply the Regex Replace Attributes advice directly to a System for Cross-domain Identity Management (SCIM) search.</td></tr>
<tr><td>Additional information</td><td>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.
<table>
<tr><th>Key</th><th>Description</th></tr>
<tr><td><code>"regex"</code></td><td>Required.
Represents the regular expression to use to find the attribute values to replace.</td></tr>
<tr><td><code>"replace"</code></td><td>Required.
Represents the regex replacement string to use to replace the attribute values with a new value.</td></tr>
<tr><td><code>"path"</code></td><td>Optional.
Is a JSONPath expression that represents the nodes to start searching under.</td></tr>
<tr><td><code>"flags"</code></td><td>Optional.
Is a string that contains the regex flags to use.
Recognized flags are:
▪ <code>"i"</code>
Performs case-insensitive matching.
▪ <code>"l"</code>
Treats the <code>"regex"</code> value as a literal string.
▪ <code>"c"</code>
Performs "canonical equivalence" matching.
You can combine flags. For example: <code>"il"</code></td></tr>
</table>
PingAuthorize replaces any portion of the attribute value that matches the regular expression in the <code>"regex"</code> value in accordance with the <code>"replace"</code> replacement string. If multiple substrings within the attribute value match the regular expression, PingAuthorize 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.</td></tr>
</table>

Example

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":"XXX-XX-$1"
}
```

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'.",
   "secrets":{
      "description":"Has an SSN of 'XXX-XX-4321'."
   }
}
```

REST API documentation

The PingAuthorize Policy Editor provides a set of REST APIs for managing policies, snapshots, and deployment packages. Swagger documentation for these APIs is available through the PingAuthorize Policy Editor if it was installed in demo mode.

For more information, click **API Reference** in the Policy Editor.