PingAuthorize

July 1, 2025

PINGAUTHORIZE

Version: 10.3;latest

Copyright

All product technical documentation is
Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Refer to https://docs.pingidentity.com for the most current product documentation.

Trademark

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

Disclaimer

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

Table of Contents

PingAuthorize . . . 12
Release Notes . . . 15
Introduction to PingAuthorize . . . 35
PingAuthorize architectural overview . . . 38
Getting started with PingAuthorize (tutorials) . . . 43
Using the tutorials . . . 45
Importing default policies . . . 49
Configuring fine-grained access control for an API . . . 55
Configuring attribute-based access control for API resources . . . 74
Tutorial (optional): Creating SCIM policies . . . 91
Installing PingAuthorize . . . 111
System requirements . . . 113
Docker deployment . . . 114
Deployment requirements when using Docker . . . 114
Deploying PingAuthorize Server and Policy Editor using Docker . . . 116
Post-setup steps (Docker deployment) . . . 119
Next steps . . . 120
Manual installation . . . 120
Before you install manually . . . 120
About license keys . . . 120
Creating a Java installation dedicated to PingAuthorize . . . 121
Preparing a Linux environment . . . 122
Obtaining the installation packages . . . 126
Setting up a PostgreSQL database . . . 127
Installing the server and the Policy Editor manually . . . 128
Installing the server manually . . . 128
Installing the Policy Editor manually . . . 132
Installing the Policy Editor interactively . . . 132
Installing the Policy Editor non-interactively . . . 137
Clustering and scaling . . . 147
Post-setup steps (manual installation) . . . 147
Next steps . . . 148
Signing on to the administrative console . . . 148
Signing on to the Policy Editor . . . 150
Changing the Policy Editor authentication mode . . . 150
Configuring an OIDC provider for single sign-on requests from PingAuthorize . . . 152
Uninstalling PingAuthorize . . . 170
Upgrading PingAuthorize . . . 171
Upgrade considerations . . . 173
Docker upgrades . . . 176

Manual upgrades . . . 178
Policy-related upgrades . . . 180
Backing up policies . . . 181
Upgrading the Trust Framework and policies . . . 181
Upgrading a PostgreSQL policy database . . . 182
PingAuthorize Integrations . . . 184
Apigee API gateway integration . . . 187
Setting up PingAuthorize . . . 187
Configuring Apigee . . . 188
Troubleshooting the Apigee integration . . . 197
Kong API gateway integration . . . 200
Preparing PingAuthorize for Kong Gateway integration . . . 200
Setting up Kong Gateway . . . 201
Troubleshooting the Kong Gateway integration . . . 207
Kong Konnect integration . . . 209
Preparing PingAuthorize for Konnect integration . . . 210
Configuring Konnect for PingAuthorize integration . . . 211
MuleSoft API gateway integration . . . 220
Deploying the custom MuleSoft policy for PingAuthorize . . . 220
Applying the custom MuleSoft policy for PingAuthorize . . . 221
PingAuthorize Server Administration Guide . . . 226
Running PingAuthorize Server . . . 228
Starting PingAuthorize Server . . . 228
Running PingAuthorize Server as a foreground process . . . 228
Starting PingAuthorize Server at boot time (Unix/Linux) . . . 228
Starting PingAuthorize Server at boot time (Windows) . . . 230
Registering PingAuthorize Server as a Windows service . . . 230
Running multiple service instances . . . 230
Deregistering and uninstalling services . . . 231
Log files for Windows services . . . 231
Starting PingAuthorize Policy Editor . . . 231
Stopping PingAuthorize Server . . . 235
Stopping PingAuthorize Policy Editor . . . 236
Restarting PingAuthorize Server . . . 236
About the API security gateway . . . 236
API gateway request and response flow . . . 237
Gateway configuration basics . . . 238
Tuning API security gateway performance . . . 239
API security gateway authentication . . . 241
API security gateway policy requests . . . 243
API gateway policy request attributes . . . 244
Endpoint configuration properties that affect policy requests . . . 248
Path parameters . . . 249
API security gateway HTTP 1.1 support . . . 250

Gateway error templates . . . 251
About the Sideband API . . . 253
API gateway integration . . . 254
Sideband API configuration basics . . . 256
Authenticating to the Sideband API . . . 257
API server request authentication . . . 259
Sideband API policy requests . . . 261
Sideband API policy request attributes. . . . 261
Sideband API Endpoint configuration properties . . . 265
Sideband API path parameters . . . 266
Request context configuration . . . 267
Sideband access token validation . . . 268
Sideband error templates . . . 269
About the SCIM service . . . 271
SCIM API request and response flow. . . . 271
SCIM configuration basics . . . 272
SCIM endpoints . . . 276
SCIM authentication . . . 277
SCIM policy requests . . . 277
Policy request attributes . . . 278
About SCIM searches . . . 282
SCIM search policy processing . . . 282
Using paged SCIM searches . . . 284
Lookthrough limit for SCIM searches . . . 288
Disabling the SCIM REST API . . . 288
About the SCIM user store . . . 289
Defining the LDAP user store . . . 291
Defining the LDAP user store with create-initial-config . . . 291
Defining the LDAP user store manually . . . 292
Location management for load balancing. . . . 294
Automatic backend LDAP server discovery . . . 295
Joining a PingAuthorize Server to an existing PingDirectory Server topology 295
Joining a topology at setup . . . 296
Joining a topology with manage-topology . . . 296
Configuring a load-balancing algorithm with an LDAP external template . . . 297
Configuring automatic backend LDAP server discovery . . . 298
LDAP health checks . . . 300
Configuring a health check using dsconfig. . . . 302
Connecting non-LDAP data stores . . . 304
Managing Server SDK Extensions . . . 304
About the Authorization Policy Decision APIs . . . 307
JSON PDP API request and response flow . . . 308
Authenticating to the JSON PDP API . . . 317
XACML-JSON PDP API request and response flow . . . 319

Policy Editor configuration . . . 332
Specifying custom configuration with an options file . . . 333
Configuring a key store for a policy information provider . . . 334
Example: Configuring a trust store for a policy information provider . . . 335
Configuring the Policy Editor with runtime environment variables . . . 337
Changing the default JWT claim for the OIDC user ID . . . 340
Configuring the JWKS endpoint cache . . . 342
Configuring Trust Framework attribute caching for development . . . 344
Configuring Policy Editor security headers . . . 345
Configuring Policy Editor database service connections . . . 347
Configuring Policy Editor policy request header mappings . . . 349
Manage policy database credentials . . . 350
Setting database credentials at initial setup . . . 351
Changing database credentials . . . 352
Specifying database credentials when you start the GUI . . . 353
Docker: Setting the initial database credentials . . . 354
Docker: Changing database credentials . . . 355
Configuring SpEL java classes for value processing . . . 356
Setting the request list length for Decision Visualizer . . . 358
HTTP caching . . . 359
Enabling JSON formatting for Policy Editor logs . . . 360
Enabling Mapped Diagnostic Context for Policy Editor logs . . . 361
Configuring policy query debug logging in the Policy Editor . . . 362
Policy administration . . . 365
About the Trust Framework . . . 365
Environment-specific Trust Framework attributes . . . 366
Create policies in a development environment . . . 372
Configuring external PDP mode . . . 373
Changing the active policy branch . . . 376
Default and example policies . . . 377
Importing and exporting policies . . . 377
Deploy policies in a production environment . . . 380
Exporting a policy deployment package . . . 380
Using the Deployment Manager . . . 381
Setting up an Amazon S3 deployment package store . . . 383
Configuring the Policy Editor to publish to a deployment package store . . . 386
Publishing to a deployment package store . . . 388
Adding a filesystem deployment package store . . . 390
Adding an Amazon S3 deployment package store to PingAuthorize . . . 392
Adding an Azure deployment package store . . . 394
Prepare policies for production . . . 397
Configuring embedded PDP mode . . . 400
Configuring policy request header mappings . . . 402

Policy database backups. . . . 406
Restoring a policy database from a backup . . . 408
Policy application management with signed deployment packages . . . 409
Configuring Trust Framework attribute caching for production . . . 414
User profile availability in policies . . . 418
Access token validators . . . 421
Access token validator types . . . 423
Handling signed JWTs . . . 429
Handling signed and encrypted JWTs . . . 431
Token resource lookup methods . . . 433
Server configuration . . . 435
Administration accounts . . . 435
About the dsconfig tool . . . 435
PingAuthorize administrative console . . . 436
About the configuration audit log . . . 437
About the config-diff tool . . . 437
Certificates . . . 438
Replacing the server certificate . . . 439
Preparing a new keystore with the replacement key pair . . . 439
Importing earlier trusted certificates into the new keystore. . . . 441
Updating the server configuration to use the new certificate . . . 442
Replacing the key store and trust store files . . . 443
Retiring the previous certificate . . . 443
Listener certificates . . . 443
Replacing listener certificates . . . 444
X.509 certificates . . . 447
Certificate subject DNs . . . 448
Certificate key pairs . . . 449
Certificate extensions . . . 450
Certificate chains . . . 452
About representing certificates, private keys, and certificate signing requests. . . . 453
Certificate trust . . . 453
Keystores and truststores . . . 455
Transport Layer Security (TLS). . . . 456
TLS handshakes . . . 457
Key agreement . . . 459
LDAP StartTLS extended operation . . . 459
About the manage-certificates tool. . . . 460
Available manage-certificates subcommands . . . 460
Using manage-certificates as a simple certification authority . . . 461
Common manage-certificates arguments . . . 464
Listing the certificates in a keystore. . . . 465
Generating self-signed certificates . . . 468

Generating certificate signing requests . . . 473
Importing signed and trusted certificates . . . 476
Exporting certificates . . . 478
Enabling TLS support during server setup . . . 481
Enabling TLS support after setup . . . 483
Configuring key and trust manager providers . . . 483
Configuring TLS connection handlers . . . 485
Updating the topology registry . . . 486
Troubleshooting TLS-related issues . . . 488
Log messages . . . 488
About manage-certificates check-certificate-usability . . . 489
ldapsearch for TLS-related arguments . . . 492
Using low-level TLS debugging . . . 496
Policy Decision Service configuration . . . 497
Configuring the Decision Response View . . . 497
User store configuration . . . 499
Configuring database service connections . . . 500
Configuring Access Token Validation . . . 504
Configure PingOne to use SSO for the administrative console . . . 505
Configuring traffic through a load balancer . . . 507
PingAuthorize Server configuration with dsconfig . . . 509
Configuring the PingAuthorize User Store . . . 509
Configuring the PingAuthorize OAuth subject search . . . 510
Configuring PingAuthorize logging . . . 511
Deployment automation and server profiles . . . 512
Variable substitution using manage-profile . . . 513
Layout of a server profile . . . 514
About the manage-profile tool . . . 516
Common manage-profile workflows . . . 518
Server profiles in a pets service model . . . 522
Server status . . . 523
Server availability . . . 524
User Store Availability gauge . . . 524
Endpoint Average Response Time (Milliseconds) gauge . . . 525
HTTP Processing (Percent) gauge . . . 526
Policy Decision Service Availability gauge . . . 527
Auto-healing for unavailable servers . . . 528
Available gauges . . . 528
Managing logging . . . 531
Default PingAuthorize Server logs . . . 531
Enable detailed logging . . . 532
Types of log publishers . . . 535
Viewing the list of log publishers . . . 537
Enabling or disabling a default log publisher . . . 538

About log compression . . . 538
About log signing . . . 539
About encrypting log files . . . 539
Creating new log publishers. . . . 542
Configuring log rotation . . . 543
Configuring log rotation listeners . . . 544
Configuring log retention . . . 545
Configuring filtered logging . . . 546
Managing the File-Based Error Log Publisher . . . 547
Managing the Syslog-Based Error Log Publisher . . . 548
Creating File-Based Debug Log Publishers . . . 549
Manage monitoring . . . 550
Monitoring with the administrative console . . . 550
Profiling server performance using the Stats Logger . . . 551
Logging HTTP performance statistics using a Periodic Stats Logger . . . 553
Sending Metrics with the Periodic Stats Logger and the Splunk Universal Forwarder 554
StatsD monitoring endpoint . . . 554
Sending StatsD metrics to Splunk . . . 556
Monitoring server metrics with Prometheus . . . 556
Managing notifications and alerts . . . 559
Common server alarms . . . 560
Managing HTTP correlation IDs . . . 562
About HTTP correlation IDs . . . 562
Enabling or disabling correlation ID support . . . 563
Configuring the correlation ID response header . . . 563
How the server manages correlation IDs . . . 564
Command-line tools . . . 565
Saving command options in a file . . . 570
Creating a tools properties file . . . 571
Evaluation priority of command-line options . . . 572
Sample dsconfig batch files . . . 573
Running task-based tools . . . 574
About the layout of the PingAuthorize Server folders. . . . 575
About the layout of the Policy Editor folders . . . 577
PingAuthorize Policy Administration Guide . . . 577
Getting started . . . 579
Version control (Branch Manager) . . . 580
Managing branches . . . 581
Creating a branch . . . 581
Merging branches . . . 584
Deleting a branch . . . 586
Branch changes and history. . . . 586
Committing changes . . . 586
Reverting changes . . . 587

Creating a snapshot . . . 587
Merging snapshots . . . 589
Deployment packages . . . 590
Deleting a deployment package . . . 591
Trust Framework . . . 591
Domains (Authorization Policy Decision APIs only) . . . 591
Services . . . 591
Connecting a service . . . 592
Database services . . . 595
HTTP services . . . 599
LDAP services . . . 603
Enabling Camel service connections . . . 605
Service caching . . . 607
Attributes . . . 608
Creating an attribute . . . 609
Attribute name, description, and location . . . 609
Resolvers . . . 609
Attribute caching . . . 613
Value settings . . . 614
Attribute interpolation . . . 618
Conditions . . . 618
Adding a named condition . . . 622
Actions . . . 623
Identity classifications and IdP support . . . 624
Processors . . . 625
SpEL processing examples . . . 630
Copying elements . . . 636
Deleting persistent copies . . . 638
Testing Trust Framework elements . . . 638
Viewing dependents . . . 642
Policy management . . . 643
Policy sets, policies, and rules . . . 644
Creating policies and policy sets . . . 644
Adding targets to a policy . . . 645
Conditional targets (Applies to) . . . 647
Statements . . . 648
Provided statements . . . 651
Custom statements . . . 662
Combining algorithms . . . 662
Rule structure . . . 666
Testing policies . . . 668
Visualizing a policy decision response . . . 670
Policy queries . . . 672
Enabling query settings . . . 676

Policy query request and response examples . . . 679
Troubleshooting policy queries . . . 686
Policy query logging . . . 690
Repeating policies and attributes . . . 696
Self-governance . . . 698
Self-governance use cases . . . 700
Self-governance Trust Framework . . . 702
Importing a self-governance policy snapshot . . . 709
Policy solutions . . . 709
Use case: Using consent to determine access to a resource . . . 710
Getting a path component from the request URL . . . 710
Getting the requestor identifier from the access token . . . 716
Searching for consent granted by resource owner to requestor . . . 716
Getting consent status from the consent record . . . 721
Creating a policy to check consent and then permit or deny access . . . 722
Use case: Using consent to change a response . . . 726
Creating a policy to check consent and then change the server response . . . 727
Use case: Using a SCIM resource type or a policy request action to control behavior . . . 733
Getting the SCIM resource type and the action being executed . . . 734
Creating a policy to permit or deny the creation of resources . . . 736
Creating a policy to control the set of actions for a specific resource . . . 738
Creating a policy to restrict the ability to delete based on resource type . . . 741
Creating a policy to modify a resource differently based on the SCIM resource type . . . 743
Restricting the attributes that can be modified . . . 746
Allowing attributes to be modified by administrators . . . 747
Adding attributes to an allow list . . . 747
Test Suite . . . 748
Management REST API documentation . . . 751
Troubleshooting PingAuthorize Server . . . 751
Working with the collect-support-data tool . . . 753
PingAuthorize Server logs for troubleshooting and monitoring . . . 758
Advanced HTTP event logging . . . 762
Troubleshooting resources for Java applications . . . 766
Java troubleshooting tools . . . 766
Java diagnostic information . . . 768
Troubleshooting resources in the operating system . . . 769
Common problems and potential solutions . . . 773
General troubleshooting methodology . . . 773
The server will not run setup . . . 773
The server will not start . . . 774
The server has crashed or shut itself down . . . 778
Conditions for automatic server shutdown . . . 778
The server will not accept client connections . . . 778

The server is unresponsive 779
The server is slow to respond to client requests 780
The server returns error responses to client requests 781
Problems with the administrative console 782
Problems with the HTTP Connection Handler 783
Detecting expensive HTTP threads 784
Providing information for support cases 788
PingAuthorize API Reference 788

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
- Previous releases

Get Started with PingAuthorize

- Introduction to PingAuthorize
- Installing PingAuthorize
- PingAuthorize Tutorials

Use PingAuthorize

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

Copyright © 2025 Ping Identity Corporation

Troubleshoot PingAuthorize

- Enable detailed logging
- 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)

Copyright © 2025 Ping Identity Corporation

Release Notes

New features and improvements in PingAuthorize. Updated July 1, 2025.

Subscribe to get automatic updates: PingAuthorize Release Notes RSS feed

Latest releases

PingAuthorize 10.3.0.0 (July 2025)

Removed support for Java 11

Info

Support for Java 11 has been removed. You must be running Java 17 or a later supported version, as detailed in the System requirements. Learn more about upgrading a PingAuthorize instance running Java 11 in Considerations when upgrading to version 10.3.

Support for Internet Explorer 11 has been deprecated

Info

Support for Internet Explorer 11 has been deprecated and will be removed in a future release.

Policy Editor configuration properties have been deprecated

Info

The following `configuration.yml` properties have been deprecated in the Policy Editor:

- `core.DecisionResponse.AuditLogView`: To avoid issues when upgrading to future versions of PingAuthorize, define a new top-level `auditLogging.responseView` property and migrate your audit logging configuration to it. The `auditLogging` property overrides any configuration defined with the deprecated `core.DecisionResponse.AuditLogView` property.
- `core.Attribute.Cache`: To avoid issues when upgrading to future versions of PingAuthorize, define a new top-level `cache` property and migrate your caching configuration to it. The `cache` property overrides any configuration defined with the deprecated `core.Attribute.Cache` property.

Removed extra Policy Editor version number

Info

We removed an extra application version number displayed by the Policy Editor.

Documented custom SpEL functions

Info

We added documentation for a library of custom SpEL functions introduced in PingAuthorize 10.2, focusing on common use cases for working with collections, dates and times, and Uniform Resource Identifiers (URIs).

Learn more in SpEL processing examples.

Copyright © 2025 Ping Identity Corporation

Added runtime support for Java 21

New PAZ-15881, PAZ-17469

We added JRE support for Oracle JDK 21 and OpenJDK 21.

Added support for Camel 4.0

New PAZ-17485

Although Camel services have been removed from the default PingAuthorize configuration, you can now use Camel version 4.0 if your policies depend on such services. Learn more in Enabling Camel service connections.

Override short-circuiting in combining algorithms

New PAZ-11577

By default, combining algorithms stop evaluating a policy's child elements as soon as a final decision is reached. You can use the new **Evaluate All** option to override this behavior and ensure that all child policies or rules are evaluated, even if a final decision has been reached. This option allows additional statement information to propagate up to the parent policy, which can help with auditing, debugging, and analyzing policy outcomes.

Learn more in Combining algorithms.

Added circuit breaker functionality for services

New PAZ-18011

We added circuit breaker functionality for policy information points (PIPs). Circuit breakers enhance system resilience by preventing cascading failures from unhealthy PIPs. When a service fails repeatedly, the circuit breaker temporarily halts requests to that service and periodically tests for recovery.

Learn more about circuit breaker configuration in Connecting a service.

Added a branch counter to the Policy Editor

New PAZ-17552

The Policy Editor Branch Manager now displays a visual count of all branches defined in the system. You can define up to 100 top-level branches, including system and self-governance branches.

Learn more in Creating a branch.

Added advanced logging for HTTP event types

New PAZ-17947

We added more detailed HTTP event logging to the file-based trace logger. You can now enable logging for individual event types across the entire HTTP request lifecycle. These events provide deeper visibility into request handling and can help identify performance bottlenecks.

Learn more in Advanced HTTP event logging.

Copyright © 2025 Ping Identity Corporation

Added support for expensive HTTP thread detection

New PAZ-18104

We added expensive HTTP thread monitoring to help identify and troubleshoot performance bottlenecks in the HTTP connection handler. When an HTTP thread's execution time exceeds a configured threshold, PingAuthorize can generate a thread dump. This provides valuable insight into the exact point of delay, enabling quicker identification and resolution of slowdowns.

Learn more in Detecting expensive HTTP threads.

Added correlation ID support in service logs

New PAZ-18013

We added a `correlationId` field to trace log messages that record details of downstream service calls. Correlation IDs help link service logs with other messages from the same decision request, giving you deeper visibility into the full request lifecycle.

Learn more about HTTP correlation ID behavior in About HTTP correlation IDs.

Removed unused dependencies

Improved DS-49566

We reduced the size of the PingAuthorize Server distribution by removing unnecessary `.jar` files.

Improved API security gateway timeout behavior

Improved PAZ-17936

We made the following improvements to API security gateway request timeout behavior:

- The security gateway now provides clearer JSON-formatted error messages for easier troubleshooting.
- Error logs now record a `GatewayTimeout` message. This replaces the less specific `IllegalStateException` message, enabling quicker and more precise identification of the issue.

Improved policy query debug logging

Improved PAZ-18509

We changed how policy query permutations are recorded in the policy query debug log. Now, each permutation is logged as an individual entry. This makes it easier to ingest and analyze detailed query logs in your SIEM or other log management systems, especially for query requests that generate many permutations.

Learn more in Policy query logging.

Made the collect-support-data tool more secure

Improved PAZ-18106

To better protect sensitive information, we updated the **`collect-support-data`** tool. The tool now automatically redacts deployment package content from generated support file bundles, ensuring your proprietary data is kept more secure.

Copyright © 2025 Ping Identity Corporation

Added support for Policy Editor paging

Improved PAZ-18422

To improve performance and usability when navigating large datasets, we added support for paging across all list and tree elements in the Policy Editor, including nested elements. Elements are loaded in fixed pages of 100, with a **Load more** option available to retrieve additional items incrementally.

Fixed an issue with policy query responses

Fixed PAZ-18152

We fixed an issue where attributes marked as secret were incorrectly encrypted in responses from the `governance-engine/query` endpoint, in addition to being encrypted in the policy query debug log.

Fixed an issue with custom OAuth parameters

Fixed PAZ-16760

We fixed an issue where key-value pairs defined as custom OAuth parameters didn't reset after the authentication type for an HTTP service was changed.

Fixed an issue with self-governance for deployment package stores

Fixed PAZ-16349

We fixed an issue where the self-governance decision engine didn't intercept requests to deployment package stores.

Fixed an issue with AWS S3 logging

Fixed PAZ-18565

We fixed an issue where AWS S3 credentials were inadvertently included in application logs.

Fixed an issue with decision request handling

Fixed PAZ-16549

To improve request handling, we fixed an issue that caused a `NullPointerException` message to be logged when decision requests were missing the `attributes` property in the request body.

Fixed an issue with database connection pools

Fixed PAZ-18403

We fixed an issue where some database service connections didn't properly close JDBC statement objects. This could lead to connection pool limits being reached because of unreleased resources, even at low traffic levels.

Fixed an issue with SCIM responses

Fixed DS-48511

Copyright © 2025 Ping Identity Corporation

We fixed an issue with inconsistencies in `id-attribute` values returned in SCIM operation responses. Also, we fixed an issue with SCIM GET operations where a filter used to search for an entry would result in a 404 error.

Fixed an issue with the Monitor History Plugin

Fixed DS-46253

We fixed an issue where the Monitor History Plugin wouldn't preserve files longer than 14 days when the `retain-files-sparsely-by-age` setting was set to `true`.

PingAuthorize 10.2.0.1 (March 2025)

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 10.2 (December 2024)

Deprecated support for Java 11

Info PAZ-16269

Support for Java 11 has been deprecated. PingAuthorize 10.3 and later will not support Java 11. To prevent problems when upgrading PingAuthorize, you should upgrade to Java 17. Learn more in System requirements.

Custom SDK extensions using Javax packages will need to be migrated and recompiled in 10.3

Info

Several components will be upgraded in version 10.3 of PingAuthorize. If any of your custom Server SDK extensions have classes that import `javax.*` packages, you will need to migrate them to the equivalent `jakarta.*` packages and then recompile the extensions.

Advanced OAuth Settings for HTTP Services

New PAZ-16062

For HTTP Services using the OAuth 2.0 (Client Credentials) grant type for authentication, we added the ability to include custom key-value pairs as additional parameters in the body of token endpoint requests. This level of customization is useful when integrating with authorization services that enforce specific configuration constraints. Learn more in HTTP services.

Added a new policy query response view

New PAZ-12855

Copyright © 2025 Ping Identity Corporation

To provide enhanced visibility into the logic underlying policy query responses, we added support for a new `unfiltered` policy query response view. This view includes every decision used to compose the final policy query response, regardless of the outcome or the presence of statements. Learn more in Configuring query response granularity.

Added debug logging for policy query requests

New PAZ-12855

We added debugging capabilities for policy query audit logging in the PingAuthorize Server and the Policy Editor. Enabling debug logging brings enhanced visibility to the attribute data and policy dependencies involved in policy query responses. Learn more in Policy query logging.

Added ability to customize policy query debug log views

New PAZ-16050

For policy query audit logs with debug logging enabled, we added the ability to specify additional levels of detail to include in each query permutation's `response` field. This level of customization enables you to increase or decrease visibility into query requests and responses in both development and production environments. Learn more in Configuring the query permutation view.

Added support for attribute logging to policy query responses

New PAZ-16050

You can now exercise control over which attributes get logged as part of policy query responses in embedded or external PDP mode. This configuration logs full details of the specified attributes when they're evaluated as part of the policy query request.

Added support for Mapped Diagnostic Context

New PAZ-13188

We added support for Mapped Diagnostic Context (MDC) capabilities in PingAuthorize Server and Policy Editor logging. MDC enriches log messages with additional request context, enabling you to diagnose issues across application components. Learn more in Enabling Mapped Diagnostic Context for Policy Editor logs.

> **Note**
>
> MDC logging is enabled by default in the PingAuthorize Server.

Added support for a new signature algorithm

New PAZ-13089

We added support for Elliptic Curve Digital Signature Algorithm (ECDSA)-encoded signatures for JWT tokens when using the `Has Valid Signature For JWKS` and `Has Invalid Signature For JWKS` attribute comparators. Learn more in Conditions.

Added native support for `decision-point-spel-functions`

New PAZ-15718

Copyright © 2025 Ping Identity Corporation

We added the `decision-point-spel-functions` library to the PingAuthorize Server distribution. These functions add native capabilities for collection, datetime, and URI processing with SpEL.

Added support for Camel 3.22.2

New PAZ-15845

Although Camel services have been removed from the default PingAuthorize configuration, you can now use Camel version 3.22.2 if your policies depend on such services. Learn more in Enabling Camel service connections.

Added key and trust manager caching

New DS-49135

Added the ability to cache key managers and trust managers to prevent loading keystore and truststore files from disk when establishing connections to process requests. Use the `enable-key-manager-caching` and `enable-trust-manager-caching` configuration properties to enable or disable caching. Learn more in Configuring key and trust manager providers.

Improved policy query error handling

Improved PAZ-12527

We improved and standardized error handling for requests sent to the JSON PDP API's `governance-engine/query` endpoint.

Made it easier to resolve policy query attributes

Improved PAZ-12724

When making a policy query request, you might want to use a single-valued attribute to resolve a `query` attribute. Now, you can include this single-valued attribute in the `query` section, rather than the `context` section of the JSON PDP API request.

Allowed proxied requests for HTTP external servers

Improved DS-48729

We updated the PingAuthorize Server's HTTP external server configuration to allow requests to be forwarded through an HTTP proxy server.

Fixed an issue with Policy Query API responses

Fixed PAZ-12752

We fixed an issue where the same request sent to the JSON PDP API's `governance-engine/query` endpoint could produce inconsistent responses.

Fixed an issue with testing large elements

Fixed PAZ-11133

We fixed an issue where running test scenarios that exceeded 50 kB in size would produce the following error: `Failed to execute 'setItem' on 'Storage': Setting the value of 'xxx' exceeded the quota.`

Copyright © 2025 Ping Identity Corporation

Now, although test scenarios of this size will still execute, these scenarios will not be stored in the Policy Editor's local cache. Learn more in Testing policies.

Fixed an issue with redundant policies when using the Policy Editor Management API

Fixed PAZ-12166

We fixed an issue where creating a copy of an existing policy with the Policy Editor Management API would generate a redundant copy of that policy. This occurred because the version ID of the original policy was supplied in the POST request to create the policy copy. With this fix, users can no longer provide a version ID when creating a new policy.

Fixed an issue with the Policy Editor login screen

Fixed PAZ-2355

We fixed an issue where the Policy Editor login screen was incorrectly aligned on displays with insufficient vertical height.

Fixed a performance issue with policy queries

Fixed PAZ-16415

We fixed an issue where policy query responses were constructed with an excessively broad response view, causing performance issues.

Fixed an issue with merging branches

Fixed PAZ-15930

We fixed an issue that would cause false conflicts when merging policy branches in the Policy Editor.

Fixed a Policy Editor sign-off issue

Fixed PAZ-16253

We fixed an issue that would prevent complete sign-off from the Policy Editor when using the Authorization Code with PKCE grant type.

Fixed an issue with the Decision Visualizer

Fixed PAZ-11767

We fixed an issue where pasting an audit log entry without the `decisionTree` or `evaluationLog` field into the Policy Editor's Decision Visualizer would incorrectly return a `Cannot read properties of undefined (reading 'create')` error in addition to a `decisionTree or evaluationLog must be enabled to visualize decision tree` error.

Fixed an issue with deployment package store requests

Fixed PAZ-16140

We fixed an issue with the PingAuthorize Server that caused logs and monitors to make requests to deployment package stores more frequently than the configured polling interval, creating performance issues.

Copyright © 2025 Ping Identity Corporation

Changed the `collect-support-data` monitor file behavior

Fixed DS-47384

For the PingAuthorize Server, we changed the **`collect-support-data`** tool to use the latest `monitor-history` file if it can't find `ldap/monitor.ldif` when exchanging monitor data.

Fixed an issue with Promotheus HTTP servlet error messages

Fixed DS-49161

We fixed an issue with the PingAuthorize Server where the Prometheus HTTP servlet would publish an excessive number of error messages to the error log when it lost connection to its remote counterpart.

Fixed an issue with `config-diff`

Fixed DS-49071

We fixed an issue with the PingAuthorize Server where running the **`config-diff`** tool would result in an `Unknown property` error when comparing configuration objects of different types.

Removed suppression messages for disabled alerts

Fixed DS-49119

We fixed an issue with the PingAuthorize Server where alert types that were disabled would still output suppression messages.

PingAuthorize 10.1.0.3 (May 2025)

Added key and trust manager caching

New DS-49135

We added the ability to cache key managers and trust managers to prevent loading key store and trust store files from disk when establishing connections to process requests. Use the `enable-key-manager-caching` and `enable-trust-manager-caching` configuration properties to enable or disable caching. Learn more in Caching key and trust managers.

Fixed an issue with deployment package store requests

Fixed PAZ-16140

We fixed an issue with the PingAuthorize Server that caused logs and monitors to make requests to deployment package stores more frequently than the configured polling interval, causing performance issues.

Fixed SCIM response errors

Fixed DS-48511

We fixed an issue with inconsistencies in `id-attribute` values returned in SCIM operation responses. We also fixed an issue with SCIM GET operations, where a filter used to search for an entry would result in a 404 error.

Copyright © 2025 Ping Identity Corporation

Removed suppression messages for disabled alerts

Fixed DS-49119

We fixed an issue where alert types that were disabled would still output suppression messages.

PingAuthorize 10.1.0.2 (September 2024)

Added a new policy query response view

New

We added support for a new `unfiltered` policy query response view. This view includes every decision used to compose the final query response, regardless of the outcome or the presence of statements. Learn more in Configuring query response granularity.

Added policy query debug logging

New

We added debugging capabilities for policy query audit logging. Enabling the `DEBUG` log level brings enhanced visibility to the attribute data and decision logic underlying query responses. Learn more in Troubleshooting policy queries.

Added support for Mapped Diagnostic Context

New

We added support for Mapped Diagnostic Context (MDC) capabilities in logging. MDC enriches log messages with additional request context, enabling you to diagnose issues across application components. Learn more in Enabling Mapped Diagnostic Context for Policy Editor logs.

Added support for a new signature algorithm

New

We added support for Elliptic Curve Digital Signature Algorithm (ECDSA)-encoded signatures for JWT tokens when using the `Has Valid Signature For JWKS` and `Has Invalid Signature For JWKS` attribute comparators. Learn more in Conditions.

Improved policy query error handling

Improved

We improved and standardized error handling for requests sent to the `governance-engine/query` endpoint.

Improved resolution of query attributes

Improved

When making a policy query request, you might want to use a single-valued attribute to resolve a `query` attribute. Now, you can include this single-valued attribute in the `query` section, rather than the `context` section of the request.

Copyright © 2025 Ping Identity Corporation

Fixed an issue with Policy Query API responses

Fixed PAZ-12752

We fixed an issue where the same request sent to the `governance-engine/query` endpoint could produce inconsistent responses.

Fixed an issue with the Decision Visualizer

Fixed PAZ-11767

We fixed an issue where pasting an audit log entry without the `decisionTree` or `evaluationLog` field into the Decision Visualizer would incorrectly return a `Cannot read properties of undefined (reading 'create')` error in addition to a `decisionTree or evaluationLog must be enabled to visualize decision tree` error.

Fixed a memory leak issue

Fixed PAZ-13013

We fixed an issue where, when using the PingAuthorize Server's API security gateway in embedded PDP mode, policy decision logging could cause a memory leak and negatively impact the performance of long-running server instances.

Fixed an issue with `config-diff`

Fixed DS-49071

We fixed an issue where `config-diff` would result in an `Unknown property` error when comparing configuration objects of different types.

PingAuthorize 10.1 (June 2024)

Make real-time data calls to relational databases

New

We added a new **Database** service type, enabling you to use relational databases as policy information points (PIP) during policy development. Services retrieve information from external data sources for use in context-aware authorization decisions. Now, you can dynamically query and transform such information from relational databases. Learn more in Database services.

Implement third-party decision log publishers with the PingData Server SDK

New

We added a new Policy Decision Logger extension to the Server SDK for developing third-party decision log publishers. This extension enables you to configure custom decision logging behavior and log destinations. You can extend the provided `PolicyDecisionLogger` API in the Server SDK to implement your custom logic. Learn more in Managing Server SDK Extensions and the `doc/getting-started` directory in your SDK download.

Copyright © 2025 Ping Identity Corporation

Map decision request headers to Trust Framework attributes

New

We added the ability to map headers of incoming JSON PDP API requests to Trust Framework attributes. PingAuthorize uses these mappings to dynamically populate attribute values with the values of incoming request headers, enabling you to leverage header data as decision context in request bodies. Learn more in Configuring policy request header mappings.

Determine whether service call results were retrieved from cache

New

We added the ability, for any call to external services with caching enabled, to determine whether the result of that call was retrieved from the cache. Learn more in Service caching.

Enable Camel service connections with the command line

New

We added the ability to enable Camel service connections in the Policy Editor with the command line. To ensure that Camel is used with the appropriate permissions and security controls, Camel services are disabled by default in the Policy Editor. Now, instead of manually updating the Policy Editor's configuration, you can enable Camel service connections with the `--enableCamelService` option in non-interactive setup mode. Learn more in Enabling Camel service connections.

Enable JSON formatting for default Policy Editor loggers

New

We added support for the `dropwizard-json-logging` library to the default Policy Editor loggers. Now, you can add this library in the Policy Editor's `configuration.yml` file to each logger stream you wish to enable JSON formatting for. The availability of this library does not impact the application's default configuration. Learn more in Enabling JSON formatting for Policy Editor logs.

Added support for new platforms

New

We added support for Rocky Linux 9.3 and Red Hat Enterprise Linux 9.3. Learn more in System requirements.

Improved database service security

Improved

To address the possibility of remote code execution attacks with H2 database services, we made the database driver allow list configurable and unlisted H2 by default. Learn more in Database services.

Updated default configuration archive maximum

Info

To mitigate the performance impact of large archives, we updated the configuration archive to keep a maximum of 100 previous configurations by default

Copyright © 2025 Ping Identity Corporation

Fixed an issue with creating copies of policies

Fixed PAZ-12150

We fixed an issue where, in some cases, copying a policy created a redundant instance of that policy.

Fixed an issue with Library statement duplication

Fixed PAZ-9092

We fixed an issue where copying a rule containing a **Library** statement would create a new instance of that statement instead of reusing the existing one.

Fixed an issue with saving LDAP services

Fixed PAZ-12017

We fixed an issue where, after enabling caching for an LDAP service, specifying a **Time to Live** (TTL), and clicking **Save**, the specified TTL disappeared from the UI and backend configuration.

Fixed an issue with HTTP service requests

Fixed PAZ-12145

We fixed an issue where, when making HTTP service calls, the policy decision point would incorrectly assign default values to the request body and the `content-type` header.

Fixed an issue with self-governance decision requests

Fixed PAZ-3306

We fixed an issue where, when using an imported policy snapshot, self-governance decision requests were missing values in the `action` field.

Fixed an issue with Policy Query API responses

Fixed PAZ-12245

We fixed an issue where, when sending a Policy Query API request with an unbounded attribute in the `query` array, the system would return a 500 error status code if the unbounded attribute's value was resolved to an empty collection.

Fixed an issue with the CLI tools reference

Fixed PAZ-3469

We fixed an issue where the CLI tools reference page was incorrectly titled **Configuration Reference**. Now, the page is correctly titled **CLI Tools Reference**.

Fixed an issue with the comparators list

Fixed PAZ-11768

Copyright © 2025 Ping Identity Corporation

We fixed an issue where, when creating rules in the Policy Editor, the comparators list extended outside of the list area, preventing you from scrolling through the list.

Fixed an issue with unnamed Trust Framework elements

Fixed PAZ-12150

We fixed an issue where a user could leave new elements defined in the Trust Framework unnamed, giving them a default name of **Untitled**. Now, you must specify a name for such elements before saving them.

Fixed an issue with copying Policy Editor elements

Fixed PAZ-12150

We fixed an issue where Policy Editor elements created as copies would inherit the version ID of the original element. As a result, copies of elements would persist in the Policy Editor UI after being deleted but would return 404 errors when selected. Now, copies of Policy Editor elements have distinct version IDs.

Fixed an issue with HTTP service log messages

Fixed PAZ-12454

We fixed an issue where the `status` field of an HTTP service log message would include a status message, such as `OK`, rather than a status code.

Fixed an issue with PIP key store service calls

Fixed PAZ-12014

We fixed an issue where, when making a service call with a PIP key store for MTLS configured and the **Server (TLS)** option set to **None** or **Default**, the service would incorrectly return a `Client TLS certificate is required` error.

PingAuthorize 10.0.0.6 (June 2025)

Fixed an issue with deployment package store requests

Fixed PAZ-16140

We fixed an issue with the PingAuthorize Server where logs and monitors made requests to deployment package stores more frequently than the configured polling interval, causing performance issues.

PingAuthorize 10.0.0.4 (October 2024)

Fixed a PingAuthorize Server performance issue

Fixed PAZ-13013

We fixed an issue where, when using PingAuthorize Server's API security gateway in embedded PDP mode, policy decision logging could cause memory leaks and negatively impact the performance of long-running server instances.

Copyright © 2025 Ping Identity Corporation

Fixed a `config-diff` error

Fixed DS-49071

We fixed an issue where `config-diff` would result in an `Unknown property` error when comparing configuration objects of different types.

Fixed an alert types issue

Fixed DS-49119

We fixed an issue where alert types that were disabled would still produce suppression messages.

PingAuthorize 10.0.0.3 (July 2024)

Fixed an issue with HTTP service requests

Fixed PAZ-12145

We fixed an issue where, when making HTTP service calls, the policy decision point would incorrectly assign default values to the request body and the `content-type` header.

Fixed an issue with unbounded query attributes

Fixed PAZ-12245

We fixed an issue where, when sending a Policy Query API request with an unbounded attribute in the `query` array, the system would return a 500 error status code if the unbounded attribute's value was resolved to an empty collection.

Fixed an issue with Policy Query API responses

Fixed PAZ-12752

We fixed an issue where the same request to the Policy Query API could produce inconsistent responses.

Fixed an issue with HTTP service log messages

Fixed PAZ-12454

We fixed an issue where the `status` field of an HTTP service log message would include a status message, such as `OK`, rather than a status code.

Fixed an issue with PIP key store service calls

Fixed PAZ-12014

We fixed an issue where, when making a service call with a PIP key store for MTLS configured and the **Server (TLS)** option set to **None** or **Default**, the service would incorrectly return a `Client TLS certificate is required` error.

Copyright © 2025 Ping Identity Corporation

PingAuthorize 10.0.0.2 (March 2024)

Fixed a header exclusion issue with HTTP service caching

Fixed STAGING-22303

We fixed an issue with the exclusion of certain headers from the cache key of cached HTTP service responses. Now, each change to these header values no longer invalidates the service response cache, and the decision engine isn't forced to invoke the service again on subsequent requests.

Fixed a header display issue with HTTP service caching

Fixed PAZ-11726

We fixed an issue with the display of headers excluded from cached HTTP service responses in the Trust Framework. Now, you can navigate away from an HTTP service with caching enabled, navigate back to that service, and still see the excluded headers you originally defined.

PingAuthorize 10.0.0.1 (January 2024)

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 10.0 (December 2023)

Send more flexible decision requests with policy queries

New

With the new Policy Query API, you can now issue decision requests containing valueless and multivalued attributes to receive decisions more complex than `Permit` or `Deny`, enabling you to dynamically drive user interfaces. Learn more in Policy queries.

Cache dynamic service responses

New

To improve decision evaluation performance and reduce latency, you can cache dynamic service response values for faster retrieval on subsequent requests. When enabling caching for HTTP services, you can exclude certain headers from the service response. This prevents invalidation of the cache when values of those headers change. Learn more in Service caching.

Copy Trust Framework attribute resolvers

New

Copyright © 2025 Ping Identity Corporation

To build your authorization logic more efficiently, you can make editable copies of attribute resolvers. Learn more in Copying elements.

Disable rules in the policy tree

New

To control the granularity of policy evaluation, you can disable rules in policies. This causes the decision engine to skip disabled rules during policy evaluation and allows you more flexibility in testing and deployment of policy logic. Learn more in Creating policies and policy sets.

Added support for Apache Camel 3.21.2

Info

Although Camel services have been removed from the default PingAuthorize configuration, you can now enable Camel version 3.21.2 if your policies depend on such services. Learn more in Enabling Camel service connections.

Added support for Java 17 and removed support for Java 8

Info

We have added support for Java 17 and removed support for Java 8. Learn more in System requirements. For information on upgrading from a PingAuthorize instance installed with Java 8, refer to Upgrade considerations introduced in PingAuthorize 10.0.

Disabled SNI hostname checks by default

Info PAZ-10754

To avoid `HTTP 400` responses when SNI hostname checks fail, these checks are now disabled by default for the PingAuthorize server and Policy Editor. We added a new `setup` option, `--disableSniHostnameChecks`, to control whether PingAuthorize performs this check. For important considerations when upgrading from a previous version and attempting to reuse your configuration, refer to Upgrade considerations introduced in PingAuthorize 10.0.

Disabled OIDC Implicit grant flow

Info PAZ-1795

We have disabled the OIDC Implicit flow implementation in the Policy Editor because the OAuth Working Group no longer recommends its use. In its place, you should use the Authorization Code with PKCE flow. Learn more in Configuring an OIDC provider for single sign-on requests from PingAuthorize.

Added indexes to improve database query performance

Improved

We added two database indexes to the `db-cli` module to improve performance when querying the `CurrentEntityVersion` and `EntityRelationship` tables.

Copyright © 2025 Ping Identity Corporation

Fixed SCIM case-sensitivity issue

Fixed PAZ-8473

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.

Fixed attribute caching memory error

Fixed PAZ-10643

We fixed an issue where the decision engine only checked if an attribute cache entry had expired when accessing that entry, leading to `Out of Memory` errors. Now, attribute caching uses the Redis library directly, allowing a unique **Time to Live** (TTL) for each cache entry. Redis instances invalidate cache entries once the TTL has elapsed, rather than when the entries are accessed. Learn more in Attribute caching.

Fixed missing statements array in policy testing

Fixed PAZ-6335

We fixed an issue, where, in the **Response** tab of policy testing, the root-level `statements` array was not appearing if left empty in the testing scenario.

Fixed error response handling in `APP WARN`

Fixed PAZ-10350

We fixed an issue where the HTTP Service Executor was not properly capturing error messages in the `APP WARN` logs from the policy information provider (PIP) endpoint.

Removed `--serverRoot` requirement from the `check-replication-domains` tool

Fixed DS-47655

We fixed the `check-replication-domains` tool so that the `--serverRoot` argument is no longer required. This argument now defaults to the server's root directory.

Fixed duplication issue when running `dsjavaproperties --initialize`

Fixed DS-45206

We fixed an issue where running `dsjavaproperties --initialize` would append duplicate arguments to the `common.java-args` in the `java.properties` file.

Replaced `NullPointerException` error for alert handlers lacking configuration

Fixed DS-47455

We fixed an issue where a `NullPointerException` error occurred when an alert or alarm was raised, and one more of the alert handlers was not configured. An alert notification is now recorded in `logs/errors` instead.

Copyright © 2025 Ping Identity Corporation

Addressed inability of LDAP Request Handlers to respond to incoming client requests

Fixed DS-46312

We fixed an issue where TLS timeouts prevented LDAP Request Handlers from responding to client requests. The `request-handler-per-connection` configuration property is now available for LDAP and LDAPS Connection Handlers.

Previous Releases

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

- 9.3
- 9.2
- 9.1

Copyright © 2025 Ping Identity Corporation

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 attributes, consents, entitlements, resources, or contexts to make attribute-based access control (ABAC) decisions in real time. PingAuthorize gives you centralized control over your digital transactions and application-level access to your protected resource.

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 source 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) application programming interface (API). Applications then enforce these decisions in their own application or service code. This integration configuration is the most flexible, supporting any application or service use case.

Copyright © 2025 Ping Identity Corporation

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 datastore like Lightweight Directory Access Protocol (LDAP) and RDBMS, customers can deploy the System for Cross-domain Identity Management (SCIM) service in front of their datastores.

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 license. For more information, check your PingAuthorize license key or contact your Ping Identity account representative.

Learn more about PingAuthorize architecture and policy decision environments in PingAuthorize architectural overview.

Get started

Learn how to quickly spin up a PingAuthorize solution and walk through some simple use cases in Getting started with PingAuthorize (tutorials).

PingAuthorize architectural overview

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

Components

Policy Editor

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

PingAuthorize Server

The PingAuthorize Server enforces policies to control fine-grained access to data.

When you configure a REST API to access data through the PingAuthorize Server, the server applies data-access policies that allow, block, filter, or modify protected resource and data attributes.

Copyright © 2025 Ping Identity Corporation

Implementation architectures

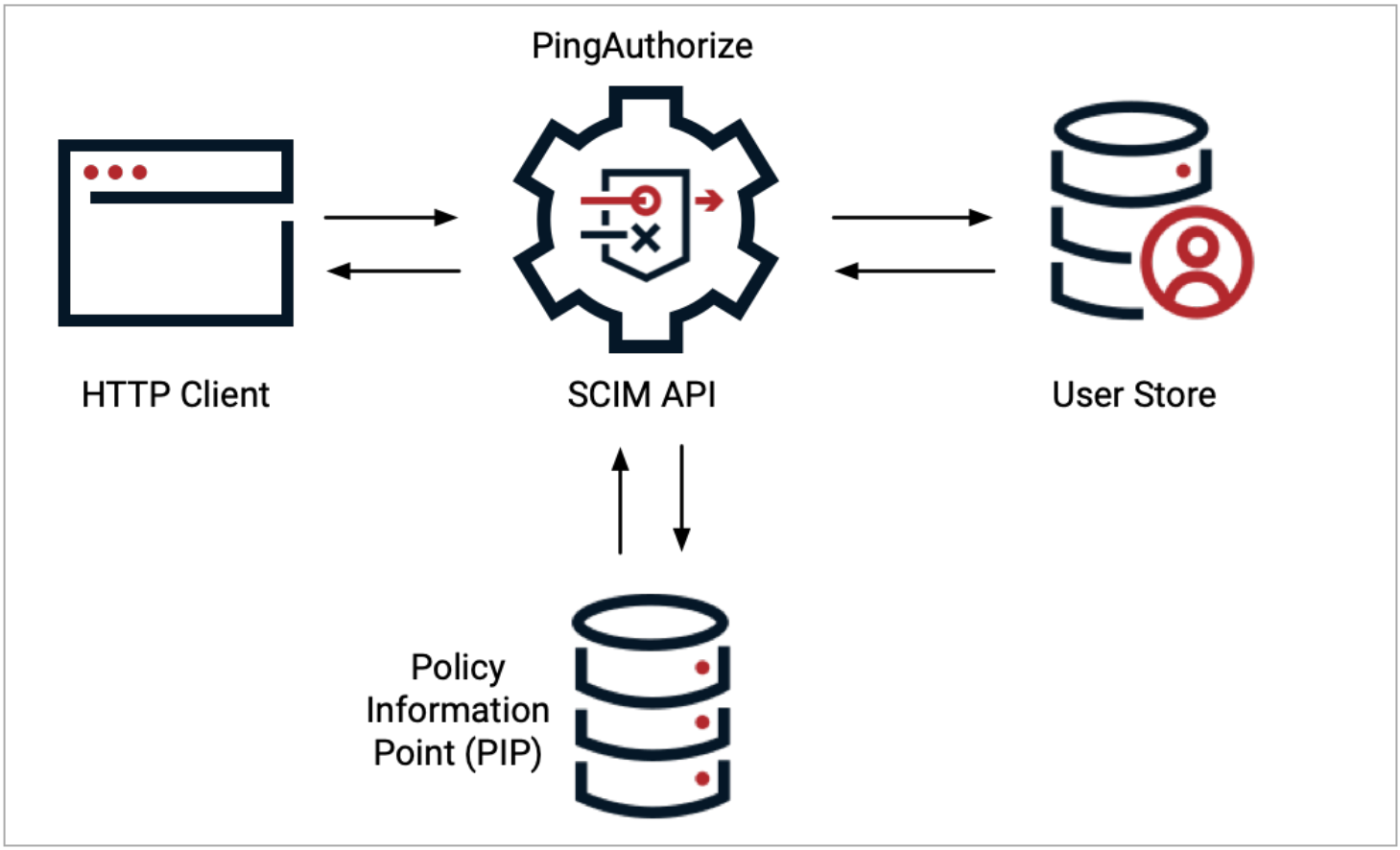
PingAuthorize Server supports the following implementation and data flow architectures for enforcing fine-grained access to data:

- SCIM API to datastores
- API security gateway as reverse proxy
- API security gateway in sideband configuration
- PDP APIs for non-API use cases

The following sections describe these architectures in more detail.

SCIM API to datastores

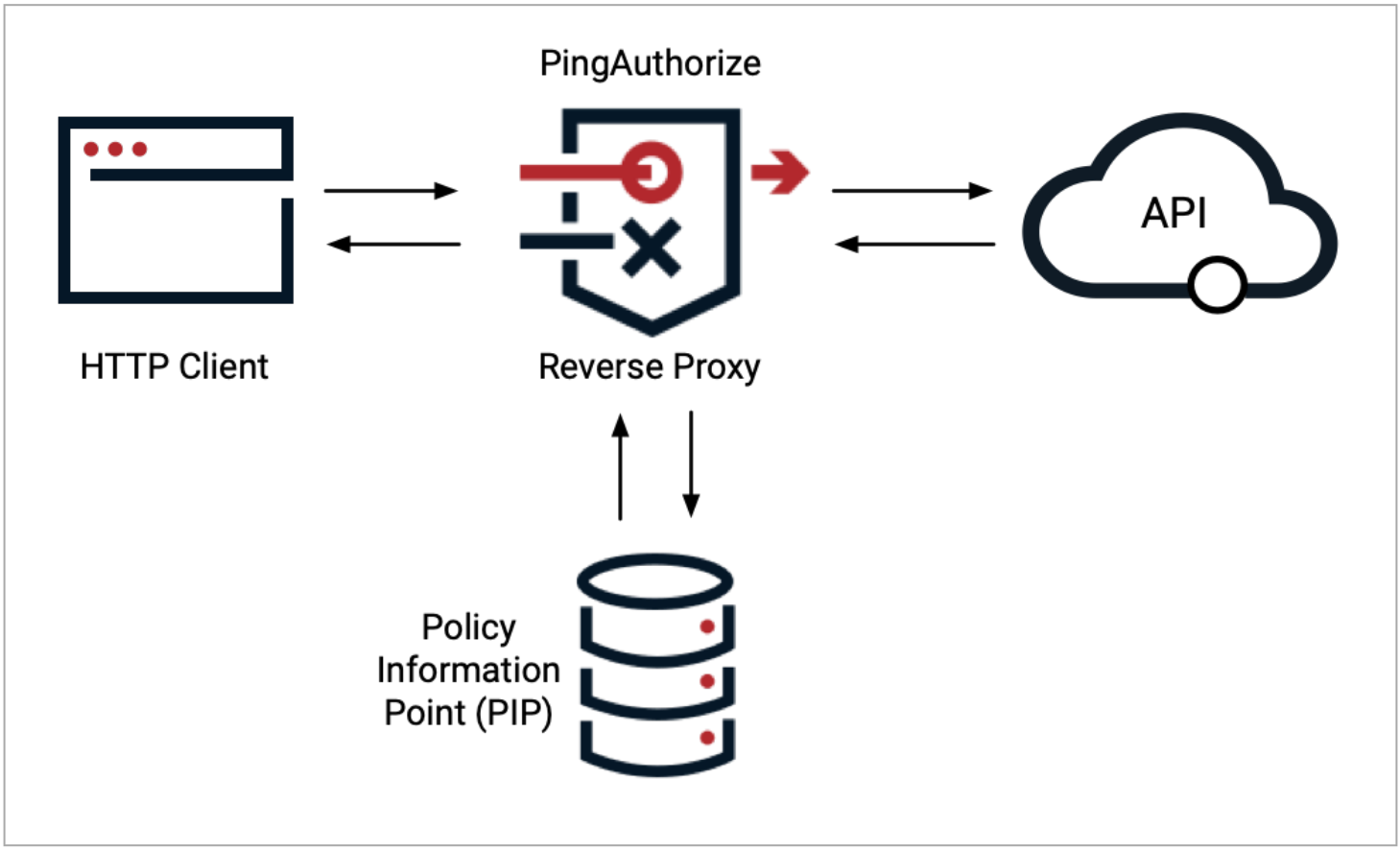
The PingAuthorize Server System for Cross-domain Identity Management (SCIM) service provides a REST API for data that is stored in one or more external datastores, based on the SCIM 2.0 standard. Authorization policies are enforced by the SCIM service. Learn more in SCIM API request and response flow.



API security gateway as reverse proxy

You can configure PingAuthorize Server's API security gateway as a reverse proxy to an existing JSON REST API. In this architecture, PingAuthorize Server acts as an intermediary between clients and existing API services. Authorization policies are enforced by the API security gateway. Learn more in API gateway request and response flow.

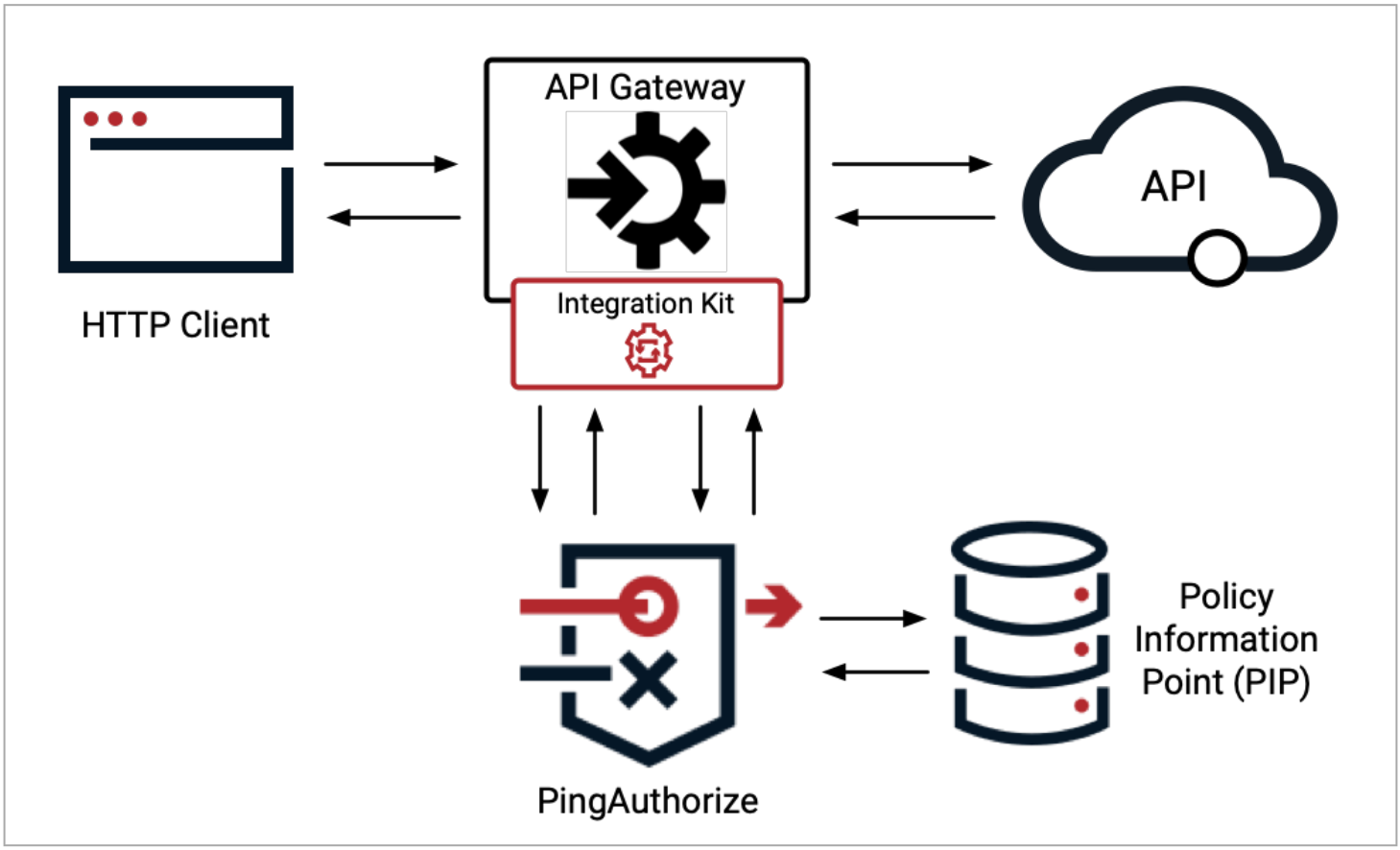
Copyright © 2025 Ping Identity Corporation



API security gateway in sideband configuration

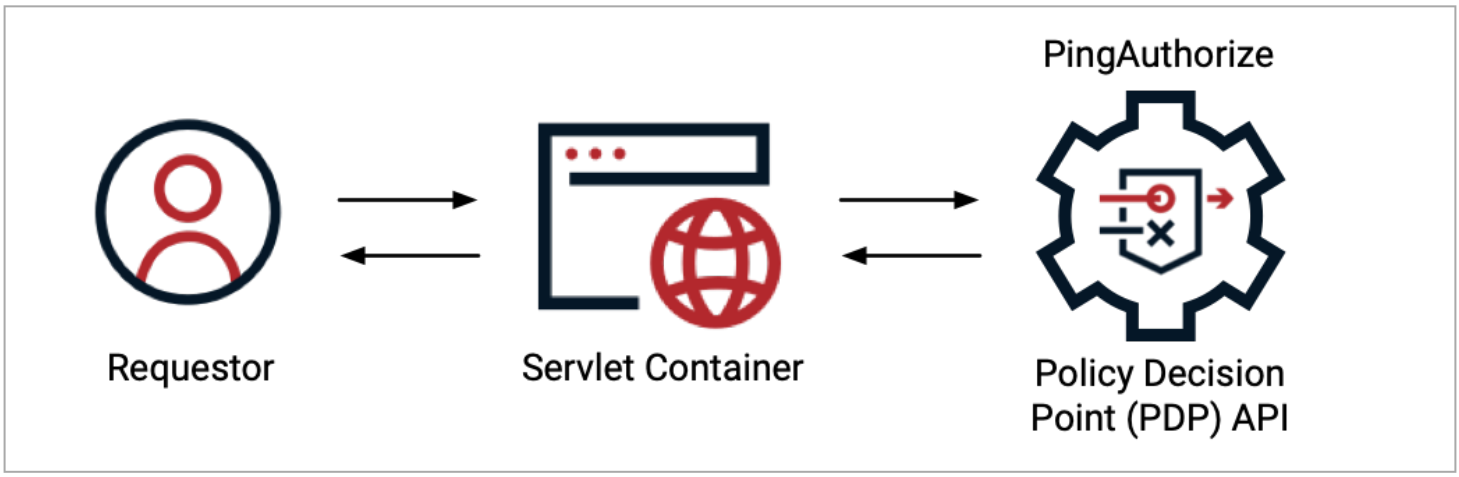
You can configure the PingAuthorize Server's API security gateway as an extension to an existing API lifecycle management gateway, commonly known as a sideband configuration. In this architecture, the API lifecycle management gateway functions as the intermediary between clients and existing API services and enforces policy decisions made by the PingAuthorize Server. Learn more in API gateway integration.

Copyright © 2025 Ping Identity Corporation



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. Learn more about Authorization Policy Decision APIs.



Policy decision environments

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

Copyright © 2025 Ping Identity Corporation

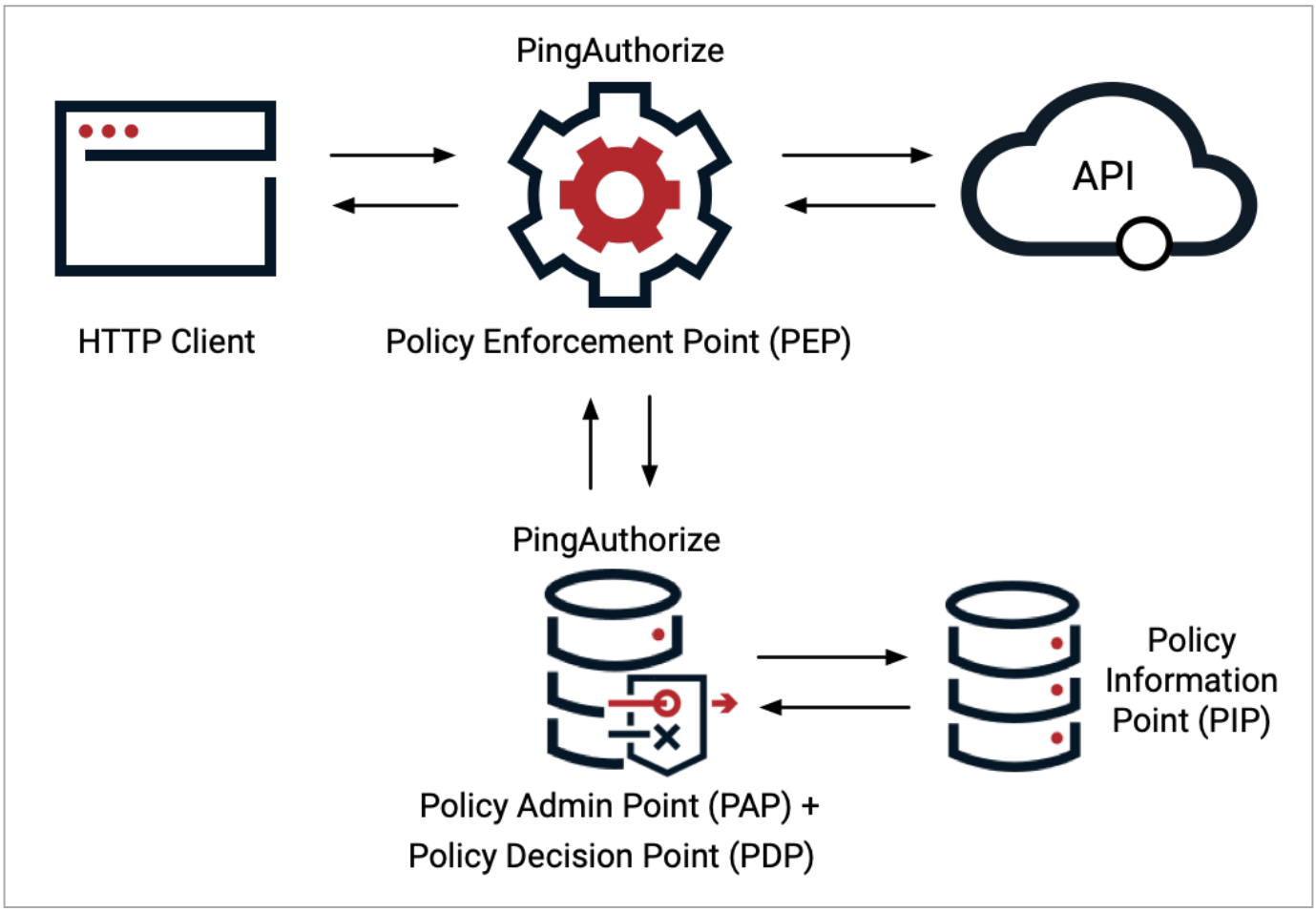
Development environment (external PDP)

Policies are developed in the Policy Editor. The Policy Editor serves as the external policy decision point (PDP), and the PingAuthorize Server serves as the policy enforcement point (PEP).

This configuration accelerates policy deployment by putting the decision point closer to your policy updates, making it ideal for development or testing environments.

> **Important**
>
> External PDP mode is not intended for use in production environments.

The following image shows PingAuthorize Server configured in a reverse proxy architecture. The development environment supports all PingAuthorize implementation and data flow architectures.



Learn more about configuring PingAuthorize for development environments in Configuring external PDP mode.

Pre-production and production environments (embedded PDP)

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

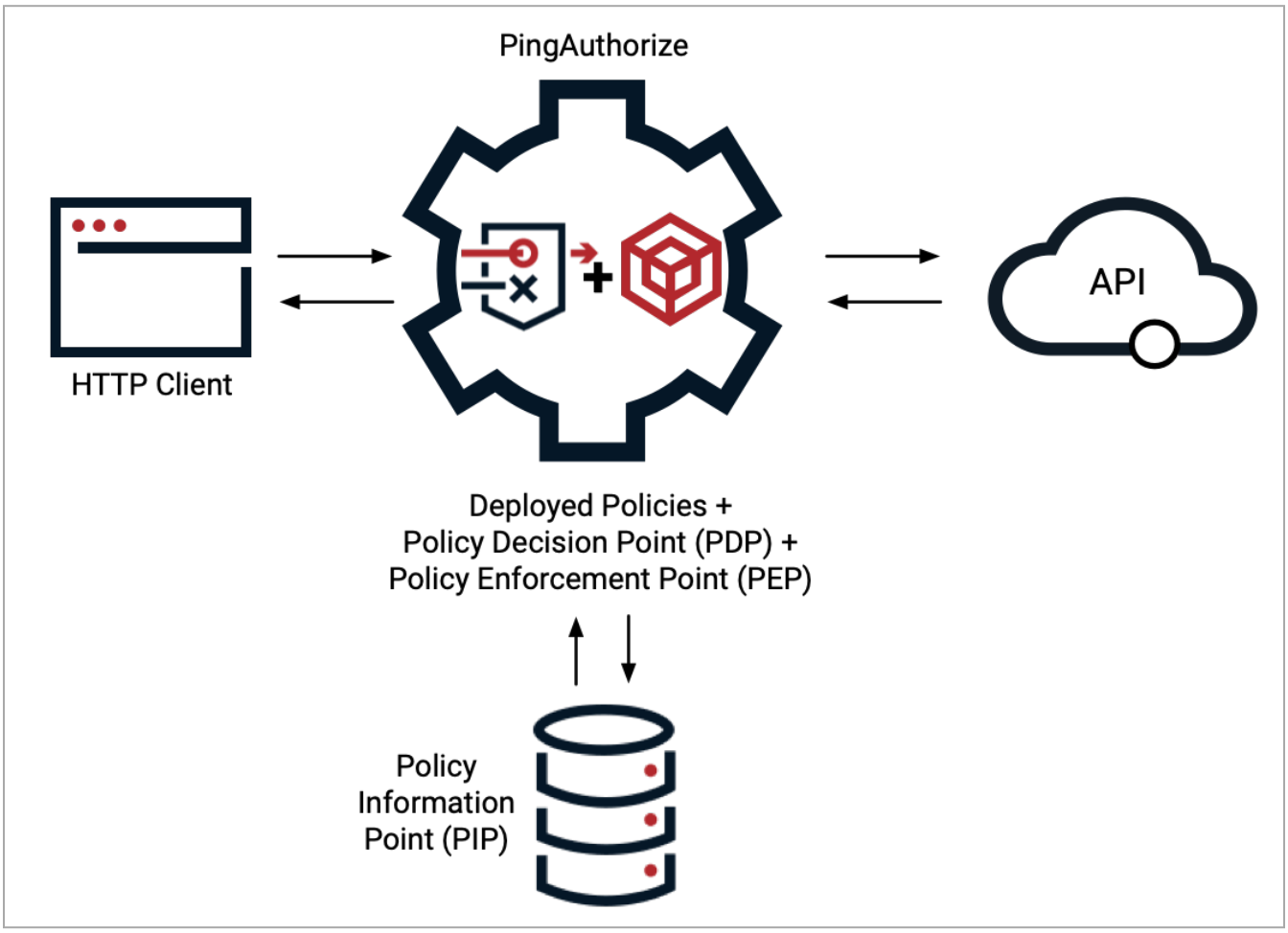
Copyright © 2025 Ping Identity Corporation

In these environments, the policy administrator bundles policies into a deployment package and then publishes them to the PingAuthorize Server. Embedding the policies in the PingAuthorize Server reduces latency in the decision request-response flow.

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

- Export the deployment package as a static file.
- Publish to a cloud package store (AWS S3, Azure) with the Deployment Manager.
- Publish to a file system package store with the Deployment Manager.

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.



Learn more about configuring PingAuthorize for production environments in Configuring embedded PDP mode.

Copyright © 2025 Ping Identity Corporation

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 these tutorials, 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 a REST application programming interface (API) and System for Cross-domain Identity Management (SCIM) resources.

Using the Tutorials

- Setting up your environment
- Starting PingAuthorize
- Verifying proper startup
- Accessing the GUIs
- Stopping PingAuthorize

PingAuthorize Tutorials

- Importing default policies
- Configuring fine-grained access control for an API
- Configuring attribute-based access control for API resources
- Creating SCIM policies

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.

Copyright © 2025 Ping Identity Corporation

- Increase your Docker memory limit to at least 4 GB.

  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 request, 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 an `.env` file in place for the environment to start.

Starting PingAuthorize

Start the Docker Compose environment.

Steps

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

Copyright © 2025 Ping Identity Corporation

```
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

PingAuthorize has two GUIs:

- Administrative console
- Policy Editor

> **Tip**
>
> If you have problems connecting because of self-signed certificates, click **Advanced** or 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

Copyright © 2025 Ping Identity Corporation

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.

◦ 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

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 standard installations, use persistent volumes to maintain data across container deployments.

Steps

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

```
docker-compose down
```

About the tutorial configuration

We provide the following pre-configured Docker containers through the Docker Compose environment, so that you can develop policies immediately.

Copyright © 2025 Ping Identity Corporation

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 protect your data.
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.

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 application programming interface (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.

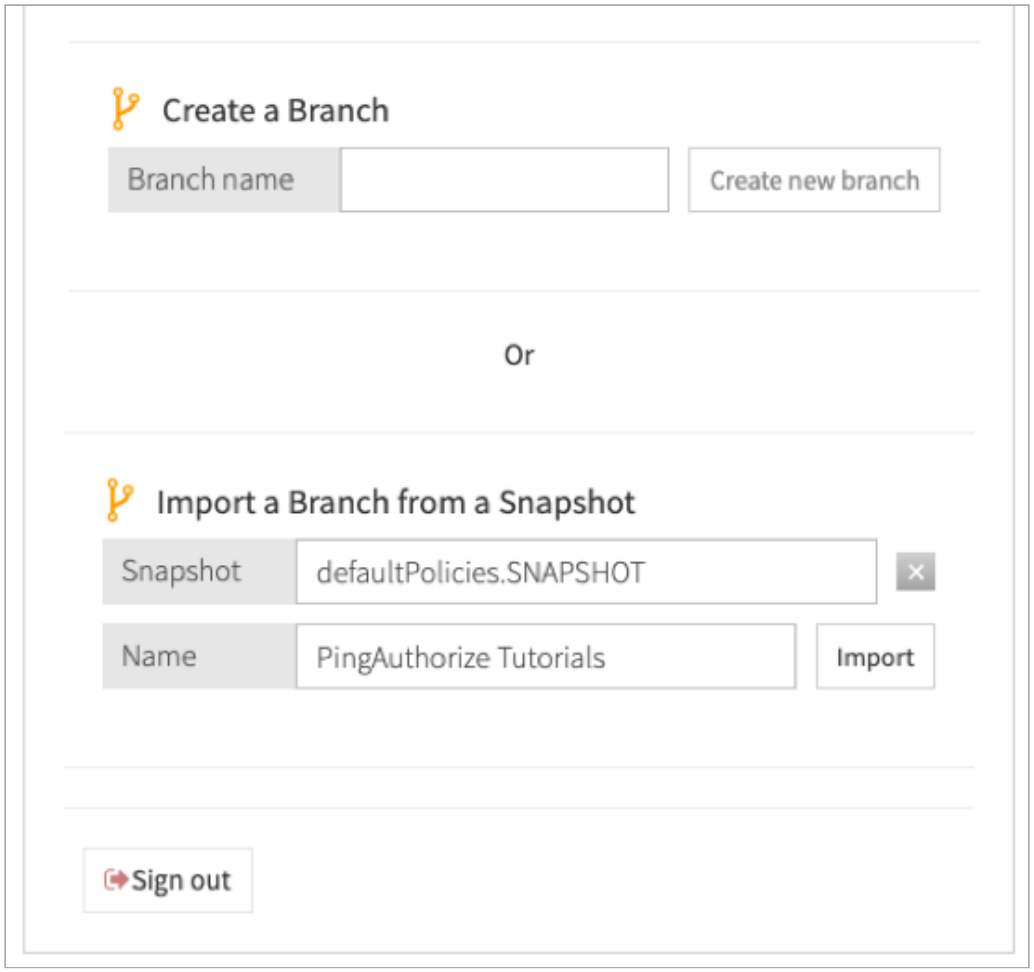
   Note

   Be sure to include the trailing `.` character.

Copyright © 2025 Ping Identity Corporation

```
docker cp  {PAP_CONTAINER_NAME}:/opt/out/instance/resource/policies/defaultPolicies.SNAPSHOT .
```

2. Sign on to the Policy Editor using the URL and credentials from Accessing the GUIs.
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:

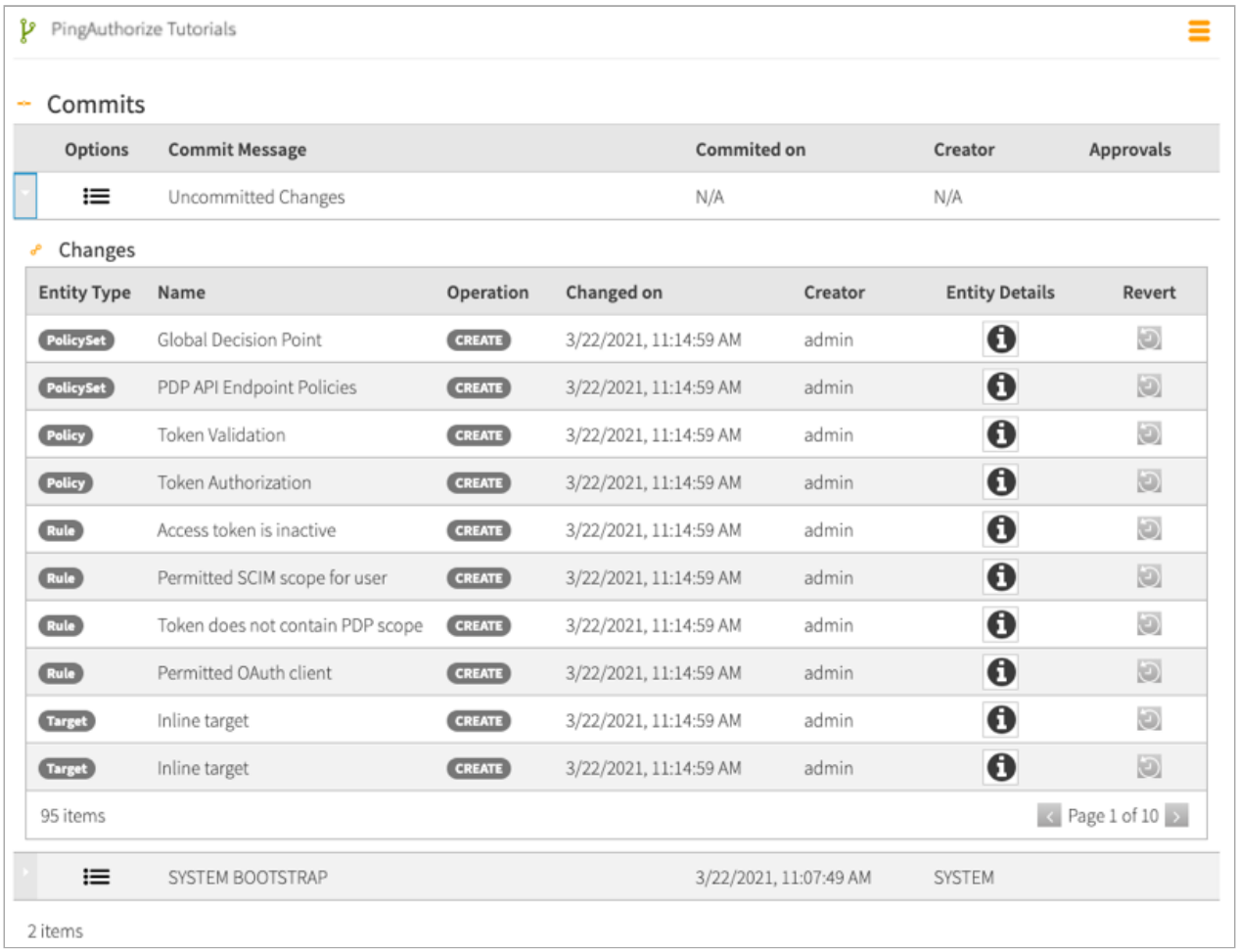
   A **Commits** table opens. This table provides a log of all changes made to a policy branch.

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

   Result:

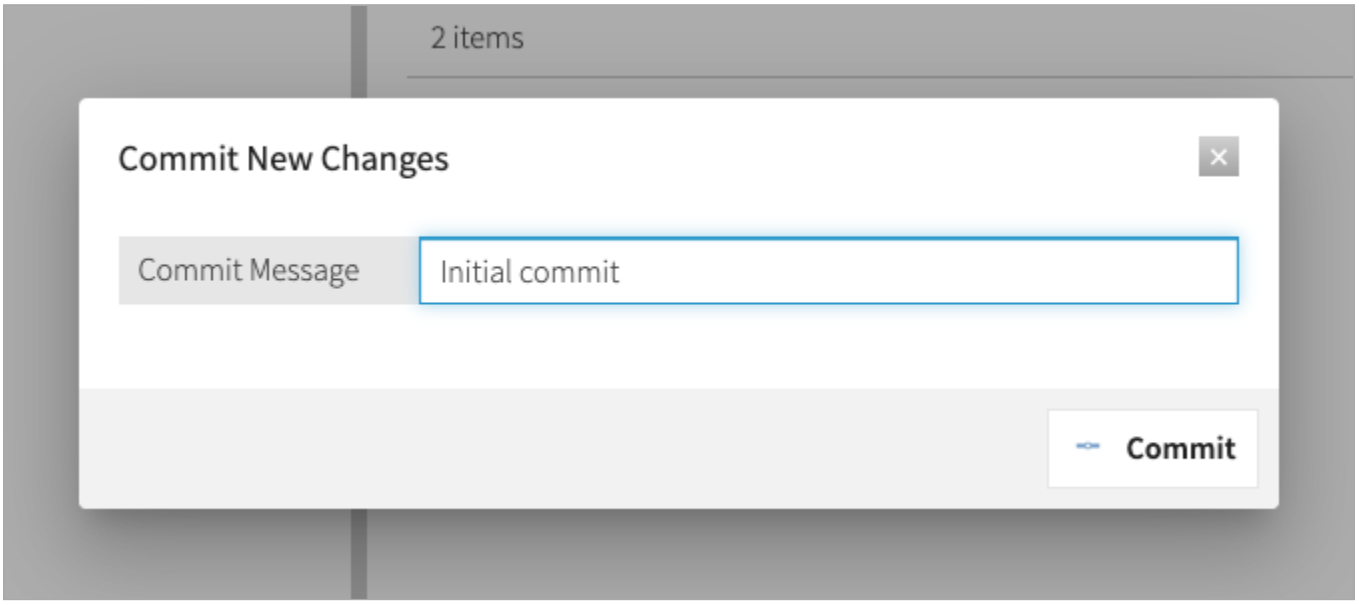
Copyright © 2025 Ping Identity Corporation

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**.

Copyright © 2025 Ping Identity Corporation



Tip

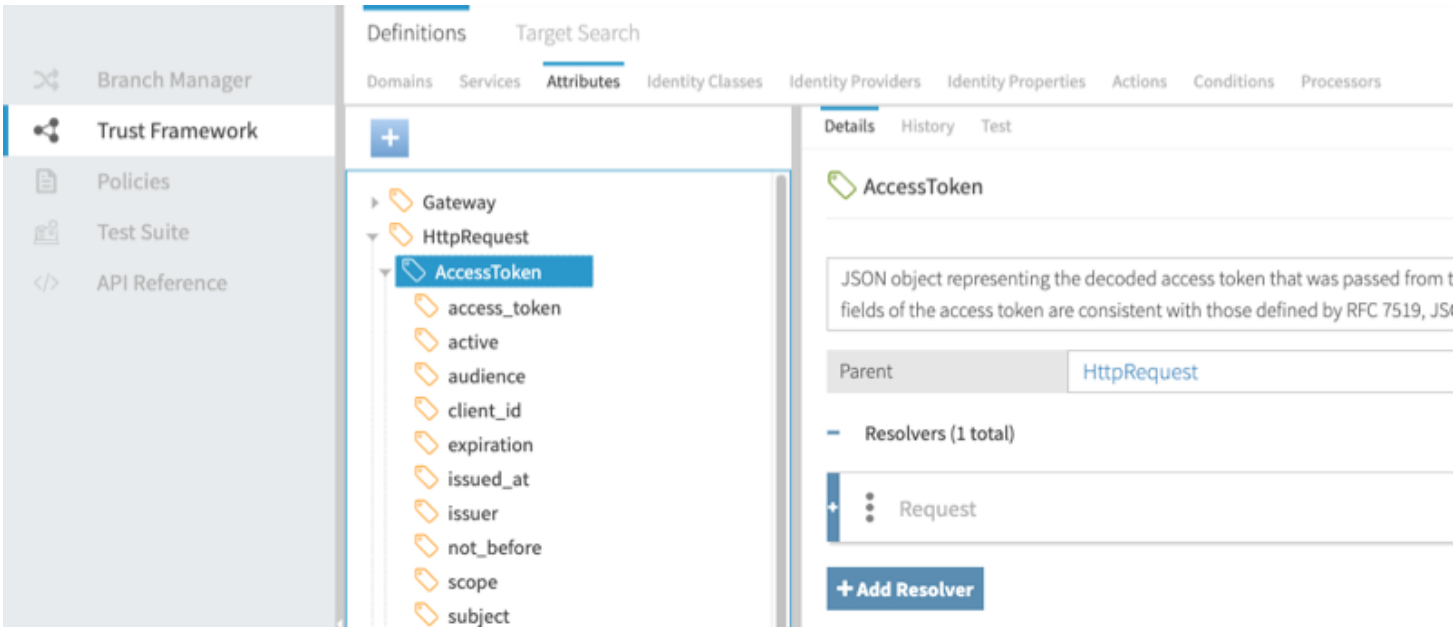
As you develop your own policies, you can use the Policy Editor's version control feature to manage your changes. Follow best practices by committing your changes every time you achieve a satisfactory working state.

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.



Copyright © 2025 Ping Identity Corporation

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 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).

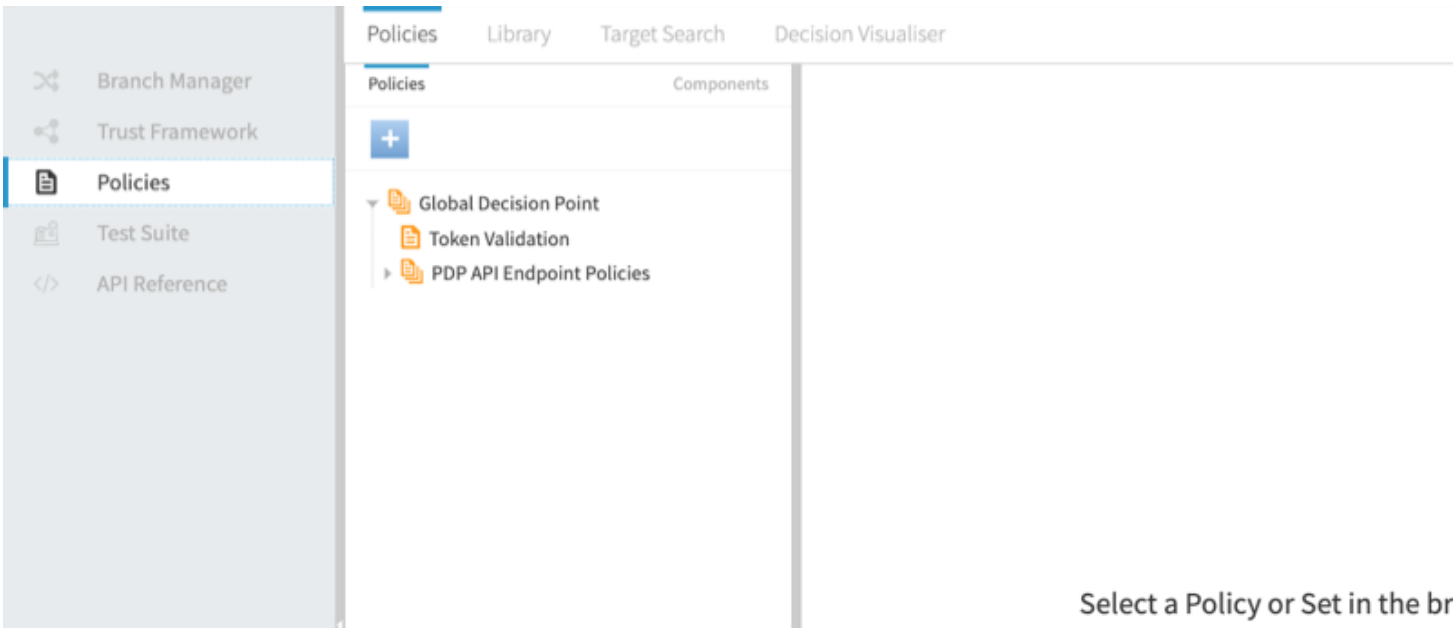
Look at the Trust Framework's default attributes and consider how you could use them in your own policies. Some important Trust Framework attributes include those in the following table.

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.

Copyright © 2025 Ping Identity Corporation



You define your policies within a hierarchical tree that 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.

Copyright © 2025 Ping Identity Corporation

Library

The default policy library contains example statements 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 JavaScript Object Notation (JSON) REST application programming interface (API).

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

Access control granularity type	Description
Coarse-grained	Users or clients have access to all of or none of an application or API.
Medium-grained	Users or clients have access to some pages or resources within an application or API.
Fine-grained	Users or clients can take specific actions on an application page or an API resource when *action-specific conditions* are met. 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 using 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 above link directs you to the Meme Game API Git project, where you can build and run the API. You can access the API server with specific API paths, such as meme-game.com/api/v1/answers.

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.

Copyright © 2025 Ping Identity Corporation

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

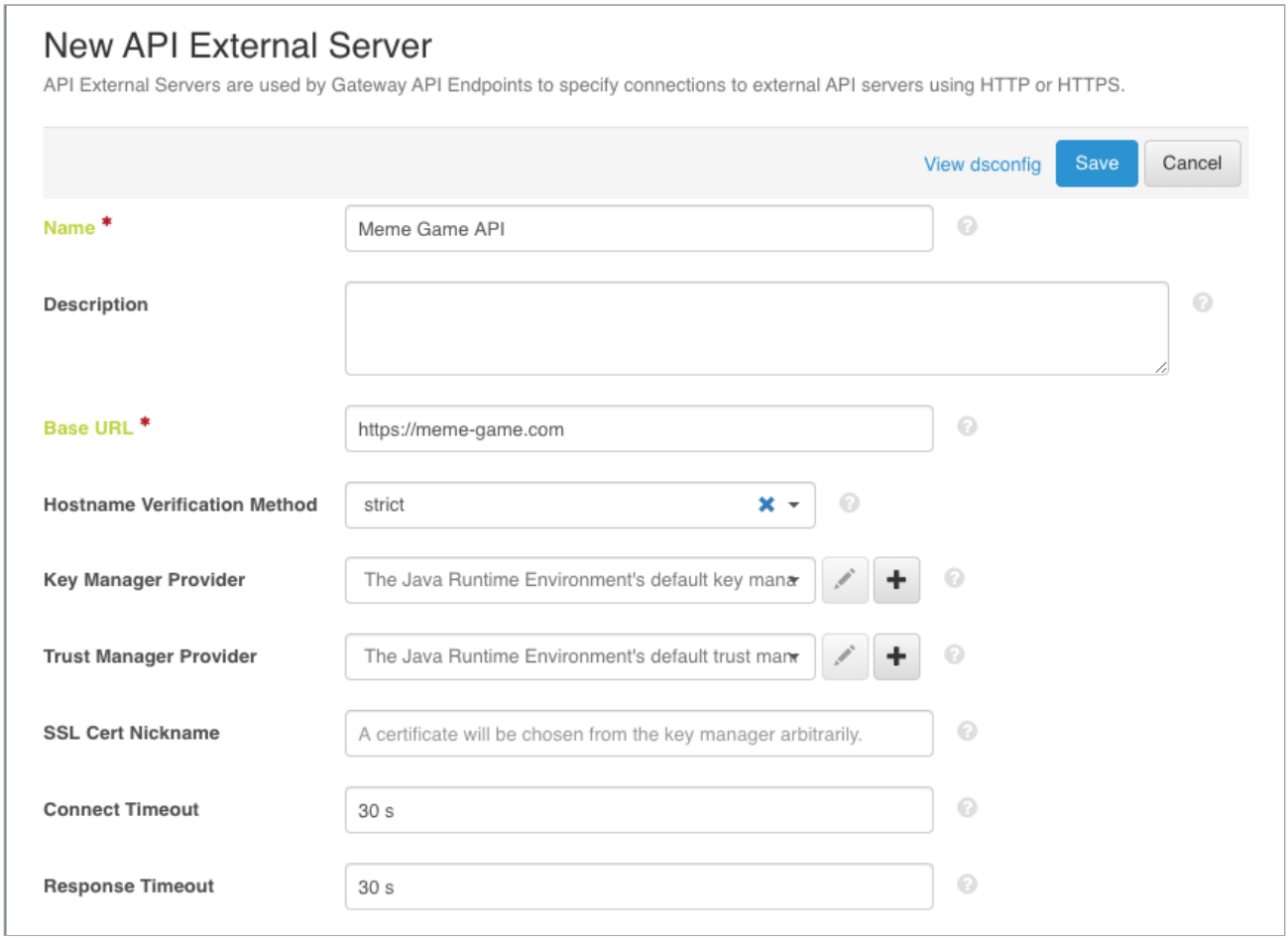
Configure a reverse proxy by configuring an API External Server and a Gateway API Endpoint. The API reverse proxy acts as an intermediary between your HTTP client and the HTTP API, providing fine-grained access control for the API.

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.
    1. Sign on to the administrative console using the URL and credentials from Accessing the GUIs.
    2. Click **External Servers**.
    3. Click **New External Server** and choose **API External Server**.
    4. For **Name**, specify `Meme Game API`.
    5. For **Base URL**, specify `https://meme-game.com`.

       The following image shows this configuration.

Copyright © 2025 Ping Identity Corporation



6. 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.

    1. In the administrative console, click **Configuration** and then **Gateway API Endpoints**.
    2. Click **New Gateway API Endpoint**.
    3. For **Name**, specify `Meme Game - Games`.
    4. For **Inbound Base Path**, specify `/meme-game/api/v1/games`.

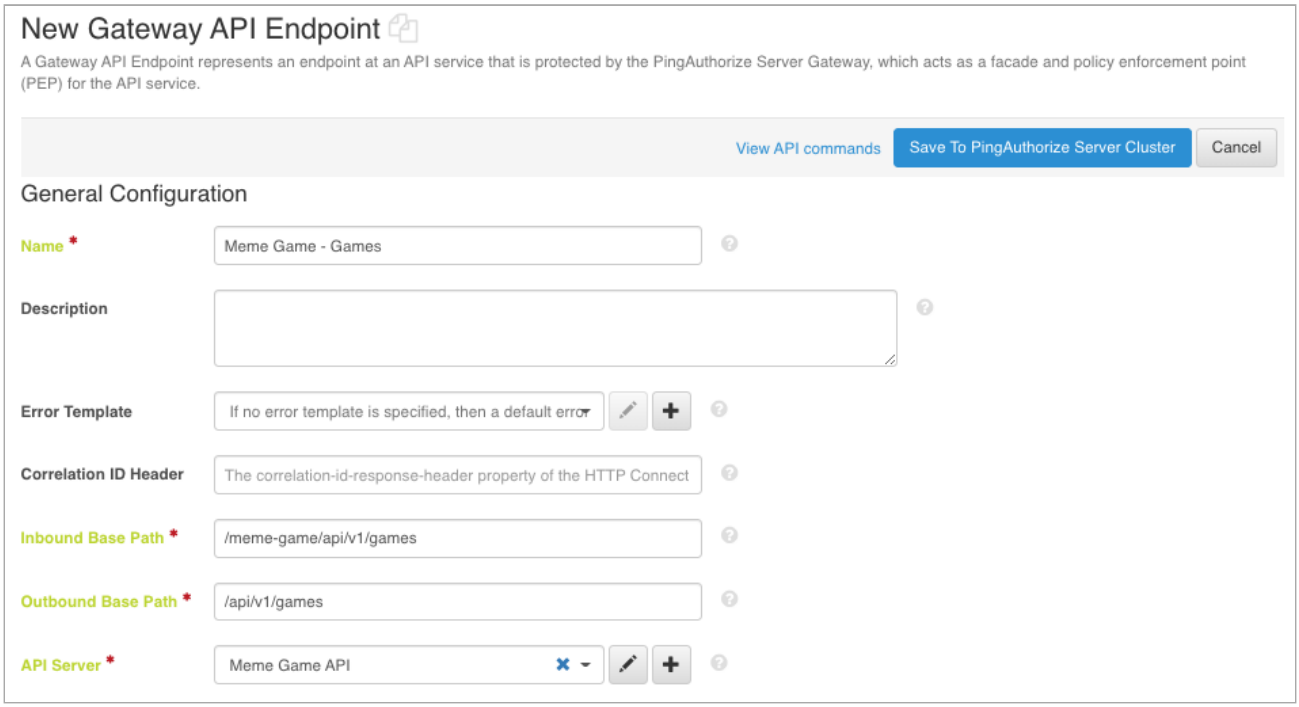
        The inbound base path defines the base request path for requests to be received by PingAuthorize Server.

    5. 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.

    6. For **API Server**, specify `Meme Game API`. This is the API External Server you defined previously.

Copyright © 2025 Ping Identity Corporation



7. 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.

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`**.

Copyright © 2025 Ping Identity Corporation

```
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. Learn more about this type of authorization in For further consideration: The PingAuthorize API security gateway, part 1.

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 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.

Copyright © 2025 Ping Identity Corporation

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.

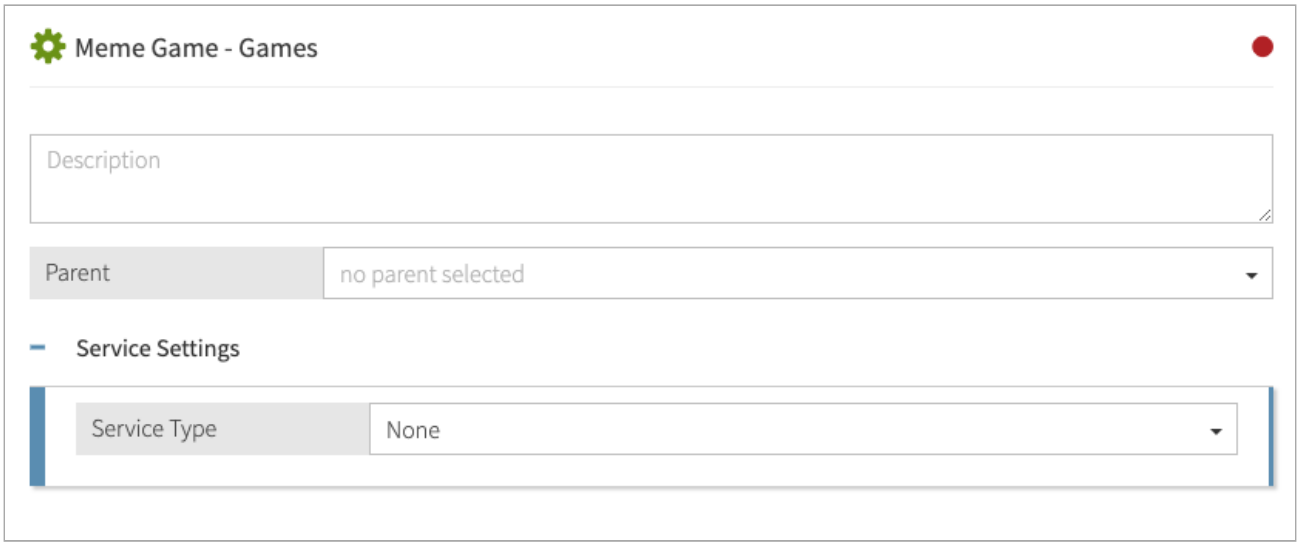
Then, we define a policy. This policy will reject any requests to start a new meme game if the user's identifier ends with `@example.com`. We will identify users using the subject of the request's access token.

Steps

1. Define the service.
    1. Sign on to the Policy Editor using the URL and credentials from Accessing the GUIs.
    2. Go to **Trust Framework** and click **Services**.
    3. From the **+** menu, select **Add new Service**.
    4. 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.
    5. 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.

    6. Click **Save changes**.

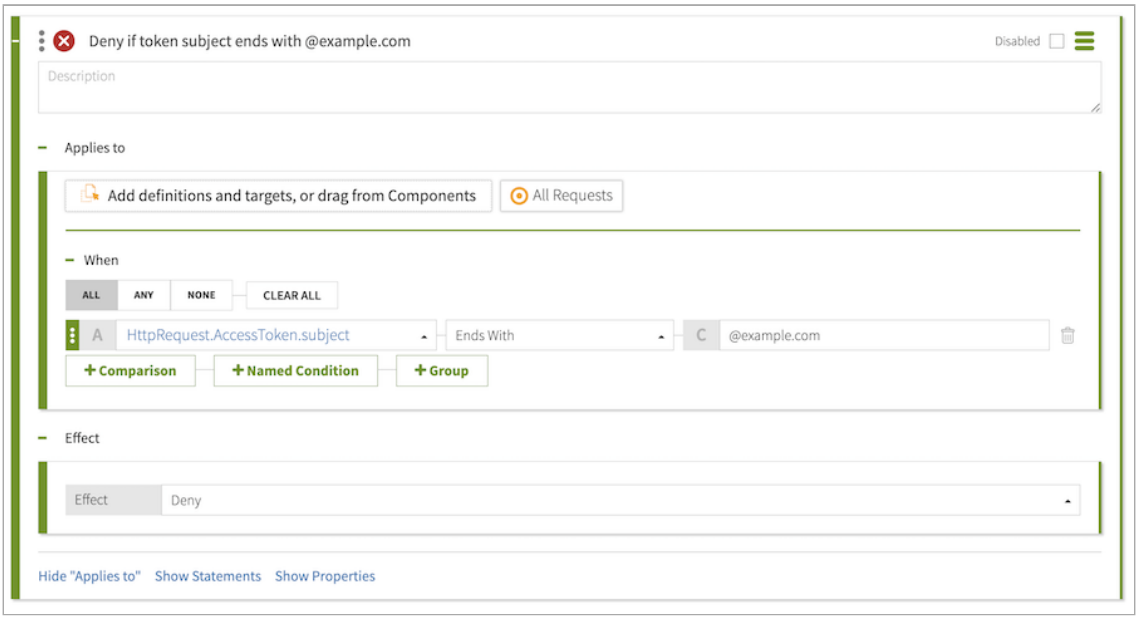
Copyright © 2025 Ping Identity Corporation

2. Define the policy.
    1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.
    2. Select **Global Decision Point**.
    3. From the + menu, select **Add Policy**.
    4. For the name, replace **Untitled** with `Users starting a new game`.
    5. Click + next to **Applies to**.
    6. 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.
    7. From the **Actions** list, drag **inbound-POST** to the **Add definitions and targets, or drag from Components** box.
    8. 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.
    9. Set the Combining Algorithm to **Unless one decision is deny, the decision will be permit**.
    10. 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.
            1. Click + **Comparison**.
            2. From the **Select an Attribute** list, select **HttpRequest.AccessToken.subject**.
            3. In the second field, select **Ends With**.
            4. In the third field, type `@example.com`.

               The following screen shows the rule.

Copyright © 2025 Ping Identity Corporation



11. Click **Save changes**.

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

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.

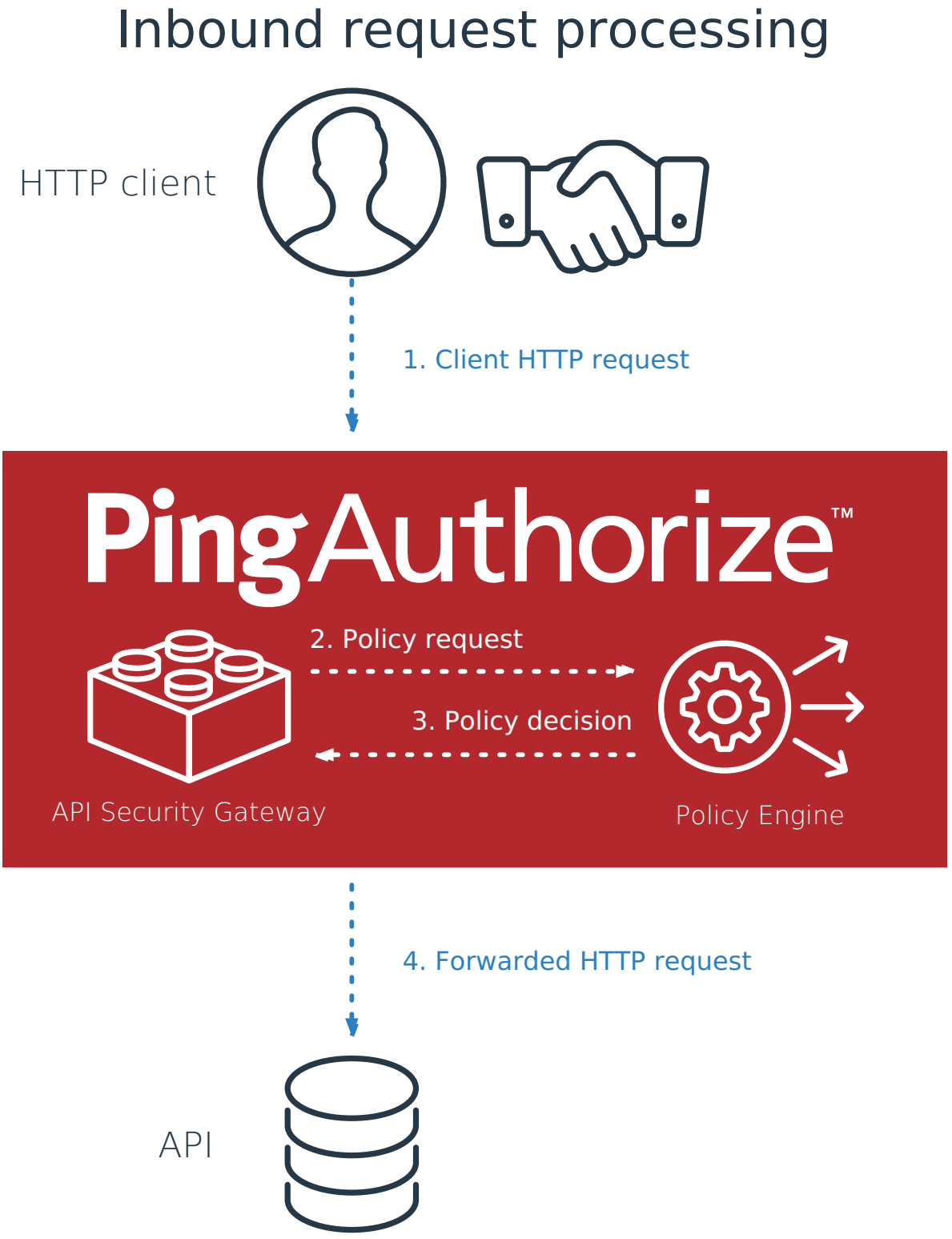
API security gateway processing occurs in two phases

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.

Copyright © 2025 Ping Identity Corporation

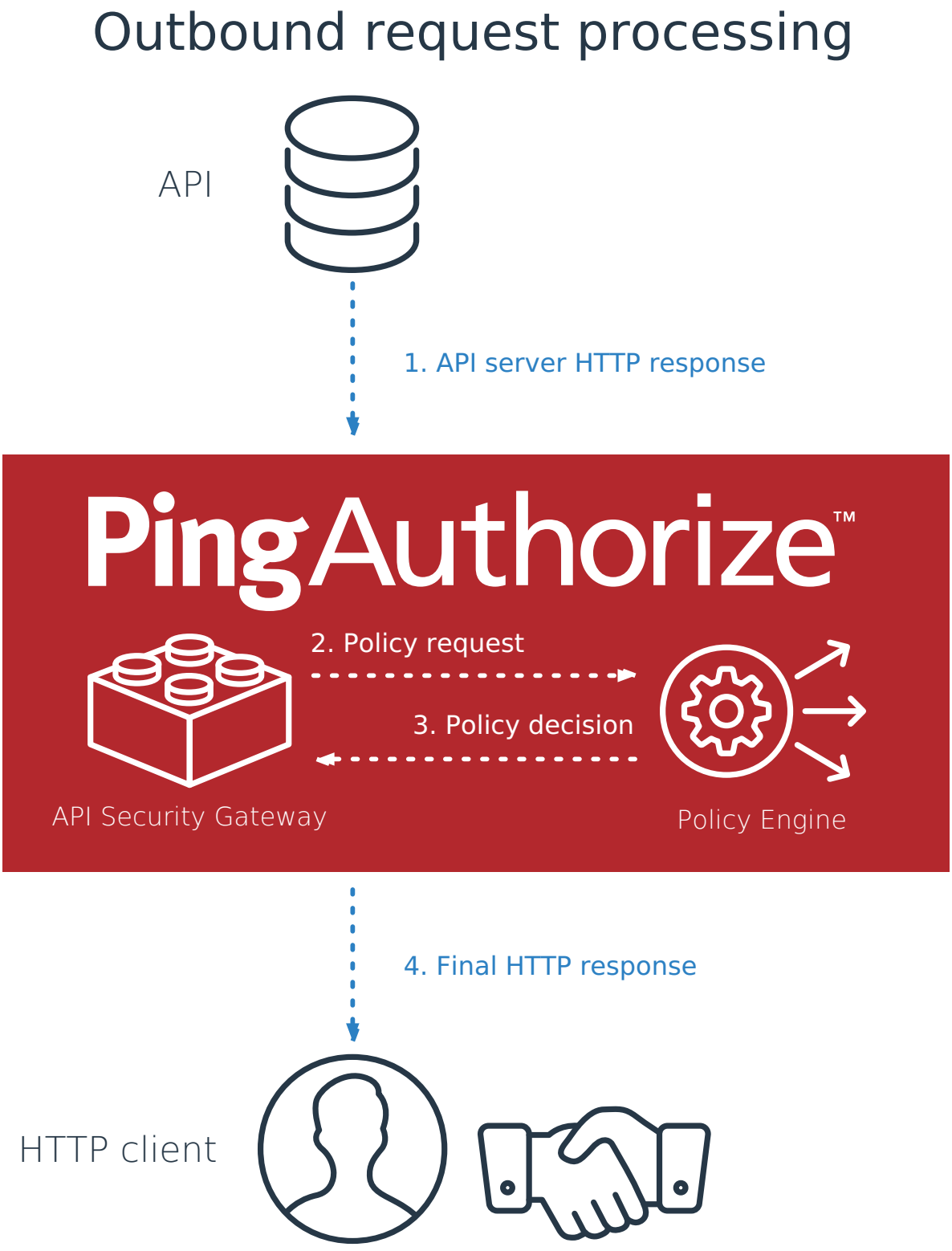


The outbound phase

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.

Copyright © 2025 Ping Identity Corporation

The following diagram illustrates the outbound request processing.



Copyright © 2025 Ping Identity Corporation

Service name must match Gateway API Endpoint name

In Adding a policy for the Create Game endpoint, 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.

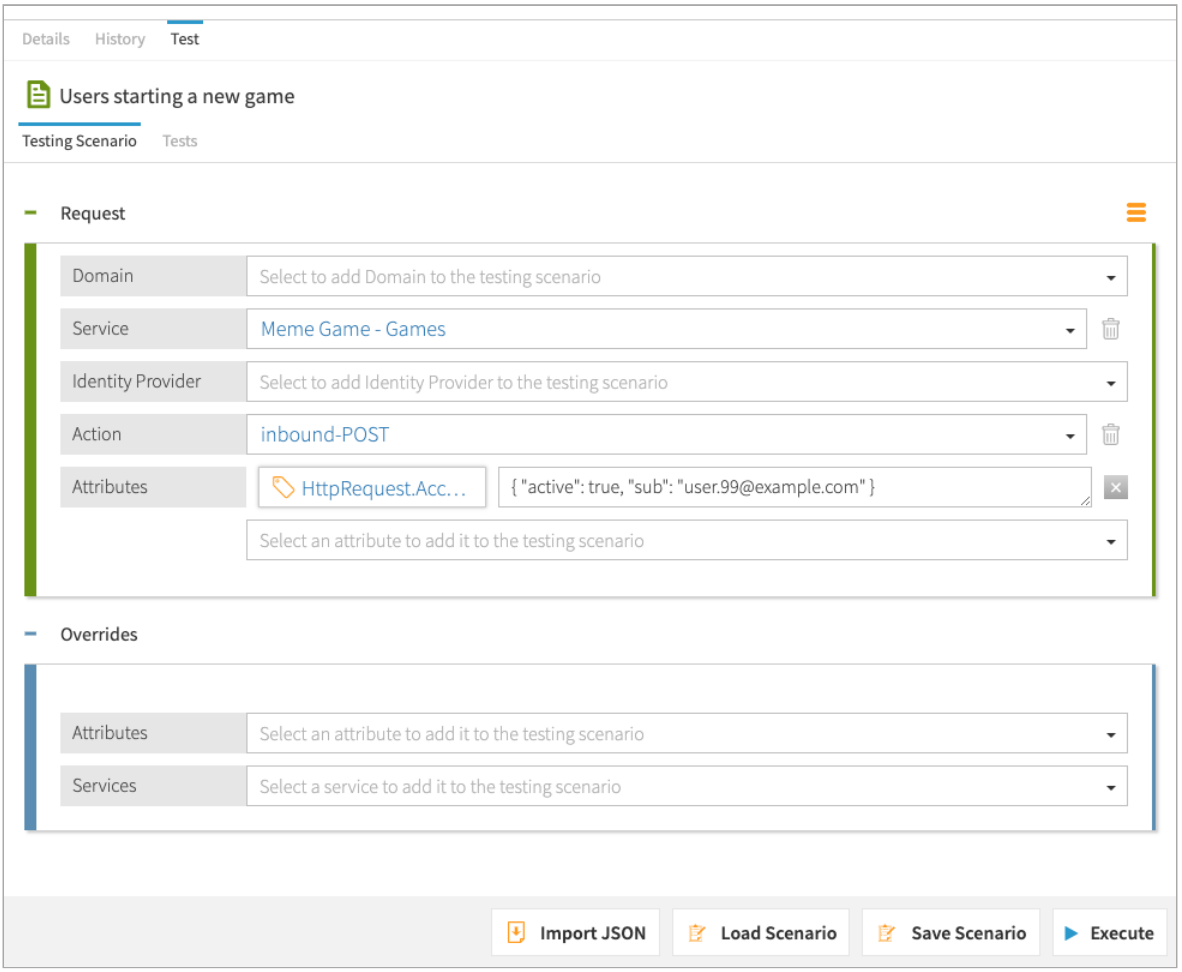
Steps

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.

Copyright © 2025 Ping Identity Corporation



3. Click **Execute**.

   Result:

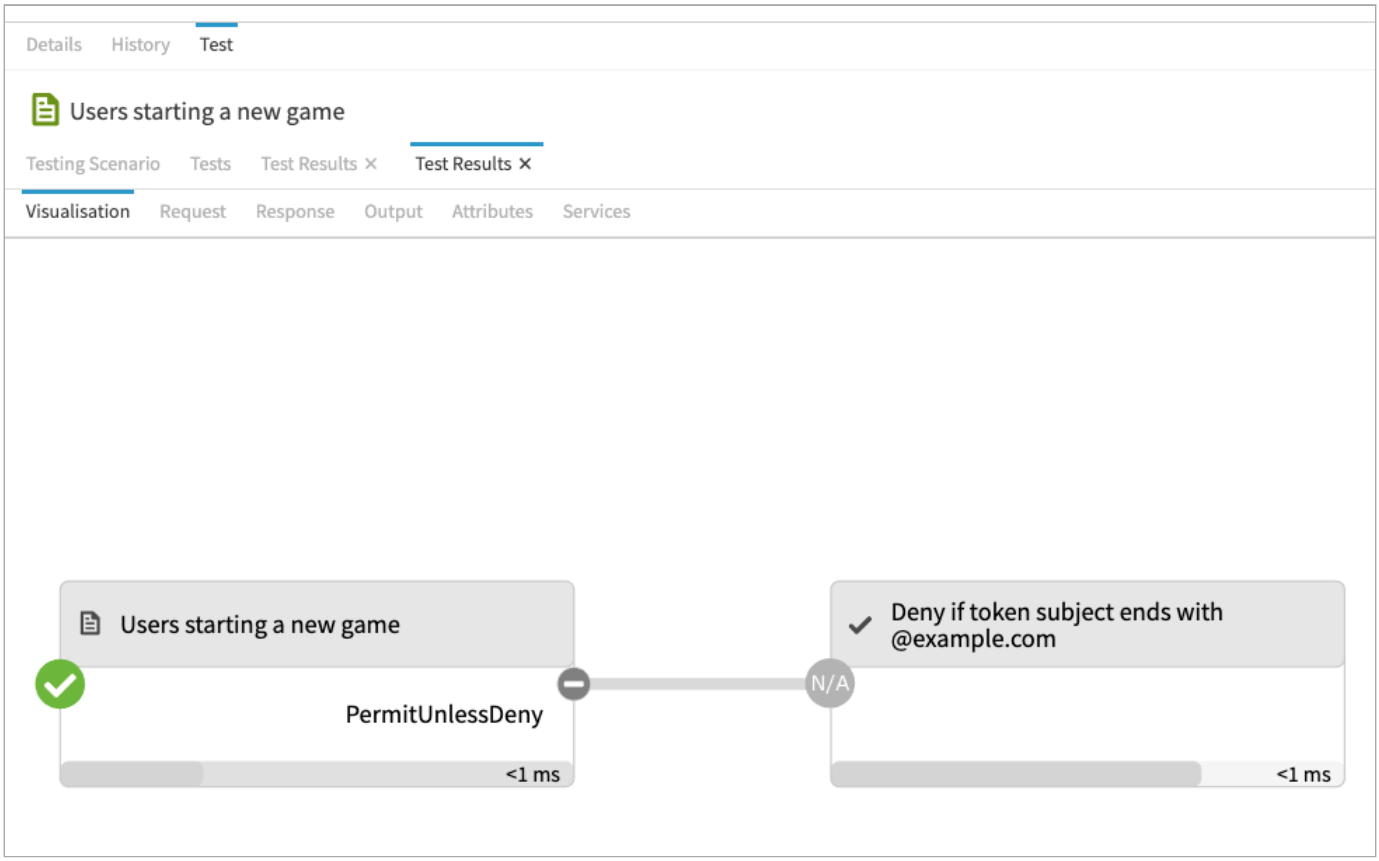
   The policy test result displays. If the policy worked as expected, the leftmost result is red, indicating a `DENY` result.

Copyright © 2025 Ping Identity Corporation

4. **Optional:** Experiment with testing.

   Click the **Testing Scenario** tab and try different inputs to see how the 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.

Copyright © 2025 Ping Identity Corporation



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`.

Copyright © 2025 Ping Identity Corporation

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.

For further consideration: Decision Visualiser

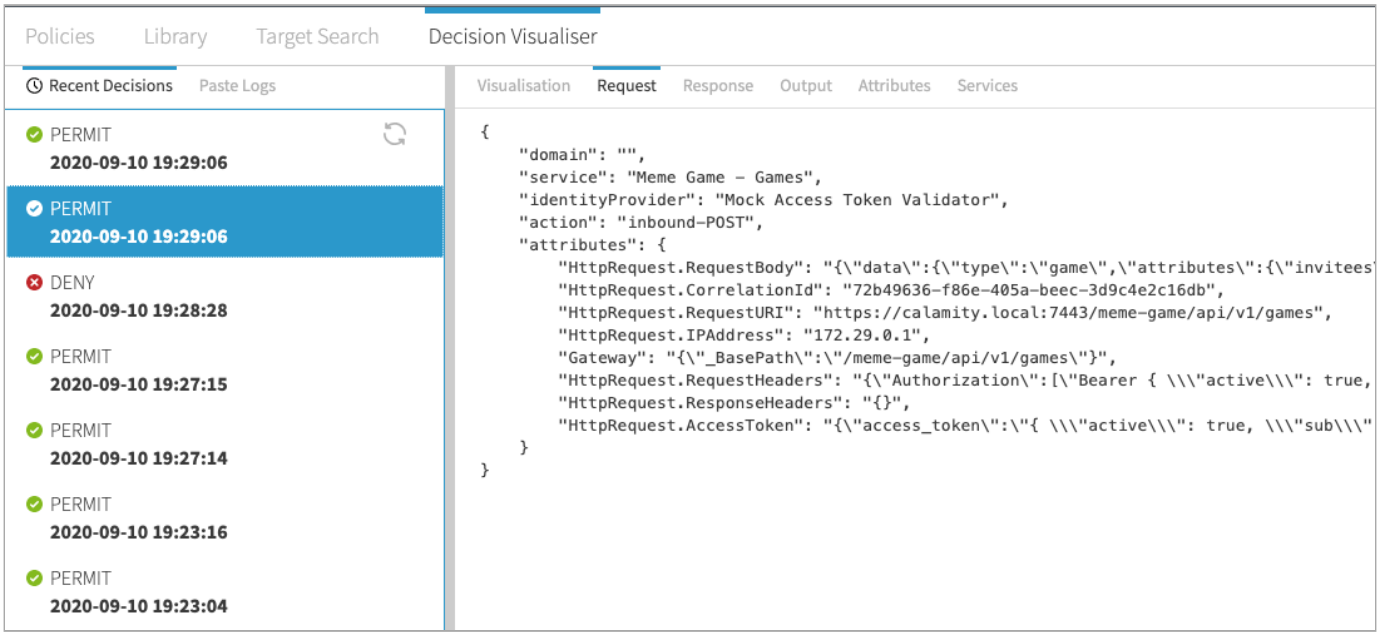
Returning to the Policy Editor, we can view a log of how the policy engine handled the HTTP request.

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.

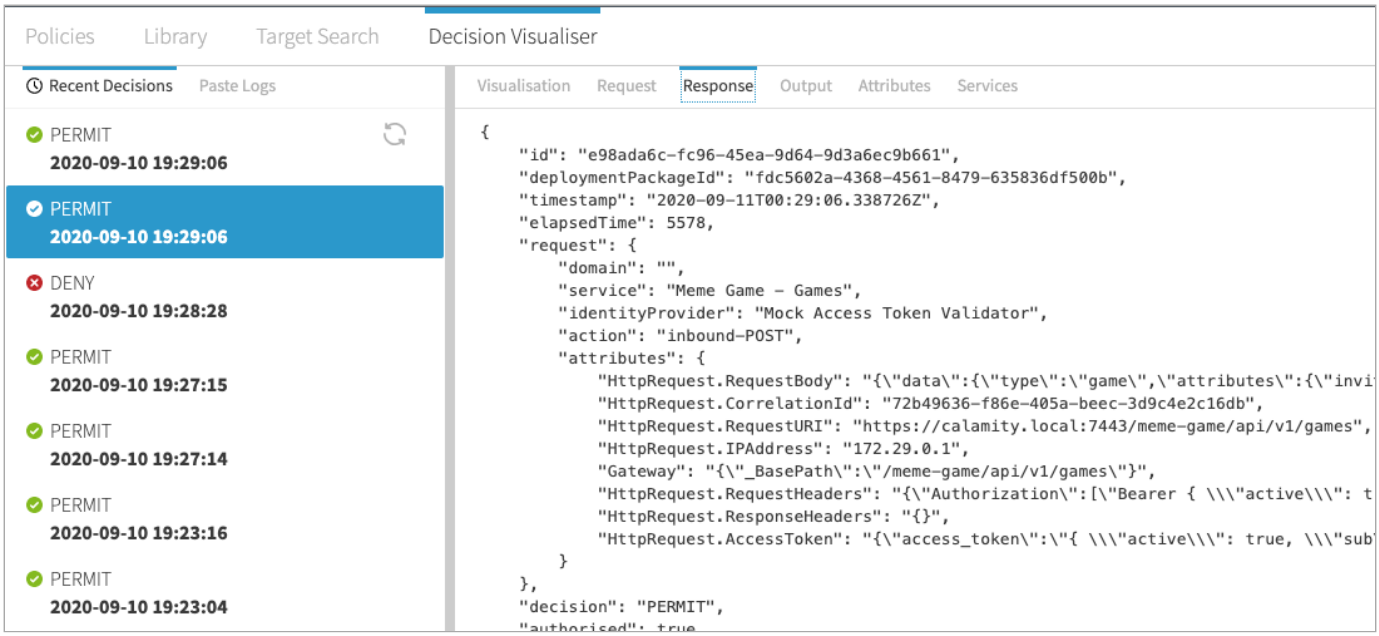
   The following image shows a request example:

Copyright © 2025 Ping Identity Corporation



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'll find that the Decision Visualiser is indispensable for troubleshooting and understanding your policies.

Copyright © 2025 Ping Identity Corporation

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.

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.

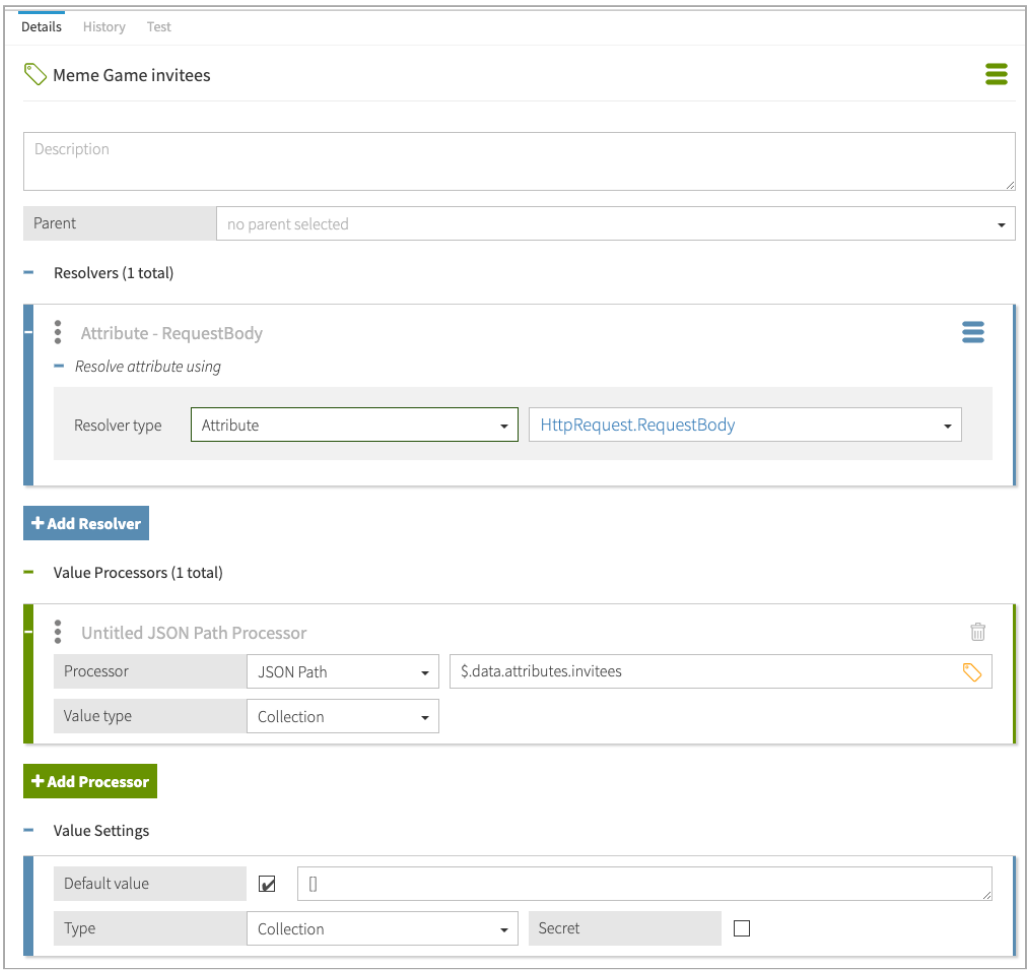
Steps

1. Define a Trust Framework attribute to represent the `data.attributes.invitees` field.
    1. In the 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 invitees`.
    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.
    5. Click the **+** next to **Resolvers** and click **+ Add Resolver**.
    6. Set **Resolver type** to **Attribute**.
    7. Select the attribute **HttpRequest.RequestBody**.
    8. Click the **+** next to **Value Processors** and click **+ Add Processor**.
    9. Set **Processor** to **JSON Path**.
    10. Set the value to `$.data.attributes.invitees`.
    11. Set **Value type** to **Collection**.
    12. For **Value Settings**, select **Default value** and specify square brackets (`[]`) to indicate an empty collection.
    13. Set **Type** to **Collection**.
    14. Click **Save changes**.

Copyright © 2025 Ping Identity Corporation

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.

2. Modify a rule to use the **Meme Game invitees** attribute we just created.

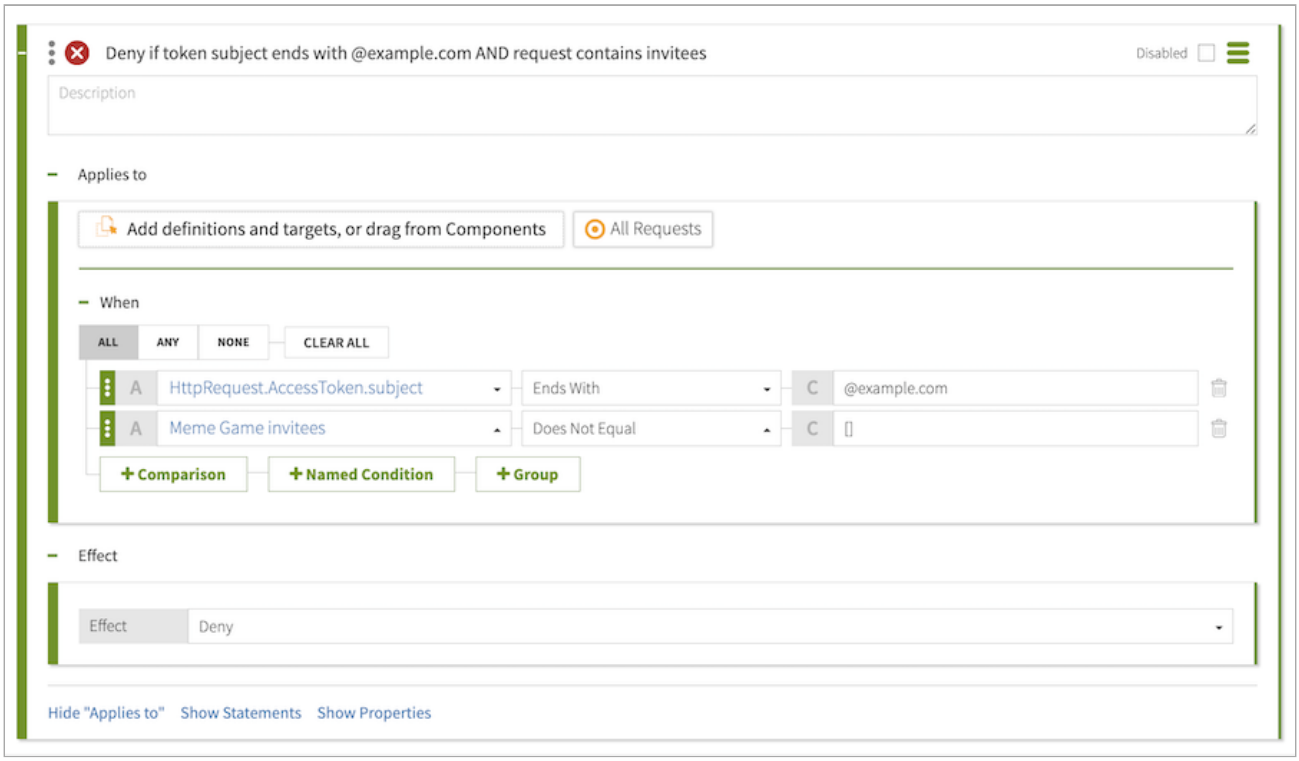
    1. In the Policy Editor, go to **Policies**.
    2. Select the **Users starting a new game** policy.
    3. Rename the **Deny if token subject ends with @example.com** rule to `Deny if token subject ends with @example.com AND request contains invitees`.
    4. Expand the rule by clicking its **+** icon.
    5. For **Effect**, select **Deny**.
    6. 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**.

Copyright © 2025 Ping Identity Corporation

4. In the third field, type `[]`.

7. Click **Save changes**.

   The following image shows the rule.



3. Test the policy.

   As before, you can test your policy by sending an HTTP request or using the Policy Editor test interface. 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 introduces their use. Here is more about how you use them in your policies.

Copyright © 2025 Ping Identity Corporation

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 JSONPath 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 protected 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.

Copyright © 2025 Ping Identity Corporation

> **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 for ages under 13.
5. Add a statement to set the API error response when a 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

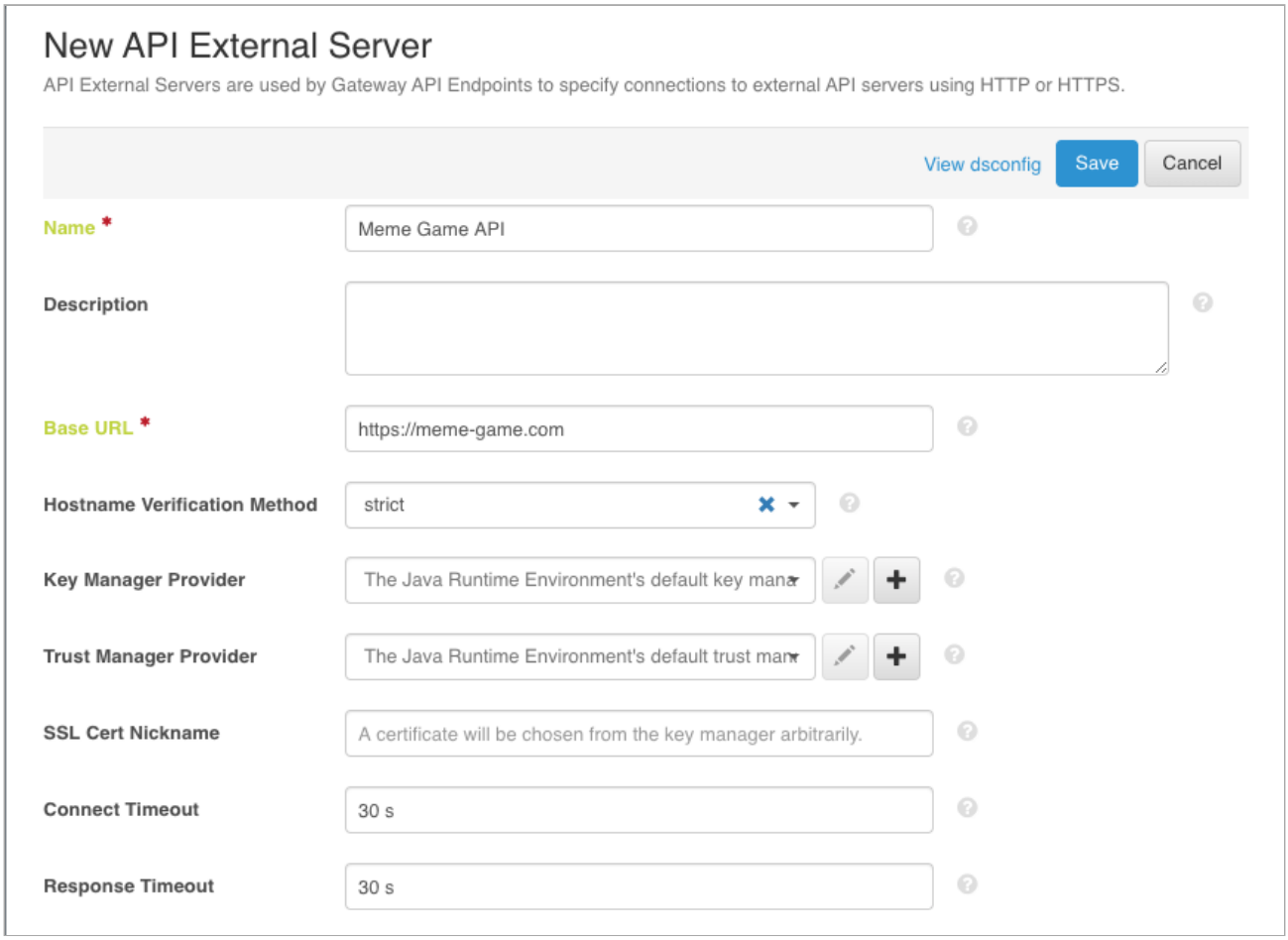
1. (Optional) Configure an API External Server for the Meme Game API. An API External Server controls how PingAuthorize Server handles connections to an HTTPS API server, including configuration related to TLS. In this case, we simply need to provide a base URL.

   > **Note**
   >
   > If you completed Tutorial 2: Configuring fine-grained access control for an API, then you already set up this API External Server.

   1. Sign on to the administrative console using the URL and credentials from Accessing the GUIs.
   2. Click **External Servers**.
   3. Click **New External Server** and choose **API External Server**.
   4. For **Name**, specify `Meme Game API`.
   5. For **Base URL**, specify `https://meme-game.com`.

      The following image shows this configuration:

Copyright © 2025 Ping Identity Corporation



6. 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.

    1. In the administrative console, click **Configuration** and then **Gateway API Endpoints**.
    2. Click **New Gateway API Endpoint**.
    3. For **Name**, specify `Meme Game - Shared Answers`.
    4. 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. You can also use this parameter to inform other elements of the policy request, such as the service.

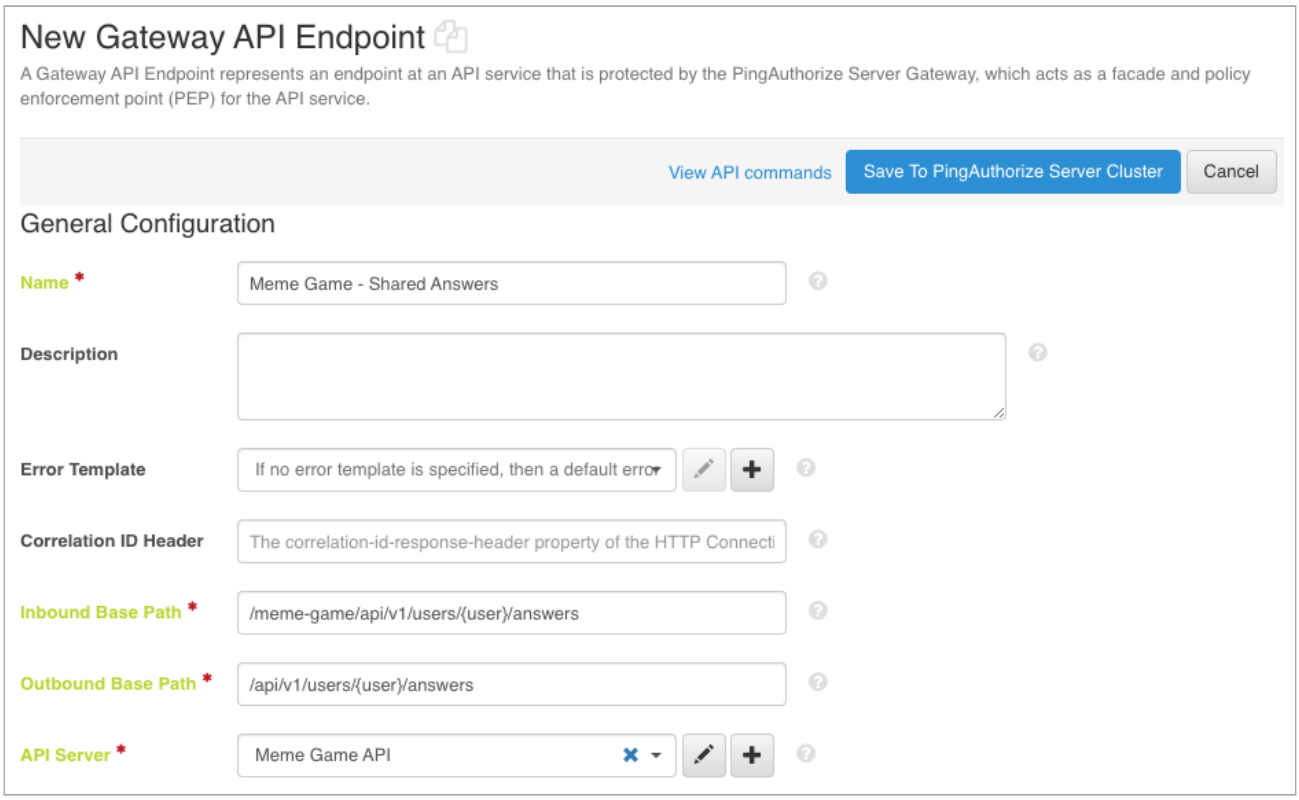
    5. For **Outbound Base Path**, specify `/api/v1/users/{user}/answers`.

        The outbound base path defines the base request path for requests that PingAuthorize Server forwards to an API server.

    6. For **API Server**, specify `Meme Game API`. This is the API External Server you defined in Configuring a reverse proxy for the Meme Game API.

Copyright © 2025 Ping Identity Corporation

Your screen should look like the following one.



7. 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 a `data` array.

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.

Copyright © 2025 Ping Identity Corporation

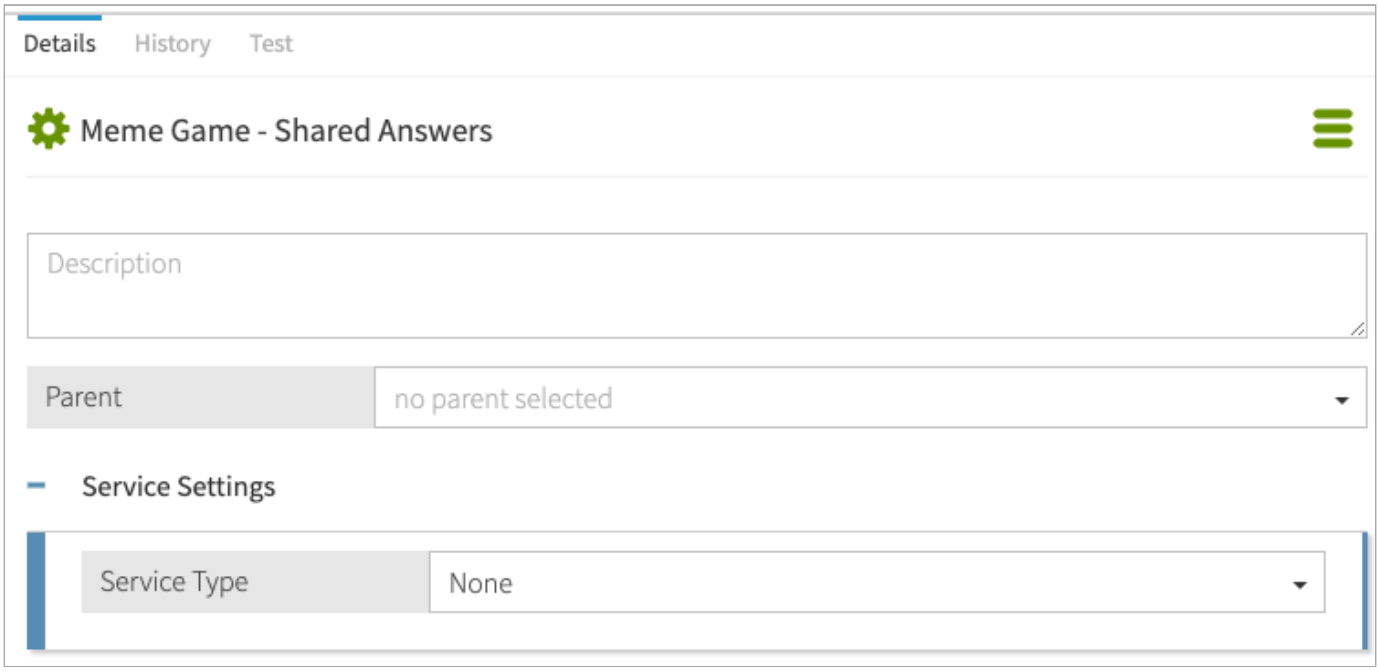
This task passes the name of the Gateway API Endpoint configured in PingAuthorize Server as the service to the PingAuthorize policy decision point (PDP).

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.

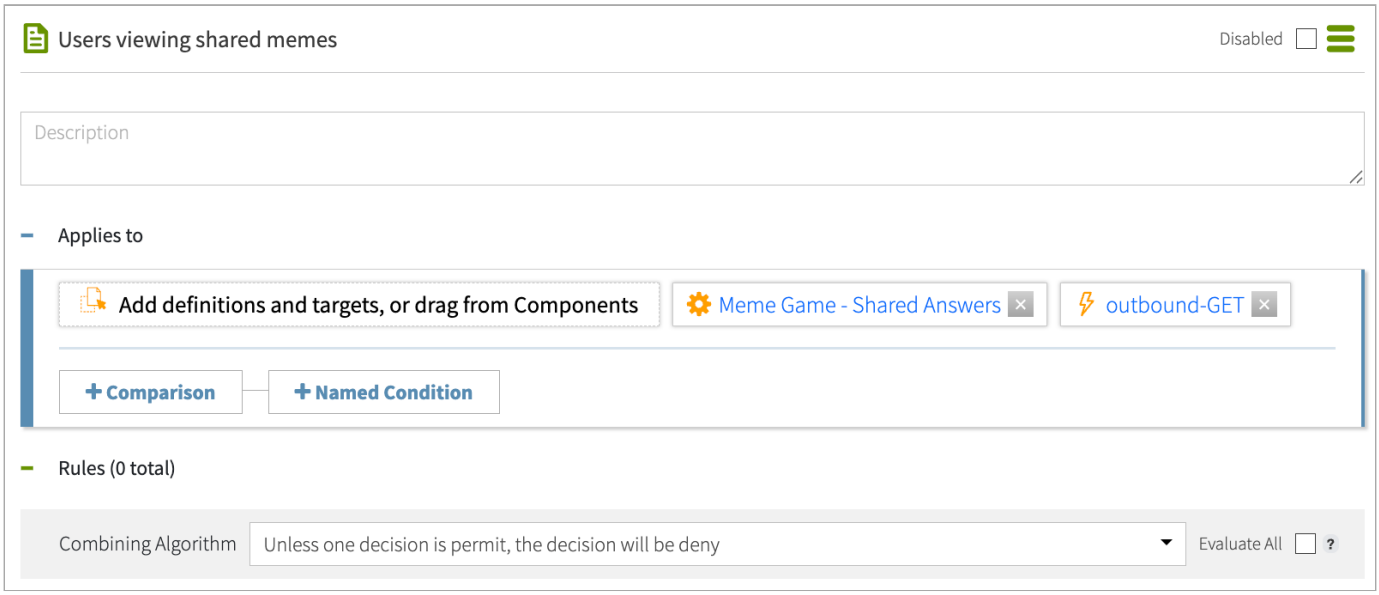
Steps

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**.

Copyright © 2025 Ping Identity Corporation

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 example in the following image:



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
}
```

Copyright © 2025 Ping Identity Corporation

Creating an attribute from user data

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

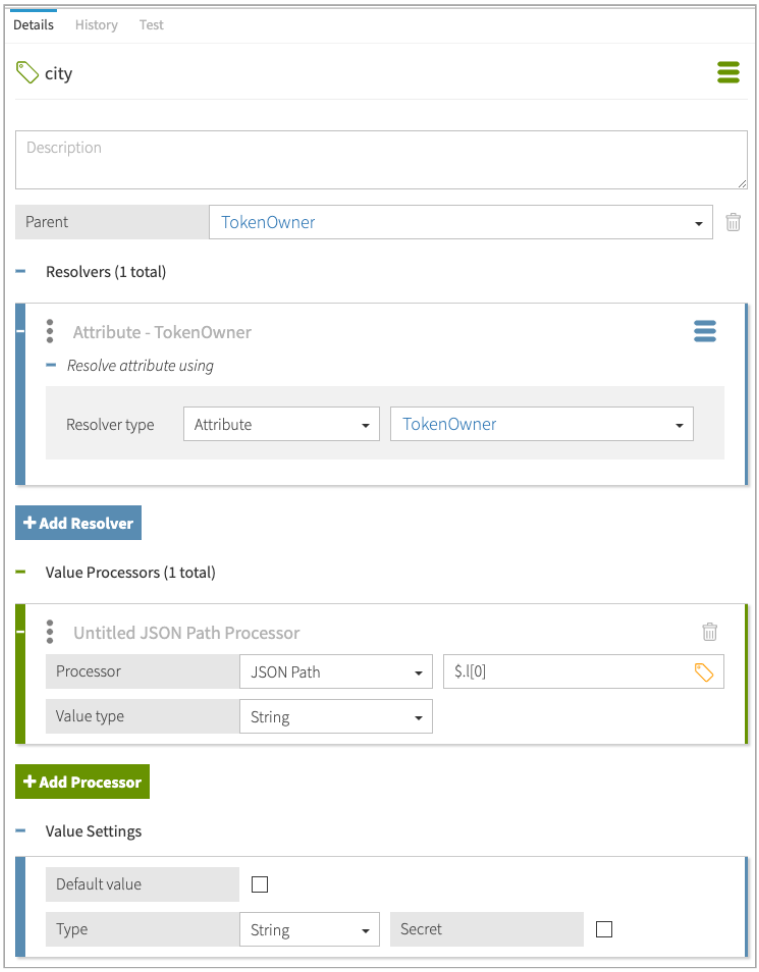
Steps

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` represents 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.

Copyright © 2025 Ping Identity Corporation



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

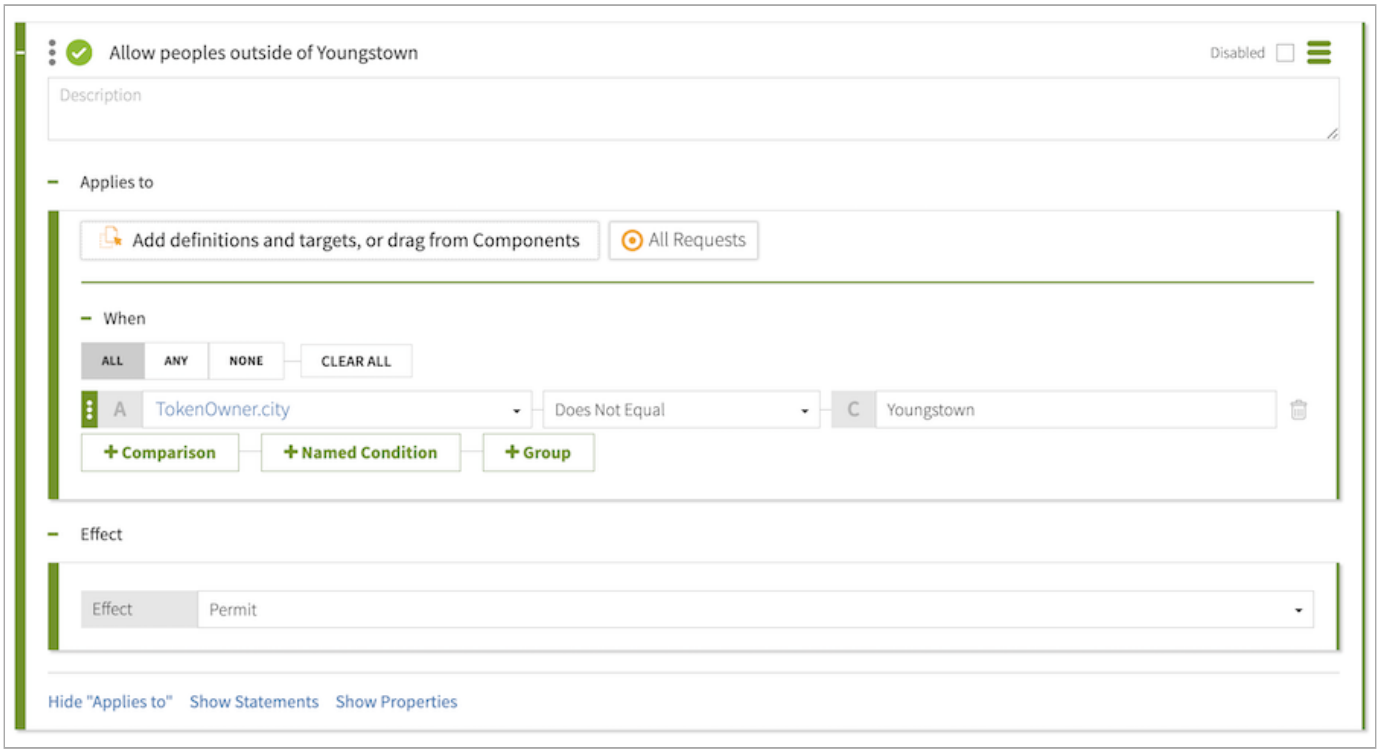
1. From the PingAuthorize Policy Editor, go to the **Policies** tab.
2. Select **Users viewing shared memes**.
3. Click **+ Add Rule**.
4. For the name, replace **Untitled** with `Allow people outside of Youngstown`.
5. For **Effect**, select **Permit**.
6. To specify a **Condition**, perform the following steps:
    1. Click **+ Comparison**.
    2. From the **Select an Attribute** list, select **TokenOwner.city**.
    3. In the second field, select **Does Not Equal**.
    4. In the third field, type `Youngstown`.

Copyright © 2025 Ping Identity Corporation

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:

Copyright © 2025 Ping Identity Corporation

```
{
        "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.

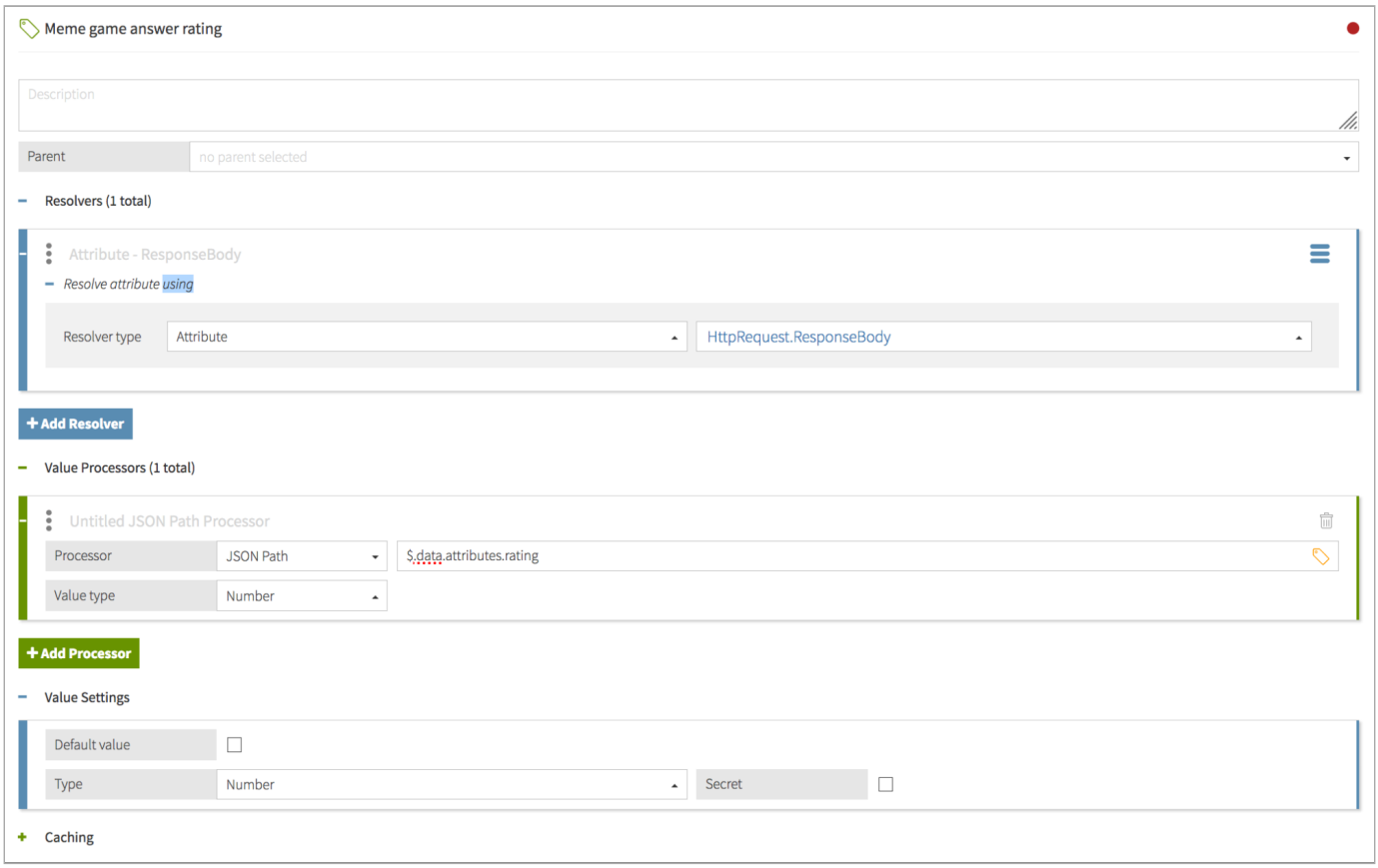
   To remove a parent, click the delete icon to the right of the **Parent** field.

Copyright © 2025 Ping Identity Corporation

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

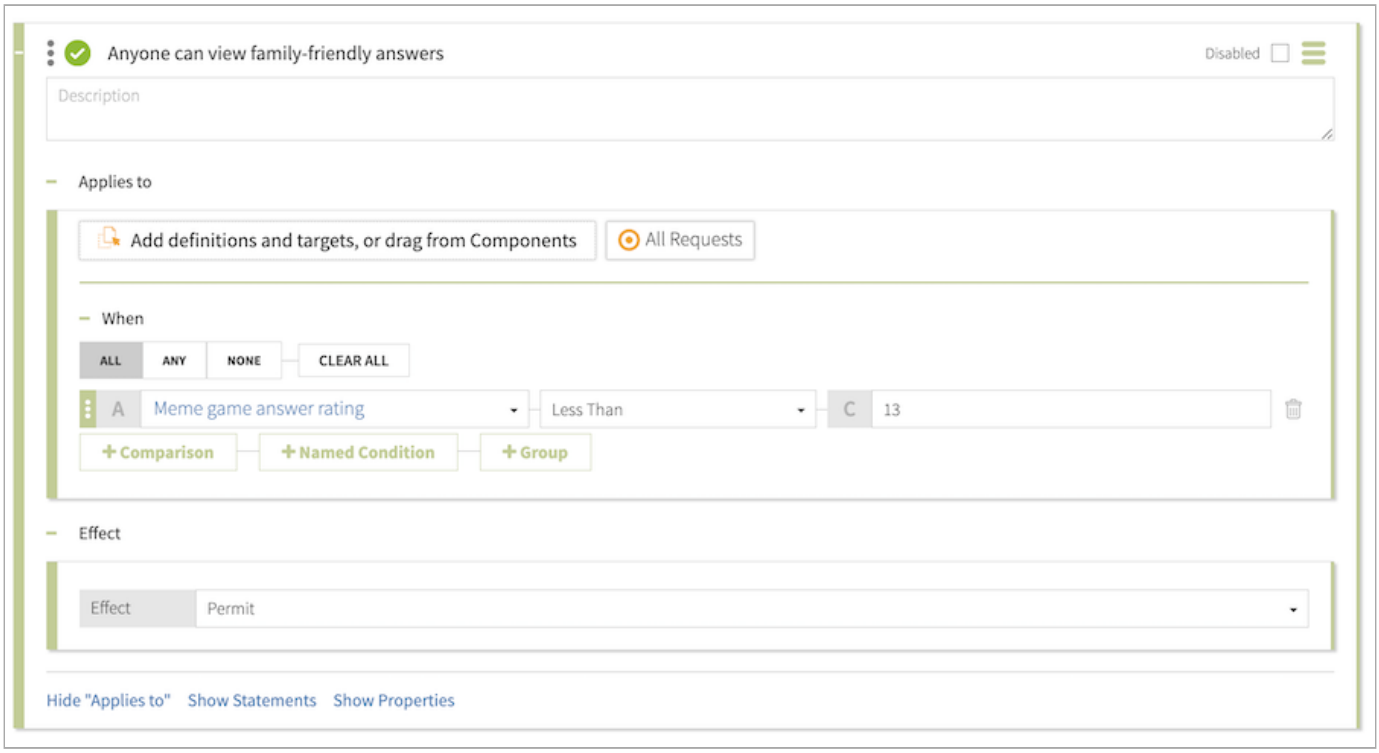
1. From the PingAuthorize Policy Editor, go to the **Policies** tab.
2. Select **Users viewing shared memes**.
3. Click + **Add Rule**.

Copyright © 2025 Ping Identity Corporation

4. For the name, replace **Untitled** with `Anyone can view family-friendly answers`.
5. For **Effect**, select **Permit**.
6. Specify a **Condition**.
    1. Click **+ Comparison**.
    2. From the **Select an Attribute** list, select **Meme game answer rating**.
    3. In the second field, select **Less Than**.
    4. In the third field, type `13`.
7. Click **Save changes**.

    Result:

    Your new rule to allow family-friendly memes should look 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:

Copyright © 2025 Ping Identity Corporation

```
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:

Copyright © 2025 Ping Identity Corporation

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.

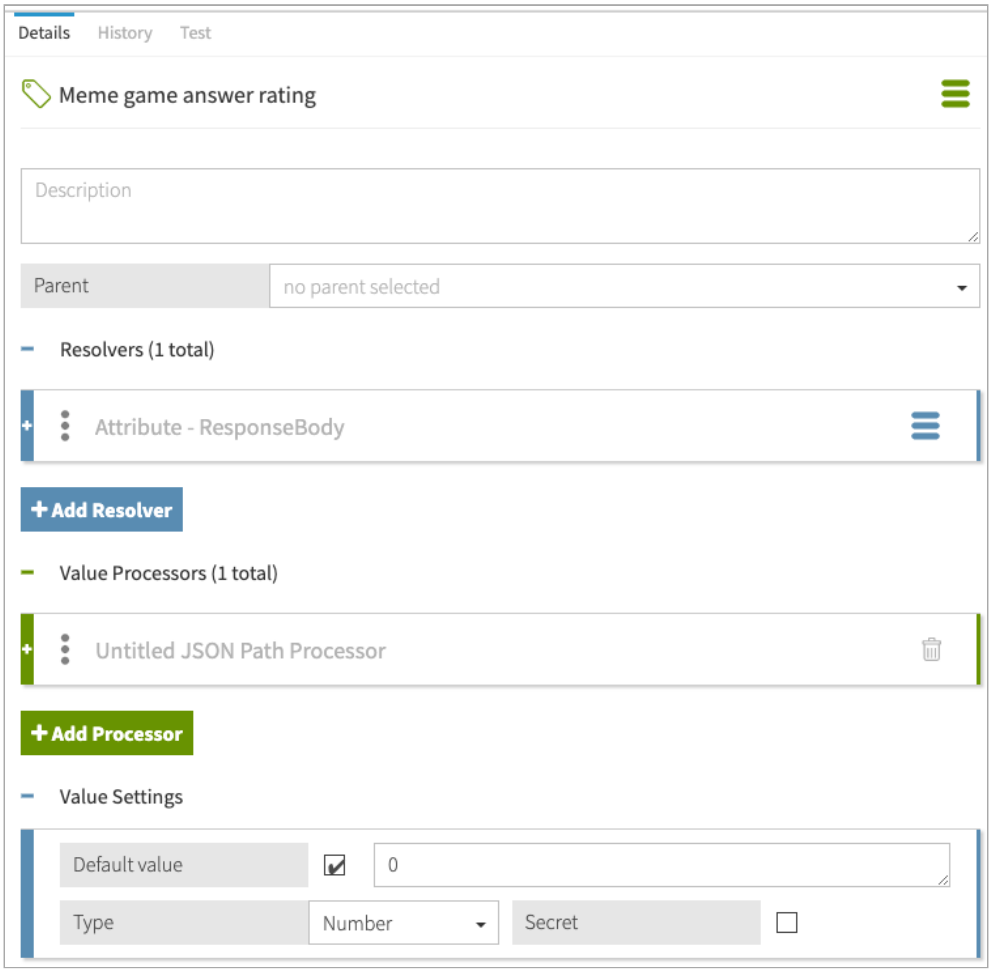
Steps

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**.

Copyright © 2025 Ping Identity Corporation

Result:

Your attribute for answer age ratings has a default value of 0, as shown in the following image:



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:

Copyright © 2025 Ping Identity Corporation

```
{
        "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 a statement to provide a more useful error message

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

Because the reason for denial is a user attribute (the user's location), use a 4xx response code to indicate a user issue. The `451` status code is commonly used when content is unavailable because of legal restrictions.

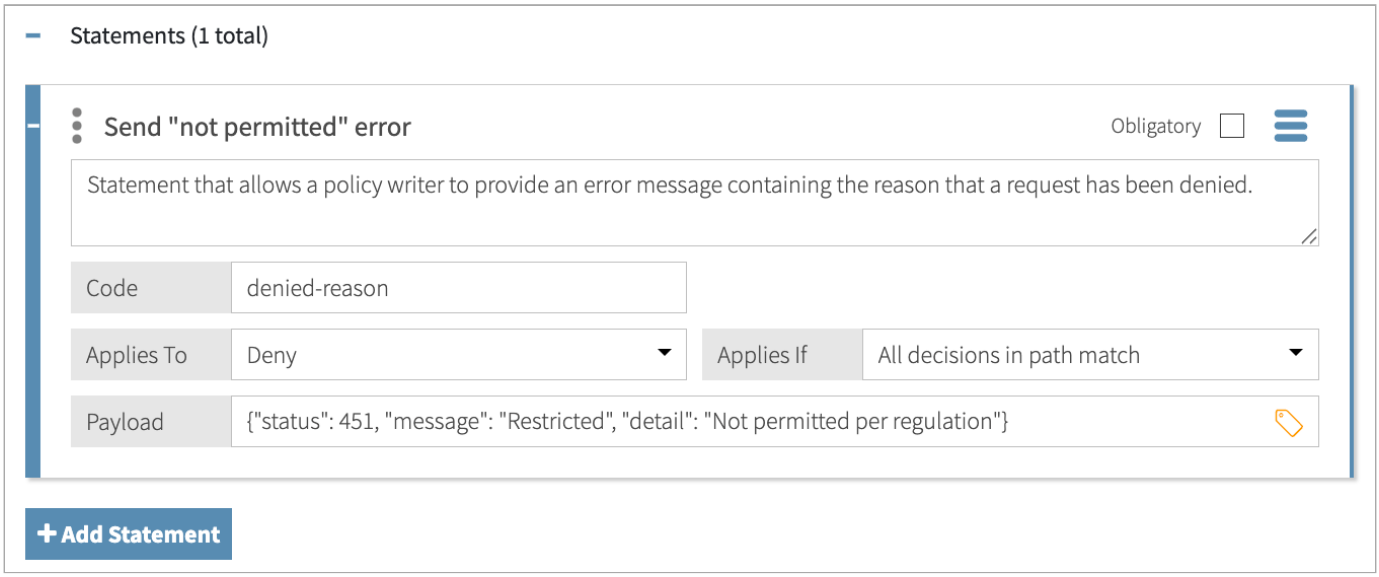
Steps

1. In the Policy Editor, click **Policies**.
2. Select **Users viewing shared memes**.
3. Click **+ Statements**.
4. Click **+ Add Statement** and select **Denied Reason**.
5. For the name, replace **Untitled** with `Send "not permitted" error`.
6. In the **Applies to** list, select **Deny**.
7. In the **Payload** field, enter `{"status": 451, "message": "Restricted", "detail": "Not permitted per regulation"}`.
8. Click **Save changes**.

   Result:

   You have a new statement that looks like the following image:

Copyright © 2025 Ping Identity Corporation



Testing the statement

You can test that the statement 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 statements to modify response bodies.

You also learned:

- Policies can apply to outbound upstream server API responses before they are sent to the API client.

Copyright © 2025 Ping Identity Corporation

- `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.
- Statements are 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: 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 datastore like Lightweight Directory Access Protocol (LDAP) directories or 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 subject (SAML), 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 attributes 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"]}
```

Copyright © 2025 Ping Identity Corporation

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 scope `email` and `profile`.

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

Tasks

This tutorial walks you through the following 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

Create a policy tree structure and ensure that your policies apply only to 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:
    1. Sign on to the PingAuthorize Policy Editor using the URL and credentials from Accessing the GUIs.
    2. Click **Policies**.
    3. Select **Global Decision Point**.
    4. From the + menu, select **Add Policy Set**.
    5. For the name, replace **Untitled** with `SCIM Policy Set`.
    6. 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.
    7. Click + **Applies to**.
    8. Click **Components**.

Copyright © 2025 Ping Identity Corporation

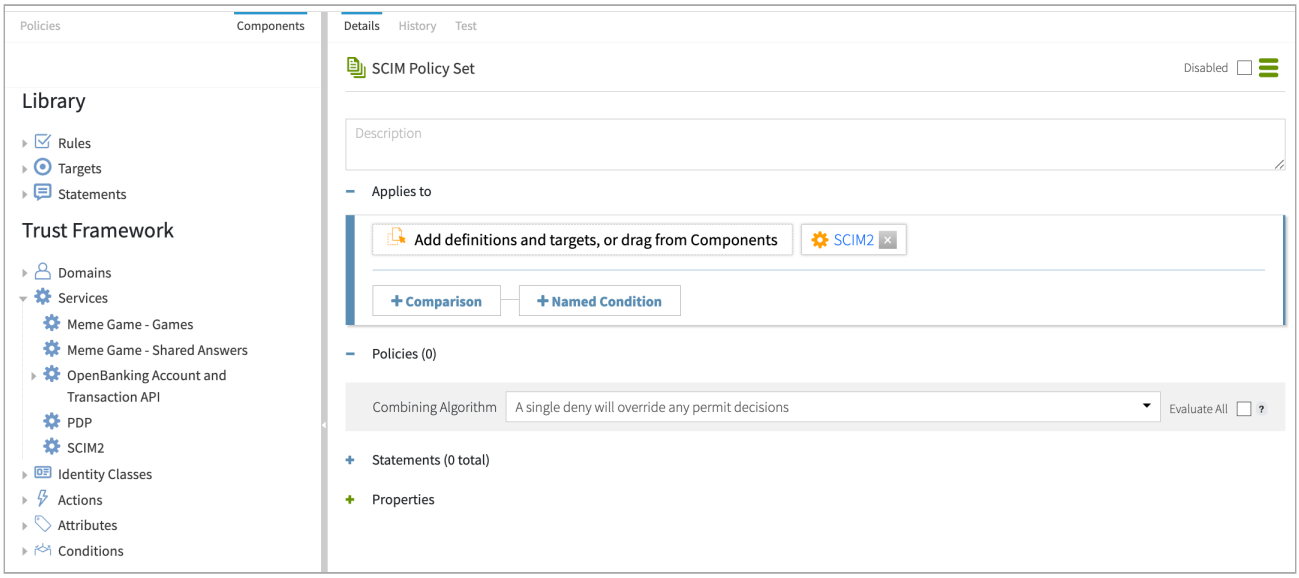
9. From the **Services** list, drag **SCIM2** to the **Add definitions and targets, or drag from Components** box.

   This step ensures that policies in the SCIM policy set apply only to SCIM requests.

10. 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:

   1. Select **SCIM Policy Set**.
   2. From the **+** menu, select **Add Policy Set**.
   3. For the name, replace **Untitled** with `Token Policies`.
   4. In the **Policies** section, set the **Combining algorithm** to **A single deny will override any permit decisions**.
   5. Click **Save changes**.

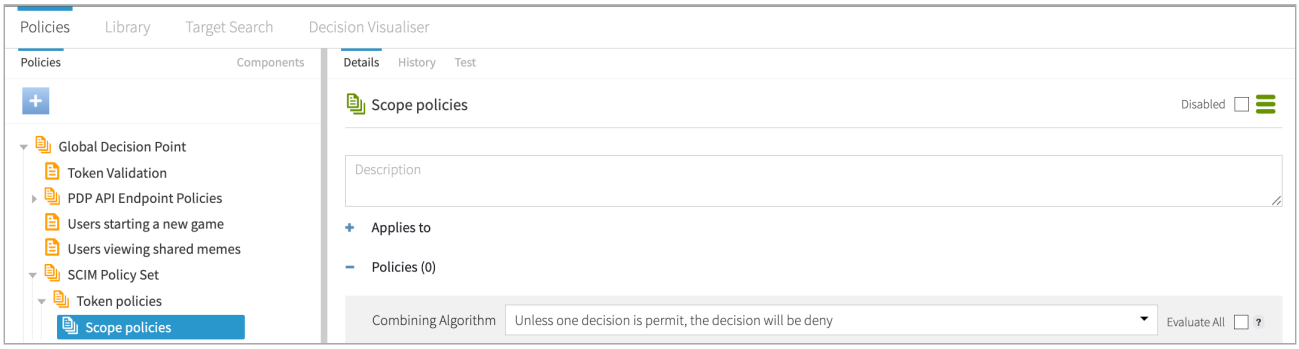
3. To add another branch that holds a policy specific to access token scopes, perform the following steps:

   1. Select **Token Policies**.
   2. From the **+** menu, select **Add Policy Set**.
   3. For the name, replace **Untitled** with `Scope Policies`.
   4. In the **Policies** section, set the **Combining algorithm** to **Unless one decision is permit, the decision will be deny**.
   5. Click **Save changes**.

      Result:

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

Copyright © 2025 Ping Identity Corporation



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 data access.

Creating a policy for permitted access token scopes

The first policy defines the access token scopes that PingAuthorize Server accepts for 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.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, and **Token Policies**.
4. Select **Scope Policies**.
5. Next to **Statements**, click **+**.
6. Click **Components**.
7. From the **Statements** list, drag **Insufficient Scope** to the area immediately following the **Statements** section. A box appears for you to drop the item into.

Copyright © 2025 Ping Identity Corporation

8. Click **Save changes**.
9. Click **Policies** to the left of **Components**.
10. Select **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.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, **Token Policies**, and **Scope Policies**.
4. Select **Permitted Scopes**.
    1. 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`.

Copyright © 2025 Ping Identity Corporation

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 Statements**.
15. Click **+** next to **Statements**.
16. From the **Statements** list, drag **Include email attributes** to the **Statements** section of the rule.

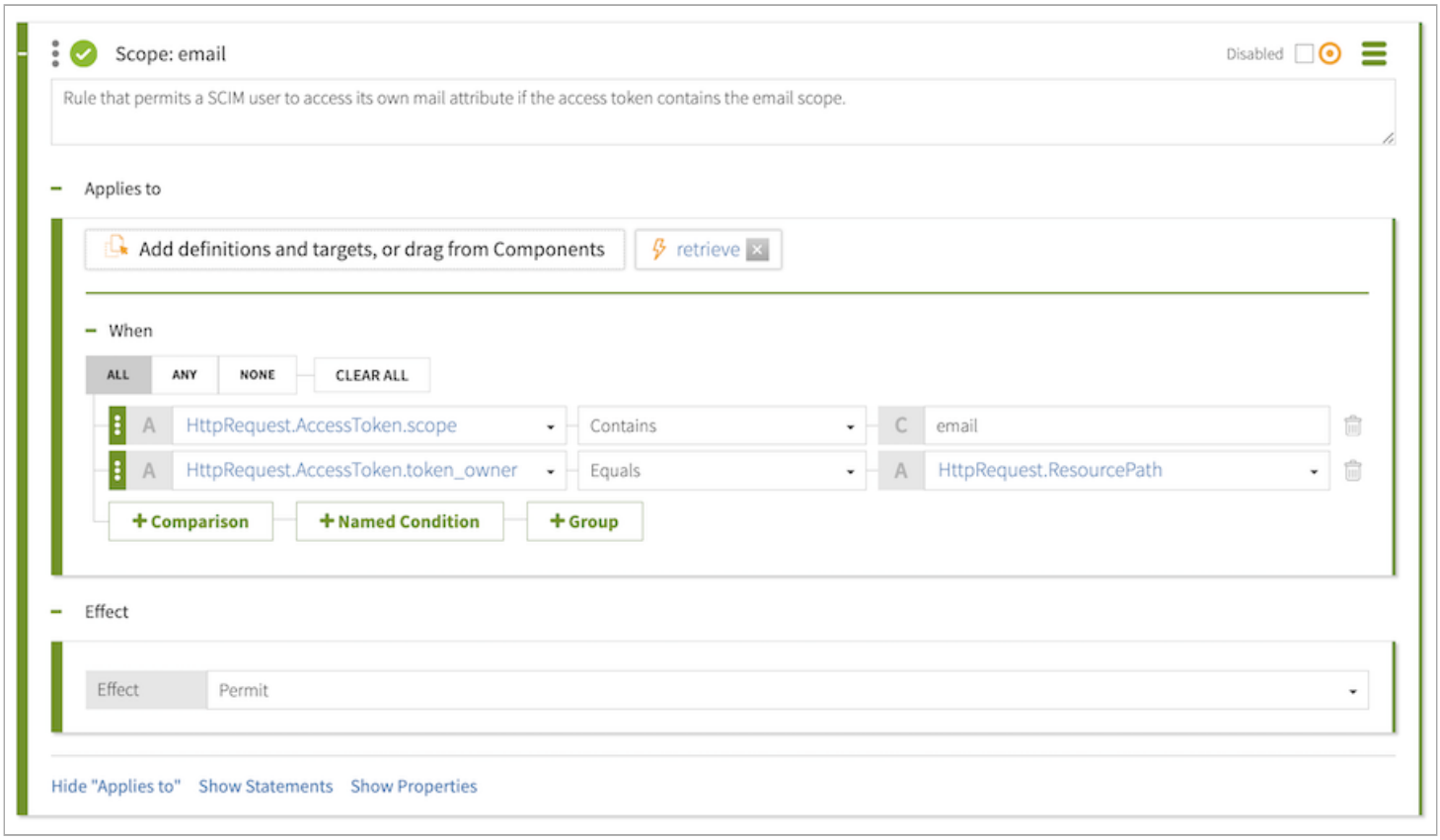
> **Note**
>
> This predefined statement includes a payload. If the condition for this rule is satisfied, the response includes the `mail` attribute.

17. Click **Save changes**.

Result

You now have a new email scope, which should look like the following.

Copyright © 2025 Ping Identity Corporation



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.

Copyright © 2025 Ping Identity Corporation

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.
2. Click **Policies**.
3. Expand **Global Decision Point**, **SCIM Policy Set**, **Token Policies**, and **Scope Policies**.
4. Select **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 Statements**.
16. Next to **Statements**, click **+**.
17. From the **Statements** list, drag **Include profile attributes** to the **Statements** section of the rule.

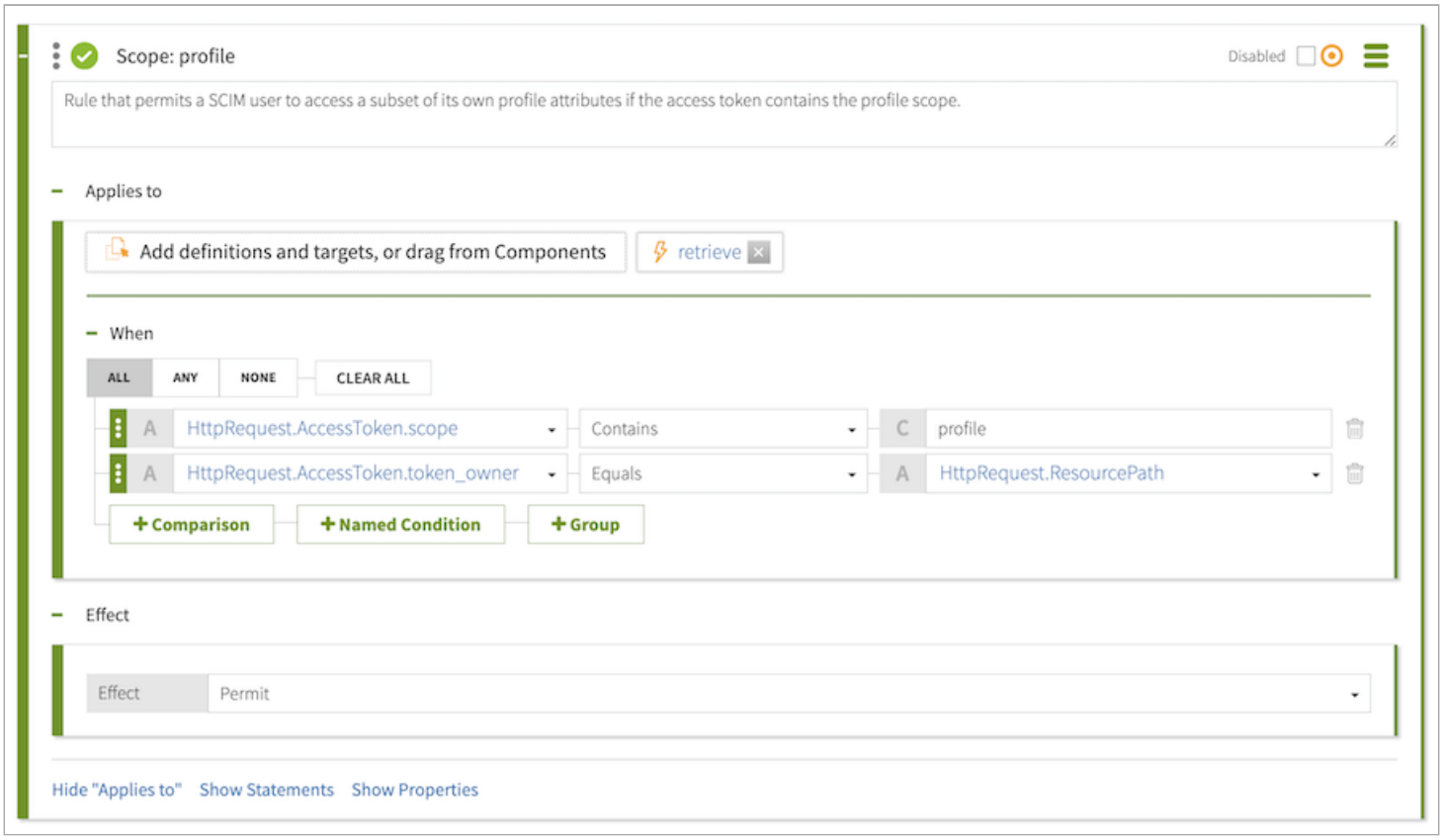
    > **Note**
    >
    > This predefined statement includes a payload. If the condition for this rule is satisfied, the response includes the `uid`, `sn`, `givenName`, and `description` attributes.

18. Click **Save changes**.

Result

You now have a new profile scope, which should look like the following:

Copyright © 2025 Ping Identity Corporation



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 statement 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:

Copyright © 2025 Ping Identity Corporation

```
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

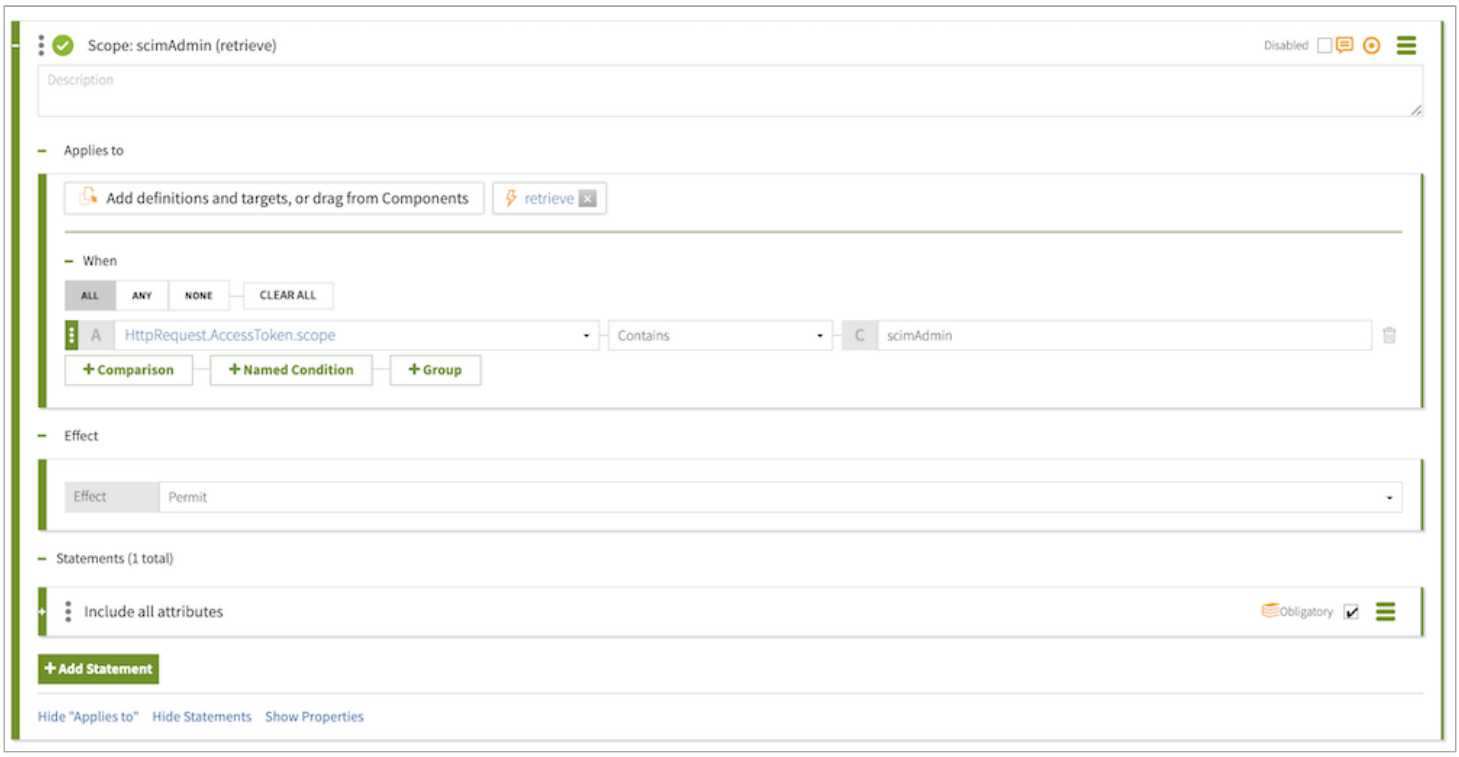
1. Sign on to the PingAuthorize Policy Editor using the URL and credentials from Accessing the GUIs.
2. Click **Policies**.
3. Select **Permitted Scopes**.
4. Click **+ Add Rule**.
5. For the name, replace **Untitled** with `Scope: scimAdmin (retrieve)`.
6. From the **Effect** list, select **Permit**.
7. In the **Condition** section, perform the following steps:
    1. Click **+ Comparison**.
    2. In the first field, select **HttpRequest.AccessToken.scope**.
    3. From the comparator list, select **Contains**.
    4. 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 Statements**.
12. Click **+** next to **Statements**.

Copyright © 2025 Ping Identity Corporation

13. From the **Statements** list, drag **Include all attributes** to the **Statements** section of the rule.
14. Click **Save changes**.

Result

You now have a new scope for the scimAdmin retrieve rule, which 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.
2. Click **Policies**.
3. Select **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:
    1. Click **+ Comparison**.

Copyright © 2025 Ping Identity Corporation

2. In the first field, select **HttpRequest.AccessToken.scope**.
3. From the comparator list, select **Contains**.
4. 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.
2. Click **Policies**.
3. Select **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:
    1. Click **+ Comparison**.
    2. In the first field, select **HttpRequest.AccessToken.scope**.
    3. From the comparator list, select **Contains**.
    4. 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**.

Copyright © 2025 Ping Identity Corporation

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.
2. Click **Policies**.
3. Select **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:
    1. Click **+ Comparison**.
    2. In the first field, type `HttpRequest.AccessToken.scope`.
    3. From the comparator list, select **Contains**.
    4. 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**.

Copyright © 2025 Ping Identity Corporation

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:
    1. Click **+ Comparison**.
    2. From the **Select an Attribute** list, select `HttpRequest.AccessToken.client_id`.
    3. From the middle, comparison-type list, select `Equals`.
    4. 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:
    1. Click **+ Comparison**.
    2. From the **Select an Attribute** list, select `HttpRequest.AccessToken.client_id`.
    3. From the middle, comparison-type list, select `Equals`.
    4. In the final field, enter `client2`.
14. Expand **+ Statements**.

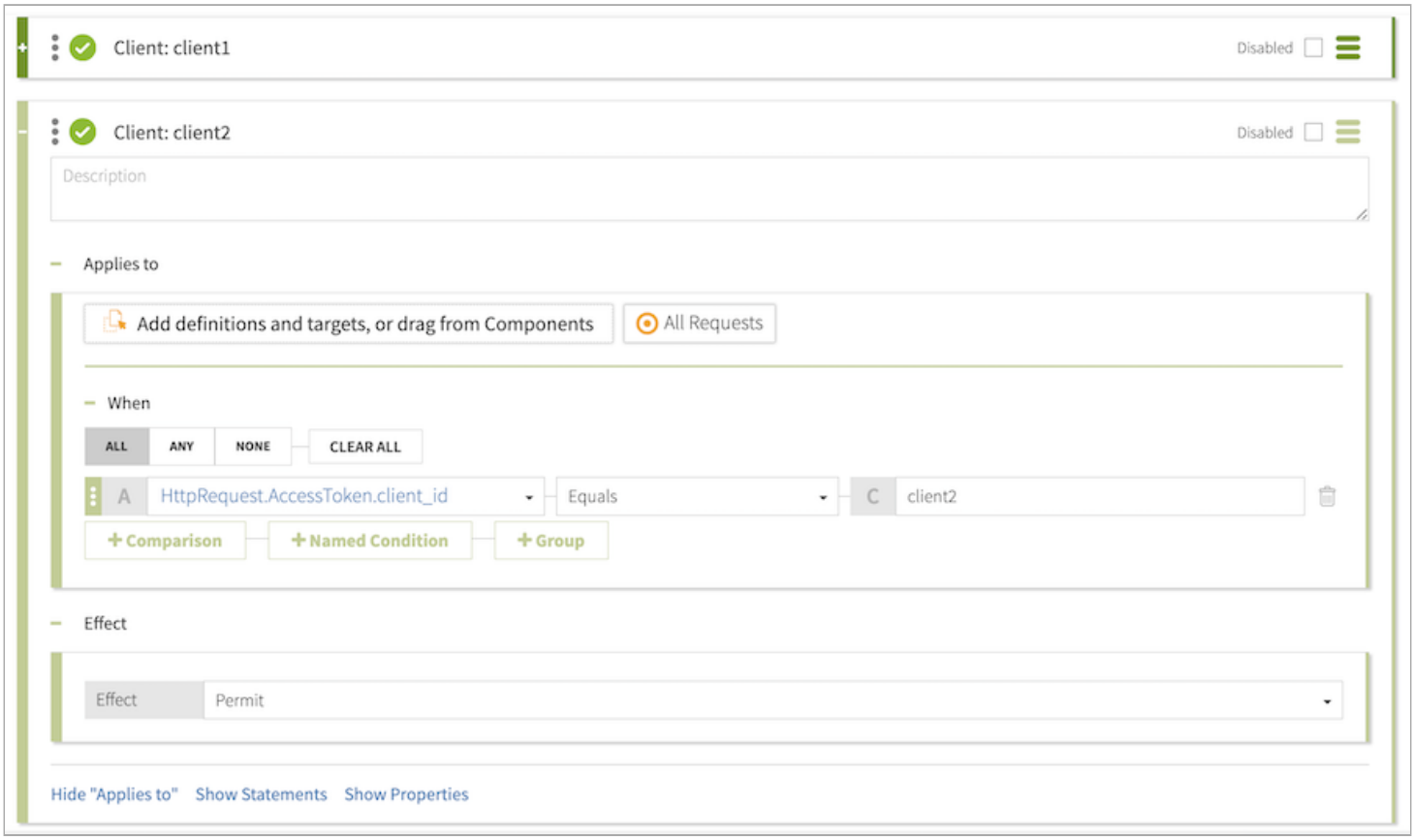
> **Note**
>
> Do not click **Show Statements** within the client1 or client2 rules.

15. Click **Components**.
16. From the **Statements** list, drag **Unauthorized Client** to the **Statements** box.
17. Click **Save changes**.

Result

The completed configuration should resemble the following image.

Copyright © 2025 Ping Identity Corporation



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.

Copyright © 2025 Ping Identity Corporation

```
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:
    1. Click **+ Comparison**.
    2. From the **Select an Attribute** list, select `HttpRequest.AccessToken.audience`.
    3. From the middle, comparison-type list, select `Equals`.
    4. In the final field, enter `https://example.com`.
10. Expand **+ Statements**.
11. Click the **Components** tab, expand **Statements**, and drag `Unauthorized Audience` to the **Statements** box.

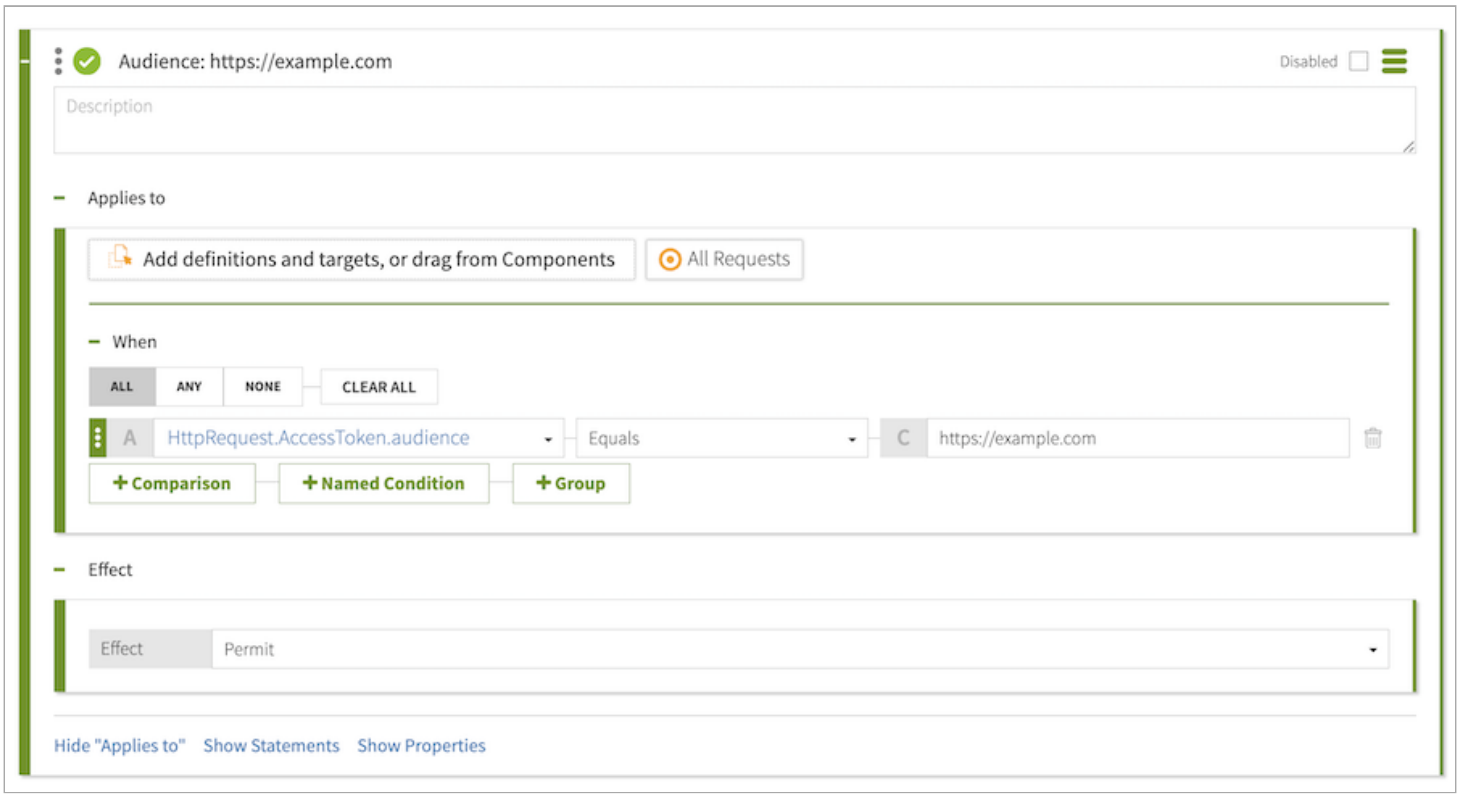
Copyright © 2025 Ping Identity Corporation

> **Note**
>
> Do not click **Show Statements** 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

To confirm that you successfully completed the previous task, 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."}
```

Copyright © 2025 Ping Identity Corporation

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 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:
    1. Click **+ Add Resolver**.
    2. 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:
    1. Select the **Default Value** check box and in the **Enter a default value** field, enter the value `[]`.

Copyright © 2025 Ping Identity Corporation

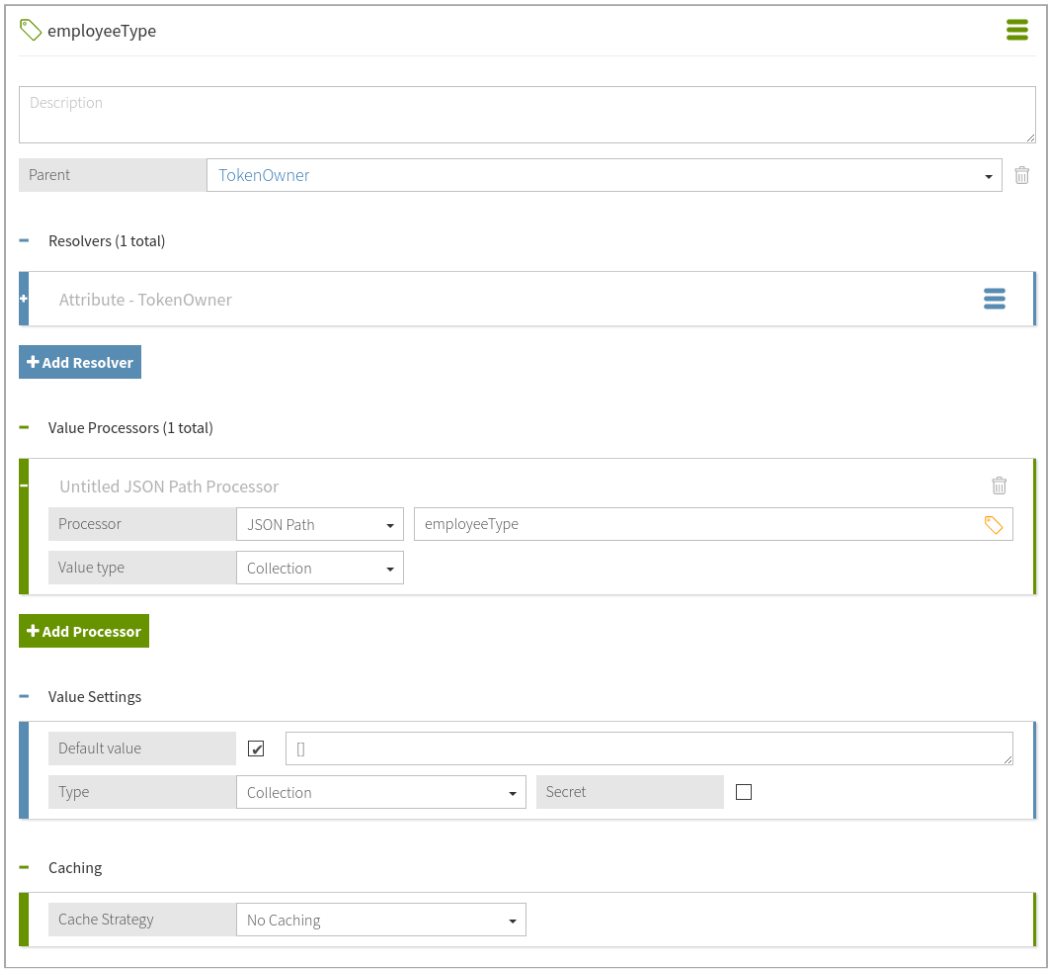
> **Note**
>
> An empty array is specified as the default value because not all users have an `employeeType` attribute. A default value of `[]` ensures that policies can safely use this attribute to define conditions.

2. From the **Type** list, select `Collection`.

10. Click **Save changes**.

Result

The final attribute configuration should resemble the following image:



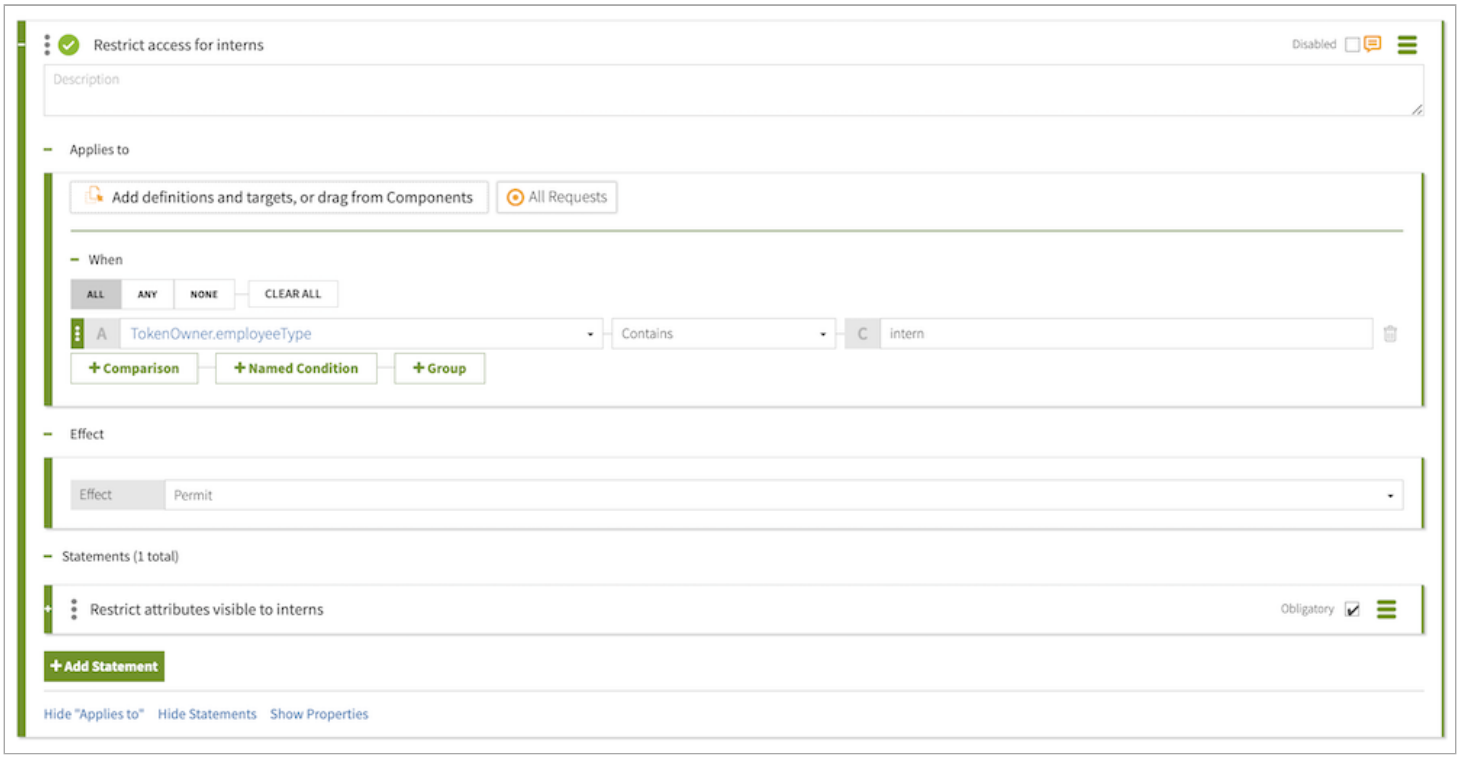
Next steps

Add a policy that uses the `employeeType` attribute.

1. Go to **Policies > Policies**.
2. Select **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**.

Copyright © 2025 Ping Identity Corporation

6. For the name, replace **Untitled** with `Restrict access for interns`.
7. From the **Effect** list, select `Permit`.
8. In the **Condition** section:
    1. Click **+ Comparison**.
    2. In the **Select an Attribute** list, select `TokenOwner.employeeType`.
    3. From the middle, comparison-type list, select `Contains`.
    4. In the **Type in constant value** field, enter `intern`.
9. Within the rule, click **Show Statements**, and then click the **+** next to **Statements**.
10. Click **+ Add Statement** > **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.

Copyright © 2025 Ping Identity Corporation

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 both contain an example Trust Framework and 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`.

Copyright © 2025 Ping Identity Corporation

Installing PingAuthorize

You can install PingAuthorize manually or deploy it with Docker.

Installation method	Recommended for
Docker	Server administrators familiar with Docker who want to use orchestration to manage their environments. Learn more in Docker deployment.
Manual	Server administrators familiar with their operating systems who want to tweak and maintain their environments themselves. Learn more in Manual installation.

Learn more about the available implementation architectures and development environments for your PingAuthorize installation in PingAuthorize architectural overview.

System requirements

Make sure that your computing environment meets the system requirements for the PingAuthorize dynamic authorization management software.

Ping Identity has tested and certified the configurations listed on this page for compatibility with PingAuthorize. Variations in operating system versions, service packs, or platform components are generally supported unless they are known to cause issues.

> **Note**
>
> The following requirements apply to both Docker deployments and manual installations.

Platforms

PingAuthorize supports actively maintained versions of the following operating systems:

- Amazon Linux
- Canonical Ubuntu LTS
- Microsoft Windows Server 2022 (Policy Editor not supported)
- Oracle Linux
- Red Hat Enterprise Linux ES
- Rocky Linux
- SUSE Linux Enterprise

> **Note**
>
> PingAuthorize has been tested with the default configurations of all operating system components. Customized implementations or third-party plugins could affect the deployment of PingAuthorize.

Copyright © 2025 Ping Identity Corporation

Java Development Kit

Install one of the following versions of the Java Development Kit (JDK) before installing PingAuthorize:

- Oracle JDK 17 and 21
- OpenJDK 17 and 21

> **Note**
>
> Because of the number of supported JDK variants, not all of them are listed here. For PingAuthorize to support a JDK, the JDK must:
>
> - Be built from the OpenJDK codebase.
>   Supported OpenJDK distributions include, but aren't limited to:
>   - AdoptOpenJDK/Eclipse Temurin Java Development Kit (Adoptium)
>   - Amazon Corretto
>   - Azul Zulu
>   - Red Hat OpenJDK
> - Use the HotSpot JIT compiler. The JDK cannot use OpenJ9 or other JIT implementations.
> - Comply with the Ping Identity Java Support Policy.

Browsers

The PingAuthorize administrative console is compatible with actively maintained versions of the following web browsers:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge

Databases

The Policy Editor persists its policies, Trust Framework, and versioning data in one of the following policy databases:

- An embedded, file-based H2 database. The Policy Editor uses this database by default.
- An external PostgreSQL database.

  Learn more in Setting up a PostgreSQL database.

Docker deployment

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

Learn more about installation and deployment methods in Installing PingAuthorize.

Copyright © 2025 Ping Identity Corporation

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

Copyright © 2025 Ping Identity Corporation

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.

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

Steps

1. Run the PingAuthorize Server container, `pingauthorize`.
2. Run the 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 Post-setup steps (Docker deployment).
- Perform Next steps as needed.

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 Get Started.

Steps

- Run the following command.

Copyright © 2025 Ping Identity Corporation

```
docker run --network=<network_name>  \
   --env-file ~/.pingidentity/config \
   --name  {SERVER_CONTAINER_NAME}  \
   --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/{SERVER_CONTAINER_NAME}  \
   --tmpfs /run/secrets \
  pingidentity/{SERVER_CONTAINER_NAME}:<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.

Deploying the Policy Editor using Docker

Deploy the 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 Get Started.
- 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.

To use a PostgreSQL policy database, make sure you have met the following prerequisites:

- The PostgreSQL instance must be reachable on the network from the Policy Editor host and listening for connections.

Copyright © 2025 Ping Identity Corporation

- The Policy Editor uses both a PostgreSQL administration user and a server runtime user. Have a database administrator create both users before providing their credentials to the `policy-db` tool. The administration user must be able to create new databases. When new releases of the Policy Editor become available, continue using the same administration user to prevent database object ownership issues.

  Learn more about creating new database users and configuring PostgreSQL to listen for remote connections securely in the PostgreSQL documentation.

- The Policy Editor uses Java Database Connectivity (JDBC) to connect to PostgreSQL. Be prepared to provide the JDBC connection string in the following format: `jdbc:postgresql://<host>:<port>/<name>`. For example: `jdbc:postgresql://example.com:5432/pap_db`

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

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/{PAP_CONTAINER_NAME}:<TAG>
```

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

    The official `pingauthorizepap` Docker image detects whether a PostgreSQL database needs to be created or upgraded when you provide the `PING_POLICY_DB_SYNC=true` environment variable along with the database connection string, database administration credentials, and server runtime credentials.

Copyright © 2025 Ping Identity Corporation

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

Note

The `PING_DB_APP_PASSWORD` and `PING_DB_ADMIN_PASSWORD` can instead be provided as Vault secrets or through a secrets volume. See Using Hashicorp Vault.

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

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  {SERVER_CONTAINER_NAME}  /opt/out/instance/bin/dsconfig create-external-server \
  --server-name "{PAP_Name}" \
  --type policy \
  --set "base-url:https://<pap-hostname>:<pap-port>" \
  --set "shared-secret:2FederateM0re" \
  --set "branch:Default Policies"

docker exec  {SERVER_CONTAINER_NAME}  /opt/out/instance/bin/dsconfig set-policy-decision-service-prop \
  --set pdp-mode:external \
  --set "policy-server:{PAP_Name}" \
  --set trust-framework-version:{TRUST_FRAMEWORK_VERSION}
```

Copyright © 2025 Ping Identity Corporation

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.
- Configure a user store.
- Sign on to the administrative console to configure endpoints for existing JSON APIs.

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

  For more information, see About the SCIM service.
- 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 install PingAuthorize manually using `.zip` archives.

Learn more about installation and deployment methods in Installing PingAuthorize.

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.

About license keys

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

Copyright © 2025 Ping Identity Corporation

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 Policy Editor non-interactively.

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
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`.

Copyright © 2025 Ping Identity Corporation

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

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

2. Expand the release bundles into the folders of your choice.

Setting up a PostgreSQL database

To set up a PostgreSQL database for your attribute-based access control policies, create the policy database using the **`policy-db`** tool.

Follow these instructions to create a PostgreSQL database for a manual installation of the Policy Editor. Refer to Deploying the Policy Editor using Docker for containerized deployments.

Before you begin

- The PostgreSQL instance must be reachable on the network from the Policy Editor host and listening for connections.
- The Policy Editor uses both a PostgreSQL administration user and a server runtime user. Have a database administrator create both users before providing their credentials to the **`policy-db`** tool. The administration user must be able to create new databases. When new releases of the Policy Editor become available, continue using the same administration user to prevent database object ownership issues.

  Learn more about creating new database users and configuring PostgreSQL to listen for remote connections securely in the PostgreSQL documentation.
- The Policy Editor uses Java Database Connectivity (JDBC) to connect to PostgreSQL. Be prepared to provide the JDBC connection string in the following format: `jdbc:postgresql://<host>:<port>/<name>`. For example: `jdbc:postgresql://example.com:5432/pap_db`

Steps

1. Run the following command:

```
$ bin/policy-db \
  --dbConnectionString "jdbc:postgresql://<host>:<port>/<name>" \
  --dbAppUsername  <server-runtime-username>  \
  --dbAppPassword  <server-runtime-password>
```

> **Note**
>
> Alternatively, you can provide the server runtime password through the `PING_DB_APP_PASSWORD` environment variable.

2. Provide the database administration credentials when prompted.

Result

The **`policy-db`** tool connects to PostgreSQL, creates the database and its objects, and grants access to the server runtime username.

Next steps

Configure the Policy Editor to use the PostgreSQL database. See Installing the Policy Editor non-interactively.

Copyright © 2025 Ping Identity Corporation

> **Important**
>
> Provide the Policy Editor with the same `--dbConnectionString`, `--dbAppUsername`, and server runtime password you used to create the PostgreSQL database.

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.

Copyright © 2025 Ping Identity Corporation

- 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.

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.

Copyright © 2025 Ping Identity Corporation

Interactive

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.

Copyright © 2025 Ping Identity Corporation

Noninteractive

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`

Copyright © 2025 Ping Identity Corporation

- 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
```

Installing the PingAuthorize Policy Editor manually

Choose the database for your fine-grained access control use case, protected resource, 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.

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 Policy Editor interactively.

   - 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 Policy Editor non-interactively.

Copyright © 2025 Ping Identity Corporation

Installing the Policy Editor interactively

You can run the Policy Editor `setup` command interactively in command-line interface (CLI) install mode.

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

> **Note**
>
> You cannot configure some setup options when installing the Policy Editor interactively, such as PostgreSQL database configuration. Learn more in Installing the Policy Editor non-interactively.

Before you begin

You must have the following information:

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

Steps

1. Choose the authentication mode for the Policy Editor:

   Choose from:

   - **Demo mode**: Configures the 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 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 Policy Editor with other OIDC providers, such as PingOne, see Installing the Policy Editor non-interactively.

2. Run the `setup` command.

   > **Note**
   >
   > If you don't want to use the default database credential, refer to Setting database credentials at initial setup.

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`.

Copyright © 2025 Ping Identity Corporation

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

Example

Refer to Example: Installing and configuring the Policy Editor interactively for a more detailed walkthrough of the previous steps.

Next steps

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

Example: Installing and configuring the Policy Editor interactively

This tutorial describes how to install an instance of the PingAuthorize Policy Editor interactively.

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. For example, `pap.example.com`.

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

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
[/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  {pingfed}
    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
```


Copyright © 2025 Ping Identity Corporation

```
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 Policy Editor non-interactively.

Installing the Policy Editor non-interactively

For an automated software installation, run `setup` in non-interactive command-line interface (CLI) install mode.

> **Note**
>
> You must run `setup` in non-interactive CLI 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 Secure Sockets Layer (SSL) certificate instead of generating a self-signed certificate.
> - Enable self-governance.
> - Enable Camel services.
>
> Learn more in Specifying custom configuration with an options file.

Steps

1. (Optional) If you 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, save the following information for installing the Policy Editor:

   - PostgreSQL Java database connectivity (JDBC) connection string, with the host, port, and database name
   - The server runtime credential provided through the `policy-db` tool

Copyright © 2025 Ping Identity Corporation

2. Choose one of the following authentication modes for the Policy Editor:

   - **Demo mode**: Configures the 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 be used only for demonstration purposes.
   - **OIDC mode**: Configures the 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 Policy Editor's connection security, including the location of a key store 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 Java Runtime Environment (JRE), do one of the following:
     > - Add the certificate to the JRE trust store. For details, refer to Configuring an OIDC provider for single sign-on requests from PingAuthorize.
     > - Disable certificate validation by starting the Policy Editor with the `PING_OIDC_TLS_VALIDATION` environment variable set to `NONE`.

     > **Tip**
     >
     > The `setup` tool's `--help` option displays the options available for a non-interactive installation.

3. Run the `setup` command with the appropriate authentication mode, as shown in Authentication mode examples.

   1. (Optional) If you're using a PostgreSQL policy database, provide the server runtime user value you used to create the database to `--dbAppUsername` as part of the `setup` command.
   2. (Optional) Refer to the CLI help documentation for the `setup` command.

Option	Command
View the general options for running `setup`.	`$ bin/setup --help`
View the options for running `setup` in demo mode.	`$ bin/setup demo --help`
View the options for running `setup` in OIDC mode.	`$ bin/setup oidc --help`

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

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

   The shared secret is used in the PingAuthorize administrative console, under **External Servers > Policy External Server > Shared Secret**.

Copyright © 2025 Ping Identity Corporation

5. Run `bin/start-server` to start the Policy Editor.

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

   1. (Optional) If you're using a PostgreSQL policy database, provide the server runtime password value you used to create the database to the `PING_DB_APP_PASSWORD` environment variable before starting the server.

Authentication mode examples

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

1. After you complete setup, refer to Post-setup steps (manual installation).
2. Consider additional configuration options in Specifying custom configuration with an options file.

Copyright © 2025 Ping Identity Corporation

OIDC mode (PingFederate)

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 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 host name verification and accept self-signed SSL certificates from the OIDC provider.

The following 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.

Copyright © 2025 Ping Identity Corporation

OIDC mode (generic)

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 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 host name verification and accept self-signed SSL certificates from the OIDC provider.

The following 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.

Copyright © 2025 Ping Identity Corporation

OIDC mode (custom scope)

Example: Set up the PingAuthorize Policy Editor in OIDC mode (custom scope)

This example sets up the PingAuthorize Policy Editor for sign-ons using OIDC with one or more custom scopes.

In OIDC mode, the Policy Editor UI requests an access token with the following default scopes: `openid email profile`. You can change the default requested scopes persistently, during server setup, or on a one-time basis, at server startup.

Add OIDC scopes during setup

To add requested OIDC scopes persistently, use the `--scope` option to provide a space-separated list of scopes to the **`setup`** command.

```
$ bin/setup oidc \
  --oidcBaseUrl https://auth.example.com/02fa3993-a851-4eb5-96c7-f0c561be23c6/auth/ \
  –-clientId 21a74125-85db-4fca-8a56-e5d45d4d8163 \
  --scope "openid email profile  <additional_scope>" \
  --generateSelfSignedCertificate \
  --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 host name verification and accept self-signed SSL certificates from the OIDC provider. The following 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/02fa3993-a851-4eb5-96c7-f0c561be23c6/auth/ \
  –-clientId 21a74125-85db-4fca-8a56-e5d45d4d8163 \
  --scope "openid email profile  <additional_scope>" \
  --generateSelfSignedCertificate \
  --hostname  <pap-hostname>  \
  --port  <pap-port>  \
  --adminPort  <admin-port>  \
  --licenseKeyFile  <path-to-license>
```

Add OIDC scopes at startup

To override persistently requested OIDC scopes for a single runtime instance of the Policy Editor, use the `PING_SCOPE` environment variable to provide a space-separated list of scopes to the **`start-server`** command:

```
$ env PING_SCOPE="openid email profile  <different_scope>" bin/start-server
```

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

Copyright © 2025 Ping Identity Corporation

OIDC mode (self-governance)

Example: Set up the PingAuthorize Policy Editor in OIDC mode (self-governance)

This example sets up the PingAuthorize Policy Editor with self-governance and OIDC authentication.

For more information about configuring OIDC authentication, see the **OIDC mode (generic)** tab on this page.

> **Note**
>
> Self-governance is not supported in clustered Policy Editor configurations.

To enable self-governance with OIDC authentication, use the following arguments:

`--enableSelfGovernance` ***(required)***

Turns on the self-governance functionality.

`--selfGovernanceSystemUser` ***(required)***

Sets the self-governance administrator username for OIDC authentication.

`--apiHttpCacheTtl` ***(optional)***

Sets the time-to-live value (in seconds) for the HTTP cache, after which the cache is refreshed and a new self-governance check is performed. This value must be 1 or greater.

> **Note**
>
> If you don't specify a value, the Policy Editor uses the default time-to-live of 60 seconds.

The following example sets up the Policy Editor to use PingOne for OIDC authentication, enables self-governance, and specifies an OIDC username for the self-governance administrator:

```
$ bin/setup oidc \
--hostname localhost \
--port 9443 \
--adminPort  <admin-port>  \
--oidcBaseUrl https://auth.pingone.com/<my-environment-id>/as \
--clientId  <my-client-id>  \
--generateSelfSignedCertificate \
--enableSelfGovernance \
--selfGovernanceSystemUsername  <oidc-authenticated-user>
```

Copyright © 2025 Ping Identity Corporation

Demo mode

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).

Copyright © 2025 Ping Identity Corporation

Demo mode (PostgreSQL)

Example: Set up the PingAuthorize Policy Editor with a PostgreSQL policy database

This example sets up the PingAuthorize Policy Editor in demo mode with the following options:

- Automatically generated self-signed server certificate
- PostgreSQL policy database with server runtime credentials (see the following caution about `--dbAppPassword`)

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

Caution

Using the `--dbAppPassword` option to provide the PostgreSQL database password to the **`setup`** tool persists the password to a configuration file.
Instead, omit `--dbAppPassword` entirely to persist the default password, and set the `PING_DB_APP_PASSWORD` environment variable before server start. For example:

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

Copyright © 2025 Ping Identity Corporation

Demo mode (custom SSL certificate)

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` option.

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`**.

Copyright © 2025 Ping Identity Corporation

Demo mode (self-governance)

Example: Set up the PingAuthorize Policy Editor in demo mode (self-governance)

This example sets up the PingAuthorize Policy Editor in demo mode with self-governance enabled.

For more information about setting up the Policy Editor in demo mode, click the **Demo mode** tab on this page.

To enable self-governance in demo mode, use the `--enableSelfGovernance` argument. The following values are set by default:

- The time-to-live value for the HTTP cache is set to 60 seconds, after which the cache is refreshed and a new self-governance check is performed.
- The self-governance administrator username is set to `selfgovernanceadmin`.
- The self-governance administrator password is set to `password123`.

The following example sets up the Policy Editor in demo mode with self-governance enabled:

```
$ bin/setup demo \
--adminUsername admin \
--enableSelfGovernance \
--generateSelfSignedCertificate \
--licenseKeyFile /opt/PingAuthorize/PingAuthorize.lic \
--decisionPointSharedSecret pingauthorize \
--hostname localhost \
--port 9443 \
--adminPort  <admin-port>
```

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.

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.

Copyright © 2025 Ping Identity Corporation

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

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

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 "{PAP_Name}" \
  --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:{PAP_Name}"
  --set trust-framework-version:{TRUST_FRAMEWORK_VERSION}
```

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.

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.
- Configure a user store.
- Sign on to the administrative console to configure endpoints for existing JSON APIs.

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

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

  For more information, see the PingAuthorize Policy Administration Guide.

Copyright © 2025 Ping Identity Corporation

Signing on to the administrative console

After deploying or installing the server, access the administrative console to verify the server configuration and manage the server settings.

Docker deployment

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. Learn more in Get Started.

   ```
   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, refer to 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.

Copyright © 2025 Ping Identity Corporation

Manual installation

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`.

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.

   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 Policy Editor authentication mode

Change the Policy Editor's authentication mode after initial setup.

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
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/{PAP_CONTAINER_NAME}:<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 single sign-on (SSO) requests from PingAuthorize.

If you chose OIDC mode when you set up the Policy Editor, you must configure an OIDC provider, such as PingFederate or PingOne, to accept sign-on requests from the Policy Editor. Refer to the following tabs for the configuration steps for PingOne and PingFederate.

If you're using another OIDC provider, refer to 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	
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 the credential to the token endpoint.
Return ID token on refresh grant	`true`

Copyright © 2025 Ping Identity Corporation

OAuth 2 client configuration	Configuration value
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 the refresh token 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. Learn more in Defining issuance criteria for IdP Browser SSO in the PingFederate documentation.

Copyright © 2025 Ping Identity Corporation

Configuring PingOne

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
About this task

Configure PingOne to authorize external access to the Policy Editor.

Steps

1. Sign on to PingOne and click your environment.

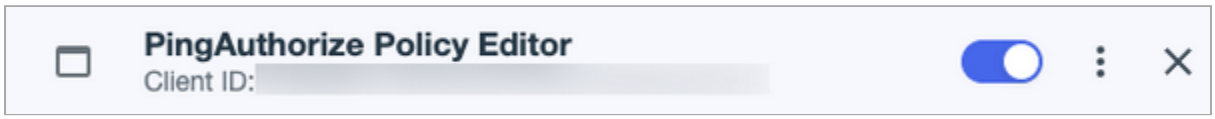
   Choose from:

   - If you have an account, go to the URL for your environment. Each environment has a unique URL for signing on that follows the https://console.pingone.com/?env=<environmentID> format.
   - If you don't already have a PingOne account, create one at Try Ping.

2. To create an application in PingOne to represent the Policy Editor, go to **Connections** > **Applications** and click the **+** icon.
3. Enter a name for the application, such as `PingAuthorize Policy Editor`.
4. (Optional) Enter a description and add an icon.
5. Click **OIDC Web App**, and then click **Save**.
6. On the **Configuration** tab, click the **Pencil** icon to edit the settings.
7. In the **PKCE Enforcement** list in the **Grant Type** section, select **S256_REQUIRED**.
8. In the **Redirect URIs** field, enter a redirect URL that follows the format `https://<pap.hostname:port>/idp-callback`.
9. In the **Token Endpoint Authentication Method** section, click **None**.
10. Click **Save**.

Copyright © 2025 Ping Identity Corporation

11. On the **Resources** tab, click the pencil icon to edit the settings.
12. In the **Scopes** list, click the **+** icon to add the **email** and **profile** scopes to the **Allowed Scopes** list.
13. Click **Save**.
14. Click the toggle to enable the application.



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 the client ID and the environment ID to configure the Policy Editor to use PingOne.

Configuring PingAuthorize policy administration to use PingOne
About this task

The following configuration enables the Policy Editor to use PingOne for authentication.

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, run the following command to configure the Policy Editor:

```
bin/setup oidc \
  --licenseKeyFile <path to PingAuthorize.lic> \
  --generateSelfSignedCertificate \
  --hostname <pap-hostname> --port <pap-port> \
  --adminPort <admin-port> \
  --oidcBaseUrl https://auth.pingone.<regional domain>/<environment id>/as \
  --clientId <client-id>
```

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

       Result:

       Your browser redirects to the URL you set in Configuring PingOne for PingAuthorize policy administration.

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> By default, the signed-on username uses the `sub` JSON Web Token (JWT) claim for the OIDC user ID. You can find details on using a non-default claim in Changing the default JWT claim for the OIDC user ID.

Copyright © 2025 Ping Identity Corporation

Configuring PingFederate

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

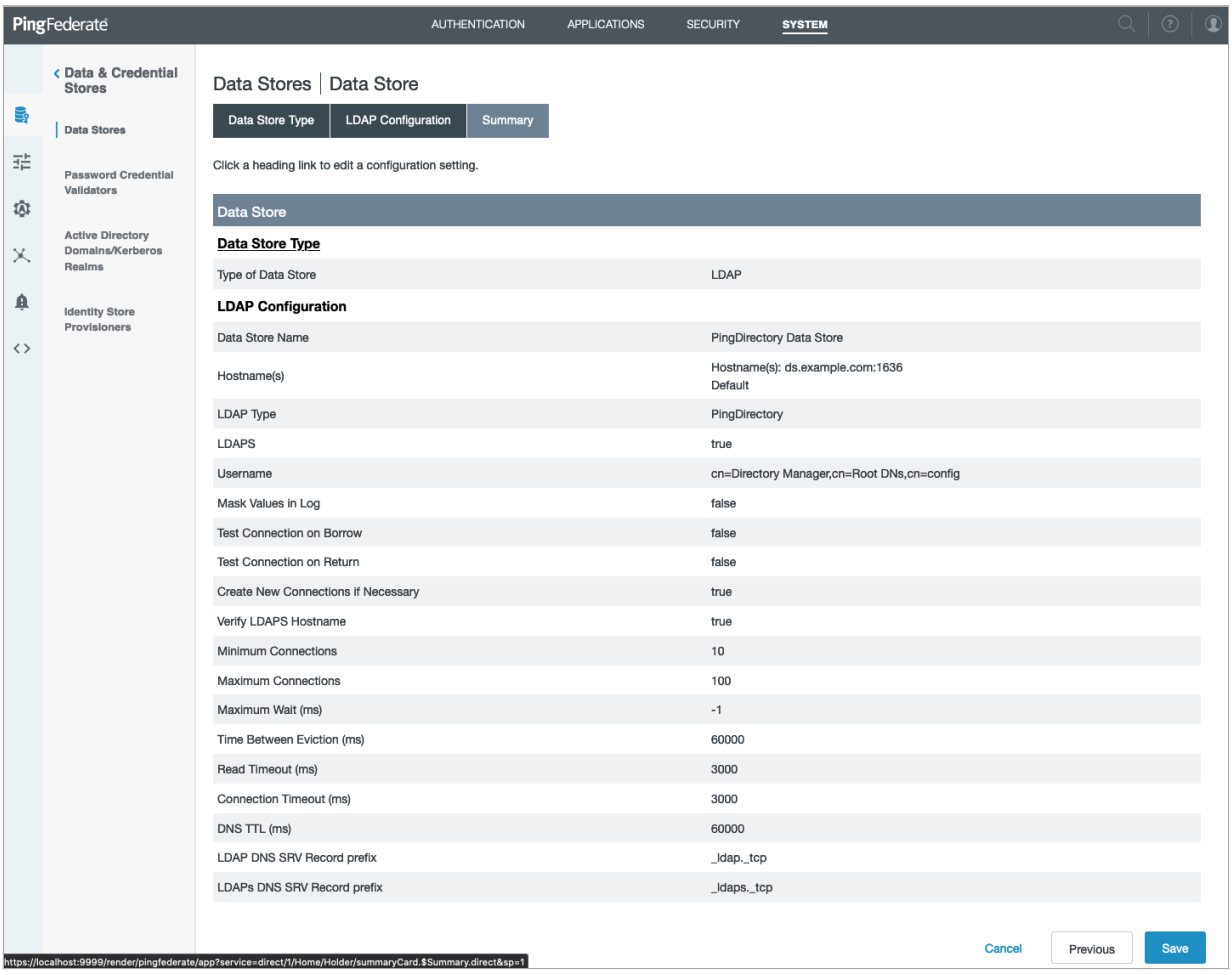
You can also use PingAccess to authorize external access through rules. You can find more information in Rule Creation in PingAccess.

Copyright © 2025 Ping Identity Corporation

The following example configuration assumes that any authenticated user can access the PingAuthorize Policy Editor. You can find more informarion about limiting access to members of a specific group in Configuring PingFederate group access for PingAuthorize.

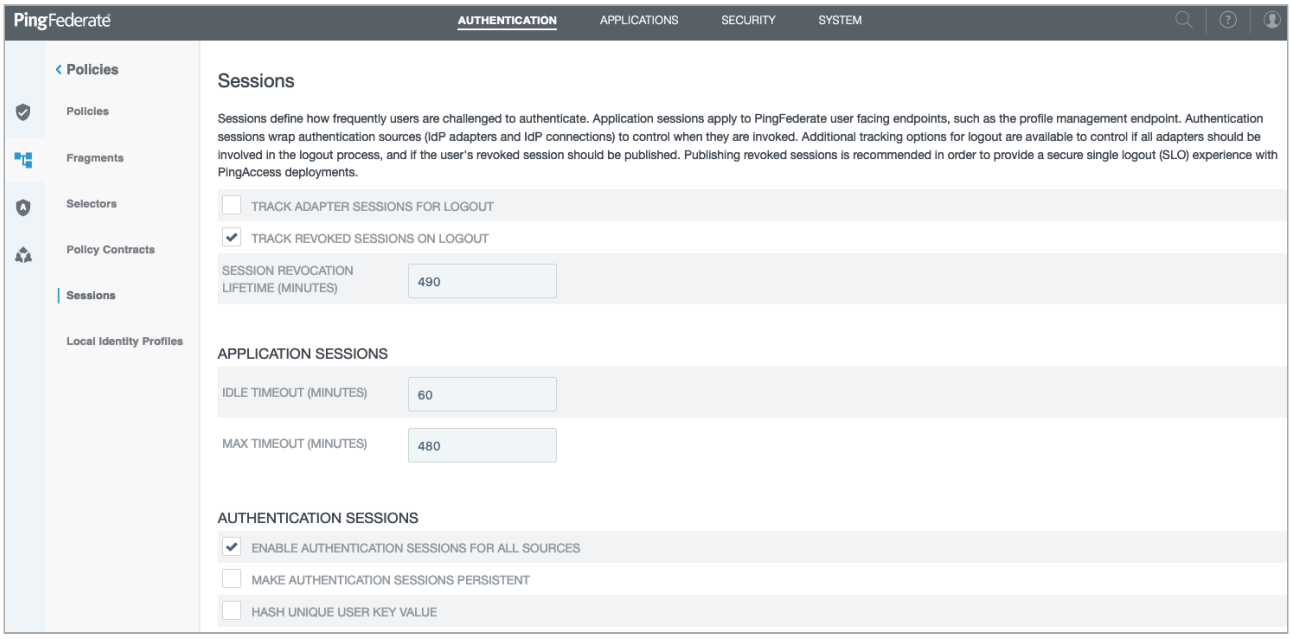
Steps

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**.

Copyright © 2025 Ping Identity Corporation

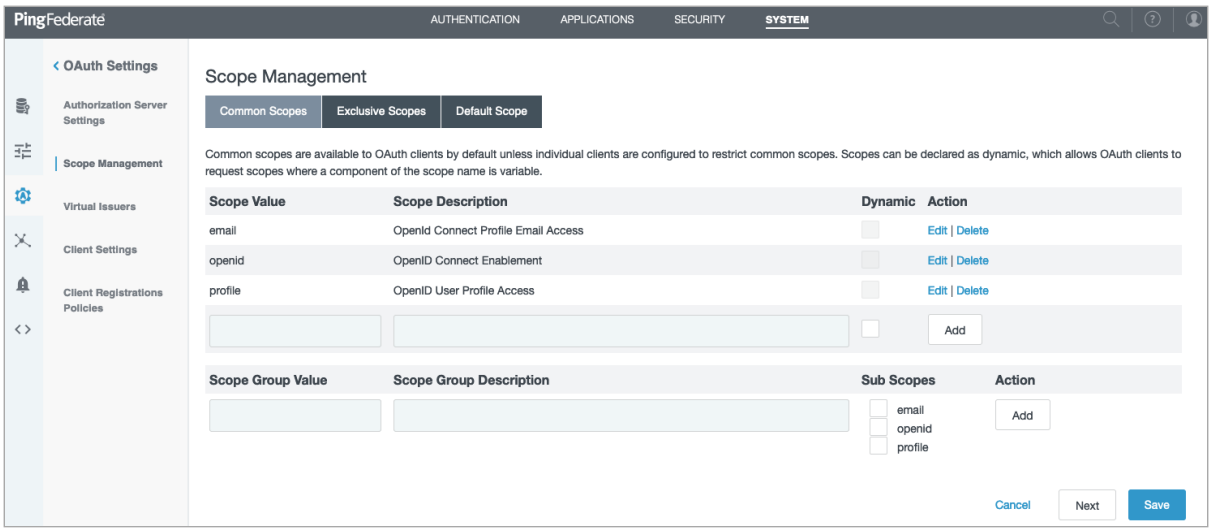


8. Go to **Security > Certificate & Key Management > SSL Client Keys & Certificates** and import your JWT signing certificate. Click **Save**.

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

9. Configure your OAuth server using the OpenID Connect protocol.

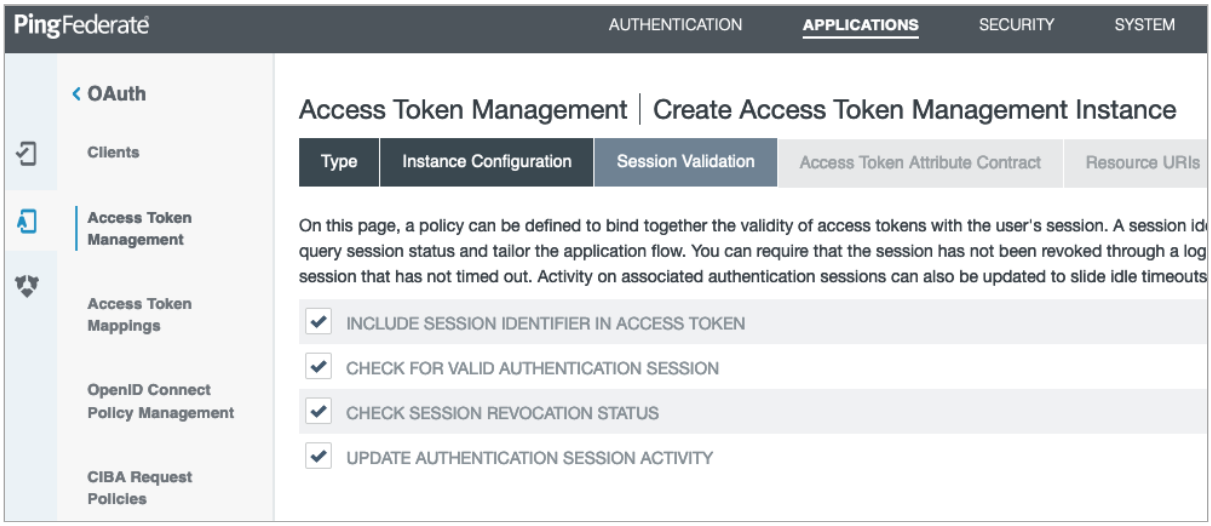
    1. Go to **System > OAuth Settings > Scope Management** and create scopes.
    2. In the **Scope Value** field, enter the `email`, `openid`, and `profile` scopes, clicking **Add** after each entry. Click **Save**.



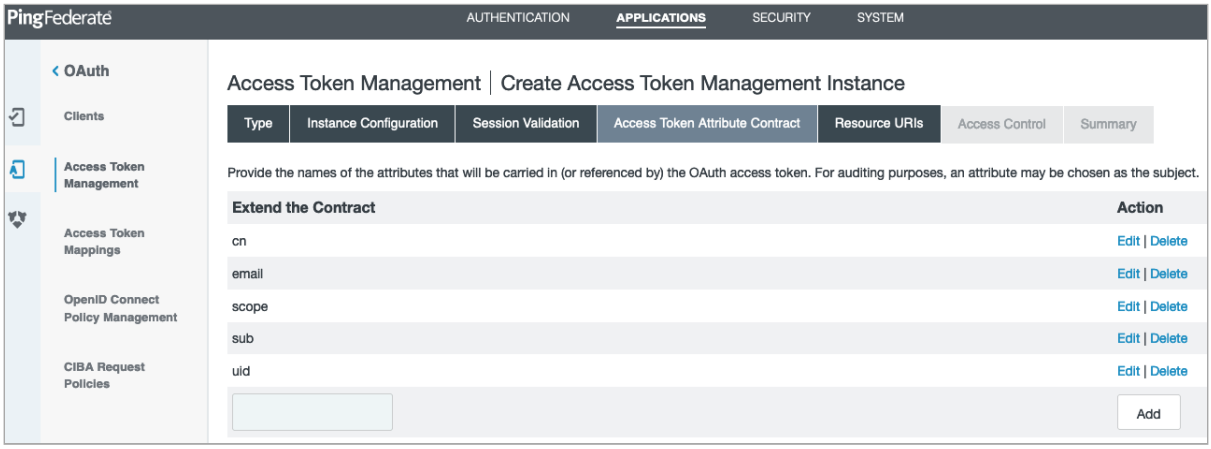
    3. Go to **Applications > OAuth > Access Token Management** and click **Create New Instance**.
    4. On the **Type** tab, from the **Type** list, select **JSON Web Tokens**. From the **Parent Instance** list, select **None**. Click **Next**.

Copyright © 2025 Ping Identity Corporation

5. 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**.
6. On the **Session Validation** tab, enable the session validation options.

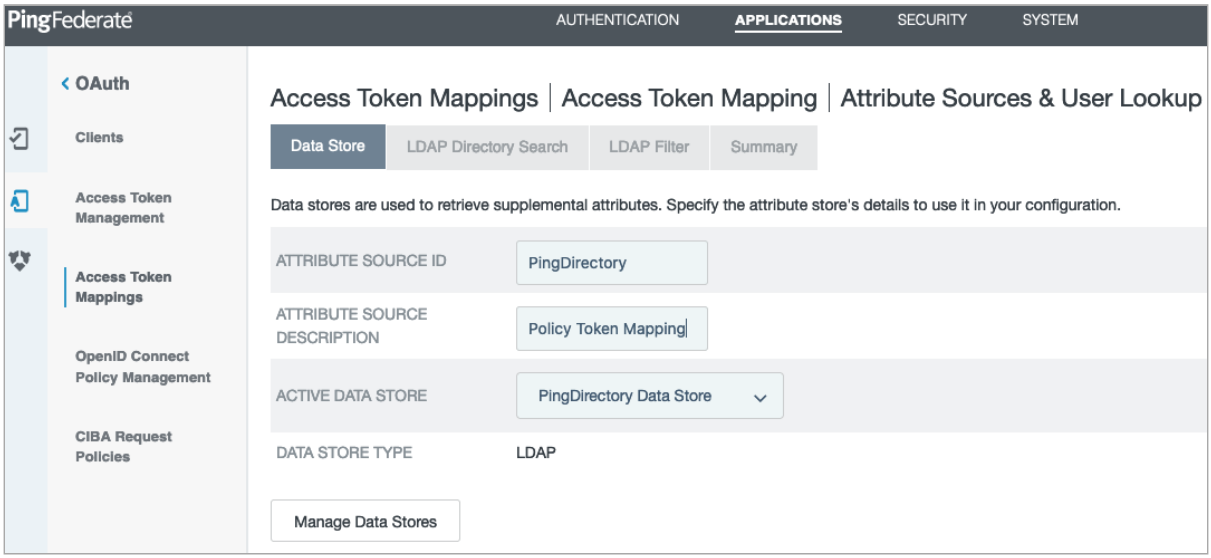


7. On the **Access Token Attribute Contract** tab, add the attributes to be included in the OAuth access token. This example extends the contract with `cn`, `email`, `scope`, `sub`, and `uid` attributes.



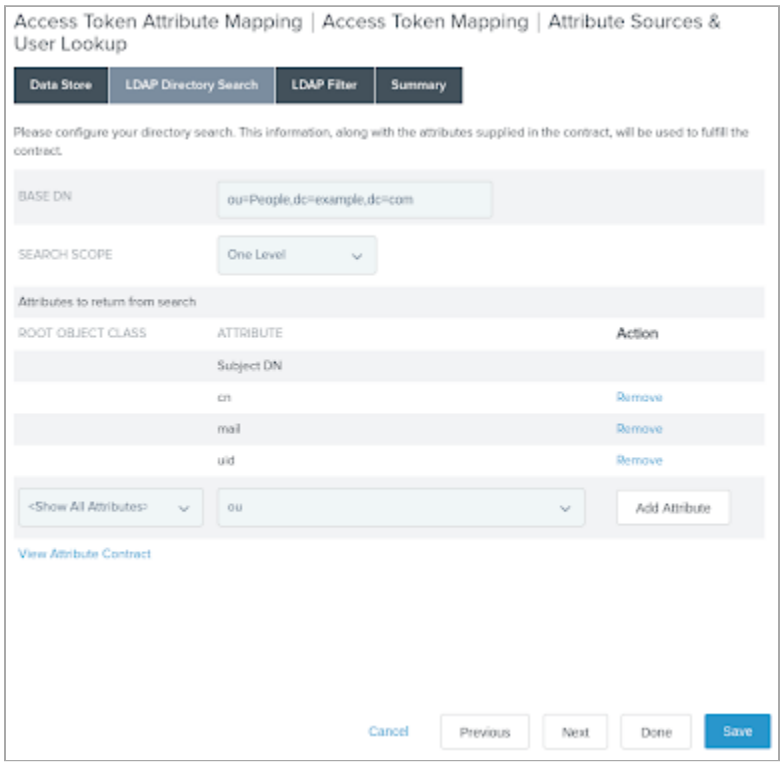
8. Click **Next** until you reach the **Summary** tab, and then click **Save**. Accept the default values for the **Resources URIs** and **Access Control** settings.
9. Go to **Applications > OAuth > Access Token Mappings** to create an Access Token Mapping in the **Default** context for the Access Token Manager you just created. Click **Add Mapping**, and then click **Add Attribute Source**.
10. From the **Active Data Store** list, select the PingDirectory data store that you created in step 2. Click **Next**.

Copyright © 2025 Ping Identity Corporation



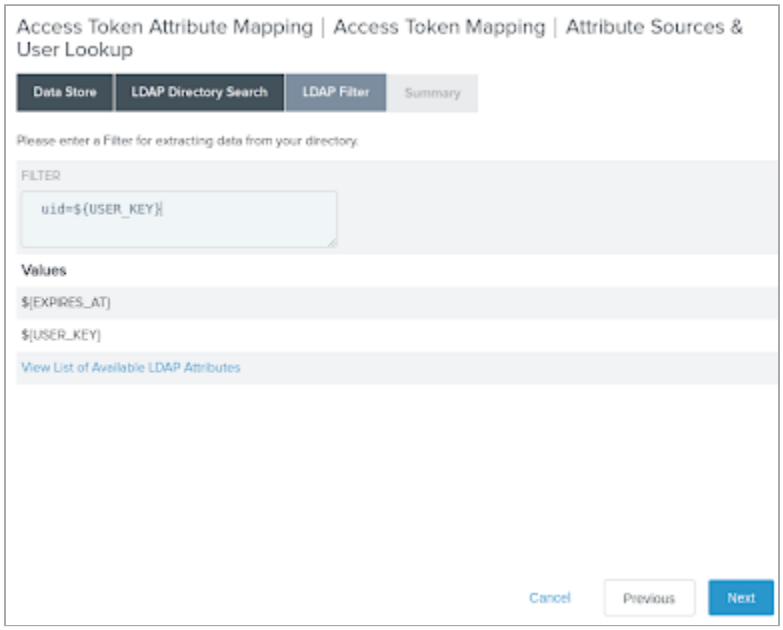
11. On the **LDAP Directory Search** tab, in the **Base DN** field, enter the base DN for the PingDirectory data that provides your identities.
12. In the **Attributes to return from search** section, click **Add Attribute** and enter the attributes to be retrieved.

    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.



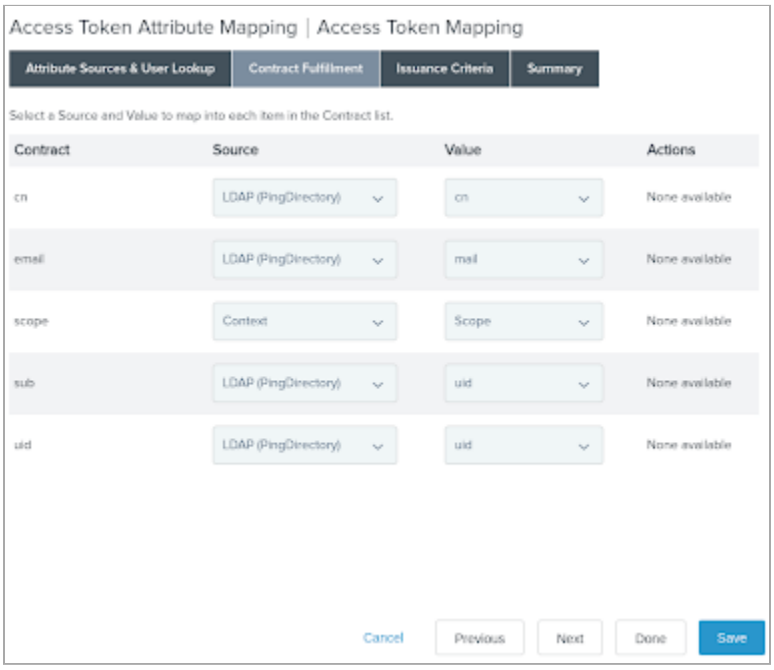
13. On the **LDAP Filter** tab, in the **Filter** field, enter `uid=${USER_KEY}` to match the PingDirectory sample data with the authenticating user information.

Copyright © 2025 Ping Identity Corporation



14. Click **Next** and **Save** on the **Summary** tab.
15. 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**.

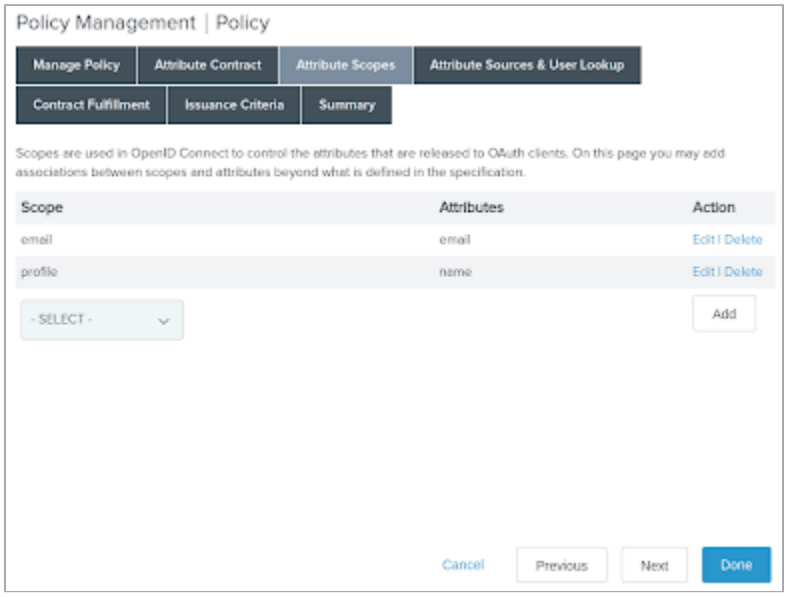
    The scope attribute is fulfilled from the OAuth context.



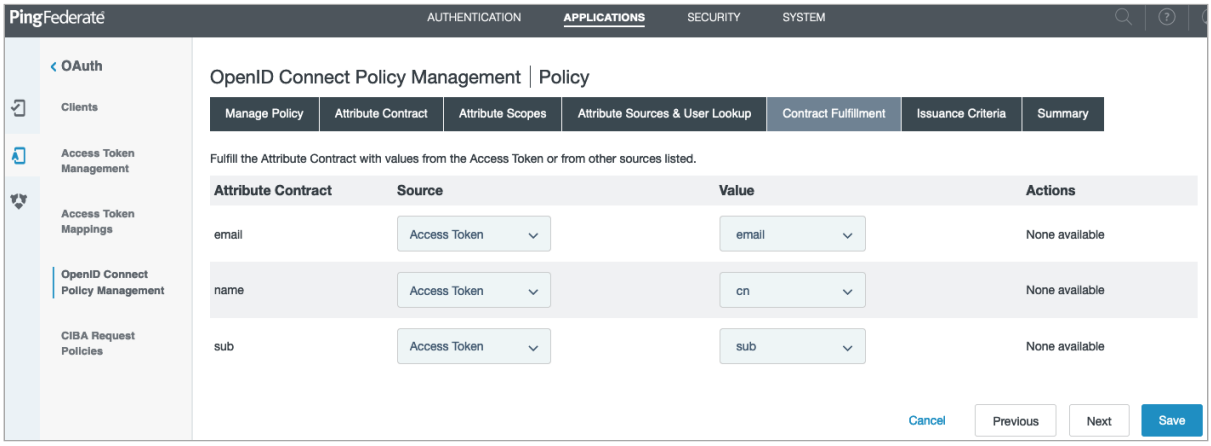
16. Go to **Applications > OAuth > OpenID Connect Policy Management** and click **Add Policy**.
17. In the **Manage Policy** tab, from the **Access Token Manager** list, select the access token manager you previously created.
18. Ensure that the **Include User Info in ID Token** checkbox is selected. Click **Next**.

Copyright © 2025 Ping Identity Corporation

19. On the **Attribute Contract** tab, extend the policy contract with the `email` and `name` attributes. Click **Next**.
20. On the **Attribute Scopes** tab, map the previously defined `email` and `profile` scopes to the `email` and `name` ID token attributes. Click **Next**.



21. 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**.



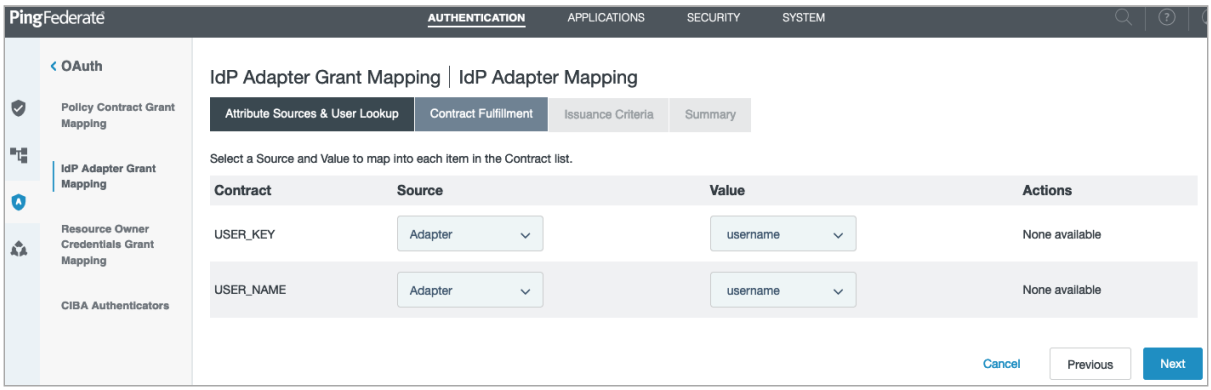
22. Click **Set as Default** to set the newly created policy as the default policy.
23. Go to **Applications > OAuth > Clients** and click **Add Client**.

    To provide the Policy Editor with appropriate defaults, configure PingFederate with a **Client ID** of `pingauthorize-pap` and select the **Authorization Code** checkbox 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.

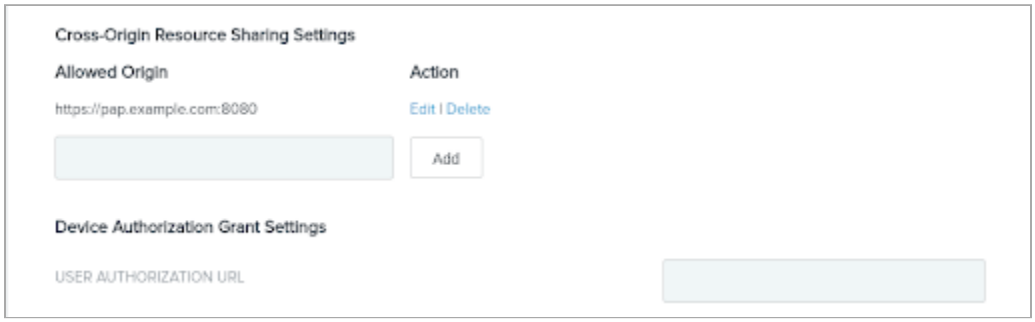
    Click **Save**.

Copyright © 2025 Ping Identity Corporation

24. 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.



25. 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.

Configuring the PingAuthorize Policy Editor to use PingFederate

Configure the Policy Editor to use PingFederate for authentication.

Before you begin

Configure PingFederate to handle OpenID Connect requests as described in Configuring PingFederate for PingAuthorize.

Steps

1. Add the certificate to the Java Trust Store.

Copyright © 2025 Ping Identity Corporation

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.
    1. Stop the application.

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

    2. Re-run **`bin/setup`** to reconfigure the application.
    3. 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
    ```

Copyright © 2025 Ping Identity Corporation

4. 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
```


Copyright © 2025 Ping Identity Corporation

5. 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
```

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

3. Verify that you can log into the Policy Editor using OpenID Connect provided by PingFederate.

    1. Go to the Policy Editor, for example, https://pap.example.com:9443. Your browser should be redirected into the OAuth flow.

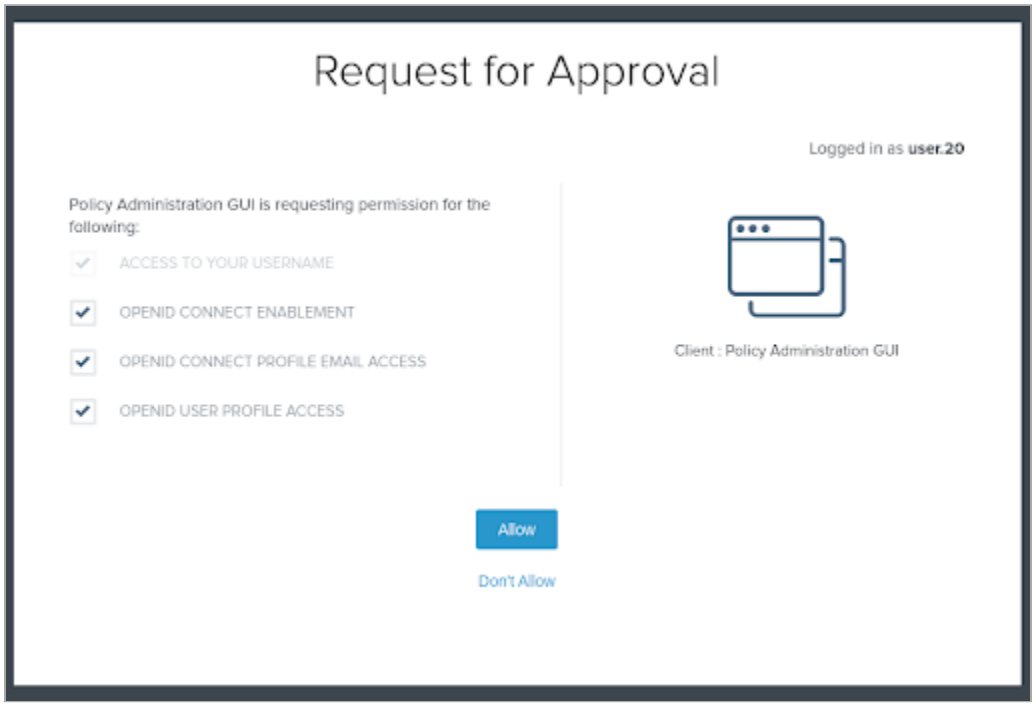
    2. Click **Click to Sign In**.

    3. Sign on with your user name and password.

Copyright © 2025 Ping Identity Corporation

The sample configuration in this document creates an identity with the user name `user.20` and password `password`.

4. 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.

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

Copyright © 2025 Ping Identity Corporation

Steps

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

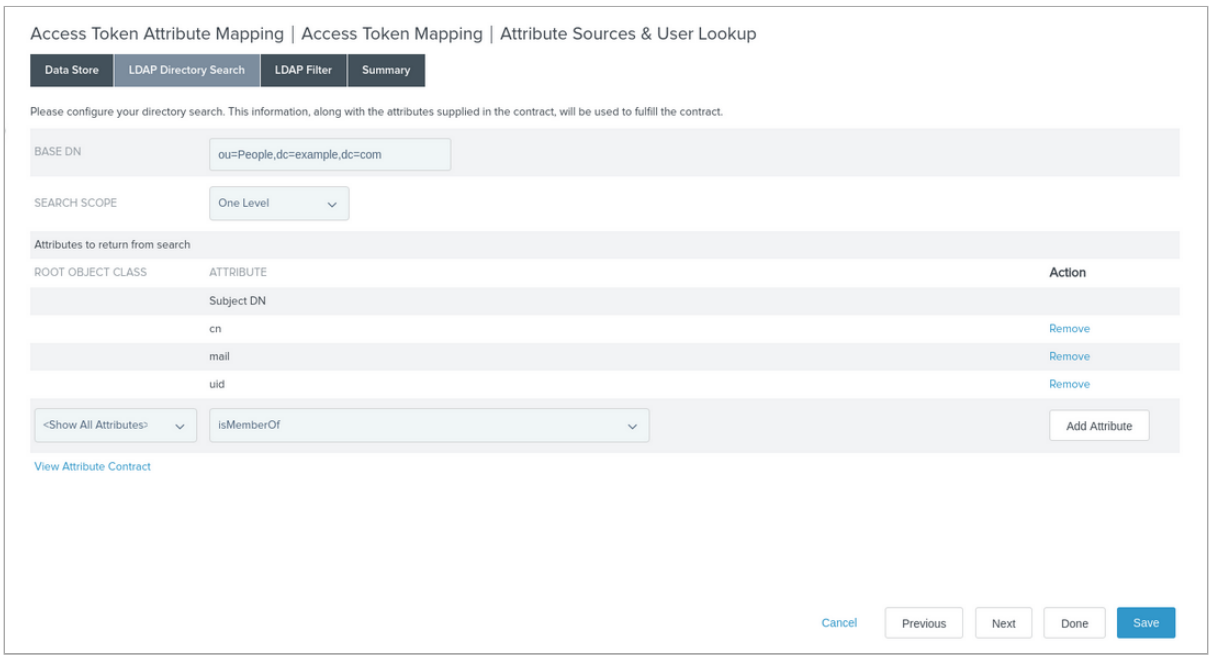
```
dn: ou=groups,dc=example,dc=com
objectclass: top
objectclass: organizationalunit
ou: groups

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
```

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

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

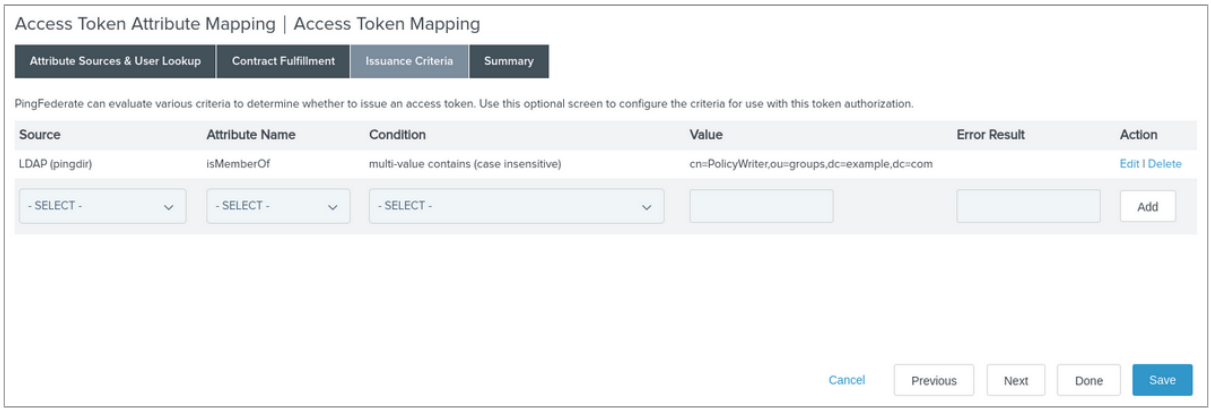
2. Add the group membership claim requirement in PingFederate.
    1. In the PingFederate console, go to **Applications > OAuth > Access Token Mappings**.
    2. Select the PingDirectory mapping from the list, and then on the **Attribute Sources & User Lookup** tab, select the PingDirectory source.
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



Copyright © 2025 Ping Identity Corporation

5. 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)**
Value	**cn=PolicyWriter,ou=groups,dc=example,dc=com**



6. 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.

1. Sign on as `user.0`.

   Result:

   Access is denied.

2. Sign on as `user.20`.

   Result:

   Access is granted.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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, refer to Planning your upgrade.

Copyright © 2025 Ping Identity Corporation

Considerations when upgrading to 10.3 or later

Custom SDK extensions using Javax must be migrated and recompiled

Several components were upgraded in version 10.3 of PingAuthorize. If any of your custom Server SDK extensions have classes that import `javax.*` packages, you must migrate them to the equivalent `jakarta.*` packages and then recompile the extensions.

Camel upgrade

PingAuthorize 10.3 now supports Apache Camel 4.0. The limitations on using Apache Camel to connect policy information points (PIP) introduced in PingAuthorize 9.3 still apply. You can find more information on working around these limitations in Enabling Camel service connections. Learn how to upgrade PingAuthorize versions 10.0-10.2 using Camel 3.x in the Apache Camel 3.x to 4.0 migration guide.

Java considerations

Support for Java 11 has been removed, and upgrading to Version 10.3 of PingAuthorize will fail unless you are running Java 17 or 21.

Prerequisites for upgrading

You must meet one of the following requirements:

- Your default Java installation is a supported version.
- You are pointing one of the following environment variables to a supported version of Java:
  - `UNBOUNDID_JAVA_HOME`
  - `JAVA_HOME`

> **Note**
>
> If you use environment variables to point to your Java installation, these will override your default Java version. We recommend you set only one variable. If both are set, `UNBOUNDID_JAVA_HOME` takes precedence.

> **Important**
>
> Java 11 is no longer supported. You can't upgrade a server instance to version 10.3 without first updating Java to a supported version.

Upgrading from a server running a supported version of Java

If you're upgrading a server running Java 17 or 21 to version 10.3, you can proceed with the server upgrade.

Copyright © 2025 Ping Identity Corporation

Upgrading from a server running an unsupported version of Java

Before upgrading the server to version 10.3, you must install either Java 17 or 21. Learn more in System requirements. Upgrading to version 10.3 after updating Java requires changes to the `java.properties` file.

> **Important**
>
> You must also meet one of the prerequisites listed in the previous section before upgrading the server.

Select one of the following options for modifying the `java.properties` file. Where a Java version is specified, substitute your installed, supported Java version.

- Option 1: Before upgrading the server, convert the file manually:
    1. Edit the `config/java.properties` file to update the Java version and convert the JVM parameters to be specific to Java 17.
    2. Run the **`bin/dsjavaproperties`** command to put the changes into effect.
- Option 2: Before upgrading the server, create a new file:
    1. Rename the old `java.properties` file.
    2. Run the **`bin/dsjavaproperties`** command to initialize a new Java 17 `java.properties` file.

       For this option, run:

       ```
       bin/dsjavaproperties --initialize
       ```

    3. Upgrade the server using the generated `java.properties` file, and then restore your customized settings from the original file.
- Option 3: Allow the upgrade to replace the file:
    1. Upgrade the server to version 10.3.

       The upgrade process will overwrite the `java.properties` file and the original file will be saved as `java.properties.old`. A `java.properties.change` file will also be created, containing the diff output between the new and old `java.properties` files.
    2. Restore or convert the JVM settings that were overwritten during the upgrade process.

Upgrading a Policy Editor instance running an unsupported version of Java

If you are upgrading from a Policy Editor instance running an unsupported version of Java, you must export the `JAVA_HOME` environment variable by running the following command:

Copyright © 2025 Ping Identity Corporation

```
export JAVA_HOME=$JAVA11_HOME
```

You must perform this export before running any scripts in PingAuthorize 10.3, including `bin/setup` and `bin/start-server`.

Considerations when upgrading to 10.1 or later

Resolving issues with copying policy elements

After upgrading from a version of PingAuthorize earlier than 10.1 and attempting to delete a copy of a Policy Editor element, that copy might persist in the UI and return an error when selected. To completely remove deleted copies from the Policy Editor, follow the steps in Deleting persistent copies.

Considerations when upgrading to 10.0 or later

SNI hostname checking disabled by default

If you are upgrading to PingAuthorize 10.0 with an existing configuration that has SNI hostname checks enabled, you might encounter an issue when using a host name not found in the key store. To migrate an existing configuration from an earlier version of PingAuthorize and disable SNI host name checks, add the following to your `configuration.yml` file:

```
server:
  ...
  applicationConnectors:
  - type: "https"
    ...
    disableSniHostCheck: "${PING_DISABLE_SNI_HOSTNAME_CHECKS:-true}"
```

This change also introduces a new `setup` option, `--disableSniHostnameChecks`, that you can set to `false` to enable SNI hostname checks.

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.

Copyright © 2025 Ping Identity Corporation

Upgrading the PingAuthorize Policy Editor using Docker

If you originally installed the Policy Editor with Docker per Deploying the Policy Editor using Docker, 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.
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.

   > **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 in Deploying the Policy Editor using Docker 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/{PAP_CONTAINER_NAME}:<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).

Copyright © 2025 Ping Identity Corporation

> **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/{PAP_CONTAINER_NAME}:<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.

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}
   ```


Copyright © 2025 Ping Identity Corporation

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.
2. Stop the Policy Editor.

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

3. Obtain the new version of the PingAuthorize Policy Editor and extract it to 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.

Copyright © 2025 Ping Identity Corporation

This example assumes the old installation is in `/opt/pingauthorize/PingAuthorize-PAP-previous`, and the new installation is in `/opt/pingauthorize/PingAuthorize-PAP`. Run the following commands to upgrade from each version:

Step 8.1 and later

```
$ cp /opt/pingauthorize/{pingauthorize}-PAP-previous/Symphonic.mv.db opt/pingauthorize/
{pingauthorize}-PAP
```

Step 8.0 earlier

```
$ 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 instructions for Upgrading a PostgreSQL policy database.

5. Run the **`setup`** tool.

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

> **Warning**
>
> The **`setup`** tool uses the default credentials to upgrade the database. If the credentials no longer match the default values, provide the correct credentials to the **`setup`** tool using the appropriate command-line options:
>
> - If you are using the default H2 policy database implementation, provide the non-default values 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.
> - If you are using a PostgreSQL policy database implementation, provide the server runtime user value through the `--dbAppUsername` command-line option. For the server runtime password, provide this value to the `PING_DB_APP_PASSWORD` environment variable before server start.

Follow the instructions in either of the following topics:

- Installing the Policy Editor interactively
- Installing the Policy Editor non-interactively

6. Start the new Policy Editor.

   Follow the instructions in Post-setup steps (manual installation).

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

Copyright © 2025 Ping Identity Corporation

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
- Upgrading the Trust Framework and policies
- Upgrading a PostgreSQL policy database

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.

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 hamburger menu 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`.

Copyright © 2025 Ping Identity Corporation

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:{TRUST_FRAMEWORK_VERSION}
   ```

Upgrading a PostgreSQL policy database

To upgrade an existing PostgreSQL policy database, use the `policy-db` tool.

Before you begin

- The PostgreSQL instance must be reachable on the network from the Policy Editor host and listening for connections.
- The Policy Editor uses both a PostgreSQL administration user and a server runtime user. Have a database administrator create both users before providing their credentials to the `policy-db` tool. The administration user must be able to create new databases. When new releases of the Policy Editor become available, continue using the same administration user to prevent database object ownership issues.

  Learn more about creating new database users and configuring PostgreSQL to listen for remote connections securely in the PostgreSQL documentation.

Copyright © 2025 Ping Identity Corporation

- The Policy Editor uses Java Database Connectivity (JDBC) to connect to PostgreSQL. Be prepared to provide the JDBC connection string in the following format: `jdbc:postgresql://<host>:<port>/<name>`. For example: `jdbc:postgresql://example.com:5432/pap_db`

About this task

When a newer version of the Policy Editor is released, it will fail to start when pointing to a PostgreSQL database with older database objects. Running **`start-server`** will print a message like the following to the console:

```
The policy database at 'jdbc:postgresql://example.com:5432/pap_db' is older than this version of
PingAuthorize Policy Editor (9.2.0.0).
Please use the policy-db tool to upgrade the database before running start-server again.
```

This message indicates that you must run the **`policy-db`** tool for this newer version of the Policy Editor to upgrade the database objects.

Follow these instructions to upgrade a PostgreSQL database for a manual upgrade of the Policy Editor. Be prepared to provide the database administration credentials and server runtime credentials you used to create the PostgreSQL database.

> **Note**
>
> See Deploying the Policy Editor using Docker for containerized deployments.

Steps

1. Complete steps 1-3 of Upgrading the PingAuthorize Policy Editor manually.
2. Run the following command:

   ```
   $ bin/policy-db \
     --dbConnectionString "jdbc:postgresql://<host>:<port>/<name>" \
     --dbAppUsername  <server-runtime-username>  \
     --dbAppPassword  <server-runtime-password>
   ```

   > **Note**
   >
   > Alternatively, you can provide the server runtime password through the `PING_DB_APP_PASSWORD` environment variable.

3. Provide the database administration credentials when prompted.

Result

The **`policy-db`** tool connects to PostgreSQL and applies the upgrades.

Next steps

Complete steps 5-7 of Upgrading the PingAuthorize Policy Editor manually.

Copyright © 2025 Ping Identity Corporation

Important

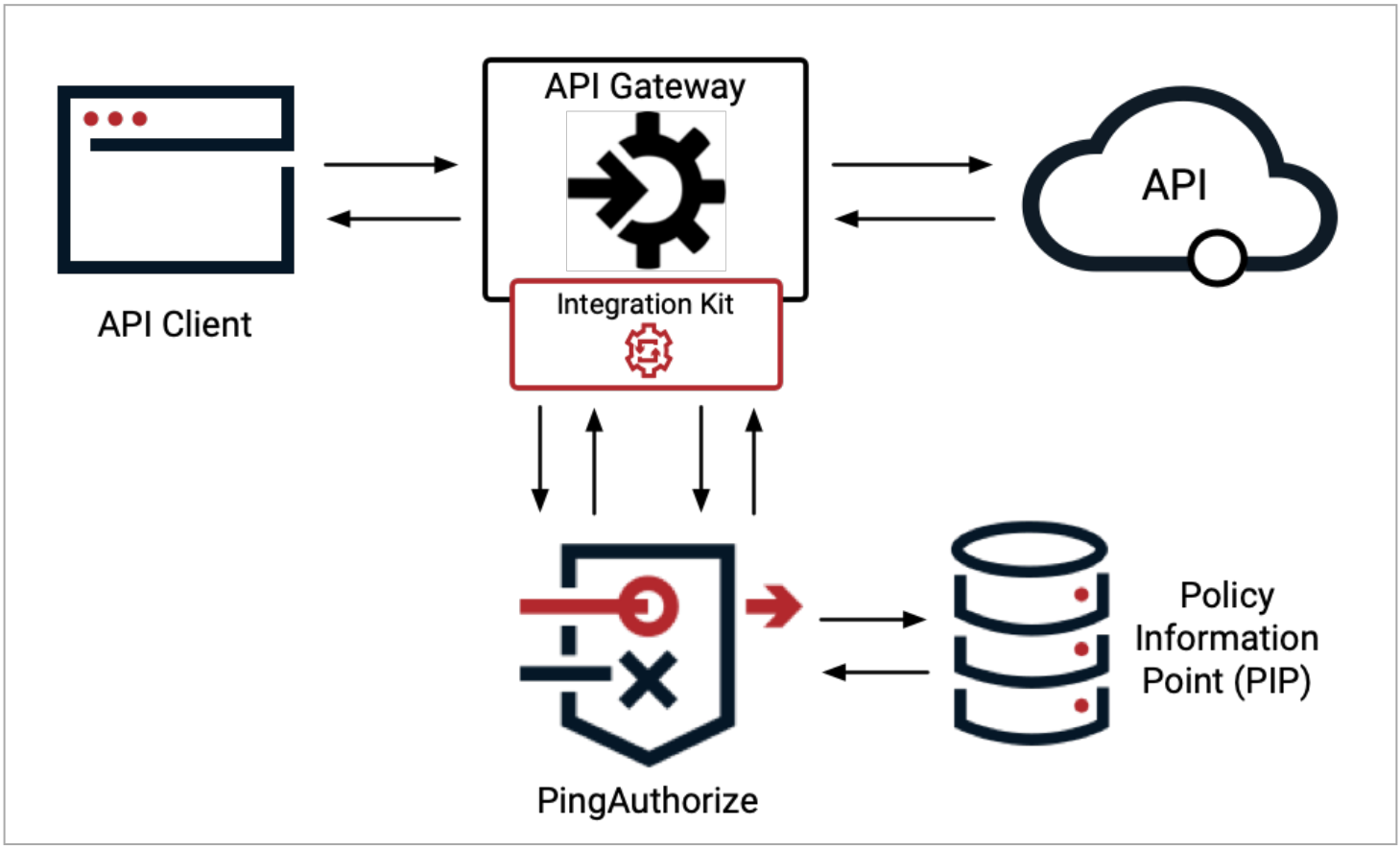
Provide the Policy Editor with the same `--dbConnectionString`, `--dbAppUsername`, and server runtime password you used to create the PostgreSQL database.

Copyright © 2025 Ping Identity Corporation

PingAuthorize Integrations

API gateway integrations enable you to use PingAuthorize for attribute-based access control and policy decisions with your application programming interface (API) gateway.

The following diagram outlines how API requests flow through your API gateway and PingAuthorize.



1. The client sends a request to the API gateway.
2. The API gateway-specific integration kit processes the client's request and sends it to the PingAuthorize Server for policy processing.
3. The PingAuthorize Server determines whether to permit or deny the API request based on policies defined in the PingAuthorize Policy Editor.
4. The API gateway analyzes the response from the PingAuthorize Server to determine whether the request should be forwarded to the upstream API and, if so, whether any modifications should be made to the request.
5. The API gateway passes the original or modified request to the API target.
6. The API resource server sends a response to the gateway with the requested resources.
7. The API gateway integration kit processes the resource server's response and forwards it to the PingAuthorize Server for policy processing.
8. The PingAuthorize Server determines whether to forward the API response to the client based on policies defined in the PingAuthorize Policy Editor.
9. The PingAuthorize Server sends a final response to the API gateway.

Copyright © 2025 Ping Identity Corporation

10. The API gateway processes the response and forwards the requested API resource to the client.

PingAuthorize provides the following API gateway integrations:

- Apigee API gateway integration
- Kong API gateway integration
- Kong Konnect integration
- MuleSoft API gateway integration

Apigee API gateway integration

Ping Identity's shared flow for Apigee extends Apigee's authorization capabilities through an external policy evaluation service.

Integration with Apigee allows centralized management of API access control and application protection in PingAuthorize while delegating enforcement to Apigee. Learn more about how this integration kit interacts with PingAuthorize in API gateway integration.

Install and configure the integration kit in Apigee to enable management of access control rules in PingAuthorize.

To configure the integration kit:

1. Set up PingAuthorize for Apigee integration.
2. Configure Apigee for PingAuthorize integration.

Setting up PingAuthorize for Apigee integration

To allow Apigee to use PingAuthorize as an external runtime authorization service, set up PingAuthorize to receive authorization requests from Apigee.

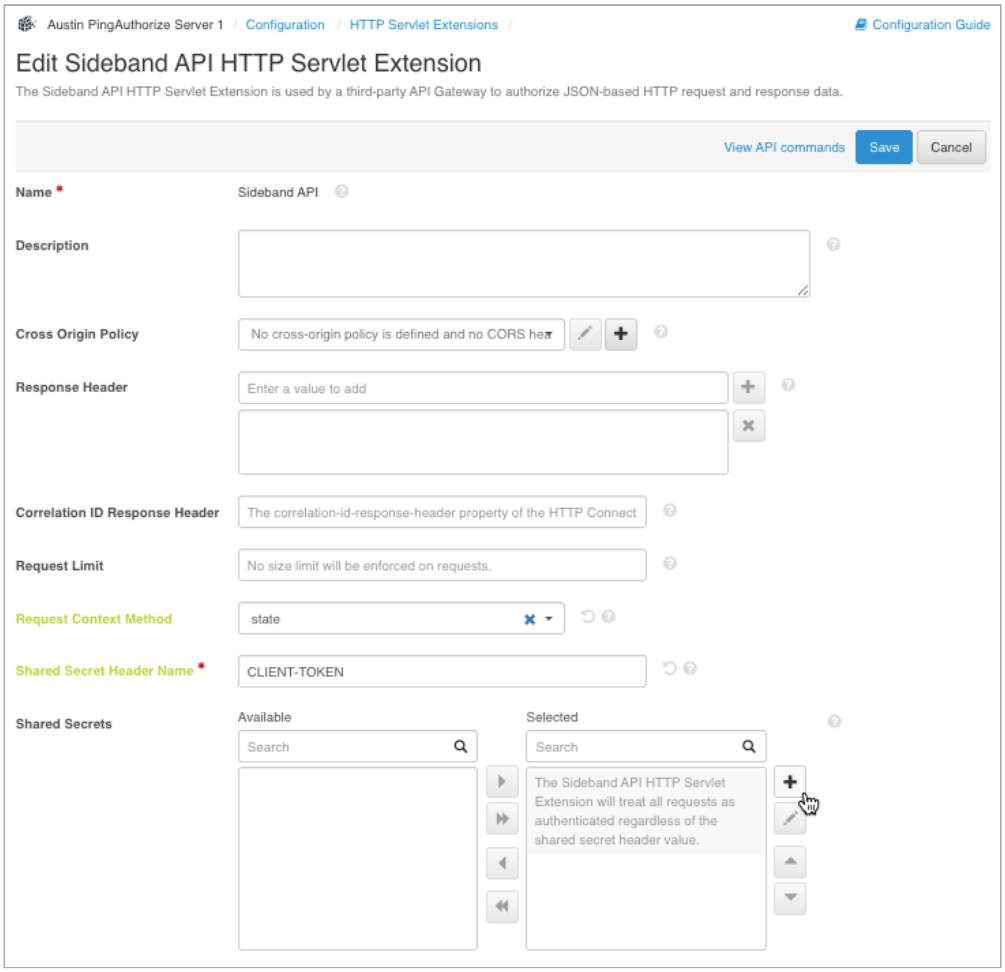
Before you begin

Install PingAuthorize.

Steps

1. On the **Configuration** page of the PingAuthorize administrative console, in the **Web Services and Applications** section, go to **HTTP Servlet Extensions > Sideband API**.
2. In the **Request Context Method** list, select **state**.

Copyright © 2025 Ping Identity Corporation



3. In the **Shared Secret Header Name** field, enter `CLIENT-TOKEN`.
4. To create a new shared secret, in the **Shared Secrets** section, click the **+** icon to the right of the **Selected** table.
5. In the **New Sideband API Shared Secret** modal, enter a shared secret name and value, and then click **Save To PingAuthorize Server Cluster**.
6. Click **Save**.

Configuring Apigee for PingAuthorize integration

Install the PingAuth shared flow bundle in Apigee and configure it to integrate with PingAuthorize.

Before you begin

Ensure you have:

- A supported Apigee environment. The Ping Identity shared flow for Apigee supports Apigee Edge, Apigee Private Cloud, and Apigee X.
- The PingAuth shared flow bundle `.zip` archive. Download the integration kit for Apigee from the Ping Identity Integration Directory.

Copyright © 2025 Ping Identity Corporation

Adding the PingAuthorize shared flow to Apigee

Steps

1. Upload the shared flow bundle:
    1. In Apigee, go to **Develop > Shared Flows** and do one of the following:
        - In Apigee X, click **Upload Bundle**.
        - In Apigee Edge or Apigee Private Cloud, click **+Shared Flow**, and then click **Upload Bundle**.
    2. For the shared flow name, enter `PingAuth`.
    3. In **File Picker**, select the PingAuth shared flow bundle `.zip` archive.
    4. Click **Create**.
2. In Apigee X, configure the connection to PingAuthorize.

> **Note**
>
> Skip this step if you're using Apigee Edge or Apigee Private Cloud.
> Apigee X doesn't support managing the configuration values stored in key-value maps through the Apigee UI. You must add these configuration values to the key-value map policy. The key-value map is created and the configuration values are added the first time the PingAuth shared flow executes at runtime.

    1. To access the PingAuth shared flow, go to **Develop > Shared Flows > PingAuth**.
    2. Click the **Develop** tab and examine **Revisions** to make sure you're on the latest revision.
    3. In the **Policies** panel on the left, click the **Load KVM Config** policy.
    4. In the **Code** panel, remove the comment lines above and below the `InitialEntries` element.
    5. Edit the value for `service_host_port` to match the host name of your PingAuthorize server instance and the port of the HTTPS connection handler.

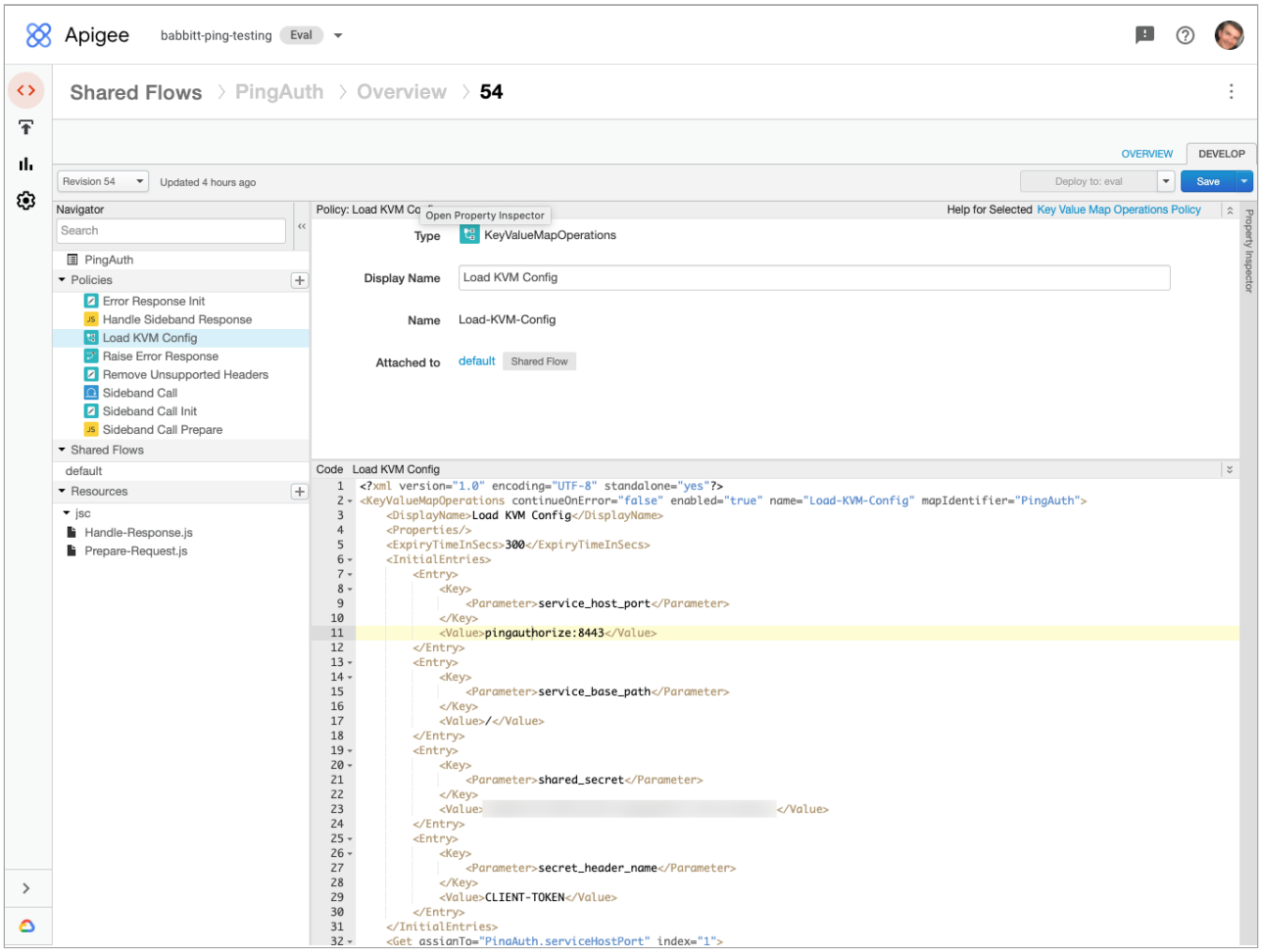
       For example, `pingauthorize:8443`.

       > **Tip**
       >
       > You can find the HTTPS connection handler port from the **Configuration** page of the PingAuthorize administrative console by going to **System > Connection Handlers**.

    6. Edit the value of `shared_secret` to match the shared secret that you created in PingAuthorize.
    7. Click **Save**.

       Your flow configuration should look like this:

Copyright © 2025 Ping Identity Corporation



3. In Apigee Edge or Apigee Private Cloud, configure the connection to PingAuthorize.

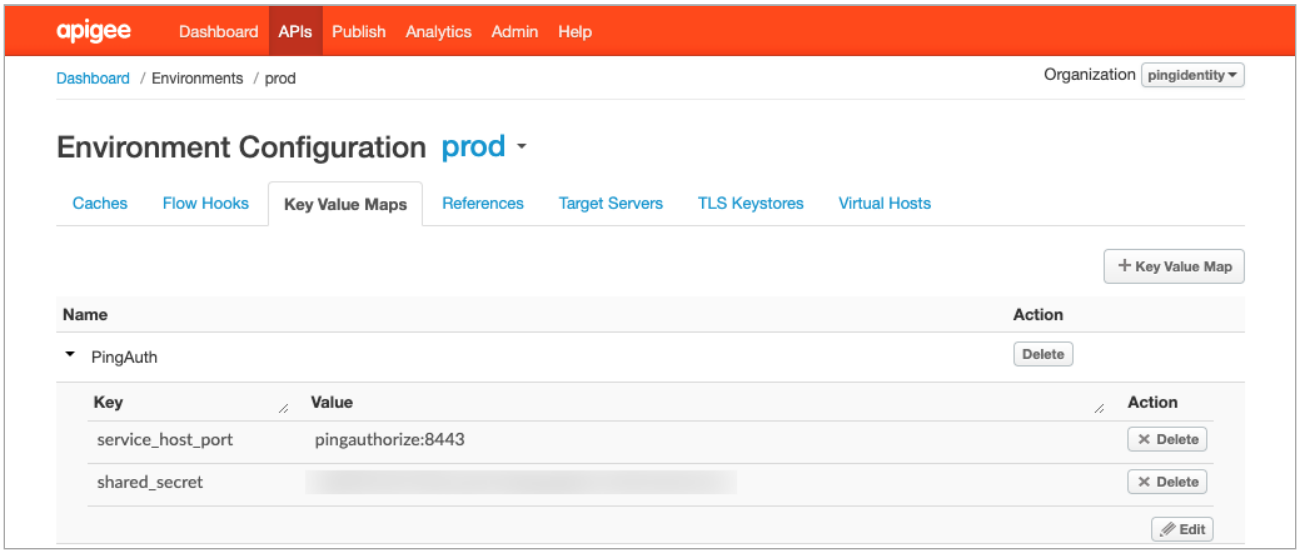
   Apigee Edge stores environment-specific configuration values in key-value maps so that the same policies can be used across multiple deployment environments without any changes to the policies.

   1. Go to **Environment > Key Value Maps** and click **+Key Value Map**.
   2. Edit the key-value map and click **Add Entry**.
   3. Add a `service_host_port` key and set the value to the host name of your PingAuthorize Server instance and the port of the HTTPS connection handler.

      For example, `pingauthorize:8443`.
   4. Add a `shared_secret` key and set the value to the shared secret that you created in PingAuthorize.
   5. Click **Save**.

      Your key-value map configuration should look like this:

Copyright © 2025 Ping Identity Corporation

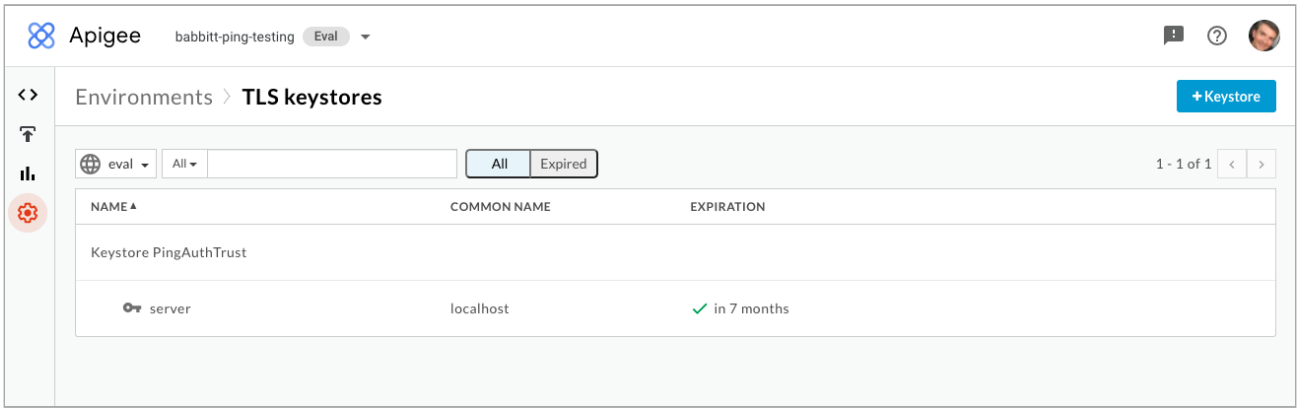


4. (Optional) Configure HTTPS trust for PingAuthorize.

   By default, the PingAuth shared flow only trusts the PingAuthorize HTTPS connection handler certificate if the certificate is issued from a well-known certificate authority. To enable Apigee to trust specific HTTPS certificates from PingAuthorize Server:

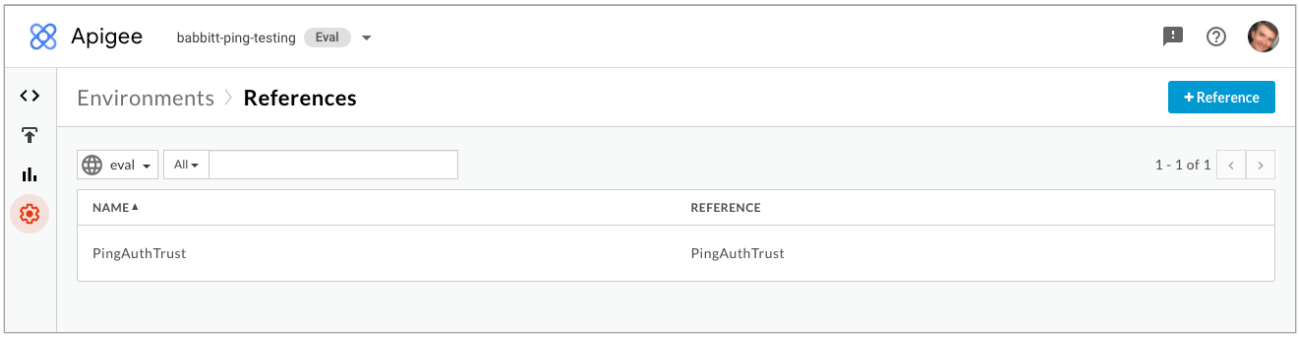
   1. Go to **Environment** > **TLS Keystores** and click **+Keystore**.
   2. Give the key store a name that helps you identify your PingAuthorize environment.

      For example, `PingAuthorize-dev-truststore`.
   3. Click the **+** button to add a certificate.
   4. Enter a certificate alias and upload the certificate configured for the HTTPS connection handler in PingAuthorize.

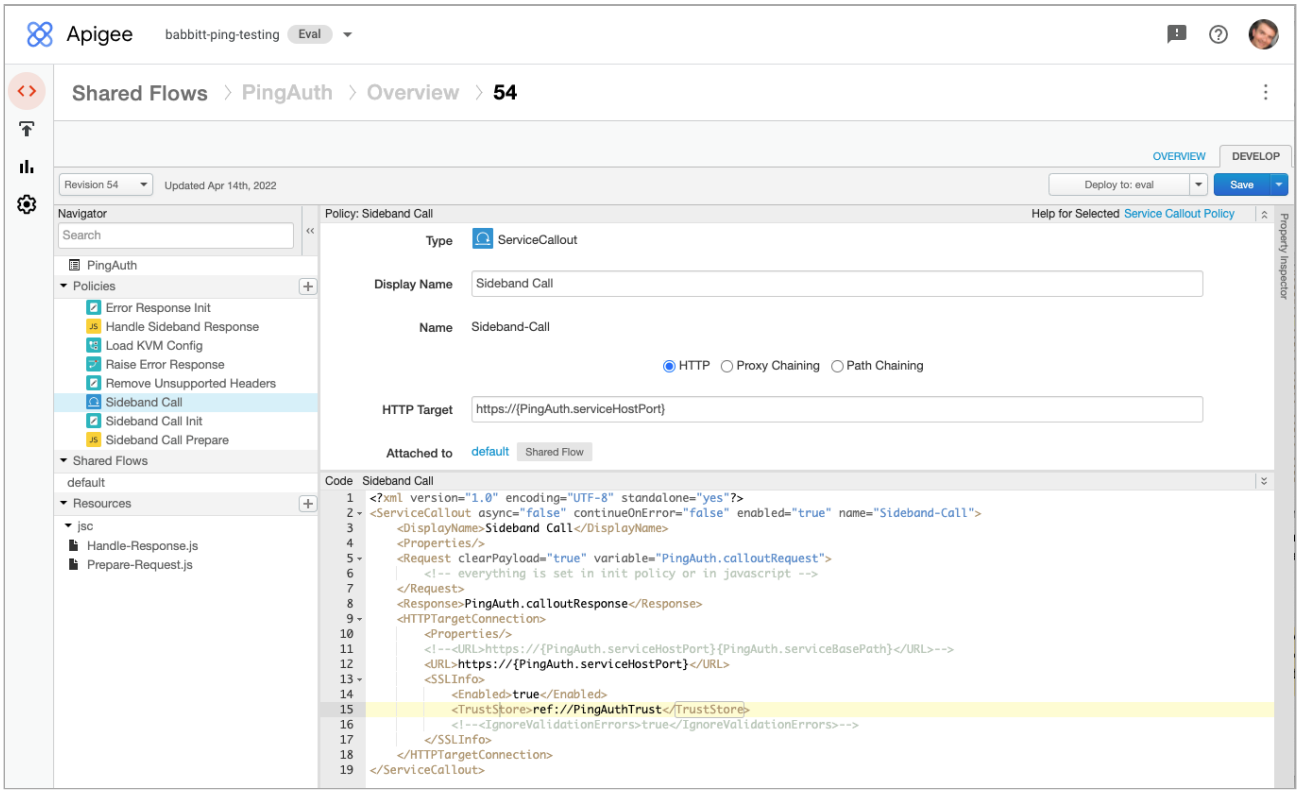


   5. Click **Save**.
   6. Go to **Environment** > **References** and click **+Reference**.
   7. Name the new reference `PingAuthTrust`.
   8. Select the key store that you created previously and click **Save**.

Copyright © 2025 Ping Identity Corporation



9. Go to **Develop > Shared Flows > PingAuth**.
10. On the **Develop** tab, examine **Revisions** to make sure you're on the latest revision.
11. In the **Policies** panel on the left, click the **Sideband Call** policy.
12. In the **Code** panel, remove the comment characters surrounding the `TrustStore` element.



13. Click **Save**.

5. Go to **Develop > Shared Flows > PingAuth** and deploy the most recent revision to your environment.

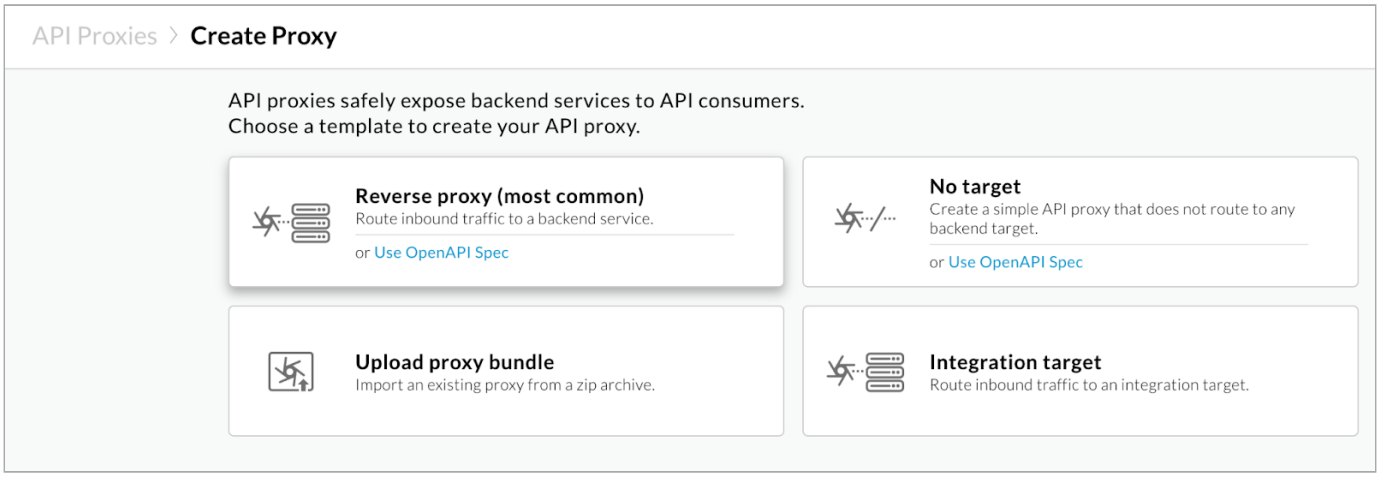
Adding an API proxy in Apigee

Configure the API proxy in Apigee to point to the target endpoint that you want to reach.

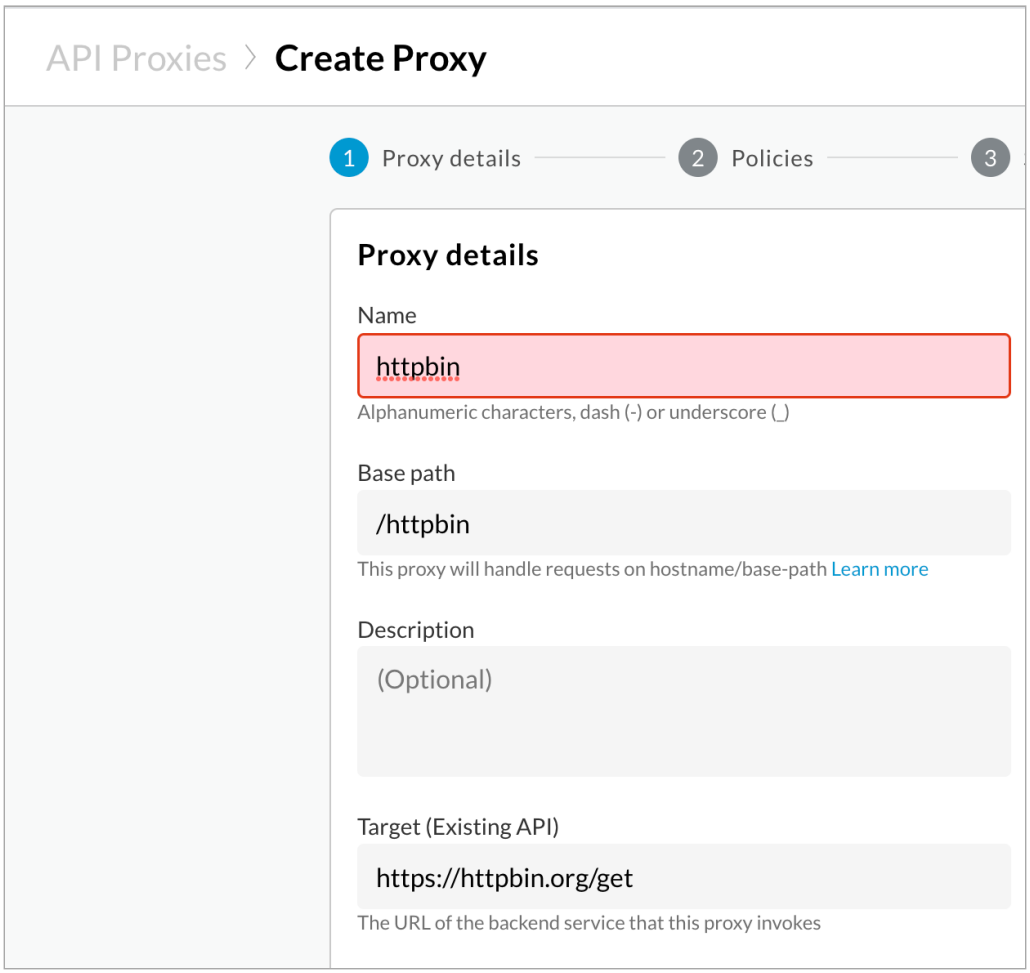
Steps

1. Go to **API Proxies > Create Proxy** and click the **Reverse proxy** tile.

Copyright © 2025 Ping Identity Corporation

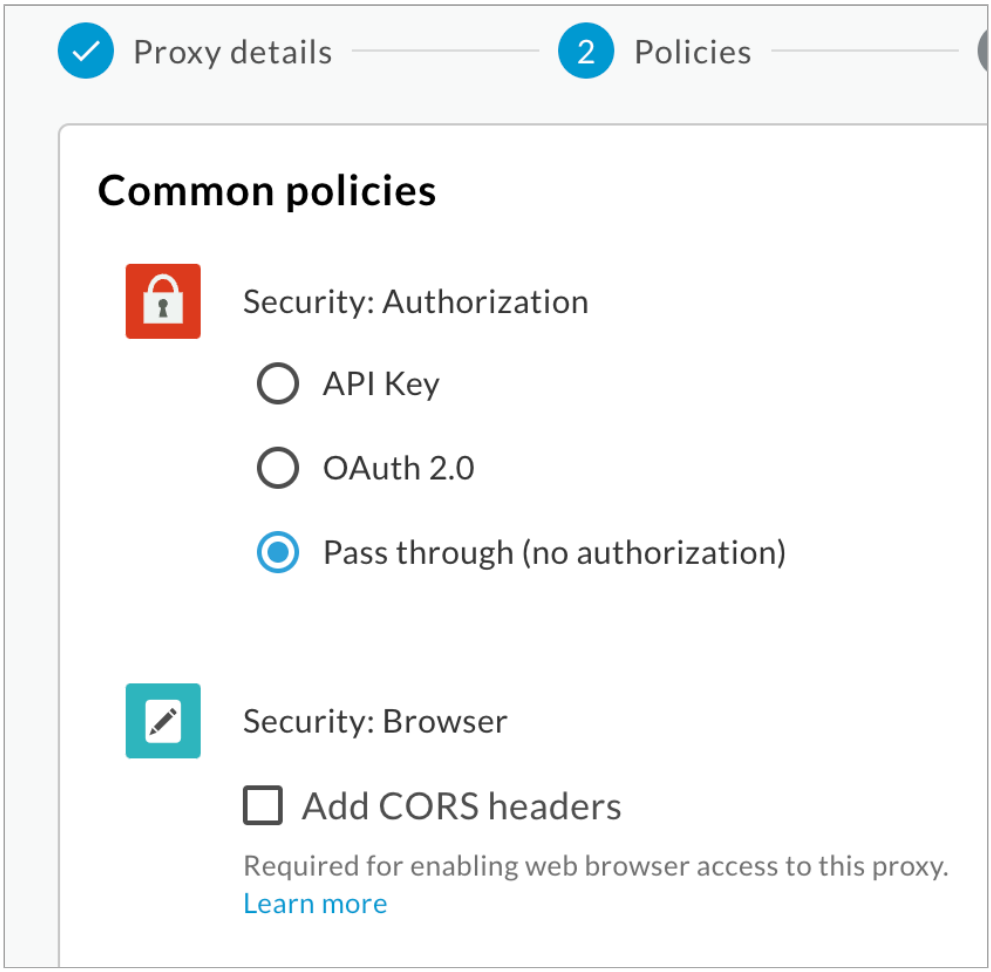


2. On the **Proxy details** page, enter the **Name**, **Base path**, and **Target (Existing API)**.

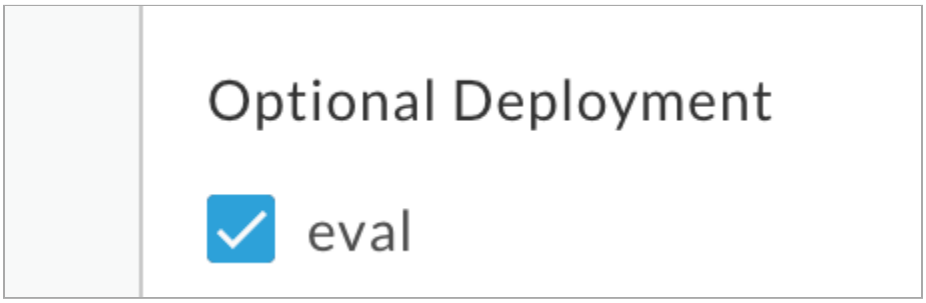


3. On the **Common policies** page, click **Pass through (no authorization)**.

Copyright © 2025 Ping Identity Corporation



4. Select the checkbox for your deployment environment.



5. Click **Create and Deploy**.

Attaching the PingAuthorize shared flow to API proxies

Attach the PingAuth shared flow to the API proxies where you want to use PingAuthorize as the external authorization policy runtime service.

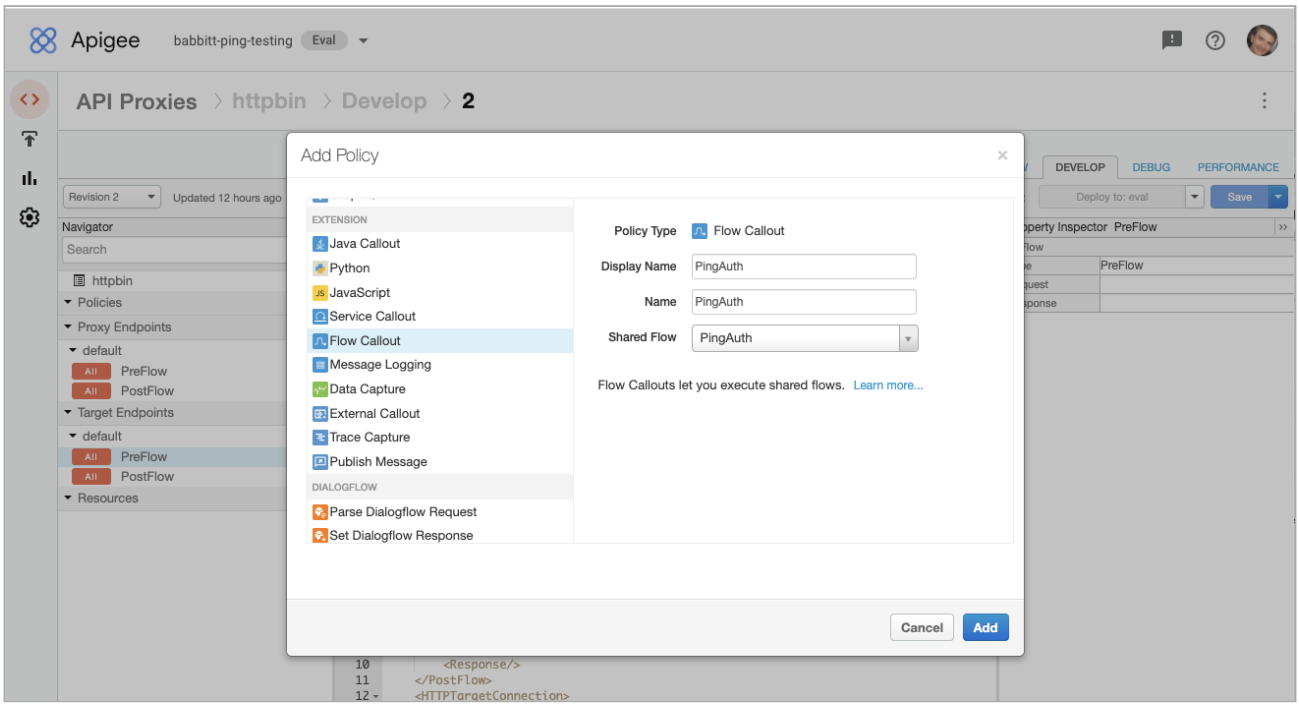
Steps

1. Add a Flow Callout policy:
    1. Go to one of your APIs in **Develop > API Proxies** and click the **Develop** tab.

Copyright © 2025 Ping Identity Corporation

> **Important**
>
> Ensure you are on the latest revision of the proxy.

2. In the **Policies** panel on the left, click the **+** icon.
3. In the **Add Policy** modal, in the **Extension** panel on the left, click **Flow Callout**.
4. Enter a **Name** for the policy.



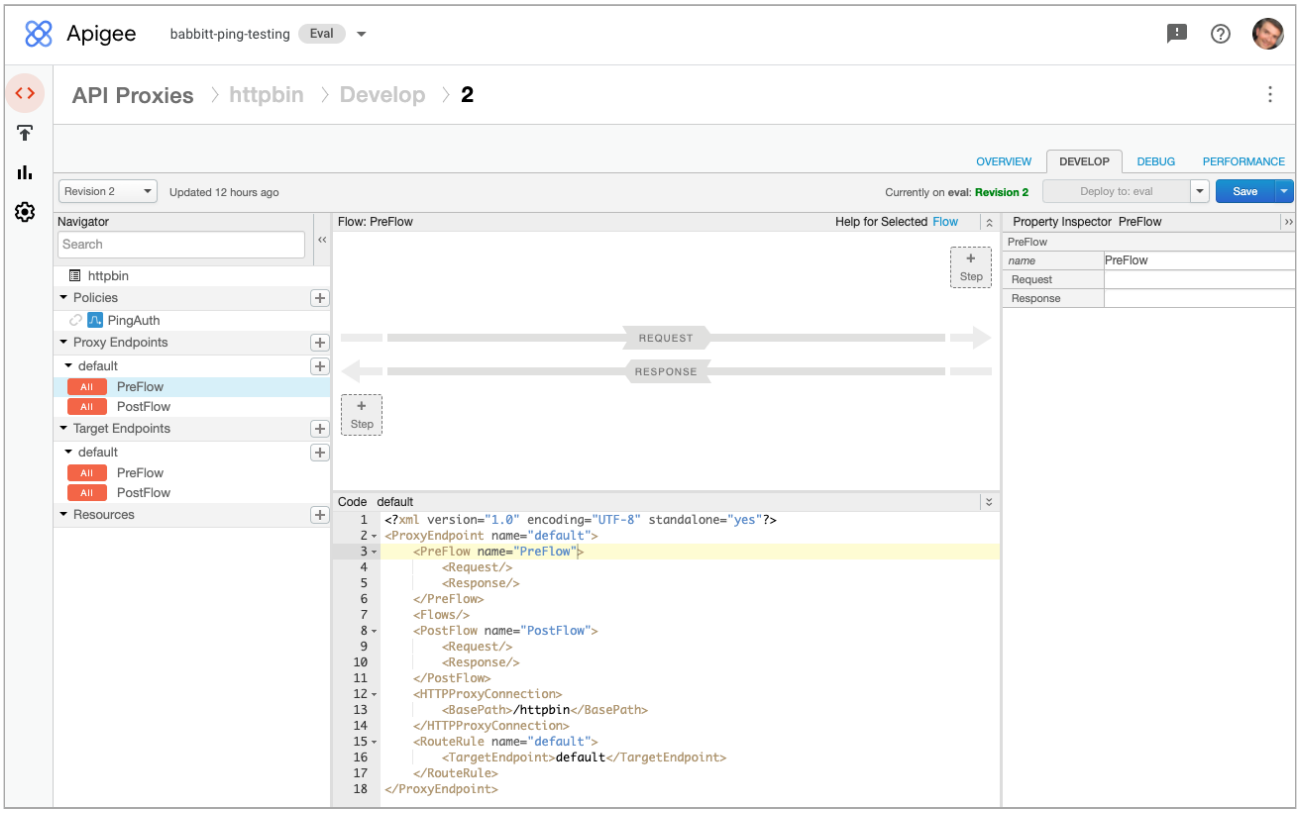
5. In the **Shared Flow** list, select **PingAuth**, and then click **Add**.

2. Attach the Flow Callout policy to flows.

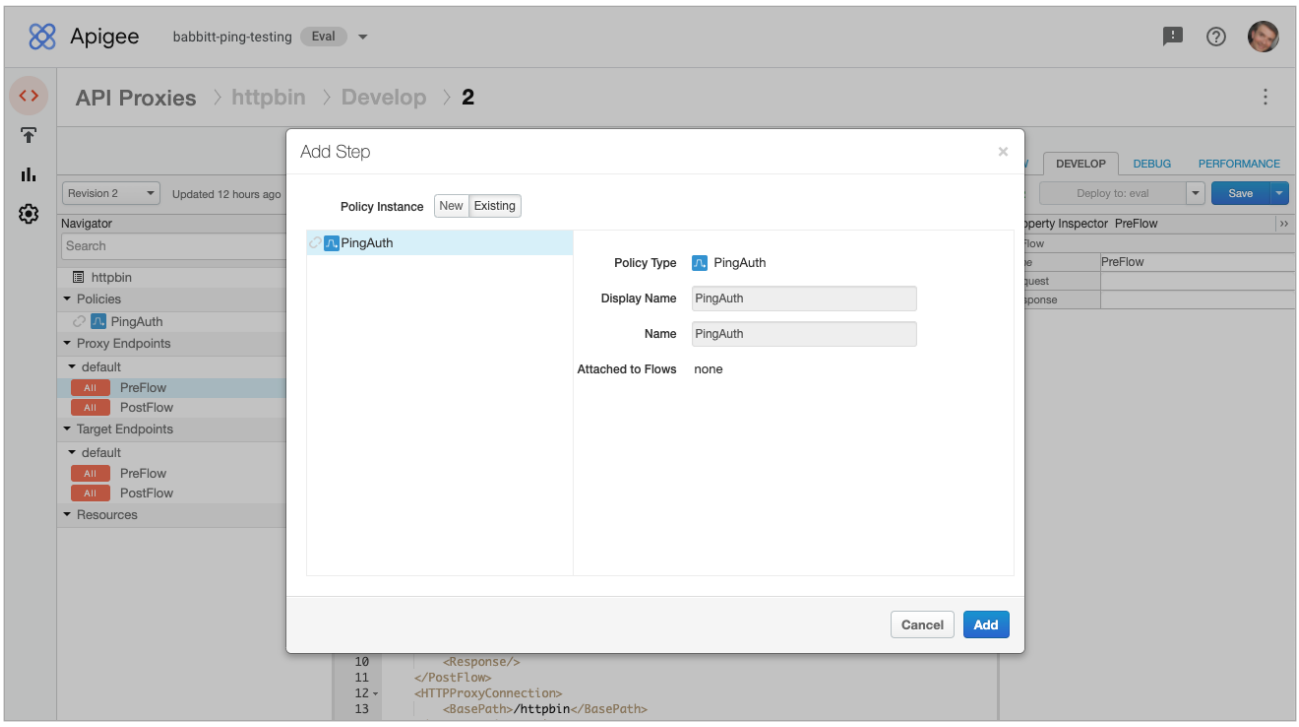
> **Tip**
>
> Because PingAuthorize provides fine-grained authorization, you should integrate PingAuthorize late in the PreFlow of the request to the proxy endpoint, after coarse-grained authentication and authorization functions. Learn more about other ways to integrate PingAuthorize in Controlling API proxies with flows in the Apigee documentation.

1. In the **Proxy Endpoints** panel on the left, click **PreFlow**.
2. In the **Request** section, click **+Step** to add a flow step to the request.

Copyright © 2025 Ping Identity Corporation



3. In the **Add Step** modal, click the **Existing** tab, and then select the Flow Callout policy you created previously.



4. Click **Add**.

5. In the **Target Endpoints** panel on the left, select **PreFlow**.

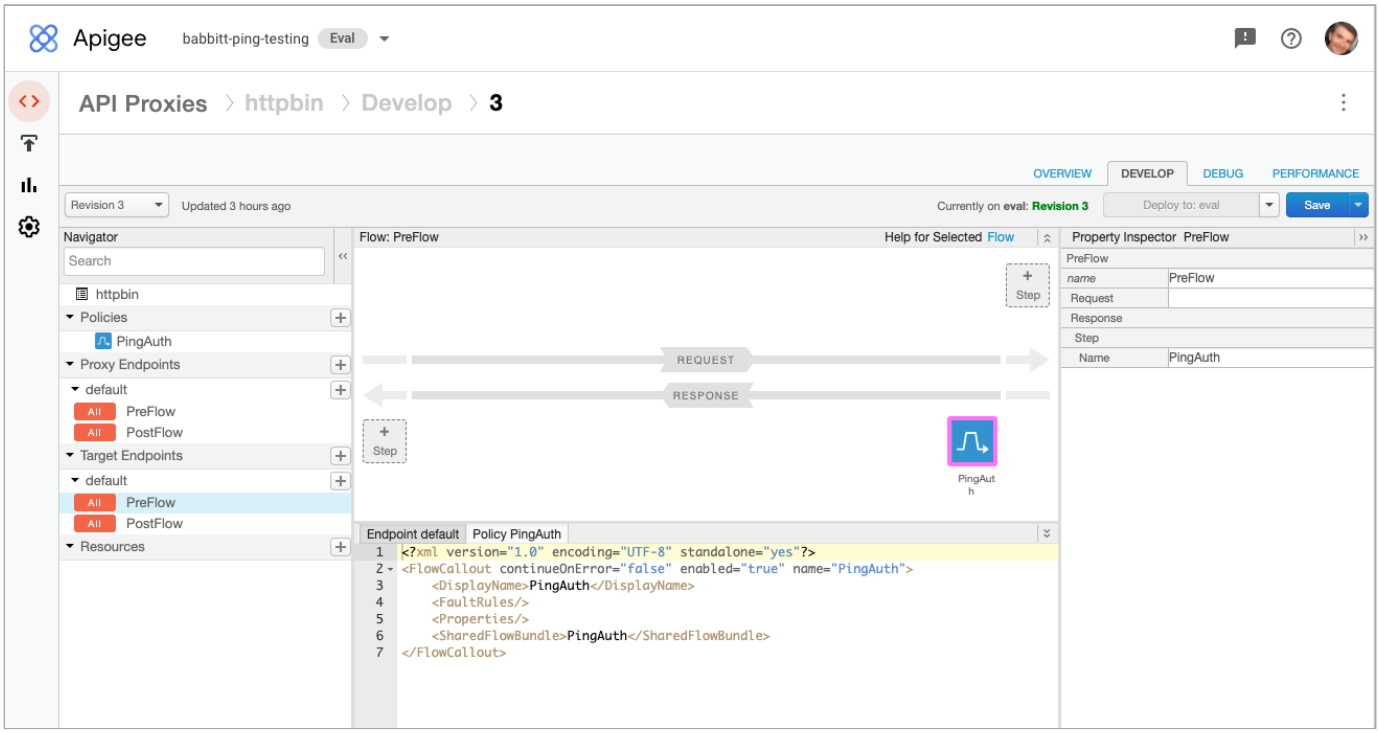
6. In the **Response** section, click **+Step** to add a flow step to the response.

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> This allows PingAuthorize to process the API response from the target API before it's processed by Apigee.

7. In the **Add Step** modal, click the **Existing** tab, and then select the Flow Callout policy you created previously.

3. Save and deploy the updated proxy.



Next steps

Configure fine-grained authorization policies in the PingAuthorize Policy Editor. You can find information on how to target specific API requests and extract other HTTP metadata to use in your policies in Sideband API policy requests.

Troubleshooting the Apigee integration

Troubleshoot the most common issues with the Apigee gateway integration with PingAuthorize.

Troubleshooting API client HTTP 5xx errors

Apigee might return `HTTP 502` when there is misconfiguration or miscommunication between the PingAuth shared flow for Apigee and PingAuthorize Server.

To address 5xx errors, make adjustments to the **Load KVM Config** policy assigned to **PingAuth** in Apigee X or the key value map that you created for the PingAuth Shared Flow in Apigee Edge or Apigee Private Cloud.

The PingAuth Shared Flow for Apigee logs warning messages to the Apigee error log when it encounters problems communicating with PingAuthorize. Learn more in Enable logging in Apigee.

Copyright © 2025 Ping Identity Corporation

Steps

- Check the PingAuth Shared Flow `service_host_port` value.

  If the Apigee `service_host_port` value does not match your PingAuthorize server environment, the Apigee error log message might indicate that the plugin received an invalid response from the server.

  1. Confirm that the value entered for `service_host_port` matches the host name of your PingAuthorize server and the port of the HTTPS connection handler.

     You can find this port number on the **Configuration** page of the PingAuthorize administration console by going to **System → Connection Handlers**.

  2. If necessary, update the `service_host_port` value to match your PingAuthorize server.

- Check the PingAuth Shared Flow shared secret.

  If the shared secret doesn't match the API gateway credential in PingAuthorize, the Apigee error log message might indicate that the plugin received an `HTTP 401` error from PingAuthorize, which gets translated to a 5xx error and then sent to the API client.

  1. Confirm that the value of the **shared_secret** key that you created in Apigee matches the shared secret value that you created for PingAuthorize.

  2. If necessary, on the **Configuration** page of the PingAuthorize administration console, go to **Web Services and Applications > HTTP Servlet Extensions > Sideband API** and update the value of the shared secret.

  3. Copy the new value of the shared secret and update the value of the Apigee **shared_secret** key.

Troubleshooting API client HTTP 4xx errors

The Apigee API gateway might return a 4xx error to the API client if the API client's request can't be authenticated by the PingAuthorize sideband API endpoint.

To troubleshoot 4xx errors caused by authentication issues against the PingAuthorize sideband API, refer to Authenticating to the Sideband API.

Setting up error response handling in the target server

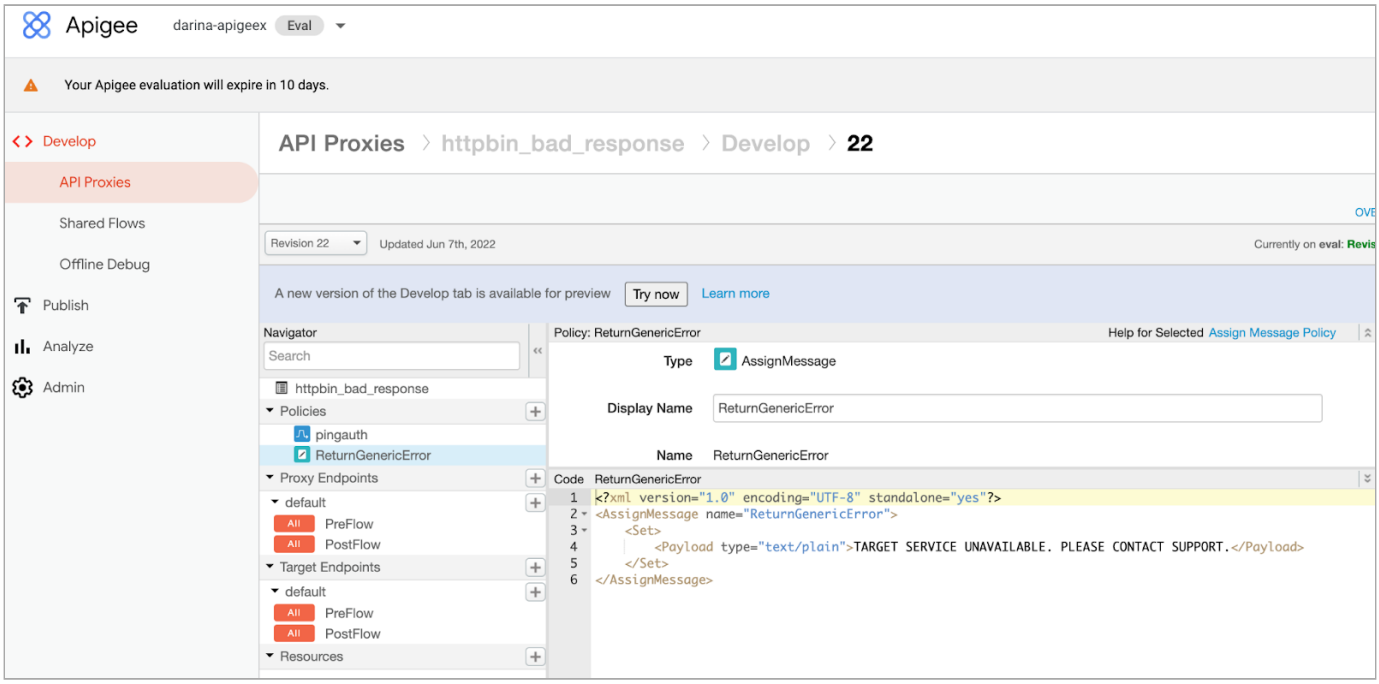
You should have an Apigee policy that handles errors returned by the target server.

If you don't configure error handling using a policy, the API proxy goes into an error state in the `<TargetEndpoint>` response, and the normal API proxy flow won't continue to the `<ProxyEndpoint>`.

Steps

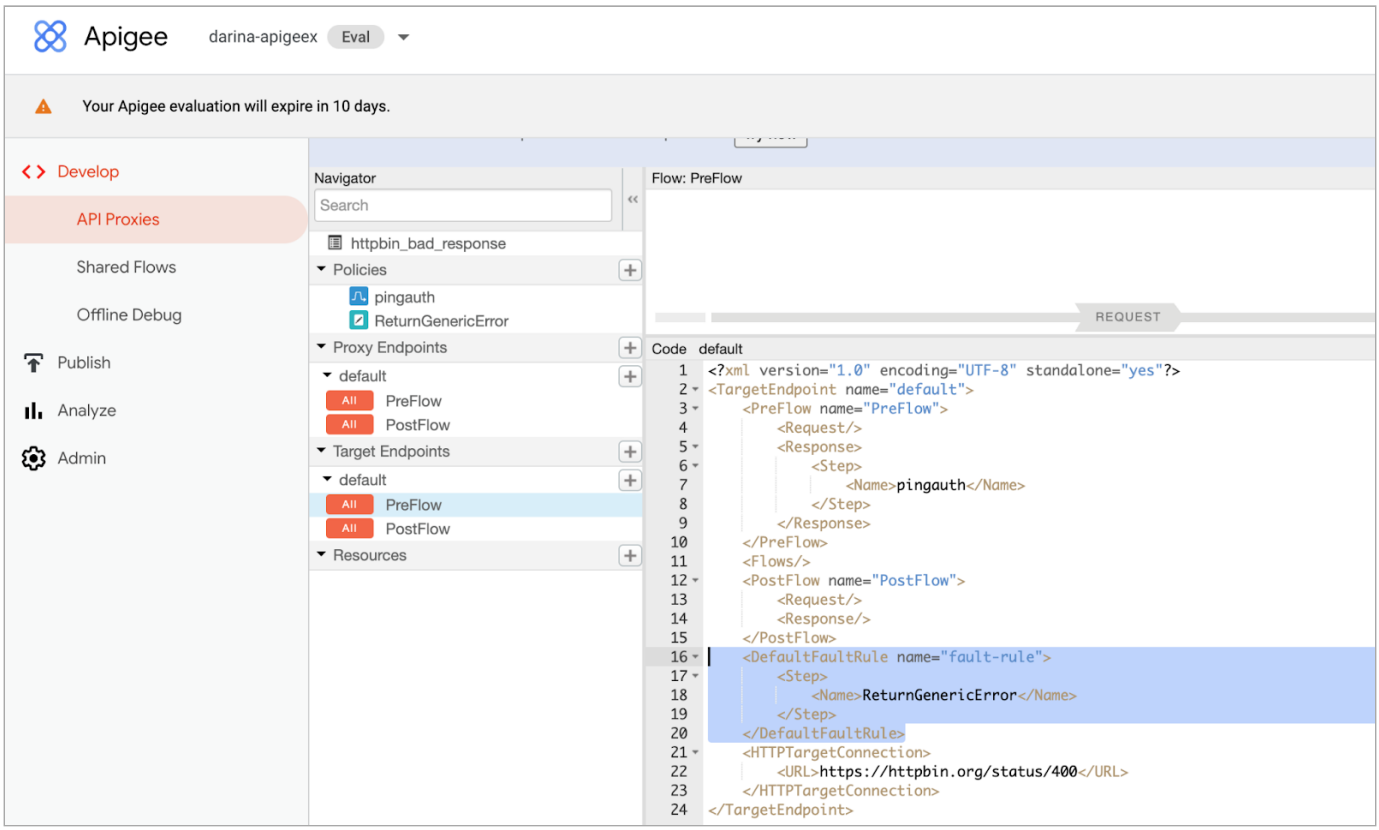
1. Go to **API Proxies > httpbin_bad_response > Develop** and create a new **ReturnGenericError** policy of type **AssignMessage**. Configure the policy as desired.

Copyright © 2025 Ping Identity Corporation



2. Select the **PreFlow** option in the **Target Endpoints** for your API proxy. Add the error policy you just created as a `<Step>` in the `<DefaultFaultRule>`.

> **Note**
>
> There are multiple methods for adding the error handling policy.



Copyright © 2025 Ping Identity Corporation

Enable logging in Apigee

To view error log messages, configure Apigee error logging. Learn more in the View message data with Trace in the Apigee documentation.

Apigee also provides debug logging for further troubleshooting. Learn more in Enabling debug logging in the Apigee documentation.

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 application programming interface (API) resources. Instead of configuring policies multiple times, deploy the Kong Gateway integration once and manage your policy rules in PingAuthorize.

The following 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. Learn more in the Kong Gateway documentation.
- Install and start PingAuthorize. Learn more in Installing PingAuthorize.

Steps

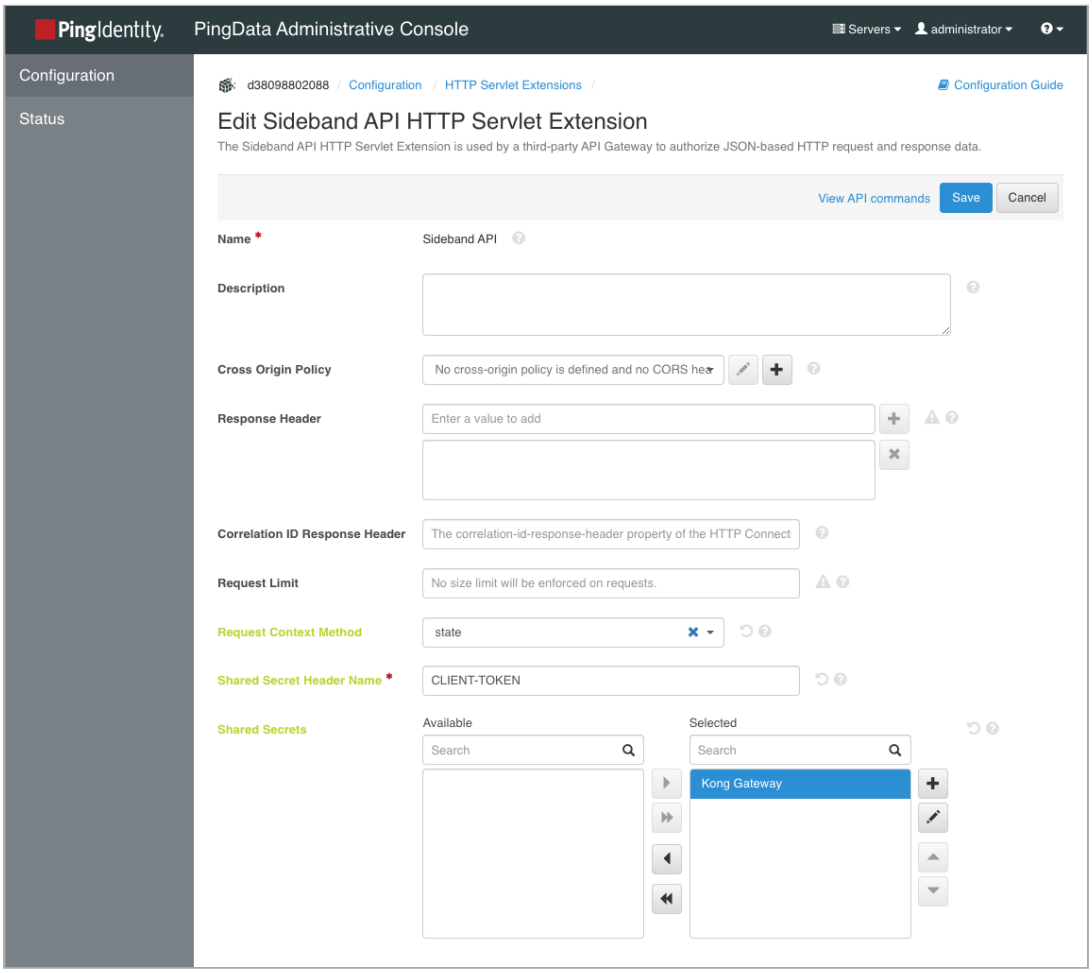
1. In the PingAuthorize administrative console, go to **Configuration** > **HTTP Servlet Extensions** > **Sideband API**.
2. In the **Request Context Method** list, select **State**.

Copyright © 2025 Ping Identity Corporation

3. In the **Shared Secret Header Name** field, enter `CLIENT-TOKEN`.
4. Next to the **Selected** table for **Shared Secrets**, click the + icon to create a new shared secret.

> **Important**
>
> The shared secret authenticates the `ping-auth` plugin to PingAuthorize. Version 1.2.0 of the plugin supports referenceable secrets. For security reasons, store the shared secret in a vault supported by Kong. Learn more in Secrets Management and Environment Variables Vault in the Kong documentation.

5. In the modal dialog, create a suitably long shared secret value, and then 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.

Steps

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

   Learn more in the Kong Gateway plugin installation guide.

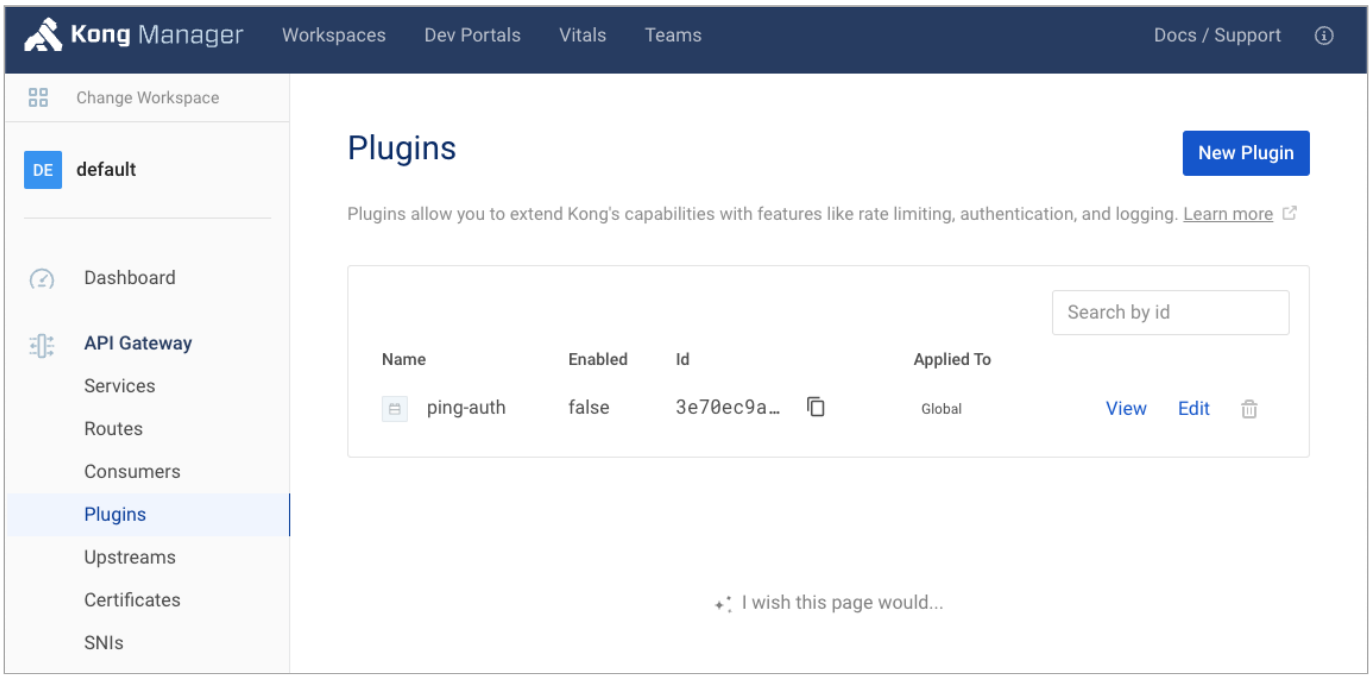
Copyright © 2025 Ping Identity Corporation

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 the plugin loads successfully, look for the debug-level `Loading plugin: ping-auth` message in Kong's `error.log` file.
5. Use the Kong Gateway UI or API to complete the configuration.

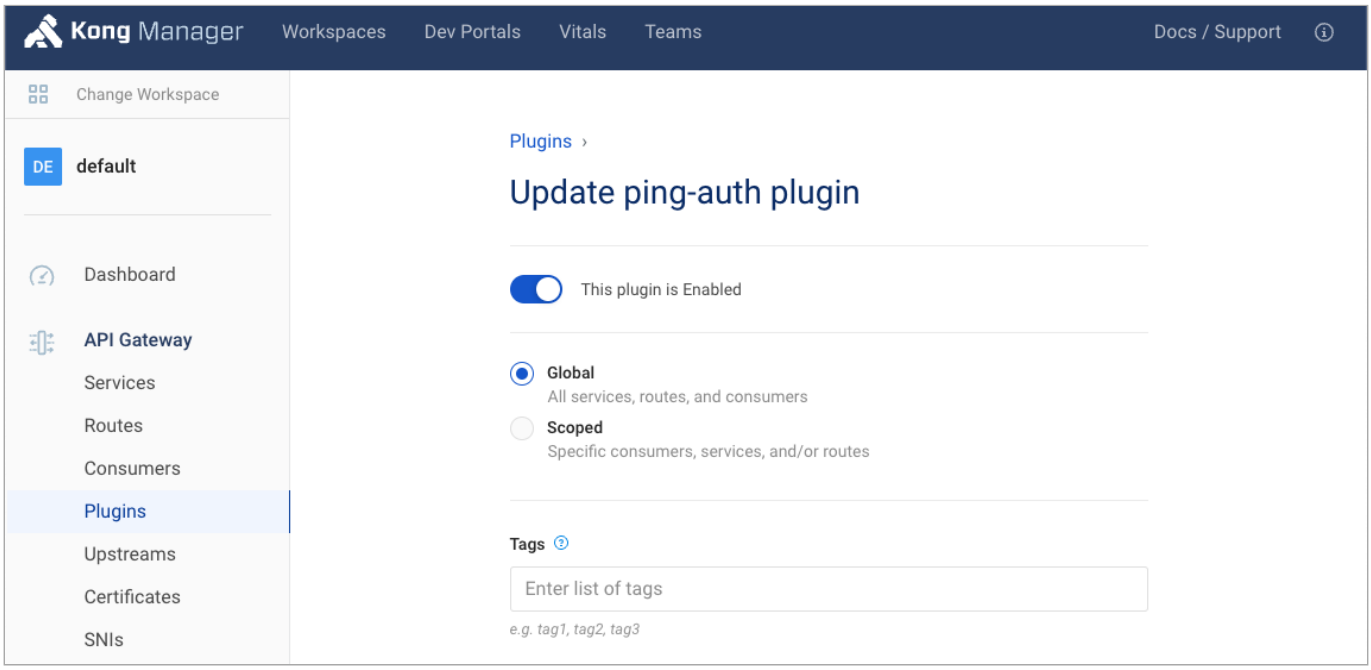
Copyright © 2025 Ping Identity Corporation

Kong Gateway UI

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



- *Next to the **ping-auth** plugin, click **Edit**, and then click the toggle to enable the plugin.*

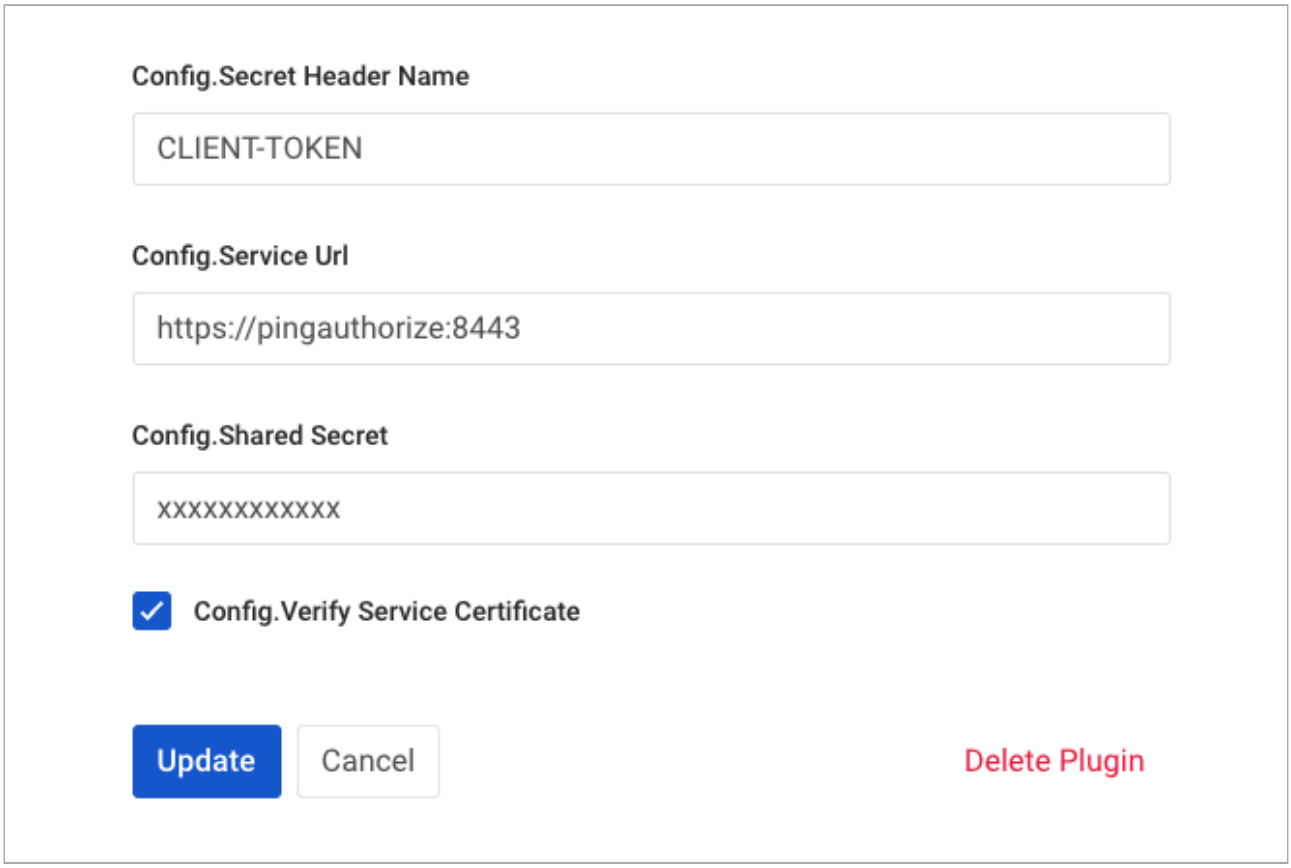


- *(Optional) To enable the plugin for specific consumers, services, or routes, click **Scoped**, and then enter **Service**, **Route**, and **Consumer** information as needed.*

- *Connect Kong Gateway to PingAuthorize.*

    1. *Make sure the **Config.Secret Header Name** value in Kong Manager matches the secret header name configured for the Sideband API Servlet Extension in PingAuthorize.*

Copyright © 2025 Ping Identity Corporation



2. *In the **Config.Service URL** field in Kong Manager, enter the hostname of your PingAuthorize Server instance and the port of the HTTPS Connection Handler.*

   For example, `https://pingauthorize:8443`.

   > **Tip**
   >
   > *To find the HTTPS Connection Handler port number in the PingAuthorize administrative console, go to **Configuration** > **System** > **Connection Handlers**.*

3. *In the **Config.Shared Secret** field, enter the sideband client's shared secret you created in Preparing PingAuthorize for Kong Gateway integration.*

   > **Important**
   >
   > *The shared secret authenticates the* `ping-auth` *plugin to PingAuthorize. Version 1.2.0 of the plugin supports referenceable secrets. For security reasons, store the shared secret in a vault supported by Kong. Learn more in Secrets Management and Environment Variables Vault in the Kong documentation.*

- *(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`.

Copyright © 2025 Ping Identity Corporation

Option	*API Field Name*	*Description*
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`.
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`.

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

Copyright © 2025 Ping Identity Corporation

Kong Gateway API

- *Include the following JSON object 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>"
  }
}
```

  - `service_url`: *The hostname of your PingAuthorize Server instance and the port of the HTTPS Connection Handler. This URL should not contain `/sideband` in the path.*

    For example, `https://pingauthorize:8443`.

  - `shared_secret`: *The shared secret value you created in the PingAuthorize administrative console.*

    > **Important**
    >
    > *The shared secret authenticates the `ping-auth` plugin to PingAuthorize. Version 1.2.0 of the plugin supports referenceable secrets. For security reasons, store the shared secret in a vault supported by Kong. Learn more in Secrets Management and Environment Variables Vault in the Kong documentation.*

  - `secret_header_name`: *The name of the header in which the shared secret is provided.*

    Learn more in the Kong Gateway Admin API documentation.

- *(Optional) Configure additional options.*

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`.*

Copyright © 2025 Ping Identity Corporation

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`.

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
- API client HTTP 4xx errors
- Enabling error logging in Kong Gateway
- Enabling debug logging for the Kong Gateway plugin

Troubleshooting API client HTTP 5xx errors

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

> **Note**
>
> The plugin for Kong Gateway logs warning messages to the Kong Gateway error log when it encounters problems communicating with PingAuthorize.
> Learn more in Enabling error logging in Kong Gateway.

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:

Copyright © 2025 Ping Identity Corporation

```
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
```

1. If there is a shared secret mismatch, go to **Configuration > Web Services and Applications > Sideband API Shared Secrets** in the PingAuthorize administrative console.
2. Update the shared secret value for PingAuthorize.
3. 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
   ```

   1. 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 administrative console by going to **Configuration → System → Connection Handlers**.
   2. 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.

Learn more in Authenticating to the Sideband API.

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.

Copyright © 2025 Ping Identity Corporation

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.

   Learn more in Logging Reference in the Kong Gateway 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 deploying 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`.

Kong Konnect integration

Ping Identity's integration kit for Kong Konnect extends Kong's authorization capabilities through an external policy evaluation service.

Integration with Kong Konnect allows centralized management of API access control and application protection in PingAuthorize while delegating enforcement to Kong Konnect. For information about how traffic flows through Kong Konnect and PingAuthorize, refer to PingAuthorize Integrations.

Copyright © 2025 Ping Identity Corporation

Configure the integration kit in Kong Konnect to enable management of access control rules in PingAuthorize. The same kit supports both Kong Gateway and Kong Konnect.

> **Note**
>
> Kong Konnect is supported in `ping-auth` plugin version 1.0.8 and later.

To configure the integration kit:

1. Prepare PingAuthorize for Kong Konnect integration.
2. Add the ping-auth plugin to your Kong Konnect control plane.
3. Upload ping-auth plugin files to each data plane node.
4. Configure the ping-auth plugin in Kong Konnect.

Learn more in Adding custom plugins in Konnect in the Kong Konnect documentation.

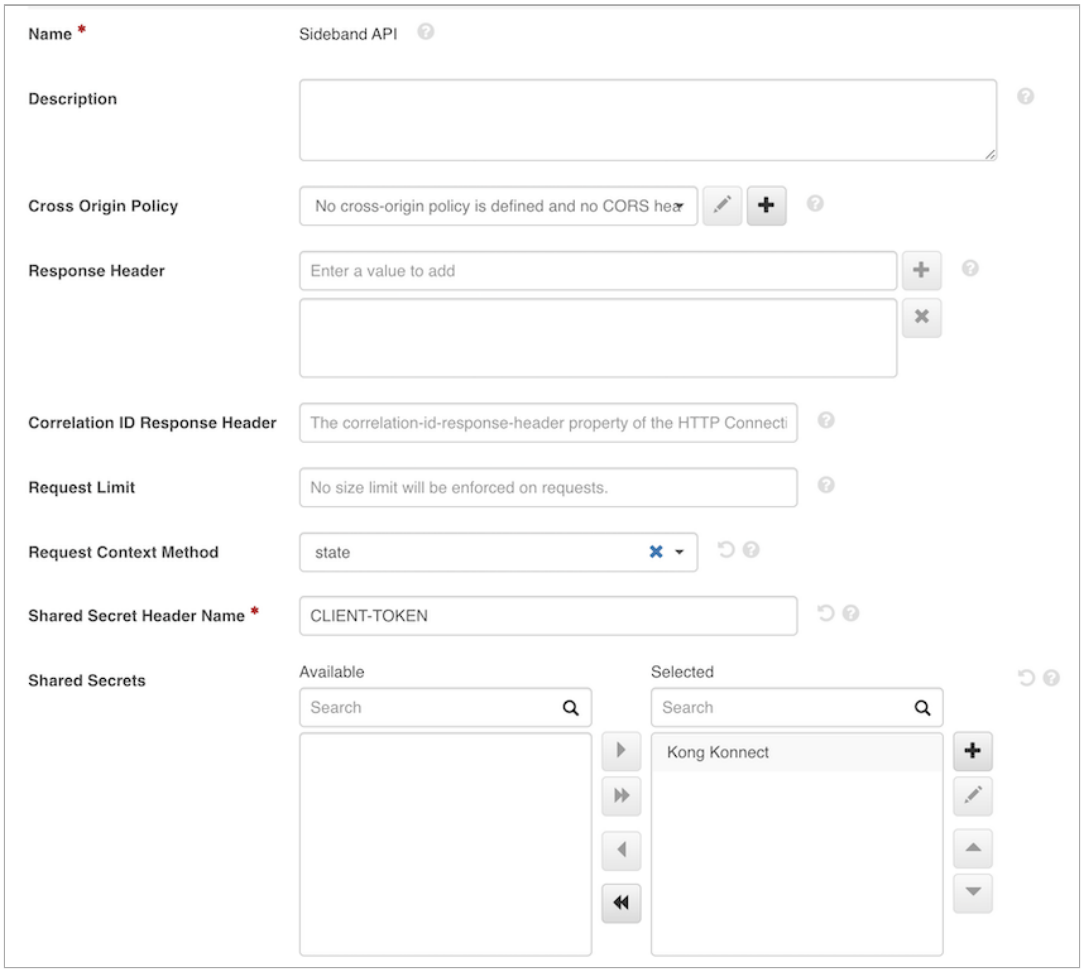
Preparing PingAuthorize for Konnect integration

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

Steps

1. In PingAuthorize, 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 **+** to create a new shared secret.
5. In the modal, 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**.

Copyright © 2025 Ping Identity Corporation



Configuring Konnect for PingAuthorize integration

For Kong Konnect to use PingAuthorize as an external authorization policy runtime service, you must download, install, and configure the `ping-auth` plugin.

To configure the `ping-auth` plugin in Kong Konnect, you must:

1. Add the ping-auth plugin to your control plane
2. Upload files to your data plane nodes
3. Configure the ping-auth plugin in Konnect

Adding the ping-auth plugin to your control plane

Use the `ping-auth` plugin's `schema.lua` file to create the configurable entity required by Konnect. Konnect uses this file to create a plugin entry in the plugin catalog for your control plane. If the plugin schema should be available in multiple control planes, add the schema individually to each one.

You can manage schemas for custom plugins through the Konnect UI or the Konnect Control Planes Config API.

Copyright © 2025 Ping Identity Corporation

Using the Konnect UI

Adding the plugin using the Konnect UI
About this task

> **Note**
>
> The UI is not available when using KIC in Konnect. Use the Konnect API instead.

Steps

1. Download version 1.0.8-1 or later of the `kong-plugin-ping-auth` src from https://luarocks.org/modules/pingidentity/kong-plugin-ping-auth.

   > **Note**
   >
   > Support for Kong Konnect is available in version 1.0.8 and later of the plugin.

2. Extract the `src.rock` file and store it on your local computer.

   Verify that you have the following files in the extracted `/kong-plugin/ping-auth` folder:

   - `access.lua`
   - `handler.lua`
   - `Network_handle.lua`
   - `response.lua`
   - `schema.lua`

3. In **Gateway Manager**, open a control plane.
4. In the sidebar, open **Plugins**, then click **New Plugin**.
5. Click the **Custom Plugins** tab, then click **Create** on the **Custom Plugin** tile.
6. Click **Select File**, then select the `schema.lua` file for the `ping-auth` plugin that you extracted in step 1.
7. Make sure your file displays correctly in the preview, then click **Save**.

Copyright © 2025 Ping Identity Corporation

Using the Konnect API

Adding the plugin using the Konnect API
About this task

When using the `/plugin-schemas` API, authenticate your requests with either a personal access token or a system account token by including it in the authorization header:

```
--header 'Authorization: Bearer kpat_xgfT'
```

Steps

1. Upload the `schema.lua` file for your plugin using the `/plugin-schemas` endpoint:

   ```
   curl -i -X POST \
   https://{region}.api.konghq.com/konnect-api/api/runtime_groups/{controlPlaneId}/v2/plugin-schemas \
   --header 'Content-Type: application/json' \
   --data "{\"lua_schema\":  <your escaped Lua schema>}"
   ```

   Result:

   You should get an `HTTP 201` response.

2. To check that your schema was uploaded, use the following request:

   ```
   curl -i -X GET \
   https://{region}.api.konghq.com/konnect-api/api/runtime_groups/{controlPlaneId}/v1/available-plugins
   ```

   Result:

   This request returns an `HTTP 200` response with the schema for your plugin as a JSON object.

Uploading files to data plane nodes

About this task

After you upload the `ping-auth` plugin's schema to Konnect, upload the following files for the plugin to each Kong Konnect data plane node:

- `access.lua`
- `handler.lua`
- `Network_handle.lua`
- `response.lua`
- `schema.lua`

Copyright © 2025 Ping Identity Corporation

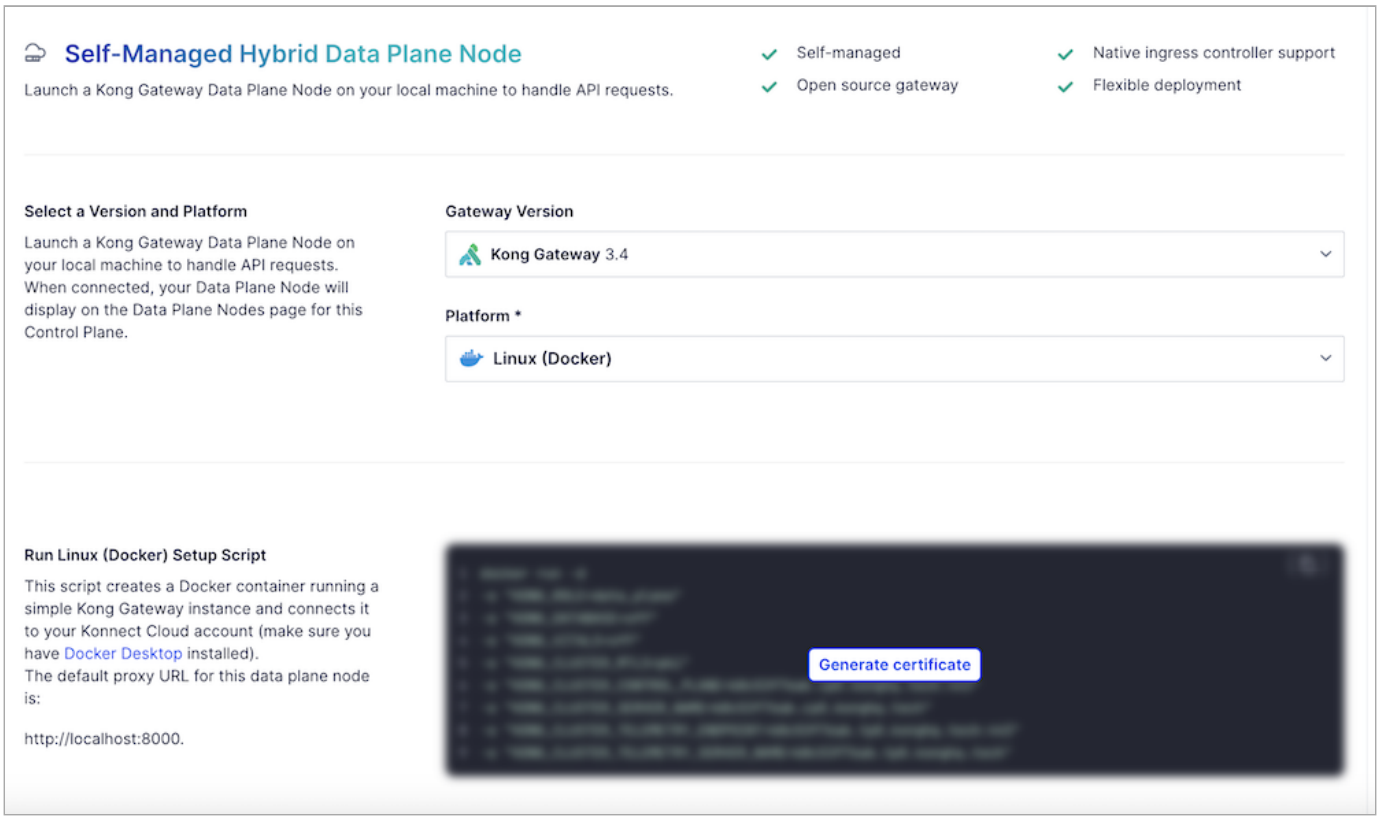
> **Important**
>
> If a data plane node doesn't have these files, the `ping-auth` plugin cannot run on that node.

Follow the Kong Gateway plugin deployment instructions to set up the plugin on each node. Instructions can vary depending on the platform. If you're running Kong Gateway on Docker, the following instructions are provided as an example.

Install the `ping-auth` plugin inside the Kong Konnect Docker container for each node. Copy or mount the plugin's source code into the container.

Steps

1. In your control plane, go to **Data Plane Nodes**, then click **New Data Plane Node**.
2. Select a **Platform**, for example **Linux (Docker)**, and **Generate a certificate**.



3. Copy the generated Docker run command and add the following snippet to it.

   Substitute your own source and target paths.

   - The <*source_path*> is the location where you extracted the `ping-auth` plugin files. This is the parent folder that contains the `ping-auth` folder.
   - The <*target_path*> is where you keep custom Kong plugins. This is the path to the `ping-auth` plugin.

```
-v "/<source_path>:/ping-auth:/<target_path>/ping-auth" \
-e "KONG_PLUGINS=bundled,ping-auth" \
-e "KONG_LUA_PACKAGE_PATH=/<target_path>/?.lua;;" \
```


Copyright © 2025 Ping Identity Corporation

4. To start a data plane node with the `ping-auth` plugin loaded, run the command.

   Example:

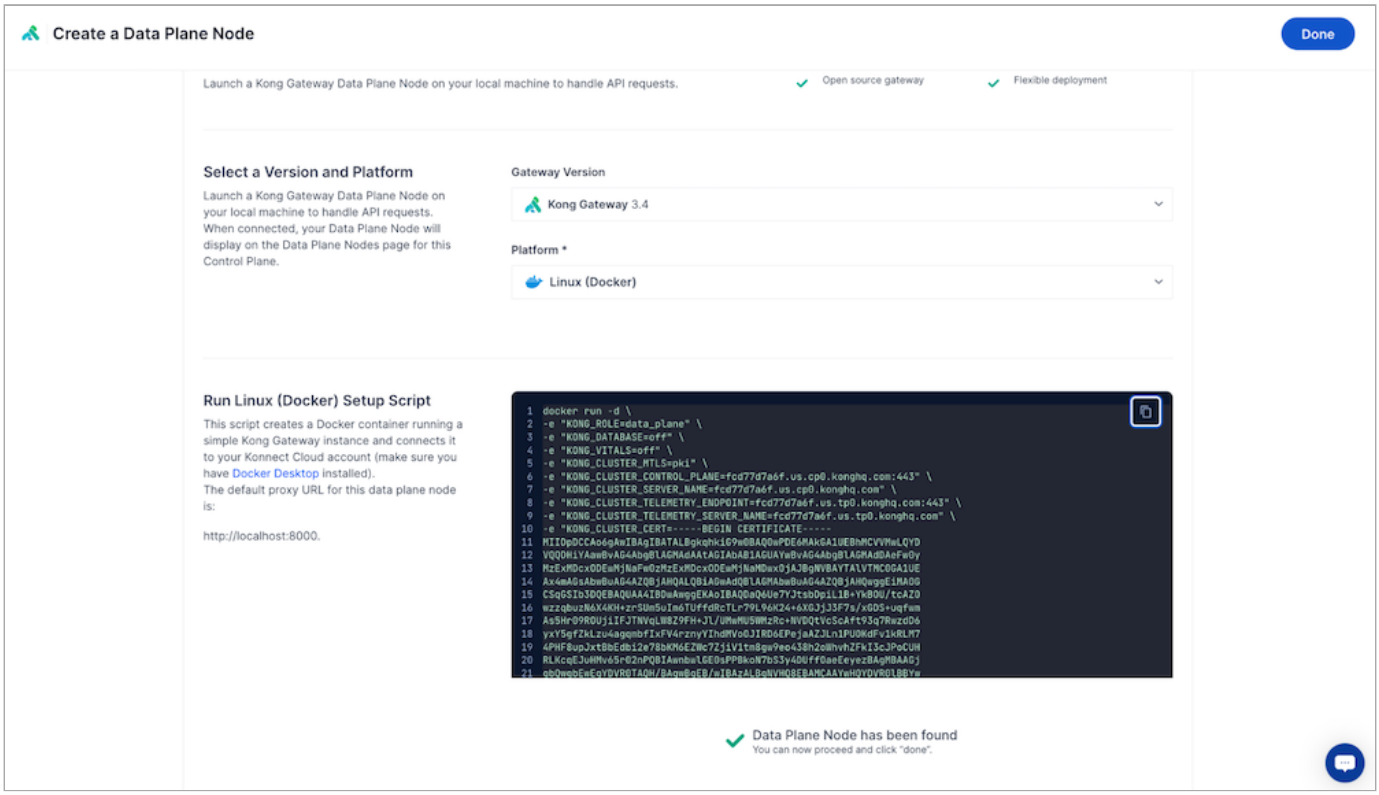
   For example, the command will look something like this, including the three snippet lines from the previous step. In this example, `plugins/kong` represents the *<source_path>* and `/usr/local/share/lua/5.1/kong/plugins` represents the *<target_path>*.

   ```
   docker run -d \
   -v "/plugins/kong/ping-auth:/usr/local/share/lua/5.1/kong/plugins/ping-auth" \
   -e "KONG_PLUGINS=bundled,ping-auth" \
   -e "KONG_LUA_PACKAGE_PATH=/usr/local/share/lua/5.1/kong/plugins/?.lua;;" \
   -e "KONG_ROLE=data_plane" \
   -e "KONG_DATABASE=off" \
   -e "KONG_VITALS=off" \
   -e "KONG_NGINX_WORKER_PROCESSES=1" \
   -e "KONG_CLUSTER_MTLS=pki" \
   -e "KONG_CLUSTER_CONTROL_PLANE=<example>.cp0.konghq.com:443" \
   -e "KONG_CLUSTER_SERVER_NAME=<example>.cp0.konghq.com" \
   -e "KONG_CLUSTER_TELEMETRY_ENDPOINT=<example>.tp0.konghq.com:443" \
   -e "KONG_CLUSTER_TELEMETRY_SERVER_NAME=<example>.tp0.konghq.com" \
   -e "KONG_CLUSTER_CERT=<cert>" \
   -e "KONG_CLUSTER_CERT_KEY=<key>" \
   -e "KONG_LUA_SSL_TRUSTED_CERTIFICATE=system" \
   -e "KONG_KONNECT_MODE=on" \
   -p 8000:8000 \
   -p 8443:8443 \
   kong/kong-gateway:<version>
   ```

5. To confirm the Docker deployment, run the following command:

   ```
   "docker logs [container id]"
   ```

Copyright © 2025 Ping Identity Corporation



Configuring the ping-auth plugin in Konnect

After you've uploaded the `ping-auth` plugin's schema to Konnect, configure the plugin in Gateway Manager or use the Kong API.

> **Important**
>
> Test the operation of the `ping-auth` plugin before you use it in production.

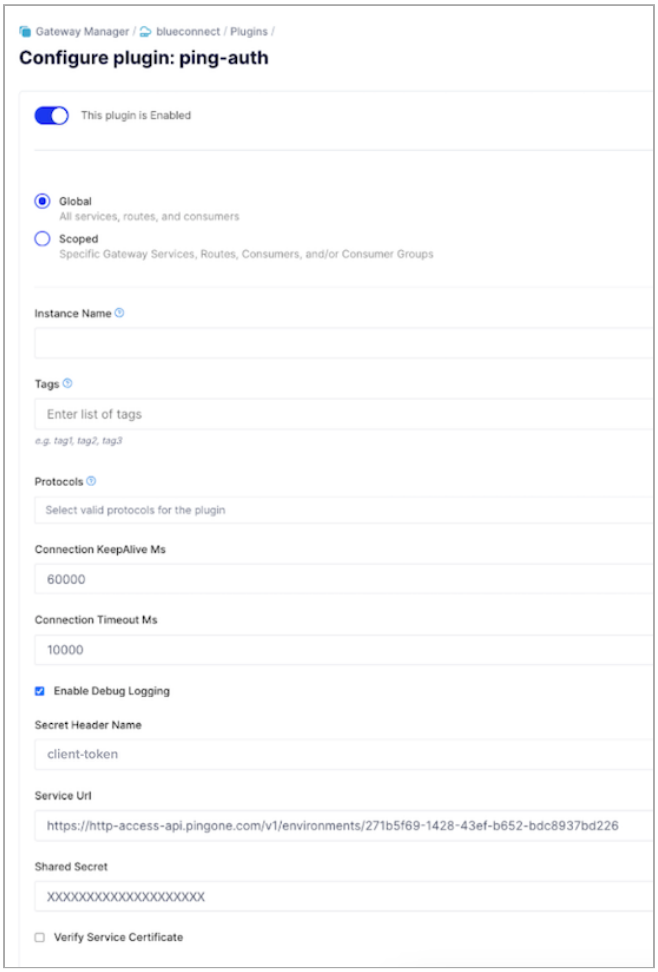
Copyright © 2025 Ping Identity Corporation

Using the Gateway Manager UI

Configuring the plugin using the Gateway Manager UI
Steps

1. In **Gateway Manager**, open **Plugins** from the sidebar, then click **New Plugin**.
2. On the **Custom Plugins** tab, click the **ping-auth** plugin.
3. **Optional:** To enable the plugin for specific consumers, services, or routes, click **Scoped**, and then enter the **Service**, **Route**, and **Consumer** information.
4. In the **Service Url** field, enter the host name of your PingAuthorize server and the port of the **HTTPS Connection Handler**.

   To find this port number in the PingAuthorize administrative console, go to **Configuration → System → Connection Handlers**. For example, `https://pingauthorize:8443`.
5. In the **Shared Secret** field, enter the PingAuthorize sideband client's shared secret.
6. Verify that the **Secret Header Name** matches the secret header name configured for the Sideband API Servlet Extension in PingAuthorize.



7. Configure the following additional fields:

Copyright © 2025 Ping Identity Corporation

Option	API Field Name	Description
Connection KeepAlive Ms	`connection_keepAlive_ms`	The duration to keep the connection alive for reuse. The default is `60000`.
Connection Timeout Ms	`connection_timeout_ms`	The duration to wait before the connection times out. The default is `10000`.
Enable Debug Logging	`enable_debug_logging`	Controls if the 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`.
Verify Service Certificate	`verify_service_certificate`	Controls whether the service certificate is verified. This is intended for testing purposes and the default is `true`.

8. Click **Save**.

Result

Kong Konnect is now configured to work with PingAuthorize.

Copyright © 2025 Ping Identity Corporation

Using the Kong API

Configuring the plugin using the Kong API
Steps

1. Send the following in a `POST` request to `https://{region}.api.konghq.com/konnect-api/api/runtime_groups/{controlPlaneId}/plugins`:

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

   The following list describes the required fields for this 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.

2. Configure the optional fields:

Option	API Field Name	Description
Connection KeepAlive Ms	`connection_keepAlive_ms`	The duration to keep the connection alive for reuse. The default is `60000`.
Connection Timeout Ms	`connection_timeout_ms`	The duration to wait before the connection times out. The default is `10000`.
Enable Debug Logging	`enable_debug_logging`	Controls if the 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`.
Verify Service Certificate	`verify_service_certificate`	Controls whether the service certificate is verified. This is intended for testing purposes and the default is `true`.

Copyright © 2025 Ping Identity Corporation

Result:

Kong Konnect is configured to work with PingAuthorize.

MuleSoft API gateway integration

Learn how to enable fine-grained access control through the MuleSoft application programming interface (API) gateway by deploying the PingAuthorize API adapter and connecting to the sideband API.

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

To integrate PingAuthorize with the MuleSoft gateway, see the following:

1. Deploying the custom MuleSoft policy for PingAuthorize
2. Applying the custom MuleSoft policy for PingAuthorize

Deploying the custom MuleSoft policy for PingAuthorize

Deploy the custom MuleSoft policy to make it available to your integrated APIs.

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. Learn more in Creating a shared secret.

  > **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. Complete the section titled Request context using the state field.

- Install Apache Maven.

Copyright © 2025 Ping Identity Corporation

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 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.

Applying the custom MuleSoft policy for PingAuthorize

You must apply the deployed custom MuleSoft policy to use MuleSoft as an application programming interface (API) gateway with PingAuthorize.

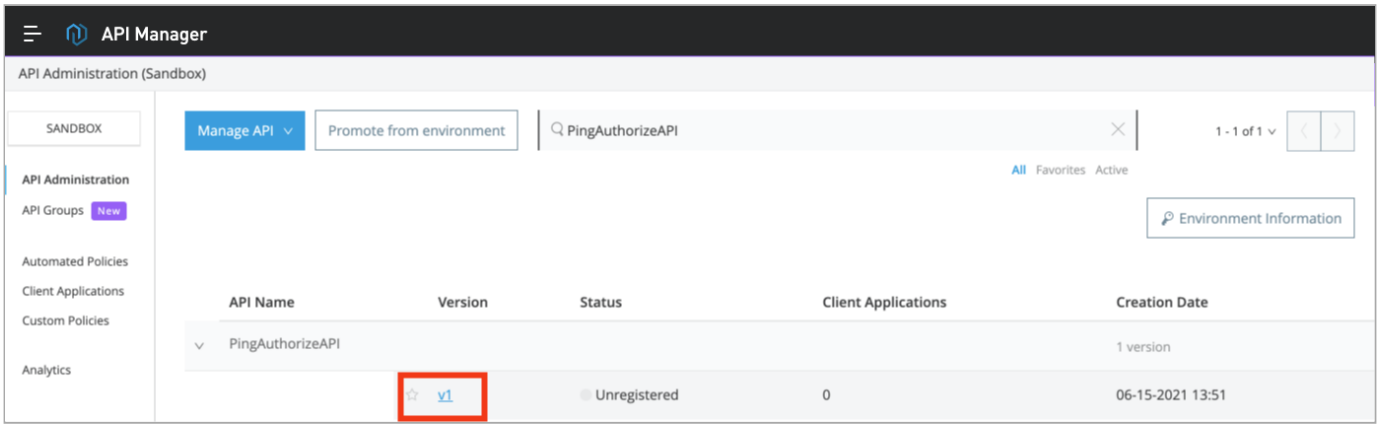
About this task

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

Steps

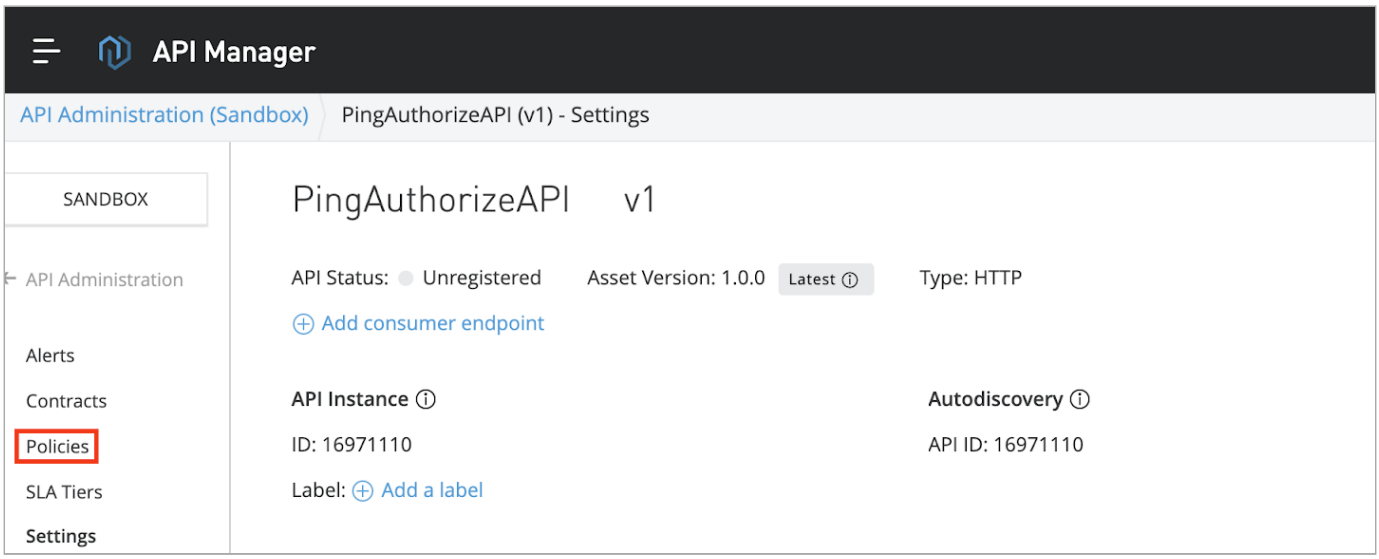
1. Sign on to your MuleSoft Anypoint account.
2. Go to the API manager, expand the API to which you want to attach the PingAuthorize policy, and click **Version**.

Copyright © 2025 Ping Identity Corporation



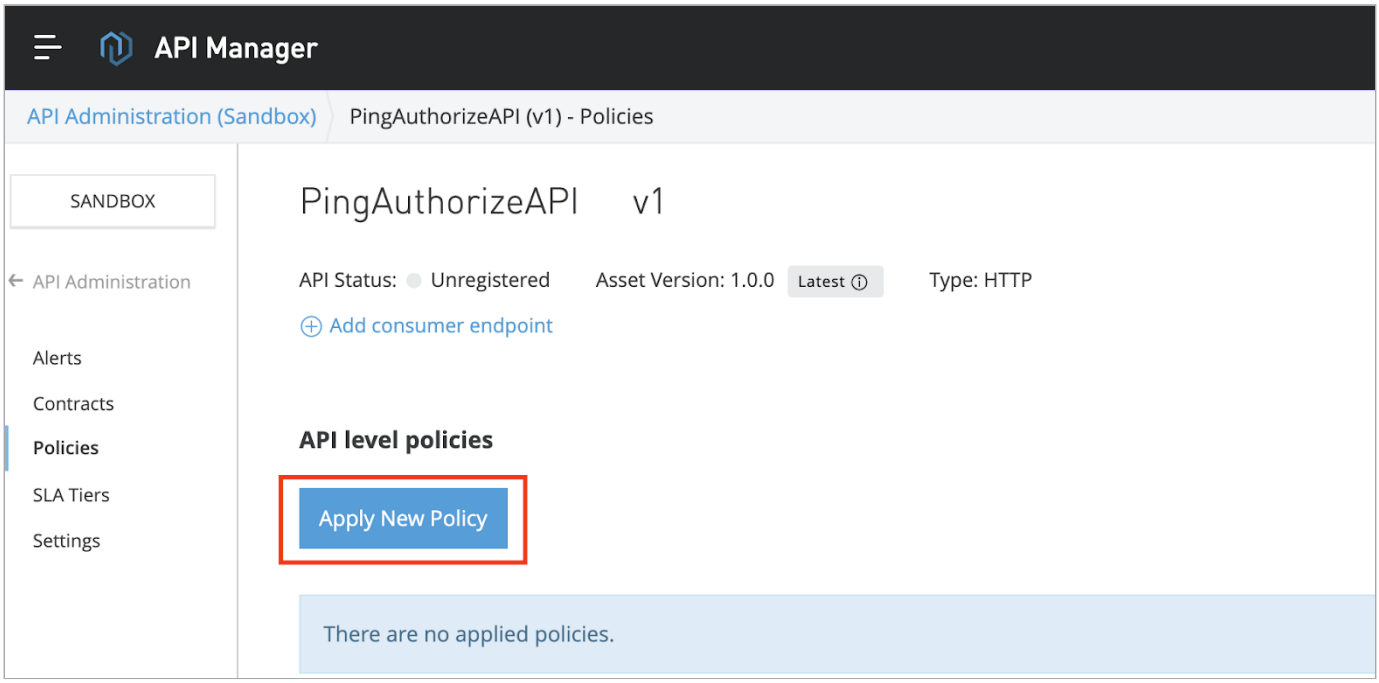
3. In the left navigation pane, click **Policies**.

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



4. Click **Apply New Policy**.

Copyright © 2025 Ping Identity Corporation

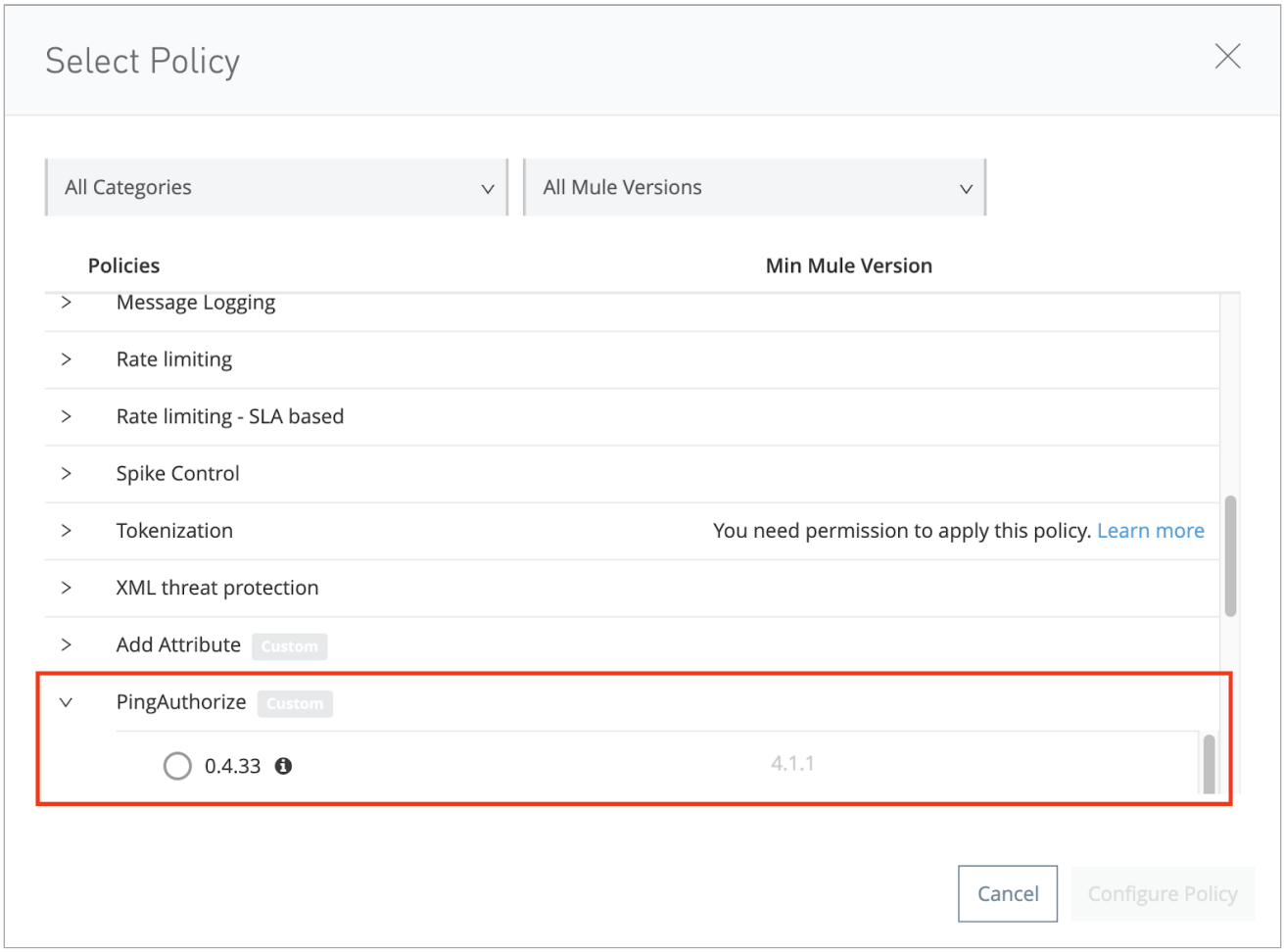


Result:

The **Select Policy** window opens.

5. In the **Select Policy** window, select the PingAuthorize policy and current version. Click **Configure Policy**.

Copyright © 2025 Ping Identity Corporation



6. On the **Apply Policy** page, enter the following values:

    1. 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.
    2. In the **PAZ Host** field, enter the PingAuthorize host and port.

        > **Note**
        >
        > Do not include the connection scheme (http:// or https://).

    3. Select the **Enable SSL** check box for a secure HTTPS connection between MuleSoft and PingAuthorize.
    4. 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.

    5. 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.

Copyright © 2025 Ping Identity Corporation

- 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.

6. **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.
Read Timeout	Governs the time the API Gateway waits for PingAuthorize's response before sending the request to the backend server.

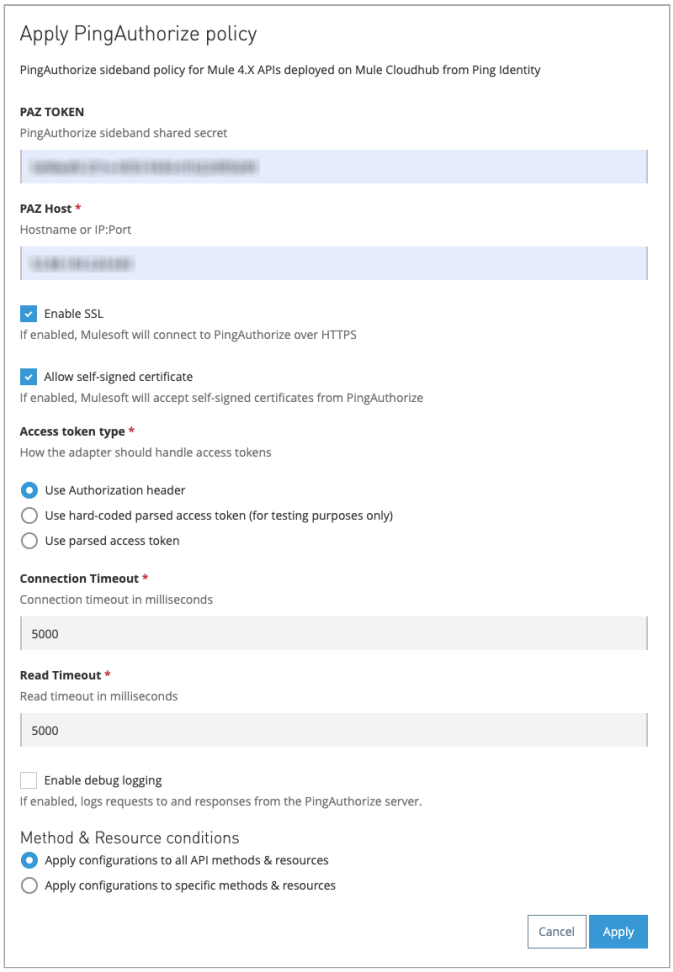
+

> **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.

1. **Optional:** Select the **Enable debug logging** check box to see requests sent to PingAuthorize Server along with responses.
2. **Optional:** Configure **Methods & Resource Conditions**.

   See Resource-Level Policies for more information.

Copyright © 2025 Ping Identity Corporation



Next steps

If there are any changes to PingAuthorize endpoints, repeat the process explained in step 6 and re-deploy the configuration.

Copyright © 2025 Ping Identity Corporation

PingAuthorize Server Administration Guide

PingIdentity.

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 command-line interface (CLI).

Starting PingAuthorize Server

To start PingAuthorize Server in a Unix/Linux computing environment, use the `bin/start-server` command-line interface (CLI) command. On Windows, use the `bat/start-server.bat` command.

Steps

1. In a terminal window, 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`.

Copyright © 2025 Ping Identity Corporation

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.

       ```
       cp ping-authorize.service /etc/systemd/
       ```

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

       ```
       $ 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.

Copyright © 2025 Ping Identity Corporation

- 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

Copyright © 2025 Ping Identity Corporation

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.
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.

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.

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.

Copyright © 2025 Ping Identity Corporation

Starting PingAuthorize Policy Editor

For a manual software installation, use the **`start-server`** command-line interface (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`	pap.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 cross-origin resource sharing (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.

Copyright © 2025 Ping Identity Corporation

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 Policy Editor non-interactively.
`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 Uniform Resource Locator (URL). Used when the Policy Editor is set up in OIDC mode.
`PING_SCOPE`	openid email profile additional_scope	Space-separated OIDC scope that the Policy Editor requests during authorization and validates during token verification. Used to override the requested OIDC scopes configured during server setup.
`PING_OIDC_TLS_VALIDATION`	NONE	The OIDC Transport Layer Security (TLS) validation setting. Set to `NONE` to configure the Policy Editor to accept self-signed Secure Sockets Layer (SSL) certificates from the OIDC 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 OIDC 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.

Copyright © 2025 Ping Identity Corporation

Environment variable	Example value	Description
`PING_DB_ADMIN_USERNAME`	db_admin	The username the **`setup`** tool uses when upgrading the policy database (H2 only).
`PING_DB_ADMIN_PASSWORD`	$3cr3T	The password the **`setup`** tool uses when upgrading the policy database (H2 only).
`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 (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-scheduler.org/documentation/quartz-2.3.0/tutorials/crontrigger.html//[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 `[.parmname]` SERVER_ROOT `/policy-backup`. Related environment variables: `PING_ADMIN_PORT`, `PING_BACKUP_SCHEDULE`
`PING_ENABLE_API_HTTP_CACHE`	false	Controls the API HTTP caching 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.
`PING_DISABLE_SNI_HOSTNAME_CHECKS`	false	Determines whether PingAuthorize performs SNI hostname checks. By default, these checks are disabled.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

> **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.

You can find specific details about the functionality of the API security gateway in the following topics:

- API gateway request and response flow
- Gateway configuration basics
- API security gateway authentication
- API security gateway policy requests

Copyright © 2025 Ping Identity Corporation

- API security gateway HTTP 1.1 support
- Gateway error templates
- Tuning API security gateway performance

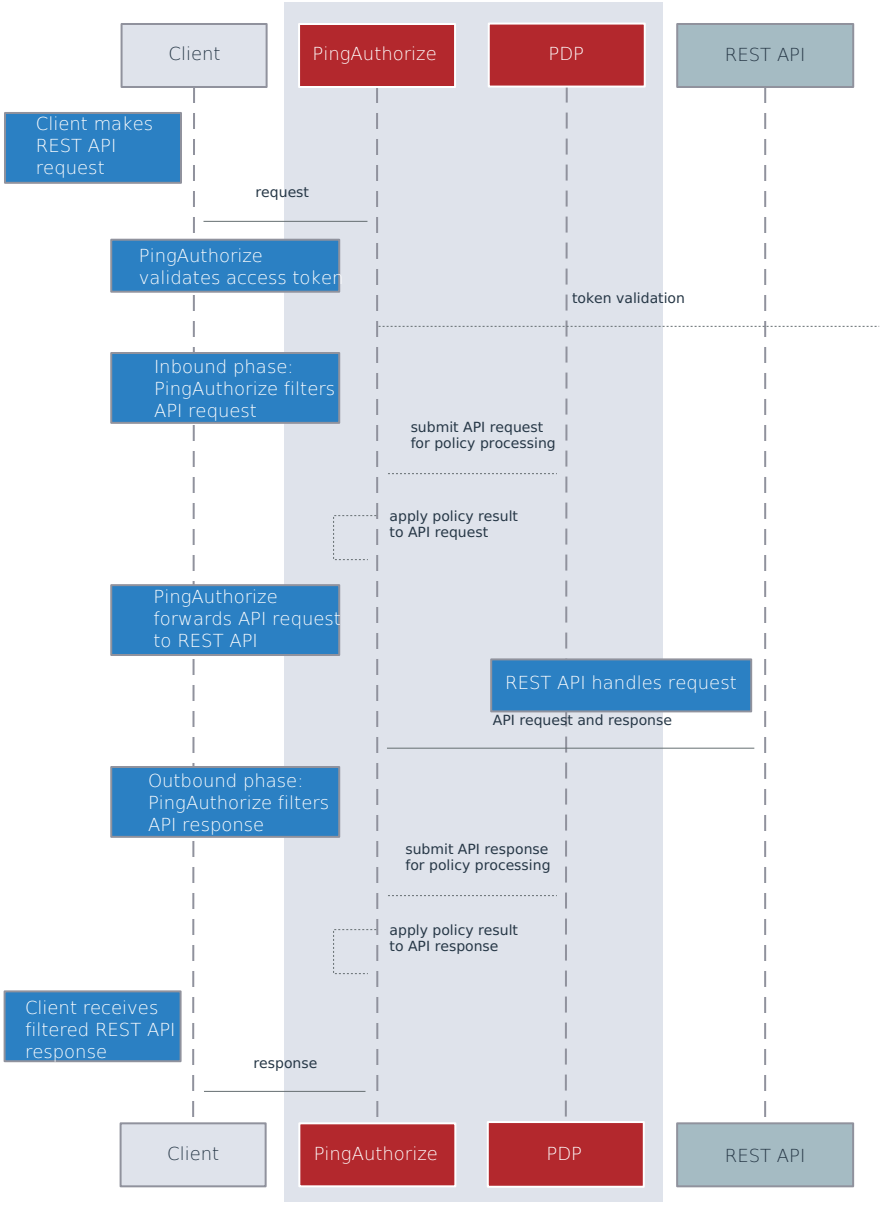
API gateway request and response flow

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 inbound 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 outbound response to the client.

Copyright © 2025 Ping Identity Corporation



The API gateway supports only JavaScript Object Notation (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 endpoint
- One or more API external servers

Copyright © 2025 Ping Identity Corporation

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.

Tuning API security gateway performance

The API security gateway processes and enforces policy decisions for HTTP requests in real time. Although its default configuration is suitable for many environments, high-throughput scenarios, such as load tests or production systems handling hundreds of concurrent requests per second, often require tuning to maintain performance and stability.

In high-throughput scenarios, you might observe increased response times, timeouts, or degraded throughput. These issues typically stem from resource contention, such as an overwhelmed request processing queue or thread pool exhaustion. Because the gateway handles requests asynchronously using multiple internal thread pools, its performance depends on how well those resources are configured for the demands of your workload.

Copyright © 2025 Ping Identity Corporation

The following sections introduce the gateway's asynchronous request flow and explain how to adjust key configuration parameters to align system behavior with your performance goals.

Gateway request flow

The security gateway uses a two-level thread model for asynchronous request processing. This model separates request acceptance from request processing, improving responsiveness under load.

HTTP connection handler threads

These threads accept incoming client connections, parse HTTP requests, and place them into an internal processing queue.

Gateway forward threads

These threads retrieve requests from the internal queue, evaluate access control policies, forward the requests to the backend API, and wait for responses.

If the backend API server is slow to respond, forward threads remain occupied longer, and the queue of waiting requests grows. If requests remain in the queue too long, they could time out.

Performance tuning ensures the system has enough processing threads to keep up with the rate of incoming requests and the latency of backend responses.

Tuning parameters

You should increase the number of HTTP connection handler threads and gateway forward threads if you need to serve higher throughput (more than 10 requests per second). Use the following configuration properties to tune gateway performance:

`num-request-handlers`

This property controls the size of the connection handler thread pool and determines how many concurrent client connections the server can accept.

Example configuration

```
dsconfig set-connection-handler-prop \
  --handler-name "HTTP Connection Handler"  \
  --set num-request-handlers:100
```

`num-forward-threads`

This property controls the size of the worker thread pool and determines how many requests can be processed in parallel. If your backend API has a high response time, you should increase this value.

Example configuration

```
dsconfig set-http-servlet-extension-prop \
  --extension-name Gateway  \
  --set num-forward-threads:500
```


Copyright © 2025 Ping Identity Corporation

> **Tip**
>
> For optimal performance, the `num-forward-threads` value should be 5 - 10 times greater than the `num-request-handlers` value.

`response-timeout`

If your backend API server has a long response time, you must ensure the gateway's timeout for that external server is configured appropriately. This prevents the gateway from terminating a request too early.

> **Important**
>
> The security gateway has a global request timeout of 30 seconds. You must set the external server's response timeout to a value less than 30 seconds for proper error handling.

Example configuration

```
dsconfig set-external-server-prop \
  --server-name "Example server"  \
  --set "response-timeout:25 s"
```

Monitoring security gateway performance

After tuning the security gateway for higher throughput, use the following monitor providers to observe gateway request processing:

Gateway Timing Monitor Provider

Reports the count and total duration for each phase of request processing in the gateway.

Gateway Request Type Timing Monitor Provider

Reports the count, total and average times of each gateway API endpoint, grouped by HTTP method (GET, POST). The monitor only lists executed gateway API endpoints.

Gateway Statistics Monitor Provider

Reports statistics related to gateway executions. Currently, this monitor only reports the gateway forward queue size.

Analyzing metrics from these providers can help you determine if you've resolved the slowdown or if further tuning is required. Learn more about server monitoring in Managing monitoring.

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 doesn't require the authentication of requests, the default policy set requires bearer token authentication.

Copyright © 2025 Ping Identity Corporation

To support this, the gateway uses the configured access token validators to evaluate bearer tokens that are included in incoming requests. The validation result 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

Statement:
  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 how client authentication is handled:

`http-auth-evaluation-behavior`

Determines whether the Gateway API Endpoint evaluates or modifies the HTTP authentication scheme and whether this scheme is forwarded to the API server.

This property accepts the following values:

`do-not-evaluate`

The Gateway API Endpoint doesn't evaluate or modify the HTTP authentication scheme. This can be useful when implementing an authentication scheme that doesn't evaluate bearer tokens, such as MTLS.

If the client request includes an `Authorization` header, the PingAuthorize Server forwards the unmodified header to the external API server.

> **Note**
>
> If you specify this value, policies protecting this endpoint should not enforce constraints on request authentication, such as the validity of the access token. The default policy snapshot enforces such a constraint in the `Token Validation` policy.

`evaluate-and-forward`

The Gateway API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server forwards the unmodified header to the external API server unless a policy decision directs otherwise.

This value is set by default.

`evaluate-and-discard`

The Gateway API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server removes this header before forwarding the request to the external API server.

Copyright © 2025 Ping Identity Corporation

`evaluate-and-replace`

The Gateway API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server replaces this header with one containing the basic authentication credentials defined for the external API server.

> **Note**
>
> If you specify this value, make sure your authorization policies enforce an appropriate level of authorization for the client.

Example

```
bin/dsconfig set-gateway-api-endpoint-prop \
  --endpoint-name <your-endpoint-name> \
  --set http-auth-evaluation-behavior:evaluate-and-replace
```

In this example, the `http-auth-evaluation-behavior` property is set to `evaluate-and-replace`.

`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.

Example

```
bin/dsconfig set-gateway-api-endpoint-prop \
  --endpoint-name <your-endpoint-name> \
  --set access-token-validator:example-token-validator
```

In this example, the `access-token-validator` property is set to `example-token-validator`.

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.

Copyright © 2025 Ping Identity Corporation

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 to view policy request attributes in action, particularly when first developing API security gateway policies. Learn more in Policy Decision logger.

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
`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`.

Copyright © 2025 Ping Identity Corporation

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
`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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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
`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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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
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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
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`.

Copyright © 2025 Ping Identity Corporation

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.

`xref:paz_gw_api_endpoint_config_parms.adoc[] 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.

Copyright © 2025 Ping Identity Corporation

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 statement extension.

Copyright © 2025 Ping Identity Corporation

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 statement changes

The API security gateway does not support using statements 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.

Copyright © 2025 Ping Identity Corporation

- 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.

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`

Copyright © 2025 Ping Identity Corporation

- `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"
}
```

Copyright © 2025 Ping Identity Corporation

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

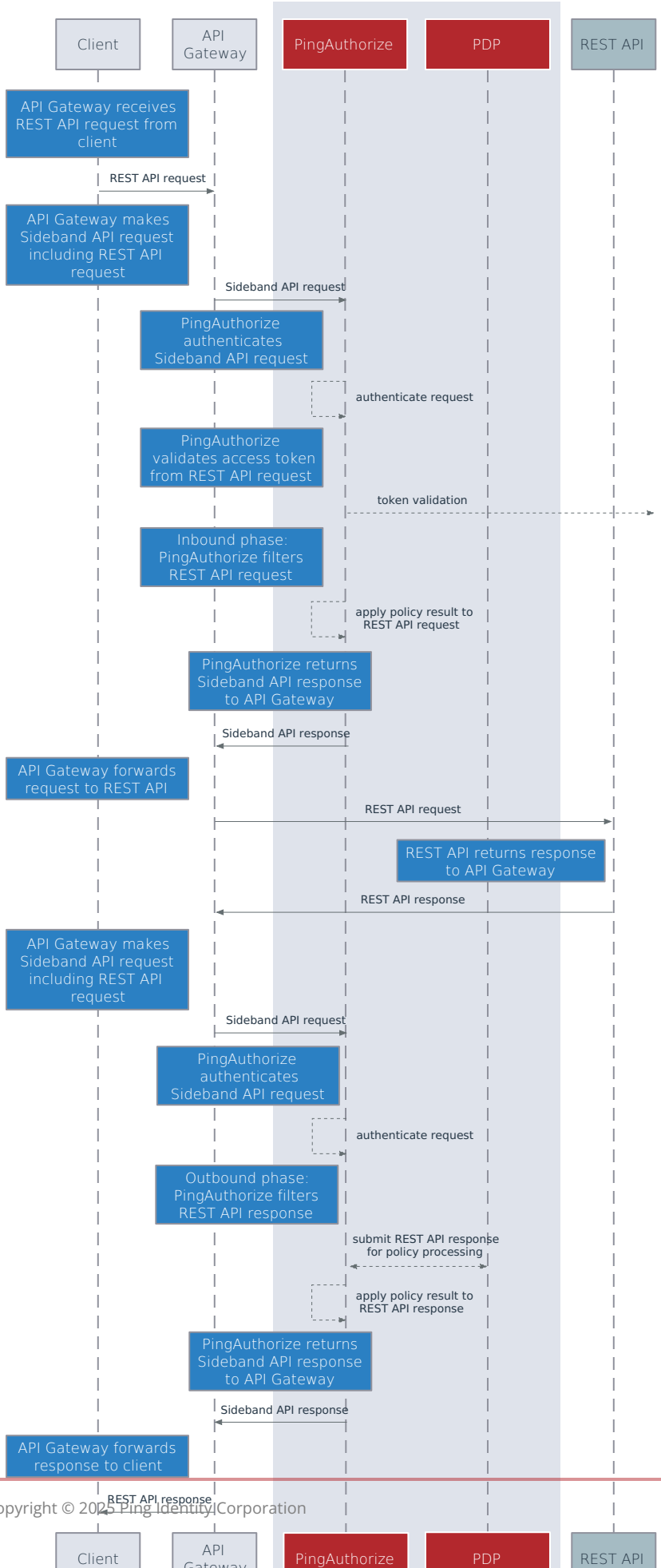
Using the sideband API, you can configure the PingAuthorize Server instead as an adapter to an external API gateway. In sideband mode, an API gateway integration point intercepts client traffic to a backend REST API service and passes intercepted traffic to the PingAuthorize sideband API.

API gateway integration

Enable attribute-based access control (ABAC) through your API gateway by installing the PingAuthorize API integration adapter (where supported) and connecting to the Sideband API.

For more information on specific API gateway integrations, see PingAuthorize Integrations.

Copyright © 2025 Ping Identity Corporation



Copyright © 2025 Ping Identity Corporation

Processing steps

1. When the API gateway receives a request from an API gateway adapter, 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 credential that the sideband API might require an API gateway adapter to present. For more information, see Authenticating to the Sideband API.

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.

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.

Copyright © 2025 Ping Identity Corporation

> **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

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:

Copyright © 2025 Ping Identity Corporation

```
{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.

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. Learn more in Creating a shared secret.
2. Update the API gateway plugin to use the new shared secret.
3. Remove the previous Sideband API Shared Secret. Learn more in Deleting a shared secret.

Customizing the shared secret header

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

Copyright © 2025 Ping Identity Corporation

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
  ```

API server request authentication

As with the API security gateway, API server requests authorized by the Sideband API don't require authentication. However, the default policy set requires bearer token authentication.

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

Policies use this authentication information to affect the processing requests and responses. For example, 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

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

Sideband API Endpoints include the following configuration properties to specify how client authentication is handled:

`http-auth-evaluation-behavior`

Determines whether the Sideband API Endpoint evaluates or modifies the HTTP authentication scheme and whether this scheme is forwarded to the API server through the API gateway.

This property accepts the following values:

Copyright © 2025 Ping Identity Corporation

`do-not-evaluate`

The Sideband API Endpoint doesn't evaluate or modify the HTTP authentication scheme. This can be useful when implementing an authentication scheme that doesn't evaluate bearer tokens, such as MTLS.

If the client request includes an `Authorization` header, the PingAuthorize Server forwards the unmodified header to the external API server through the API gateway.

> **Note**
>
> If you specify this value, policies protecting this endpoint should not enforce constraints on request authentication, such as the validity of the access token. The default policy snapshot enforces such a constraint in the `Token Validation` policy.

`evaluate-and-forward`

The Sideband API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server forwards the unmodified header to the external API server through the API gateway unless a policy decision directs otherwise.

This value is set by default.

`evaluate-and-discard`

The Sideband API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server removes this header before forwarding the request to the external API server through the API gateway.

`evaluate-and-replace`

The Sideband API Endpoint evaluates the provided authentication credentials and makes authentication information available for policy evaluation. If the client request includes an `Authorization` header, the PingAuthorize Server replaces this header with one containing the basic authentication credentials defined for the external API server.

> **Note**
>
> If you specify this value, make sure your authorization policies enforce an appropriate level of authorization for the client.

Example

```
bin/dsconfig set-sideband-api-endpoint-prop \
  --endpoint-name <your-endpoint-name> \
  --set http-auth-evaluation-behavior:evaluate-and-replace
```

In this example, the `http-auth-evaluation-behavior` property is set to `evaluate-and-replace`.

Copyright © 2025 Ping Identity Corporation

`access-token-validator`

Sets the access token validators that the Sideband API Endpoint uses. By default, this property has no value, and the Sideband 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 Sideband 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.

Example

```
bin/dsconfig set-sideband-api-endpoint-prop \
  --endpoint-name <your-endpoint-name> \
  --set access-token-validator:example-token-validator
```

In this example, the `access-token-validator` property is set to `example-token-validator`.

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.

Policy request attributes 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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`domain`	Unused.	String
`identityProvider`	Name of the Access Token Validator that evaluates the bearer token in an incoming request.	String
`service`	Identifies the API service. By default, this value is set to the name of the Sideband API Endpoint. To override the default value, set the Sideband API Endpoint's `service` property. Multiple Sideband API Endpoints can use the same service value.	String

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

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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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
`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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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.

Copyright © 2025 Ping Identity Corporation

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.

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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
```


Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

- 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:

Copyright © 2025 Ping Identity Corporation

```
#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
- SCIM configuration basics
- SCIM endpoints
- SCIM authentication
- SCIM policy requests
- Lookthrough limit for SCIM searches
- Disabling the SCIM REST API

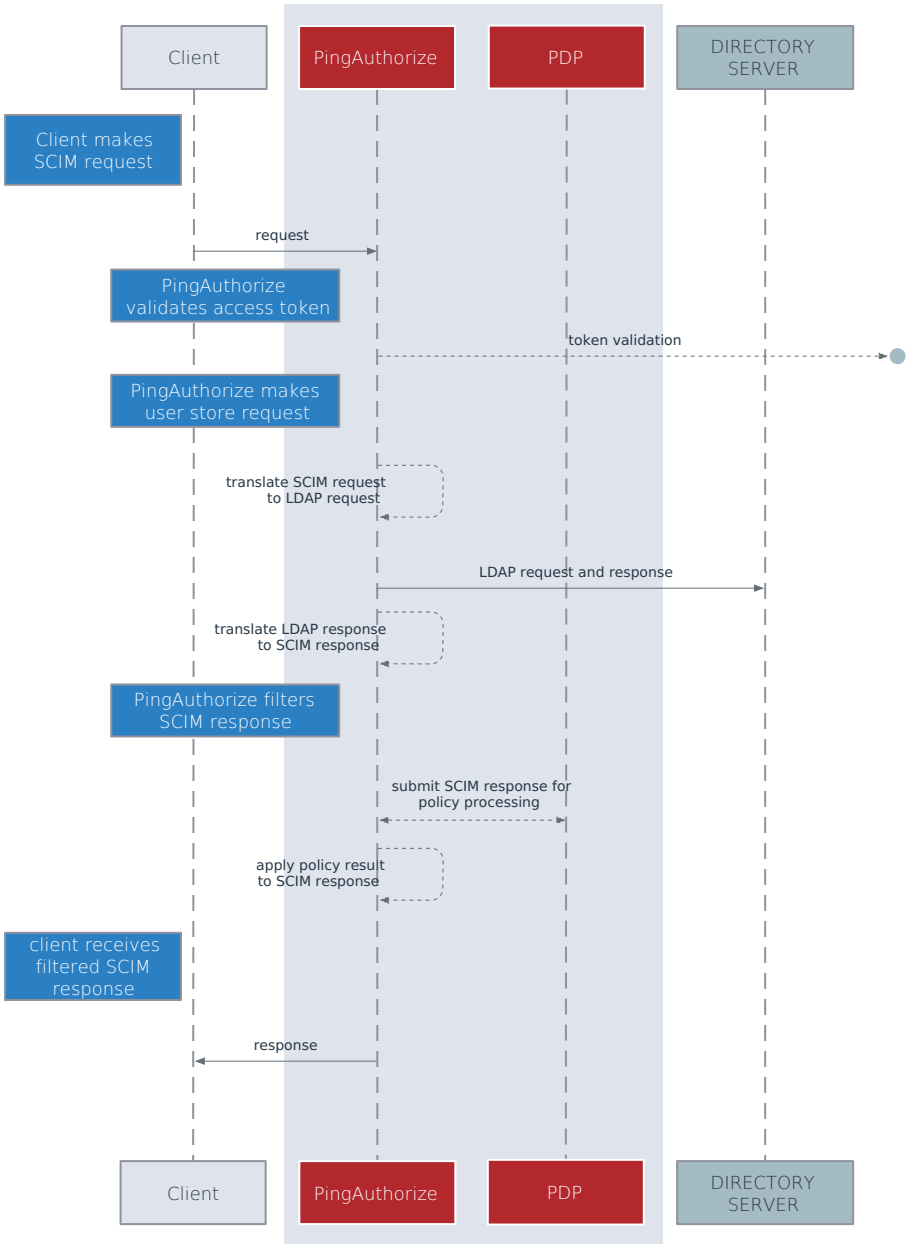
SCIM API request and response flow

The System for Cross-domain Identity Management (SCIM) REST API provides an HTTP API for data contained in a user store.

Copyright © 2025 Ping Identity Corporation

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 statements 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

```
{pingdir}/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`:

Copyright © 2025 Ping Identity Corporation

```
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:

Copyright © 2025 Ping Identity Corporation

```
{"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
`/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

Copyright © 2025 Ping Identity Corporation

Endpoint	Description	Supported HTTP methods
`/<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.

Copyright © 2025 Ping Identity Corporation

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`
	Learn more about authorizing searches in About SCIM searches.

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.

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
`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

Copyright © 2025 Ping Identity Corporation

Policy request attribute	Description	Type
`identityProvider`	Name of the access token validator that evaluates the bearer token used in an incoming request.	String
`service`	Identifies the SCIM service and resource type using a value of the form `SCIM2.<resource type>`. For example, for a request using the "Users" resource type, the service value would be `SCIM2.Users`.	String

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

Attribute	Description	Type
`HttpRequest.AccessToken`	Parsed access token. For more information, see the following table.	Object
`HttpRequest.ClientCertificate`	Properties of the client certificate, if one is used.	Object
`HttpRequest.CorrelationId`	A unique value that identifies the request and response, if available.	String
`HttpRequest.IPAddress`	The client IP address.	String
`HttpRequest.QueryParameters`	Request URI query parameters.	Object
`HttpRequest.RequestBody`	The request body, if available. This attribute is available for POST, PUT, and PATCH requests.	Object
`HttpRequest.RequestHeaders`	The HTTP request headers.	Object
`HttpRequest.RequestURI`	The request URI.	String
`HttpRequest.ResourcePath`	Uniquely identifies the SCIM resource that is being requested, in the format `<Resource Type>/<SCIM ID>`, as the following example shows: `Users/0450b8db-f055-35d8-8e2f-0f203a291cd1`	String
`HttpRequest.ResponseBody`	The response body, if available. This attribute is provided only for outbound policy requests.	Object
`HttpRequest.ResponseHeaders`	The HTTP response headers, if available.	Object
`HttpRequest.ResponseStatus`	The HTTP response status code, if available.	Number

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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

Copyright © 2025 Ping Identity Corporation

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

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

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

Copyright © 2025 Ping Identity Corporation

Attribute	Description	Type
`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.	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

SCIM policy processing involves denying or modifying a search request and then filtering the results.

Copyright © 2025 Ping Identity Corporation

Policy processing for System for Cross-domain Identity Management (SCIM) searches occurs in the following phases:

1. Policies deny or modify a search request. Learn more in Search request authorization.
2. Policies filter the search result set. Learn more in Search response authorization.

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, statements in the policy result are applied to the search filter, giving statements a chance to alter the filter.

> **Note**
>
> You can only use statement types that are written specifically for the `search` action. For example, you can use the Add Filter statement type to constrain the scope of a search.

You can also use the Combine SCIM Search Authorizations statement type at this point. If you use this statement, search results are authorized by using a special mode described in Search response authorization.

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, statements in the policy result are applied to the SCIM resource, which gives statements 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 statement 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. Statements that the policy result returns are applied iteratively to each member of the result set. The input object that is provided to statements also contains a Resources array, but it contains only the single result currently under consideration.

The following JSON provides an example input object:

Copyright © 2025 Ping Identity Corporation

```
{
  "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:

Copyright © 2025 Ping Identity Corporation

```
$ 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
```

`<proxying-request-processor>` is the request processor handling the entries targeted by the search.

About this task

PingAuthorize performs SCIM searches using LDAP requests. After you complete the steps below, PingAuthorize creates LDAP requests that include request controls asking 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, 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:

   ```
   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.

   1. 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.

Copyright © 2025 Ping Identity Corporation

2. Find the corresponding adapter by running the following command:

```
$ dsconfig list-store-adapters
```

3. Find the `structural-ldap-objectclass`, `include-base-dn`, and `include-filter` values for the adapter by running the following 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:

1. 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`.

Copyright © 2025 Ping Identity Corporation

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
```

2. 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
```

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

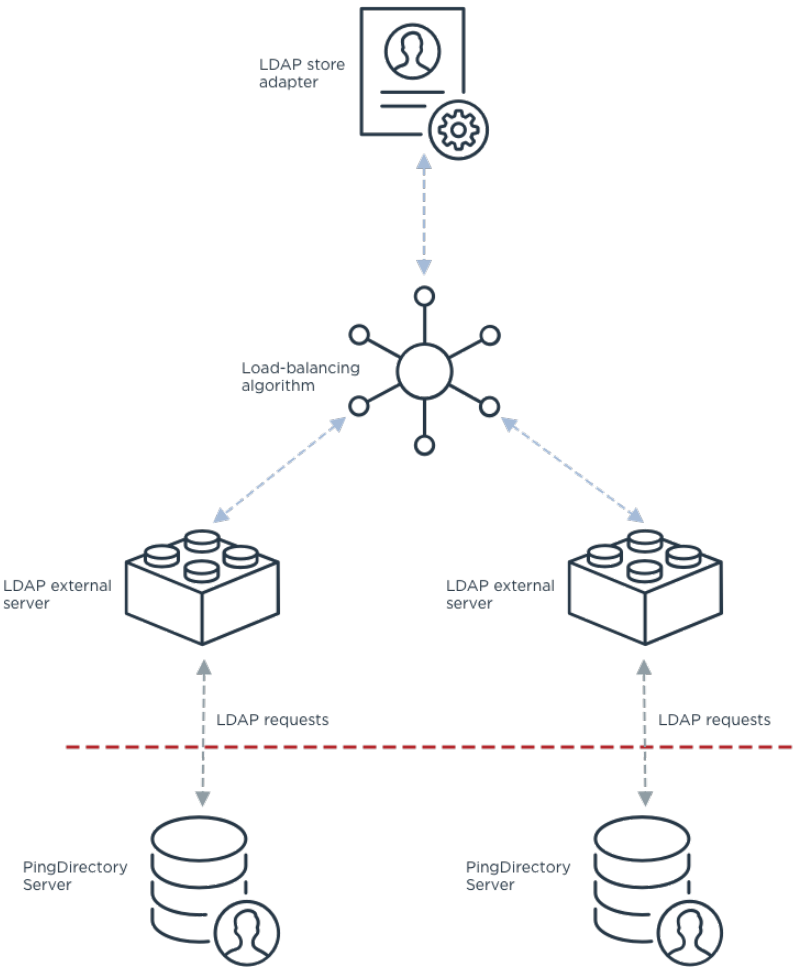
About the SCIM user store

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.

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.

Copyright © 2025 Ping Identity Corporation



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.
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.

Copyright © 2025 Ping Identity Corporation

> **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.

LDAP external servers monitor and report the availability of backend LDAP servers using LDAP health checks. See LDAP health checks.

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
- Defining the LDAP user store manually

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.

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.

Copyright © 2025 Ping Identity Corporation

Option	Description
`--governanceBindPassword`	The password for the above account.
`--userStore`	The host, LDAP / LDAPS port, and optional location of a backend LDAP server. You can specify this option once per each backend server.
`--userStoreBaseDN`	The base DN under which entries are stored.
`--userObjectClass`	The structural LDAP object class of entries for the SCIM subsystem to handle if `--initialSchema` has the `none` or `pass-through` value.
`--initialSchema`	The SCIM schema and resource type configuration to use. Supports the following values: • `pass-through` Creates a pass-through SCIM resource type called `Users` for the LDAP object class specified by the `--userObjectClass` option. • `user` Creates a mapping SCIM resource type called `Users` with an example schema. For more information about this schema, see `<server-root>/resource/starter-schemas/README.txt`. • `none` Does not create a SCIM resource type.

For more information about running **`create-initial-config`**, see its help by running the following command:

```
create-initial-config --help
```

When using **`create-initial-config`** noninteractively, you should also run **`prepare-external-store`** for each backend LDAP server. This tool creates a privileged user account on the LDAP server for use by 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`.

Copyright © 2025 Ping Identity Corporation

> **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:

   ```
   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:

   ```
   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.

Copyright © 2025 Ping Identity Corporation

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:

   ```
   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, 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`

Copyright © 2025 Ping Identity Corporation

Task	Corresponding command example
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

Copyright © 2025 Ping Identity Corporation

• Joining a topology with manage-topology

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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'
```


Copyright © 2025 Ping Identity Corporation

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:

   ```
   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:

   ```
   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:

   ```
   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:

   ```
   manage-topology add-server \
     --remoteServerHostname ds1.example.com \
     --remoteServerPort 636 \
     --remoteServerConnectionSecurity useSSL \
     --remoteServerBindDN "cn=Directory Manager" \
     --remoteServerBindPassword password \
     --adminUID admin \
     --adminPassword password
   ```

Copyright © 2025 Ping Identity Corporation

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:

   ```
   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.

Copyright © 2025 Ping Identity Corporation

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. 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. If the `cn=monitor` entry has a value for the `unavailable-alert-type` attribute, PingAuthorize will consider the PingDirectory server UNAVAILABLE. If the `cn=monitor` entry has a value for the `degraded-alert-type` attribute, PingAuthorize will consider the PingDirectory server DEGRADED.

Copyright © 2025 Ping Identity Corporation

Health check	Description
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.

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.

   1. Enter the number to create a new LDAP health check, then press `n` to create a new health check from scratch.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

Managing Server SDK Extensions

You can create extensions that use the Server SDK to add new functionality to your PingAuthorize Server.

To download the PingAuthorize Server SDK, go to the PingAuthorize downloads page and click the **Add-ons** tab.

Copyright © 2025 Ping Identity Corporation

About the Server SDK

Extension bundles are installed from a `.zip` archive or a file system directory. You can use the **`manage-extension`** tool to install or update any extension that is packaged using the extension bundle format. It opens and loads the extension bundle, confirms the correct extension to install, stops the server if necessary, copies the bundle to the server install root, and then restarts the server.

> **Note**
>
> You can only use the **`manage-extension`** tool with Java extensions packaged using the extension bundle format. Groovy extensions do not use the extension bundle format. For more information, see the `docs/getting-started/java-extensions.html` directory in your Server SDK download, which describes the extension bundle format and how to build an extension.

Available types of extensions

Learn more about specific extension types in the `docs/getting-started/extension-types.html` page in your Server SDK build.

The Server SDK supports the following extensions:

Extensions	Description
Access Loggers	Record information about operations processed by the server. This includes information about connections that are established and closed, as well as whenever requests are received from clients or responses are returned to clients.
Access Token Validators	Validate access tokens submitted by client applications for access to protected HTTP resources.
Alert Handlers	Convey alert notifications generated within the server to administrators so they can take appropriate action. Alert notifications report significant errors, warnings, or events that may warrant immediate attention.
Error Loggers	Record information about events occurring in the server, including warning and error conditions, informational messages, and some limited debugging information (although most debugging information is made available through debug loggers rather than error loggers).
HTTP Operation Loggers	Record information about communication performed by HTTP clients, including requests received and responses written.

Copyright © 2025 Ping Identity Corporation

Extensions	Description
HTTP Servlet Extensions	Create servlets that perform custom processing in response to requests received from HTTP clients. HTTP Servlet Extensions can customize the paths for which they should be invoked, the set of initialization parameters, the initialization order, and an optional set of filters that may be used in conjunction with the servlet.
Key Manager Providers	Provide access to Java key managers, which you use to obtain access to a certificate that you might need to present to another system. This includes cases where the server is configured to accept connections from secure clients using SSL or StartTLS, as well as when it needs to establish secure connections to other systems with a certificate for client authentication.
Manage Extensions Plugins	Introduce custom processing at various points in the extension bundle installation process while using the `manage-extension` tool. You can invoke Manage Extension Plugins in the following contexts: • Before any files are copied during a first-time install of the extension bundle • After files are copied during a first-time install of the extension bundle • Before any files are updated during the update of an installed extension bundle • After all files are updated during the update of an installed extension bundle
Monitor Providers	Report information about the state of components within the server, and can be used for health-checking purposes, real-time and historical monitoring, and debugging and troubleshooting. You can use each Monitor Provider instance to generate a single monitor entity, generally with information about a single component of the server.
OAuth Token Handlers	Validate incoming SCIM requests using OAuth 2.0 bearer tokens for authentication. Implementations of this API are responsible for decoding the bearer token and checking it for authenticity and validity.

Copyright © 2025 Ping Identity Corporation

Extensions	Description
Plugins	Introduce custom processing at various points in the server life cycle or in interaction with clients. You can use Plugins to alter some content before the server performs other processing on it. For example, you can use pre-parsing Plugins to alter the content of a request read from a client or reject that request with an error.
Policy Decision Loggers	Enable custom logging behavior for policy decision point (PDP) responses. This includes defining custom log formats and destinations, as well as specifying which policy messages to include or exclude from the loggers based on criteria such as the message type or defined key-values. This extension also enables integrations with external logging systems.
Trust Manager Providers	Provide access to Java trust managers, which the server uses to determine whether to trust a certificate presented to it. This includes cases where a client using SSL or StartTLS presents its own certificate to the server, and also when the client needs to establish secure connections with other systems.
Advices	Carry out custom processing directives included in a policy decision result. For example, you can use an advice to add or remove response content.
Store Adapters	Serve as a native interface to a backend datastore. Store Adapters are aggregated into a SCIM Resource Type in the PingAuthorize Server, which supports a SCIM front-end which can be backed by any number and type of native datastores.
Store Adapter Plugins	Perform processing on Store Adapter operations before and after those operations are processed by a Store Adapter.
Token Resource Lookup Methods	Look up the attributes of an access token owner. Using the result of Access Token Validator processing as input, a Token Resource Lookup Method can query a datastore to obtain the token owner's attributes. These attributes are then made available to policies for making access control decisions.

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 (ABAC).

Copyright © 2025 Ping Identity Corporation

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 an application programming interface (API). 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 policy decision point (PDP), and the application server becomes the policy enforcement point (PEP).

The Authorization Policy Decision APIs consist of the following 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 statements, but the application server must implement the statement.

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 or use token validation within your rules and policies. For more information, see Access token validators or Policy conditions.

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 an individual decision request endpoint, an individual `query` decision request endpoint, and a batch request endpoint that consuming application servers can access using POST requests to the `/governance-engine`, `/governance-engine/query`, or `/governance-engine/batch` paths, respectively. You can find more information about the `/query` endpoint in Policy queries.

The HTTP request 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.

Copyright © 2025 Ping Identity Corporation

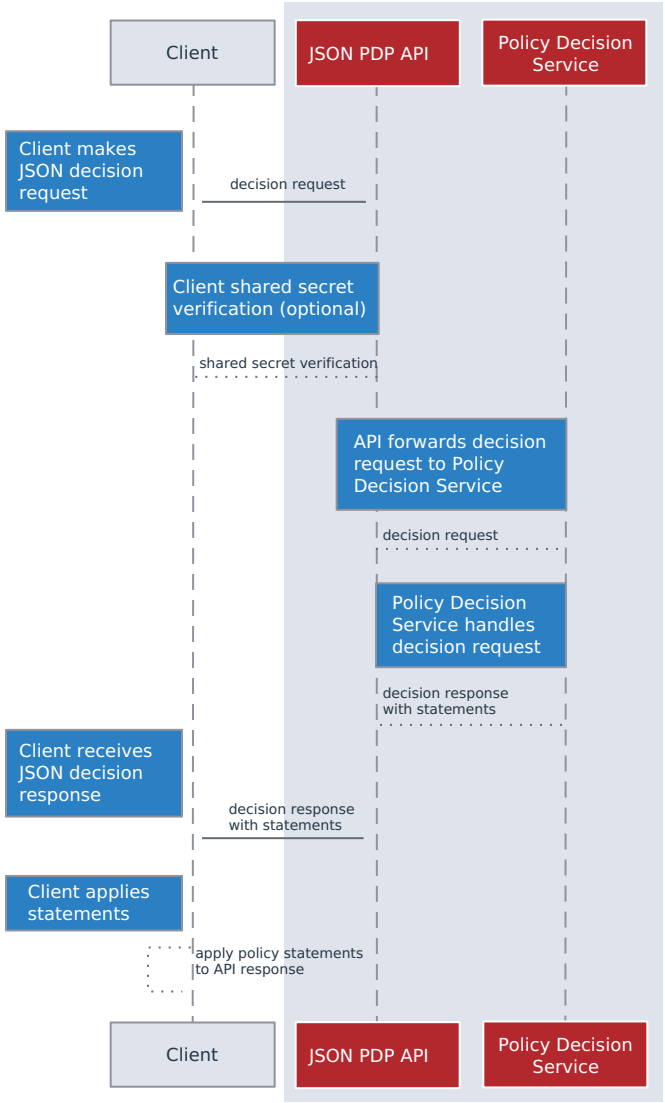
JSON PDP API Endpoint path	Action	Content-Type/Accept	Request data
`/governance-engine`	POST	`application/json`	JSON
`/governance-engine/batch`	POST	`application/json`	JSON
`/governance-engine/query`	POST	`application/json`	JSON

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 Policy Decision Service.
2. When the Policy Decision Service returns a response, the API sends the response to the client.

> **Note**
>
> The Policy Enforcement Point (PEP) must apply any obligations or statements. Learn more about making JSON PDP API requests in the JSON PDP API Reference.

Copyright © 2025 Ping Identity Corporation



JSON PDP API request format

Individual requests

A valid JSON PDP API request is a simple JSON object that a client application sends to the Policy Decision Service. Policies can match a decision request by `service`, `domain`, `action`, or other attributes.

The following table describes the fields in a valid JSON PDP API request:

Field	Value type	Required	Trust Framework type	Example value
`domain`	String	Optional	Domain	`Sales.Asia Pacific`
`action`	String	Optional	Action	`Retrieve`
`service`	String	Optional	Service	`Mobile.Landing page`

Copyright © 2025 Ping Identity Corporation

Field	Value type	Required	Trust Framework type	Example value
`identityProvider`	String	Optional	Identity Provider	`Social Networks.Spacebook`
`attributes`	Map <string, string>	Required	Attribute map	`{"Prospect name": "B. Vo"}`

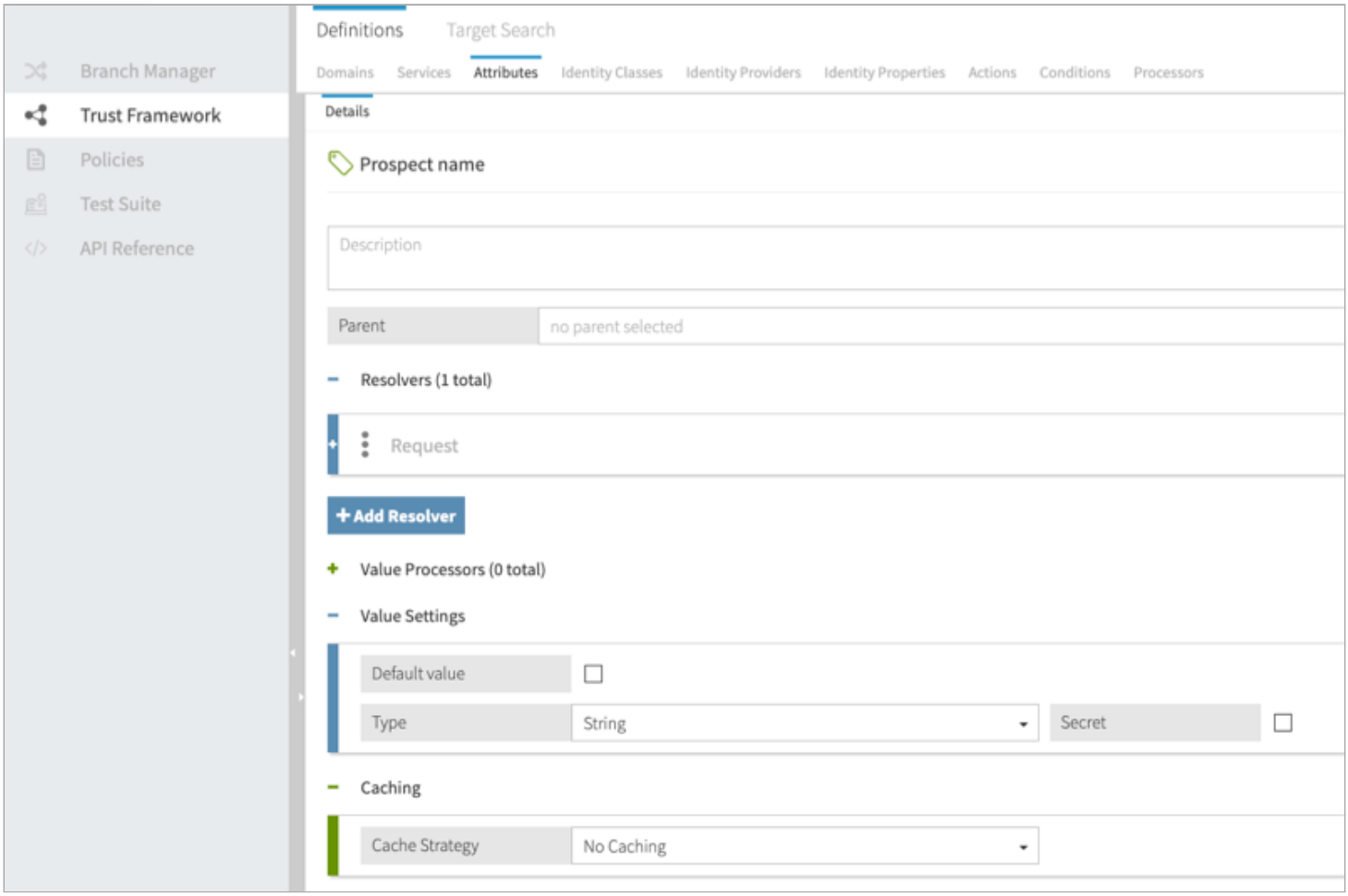
> **Tip**
>
> Although the `attributes` field is required, you can leave it empty.

The following is an example of a valid individual JSON 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 Editor. In this example, the **Prospect name** attribute has a **Request** resolver and a **Type** of string.

Copyright © 2025 Ping Identity Corporation



> **Note**
>
> The Trust Framework attribute name must match the key of the `attributes` map.

Batch requests

Batch requests consist of an array named `requests` that contains individual JSON requests.

The following example shows a valid batch JSON request:

Copyright © 2025 Ping Identity Corporation

```
{
  "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"
      }
    }
  ]
}
```

Query requests

Query requests differ from individual and batch JSON requests in allowing the following types of attributes:

- Unbounded: A request attribute without any value specified. These attributes' values can be populated at decision runtime by making calls to external services.
- Multivalued: A request attribute with multiple values specified.

Query requests consist of the following fields:

- `query`: An array containing the following elements:
  - `attribute`: The full name of an unbounded, multivalued, or standard authorization attribute.
  - `values`: An optional array defining the values of the attribute. If you include more than one value in this array, the JSON PDP API treats the attribute as multivalued. If the attribute is unbounded, do not include this array.

    > **Note**
    >
    > You cannot leave the `values` array empty in a query request sent in embedded policy decision point (PDP) mode. If you leave the `values` array empty in a request sent in external PDP mode, the relevant attribute is treated as an unbounded attribute.

  The `query` array has the following constraints:
  - At most one attribute can be included without values (unbounded).
  - At most two attributes can be multivalued.
  - At most three attributes can be included in the array, but not all three can be multivalued or unbounded.
- `context` (optional): A JSON object containing the fields included in a typical individual JSON request.

Copyright © 2025 Ping Identity Corporation

You can include single-valued attributes in the `query` or `context` fields. Including a single-valued attribute in the `query` field adds that attribute and its value to each `results` array element in the query response.

The following example asks, "Which actions can Joe perform on the account?":

```
{
  "query": [
    {
      "attribute": "action"
    },
    {
      "attribute": "Subject",
      "values": ["{\"id\": 23, \"name\":\"Joe\"}"]
    },
    {
      "attribute": "Resource",
      "values": ["account"]
    }
  ]
}
```

Learn more in Policy queries.

JSON PDP API response format

After the Policy Decision Service makes a decision, it returns the decision to the JSON PDP API, which then forwards the decision response to the client. JSON PDP API responses include decisions, such as `PERMIT` or `DENY`, and any obligations or statements that were matched during policy evaluation.

Individual response

The following example shows a valid response to an individual JSON request:

Copyright © 2025 Ping Identity Corporation

```
{
  "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": "Statement Name",
      "code": "statement-code",
      "payload": "{\"data\": \"some data\"}",
      "obligatory": true,
      "fulfilled": false,
      "attributes": {}
    }
  ],
  "status": {
    "code": "OKAY",
    "messages": [],
    "errors": []
  }
}
```

The `decision` and `authorized` values identify whether the policies authorize the request, and the `statements` array contains statements to be applied by the policy enforcement point.

> **Note**
>
> `elapsedTime` is measured in microseconds.

Batch response

Batch responses are returned as an array named `responses` that contains individual JSON responses. The order of the responses matches the order of the original requests, meaning the first response in the `responses` array corresponds to the first request in the batch, the second response to the second request, and so on.

The following example shows a valid response to a batch JSON request:

Copyright © 2025 Ping Identity Corporation

```
{
  "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": []
      }
    }
  ]
}
```


Note

`elapsedTime` is measured in microseconds.

Copyright © 2025 Ping Identity Corporation

Query response

The following example shows a valid response to a query request:

```
{
  "requestId": "8245be35-ec9e-40f1-a79a-80890041f4b0",
  "timeStamp": "2023-11-14T03:21:47.734842Z",
  "elapsedTime": 22,
  "results": [
    {
      "attribute": "action",
      "value": "delete",
      "decision": "PERMIT"
    }
  ]
}
```

The `results` array contains a list of `query` attribute values that either produced a `PERMIT` decision or a `DENY` decision, along with any obligations or statements that were matched.

> **Note**
>
> `elapsedTime` is measured in milliseconds.

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.

Copyright © 2025 Ping Identity Corporation

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.

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. Learn more in Creating a shared secret.
2. Update the client to use the new shared secret.
3. Remove the previous JSON PDP API shared secret. Learn more in Deleting a shared secret.

Customizing the shared secret header

By default, the JSON PDP API accepts a shared secret from a client through the CLIENT-TOKEN header.

Copyright © 2025 Ping Identity Corporation

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 is a standards-based HTTP API used to make authorization decisions based on policies configured in 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. Learn more in Requests.

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.

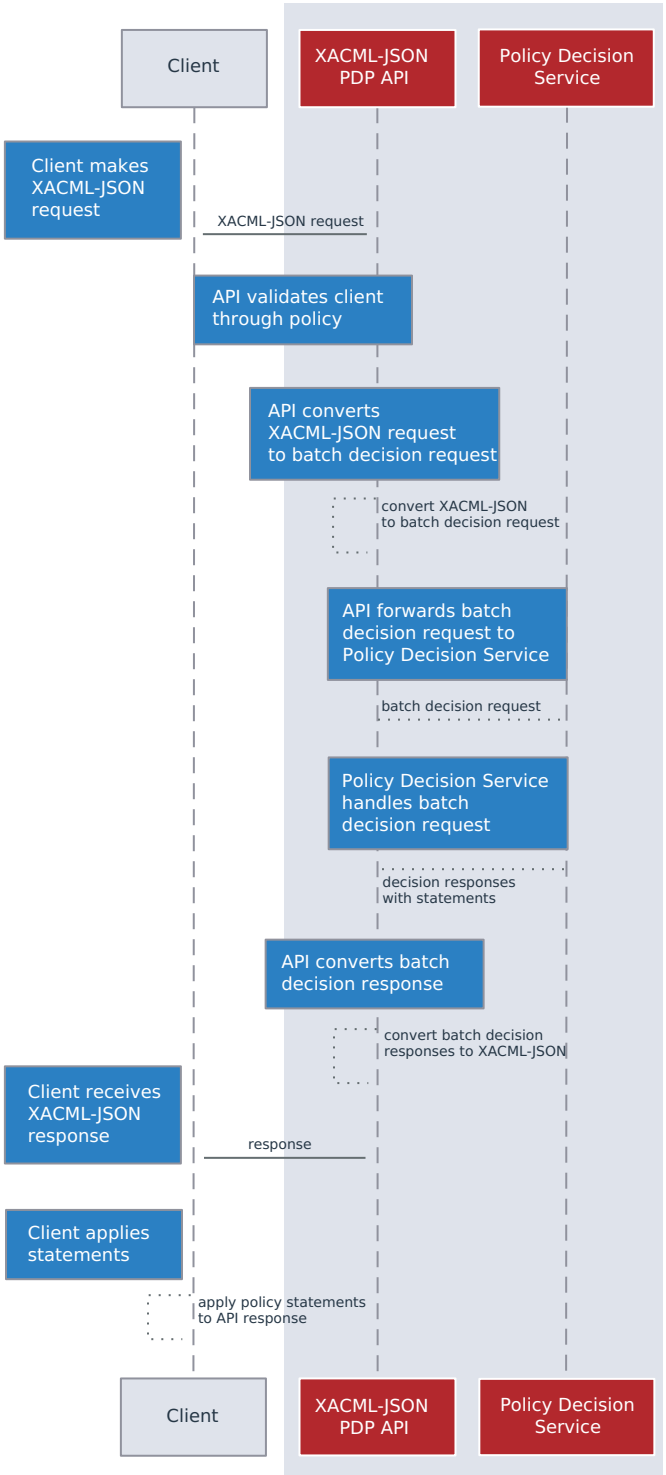
> **Note**
>
> Because this object also supports single decision requests, it's the only supported XACML-JSON request format. Learn more about making XACML-JSON PDP API requests in the XACML-JSON PDP API Reference.

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.

Copyright © 2025 Ping Identity Corporation

2. When the client is successfully authorized, 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.

Copyright © 2025 Ping Identity Corporation

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, refer to Example.

Copyright © 2025 Ping Identity Corporation

```
{
  "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`

Copyright © 2025 Ping Identity Corporation

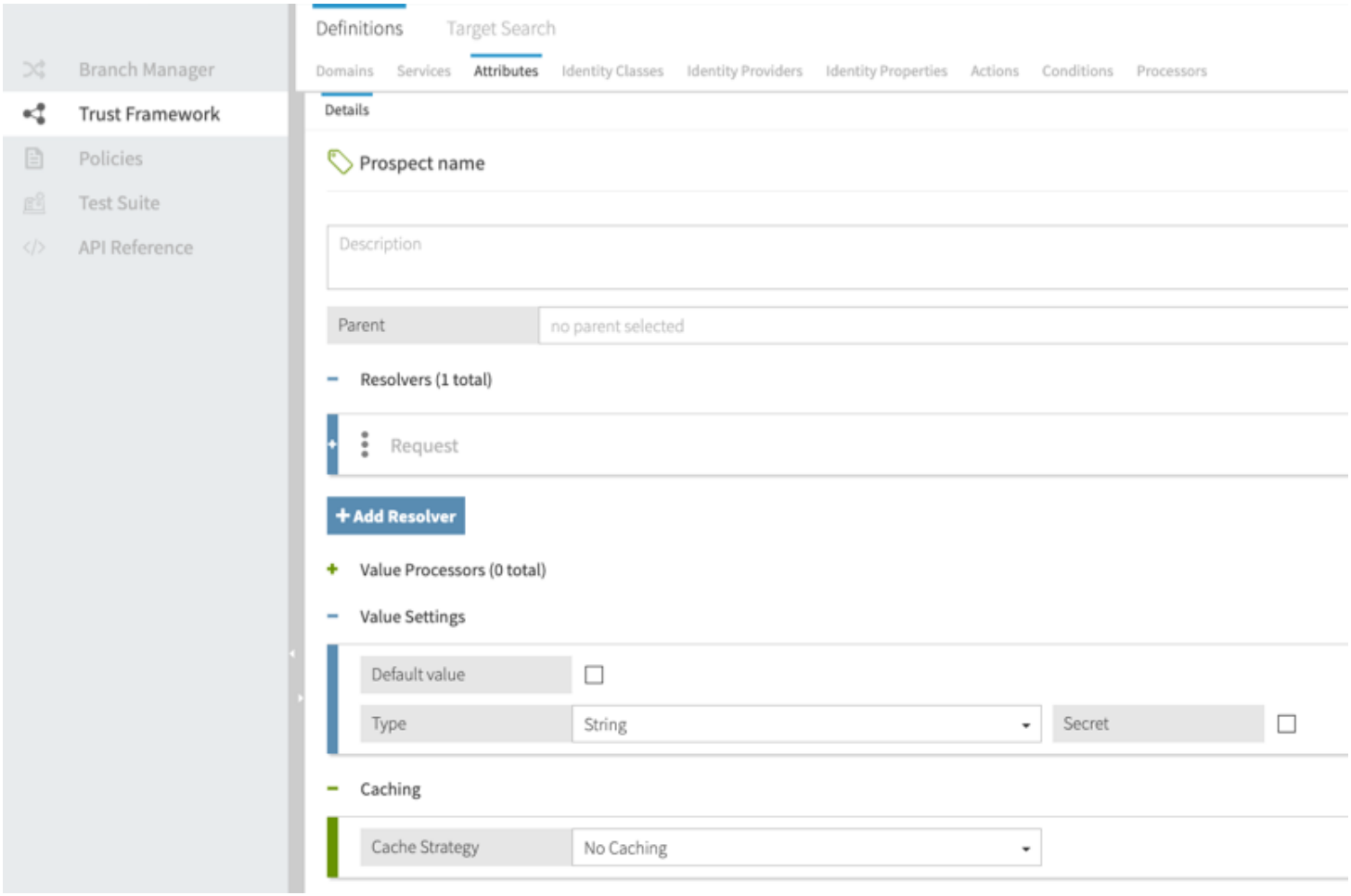
- An identity provider of `Social networks.Spacebook`
- A single attribute named `Prospect name`, with a value of `B. Vo`

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`.

Copyright © 2025 Ping Identity Corporation



> **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.

Copyright © 2025 Ping Identity Corporation

```
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. Learn more in Requests. After the Policy Decision Service determines a decision response, it forwards 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.

Copyright © 2025 Ping Identity Corporation

```
{
  "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.

Copyright © 2025 Ping Identity Corporation

```
{
   "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"
```

Copyright © 2025 Ping Identity Corporation

```
                }
            ]
        }
    ],
    "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
                },
                {
```


Copyright © 2025 Ping Identity Corporation

```
                    "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`
`$.Request.Action[*]`	`$.Attribute[?(@.AttributeId == "action")].Value`	Action	`Update`
`$.Request.Resource[*]`	`$.Attribute[?(@.AttributeId == "service")].Value`	Service	`Peer Recognition.Point allocation`

Copyright © 2025 Ping Identity Corporation

Parent (JSON Path)	Field (JSON Path)	PingAuthorize Trust Framework type	Example value
`$.Request.Environment[*]`	`$.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.Products`
`$.Request.Environment[*]`	`$.Attribute[?(@.AttributeId == "symphonic-idp")].Value`	Identity Provider	`AnyCompany SSO`

Copyright © 2025 Ping Identity Corporation

Parent (JSON Path)	Field (JSON Path)	PingAuthorize Trust Framework type	Example value
`$.Request.Category[*]`	`$.Attribute[?(@.AttributeId == "attribute:User input.Travel")]`	Attribute	`5`

The following is an example response to the previous example request.

The XACML-JSON response contains the decision responses for each of the three decision requests. The order of the decision responses corresponds to the order of the decision requests. In the first decision response, the system policy does not detect any problems and permits the employee to change her point allocation. In the second decision response, the system policy allows the employee to view her own unspent points and indicates that the value is now 0. In the third decision response, the system permits the retrieval of the employee's own product catalog and indicates which of the products should be unlocked for purchase.

Given the response, the web application can now render the content for the three decision requests. It renders the 0 unspent points and all catalog products, with purchase buttons disabled where appropriate.

Copyright © 2025 Ping Identity Corporation

```
{
  "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

Copyright © 2025 Ping Identity Corporation

- 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.

Copyright © 2025 Ping Identity Corporation

2. Configure the Policy Editor with an options file:

    1. Stop the Policy Editor:

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

    2. Run the `setup` tool.

    3. 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
```

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
```


Copyright © 2025 Ping Identity Corporation

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
```

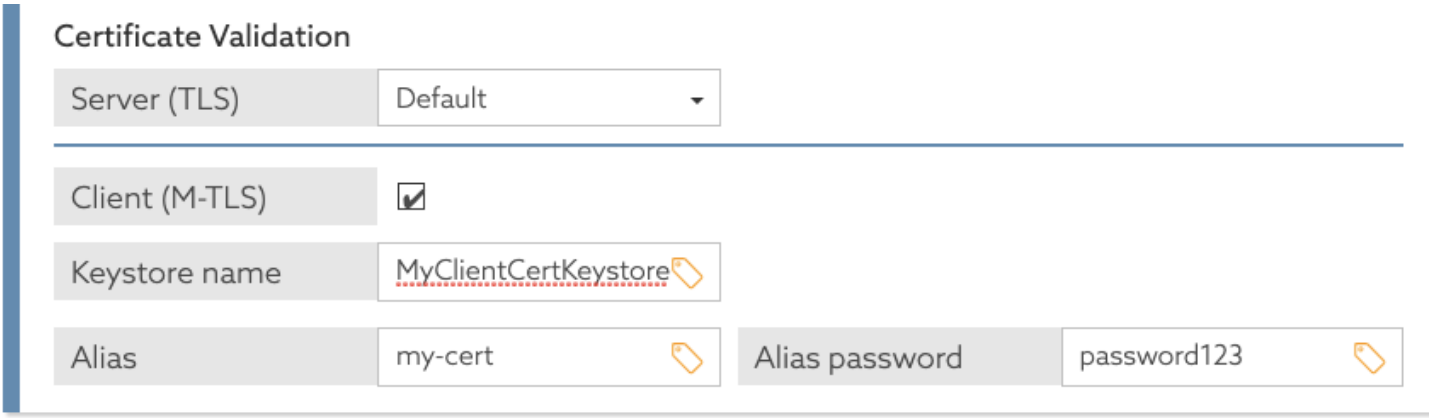
4. Run **`setup`** using the `--optionsFile` argument. Customize all other options as appropriate for your needs.

```
$ 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`



Copyright © 2025 Ping Identity Corporation

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.

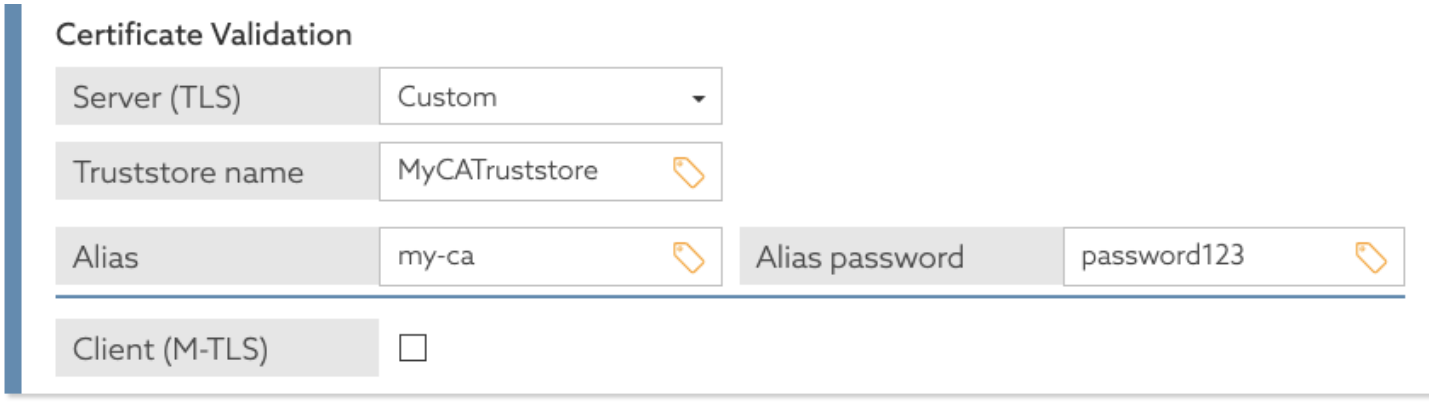
```
truststores:
  - name: MyCATruststore
    resource: /path/to/my-ca-truststore.jks
    password: password123
# Other options omitted for brevity...
```

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`.

Copyright © 2025 Ping Identity Corporation



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 URI using an environment variable

This example takes the scenario in Define policy configuration keys in a development environment and modifies it to specify the Consent API base URI at runtime using an environment variable.

To specify the base URI 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 `ConsentBaseUri`. Instead of hard-coding its value, specify a variable called `CONSENT_BASEURI`.

```
core:
  ConsentBaseUri: ${CONSENT_BASEURI}
# 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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_BASEURI` environment variable and then start the server.

```
$ export CONSENT_BASEURI=https://consent-us-east.example.com/consent/v1; bin/start-server
```

To set a different host name, redefine the `CONSENT_BASEURI` environment variable and restart the server.

```
$ bin/stop-server
$ export CONSENT_BASEURI=https://consent-us-west.example.com/consent/v1; 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 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...
```


Copyright © 2025 Ping Identity Corporation

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.

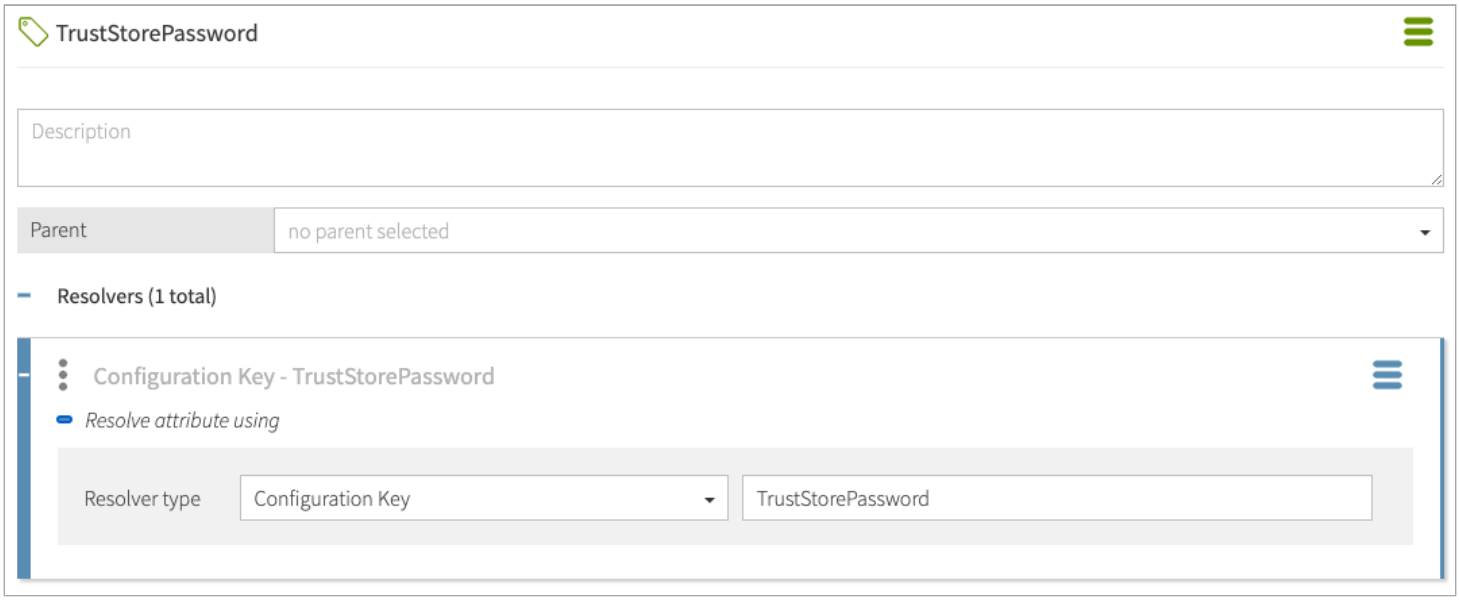
```
$ export TRUST_STORE_PASSWORD=password123; bin/start-server
```

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.

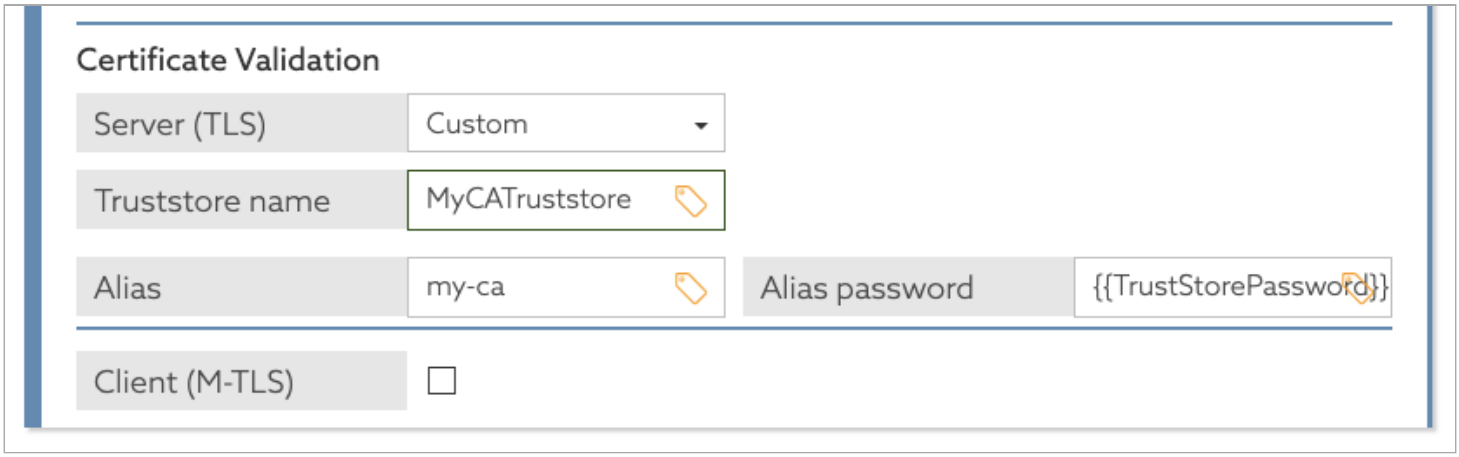
Property	Value
Name	TrustStorePassword
Resolver Type	Configuration Key
Resolver Value	TrustStorePassword

The following image shows the attribute in the Policy Editor.

Copyright © 2025 Ping Identity Corporation



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
```

Changing the default JWT claim for the OIDC user ID

Change the `sub` JSON Web Token (JWT) claim for the OpenID Connect (OIDC) user ID under the `options.yml` file's `core` section.

By default, when a user signs on to the Policy Editor with OIDC, the Policy Editor uses the `sub` JWT claim to:

- Extract the `sub` claim value from the ID token and:
  - Record the `sub` claim value in the **Creator** column of the **Commits** table when the user makes commits (see **Branch Manager > Version Control**).

Copyright © 2025 Ping Identity Corporation

- Make a request to the UserInfo endpoint and:
  - Use the `sub` claim value from the response as the user data.
  - Display the user data in the upper-right corner of the Policy Editor.

If your organization wants to use a non-default claim for the OIDC user ID, such as `email`, define this claim by completing the following steps.

> **Important**
>
> You must configure your OIDC provider to include the claim in both the `UserInfo` endpoint and the ID token for the name to display. Refer to your OIDC provider's documentation for instructions.

Steps

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"
```

   1. (Optional): Update the `email` claim to your organization's preferred claim.

3. Stop the Policy Editor:

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

4. Run `setup` using the `--optionsFile` argument and customize all other options to meet 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:

Copyright © 2025 Ping Identity Corporation

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

6. In the Policy Editor, go to **Branch Manager > Version Control** and commit a policy change.
7. Verify that your claim is being used:
    1. Select any branch and verify that the new claim value appears in the upper-right corner of the Policy Editor.
    2. Verify that the new claim value appears in the **Creator** column of the **Commits** table for the commit you made in step 6.

Configuring the JWKS endpoint cache

Configure the JSON Web Key Set (JWKS) endpoint cache to manage the key set caching behavior for the Policy Editor in OIDC mode.

About this task

To improve performance, the Policy Editor is configured to cache the JWKS endpoint response indefinitely (when key set caching is enabled). Whenever the Policy Editor encounters a key ID not present in the cache, it makes a request to the JWKS endpoint, regardless of the caching configuration.

You can choose from the following configuration values. This configuration option only affects server-side behavior:

Value	Behavior
Any negative integer	Caches the key set indefinitely (default configuration)
0	Disables key set caching
Positive integer	Sets the key set cache expiry time in seconds

> **Note**
>
> In general, using the `options.yml` file to modify the behavior and output of `setup` requires restarting the Policy Editor. If you have already run `setup` once, provide the `--ignoreWarnings` option to overwrite any existing configuration files.
> Doing so, however, overwrites the admin keystore and decision point shared secrets. Additionally, providing `--generateSelfSignedCertificate` overwrites the server keystore. Be sure to back up the admin and server keystores and your original `configuration.yml` file if you intend to reuse them.

Steps

1. Make a copy of the default options file:

    Example:

    ```
    $ cp config/options.yml my-options.yml
    ```


Copyright © 2025 Ping Identity Corporation

2. In the `core` section of the new options file, uncomment the `Authentication.oidcJwksCacheExpirySeconds` field.

    1. **Optional:** Change the default value to set a cache expiry limit or disable key set caching:

    Example:

    ```
    # This option only affects server-side behavior.
    #
    Authentication.oidcJwksCacheExpirySeconds: 3600
    ```

3. If necessary, stop the Policy Editor:

Example:

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

4. Run `setup` using the `--optionsFile` argument and 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 and provide the OIDC well known configuration URL to the `PING_OIDC_CONFIGURATION_ENDPOINT` environment variable at startup:

Example:

```
$ env PING_OIDC_CONFIGURATION_ENDPOINT=<well-known-url>  \
  bin/start-server
```

> **Note**
>
> Instead of using the OIDC well known configuration URL to provide the value for the JWKS endpoint, you can specify different values in the `ui` section of the new options file. Uncomment the fields as specified in the following example and set the URL values for `authorizationEndpoint` and `jwksUri`:
>
> ```
> ui:
>   authClientConfig:
>     authWellKnownEndpoints:
>       authorizationEndpoint: https://<oidc-host>:<oidc-port>/as/authorize
>       jwksUri: https://<oidc-host>:<oidc-port>/JWKS
> ```
>
> If you provide a custom value for `jwksUri` in `options.yml`, omit the `PING_OIDC_CONFIGURATION_ENDPOINT` environment variable in this step.

Copyright © 2025 Ping Identity Corporation

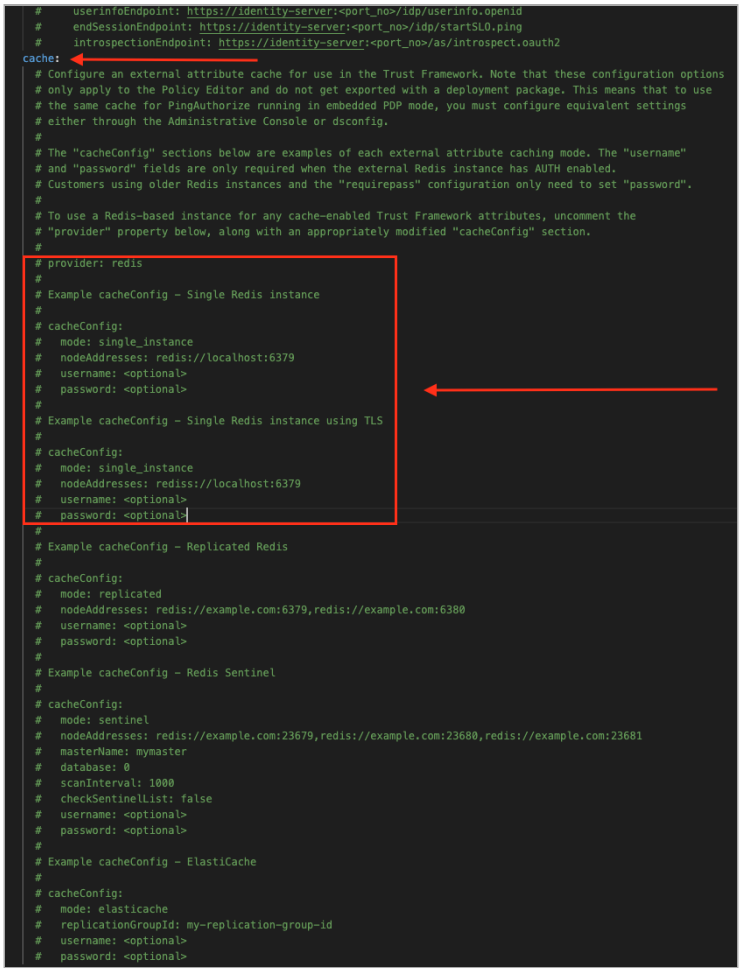
Configuring Trust Framework attribute caching for development

While building and testing policies in a development environment, you can define an external attribute cache for the Trust Framework.

About this task

When you set the Policy Decision Service to external policy decision point (PDP) mode, the Policy Editor is configured by default to cache attribute values in memory on the PingAuthorize Server (for any attributes with a defined caching strategy). Alternatively, you can define an external attribute cache using Redis by configuring the `options.yml` file. The available Redis modes include:

- Single Redis instance
- Single Redis instance using TLS
- Replicated Redis
- Redis Sentinel
- Amazon Web Services (AWS) ElastiCache Redis



Steps

1. Make a copy of the default options file:

Copyright © 2025 Ping Identity Corporation

Example:

```
$ cp config/options.yml my-options.yml
```

2. In the `cache` section of the new options file, uncomment the line `provider: redis` to enable the Redis caching options.
3. Uncomment the desired Redis `cacheConfig` block and modify it to reflect your Redis instance settings.

   Example:

   ```
   cache:
     provider:redis
     cacheConfig:
       mode: single_instance
       nodeAddresses: redis://localhost:6379
   #   username: <optional>
   #   password: <optional>
   ```

   1. **Optional:** Uncomment `username` and `password` and add the appropriate values.
4. Stop the Policy Editor:

   Example:

   ```
   $ bin/stop-server
   ```
5. Run `setup` using the `--optionsFile` argument and 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
   ```
6. Start the Policy Editor:

   Example:

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

Configuring Policy Editor security headers

Use an options file to configure the Policy Editor.

Copyright © 2025 Ping Identity Corporation

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.

   1. 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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
```

Configuring Policy Editor database service connections

You can configure the pooling mechanism and add allowed drivers for database service connections when developing and testing such services in external policy decision point (PDP) mode.

About this task

A database pool is a cache of database connections that PingAuthorize uses to manage system performance. Instead of establishing a new connection each time the server needs to retrieve policy information from the database, PingAuthorize leverages an existing connection from the database pool. PingAuthorize creates a database pool for each database service you define, and the pool configuration specified in the Policy Editor will apply to each pool.

Steps

1. Make a copy of the default options file:

```
$ cp config/options.yml my-options.yml
```

2. In the `databasePools` section of the new `options` file, modify the following database pool properties:

Property	Description
`connectionTimeoutMillis`	Specifies the maximum number of milliseconds that a connection request waits for an available connection in the database pool.
`validationTimeoutMillis`	Specifies the maximum number of milliseconds that a database pool tests a connection for aliveness.

Copyright © 2025 Ping Identity Corporation

Property	Description
`maximumLifetimeMillis`	Specifies the maximum number of milliseconds that a connection stays in the database pool. The database pool will only remove a connection if the maximum lifetime elapses and the connection is no longer active. Setting this property to any value between 0 and 30000 will have no effect. Setting this property to 0 makes the maximum lifetime indefinite.
`maximumSize`	Specifies the maximum number of connections in a database pool.
`readOnly`	Specifies whether the database pools are read-only. Some database types do not support the read-only mode. If the database type does not support the read-only mode, the database pools will be read-write regardless of the value of this property.

3. To add a database driver other than PostgreSQL or Oracle, in the `allowedDatabaseDrivers` section of the new options file, add a new driver in the form of a key-value pair.

   The key must be in the format `.driver`, and the value must be the fully qualified Java class name of the driver.

   Example:

   ```
   postgresql.driver: "org.postgresql.Driver"
   oracle.driver: "oracle.jdbc.driver.OracleDriver"
   ```

4. Stop the Policy Editor.

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

5. Run **`setup`** using the `--optionsFile` argument and customize all other options as needed.

   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
   ```

6. Start the Policy Policy Editor.

Copyright © 2025 Ping Identity Corporation

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

Configuring Policy Editor policy request header mappings

You can configure the Policy Editor to enforce request header mappings during policy development and testing.

About this task

By defining a policy request header mapping, you can map a decision request header to a Trust Framework attribute. The Policy Editor uses this mapping to dynamically populate the attribute's value with the value of an incoming request header, enabling you to leverage header data as additional context in the request body.

You can configure policy request header mappings in embedded policy decision point (PDP) mode for production environments. Learn more in Configuring policy request header mappings.

Steps

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 `decisionRequestHeaderMapping` field.

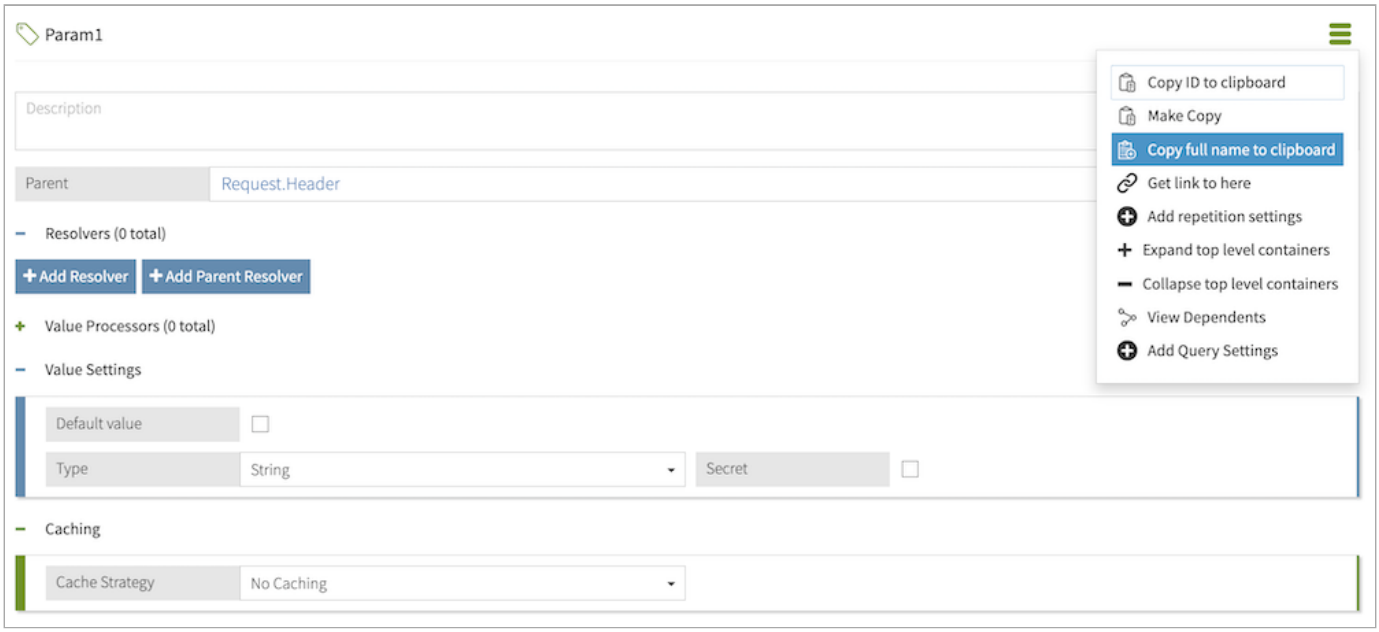
3. In the `headerKeys` field, enter a list of mappings in the form of key-value pairs, where the key is the header name and the value is the attribute's full name:

   ```
   decisionRequestHeaderMapping:
     enabled:true
     headerKeys:
       "x-correlation-id": "Request.Headers.x-correlation-id"
   ```

   Tip

   The full name of an attribute represents that attribute's full path in the Trust Framework hierarchy. For example, a **Param1** attribute with parent attributes **Request** and **Header** would have a full name of **Request.Header.Param1**. To quickly obtain an attribute's full name, in the attribute's hamburger menu , select **Copy full name to clipboard.**

Copyright © 2025 Ping Identity Corporation



> **Note**
>
> The header name is not case sensitive.

4. Stop the Policy Editor.
5. Run `setup` using the `--optionsFile` argument and customize all other options as appropriate:

```
$ 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
```

6. 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.

Copyright © 2025 Ping Identity Corporation

> **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.

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`.

Copyright © 2025 Ping Identity Corporation

```
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.

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`**

Copyright © 2025 Ping Identity Corporation

- `--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.

Steps

1. Stop the Policy Editor.

Copyright © 2025 Ping Identity Corporation

```
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 the Policy Editor using Docker.

> **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.

Copyright © 2025 Ping Identity Corporation

```
$ 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/{PAP_CONTAINER_NAME}
```

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.

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 the Policy Editor using Docker, 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/{PAP_CONTAINER_NAME}:<TAG>
```

Copyright © 2025 Ping Identity Corporation

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. Learn more in 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:

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

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:

Copyright © 2025 Ping Identity Corporation

```
$ 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:

Copyright © 2025 Ping Identity Corporation

```
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`**.

Enabling JSON formatting for Policy Editor logs

Enable JSON-formatted data for default Policy Editor loggers.

About this task

You can add the `dropwizard-json-logging` library in the `PingAuthorize-PAP/config/configuration.yml` file for each logger you wish to enable JSON formatting for. The availability of this library does not impact the Policy Editor's default configuration.

Steps

1. Stop the Policy Editor.

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

2. In the `PingAuthorize-PAP/config/configuration.yml` file, add the following to the `appenders` section of each logger for which you want to enable JSON formatting:

   ```
   layout:
     type: json
   ```

   Example:

   To enable JSON-formatted data in `management-audit.log`:

   ```
   ADMIN_POINT_AUDIT:
     level: "INFO"
     additive: false
     appenders:
     - type: "file"
       layout:
         type: json
       currentLogFilename: "logs/management-audit.log"
       archive: false
       archivedLogFilenamePattern: "logs/management-audit.%i.log.gz"
       archivedFileCount: 10
       logFormat: "[%date{ISO8601}] %msg%n"
   ```


Copyright © 2025 Ping Identity Corporation

3. To enable JSON formatting for all HTTP request log messages, add the following to the `server` section of the `PingAuthorize-PAP/config/configuration.yml` file:

```
requestLog:
  appenders:
    - type: console
      layout:
        type: access-json
```

4. Save your changes and restart the Policy Editor.

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

Enabling Mapped Diagnostic Context for Policy Editor logs

You can enable Mapped Diagnostic Context (MDC) in the default Policy Editor logs for enhanced visibility of application behavior.

About this task

MDC is a map of key-value pairs that you can use to enrich log messages. This can help when correlating log messages to a specific decision request. For example, if you want to correlate an HTTP service call recorded in the application logs to the decision request that invoked that service, you can use the `decisionRequestId` generated by MDC.

> **Note**
>
> MDC log data is enabled by default in the File-based Trace Log Publisher in embedded policy decision point (PDP) mode.

The following table outlines the MDC keys available for each decision request type:

Decision request type	Key	Description
Individual requests	`decisionRequestId`	Unique ID for the decision request.
Batch requests	`batchId`	Unique ID for the batch request.
	`decisionRequestID`	Unique ID for an individual decision request in the batch.
Policy query requests	`queryId`	Unique ID for the policy query request.
	`decisionRequestID`	Unique ID for an individual decision request in the query request.

Steps

1. Stop the Policy Editor.

Copyright © 2025 Ping Identity Corporation

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

2. In the `PingAuthorize-PAP/config/configuration.yml` file, add the `%mdc` configuration option to the `logFormat` field of the application log `appenders` section:

```
appenders:
- type: "console"
  threshold: "INFO"
  target: "stdout"
  logFormat: "%-5level [%date{ISO8601}] %c, %mdc: %msg%n"
- type: "file"
  threshold: "DEBUG"
  currentLogFilename: "logs/debug.log"
  maxFileSize: "250MB"
  archive: true
  archivedLogFilenamePattern: "logs/debug.%i.log.gz"
  archivedFileCount: 10
  logFormat: "%-5level [%date{ISO8601}] [%thread] %c, %mdc: %msg%n"
- type: "file"
  threshold: "INFO"
  currentLogFilename: "logs/authorize-pe.log"
  maxFileSize: "250MB"
  archive: true
  archivedLogFilenamePattern: "logs/authorize-pe.%i.log.gz"
  archivedFileCount: 10
  logFormat: "%-5level [%date{ISO8601}] %c,%mdc: %msg%n"
```

3. Save your changes and restart the Policy Editor.

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

Configuring policy query debug logging in the Policy Editor

You can configure the granularity of policy query audit log entries when developing and testing policy queries in external policy decision point (PDP) mode.

Policy queries enable you to pose open-ended authorization questions to the JSON PDP API with the `/query` endpoint. For example, instead of being limited to questions like "Can John Smith edit account1?" you can ask "Which accounts can John Smith edit?" or "Which actions can John Smith perform on account1?"

When testing query requests, you can enrich the level of detail in the policy query audit log for enhanced debugging capabilities.

> **Note**
>
> By default, the policy query audit log file is located at `PingAuthorize-PAP/logs/query-audit.log`.

In debug mode, a policy query audit log entry includes the following fields:

- `requestId`: A unique identifier for the query request

Copyright © 2025 Ping Identity Corporation

- `permutationId`: A unique identifier for the query permutation

  A query permutation is a combination of query attributes used for a decision in the final query response. Use this identifier and the `requestId` for increased visibility of query request information across your logging system. For example, a logged call to an external information point would include identifiers for the request and permutation that invoked that service.

- `permutation`: A query permutation as an array of JSON objects containing each query attribute and its value

- `response`: The complete, high-verbosity response for a query permutation's associated decision, including expanded errors and other helpful information

  By default, this field includes details about the resolution and policy dependencies of each attribute involved in the permutation's corresponding decision, along with details about any external service used in that decision.

> **Note**
>
> You can increase the `response` field's level of detail by controlling the query permutation view. Learn more in Configuring the query permutation view.

Enabling debug logging

Steps

1. Stop the Policy Editor.

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

2. In the `PingAuthorize-PAP/config/configuration.yml` file, change the policy query audit log level to `DEBUG`:

```
logging:
  level: "INFO"
  loggers:
    QUERY_AUDIT_LOG:
      level: "DEBUG"
```

3. Save your changes and restart the Policy Editor.

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

You can now view resolution and policy dependency information for each possible combination of the attributes included in the query request, as well as each combination's corresponding decision response.

Configuring the query permutation view

In addition to enabling query permutations in the policy query audit log, you can specify additional levels of detail to include in each permutation's `response` field. The policy query audit log provides the following additional views:

- `request`: Includes the decision request object for each query permutation

Copyright © 2025 Ping Identity Corporation

- `decisionTree`: Includes details of the policy tree's evaluation flow
- `attributes`: Includes details of attributes used during policy evaluation, including the attribute's value and type

> **Note**
>
> Specifying this view overrides any logged attributes configuration.

- `services`: Includes details of services invoked during policy evaluation

Steps

1. Stop the Policy Editor.

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

2. In the `PingAuthorize-PAP/config/configuration.yml` file, add a new top-level `auditLogging` section, and specify additional query permutation views with the `queryDebugLogView` configuration property:

```
auditLogging:
  queryDebugLogView: attributes, services, decisionTree
```

3. Save your changes and restart the Policy Editor.

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

Configuring logged attributes

With the `loggedAttributes` configuration property, you can exercise control over which attributes get logged as part of the decision response. This includes details about the attribute's value and type.

Steps

1. Stop the Policy Editor.

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

2. In the `PingAuthorize-PAP/config/configuration.yml` file, add a new top-level `loggedAttributes` configuration property, and provide a list of full names of the attributes you want logged:

```
loggedAttributes:
  - name: <attribute1>
  - name: <attribute2>
```


Copyright © 2025 Ping Identity Corporation

> **Note**
>
> The `name` key is case-sensitive.

3. Save your changes and restart the Policy Editor.

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

> **Note**
>
> Attributes specified using this configuration property are only logged if they get evaluated as part of the decision request. Enabling the `attributes` query permutation view will override the `loggedattributes` configuration and log all evaluated attributes.

You can also configure policy query logging for decisions executed in embedded PDP mode. Learn more in Policy query logging.

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 statements 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 statement could cause PingAuthorize Server to remove specific attributes from an API response because the requesting user lacks a necessary entitlement.

Copyright © 2025 Ping Identity Corporation

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
- Sideband API policy requests
- SCIM policy requests

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.

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:{TRUST_FRAMEWORK_VERSION}
```

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.

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.

Copyright © 2025 Ping Identity Corporation

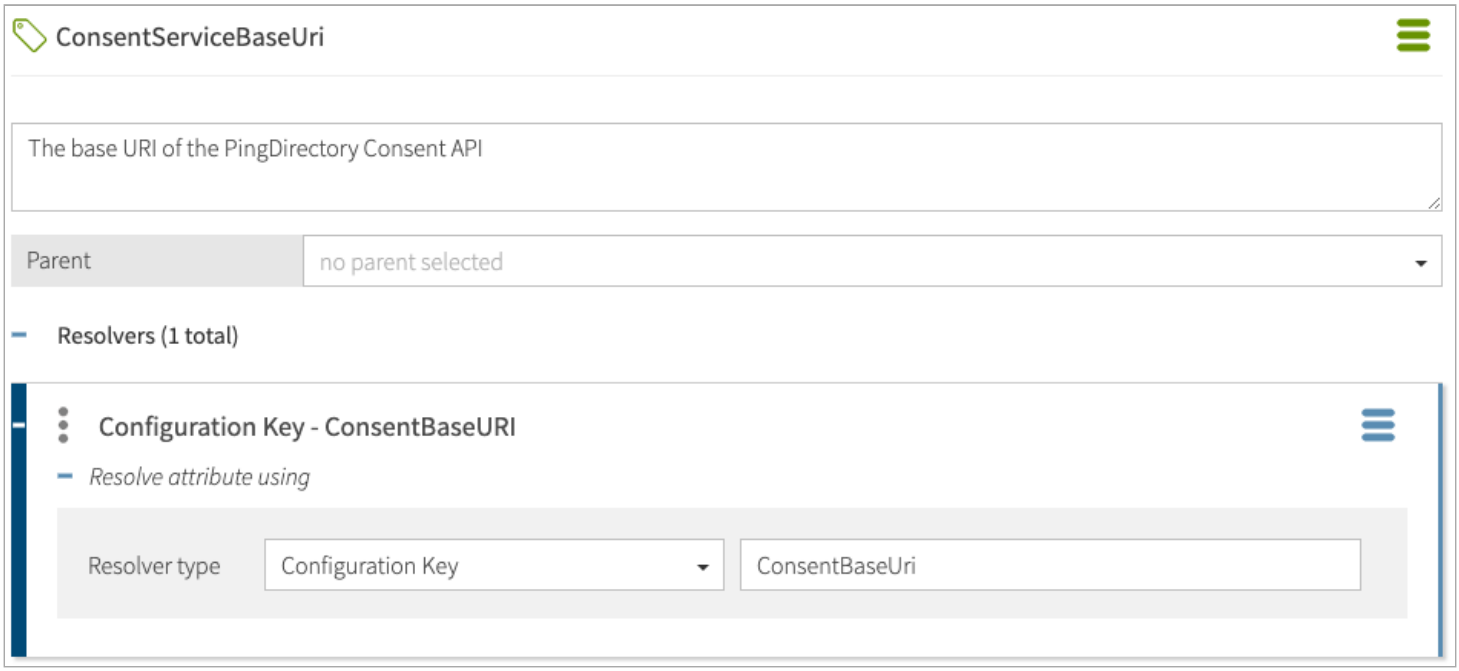
For example, you might use a Trust Framework service to retrieve a user's consent from the PingDirectory Consent API. This service depends on the URL of the Consent API, the username and password that are used for authentication, and other items that vary between development, preproduction, and production 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. Learn more in Define policy configuration keys in a development environment.
Embedded PDP mode	The PingAuthorize Server configuration. Learn more in Define policy configuration keys in a preproduction environment.

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.

Copyright © 2025 Ping Identity Corporation

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:

`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

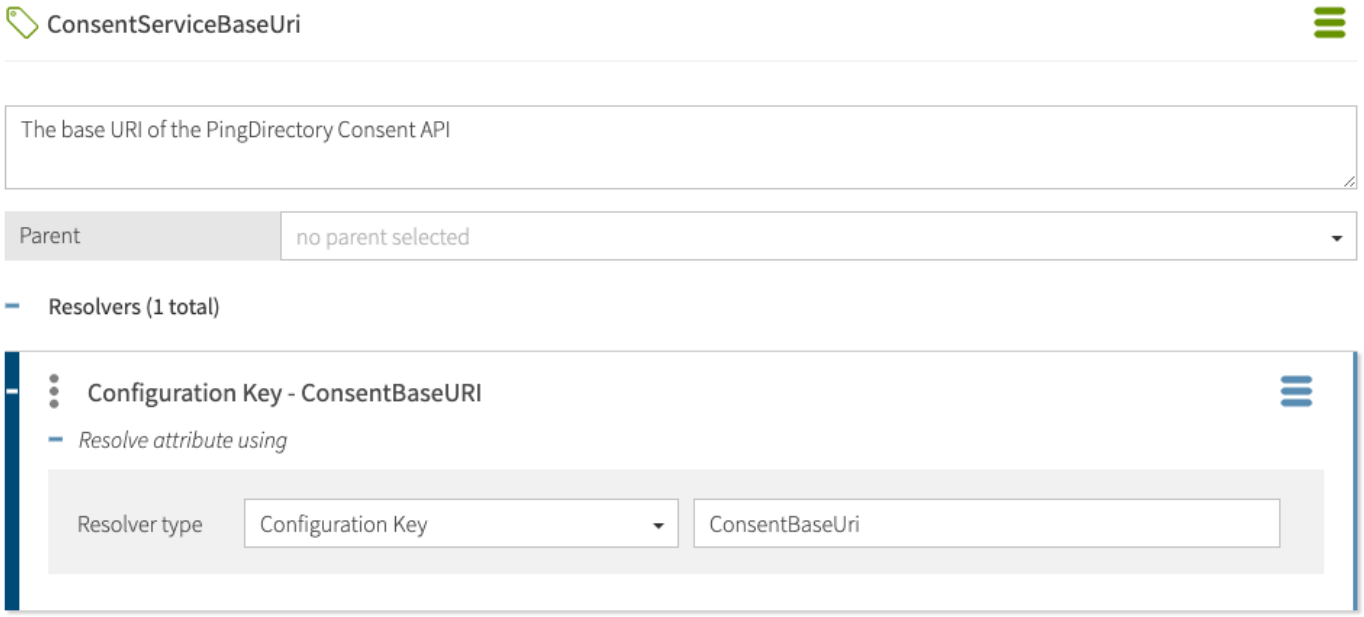
Complete the following steps to define the policy information provider (PIP).

Steps

1. Define an attribute in the Trust Framework for the Consent API's base HTTPS URL.
    1. Go to **Trust Framework** and then click **Attributes**.
    2. 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.

Copyright © 2025 Ping Identity Corporation

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.

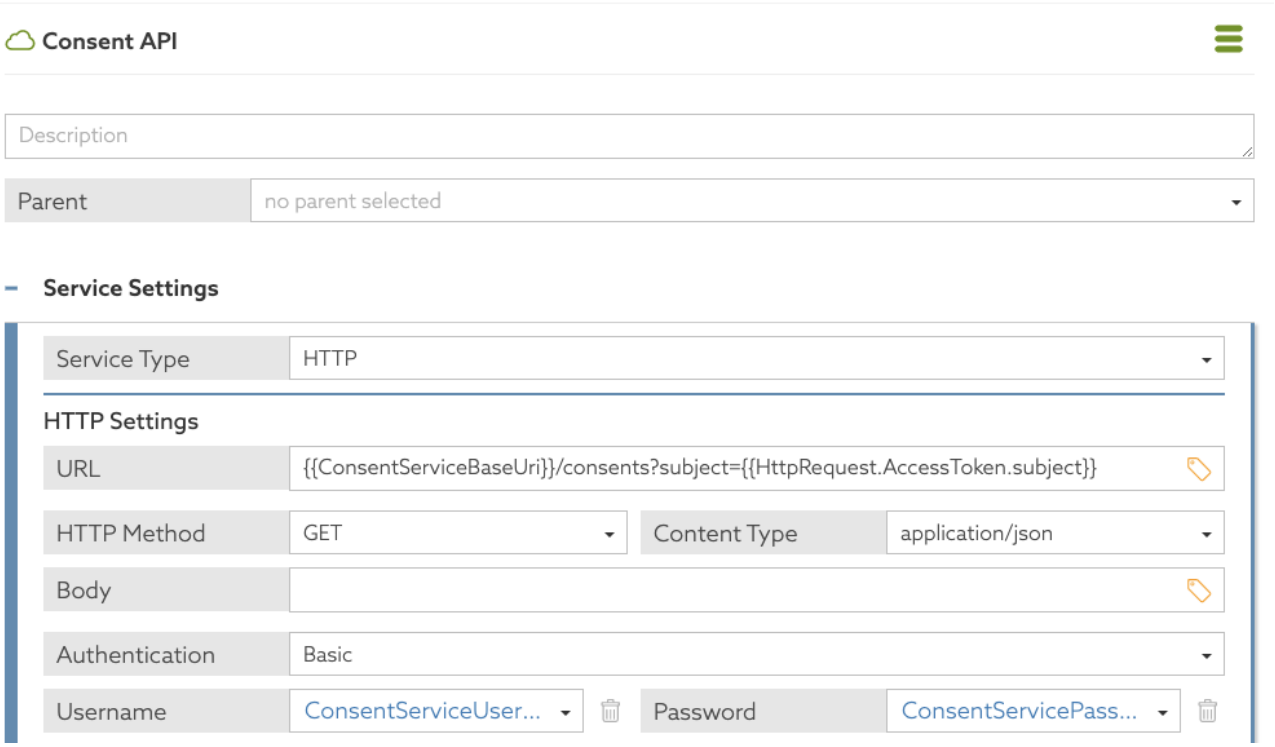
3. Define the policy information provider using the attributes that you just defined.
   1. Go to **Trust Framework** and then **Services**.
   2. Add a new service.
      1. Name the service Consent API.
      2. Leave the **Parent** value blank. If a value is already present, clear it.
      3. Set **Service Type** to HTTP.
      4. Set the **URL** to \{{ConsentServiceBaseUri}}/consents?subject=\{\{HttpRequest.AccessToken.subject}}.

Copyright © 2025 Ping Identity Corporation

5. Set **Authentication** to **Basic**.
6. For **Username**, select the attribute `ConsentServiceUsername`.
7. For **Password**, select the attribute `ConsentServicePassword`.

3. 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.

> **Note**
>
> See Policy Editor configuration with runtime environment variables for an example of defining policy configuration keys at runtime.

Steps

1. Make a copy of the default options file.

Copyright © 2025 Ping Identity Corporation

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

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. You can use either the administrative console or **`dsconfig`** to define policy configuration keys.

Copyright © 2025 Ping Identity Corporation

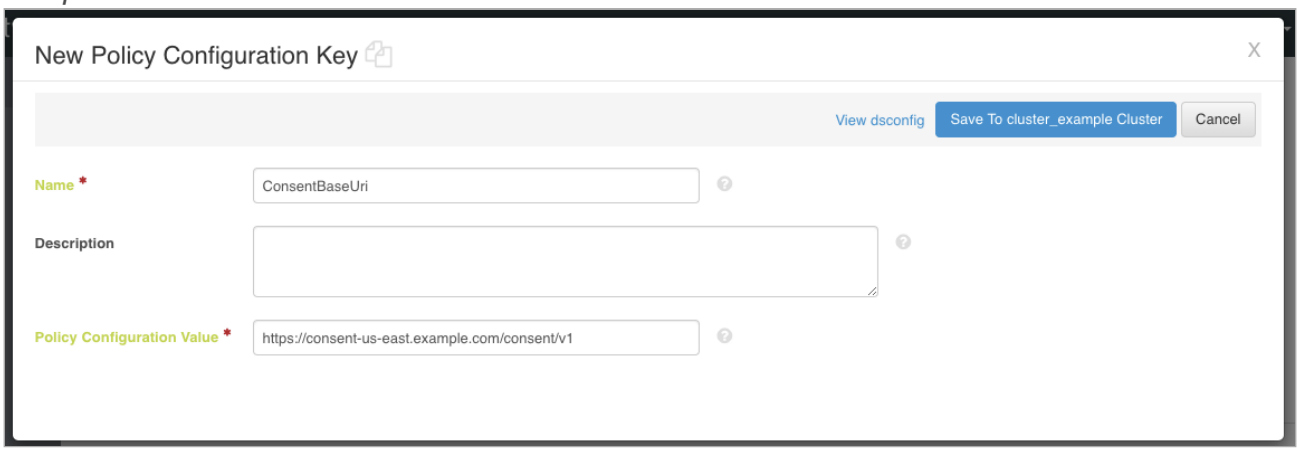
> **Note**
>
> Policy configuration key values are stored in encrypted form in the PingAuthorize Server configuration, so they are suitable for storing sensitive values such as server credentials.

Define policy configuration keys using the administrative console by following these steps:

Steps

1. In the administrative console, under **Authorization and Policies**, click **Policy Decision Service**.
2. Click **New Policy Configuration Key**.
    1. For **Name**, enter `ConsentBaseUri`.
    2. For **Policy Configuration Value**, type the base URI. For example, `https://consent-us-east.example.com/consent/v1`.

    Example:

3. Save the policy configuration key.
4. Repeat the previous steps for the policy configuration keys `ConsentUsername` and `ConsentPassword`.

Example

The following example shows how to use `dsconfig` 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
```

Create policies in a development environment

Develop policies using the Policy Editor as the policy decision point (PDP), which causes all saved policy changes to take effect immediately.

Copyright © 2025 Ping Identity Corporation

To take advantage of rapid deployment and advanced testing capabilities during policy development, configure PingAuthorize Server in external PDP mode. In this mode, PingAuthorize Server forwards all policy requests to the Policy Editor, which acts as PingAuthorize Server's PDP.

To get started with policy development, refer to Getting started with PingAuthorize (tutorials) or Loading a policy snapshot.

> **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.

After developing your policies, deploy them using one of the following methods:

- Exporting a policy deployment package
- Using the Deployment Manager

Configuring external PDP mode

You can use the administrative console or `dsconfig` to configure external PDP mode. To prepare the Policy Editor for making authorization decisions, you must create a Policy External Server to represent the Policy Editor, assign the Policy External Server to the Policy Decision Service, and set the PDP mode to external.

> **Important**
>
> To send a request in external PDP mode, the major versions of the PingAuthorize Server and the Policy Editor must match, and the Policy Editor's minor version must be greater than or equal to that of the PingAuthorize Server. For example:
>
> - If the PingAuthorize Server version is 10.1 and the Policy Editor version is 10.2, the request succeeds.
> - If the PingAuthorize Server version is 10.2 and the Policy Editor version is 10.1, the request fails.

Before you begin

You need the following values to configure the PingAuthorize Server to use external PDP mode:

- The shared secret, which is specified or automatically generated when you install the Policy Editor.

  To obtain the shared secret after installation, copy the `core.Authentication.SharedSecret` value from the `PingAuthorize-PAP/config/configuration.yml` file.

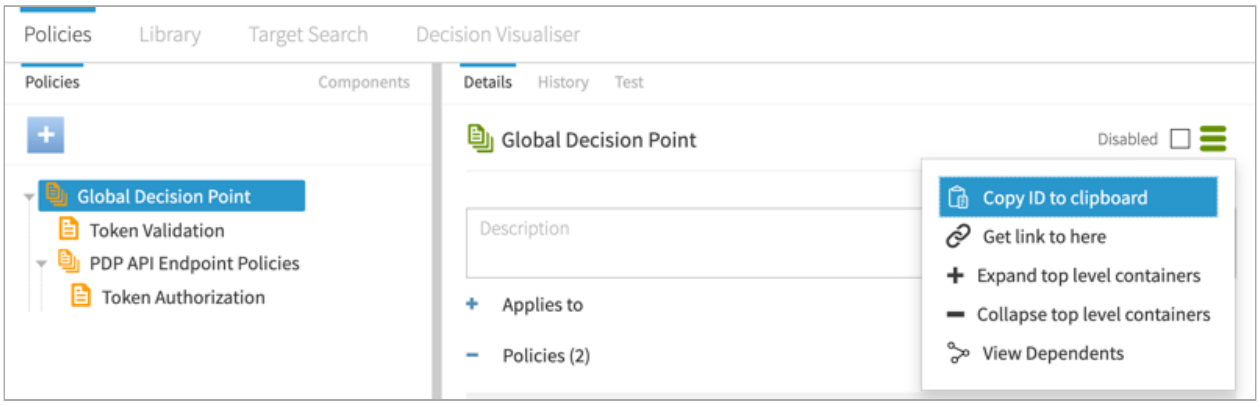
- The branch name, which corresponds to the policy branch you want to evaluate requests against in the Policy Editor.
- The decision node, which is the ID of the policy tree node that is evaluated first during policy processing. To get the decision node ID:

  1. In the Policy Editor, go to **Policies**.
  2. In the policy tree, select the node that you want to use as the root node.

     This is typically the top-level node of your policy tree.

Copyright © 2025 Ping Identity Corporation

3. Click the hamburger menu and select **Copy ID to clipboard**.



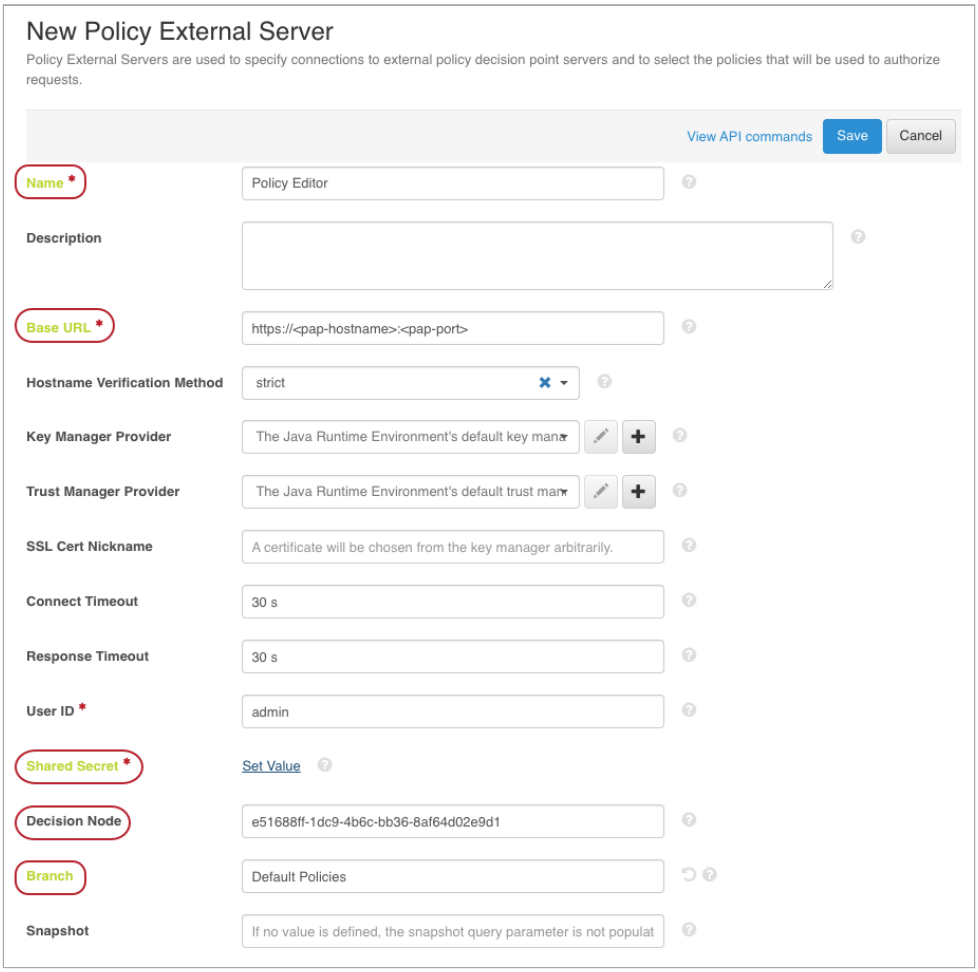
Copyright © 2025 Ping Identity Corporation

Admin console

Configuring external PDP mode using the administrative console

Steps

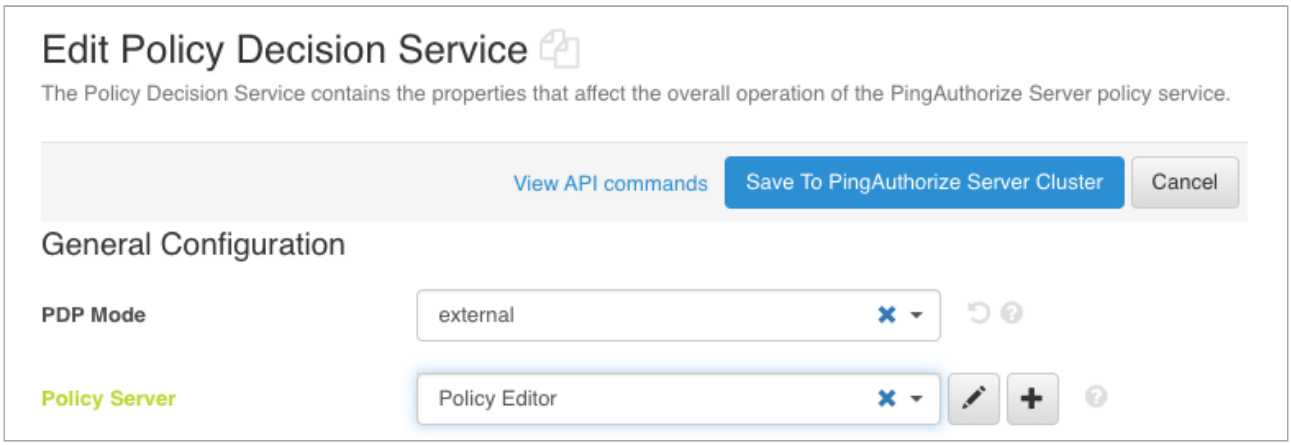
1. In the PingAuthorize administrative console, go to **Configuration > Data Sources > External Servers**.
2. Click **New External Server** and, in the list, select **Policy External Server**.
3. In the **New Policy External Server** window, specify the following information:
   - **Name**
   - **Base 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**.

Copyright © 2025 Ping Identity Corporation

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**.

dsconfig

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 "{PAP_Name}" \
  --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:{PAP_Name}"
```

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.

Copyright © 2025 Ping Identity Corporation

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).
`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).

Importing and exporting policies

PingAuthorize supports two import and export file formats for Trust Framework and policy data.

Copyright © 2025 Ping Identity Corporation

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. Use a snapshot to load data into the Policy Editor for policy development using external policy decision point (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. Use a deployment package to load data into the PingAuthorize Server for policy deployment using embedded PDP mode.

The following sections describe how to import and export policy snapshots using the Policy Editor. See Exporting a policy deployment package and Using the Deployment Manager for more information on deployment packages.

Loading a policy snapshot

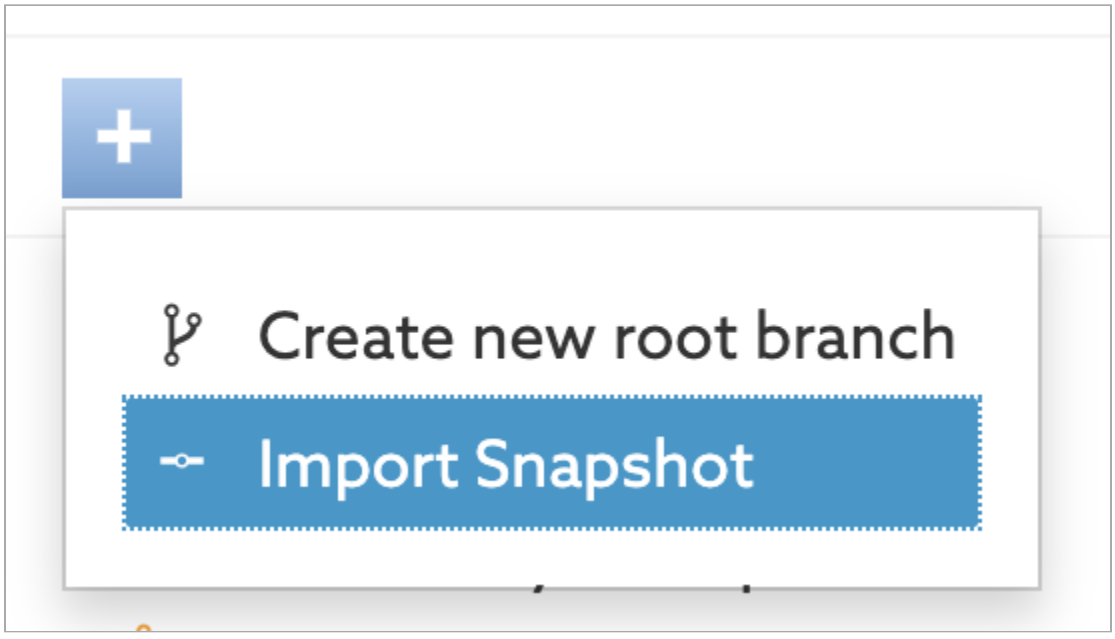
To import a policy snapshot into the Policy Editor for policy development, complete the following steps.

About this task

To create a new policy branch with the Trust Framework and policies of the provided snapshot:

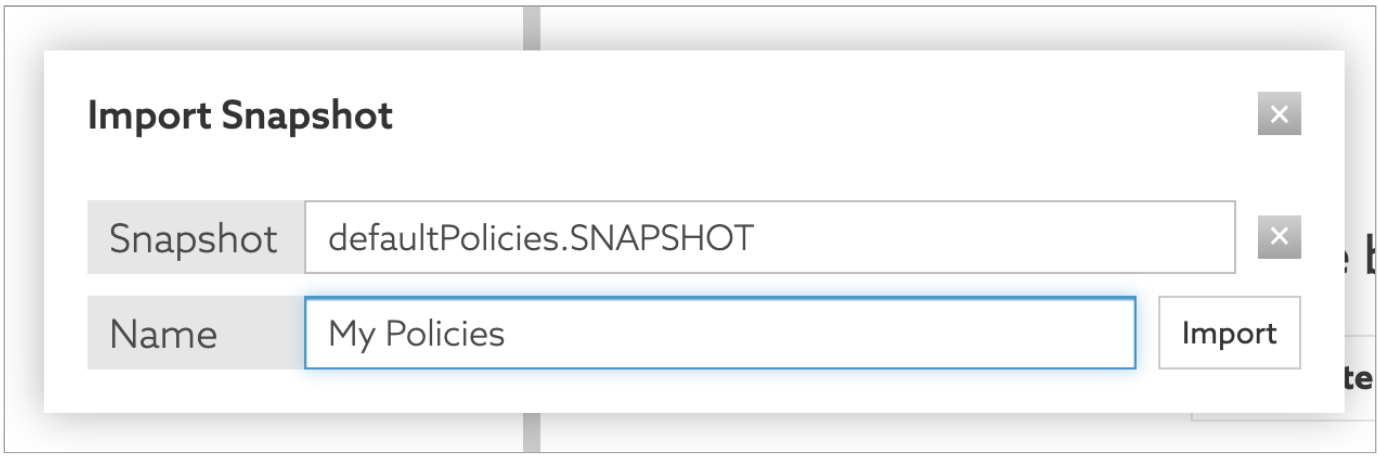
Steps

1. Go to the **Branch Manager** section.
2. Click the **Version Control** tab.
3. In the **+** menu, select **Import Snapshot**.



Copyright © 2025 Ping Identity Corporation

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

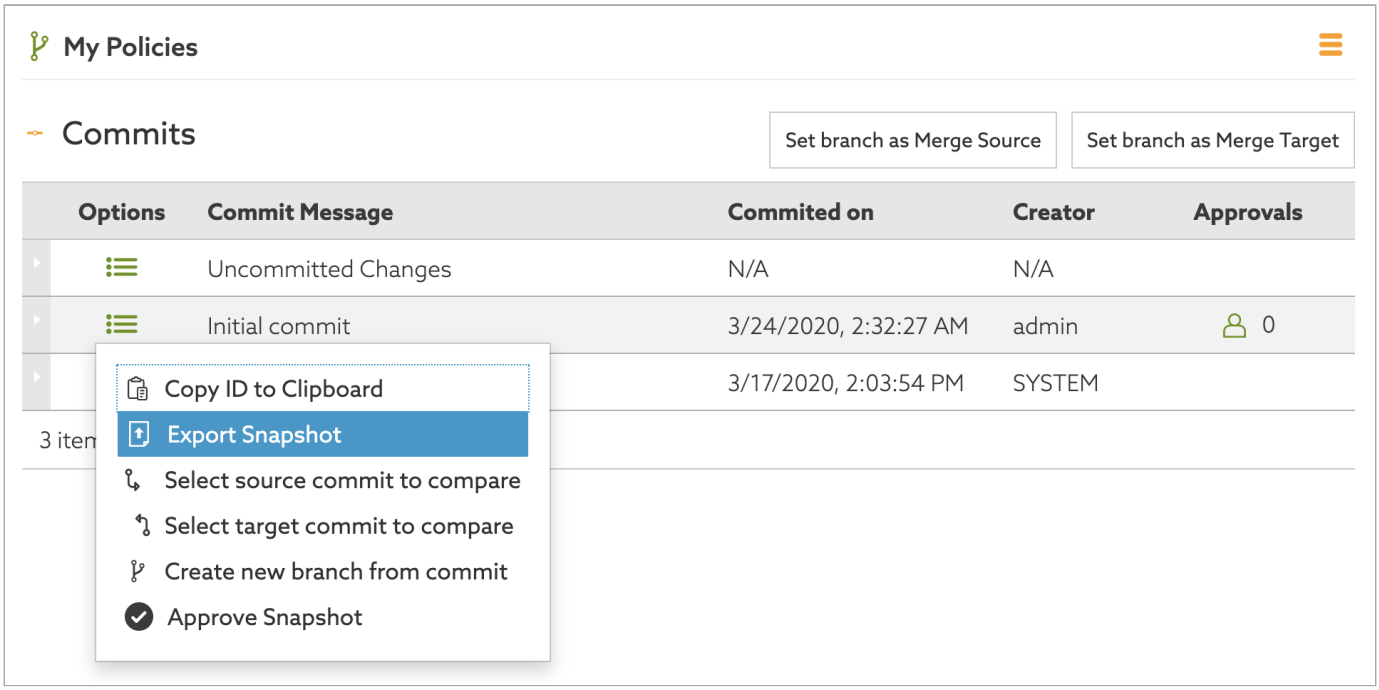
About this task

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**.

Copyright © 2025 Ping Identity Corporation



5. Provide a snapshot filename and click **Export**.

Result

The snapshot file is downloaded to your computer.

Deploy policies in a production environment

After developing and testing policies in external policy decision point (PDP) mode, you should configure PingAuthorize Server in embedded PDP mode for other pre-production or production environments.

Embedded PDP mode is much more performant than external PDP mode when making authorization decisions. This improvement in performance happens because in embedded PDP mode, the PingAuthorize Server doesn't make an additional call to the Policy Editor to retrieve the authorization decision.

In embedded PDP mode, the PingAuthorize Server's decision engine uses a file called a deployment package to handle all decision requests. A deployment package includes the set of policies and Trust Framework elements evaluated by the decision engine when making decisions. You can load the deployment package into the server in two ways:

- Export a deployment package from the Policy Editor and load it into the decision engine. Learn more in Exporting a policy deployment package.
- Export the deployment package to a central deployment package store, which is in turn polled by the decision engine at a configurable interval. Learn more in Using the Deployment Manager.

> **Note**
>
> If you expect policies to change in production, use the Deployment Manager instead of exporting deployment packages manually.

Copyright © 2025 Ping Identity Corporation

Exporting a policy 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.

Before you begin

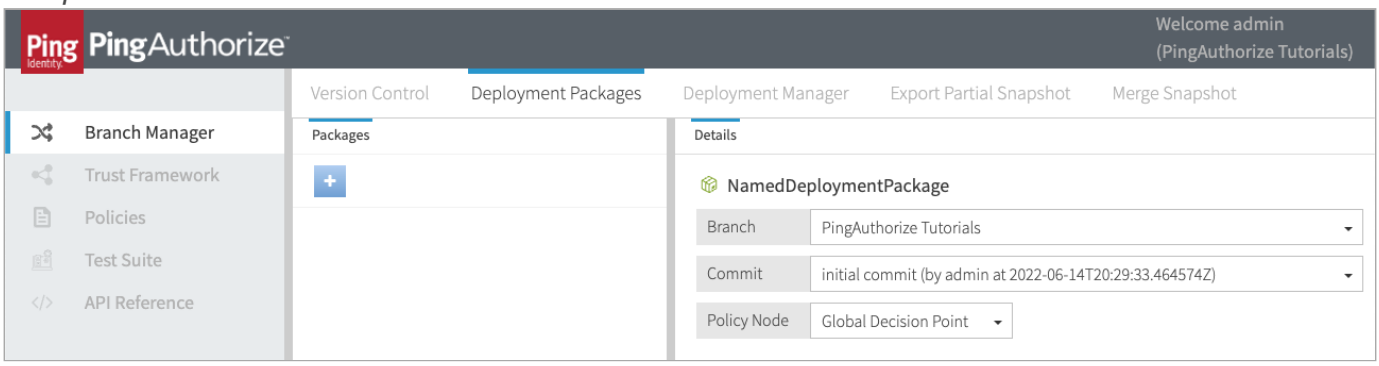
If you want to configure your policy deployment packages to be signed upon export, see Policy application management with signed deployment packages.

Steps

1. Export a snapshot.

   Learn more in Exporting a policy snapshot.
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.

Copyright © 2025 Ping Identity Corporation

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.

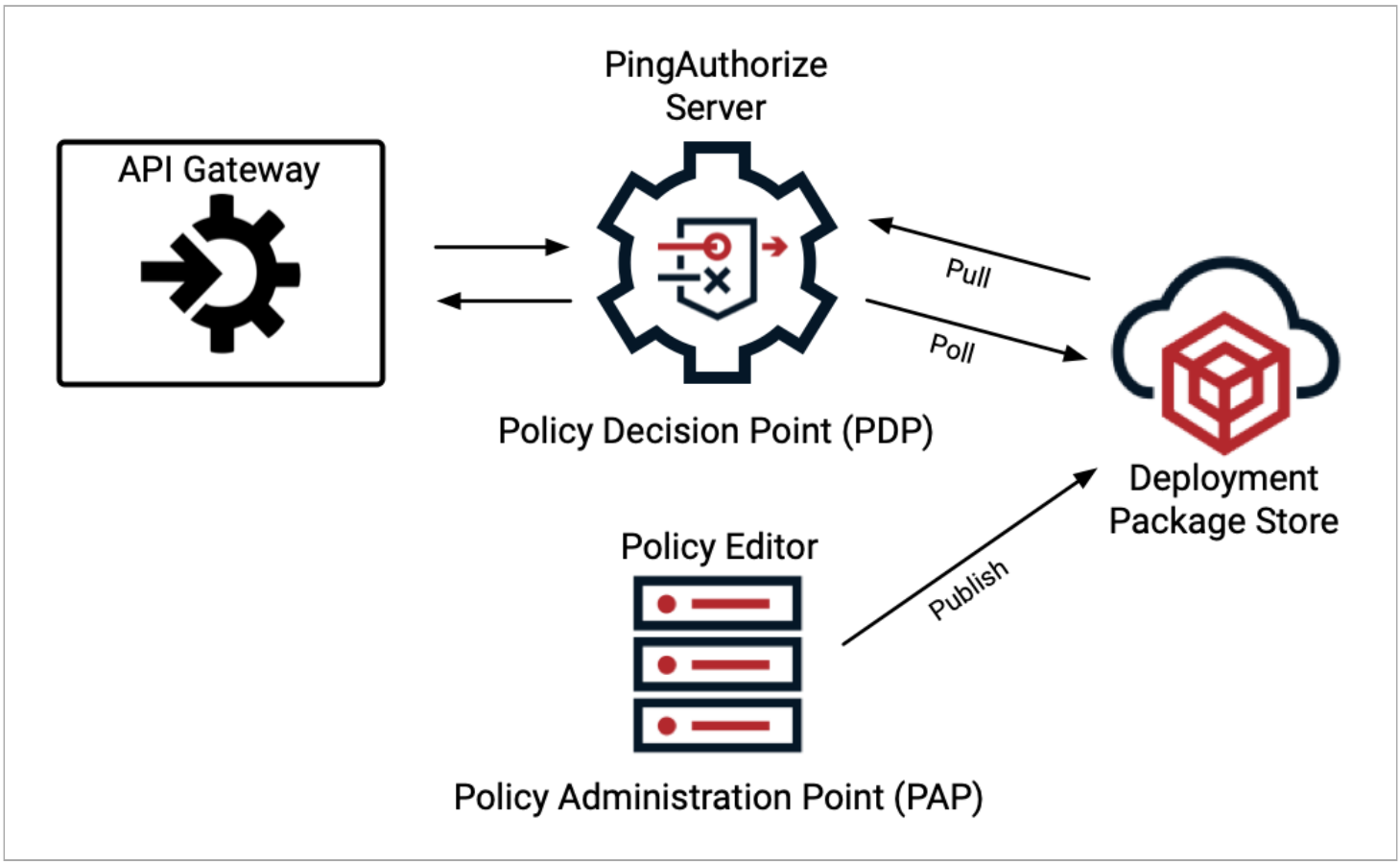
> **Note**
>
> You configure the interval that the server checks for updates in the store during setup.

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, as illustrated in the following diagram:



To use the Deployment Manager:

Copyright © 2025 Ping Identity Corporation

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. See Setting up an Amazon S3 deployment package store for more information.
> - 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:
    1. Add a filesystem deployment package store.
    2. Add an Amazon S3 deployment package store.
    3. Add an Azure deployment package store,
5. Configure the Policy Decision Service to read from your deployment package store.

Setting up an Amazon S3 deployment package store

Follow these procedures to create and configure an Amazon Simple Storage Service (S3) bucket that you can publish deployment packages to from the Policy Editor for consumption by PingAuthorize Server.

> **Important**
>
> See Using the Deployment Manager for the steps you need to complete for an end-to-end deployment package store configuration in embedded policy decision point (PDP) mode.

Creating an S3 bucket

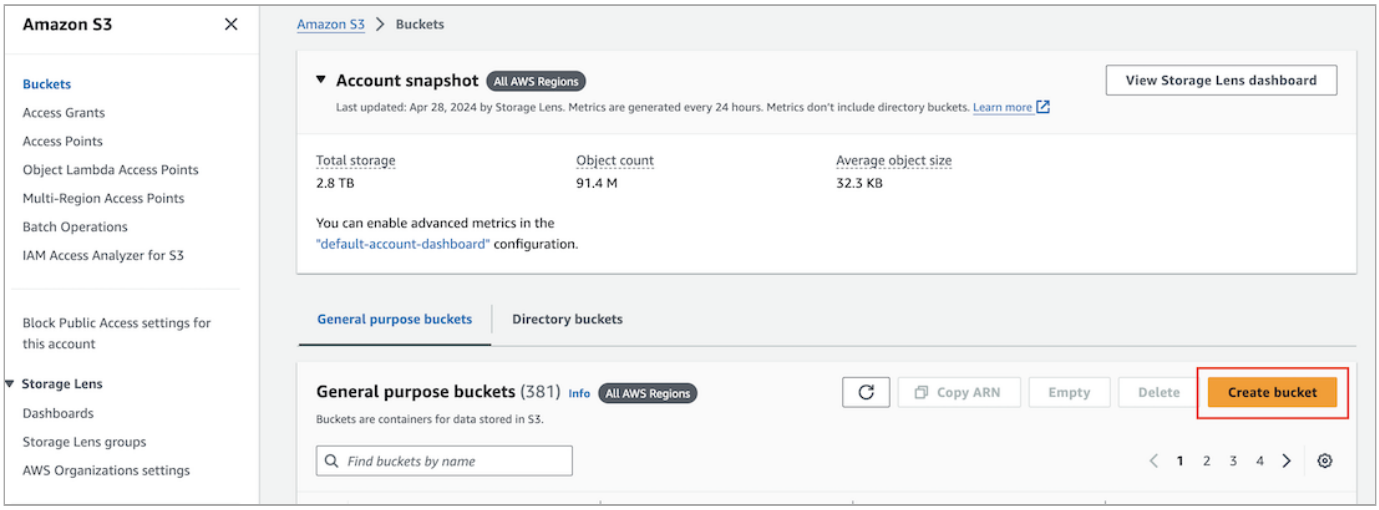
About this task

To store your deployment packages, create a new S3 bucket in the S3 dashboard of Amazon Web Services (AWS) Management Console.

Steps

1. In the AWS Management Console, on the S3 dashboard, go to **Buckets** and click **Create bucket**.

Copyright © 2025 Ping Identity Corporation



2. Configure the bucket.

   Learn more in Creating a bucket in the Amazon documentation.

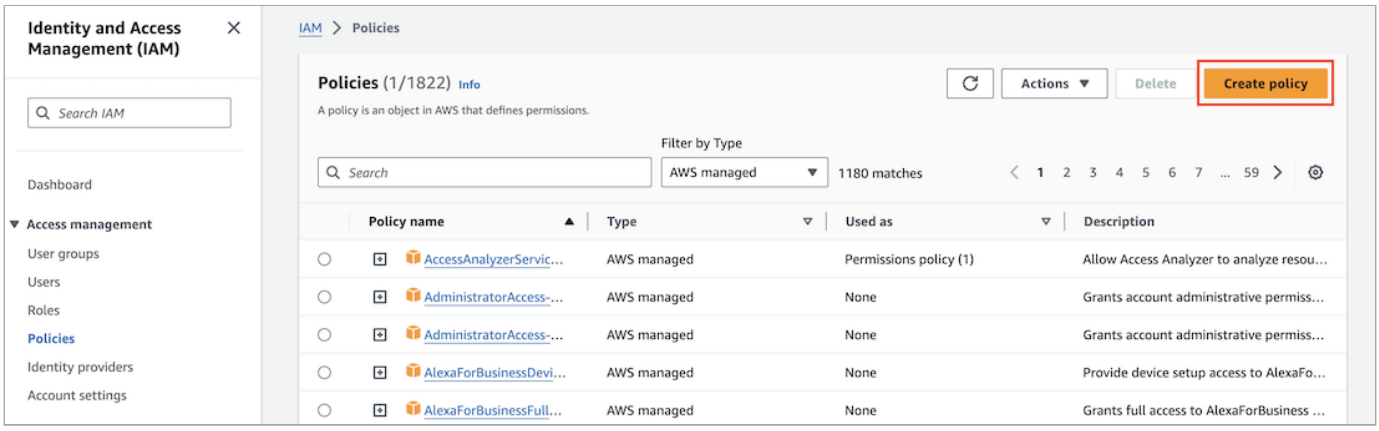
Configuring the IAM user policy

About this task

To manage your deployment package store, configure a new identity and access management (IAM) policy in the IAM dashboard of AWS Management Console.

Steps

1. In the AWS Management Console, on the IAM dashboard, go to **Access Management → Policies** and click **Create policy**.



2. In the **Policy editor** wizard, select the **JSON** tab and enter these permissions:

   Example:

Copyright © 2025 Ping Identity Corporation

```
{
    "Version": "<policy_creation_date>",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> You can manage S3 bucket access in multiple ways, including IAM policies, S3 policies, or S3 access control lists (ACLs). We provide a working IAM policy example, but you should be aware of current AWS best practices, industry best practices, and your organization's conventions when configuring S3 bucket access.

Replace the asterisk wildcard character to restrict the access scope of the previous policy.

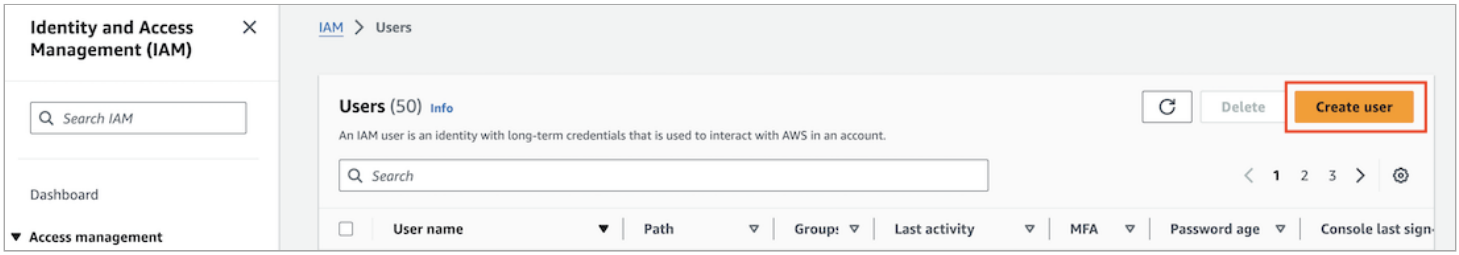
3. Complete the **Create policy** wizard according to your organization's specifications.
4. On the **Review and create** confirmation page, review your configurations and click **Create policy** to create the IAM user policy.

Configuring the IAM user

To publish to your AWS deployment package store from the Policy Editor, configure a new IAM user to access the S3 bucket in accordance with the policy that you created in Configuring the IAM user policy.

Steps

1. In the AWS Management Console, on to the IAM dashboard, go to **Access Management > Users** and click **Create user**.



1. Define a username and click **Next**.

Copyright © 2025 Ping Identity Corporation

2. On the **Set permissions** page, select **Attach policies directly**, and then select the policy you created in Configuring the IAM user policy.
    1. (Optional) Select **Create policy** to configure a different user policy.
3. On the **Review and create** confirmation page, review your configurations and click **Create user** to create the IAM user.
4. (Optional) Attach user attribute tags.
5. On the **Users** page, search for and click on the newly created user.
6. On the **Security credentials** tab, in the **Access keys** section, click **Create access key**.
7. Click **Application running outside AWS**. Click **Next**.
8. (Optional) In the **Description tag value** field, enter a description for the access key and click **Create access key**.
9. Copy both the **Access key ID** and **Secret access key** values to a secure location. Click **Done**.

    These values will be used when configuring connections to the Policy Editor and PingAuthorize server.

    > **Important**
    >
    > You cannot recover these credentials at a later time.

Configuring the Policy Editor to publish to a deployment package store

Use an `options.yml` file to configure the Policy Editor.

Before you begin

If you want to configure your policy deployment packages to be signed upon publication to a store, refer to Policy application management with signed deployment packages.

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.

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.

    1. Duplicate the desired deployment package store type, uncomment, and modify its values according to your deployment.

Copyright © 2025 Ping Identity Corporation

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.
- For an Amazon Web Services (AWS) deployment package store, review your existing Simple Storage Service (S3) bucket configurations on the S3 dashboard of AWS Management Console.
- Each deployment package store has its own signing key configuration under `deploymentPackageStores`. See the `Signed filesystem store` block for an example. The signing key configuration under `deploymentPackageData` applies only to exported deployment packages—not deployment package stores.

```
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
  #   keystore:
  #     resource: /path/to/deployment-package-signing-keystore.jks
  #     password: keystore-password
  #   signingKey:
  #     alias: signing-cert-alias
  #     password: private-key-password
  #   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...
```

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> For information on properly formatting an S3 endpoint or Azure connection string, refer to S3 endpoints or Storage connection strings, respectively.

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**.

Publishing 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.yml` file.

Learn more in Configuring the Policy Editor to publish to a deployment package store.

About this task

To publish deployment packages to a deployment package store:

Steps

1. Export a snapshot.

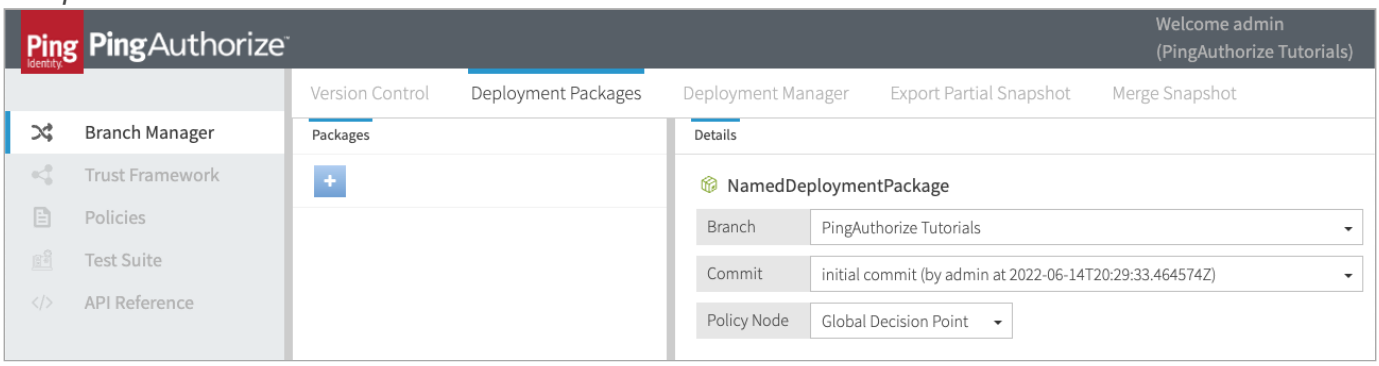
   Learn more in Exporting a policy snapshot.

2. Go to **Branch Manager > Deployment Packages**.

Copyright © 2025 Ping Identity Corporation

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**.

Result

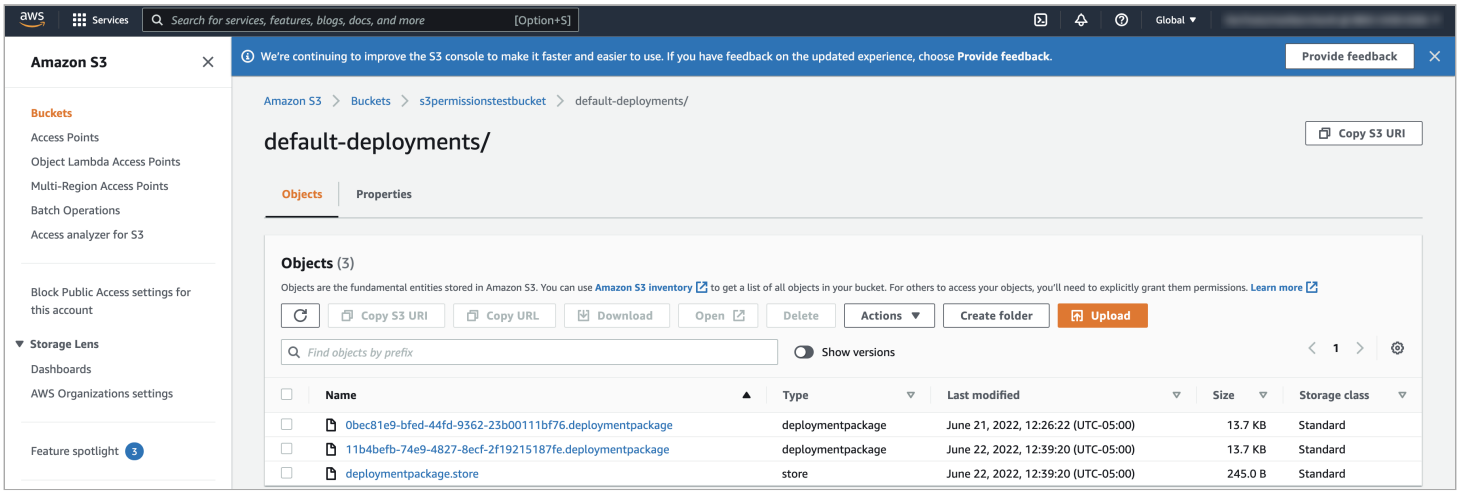
Your deployment package is visible in your configured deployment package store.

Example

If you configured an Amazon Simple Storage Service (S3) bucket for your deployment package store, your published deployment package objects should be visible on the **Objects** tab of your bucket in Amazon Web Services (AWS) Management Console.

> **Note**
>
> Depending on your configured prefix, AWS may have produced an object hierarchy. If so, drill into the hierarchy to locate your published deployment package.

Copyright © 2025 Ping Identity Corporation



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.

Adding a filesystem deployment package store

To use the Deployment Manager, add a deployment package store for read access to the PingAuthorize server.

Use the administrative console or `dsconfig` to add the deployment package store.

Copyright © 2025 Ping Identity Corporation

Administrative console

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:
    1. In the **Name** field, enter a name for the deployment package store.
    2. 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.

    3. 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.

Copyright © 2025 Ping Identity Corporation

Dsconfig

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 PingAuthorize

To use Amazon Simple Storage Service (S3) as your deployment package store, add read access for your S3 bucket to the PingAuthorize server.

Before you begin

You must create an access key and accompanying secret key for your S3 bucket. See Setting up an Amazon S3 deployment package store for more information.

About this task

Use the administrative console or `dsconfig` to add the Amazon S3 deployment package store. If needed, review your existing S3 bucket configurations on the S3 dashboard of Amazon Web Services (AWS) Management Console.

Copyright © 2025 Ping Identity Corporation

Administrative console

Adding an Amazon S3 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** menu, select **S3 Deployment Package Store**.
4. Complete the **General Configuration** fields:
    1. In the **Name** field, enter a name for the deployment package store.
    2. 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.

    3. In the **S3 Bucket Name** field, enter the name of your Amazon S3 bucket as shown on your AWS services page.
    4. In the **S3 Bucket Prefix** field, enter your Amazon S3 bucket prefix.
    5. In the **S3 Server Endpoint** field, enter your Amazon S3 bucket AWS endpoint.
    6. In the **S3 Region Name** field, enter the AWS region for your S3 bucket.
    7. Next to the **S3 Access Key ID** field, click **Set Value**, and then enter the S3 Access Key ID for your S3 bucket.
    8. 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**.

    9. Next to the **S3 Secret Key** field, click **Set Value** and enter the S3 Secret Key for your S3 bucket.
    10. 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** list, you must complete the **Deployment Package Trust Store** field.

6. Click **Save To PingAuthorize Server Cluster**.

Result:

Copyright © 2025 Ping Identity Corporation

Your Amazon S3 deployment package store is displayed on the **Deployment Package Stores** page.

Dsconfig

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

Administrative console

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.
    1. In the **Name** field, enter a name for the deployment package store.
    2. 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.

    3. 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**.

    4. In the **Azure Blob Container** field, enter the name of your container.
    5. 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.

Copyright © 2025 Ping Identity Corporation

Next steps

Configure the PingAuthorize server to use embedded PDP mode with your deployment package store.

Dsconfig

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.

Copyright © 2025 Ping Identity Corporation

Prepare policies for production

If you configured any of the following features during policy development and testing using the Policy Editor as your decision point, you need to reconfigure these features before putting your policies into a production environment using PingAuthorize Server as your decision point:

- Policy configuration keys
- Example: Define a policy information provider key store for MTLS
- Example: Define a policy information provider trust store
- Example: Add SpEL Java classes to the allowed list

Example: Define a policy information provider key store for MTLS

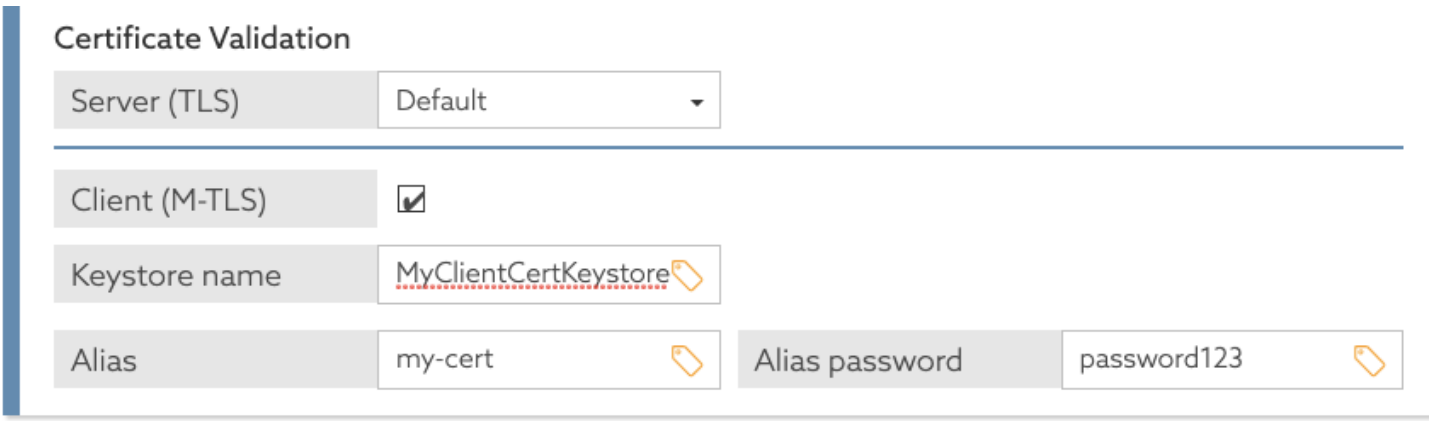
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. In these cases, the policy engine can use a client certificate contained in a Java KeyStore (JKS) or PKCS12 key store.

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`.



Copyright © 2025 Ping Identity Corporation

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.

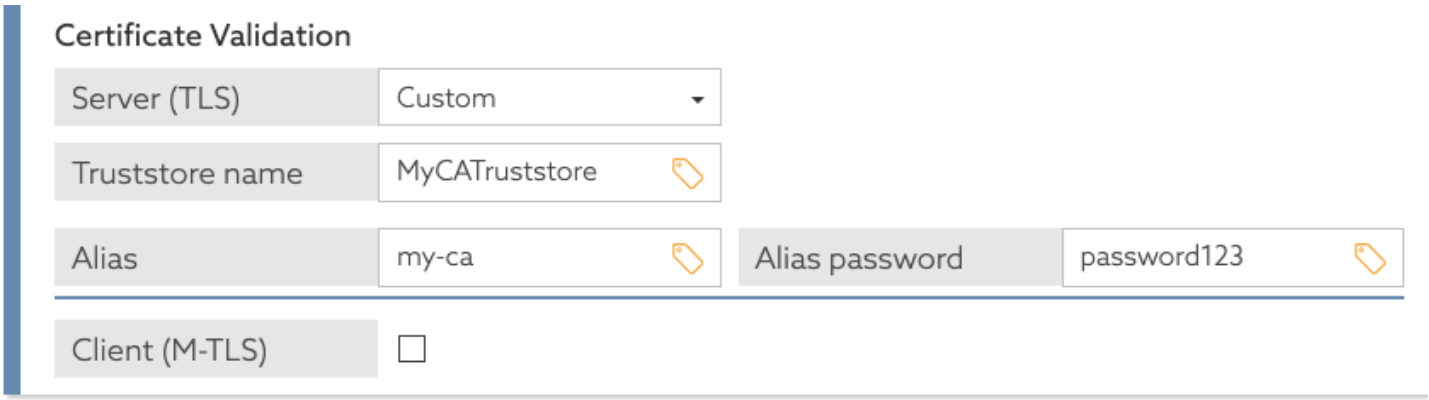
The policy engine supports the use of PIPs to dynamically retrieve data from external services at runtime. By default, the policy engine determines whether it should accept a PIP's server certificate using the Java Runtime Environment (JRE)'s default trust store, which contains public root certificates 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.

Copyright © 2025 Ping Identity Corporation

```
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.

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
```

Configuring embedded PDP mode

Configure PingAuthorize Server in embedded PDP mode to use either:

- A deployment package store using the Deployment Manager functionality
- An exported deployment package

Copyright © 2025 Ping Identity Corporation

Deployment package store

About this task

Follow these steps to assign a deployment package store to the Policy Decision Service and set the policy decision point (PDP) mode to embedded.

> **Note**
>
> Learn more about the deployment package store option and the requirements for the Deployment Manager feature in Using the Deployment Manager.

Steps

- Use `dsconfig` or the administrative console:

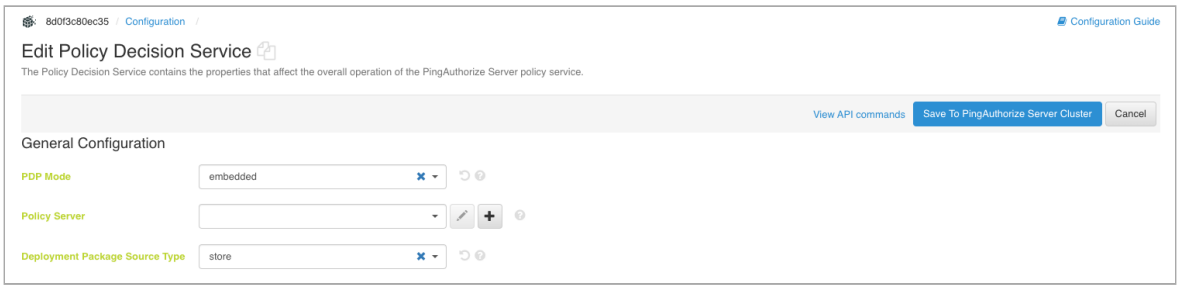
  Choose from:

  - 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** list, select your deployment package store.
    4. In the **Policy Request Configuration** section, select a **Trust Framework Version**.
    5. Click **Save To PingAuthorize Server Cluster**.

Exported deployment package

About this task

To assign an exported deployment package to the Policy Decision Service and set the PDP mode:

Copyright © 2025 Ping Identity Corporation

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 policy deployment package.

```
dsconfig set-policy-decision-service-prop \
  --set pdp-mode:embedded \
  --set "deployment-package</path to deployment-package.deploymentpackage"
```

Configuring policy request header mappings

With the Policy Decision Service set to external or embedded policy decision point (PDP) mode, you can configure the PingAuthorize server to enforce policy request header mappings on JSON PDP API requests.

By defining a policy request header mapping, you can map a decision request header to a Trust Framework attribute. The PingAuthorize server uses this mapping to dynamically populate the attribute's value with the value of an incoming request header, enabling you to leverage header data in the decision request body. In embedded PDP mode, the PingAuthorize server applies these mappings to individual and batch JSON PDP API requests only.

Note

- If a decision request includes different values for a header and attribute that are mapped to each other, the Policy Decision Service replaces the attribute value with the header value.
- You can define multivalued headers with either comma-separated values or with duplicate header names. If the header's values are comma-separated, the Policy Decision Service treats these values as a single string. If there are duplicate header names with different values, the Policy Decision Service uses the first value specified in the request.
- If a decision request includes different header names mapped to the same attribute, the Policy Decision Service uses the value of the header that appears last in the administrative console list. This list is ordered alphabetically.

You can configure policy request header mappings for development and testing in the Policy Editor. Learn more in Configuring Policy Editor policy request header mappings.

Copyright © 2025 Ping Identity Corporation

Administrative console

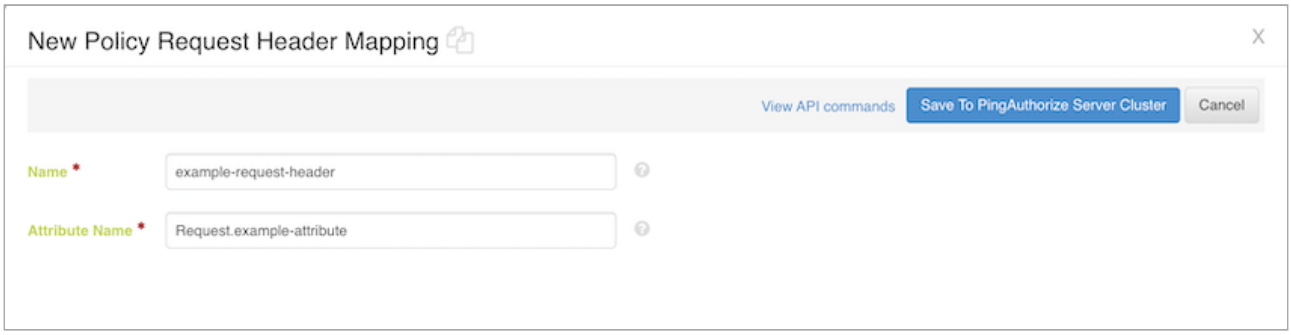
Configuring policy request header mappings using the administrative console

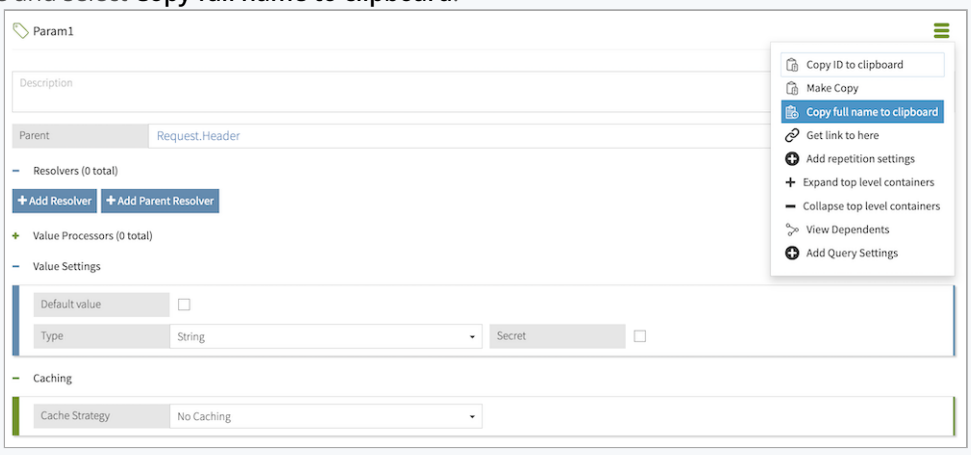
Steps

1. On the **Configuration** page of the PingAuthorize administrative console, go to **Authorization and Policies > Policy Decision Service**.
2. In the **Policy Request Header Mappings** section, click **New Policy Request Header Mapping**.
3. In the **Name** field, enter the name of the header.

> **Note**
>
> The request header name isn't case-sensitive.

4. In the **Attribute Name** field, enter the full name of the attribute that you want to map to the relevant header.



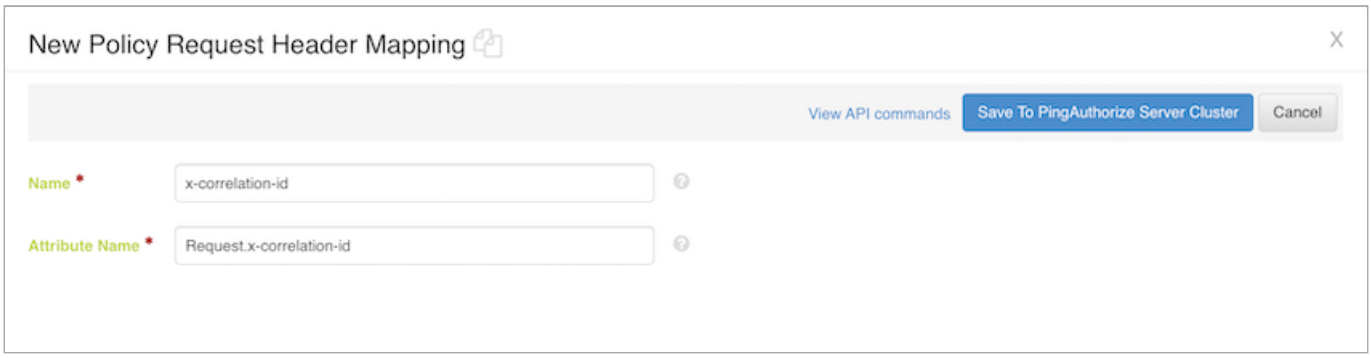
> **Tip**
>
> The full name of an attribute represents that attribute's full path in the Trust Framework hierarchy. For example, a **Param1** attribute with parent attributes **Request** and **Header** would have a full name of **Request.Header.Param1**. To quickly obtain an attribute's full name, click the hamburger menu of that attribute and select **Copy full name to clipboard**.
>
> 
>

5. Click **Save to PingAuthorize server cluster**.

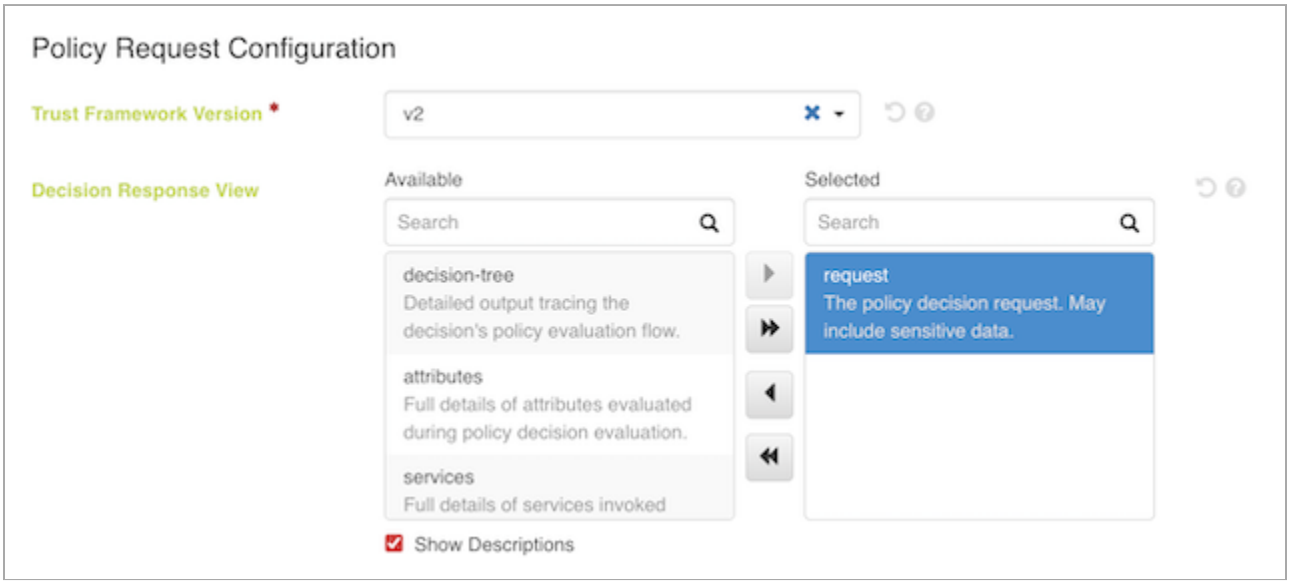
Example

Copyright © 2025 Ping Identity Corporation

Suppose you want to add a request correlation ID so that your logging and monitoring services have full visibility of a decision request's processing journey. This correlation ID is represented by the **x-correlation-id** authorization attribute, which has a parent **Request** attribute. The following policy request header mapping creates a mapping between the `x-correlation-id` header and the **x-correlation-id** attribute:



To include the modified decision request body in the JSON PDP API response, select the **request** view in the Decision Response View.



> **Warning**
>
> Selecting the **request** view causes the Policy Decision Logger to record potentially sensitive data in API requests and responses.

Copyright © 2025 Ping Identity Corporation

Suppose a decision request includes a sample **Attribute1** attribute and a header value of `x-correlation-id:abc`. The relevant authorization policy produces a `PERMIT` decision if the **Request.x-correlation-id** attribute equals `"abc"` and a `DENY` decision otherwise:

```
{
  "domain": "example.Domain",
  "action": "example.Action",
  "service": "example.Service",
  "identityProvider": "example.Identity Provider",
  "attributes": {
    "Attribute1": "A request body attribute"
  }
}
```

The following decision response includes the modified request body, which now includes the `Request.x-correlation-id` attribute set to the `x-correlation-id` header's value:

```
{
  "id": "18e98969-3915-4096-b437-71100ac1d70f",
  "deploymentPackageId": "502bdfdf-da19-47c9-b474-0047f77d18de",
  "timestamp": "2024-05-23T15:29:30.115879Z",
  "elapsedTime": 193449,
  "request": {
    "domain": "example.Domain",
    "service": "example.Service",
    "action": "example.Action",
    "attributes": {
      "Attribute1": "A request body attribute",
      "Request.x-correlation-id": "abc"
    }
  },
  "decision": "PERMIT",
  "authorised": true,
  "statements": [],
  "status": {
  "code": "OKAY",
  "messages": [],
  "errors": []
  }
}
```

Copyright © 2025 Ping Identity Corporation

Dsconfig

Configuring policy request header mappings using **`dsconfig`**

Steps

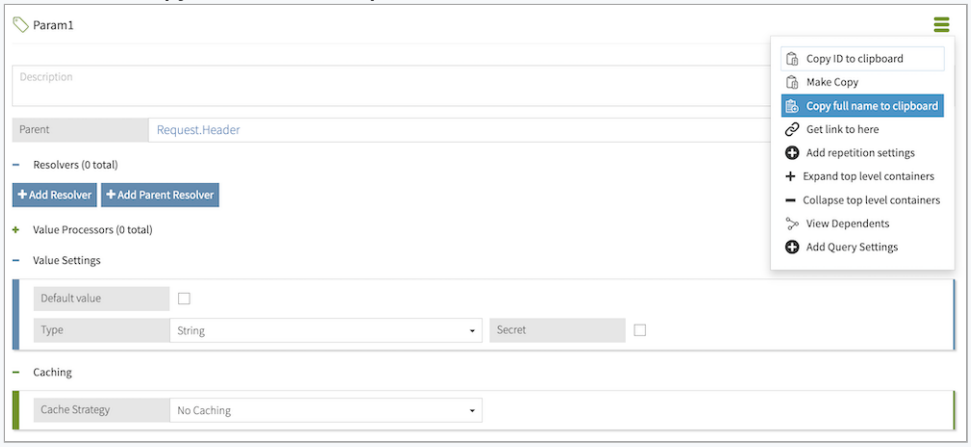
- To create a new policy request header mapping, use the **`dsconfig create-policy-request-header-mapping`** command and supply the `mapping-name` and `set attribute-name` arguments.

```
dsconfig create-policy-request-header-mapping \
--mapping-name x-param-1 \
--set attribute-name: Request.Header.param1
```

The `mapping-name` should match the name of the request header, and the `attribute-name` should match the full name of the attribute defined in the Trust Framework.

Tip

The full name of an attribute represents that attribute's full path in the Trust Framework hierarchy. For example, a **Param1** attribute with parent attributes **Request** and **Header** would have a full name of **Request.Header.Param1**. To quickly obtain an attribute's full name, click the hamburger menu of that attribute and select **Copy full name to clipboard**.



Policy database backups

The 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. The Policy Editor retains up to five policy database backups at a time. Older backups are automatically deleted after this limit is reached.

To customize the handling of policy database backups, configure the admin port, backup schedule, and output location.

Copyright © 2025 Ping Identity Corporation

> **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 policy database backups

Do one of the following to change the backup configuration:

- 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. Learn more about using the environment variables in Starting PingAuthorize Policy Editor.

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. Learn more in Quartz 2.3.0 cron format. **Note** The PAP evaluates the expression against the system timezone. For 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 periodic policy database backups.

Copyright © 2025 Ping Identity Corporation

Learn more about using a backup in Restoring a policy database from a backup.

Restoring a policy database from a backup

The policy database stores 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.

Learn more about configuring backups in Policy database backups.

> **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.

Not Using Docker

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` archive 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` archive 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
   ```


Copyright © 2025 Ping Identity Corporation

Using Docker

Restoring a database when using Docker
About this task

To restore a policy database in a Docker environment:

Steps

1. Locate the backup `.zip` archive 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.

2. Extract the `.zip` archive to the database location that you'll specify using the `PING_H2_FILE` environment variable when you start the Docker container.

3. Start the Policy Editor 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/{PAP_CONTAINER_NAME}:<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. You can find actual tag values in the 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

Copyright © 2025 Ping Identity Corporation

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:

1. Set up the healthcare configuration.
    1. Create a signing key pair with a private key and a public key for healthcare.
    2. Set up a Policy Editor to create all healthcare policies. Configure that GUI to sign its deployment packages with the healthcare private key.
    3. 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.
    1. Create a signing key pair with a private key and a public key for banking.
    2. Set up a Policy Editor to create all banking policies. Configure that GUI to sign its deployment packages with the banking private key.
    3. 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.

About this task

This example uses the `manage-certificates` tool that comes with PingAuthorize. The tool provides many of the same features as the Java `keytool` utility. If you prefer to use `keytool`, running `manage-certificates --display-keytool-command` shows a command that you can use to obtain a similar result.

Steps

1. Generate a signing key pair for the Policy Editor:
    1. Create a key pair consisting of a private key and the corresponding public key and put the key pair in a key store so that the Policy Editor can use it.

        Example:

        The following command performs both of these actions by generating a key store with a self-signed certificate:

Copyright © 2025 Ping Identity Corporation

```
$ 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 name `healthcare-pap-signing.jks`, containing the Policy Editor's private signing key and the corresponding public key.
- The Policy Editor uses this key store to sign deployment packages.
- The key store itself and the private key each have passwords.
- The signing key pair has the alias `healthcare-pap`.
- The subject DN is arbitrary.
- The keys are valid for 90 days.

2. Export a public certificate from the Policy Editor's key store.

   Example:

   ```
   $ 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 name `healthcare-pap.pem`. This file becomes an input during the next step, and it is not used directly by either the Policy Editor or PingAuthorize. The public certificate specified by the `--alias` argument represents the public key created in the previous step.

3. Create a trust store for PingAuthorize Server for the public certificate from the previous step.

   Example:

   ```
   $ 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 name `healthcare-pap-verification.jks`, containing the Policy Editor's public certificate. PingAuthorize Server uses this trust store to verify that deployment packages created by the Policy Editor were actually created by that GUI.

Copyright © 2025 Ping Identity Corporation

4. Configure the Policy Editor to use the key store to sign the deployment packages it creates:

    1. Make a copy of the default options file.

    ```
    $ cp config/options.yml my-options.yml
    ```

    2. Edit the new options file for your policy deployment method, uncommenting the appropriate block and substituting your passwords and other values.

    > **Note**
    >
    > Using `deploymentPackageData` to define the signing key configuration only affects exported deployment packages. Deployment packages published to a store only use their store-specfic configurations defined in the `deploymentPackageStores` block to apply a signing key.

    3. Use the `deploymentPackageData` block to configure signing for exported deployment packages.

    ```
    deploymentPackageData:
      #  contentType: json
      #  keystore:
      #    resource: /path/to/healthcare-pap-signing.jks
      #    password: keystore-password
      #  securityLevel: signed
      #  signingKey:
      #    alias: healthcare-pap
      #    password: private-key-password
    ```

    4. Under the `deploymentPackageStores` block, use the appropriate store type to configure signing for deployment packages published to a deployment package store.

    ```
    deploymentPackageStores:
      # Define deployment package store publishing targets here.
      #
      # - 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/healthcare-pap-signing.jks
      #     password: keystore-password
      #   signingKey:
      #     alias: healthcare-pap
      #     password: private-key-password
      #  Other deployment package store types omitted for brevity...
    ```

    5. Stop the Policy Editor.

Copyright © 2025 Ping Identity Corporation

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

6. Run `setup` using the `--optionsFile my-options.yml` argument.

   Customize all other options as appropriate for your needs.

7. 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:

   1. Create a trust manager provider and include the path to the trust store file and the trust store's password.

      The trust manager provider is how the PingAuthorize Server configuration refers to a trust store file.

      ```
      $ 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>"
      ```

   2. 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.

Copyright © 2025 Ping Identity Corporation

Property	Description
`deployment-package-security-level`	Determines whether PingAuthorize Server require a deployment package to be signed. Valid values are: **`unsigned`** PingAuthorize Server does not check a deployment package for a trusted signature. This value is set by default. **`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 the value of `deployment-package-security-level` is `signed`.
`deployment-package-verification-key-nickname`	Specifies the alias of the Policy Editor's public certificate. This property is required if the value of `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.

Configuring Trust Framework attribute caching for production

For higher environments, including testing and production, you can define an external attribute cache for the Trust Framework.

With the Policy Decision Service set to embedded policy decision point (PDP) mode, the PingAuthorize Server is configured by default to cache attribute values in memory (for any attributes with a defined caching strategy). Alternatively, you can define an external attribute cache using the following Redis modes:

- Single Redis instance
- Single Redis instance using TLS
- Replicated Redis
- Redis Sentinel
- Amazon Web Services (AWS) ElastiCache Redis

Copyright © 2025 Ping Identity Corporation

Using the admin console

Setting up Redis external attribute caching in the UI

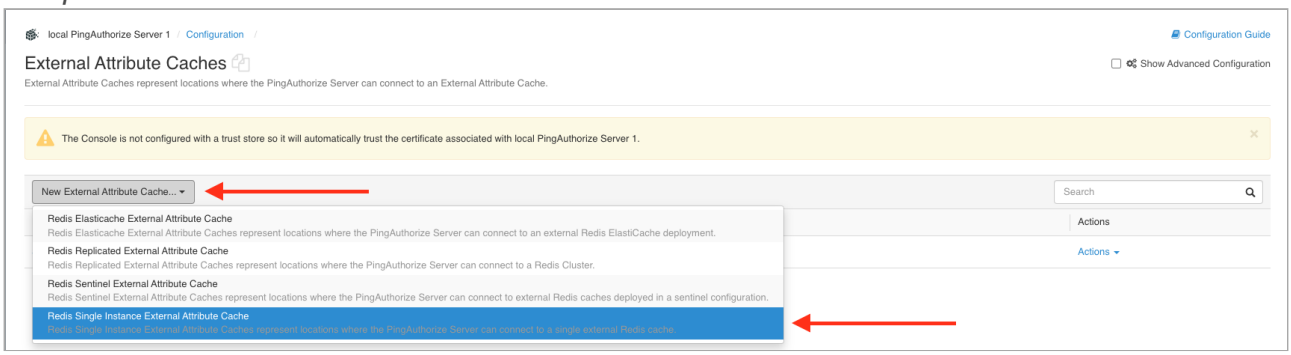
Before you begin

To successfully assign an external Redis attribute cache to the Policy Decision Service, you must set **PDP Mode** to **embedded**.

Steps

1. On the **Configuration** page of the PingAuthorize administrative console, go to **Authorization and Policies > External Attribute Caches**.
2. Select your desired Redis mode in the **New External Attribute Cache** list.

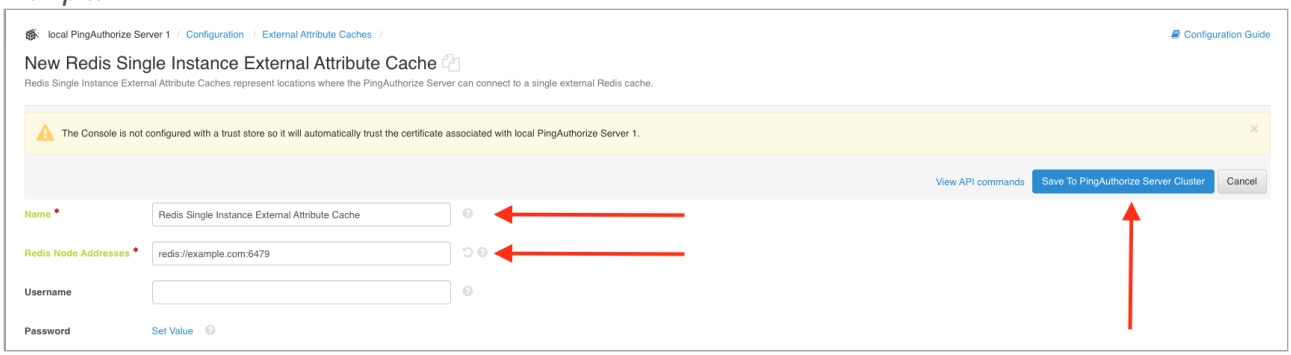
   Example:

3. At minimum, enter the required values, as indicated by a red asterisk, and click **Save To PingAuthorize Server Cluster**.

   > **Note**
   >
   > For more information on a field, click the question mark icon.

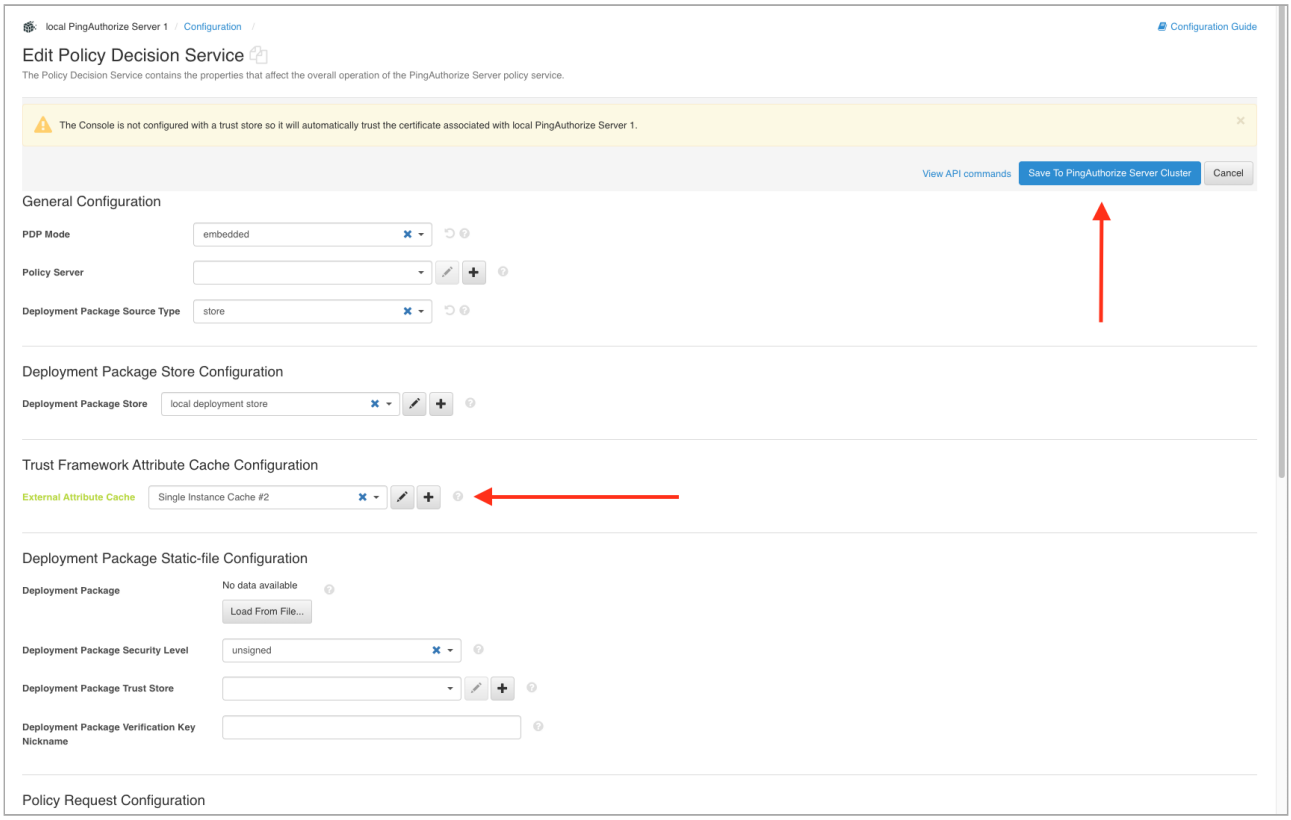
   Example:

4. Go to **Authorization and Policies > Policy Decision Service**.
5. In the **Trust Framework Attribute Cache Configuration** section, in the **External Attribute Cache** list, select your Redis cache name and click **Save To PingAuthorize Server Cluster**.

   Example:

Copyright © 2025 Ping Identity Corporation



Note

Alternatively, you can use the controls next to the **External Attribute Cache** list to create, edit, or remove external Redis caches:

- Click the **Plus** icon to create a new external attribute cache.
- Click the **Pencil** icon to edit the configuration of the selected attribute cache.
- Click **X** to remove the attribute cache from the Policy Decision Service and revert the PDP to an in-memory attribute cache.

Copyright © 2025 Ping Identity Corporation

Using dsconfig

Setting up Redis external attribute caching with dsconfig
Before you begin

When using the **`dsconfig set-policy-decision-service-prop`** command, the new configuration must still be compliant with the following:

- The `pdp-mode` property must be set to `embedded`.
- The `deployment-package-source-type` property must be set to `store` or `static-file`.
  - If the `deployment-package-source-type` property is set to `store`, the `deployment-package-store` property must resolve to a valid deployment package store.
  - If the `deployment-package-source-type` property is set to `static-file`, the `deployment-package-store` property must resolve to a valid deployment package.

About this task

Here are the configuration options available for creating Redis external caches using the **`dsconfig`** tool. When using the **`dsconfig create-external-attribute-cache`** command, the new configuration must still be compliant with the required attributes associated with the specified cache type:

Option	Description
`mode`	Required. Specifies Redis mode. Accepted values: `single_instance`, `replicated`, `elasticache`, or `sentinel`.
`nodeAddresses`	Required, only when `mode` is `single_instance`, `replicated`, or `sentinel`. Defines node addresses. A comma-separated list of Redis nodes.
`replicationGroupId`	Required, only when `mode` is `elasticache`. Replication group ID.
`masterName`	Required, only when `mode` is `sentinel`. Specifies name of the master node.
`database`	Optional, only when `mode` is `sentinel`. Database index used for Redis connection. Default value is `0`.
`scanInterval`	Optional, only when `mode` is `sentinel`. Redis cluster scan interval in milliseconds. Default value is `1000`.
`checkSentinelList`	Optional, only when `mode` is `sentinel`. Enables Sentinels list check during startup. Default value is `false`.
`username`	Optional, only when AUTH token authentication is enabled in the Redis provider.

Copyright © 2025 Ping Identity Corporation

Option	Description
`password`	Optional, only when AUTH token authentication is enabled in the Redis provider.

Steps

1. Create the external attribute cache using the **`dsconfig create-external-attribute-cache`** command. For example:

   Example:

   ```
   $ dsconfig create-external-attribute-cache \
     --cache-name 'Single Instance' \
     --type redis-single-instance \
     --set redis-node-addresses:redis://localhost:6379
   ```

2. Assign the defined external attribute cache to the Policy Decision Service. For example:

   Example:

   ```
   $ dsconfig set-policy-decision-service-prop \
     --set 'external-attribute-cache:Single Instance'
   ```

Result

Your external attribute cache has been defined and attached to the Policy Decision Service. There is no need to restart the server.

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. Using a SCIM token resource lookup method requires a SCIM resource type to be configured, along with that resource type's prerequisite configuration objects. Learn more about SCIM configuration, such as SCIM resource types, store adapters, load-balancing algorithms, and LDAP external servers, in SCIM configuration basics.

You can find examples that show how to set up a token resource lookup method in:

- Configuring the PingAuthorize OAuth subject search
- Sideband access token validation

Copyright © 2025 Ping Identity Corporation

- SCIM token resource lookup methods

User profile data from access tokens

When processing an incoming HTTP request, the 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, the token validator then invokes any related token resource lookup methods. If a token resource lookup method succeeds in retrieving the attributes for the token owner, then the 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 using an HTTP GET operation 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:

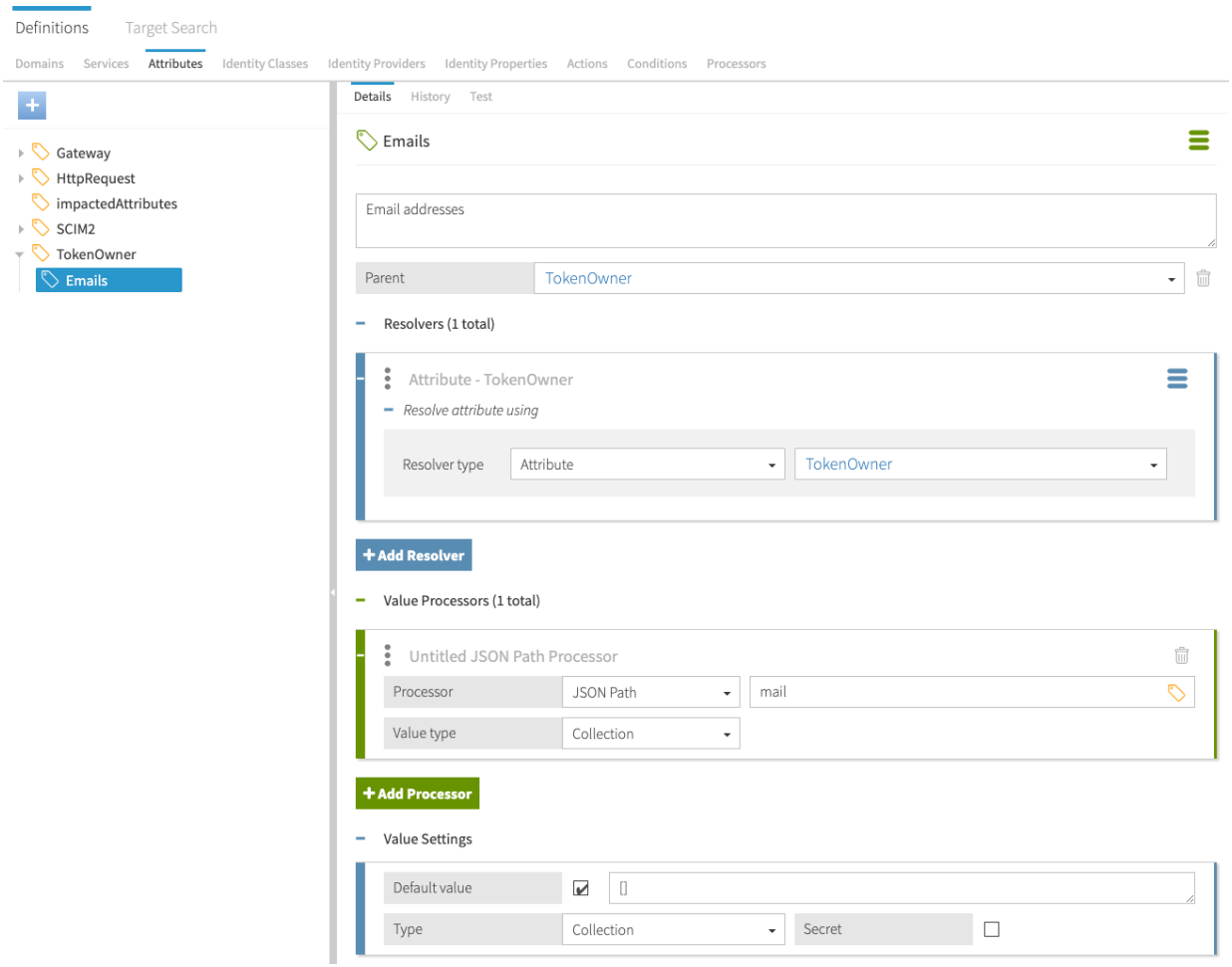
Copyright © 2025 Ping Identity Corporation

```
{
    "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 that 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 using the token owner's email address, you can add an `Emails` attribute under `TokenOwner` in the PingAuthorize Policy Editor:

Copyright © 2025 Ping Identity Corporation



Access token validators

Access token validators verify the tokens that client applications submit when they request access to protected resource.

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:

```
"Authorization": "Bearer  <access-token>"
```

> **Note**
>
> PingAuthorize does not support the use of query parameters for token validation.

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.

Copyright © 2025 Ping Identity Corporation

Most access tokens identify a user, also called the token owner, as its subject (SAML). Access token validators can retrieve the token owner's attributes from the datastore 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.

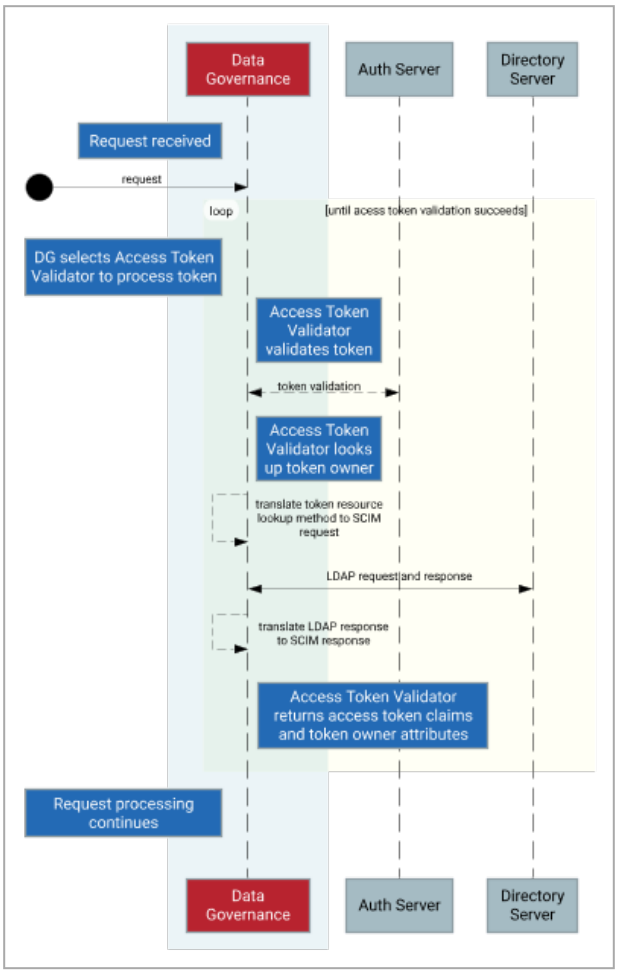
For more information about the types of access tokens PingAuthorize can validate, see Access token validator types.

For information about validating a JSON Web Token (JWT) in policy instead of using a validator, see Conditions.

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 inbound HTTP request contains an access token, PingAuthorize sends the token 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.

Copyright © 2025 Ping Identity Corporation

- 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.

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 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 application programming interface (API) gateway

Copyright © 2025 Ping Identity Corporation

PingFederate

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 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

Copyright © 2025 Ping Identity Corporation

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 "{pingfed}  External Server" \
  --type http \
  --set base-url:https://example.com:9031
# Create the Access Token Validator
dsconfig create-access-token-validator \
  --validator-name "{pingfed}  Access Token Validator" \
  --type ping-federate \
  --set enabled:true \
  --set "authorization-server:{pingfed}  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 "{pingfed}  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.

Copyright © 2025 Ping Identity Corporation

JWT

JWT access token validator

The JWT access token validator verifies access tokens that are encoded in JWT format, which can be signed in JSON web signature (JWS) format or signed and encrypted in JSON web encryption (JWE) format.

The JWT access token validator inspects the JWT token without presenting it to an authorization server for validation. 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

Mock access token

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 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 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
```


Copyright © 2025 Ping Identity Corporation

Third-party

Third-party access token validator

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

External API gateway

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.

> **Note**
>
> External API gateway access token validators are exclusively for use by Sideband API endpoints. If you assign an external API gateway access token validator to any other server component, either explicitly or implicitly, it is ignored.

Example configuration

The following example shows how to configure an external API gateway access token validator with a token resource lookup method and 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"
```

Handling signed JWTs

All access tokens the JSON web token (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.

Copyright © 2025 Ping Identity Corporation

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 `signing-certificate` property.

    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 allowed-signing-algorithm:RS256 \
      --set "signing-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
    ```

  - 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.

    The following example configures a JWT access token validator to retrieve public keys from a PingFederate authorization server's JWKS endpoint.

Copyright © 2025 Ping Identity Corporation

```
# Change the host name and port below, as needed
dsconfig create-external-server \
  --server-name "{pingfed}  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 allowed-signing-algorithm:RS256 \
  --set "authorization-server:{pingfed}  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 signed and encrypted JWTs

The JSON web token (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.

About this task

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:

Copyright © 2025 Ping Identity Corporation

Choose from:

- Store the public key as a trusted certificate in PingAuthorize Server's local configuration using the `signing-certificate` property.

  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 allowed-signing-algorithm:ES256 \
    --set "signing-certificate:JWT Signing Certificate" \
    --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
  ```

- 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.

  The following example configures a JWT access token validator to retrieve public keys from a PingFederate authorization server's JWKS endpoint.

Copyright © 2025 Ping Identity Corporation

```
# Change the host name and port below, as needed
dsconfig create-external-server \
  --server-name "{pingfed}  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 allowed-signing-algorithm:ES256 \
  --set "authorization-server:{pingfed}  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.

> **Note**
>
> 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. Be prepared to enter your connection properties and bind password when prompted.

```
dsconfig get-key-pair-prop \
  --pair-name "JWT Elliptic Curve Encryption Key Pair" \
  --property certificate-chain
```

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.

Copyright © 2025 Ping Identity Corporation

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
- Third-party token resource lookup methods

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.

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"`.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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}  {PAP_Name}" \
  --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.
```


Copyright © 2025 Ping Identity Corporation

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"
```

Copyright © 2025 Ping Identity Corporation

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[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*.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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:

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

```
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:

    1. 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
    ```

    2. 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.

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.
`--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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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. **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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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.

Copyright © 2025 Ping Identity Corporation

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.

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.

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

> **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:

Copyright © 2025 Ping Identity Corporation

`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.

Copyright © 2025 Ping Identity Corporation

`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.

Copyright © 2025 Ping Identity Corporation

> **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-----
```

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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, refer to 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.

Copyright © 2025 Ping Identity Corporation

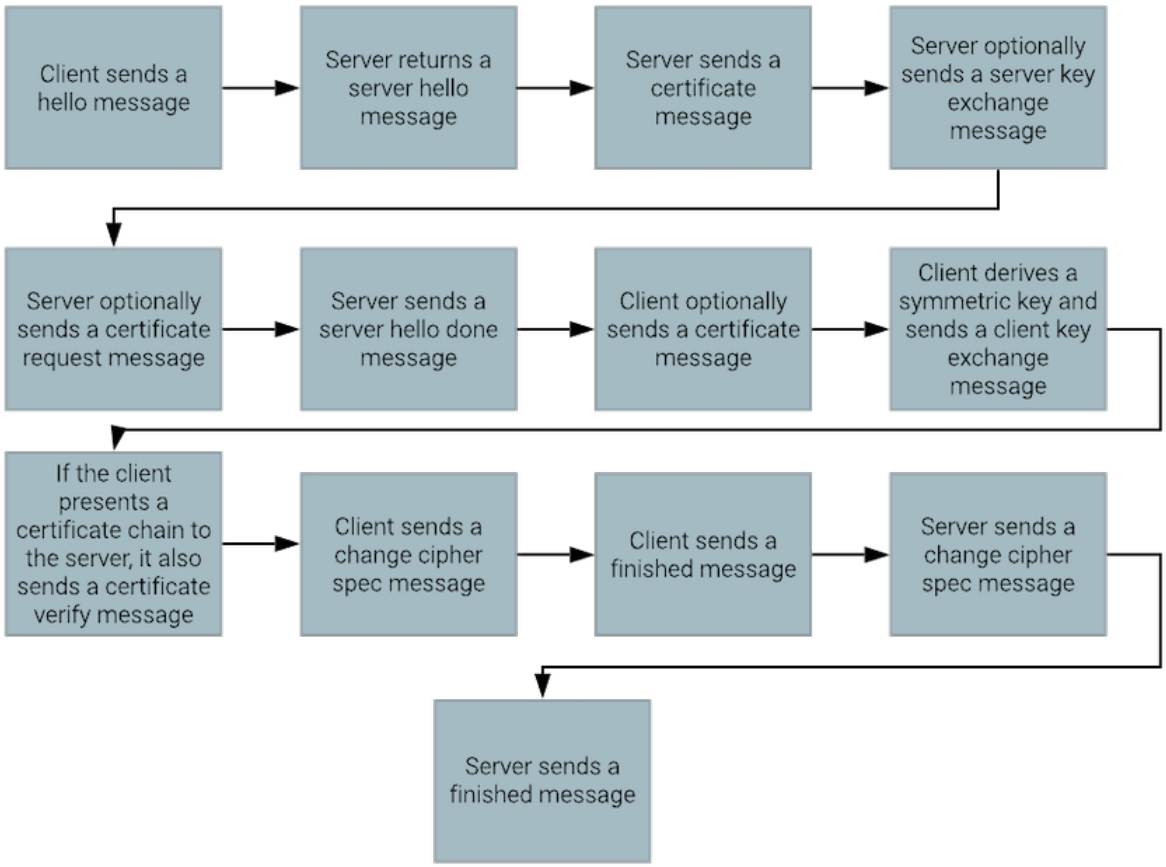
- If you rely entirely on the security of a private key to ensure the secrecy of a communication, and if the private key becomes compromised, data that has been encrypted with the private key must also be considered compromised.

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

Copyright © 2025 Ping Identity Corporation

- 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.
`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.

Copyright © 2025 Ping Identity Corporation

Subcommand	Function
`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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

- `--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.
`--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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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
```

Copyright © 2025 Ping Identity Corporation

```
          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:
   -20753103001920939059800335367415761738764700353772539765405069978726324039 64
Elliptic Curve Y-Coordinate:
   67079356503908427292378918440889412002659485731683570737365127953554508553 73
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.

Copyright © 2025 Ping Identity Corporation

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.
`--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

Copyright © 2025 Ping Identity Corporation

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.
`--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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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.
`--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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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.

Copyright © 2025 Ping Identity Corporation

```
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

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

- 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
$ 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 .

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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.
`--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.

Copyright © 2025 Ping Identity Corporation

```
$ ./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. 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.

3. Configure connection handlers.

   For more information, see Configuring TLS connection handlers.

Copyright © 2025 Ping Identity Corporation

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, as indicated in the following 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, as indicated in the following 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.

Caching key and trust managers

When you create key and trust manager providers, caching is enabled by default, allowing the manager providers to avoid loading key store and trust store files from disk when establishing connections to process requests.

Invalidating the cache

The manager provider reloads files from the configured key store or trust store and refreshes the cache under any of the following conditions:

- The cached manager for the configured store has a `null` value.
- The path to the cached store doesn't match the path of the configured store.
- The length of the cached store doesn't match the length of the configured store.
- The last-updated time for the cached store doesn't match the last-updated time for the configured store.

Enabling or disabling caching (optional)

Copyright © 2025 Ping Identity Corporation

You can define whether caching is enabled by using the `enable-key-manager-caching` or `enable-trust-manager-caching` properties. Supply a value:

- `false` to disable caching, causing manager providers to load managers for each connection
- `true` to re-enable caching

To create a key manager provider with caching disabled, supply the `enable-key-manager-caching` property with a value of `false`, as shown in the following example:

```
dsconfig create-key-manager-provider \
    --provider-name JKS \
    --type file-based \
    --set enabled:true \
    --set key-store-file:config/keystore \
    --set key-store-type:JKS \
    --set key-store-pin-file:config/keystore.pin \
    --set enable-key-manager-caching:false
```

To create a trust manager provider with caching disabled, supply the `enable-trust-manager-caching` property with a value of `false`, as shown in the following example:

```
dsconfig create-trust-manager-provider \
    --provider-name JKS \
    --type file-based \
    --set enabled:true \
    --set trust-store-file:config/truststore \
    --set trust-store-type:JKS \
    --set enable-trust-manager-caching:false
```

To re-enable caching, set a value of `true` for the same caching property used to create the manager provider, as shown in the following example:

```
dsconfig set-trust-manager-provider-prop \
    --provider-name JKS \
    --set enable-trust-manager-caching:true
```

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:

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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

Copyright © 2025 Ping Identity Corporation

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

Copyright © 2025 Ping Identity Corporation

- 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.

Copyright © 2025 Ping Identity Corporation

```
$ 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.
```

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

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.
`--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.

Copyright © 2025 Ping Identity Corporation

Argument	Description
`--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

Copyright © 2025 Ping Identity Corporation

```
$ 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: 20191113061149Z

# 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

Copyright © 2025 Ping Identity Corporation

```
$ 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.

Copyright © 2025 Ping Identity Corporation

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 Ping Identity support team can use the output to identify the issue.

Policy Decision Service configuration

You can configure the Policy Decision Service for either of the following policy decision environments:

Development environment (external)

Policies are developed in the Policy Editor. The Policy Editor serves as the external policy decision point (PDP), and the PingAuthorize Server serves as the policy enforcement point (PEP).

Learn more about configuring the Policy Decision Service for development environments in Configuring external PDP mode.

Pre-production and production environments (embedded)

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

Learn more about configuring the Policy Decision Service for production environments in Configuring embedded PDP mode.

Configuring the Decision Response View

Use the Decision Response View to increase or decrease the size of the policy decision response from the policy decision point (PDP).

Copyright © 2025 Ping Identity Corporation

When a client application makes a request for API resources, PingAuthorize Server returns a decision response payload that includes, at minimum, basic information about the server instance, the requested API resources, and the inbound and outbound flow of data. The payload also includes any views added to the Decision Response View. By default, no views are present. PingAuthorize then passes the full response payload to the Policy Decision logger.

Adding or removing views in the Decision Response View alters the verbosity of the response payload and the size of the `policy-decision.log` file.

Note

- 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 add the following views to the Decision Response View:

Decision Response View	Description
attributes	Inclludes full details of attributes evaluated during policy decision evaluation.
decision-tree	Includes detailed output tracing the decision's policy evaluation flow.
evaluated-entities	Includes attribute and service resolution details. This is equivalent to specifying both **attributes** and **services**.
evaluation-log	Includes 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	Includes attribute and service resolution details. This is equivalent to specifying **evaluation-log** but also includes values and types for successful attribute resolutions.
request	Includes the full decision request object. **Warning** Selecting the **request** view causes the Policy Decision Logger to record potentially sensitive data in API requests and responses.
services	Includes full details of services invoked during policy evaluation.

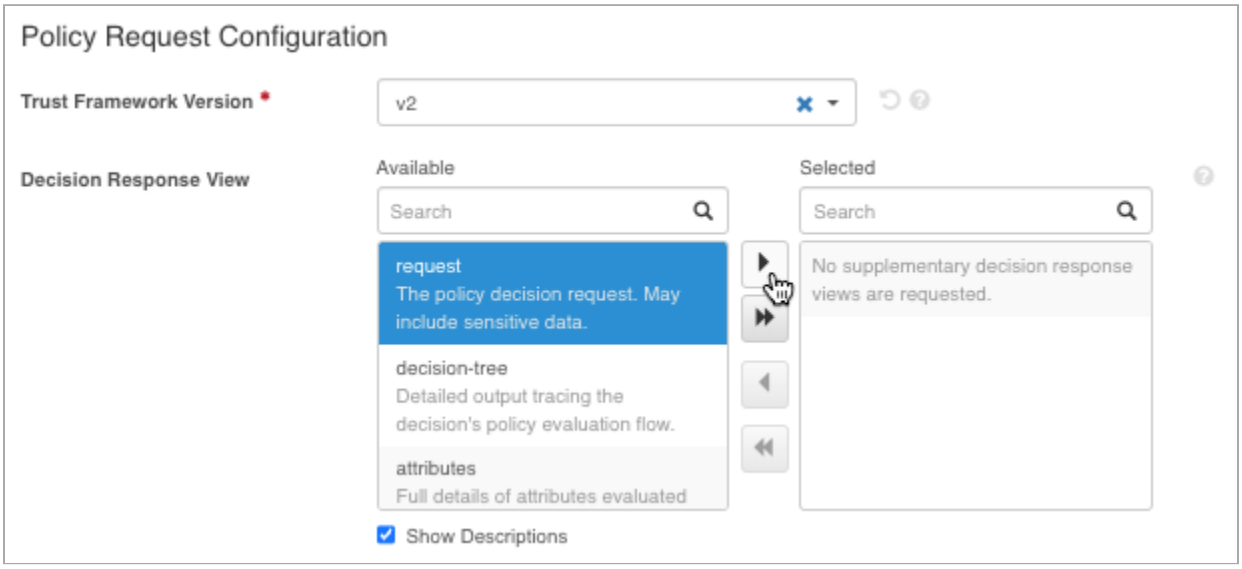
Use the administrative console or `dsconfig` to configure the Decision Response View.

Copyright © 2025 Ping Identity Corporation

Admin console

Use the administrative console
Steps

1. Go to **Configuration > Authorization and Policies** and click **Policy Decision Service**.
2. In the **Policy Request Configuration** section, next to **Decision Response View**, select a response view and click the arrow.



3. Click **Save to PingAuthorize Server Cluster**.

dsconfig

Use `dsconfig`
Steps

1. Run `dsconfig` with the `set-policy-decision-service-prop` subcommand.

   Example:

   ```
   PingAuthorize/bin/dsconfig set-policy-decision-service-prop \
     --no-prompt --port 5409 --useSSL --trustAll \
     --bindDN "cn=directory manager" \
     --bindPassword secret \
     --add decision-response-view:request
   ```

   In this example, the `request` view is added to the Decision Response View.

Copyright © 2025 Ping Identity Corporation

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.

For more information about configuring SCIM, see About the SCIM service.

Example

For an example, see Configuring the PingAuthorize user store.

Configuring database service connections

You can configure the pooling mechanism and add allowed drivers for database service connections.

Copyright © 2025 Ping Identity Corporation

A database pool is a cache of database connections that PingAuthorize uses to manage system performance. Instead of establishing a new connection each time the server needs to retrieve policy information from the database, PingAuthorize leverages an existing connection from the database pool. PingAuthorize creates a database pool for each database service that you define. The pool configuration specified in the PingAuthorize server will apply to each pool.

Copyright © 2025 Ping Identity Corporation

Admin console

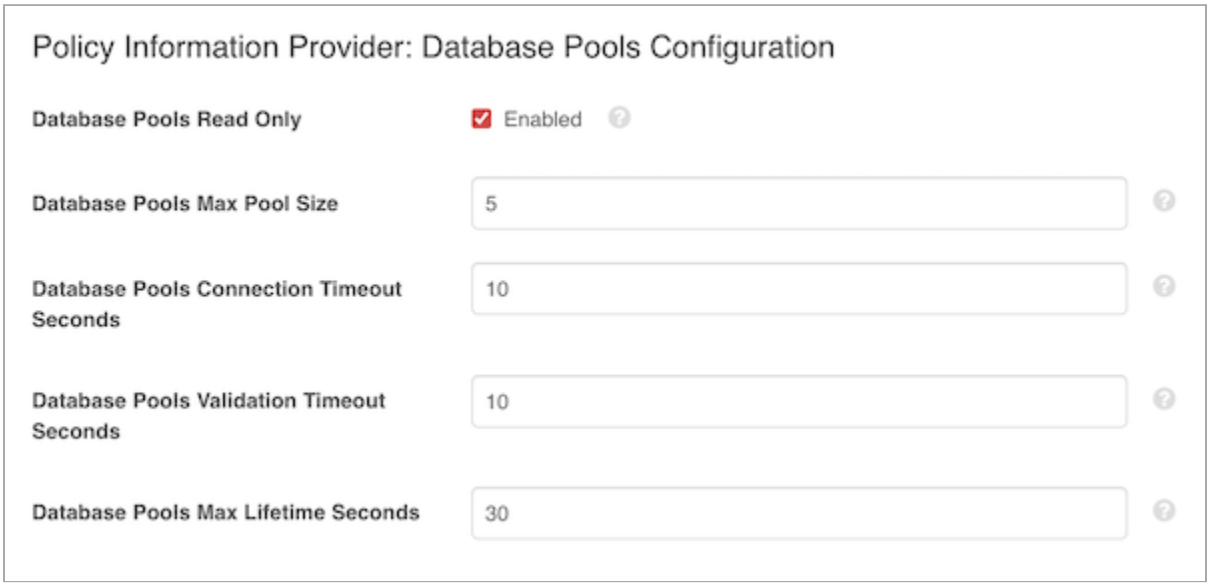
Configuring database services using the administrative console

Steps

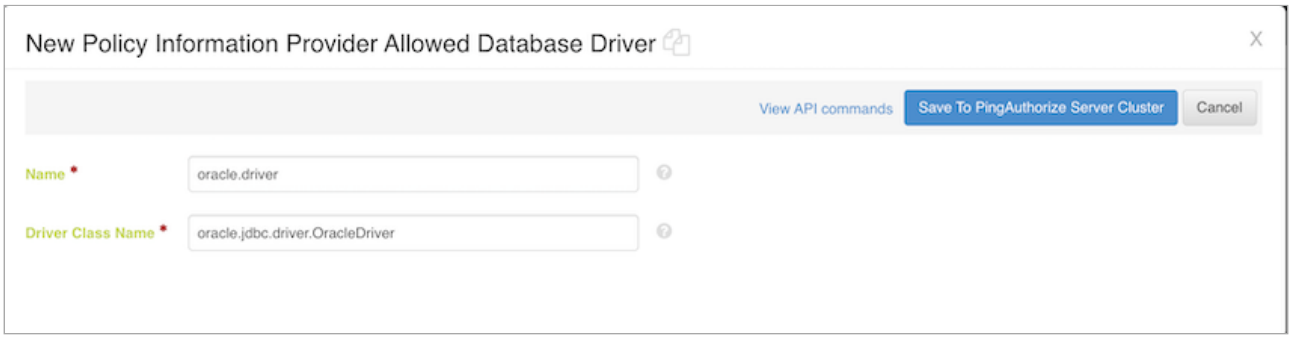
1. In the administrative console, go to **Configuration > Policy Decision Service**.
2. In the **Policy Information Provider: Database Pools Configuration** section, configure the following database pool properties:

Property	Description
Database Pools Read Only	Specifies whether the database pools are read-only. Some database types do not support the read-only mode. If the database type does not support the read-only mode, the database pools will be read-write regardless of the value of this property.
Database Pools Max Pool Size	Specifies the maximum number of connections in a database pool.
Database Pools Connection Timeout Seconds	Specifies the maximum number of seconds that a connection request waits for an available connection in the database pool.
Database Pool Validation Timeout Seconds	Specifies the maximum number of seconds that a database pool tests a connection for aliveness.
Database Pool Max Lifetime Seconds	Specifies the maximum number of seconds that a connection stays in the database pool. The database pool will only remove a connection if the maximum lifetime elapses and the connection is no longer active. Setting this property to any value between 0 and 30 will have no effect. Setting this property to 0 makes the maximum lifetime indefinite.

Copyright © 2025 Ping Identity Corporation



3. To add a database driver other than PostgreSQL or Oracle, go to the **Policy Information Provider allowed Database Drivers** section and click **New Policy Information Provider Allowed Database Driver**.
4. In the **Name** field, enter the name of the database driver in the format `.driver.`
5. In the **Driver Class Name** field, enter the fully qualified Java class name of the database driver.



Copyright © 2025 Ping Identity Corporation

dsconfig

Configuring database services using **`dsconfig`**

Steps

- To create or delete allowed database drivers in the PingAuthorize server, use the **`dsconfig create-policy-information-provider-allowed-database-driver`** or the **`dsconfig delete-policy-information-provider-allowed-database-driver`** command.

  With the `--drivername` argument, specify the name of the database driver in the format `.driver`.

  With the `--set driver-class-name` argument, specify the fully qualified Java class name of the database driver.

  Example:

  ```
  dsconfig create-policy-information-provider-allowed-database-driver --drivername
  example.driver --set driver-class-name:org.example.driver
  ```

- To configure the database pool properties, use the **`dsconfig set-policy-decision-service-prop`** command and include the following arguments:

Argument	Description
`--set database-pools-read-only`	Specifies whether the database pools are read-only. Some database types do not support the read-only mode. If the database type does not support the read-only mode, the database pools will be read-write regardless of the value of this property.
`--set database-pools-max-pool-size`	Specifies the maximum number of connections in a database pool.
`--set database-pools-connection-timeout-seconds`	Specifies the maximum number of seconds that a connection request waits for an available connection in the database pool.
`--set database-pools-validation-timeout-seconds`	Specifies the maximum number of seconds that a database pool tests a connection for aliveness.
`[.codeph]`--set database-pools-max-lifetime-seconds````	Specifies the maximum number of seconds that a connection stays in the database pool. The database pool will only remove a connection if the maximum lifetime elapses and the connection is no longer active. Setting this property to any value between 0 and 30 will have no effect. Setting this property to 0 makes the maximum lifetime indefinite.

Copyright © 2025 Ping Identity Corporation

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.

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.
    1. When prompted, select the `It's already been deployed` option.
    2. 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>
    > ```

Copyright © 2025 Ping Identity Corporation

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.

    1. Locate the desired user under the **Identities** tab. For the example purposes, we will assume the desired PingOne user has the following properties:

        - Given name: Jane
        - Family name: Smith
        - Username: jsmith

    2. 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 list:

    - 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 list, following the instructions in .pingidentity.com/bundle/pingone/page/jez1625773795534.html//[Editing an application - OIDC] in the PingOne Administration Guide.

    - 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**

Copyright © 2025 Ping Identity Corporation

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.

Configuring 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.

Copyright © 2025 Ping Identity Corporation

dsconfig

Use **`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 your changes, use **`dsconfig`** to restart the HTTP or HTTPS 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.

Copyright © 2025 Ping Identity Corporation

Admin console

Use the administrative console
Steps

1. In the PingAuthorize administrative console, go to **Configuration** > **Connection Handlers**.
2. In the **Connection Handlers** list, select the HTTP or HTTPS connection handler you want to edit.
3. To enable `Forwarded` headers, go to **Use Forwarded Headers**, and select the **Enabled** checkbox.
4. Click **Save**.
5. To finalize your changes, use `dsconfig` to restart the HTTP or HTTPS 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 stops 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
- Configuring the PingAuthorize OAuth subject search
- Configuring PingAuthorize logging

Configuring the PingAuthorize user store

Configure PingAuthorize Server to use PingDirectory Server as its user store.

Copyright © 2025 Ping Identity Corporation

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.

   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.

Copyright © 2025 Ping Identity Corporation

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.

- 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:

Copyright © 2025 Ping Identity Corporation

```
{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. Learn more in Configuring the Decision Response View.

- 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.

Learn more about enabling detailed debug logging for troubleshooting purposes in Enable detailed logging.

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

Copyright © 2025 Ping Identity Corporation

- 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.
- 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.

Copyright © 2025 Ping Identity Corporation

```
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 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.

Copyright © 2025 Ping Identity Corporation

```
-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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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
- Installing a new environment
- Scaling up your environment
- Rolling out an update

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:

Copyright © 2025 Ping Identity Corporation

```
$ 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:

     1. Copy the license key file to the server profile's `misc-files` directory.

        ```
        $ cp  {pingauthorize}.lic /opt/server-profiles/pingauthorize/misc-files/
        ```

     2. Open the `setup-arguments.txt` file in a standard text editor.
     3. Locate the `--licenseKeyFile` argument.
     4. 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
        ```

     5. 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:

     1. Open the `/opt/server-profiles/pingauthorize/dsconfig/00-config.dsconfig` file in a standard text editor.
     2. Locate the value specified for `base-url` for the external server that identifies your PingFederate server.

Copyright © 2025 Ping Identity Corporation

3. Replace the value with a variable, like `${PF_BASE_URL}`.
4. Save your changes.
5. Create or update a server profile variables file for your development environment.
6. Add a row like the following example to the variables file.

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

7. Save your changes.
8. 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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 profiles in a pets service model

In a pets service model, server profiles and other DevOps concepts can be invaluable for efficiency and consistency in server management, maintenance, and testing.

For example, the step of using the `manage-profile generate-profile` subcommand to generate a server profile from a production server creates an easily consumable representation of the server's configuration. In nearly every scenario, the generation of a profile from an existing server is simpler than the piecing together manually of schemas, extensions, and other configuration information to create an image of that server. Additionally, generated profiles can be backed up or checked in to source control to maintain a consistent picture of an active server's configuration.

Copyright © 2025 Ping Identity Corporation

Another valuable use of server profiles involves setting up servers in a test environment that is separate from production. For example, a profile that matches the profile of a production server can be generated and used to install a fresh test server that matches the production server. Further, variable substitution allows environmental changes, such as local host name or instance name, without requiring a separate profile. Because the server's original configuration matches the running production server, adjustments can be tested easily. This approach provides more consistency when you validate changes before moving them to production.

If a new pet server has been set up with a server profile, `manage-profile replace-profile` can be used to apply changes to the profile. Rather than using scripts or a manual process to apply individual changes, `replace-profile` provides a consistent, repeatable method of moving to a new server profile. This strategy automates more easily and is less prone to human error.

For more information about the `manage-profile` tool, see About the manage-profile tool.

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.

`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

There are cases in which you might want to obtain basic information about the state of the server over HTTP, or ideally, HTTPS. These might include:

- When server HTTP endpoints are fronted by a load balancer capable of issuing and interpreting the results of HTTP requests for health-checking purposes.
- When using an orchestration framework such as Kubernetes, you can assess the current state of the server to determine if the instance is in a state that is ready to be used or if it has encountered an issue that has caused it to become degraded or unavailable. In this case,you can call the availability servlet in a liveness or readiness probe.

To assist with this, the PingAuthorize server includes an availability state HTTP servlet that you can use to determine whether the server considers itself to be available, degraded, or unavailable. You can access the servlet using the GET, POST, or HEAD methods, and the servlet returns a configurable status code and an optional JSON-encoded response body that you can use to determine the server's self-assessed health state.

The availability state HTTP servlet extension supports two configuration properties for the JSON response body:

- `include-response-body` : Indicates whether the servlet response should include a JSON object. If enabled, the response will include an `availability-state` field with a default value of `AVAILABLE` , `DEGRADED` , or `UNAVAILABLE` . This configuration property allows a value of either `true` or `false` .

Copyright © 2025 Ping Identity Corporation

- `additional-response-contents`: Provides a JSON-formatted string containing additional fields for the servlet to include in its response.

Use `dsconfig` to update the HTTP servlet extension configuration:

```
dsconfig set-http-servlet-extension-prop \
--extension-name {name} \
(--set|--add|--remove) {propertyName}:{propertyValue}
```

There are two instances of this servlet configured by default:

- One that uses a path of `/available-state` that returns an HTTP status code of 200 (OK) if the server considers itself available or an HTTP status code of 503 (Service Unavailable) if the server considers itself degraded or unavailable
- One that uses a path of `/available-or-degraded-state` that returns an HTTP status code of 200 if the server considers itself available or degraded or an HTTP status code of 503 if the server considers itself unavailable

The server's assessment of its health state is based on the presence of any unavailable or degraded alert types that are active in the server. If the server currently has one or more unavailable alert types, then it considers itself unavailable. If the server does not have any unavailable alert types but has one or more degraded alert types, then it considers itself degraded. If the server does not have any unavailable or degraded alert types, then it considers itself available.

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. 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

Copyright © 2025 Ping Identity Corporation

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.

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.

If you followed the standard setup and configuration given in Getting started with PingAuthorize (tutorials), 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.

Copyright © 2025 Ping Identity Corporation

By default, this gauge does nothing. To begin using it, set the levels at which the gauge activates to reasonable values for your environment using `dsconfig`.

The following table explains the values you set for this gauge.

Value	Description
`minor-value`	This value, in milliseconds, represents a warning condition. An alarm is raised, but the server continues to operate as normal.
`major-value`	This value, in milliseconds, represents the point at which the server is considered DEGRADED.
`critical-value`	This value, in milliseconds, represents the point at which the server is considered UNAVAILABLE.

You can find the server's availability state by using an option discussed in Server status.

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.

Copyright © 2025 Ping Identity Corporation

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`.
`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.

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.

Copyright © 2025 Ping Identity Corporation

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.
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 and HTTP Processing (Percent) gauge, 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.

   1. For the `Endpoint Average Response Time (Milliseconds)` gauge, set `critical-value`.
   2. 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.

Available gauges

PingAuthorize makes the following gauges available. You can manage these gauges using the administrative console or the **`dsconfig`** tool.

Copyright © 2025 Ping Identity Corporation

Gauge name	Enabled by default	Description
Available File Descriptors	true	Monitors the number of file descriptors available to the server process. The server allows for an unlimited number of connections by default but is restricted by the file descriptor limit on the operating system. You can configure the number of file descriptors that the server uses by either setting the `NUM_FILE_DESCRIPTORS` environment variable or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_DESCRIPTORS=12345`. If you do not use either of these options, the server uses the default of 65535. Running out of available file descriptors can lead to unpredictable behavior and severe system instability.
Certificate Expiration (Days)	true	Monitors the expiration dates of key server certificates. A server certificate expiring can cause server unavailability, degradation, or loss of key server functionality. Replace certificates nearing the end of their validity as soon as possible. For more information about server certificates and how they are managed, see the **`status`** tool or **Status** in the administrative console.
CPU Usage (Percent)	true	Monitors server CPU use and provides an averaged percentage for the interval defined. The monitored resource is the host system's CPU, which does not include a resource identifier. If CPU use is high, check the server's current workload and other processes on the system and make any needed adjustments. Reducing the load on the system will lead to better response times.
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.

Copyright © 2025 Ping Identity Corporation

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.
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.
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`.
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.

Copyright © 2025 Ping Identity Corporation

Gauge name	Enabled by default	Description
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.
Strong Encryption Not Available	true	Indicates the JVM does not appear to support strong encryption algorithms, like 256-bit AES. The server will fall back to using weaker algorithms, like 128-bit AES. To enable support for strong encryption, update your JVM to a newer version that supports it by default; alternatively, install or enable the unlimited encryption strength jurisdiction policy files in your Java installation.
User Store Availability	true	Monitors availability of the SCIM user store. If the LDAP directory servers are unavailable, the "UserStoreAdapter" cannot forward requests. Also, the server cannot process SCIM requests or perform token owner lookups. Ensure that LDAP directory servers are available. For more information, see User Store Availability gauge.

Managing logging

PingAuthorize Server supports a rich set of log publishers to help you monitor configuration, debug, and error messages that occur during normal server processing.

Administrators can view the standard set of default log files and configure custom log publishers with pre-defined filtering with its own log rotation and retention policies.

Copyright © 2025 Ping Identity Corporation

Default PingAuthorize Server logs

PingAuthorize Server provides a standard set of default log files to monitor the server activity.

PingAuthorize Server logs

View this set of logs in the `PingAuthorize/logs` directory. The following default log files are available on PingAuthorize Server and are defined in the following table.

Log File	Description
`config-audit.log`	Records information about changes made to the PingAuthorize Server configuration in a format that can be replayed using the **`dsconfig`** tool.
`errors`	File-based Error Log. Provides information about warnings, errors, and significant events that occur during server processing.
`policy-decision`	Records decision responses that are received from the policy decision point (PDP). Learn more in Policy Decision logger.
`server.out`	Records anything written to standard output or standard error, which includes startup messages. If garbage collection debugging is enabled, then the information is written to `server.out`.
`server.pid`	Stores the server's process ID.
`server.status`	Stores the timestamp, a status code, and an optional message providing additional information on the server status.
`setup.log`	Records messages that occur during the initial configuration of PingAuthorize Server with the **`setup`** tool.
`tools`	Directory that holds logs for long running utilities **`import-ldif`**, **`export-ldif`**, **`backup`**, **`restore`**, and **`verify-index`**. Current and previous copies of the log are present in PingAuthorize Server.
`update.log`	Records messages that occur during a PingAuthorize Server update.

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.

Copyright © 2025 Ping Identity Corporation

Debug Trace logger

The Debug Trace logger records detailed information about the processing of HTTP requests and responses.

To enable the logger, run the following command:

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

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

The Debug Trace logger generates various types of log messages, providing insight into different aspects of request processing. The following sections provide two examples of information recorded by the logger.

Service cache logging

The Debug Trace logger can help determine if the policy decision point (PDP) retrieved service data from its cache or from a direct call to the service.

For services without caching enabled, the logger records a message like the following:

```
[21/Jun/2024:10:58:35.750 +0100] PDP INFO msg="<service-type> service call: name=example-service,
duration=1ms, configuredTimeout=2000ms
```

For services with caching enabled, the logger records a message like the following:

```
[21/Jun/2024:10:57:52.142 +0100] PDP INFO msg="Cache hit for <service-type> service: name=example-service,
cacheKey=FD45D66BE078EAF4EE365EA766460CBF"
```

Learn more about caching in Service caching.

Circuit breaker logging

When circuit breaker functionality is enabled for a service, the Debug Trace logger records a message each time the breaker transitions between states. Circuit breakers can be open (blocking service requests), half-open (allowing test service requests), or closed (allowing service requests).

For example, when a breaker opens, the logger records a message like the following:

```
[30/May/2025:16:45:31.608 -0500] PDP WARNING correlationID="72a606d1-8576-43e4-8e9d-97045d4885b3"
decisionRequestId="499a6253-ea46-435d-9acc-4fb7a149f139" msg="Circuit breaker opened for service: Example
service, with ID: ca1fc8ec-69dc-4b90-8676-9a7130e2b72e"
```

Learn more about circuit breakers in Connecting a service.

Debug logger

The Debug logger records debugging information that a developer might find useful.

Copyright © 2025 Ping Identity Corporation

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`.

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.

Copyright © 2025 Ping Identity Corporation

Policy statements

If the policy contains a statement, it is logged after the policy-decision response JSON. Statements feature 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. Learn more in Configuring the Decision Response View.

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.

> **Note**
>
> Configuration for attribute logging will also apply to attributes evaluated as part of policy query requests. Specifying the `attributes` query permutation view will override the Policy Decision Service configuration and log all evaluated attributes. Learn more in Configuring the query permutation view.

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 logged only 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.

Copyright © 2025 Ping Identity Corporation

Types of log publishers

PingAuthorize Server provides several classes of log publishers for parsing, aggregating, and filtering information events that occur during normal processing in the server.

The primary types of log publishers include:

- Debug log publishers
- Error log publishers

Each type has multiple subtypes of log files based on the log server type. For more information on logging, see Enable detailed logging.

Debug log publishers

Debug log publishers provide information about warnings, errors, or significant events that occur within the server.

Debug log type	Description
Debug loggers	***Debug ACI Logger*** Stores debug information on access control instruction (ACI) evaluation for any request operations against the server. You can find this log publisher in the `<PingAuthorize_install>/logs/debug-aci` directory. ***Server SDK Extension Debug Logger*** Provides simplified access to debug messages generated by the Server SDK extension. You can find this log publisher in the `<PingAuthorize_install>/logs/server-sdk-extension-debug` directory.

Error log publishers

The error log reports errors, warnings, and informational messages about events that occur during the course of the server's operation.

Copyright © 2025 Ping Identity Corporation

Error log type	Description
File-based error logs	There are several types of file-based error logs: ***Error log*** Publishes error messages to the file system. Enabled by default. You can find this log publisher in the `<PingAuthorize_install>/logs/errors` directory. ***JSON Error Logger*** Publishes JSON-formatted error log messages to the file system. You can find this log publisher in the `<PingAuthorize_install>/logs/errors.json` directory. ***Console JSON Access Logger*** Publishes JSON-formatted access log messages to the JVM's standard output or standard error stream. **Note** This log publisher is only recommended when the server is run in no-detach mode and is best suited for use in Docker or other containers that can capture log content written to standard output or standard error. ***Third-Party Error Log Publisher*** Publishes error log messages using a custom log publisher created using the Server SDK. ***Third-Party File-Based Error Log Publisher*** Publishes error log messages to a file using a custom log publisher created using the Server SDK, including enhanced support for log file rotation and retention.
Syslog-based error logs	Publishes error messages to a syslog port.

Viewing the list of log publishers

View the list of log publishers on PingAuthorize Server using the **`dsconfig`** tool.

About this task

Note

You must specifically configure the Java Database Connectivity (JDBC), syslog, and admin alert log publishers using **`dsconfig`** before they appear in the list of log publishers.

Steps

- To view the log publishers, run **`dsconfig`** with the **`list-log-publishers`** subcommand.

  Example:

  ```
  $ bin/dsconfig list-log-publishers
  ```

Copyright © 2025 Ping Identity Corporation

Result:

```
Log Publisher                                : Type              : enabled
---------------------------------------------:-------------------:--------
Debug ACI Logger                             : debug-access      : false
File-Based Access Logger                     : file-based-access : true
File-Based Audit Logger                      : file-based-audit  : false
File-Based Debug Logger                      : file-based-debug  : false
File-Based Error Logger                      : file-based-error  : true
```

Enabling or disabling a default log publisher

You can enable or disable any log publisher available on PingAuthorize Server using the **`dsconfig`** tool.

About this task

By default, the following loggers are disabled and should only be enabled when troubleshooting an issue on the server:

- File-Based Audit Logger
- File-Based Debug Logger

Steps

- To enable an access log publisher, run **`dsconfig`** with the **`set-log-publisher-prop`** subcommand.

  Example:

  In this example, the Console JSON LDAP Access Logger is enabled, which publishes JSON-formatted access log messages to the JVM's original standard output or standard error stream.

  ```
  $ bin/dsconfig set-log-publisher-prop \
    --publisher-name "Console JSON LDAP Access Logger" \
    --set enabled:true
  ```

About log compression

The server supports the ability to compress log files as they are written.

This feature can significantly increase the amount of data that can be stored in a given amount of space so that log information can be kept for a longer period of time.

Because of the inherent problems with mixing compressed and uncompressed data, compression can only be enabled at the time the logger is created. Compression cannot be turned on or off when the logger is configured. Because of problems in trying to append to an existing compressed file, if the server encounters an existing log file at startup, it rotates that file and begin a new one rather than attempting to append to the previous file.

Compression is performed using the standard gzip algorithm, so compressed log files can be accessed using readily available tools. The **`summarize-access-log`** tool can also work directly on compressed log files rather than requiring them to be decompressed first.

Copyright © 2025 Ping Identity Corporation

However, because it can be useful to have a small amount of uncompressed log data available for troubleshooting purposes, administrators using compressed logging might want to have a second logger defined that does not use compression and has rotation and retention policies that minimizes the amount of space consumed by those logs while still making them useful for diagnostic purposes without the need to decompress the files before examining them.

Configure compression by setting the `compression-mechanism` property to have the value of `gzip` when creating a new logger.

About log signing

The server supports the ability to cryptographically sign a log to ensure that it has not been modified in any way.

For example, financial institutions require audit logs for all transactions to check for correctness. Tamper-proof files are needed to ensure that these transactions can be properly validated and ensure that they have not been modified by any third-party entity or internally by unscrupulous employees.

Use the **`dsconfig`** tool to enable the `sign-log` property on a log publisher to turn on cryptographic signing.

When enabling signing for a logger that already exists and was enabled without signing, the first log file is not completely verifiable because it still contains unsigned content from before signing was enabled. Only log files whose entire content was written with signing enabled are considered completely valid. For the same reason, if a log file is still open for writing, then signature validation does not indicate that the log is completely valid because the log doesn't include the necessary end signed content indicator at the end of the file.

To validate log file signatures, use the **`validate-file-signature`** tool provided in the `bin` directory of the server or the `bat` directory for Windows systems.

After you have enabled this property, you must disable and then re-enable the log publisher for the changes to take effect.

About encrypting log files

The server lets you encrypt log files as they are written.

The **`encrypt-log`** configuration property controls whether encryption is enabled for the logger. Enabling encryption causes the log file to have an **`.encrypted`** extension. If both encryption and compression are enabled, the extension is **`.gz.encrypted`**. Any change that affects the name used for the log file could prevent older files from getting properly cleaned up.

Like compression, encryption can only be enabled when the logger is created. Encryption cannot be turned on or off after the logger is configured. For any log file that is encrypted, enabling compression is also recommended to reduce the amount of data that needs to be encrypted. This reduces the overall size of the log file. The **`encrypt-file`** tool or custom code, using the LDAP SDK's **`com.unboundid.util.PassphraseEncryptedInputStream`**, is used to access the encrypted data.

To enable encryption, at least one encryption settings definition must be defined in the server. Use the one created during setup, or create a new one with the **`encryption-settings create`** command. By default, the encryption is performed with the server's preferred encryption settings definition.

To explicitly specify which definition should be used for the encryption, set the **`encryption-settings-definition-id`** property with the ID of that definition. You should set the encryption settings definition to be created from a passphrase so that the file can be decrypted by providing that passphrase even if the original encryption settings definition is no longer available. You can also create a randomly generated encryption settings definition, but the log file can only be decrypted using a server instance that has that encryption settings definition.

Copyright © 2025 Ping Identity Corporation

When using encrypted logging, a small amount of data might remain in an in-memory buffer until the log file is closed. The encryption is performed using a block cipher, and it cannot write an incomplete block of data until the file is closed. This is not an issue for any log file that is not being actively written.

To examine the contents of a log file that is being actively written, use the **`rotate-log`** tool to force the file to be rotated before attempting to examine it.

Configuring log signing

Configure log signing for a log publisher.

Steps

1. To enable log signing for a log publisher, use **`dsconfig`**.

   Example:

   In this example, the `sign-log` property is set on the File-based Audit Log Publisher.

   ```
   $ bin/dsconfig set-log-publisher-prop --publisher-name "File-Based Audit Logger" \
     --set sign-log:true
   ```

2. Disable and then re-enable the log publisher for the change to take effect.

   Example:

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

Validating a signed file

The server provides a tool, **`validate-file-signature`**, that checks if a file has not been tampered with in any way.

Steps

- Run the **`validate-file-signature`** tool to check if a signed file has been tampered with.

  Example:

  For this example, assume that the `sign-log` property was enabled for the File-Based Audit Log Publisher.

  ```
  $ bin/validate-file-signature --file logs/audit
  ```

  Result:

  ```
  All signature information in file 'logs/audit' is valid
  ```


Copyright © 2025 Ping Identity Corporation

Note

If any validations errors occur, you will see a message similar to the one as follows.

```
One or more signature validation errors were encountered
while validating the contents of file 'logs/audit':
* The end of the input stream was encountered without
  encountering the end of an active signature block.
  The contents of this signed block cannot be trusted
  because the signature cannot be verified
```

Configuring log file encryption

Configure log file encryption for a log publisher.

Steps

1. To enable encryption for a log publisher, use `dsconfig`.

   Example:

   In this example, the File-based Access Log Publisher `"Encrypted Access"` is created, compression is set, and rotation and retention policies are set.

   ```
   $ bin/dsconfig create-log-publisher-prop --publisher-name "Encrypted Access" \
     --type file-based-access \
     --set enabled:true \
     --set compression-mechanism:gzip \
     --set encryption-settings-definition-id:332C846EF0DCD1D5187C1592E4C74CAD33FC1E5FC20B726CD301CDD2B3FFBC2B \
     --set encrypt-log:true \
     --set log-file:logs/encrypted-access \
     --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
     --set "rotation-policy:Size Limit Rotation Policy" \
     --set "retention-policy:File Count Retention Policy" \
     --set "retention-policy:Free Disk Space Retention Policy" \
     --set "retention-policy:Size Limit Retention Policy"
   ```

2. Decrypt and decompress the file.

   Example:

   ```
   $ bin/encrypt-file --decrypt \
     --decompress-input \
     --input-file logs/encrypted-access.20180216040332Z.gz.encrypted \
     --output-file decrypted-access
   ```

   Result:

   +

Copyright © 2025 Ping Identity Corporation

```
Initializing the server's encryption framework...Done
Writing decrypted data to file '/ds/Data-Sync/decrypted-access' using a
key generated from encryption settings definition
'332c846ef0dcd1d5187c1592e4c74cad33fc1e5fc20b726cd301cdd2b3ffbc2b'
Successfully wrote 123,456,789 bytes of decrypted data
```

Creating new log publishers

The server provides customization options to help you create your own log publishers with the **`dsconfig`** command.

When you create a new log publisher, you must also configure the log retention and rotation policies for each new publisher.

For more information, see Configuring Log Rotation and Configuring Log Retention.

Creating a new log publisher

Steps

1. Use the **`dsconfig`** command in non-interactive mode to create and configure the new log publisher.

   Example:

   This example shows how to create a logger that only logs disconnect operations.

   ```
   $ bin/dsconfig create-log-publisher \
     --type file-based-access --publisher-name "Disconnect Logger" \
     --set enabled:true \
     --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
     --set "rotation-policy:Size Limit Rotation Policy" \
     --set "retention-policy:File Count Retention Policy" \
     --set log-connects:false \
     --set log-requests:false --set log-results:false \
     --set log-file:logs/disconnect.log
   ```

   Note

   To configure compression on the logger, add the `--set compression-mechanism: gzip` option to the previous command.
   Because compression cannot be disabled or turned off after configured for the logger, plan carefully to determine your logging requirements, including log rotation and retention with regards to compressed logs.

2. View log publishers with the following command.

   ```
   $ bin/dsconfig list-log-publishers
   ```


Copyright © 2025 Ping Identity Corporation

Creating a log publisher using dsconfig interactive command-line mode

Steps

1. In the command line, enter `bin/dsconfig`.
2. Authenticate to the server by following the prompts.
3. In the main menu, select the option to configure the log publisher.
4. In the `Log Publisher menu`, select the option to create a new log publisher.
5. Select the Log Publisher type.

   For this example, select **File-Based Access Log Publisher**.
6. Enter a name for the log publisher.
7. Enable it.
8. Enter the path to the log file relative to the PingAuthorize server root.

   For this example, `logs/disconnect.log`.
9. Select the rotation policy to use for this log publisher.
10. Select the retention policy to use for this log publisher.
11. In the `Log Publisher Properties menu`, select the option for `log-connects:false`, `log-disconnects:true`, `log-requests:false`, and `log-results:false`.
12. Enter `f` to apply the changes.

Configuring log rotation

PingAuthorize Server allows you to configure the log rotation policy for the server.

About this task

When any rotation limit is reached, the server rotates the current log and starts a new log.

If you create a new log publisher, you must configure at least one log rotation policy.

You can select the following properties:

Time Limit Rotation Policy

Rotates the log based on the length of time since the last rotation. Default implementations are provided for rotation every 24 hours and every 7 days.

Fixed Time Rotation Policy

Rotates the logs every day at a specified time (based on 24-hour time). The default time is 2359.

Copyright © 2025 Ping Identity Corporation

Size Limit Rotation Policy

Rotates the logs when the file reaches the maximum size for each log. The default size limit is 100 MB.

Never Rotate Policy

Used in a rare event that does not require log rotation.

Steps

- Use `dsconfig` to modify the log rotation policy for the access logger.

  Example:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --remove "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --add "rotation-policy:7 Days Time Limit Rotation Policy"
```

Configuring log rotation listeners

The server provides two log file rotation listeners, the copy log file rotation listener and the summarize log file rotation listener, which you can enable with a log publisher.

About this task

Log file rotation listeners allow the server to perform a task on a log file as soon as it has been rotated out of service. Custom log file listeners can be created with the Server SDK.

The copy log file rotation listener can be used to compress and copy a recently-rotated log file to an alternate location for long-term storage. The original rotated log file is subject to deletion by a log file retention policy, but the copy is not automatically removed.

The summarize log file rotation listener invokes the `summarize-access-log` tool on a recently-rotated log file and writes its output to a file in a specified location. This provides information about the number and types of operations processed by the server, processing rates and response times, and other useful metrics. Use this with access loggers that log in a format that is compatible with the `summarize-access-log` tool, including the `file-based-access` logger type.

Steps

- Use the following command to create a new copy log file rotation listener.

```
$ dsconfig create-log-file-rotation-listener \
  --listener-name "Copy on Rotate" \
  --type copy \
  --set enabled:true \
  --set copy-to-directory:</path/to/archive/directory>  \
  --set compress-on-copy:true
```


Copyright © 2025 Ping Identity Corporation

> **Note**
>
> The path specified by the `copy-to-directory` property must exist, and the file system containing that directory must have enough space to hold all of the log files that are written there. The server automatically monitors free disk space on the target file system and generates administrative alerts if the amount of free space gets too low.

- Use the following command to create a new summarize log file rotation listener.

```
$ dsconfig create-log-file-rotation-listener \
  --listener-name "Summarize on Rotate" \
  --type summarize \
  --set enabled:true \
  --set output-directory:</path/to/summary/directory>
```

> **Note**
>
> The summary output files have the same name as the rotated log file, with an extension of `.summary`. If the `output-directory` property is specified, the summary files are written to that directory. If not specified, files are placed in the directory in which the log files are written.
> As with the copy log file rotation listener, summary files are not automatically deleted. Although files are generally small in comparison to the log files themselves, make sure that enough space is available in the specified storage directory. The server automatically monitors free disk space on the file system to which the summary files are written.

Configuring log retention

The server allows you to configure the log retention policy for each log on the server.

About this task

When a retention limit is reached, the server removes the oldest archived log before creating a new log. Log retention is only effective if you have a log rotation policy in place.

If you create a new log publisher, you must configure at least one log retention policy.

File Count Retention Policy

Sets the number of log files you want the server to retain. The default file count is 10 logs. If the file count is set to 1, then the log continues to grow indefinitely without being rotated.

Free Disk Space Retention Policy

Sets the minimum amount of free disk space. The default free disk space is 500 MB.

Size Limit Retention Policy

Sets the maximum size of the combined archived logs. The default size limit is 500 MB.

Copyright © 2025 Ping Identity Corporation

Time Limit Retention Policy

Sets the maximum length of time that rotated log files should be retained.

Custom Retention Policy

Create a new retention policy that meets your server's requirements. This requires developing custom code to implement the desired log retention policy

Never Delete Retention Policy

Used in a rare event that does not require log deletion.

Steps

- Use `dsconfig` to modify the log retention policy for the access logger.

  Example:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

Configuring filtered logging

PingAuthorize Server provides a mechanism to filter access log messages based on specific criteria.

About this task

You can use the filtered log with a custom log publisher to create and generate your own custom logs. Adding new filtered logs and associate publishers doesn't change the behavior of any existing logs. For example, adding a new log that only contains operations that were unsuccessful doesn't result in those operations being removed from the default access log.

The following example shows how to create a set of criteria that matches any operation that didn't complete successfully. It then explains how to create a custom access log publisher that logs only operations matching that criteria.

> **Note**
>
> This log does not include messages for connects or disconnects, and only a single message is logged per operation. This message contains both the request and result details.

To run log filtering based on any operation result, such as result code, processing time, and response controls, turn off request logging and set the `include-request-details-in-result-messages` property to `TRUE`.

Because filtering based on the results of an operation can't be done until the operation completes, the server has no idea whether to log the request. Therefore, it might log request messages but not log any result messages. If you can only log result messages and include request details in the result messages, then only messages for operations that match the result criteria are logged. All pertinent information about the corresponding requests are included.

Steps

1. Use the `dsconfig` command in non-interactive mode to create a result criteria object set to `failure-result-codes`, a predefined set of result codes that indicate when an operation didn't complete successfully.

Copyright © 2025 Ping Identity Corporation

Example:

```
$ bin/dsconfig create-result-criteria --type simple \
  --criteria-name "Failed Operations" --set result-code-criteria:failure-result-codes
```

2. Use `dsconfig` to create the corresponding log publisher that uses the result criteria.

> **Note**
>
> The log rotation and retention policies are also set with this command.

Example:

```
$ bin/dsconfig create-log-publisher \
  --type file-based-access \
  --publisher-name "Filtered Failed Operations" \
  --set enabled:true \
  --set log-connects:false \
  --set log-disconnects:false \
  --set log-requests:false \
  --set "result-criteria:Failed Operations" \
  --set log-file:logs/failed-ops.log \
  --set include-request-details-in-result-messages:true \
  --set "rotation-policy:7 Days Time Limit Rotation Policy" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

3. View the `failed-ops.log` in the `logs` directory and verify that only information about failed operations was written to it.

Managing the File-Based Error Log Publisher

The Error Log reports errors, warnings, and informational messages about events that occur during the course of PingAuthorize Server's operation.

Each entry in the error log records the following properties (some are disabled by default and must be enabled):

Timestamp

Displays the date and time of the operation in the following format: `DD/Month/ YYYY:HH:MM:SS <offset from UTC time>`

Category

Specifies the message category that is loosely based on the server components.

Severity

Specifies the message severity of the event, which defines the importance of the message in terms of major errors that need to be quickly addressed. The default severity levels are:

- `fatal-error`
- `notice`
- `severe-error`

Copyright © 2025 Ping Identity Corporation

- `severe-warning`

Message ID

Specifies the numeric identifier of the message.

Message

Stores the error, warning, or informational message.

Modifying the File-Based Error Logs

Steps

- To modify the default File-Based Error Log, use **`dsconfig`**.

  Example:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Error Logger" \
  --set include-product-name:true --set include-instance-name:true \
  --set include-startup-id:true
```

Managing the Syslog-Based Error Log Publisher

PingAuthorize Server supports a Syslog-Based Error Log Publisher using the same mechanism as the Syslog-Based Access Log Publisher.

Configure the error logger using the **`dsconfig`** tool.

Syslog error mapping

PingAuthorize Server automatically maps error log severities to the syslog severities. The following mappings are used.

Error Log Severities	Syslog Severity
FATAL_ERROR,0	Syslog Emergency
SEVERE_ERROR,1	Syslog Alert
SEVERE_WARNING,2	Syslog Critical
MILD_ERROR,3	Syslog Error
MILD_WARNING,4	Syslog Warn
NOTICE,5	Syslog Notice
INFORMATION,6	Syslog Info

Copyright © 2025 Ping Identity Corporation

Error Log Severities	Syslog Severity
DEBUG,7	Syslog Debug

Configuring a Syslog-Based Error Log Publisher

About this task

Configure a Syslog-based Error Log Publisher using the `dsconfig` tool. You should use syslog locally on `localhost` and use `syslog-ng` to send the data packets over the UDP protocol.

Because syslog implementations differ by vendor, review your particular vendor's syslog configuration.

Steps

- Use `dsconfig` to create a log publisher of type `syslog-based-error`.

  Example:

  In this example, set the syslog facility to `4` for security and authorization messages.

  ```
  $ bin/dsconfig create-log-publisher --publisher-name "syslog-error" \
    --type syslog-based-error --set syslog-facility:4 --set enabled:true
  ```

Creating File-Based Debug Log Publishers

PingAuthorize Server provides a File-Based Debug Log Publisher that can be configured when troubleshooting a problem that could occur during server processing.

Because the debug data might be too large to maintain during normal operations, the Debug Log Publisher must be specifically configured and enabled. The Debug Log reports the following types of information:

- Exception data thrown by the server.
- Data read or written to network clients.
- Data read or written to the database.
- Access control or password policy data made within the server.

Use the `dsconfig` tool to create a debug log publisher.

> **Note**
>
> You should only create a debug logger when troubleshooting a problem because of the voluminous output PingAuthorize Server generates.

Copyright © 2025 Ping Identity Corporation

Creating a File-Based Debug Log Publisher

Steps

- To create the debug log publisher, use `dsconfig`.

  The `log-file` property (required) sets the debug log path. You must also specify the rotation and retention policy for the debug log.

  Example:

```
$ bin/dsconfig create-log-publisher \
  --publisher-name "File-Based Debug Logger" \
  --type file-based-debug \
  --set enabled:true \
  --set log-file:/logs/debug \
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --set "rotation-policy:Size Limit Rotation Policy" \
  --set "retention-policy:File Count Retention Policy"
```

Deleting a File-Based Debug Log Publisher

Steps

- To create the debug log publisher, use `dsconfig`.

  Example:

```
$ bin/dsconfig delete-log-publisher \
  --publisher-name "File-Based Debug Logger"
```

Managing monitoring

PingAuthorize Server provides a flexible monitoring framework that enables you to track server performance.

This framework exposes its monitoring information under the `cn=monitor` entry and provides interfaces through the administrative console and over LDAP.

Additionally, PingAuthorize Server provides the Periodic Stats Logger and the ability to configure a StatsD monitoring endpoint. You can send performance metrics to Splunk, Prometheus, and other StatsD monitoring tools.

Monitoring with the administrative console

Administrators can use the administrative console to configure the server. The console also provides a status option that accesses the server's monitor content.

Before you begin

Ensure that the server is running.

About this task

Copyright © 2025 Ping Identity Corporation

To view the **Monitor Dashboard**:

Steps

1. In your browser, enter http://<*server-name*>:8443/console.
2. Enter the root user distinguished name (DN) and password. Click **Login**.
3. In the top level navigation menu, select **Status**.
4. On the **Status** page, click the **Monitors** tab.

Result

The **Monitors** page opens, and you can monitor your server using the administrative console.

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> On this menu, you can also change the format from `csv` to `json`.

10. Enter `f` to save and apply the configuration.

Configuring the Periodic Stats Logger non-interactively

About this task

To configure PingAuthorize Server to log to a JSON-formatted log file with various metrics, including LDAP operation counts and GC activity, run the following command non-interactively. The metrics will be written to `logs/dsstats.json` in the server root.

```
dsconfig set-plugin-prop \
  --plugin-name "Stats Logger" \
  --set enabled:true \
  --set log-file:logs/dsstats.json \
  --set log-file-format:json \
  --set local-db-backend-info:extended \
  --set header-prefix-per-column:true \
  --set empty-instead-of-zero:false \
  --set per-application-ldap-stats:per-application-and-aggregate
```

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.

   Learn more in Enabling the Stats Logger.

Copyright © 2025 Ping Identity Corporation

2. Set the `included-http-servlet-stat` property.

   Learn more in Configuring multiple Periodic Stats Loggers.

Sending Metrics with the Periodic Stats Logger and the Splunk Universal Forwarder

You can use the Splunk Universal Forwarder to monitor the `logs/dsstats.json` file and forward the metrics to Splunk.

About this task

The Periodic Stats Logger plugin writes PingAuthorize Server metrics to a file in the server root.

Steps

- Under the Splunk Forwarder application files, update the `etc/apps/search/local/inputs.conf` file with the following configuration for `dsstats.json`:

```
<path/to/PingAuthorize>/logs/dsstats.json]
  sourcetype=log2metrics_json
  index=<pazmetrics>
  disabled=false
  host=<paz1>
```

> **Tip**
>
> You can customize the index name and configure the host value to label where the metrics are being forwarded from.

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

Copyright © 2025 Ping Identity Corporation

- 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 StatsD metrics to Splunk.

Copyright © 2025 Ping Identity Corporation

- 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 StatsD 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.

Monitoring server metrics with Prometheus

Prometheus is an open-source monitoring application that can be used to track numeric metrics over time. It uses `HTTP GET` requests to retrieve the metric data in the OpenMetrics text-based format.

The server includes an HTTP servlet extension that can publish the values of a configured set of numeric monitor attributes in this format so that they can be consumed by a Prometheus server.

Enabling Prometheus support in the server

The server is preconfigured with an HTTP servlet extension that can publish a wide variety of metrics for use by Prometheus.

To enable the servlet extension, add it to an HTTP or HTTPS connection handler, and either disable and re-enable the connection handler or restart the server. For example:

```
dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --add "http-servlet-extension:Prometheus Monitoring" \
  --set enabled:false

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


Copyright © 2025 Ping Identity Corporation

The servlet extension offers a number of configuration properties that can be used to customize its behavior. They include:

`base-context-path`

The path that the Prometheus server can use to consume any published metrics. By default, this is `/metrics`.

`include-instance-name-label`

Indicates whether all published metrics should include an `instance` label whose value is the name of the server instance.

`include-product-name-label`

Indicates whether all published metrics should include a `product` label whose value is the name of the server product, such as `Ping Identity Authorize Server` for PingAuthorize Server.

`include-location-name-label`

Indicates whether all published metrics should include a `location` label whose value is the name of the configured location for the server instance.

`always-include-monitor-entry-name-label`

Indicates whether all published metrics should include a `monitor_entry` label whose value is the name of the monitor entry (the value of the `cn` attribute) from which the metric was taken.

> **Note**
>
> Even if this property is set to `false`, the server will still add this label to metrics in cases where there are multiple monitor entries with the same object class.

`include-monitor-object-class-name-label`

Indicates whether all published metrics should include a `monitor_object_class` label whose value is the name of the object class for the monitor entry from which the metric was taken.

`include-monitor-attribute-name-label`

Indicates whether all published metrics should include a `monitor_attribute` label whose value is the name of the monitor attribute from which the metric was taken.

`label-name-value-pair`

An optional set of name-value pairs for labels that should be included in all metrics published by the server. If provided, each item should use an equal sign to separate the label name from the value, such as `label_name=label_value`.

> **Important**
>
> Label names must:
>
> - Start with a letter or underscore
> - Only contain letters, digits, and underscores
> - Be unique
>
> Label values can be any non-empty string.

Copyright © 2025 Ping Identity Corporation

After the servlet extension has been enabled, you can verify that the metric information is available by retrieving it with a simple `HTTP GET` request using a web browser or a command-line tool like `curl` or `wget`.

For example, to retrieve the Prometheus metrics from the server `paz.example.com` using an HTTPS connection handler listening on port 8443 and the `default base-context-path` value of `/metrics`, use a URL of `https://paz.example.com:8443/metrics`.

Customizing published metrics

The server is preconfigured with a wide variety of definitions that publish the values of specified monitor attributes as metrics that can be consumed by Prometheus. However, you can also configure additional metrics by creating Prometheus monitor attribute metrics.

Each of these definitions includes the following configuration properties:

`metric-name`

The name of the metric as it should be published for use in Prometheus. The metric name must start with a letter or an underscore and must contain only letters, digits, and underscores. This is required, and all metric definitions must have a unique name.

`monitor-attribute-name`

The name of the monitor attribute whose value should be published as a Prometheus metric. This is required, and only single-valued attributes are supported. The value must be an integer or floating-point number.

`monitor-object-class-name`

The name of the object class for the monitor entry that contains the attribute to publish as a Prometheus metric. This is required.

`metric-type`

The data type for the metric to be published. This is required, and the value should be one of the following:

- `counter` — Indicates that the value is one that increases over time as certain events occur in the server. This can include values that simply increment each time the event occurs, such as counting the number of connections accepted by the server since it started, but it can also include values that increase by larger amounts, such as counting the total number of bytes transferred in responses.
- `gauge` — Indicates that the value is one that can go up or down over time and whose value represents the current state of some aspect of the server, such as the number of connections that are currently established.

`filter`

An optional search filter that can be used to indicate that the metric should only be published for monitor entries that match this filter. If this is not specified, then the metric is published for any monitor entry with the specified attribute and object class.

`metric-description`

An optional human-readable string that describes the purpose for the metric or that provides some other additional information about it.

Copyright © 2025 Ping Identity Corporation

`label-name-value-pair`

An optional set of name-value pairs for labels that should be included in the definition for the metric. If provided, each item should use an equal sign to separate the label name from the value, such as `label_name=label_value`. Label names must start with a letter or underscore and must only contain letters, digits, and underscores. Label values can be any non-empty string.

> **Note**
>
> Metric-specific labels override server-wide labels, so if a metric-specific label is defined with the same name as a label that would otherwise be applied to all metrics, then the metric-specific value will be used instead of the server-wide value.

For example, to create a Prometheus metric that exposes the value of the `currentConnections` attribute of the general monitor entry, you could use a configuration change like the following:

```
dsconfig create-prometheus-monitor-attribute-metric \
  --extension-name "Prometheus Monitoring" \
  --metric-name general_monitor_current_connections \
  --set monitor-attribute-name:currentConnections \
  --set monitor-object-class-name:paz-general-monitor-entry \
  --set metric-type:gauge \
  --set "metric-description:The number of connections currently established"
```

If you want to remove any of the metrics that are included in the out-of-the-box configuration, then remove the definition for that metric from the server configuration.

Consuming metrics with Prometheus

After the server has been configured to publish its metrics, you can configure Prometheus to consume them.

The basic process is to update the server's `prometheus.yml` file to add a `job` element to the `scrape_configs` section for the server from which the metrics should be retrieved, as in the following example:

```
- job_name: "paz.example.com:8443"
  metrics_path: "/metrics"
  scheme: "https"

  tls_config:
    ca_file: paz.example.com-ca-certificate.pem

  static_configs:
    - targets: ["paz.example.com:8443"]
```

See the Prometheus documentation for complete details.

Copyright © 2025 Ping Identity Corporation

Managing notifications and alerts

PingAuthorize Server provides delivery mechanisms for administrative alerts using SMTP or SNMP in addition to standard error logging.

Alerts and events reflect state changes within the server that might be of interest to a user or monitoring service. Notifications are typically the delivery of an alert or event to a user or monitoring service.

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.
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

Copyright © 2025 Ping Identity Corporation

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

Copyright © 2025 Ping Identity Corporation

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 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. Learn more in Configuring HTTP connection handlers in the PingDirectory documentation.

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.

Copyright © 2025 Ping Identity Corporation

- 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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

```
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.

Copyright © 2025 Ping Identity Corporation

> **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.
`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.
`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.

Copyright © 2025 Ping Identity Corporation

Tool	Description
`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.
`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.

Copyright © 2025 Ping Identity Corporation

Tool	Description
`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.
`make-ldif`	Generate LDIF data based on a definition in a template file. See the server's `config/MakeLDIF` directory for example template files. In particular, the `examples-of-all-tags.template` file shows how to use all of the tags for generating values.
`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.
`prepare-external-store`	Prepare a PingAuthorize Server and an external server for communication.
`reload-http-connection-handler-certificates`	Reload HTTPS Connection Handler certificates.

Copyright © 2025 Ping Identity Corporation

Tool	Description
`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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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
```


Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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.
`--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.

Copyright © 2025 Ping Identity Corporation

Option	Description
`--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.

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.

Copyright © 2025 Ping Identity Corporation

Directories, files, and tools	Description
`config`	Stores the configuration files for the backends (admin, config) as well as the directories for messages, schema, tools, and updates.
`docs`	Provides the product documentation.
`extensions`	Stores Server SDK extensions.
`ldif`	Serves as the default location for LDIF exports and imports.
`legal`	Stores any legal notices for dependent software used with 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.

Copyright © 2025 Ping Identity Corporation

Directories, files, and tools	Description
`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 Policy Editor folders

The following table describes the contents of the Policy Editor distribution file:

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 Policy Editor.
`build-info.txt`	Contains build and version information for the Policy Editor.
`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. To make a custom Spring Expression Language (SpEL) resolver available to the Policy Editor, add the resolver's `.jar` library to the `/extensions` subdirectory and add the SpEL Java class to the allow list. For example, `PingAuthorize-PAP/lib/extensions/addl-spel-classes.jar`.
`logs`	Stores log files for the Policy Editor.
`policy-backup`	Stores H2 policy database backups when such backups are enabled.
`resource`	Stores supporting files such as policy snapshots.

Copyright © 2025 Ping Identity Corporation

PingAuthorize Policy Administration Guide

PingIdentity.

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 credential:

   - Username: `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, when, where, and why.

4. Define your policies and policy sets.

   Build your policies to reflect your business needs.

5. Test policies 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.

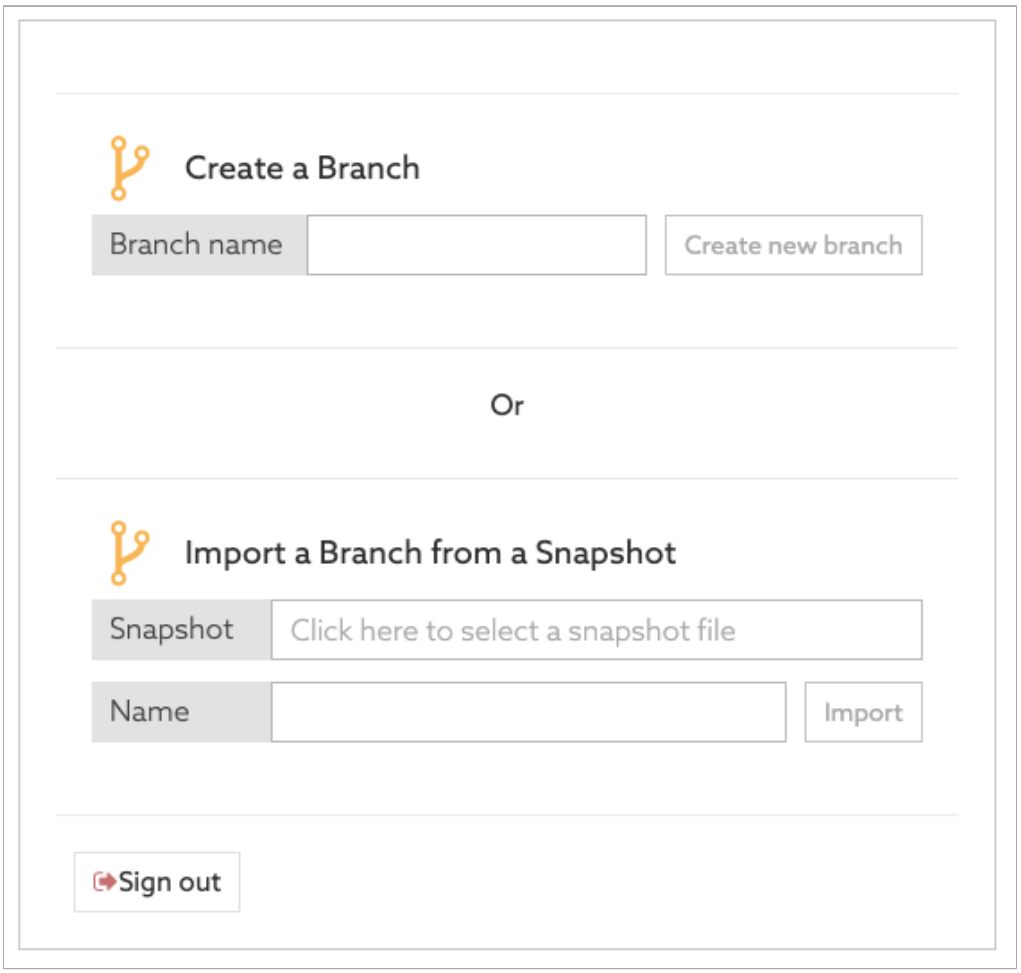
   This creates a file that you can deploy to PingAuthorize Server instances across multiple environments.

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.

Copyright © 2025 Ping Identity Corporation

The PingAuthorize Policy Editor embraces similar principles to general software source control. 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.



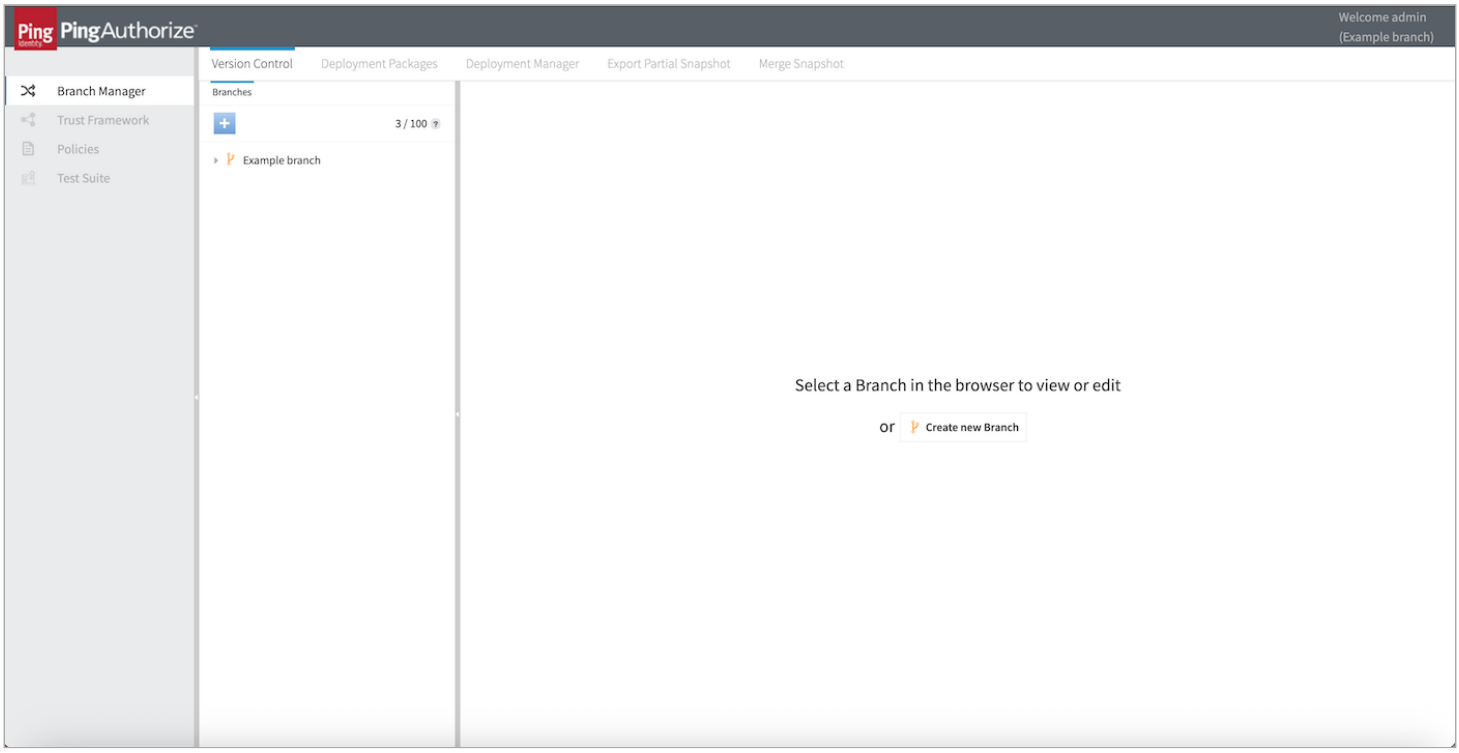
Version control (Branch Manager)

The Policy Editor Branch Manager applies version control principles to policy administration to streamline policy updates, strengthen governance, and improve policy stability.

The Branch Manager's capabilities for managing policy branches, commits, snapshots, and deployment packages enable you to:

- Maintain separate root branches to support parallel policy development across environments like development, QA, and production.
- Save sets of changes as distinct commits, creating a detailed, chronological history of what was changed, who changed it, and when and why they made the change. This history is useful for auditing, troubleshooting, and understanding policy evolution.
- Create structured workflows around initial policy creation, iterative policy development, approvals, and policy deployment.
- Quickly revert to a previously known, stable version of your policies if a newly deployed change causes issues.

Copyright © 2025 Ping Identity Corporation



Learn more about version control with the Branch Manager:

- Managing branches
- Branch changes and history
- Deployment packages

Managing branches

Branches allow multiple streams of policy updates to occur in parallel without interference.

A branch is an independent line of policy development. You begin this development by creating a top-level branch for your main development line or a specific environment, like QA or production. For specific tasks, such as developing a new feature, create subbranches from a specific point (commit) in an existing branch.

Learn more about branch operations:

- Creating a branch
- Merging branches
- Deleting a branch

Creating a branch

Create a branch to start an independent line of policy development. You can create a branch at the root level, import a snapshot as a new branch, or base a subbranch on a commit in an existing branch.

Copyright © 2025 Ping Identity Corporation

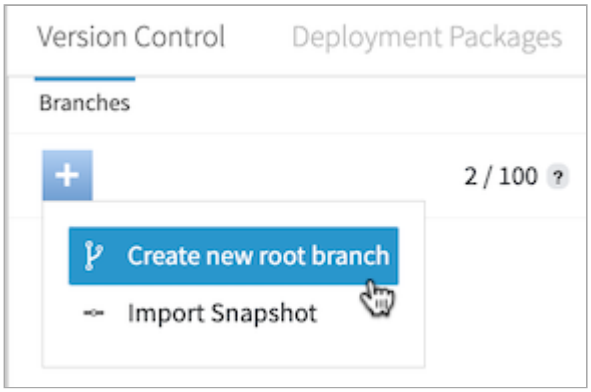
Creating a root branch

Create a root branch to begin a new line of policy development from scratch.

> **Important**
>
> You can create up to 100 root branches in the Policy Editor, including system and self-governance branches that might not be visible to the user. The Branch Manager displays a visual counter showing how many branches have been created out of the 100-branch limit.

Steps

1. In the Policy Editor, go to **Branch Manager > Version Control**.
2. From the + menu, select **Create new root branch**.



3. Enter a name for the new branch.

> **Note**
>
> Branch names must be unique.

4. Click **Save Branch**.

Importing a snapshot

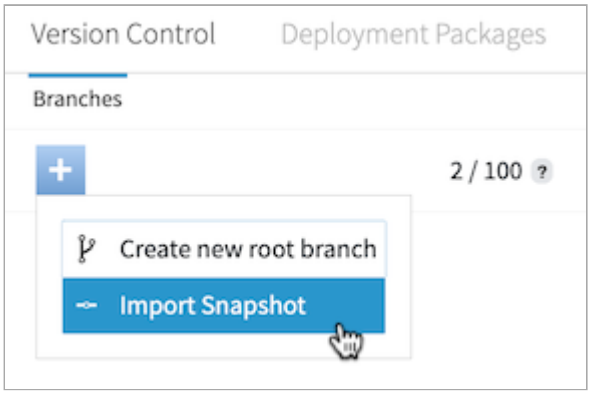
Import previously exported snapshot files as branches to share and restore Trust Framework definitions and policies across users and environments.

A snapshot file contains all the entities and policies from an existing branch. Learn more in Creating a snapshot.

Steps

1. In the Policy Editor, click **Branch Manager**.
2. On the **Version Control** tab, click +, and then select **Import Snapshot**.

Copyright © 2025 Ping Identity Corporation



3. Select the appropriate snapshot file.
4. Specify a unique name for the branch.
5. Click **Import**.
6. (Optional) Click **Commit New Changes** to commit the initial state of the policy branch.

Creating a subbranch from a commit

You can create a subbranch as a child of a commit in another branch. The subbranch shares the history and contents of the parent branch up to that commit.

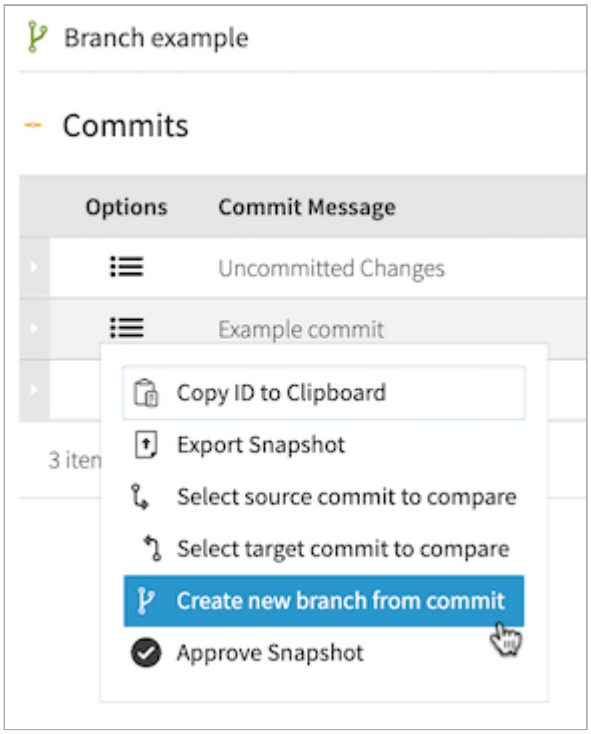
Learn more about branch commits in Committing changes.

Steps

1. In the Policy Editor, click **Branch Manager**.
2. On the **Version Control** tab, select the commit from which to branch.

   To branch from the latest uncommitted changes, make sure to commit before proceeding.
3. Click the hamburger menu and select **Create new branch from commit**.

Copyright © 2025 Ping Identity Corporation



4. Specify a unique name for the branch.
5. Click **Save Branch**.

Result

The system creates a new subbranch with the selected commit as the branch point.

Merging branches

Merge branches to apply all of the changes made in the source branch to the target branch.

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. In the Policy Editor, go to **Branch Manager** > **Version Control**.
2. Select the source branch.

   You can select a root branches or a subbranch.
3. Click **Set branch as Merge Source**.

Copyright © 2025 Ping Identity Corporation

4. Go to the target branch and click **Set branch as Merge Target**.

   When you select the source and target branch, the **Merge Branches** button becomes available.

5. Click **Merge Branches**.

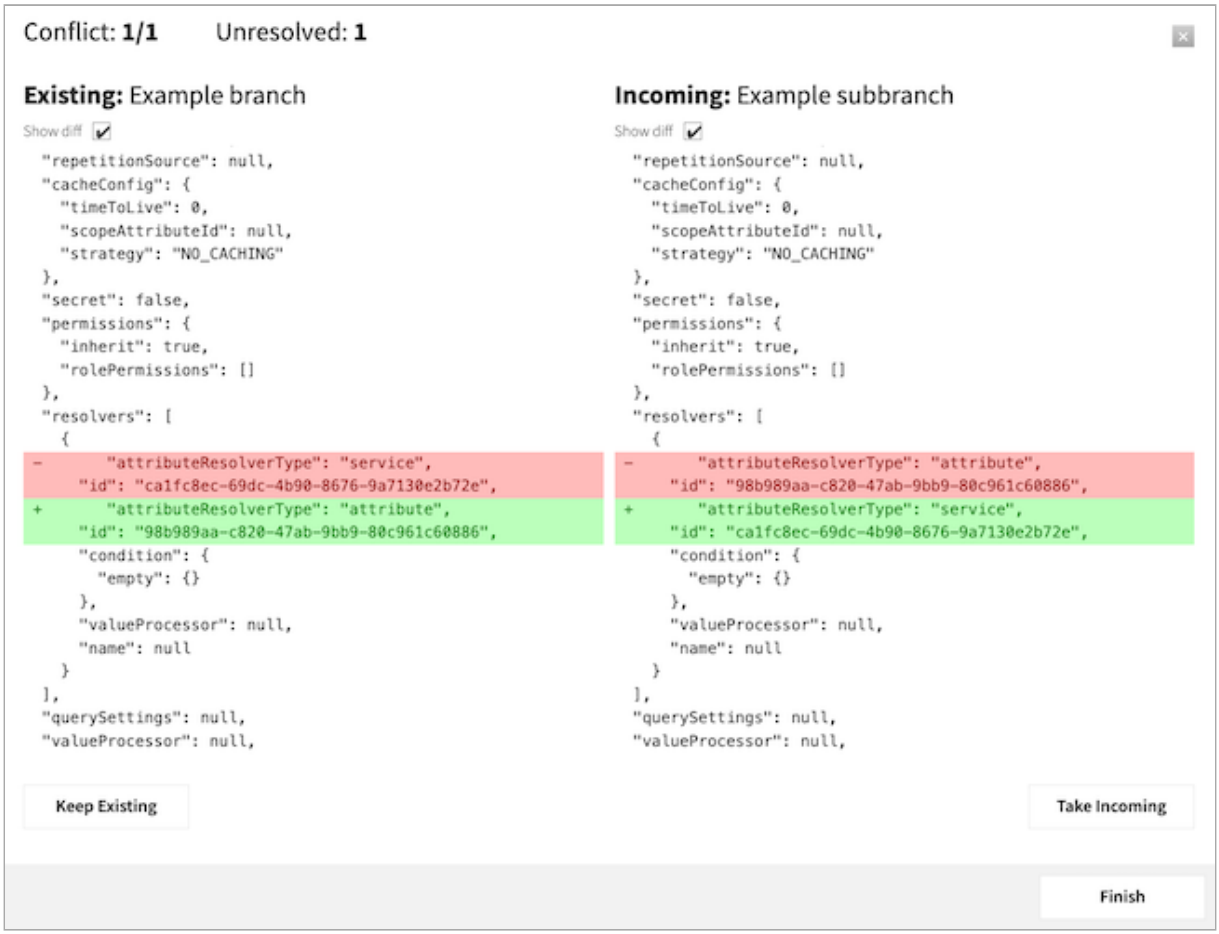
   The PingAuthorize Policy Editor checks for merge conflicts.

   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 next step.

6. Resolve conflicts.

   If an entity has changed in both the incoming (source) and existing (target) branches, the Policy Editor flags a conflict. For the merge to continue, you must resolve all conflicts listed in the **Merge Conflicts** table.

   1. If you need all or almost all of the sections from one branch, click either **Take All Incoming** or **Keep All Existing**.
   2. To examine conflicts individually, click **Resolve Individual Conflicts**.
      1. Select the **Show diff** checkbox on the resulting screen to highlight the conflicting changes.



      2. Decide which change to keep and click either **Keep Existing** or **Take Incoming**.
      3. Click **Finish**.

7. Click **Apply Merge**.

Copyright © 2025 Ping Identity Corporation

Deleting a branch

Delete a branch to remove the branch, its history, and any commits created on it from the system.

You cannot delete a branch if a deployment package has been created from that branch.

You might need a branch later, so create a snapshot before deleting it.

> **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. In the Policy Editor, click **Branch Manager**.
2. On the **Version Control** tab, select the branch to delete.
3. Click **Delete Branch**.

Branch changes and history

After creating branches to organize your organizations' distinct work contexts, you can commit changes to policies and the Trust Framework, revert unwanted changes, or create snapshots of important policy states.

Learn more about branch changes and history:

- Committing changes
- Reverting changes
- Creating a snapshot
- Merging snapshots

Committing changes

To save your policy and Trust Framework changes, commit your changes.

After you finish building, testing, and analyzing your policies, commit the changes. Committed changes cannot be reverted.

After you commit changes, you can create a deployment package from the commit. Learn more in Exporting a policy deployment package.

Steps

1. In the Policy Editor, click **Branch Manager**.
2. On the **Version Control** tab, select the branch in which to make the commit.

Copyright © 2025 Ping Identity Corporation

3. Click **Commit New Changes**.

> **Note**
>
> For this button to become available, the branch must have uncommitted changes.

Reverting changes

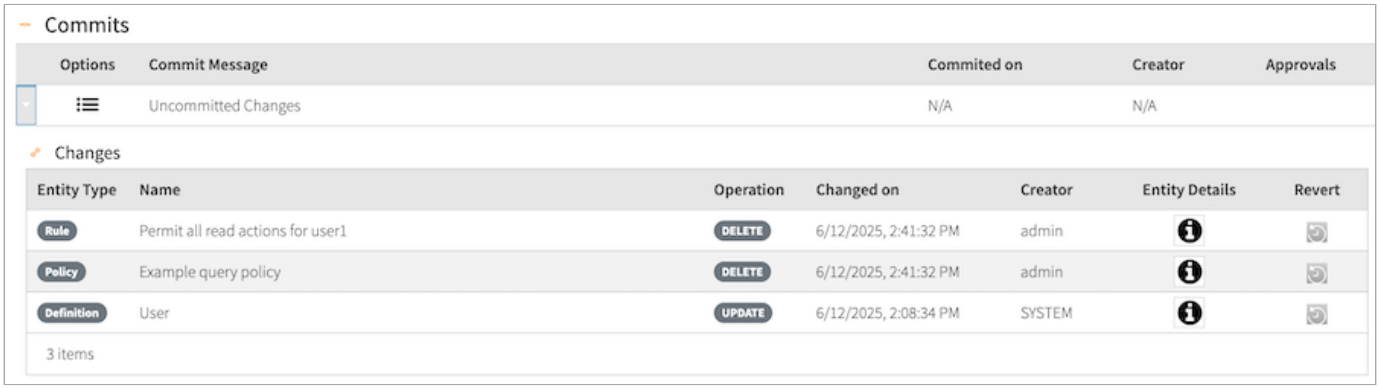
Revert unwanted changes made in a branch since the last commit.

> **Important**
>
> Reverting a change also reverts all subsequent changes made after it. Before reverting, review the entire list of affected changes to avoid unintended rollbacks.

Steps

1. In the Policy Editor, click **Branch Manager**.
2. On the **Version Control** tab, select the branch with the uncommitted changes to revert.
3. Expand the **Uncommitted Changes** section by clicking the arrow to the left of the hamburger menu.

   This displays all changes that have happened since the last commit.



4. Click **Revert** for the unwanted change and confirm the revert.

Creating a snapshot

Snapshots provide a convenient way to export policy and Trust Framework data into a separate Policy Editor instance. You can create a full or partial snapshot.

Learn more about creating branches by loading exported snapshots in Importing a snapshot.

Creating a full snapshot

A full snapshot contains all the Trust Framework and policy data from a branch commit or the branch's **Uncommitted Changes** head.

Copyright © 2025 Ping Identity Corporation

Steps

1. Go to **Branch Manager > Version Control**.
2. Select a branch.
3. In the **Options** column, click the hamburger menu for the commit you want to export, and select **Export Snapshot**.

4. Specify a name for the snapshot.
5. Click **Export**.

   Result:

   A snapshot file is downloaded to your computer.

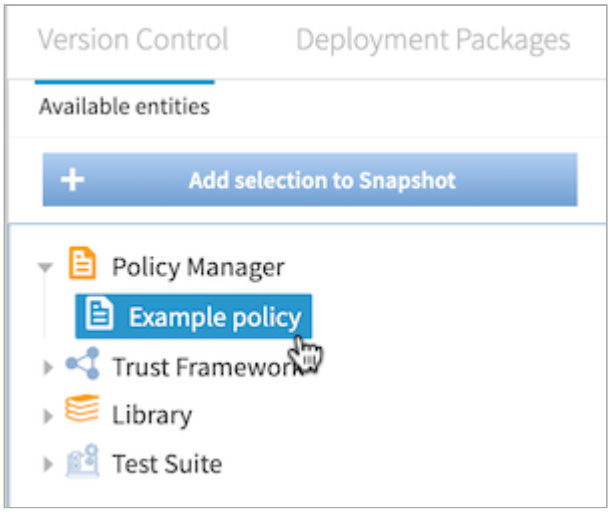
Creating a partial snapshot

Create and export a partial snapshot of selected entities from the Trust Framework, policy sets, and the Library.

Steps

1. Click **Branch Manager**.
2. Click **Export Partial Snapshot**.
3. In the **Available entities** list, select an entity to include in the snapshot.

Copyright © 2025 Ping Identity Corporation



4. Click **Add selection to Snapshot**.

   This step adds the entity to the **Selected entities** list. The exported snapshot automatically includes dependencies for each selected entity.

5. Select the **Include Children** checkbox for an entity to include that entity's children in the partial snapshot.

6. Click **Export**.

Merging snapshots

Merge a snapshot to add or update all the snapshot's entities in 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

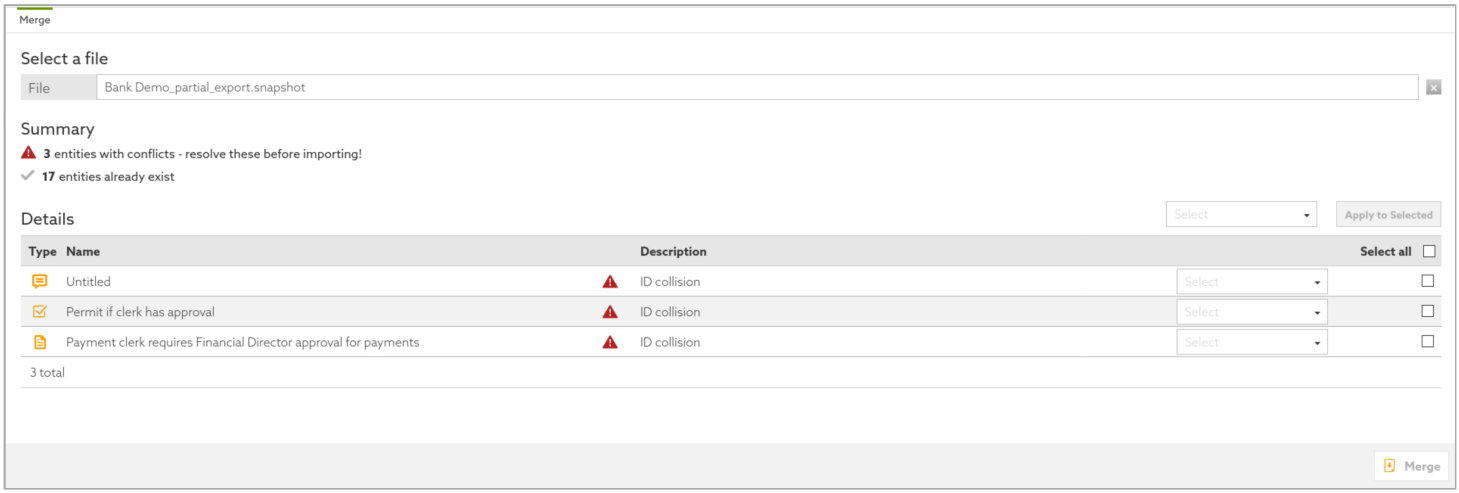
The system displays a **Summary** page that details the results of the merge.

Resolving merge conflicts

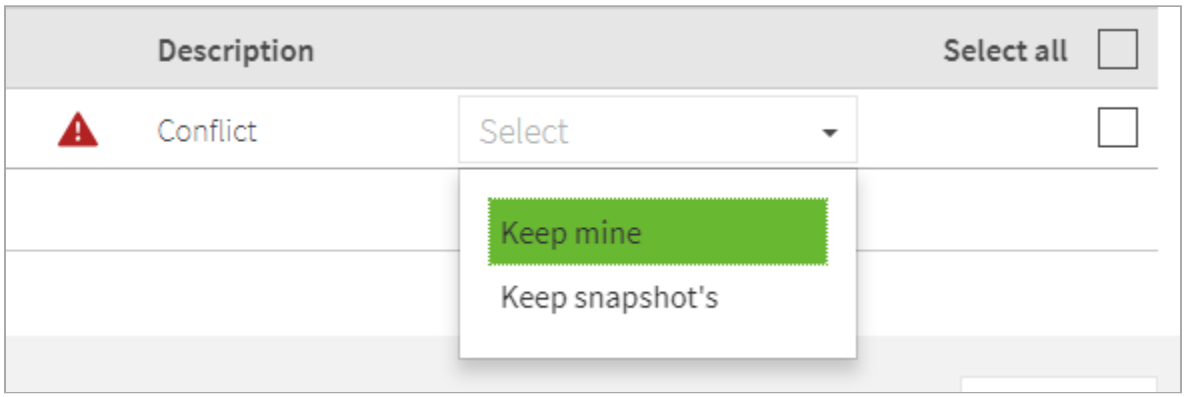
If the version of an entity in the current branch differs from the snapshot version, the merge function reports a conflict.

For example, if you update one of the merged entities in your current branch and then try to re-merge the snapshot, the system displays the following **Merge Conflict Resolution** page:

Copyright © 2025 Ping Identity Corporation



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**.

Deployment packages

Deployment packages play a critical role in the policy development lifecycle by providing a controlled, consistent way to promote finalized policies from development to production.

After policies have been developed, tested, and approved in a branch, they're bundled into a deployment package that captures a specific commit or snapshot of the policy state. This ensures that what gets deployed has been explicitly reviewed and versioned, reducing the risk of unintended changes.

Deployment packages enable predictable, auditable rollouts and support separation of duties by allowing policy authors to create and test policies while delegating deployment to administrators. Deployment packages also enhance governance, traceability, and operational confidence by standardizing how policies are promoted across environments.

While snapshots provide a convenient way to export and load policy and Trust Framework data into Policy Editor instances, deployment packages are intended for use in embedded policy decision point (PDP) mode. Learn more in Configuring embedded PDP mode.

Learn more about deployment package operations:

- Creating and exporting a deployment package

Copyright © 2025 Ping Identity Corporation

- Deleting a deployment package

Deleting a deployment package

Delete a deployment package to remove it from the **Packages** list.

Steps

1. In the Policy Editor, click **Branch Manager**.
2. On the **Deployment Packages** tab, select the deployment package.
3. 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 dependents.

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 you to define the following types of services:

- The resources to which you want to control access (what your policies will protect).

  For a resource, define only the top-level fields, such as **Name**, **Parent**, and **Description**. Leave the **Service Type** as `None` unless you plan to use the service as a policy information provider.

- The policy information providers that you use as sources of data for the attributes that comprise policy decisions.

  Setting up services as policy information providers makes use of various service connectors.

Copyright © 2025 Ping Identity Corporation

When you make a selection from the **Service Type** list, settings specific to the service are displayed, in addition to settings that apply to all service endpoints.

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.

Connecting a service

Connect PingAuthorize to external services to define data integrations.

Service connections in PingAuthorize enable you to augment authorization events with real-time data. For example, you might use signals obtained from a risk service connection in a policy that determines whether a device requires step-up authentication.

Steps

1. In the Policy Editor, go to **Trust Framework**.
2. Click the **Services** tab, click the + icon, and then select **Add new Service**.
3. Enter a unique **Name** for the service.

   > **Important**
   >
   > Periods (.) are not allowed in the name.

4. (Optional) Enter a **Description** of the service's purpose.

   The description is only visible on the **Services** tab, but it can help policy authors understand how to use services in policies.

5. (Optional) To nest the service under a parent in the tree, in the **Parent** list, select a parent service.

   Nesting helps group related services together. You can move the service to another location in the tree by selecting a different parent service. To remove nesting, click the **Delete** icon and leave **Parent** blank.

6. Select a **Service Type**.

   Choose from:

   - **None**: This is for a parent service. Nest other services under a parent to help organize services in the tree structure. There are no additional settings to complete for this type of service.
   - **Database**: Connects to relational databases and retrieves information through SQL queries. Learn more about database service settings in Database services.
   - **HTTP**: Connects to HTTP endpoints accessible over the public internet. Learn more about HTTP service settings in HTTP services.
   - **LDAP**: Connects to LDAP sources and retrieves information through database queries. Learn more about LDAP service settings in LDAP services.

7. (Optional) In the **Value Settings** section, define the data **Type** for the data returned by the service, and select the **Secrets** checkbox to encrypt that data in PingAuthorize logs.

Copyright © 2025 Ping Identity Corporation

Depending on which mode you've configured for PingAuthorize, service data secrets are recorded in one of these logs:

- Embedded PDP mode: The service data values are encrypted in `PingAuthorize/logs/policy-decision.log`.
- External PDP mode: The service data values are encrypted in `PingAuthorize-PAP/logs/decision-audit.log`.

To decrypt a service's data values, run the following command:

```
'echo -n "RSNH/SPsNJSFQyyLSxdKsw==" | base64 -d | openssl enc -aes-128-ecb -d -K
"54655374506153735068526153653939"
```

In the previous example, `RSNH/SPsNJSFQyyLSxdKsw==` represents the encrypted service value string, and `54655374506153735068526153653939` represents the encryption key in hexadecimal. By default, the encryption key is `TeStPaSsPhRaSe99` and cannot be changed.

8. (Optional) In the **Timeout and Retry** section, configure the following properties:

   - **Request Timeout**: The time that PingAuthorize waits, in milliseconds, for a service request to complete.

     The default timeout is `2000`. If the timeout elapses before there is a successful service response, the service request is canceled, resulting in a timeout error.

   - **Number of retries**: The number of times that PingAuthorize attempts a service request again when the initial request fails or times out. To try the service request only once, set this value to `0`.

     > **Note**
     >
     > If the service responds with a 4xx error, PingAuthorize won't attempt any retries.

   - **Retry Strategy**: The approach that PingAuthorize uses when retrying service requests.
     - **Fixed Interval** (default): PingAuthorize waits for the specified retry delay between service requests.
     - **Exponential Backoff**: PingAuthorize uses the specified retry delay as a base and waits for an exponentially increasing amount of time between service requests.
   - **Retry Delay**: The time that PingAuthorize waits, in milliseconds, between service requests.
     - For fixed interval retries, PingAuthorize waits this number of milliseconds between each service request.
     - For exponential backoff retries, PingAuthorize multiplies this number by 2^n, where n is the number of retries already attempted.

       For example, if the retry delay is 1000 milliseconds, PingAuthorize makes the initial service request, then waits 1000 milliseconds before making a second request, 2000 milliseconds before making a third request, and so on.
   - **Delay Jitter**: The amount of variability, as a percentage, that PingAuthorize applies to the retry delay between each service request.

     Delay jitter reduces the chance of synchronized retries and cascading service failures.

     If this value is set to `10%` and the retry delay is set to 1000 milliseconds with exponential backoff, the delays in the previous example would be 1000±100 milliseconds, 2000±100 milliseconds, 4000±100 milliseconds, and so on.

Copyright © 2025 Ping Identity Corporation

If this value is set to `10%` and the retry delay is set to a fixed interval of 1000 milliseconds, PingAuthorize waits 1000±100 milliseconds before making a second request, 1000±100 milliseconds before making a third request, and so on.

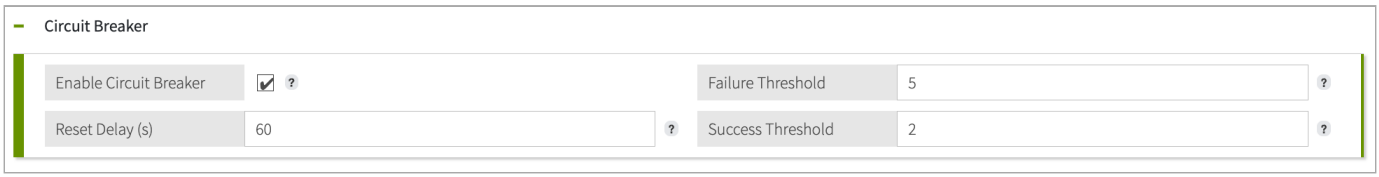
9. (Optional) In the **Rate Limits** section, enter a **Requests per Second** value to change the maximum number of requests that decision points can make to the service per second.

   The default value is `1000000`.

10. (Optional) Configure the **Circuit Breaker**.

    Circuit breakers enhance system resilience by preventing request overload. When enabled, circuit breakers monitor service calls, temporarily halt requests to services that appear unavailable, and then carefully test those services for recovery. Circuit breakers can be open (blocking service requests), half-open (allowing test service requests), or closed (allowing service requests).

    > **Important**
    >
    > Circuit breaker functionality is intended only for service requests made by the PingAuthorize Server in embedded PDP mode.

    1. Select the **Enable Circuit Breaker** checkbox.

    2. For **Failure Threshold**, enter the number of consecutive failed service calls required to open the circuit breaker.

       When the breaker is open, all requests to the service are immediately blocked.

    3. For **Reset delay**, enter the number of seconds the circuit breaker should wait in an open state before transitioning to a half-open state.

       When the breaker is half-open, the system allows a limited number of requests to the service, testing if the underlying issue has been resolved before fully closing the breaker.

    4. For **Success Threshold**, enter the number of consecutive successful service calls required to close the circuit breaker.

       When the breaker is closed, the service is considered healthy and all service requests are allowed to pass through normally. If requests to the service fail before the success threshold is met, the breaker re-opens.

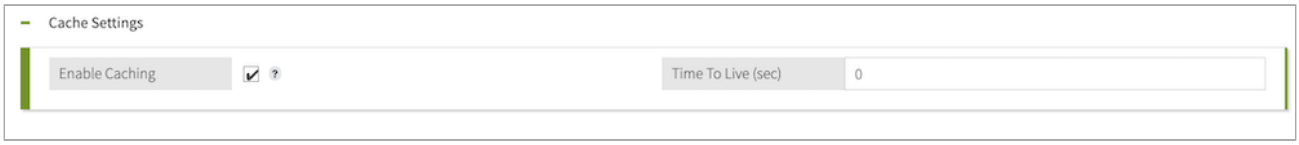
       The Debug Trace logger records messages indicating the current state of each circuit breaker. Learn more in Circuit breaker logging.

11. (Optional) Enable caching for the service.

Copyright © 2025 Ping Identity Corporation

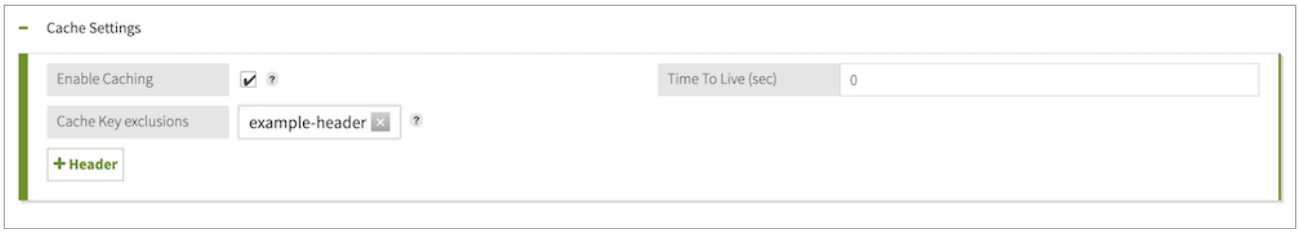
Caching improves system performance by storing data returned from a service and reusing it on subsequent service requests until the cache expires. Learn more in Service caching.

1. Select the **Enable Caching** checkbox.
2. For the **Time to Live**, enter the number of minutes that you want to store data retrieved from the service in the cache.



The maximum time to live is 1440 minutes or 1 day.

3. (Optional) Click **+ Header** and select headers to exclude from the cache key.



Because the service cache is invalidated by any changes to the cache key, removing headers preserves the validity of the cache even when values of those headers change.

Database services

Configure database settings to directly connect PingAuthorize to a relational database management system (RDBMS).

The policy decision point (PDP) can perform data requests to relational databases. By using complex SQL queries to efficiently retrieve and transform structured data, the database service type enables you to implement dynamic authorization logic based on identity information stored in an RDBMS.

PingAuthorize has been tested with the following database versions:

- Oracle
  - Driver: Oracle JDBC Thin Driver Version 21.7.0.0
  - Database: Oracle Database 12c Release 2 (12.2)
- PostgreSQL
  - Driver: PostgreSQL JDBC Driver Version 42.7.3
  - Database: PostgreSQL 15.0

To use these databases as policy information providers in embedded or external PDP mode, you must add the JDBC driver `.jar` library to the `lib` folder in your `PingAuthorize` distribution. The `.jar` libraries for both of the above database drivers are added to `PingAuthorize-PAP/lib` by default.

Copyright © 2025 Ping Identity Corporation

To use a JDBC-compliant database version not listed above, you must do the following:

- Add the JDBC `.jar` library to the `lib` folder in your `PingAuthorize` and `PingAuthorize-PAP` distributions.
- Add the JDBC driver to the database drivers allow list.
  - For a database service used for decision in embedded PDP mode, add an allowed database driver with the administrative console or `dsconfig`. Learn more in Configuring database service connections.
  - For a database service used for policy development and testing in the Policy Editor, add an allowed database driver in the `PingAuthorize-PAP/config/options.yml` file. Learn more in Configuring Policy Editor database service connections.

Core settings

- **Connection String**: The Java Database Connectivity (JDBC) connection string that defines the connection between PingAuthorize and your relational database. For the proper formatting of the connection string, consult the documentation for your specific database driver.

  The following are examples of valid connection strings:

  - `jdbc:postgresql://dbhost/dbname`
  - `jdbc:oracle:thin:@localhost:1521:SID`

  You can interpolate attributes anywhere in the connection string.

- **SQL**: The SQL query sent from the PDP to the database.

  By default, you can only interpolate attributes in the position of an attribute value. In this example, the PDP uses the **UserId** attribute defined in the Trust Framework as a query parameter and populates it at runtime:

  ```
  SELECT user, email FROM people WHERE userid = {{UserId}}
  ```

  You can interpolate attributes anywhere in the SQL query with the `unsafe` modifier. In this example, the PDP selects an entire row's values from an interpolated table, where the `userid` value equals 1234:

  ```
  SELECT * FROM {{userTable | unsafe}} WHERE userid = 1234
  ```

  Caution

  Depending on where the attribute value comes from, using the `unsafe` modifier could leave the SQL query vulnerable to injection attacks. To help prevent such attacks, make sure the interpolated attribute is a deterministic constant resolved from service values or decision request attributes.

Copyright © 2025 Ping Identity Corporation

Service responses

Database services convert the results from a database query into an array of JSON objects. For example, if the "first_name" and "last_name" columns are queried from the following table:

first_name	last_name	age
John	Smith	44
Sally	White	47

The service response will be the following array:

```
[
  {
    "first_name": "John",
    "last_name": "Smith"
  },
  {
    "first_name": "Sally",
    "last_name": "White:
  }
]
```

The service will attempt to convert SQL values of type `json` or `jsonb` to JSON objects. For example, if the following set of data is queried from a database:

personjson	jsonlength
`\{"first_name": "John", "last_name": "Smith", "age": 44}`	53
`\{"first_name": "Sally", "last_name": "White", "age": 45}`	55

The service response will be the following JSON array:

Copyright © 2025 Ping Identity Corporation

```
[
  {
    "personjson": {
      "first_name": "John",
      "last_name": "Smith",
      "age": 44
    },
    "jsonlength": 53
  },
  {
    "personjson": {
      "first_name": "Sally",
      "last_name": "White",
      "age": 45
    },
    "jsonlength": 55
  }
]
```

Note

Because JSON parsing can be an expensive operation, be careful when designing queries that include large JSON objects.

If an entry in the database query result is `null`, the service will return this value as a null JSON object. For example, if the following set of data is queried from a database:

actor_id	first_name	last_name	last_update
201	Penelope	<null>	2024-02-21 14:23:14.0

The service response will be the following JSON array:

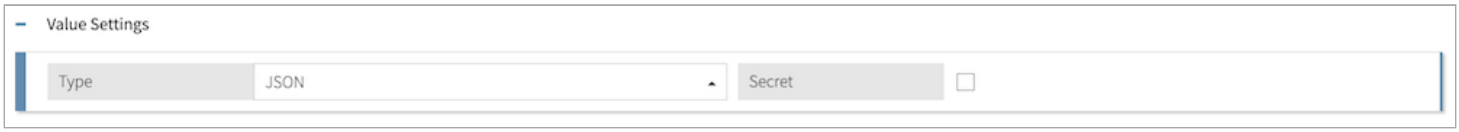
```
[
  {
    "actor_id": 201,
    "first_name": "Penelope",
    "last_name": null,
    "last_update": "2024-02-21 14:23:14.0"
  }
]
```

Note

If an entry in the database query result is of a binary or unrecognized type, the service will convert the value to a Base64-encoded string so that it can be safely returned in a JSON object.

Because PingAuthorize converts the result of an SQL query to a JSON document, you must set the service value type to **JSON** and use a JSONPath processor to extract data from the service response.

Copyright © 2025 Ping Identity Corporation



What to do next

You can configure the pool of database connections maintained by PingAuthorize to optimize data retrieval. Learn more in Configuring database service connections and Configuring Policy Editor database service connections.

HTTP services

Configure HTTP settings to connect PingAuthorize to an HTTP service.

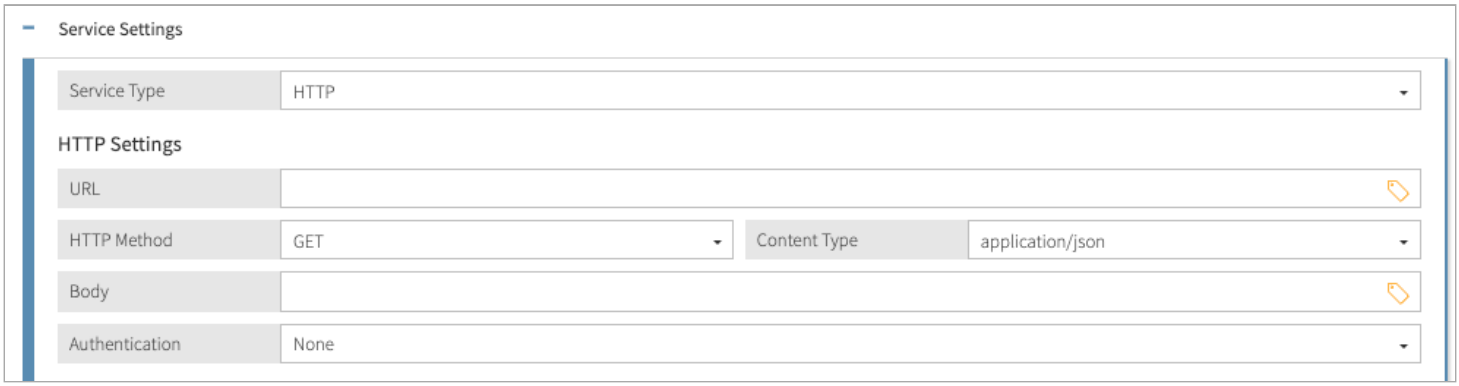
Before you connect to an HTTP service, you can add attributes that store values for service settings, such as the endpoint URL for an HTTP request. Storing settings as attributes is useful if the values are dynamic or if you want to use different values during testing.

> **Important**
>
> If the service requires OAuth 2.0 Client Credentials authentication, you must add an attribute that stores the external service's client secret before you configure service settings. You can also add attributes that store other client credentials, such as the client ID and token endpoint.

After configuring the service, test the service connection. To make the service response available in policies, add attributes that resolve against the service and process the response to extract required values.

HTTP settings

HTTP service requests can send and receive text, JSON, and XML content.



Copyright © 2025 Ping Identity Corporation

HTTP settings

Setting	Description
Target URL	The URL for the REST endpoint that the policy decision point (PDP) accesses in the HTTP request. To include an attribute anywhere in the URL, wrap the full name of the attribute in double curly brackets. **Important** Because no escaping of attribute values takes place, make certain that this action is completed in the attribute definition to prevent non-valid characters in the URL.
HTTP Method	The action performed in the HTTP request. Options are **GET**, **POST**, **PUT**, **DELETE**, and **HEAD**.
Content Type	The media type of the content in the HTTP request. Options are **application/json**, **application/xml**, **text/html**, and text.
Body	The body to send with the HTTP request. You can include attributes anywhere in the body, with no escaping, by wrapping the full name of the attribute in double curly brackets.

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.

To prevent invalidation of HTTP service response caches caused by changes to the cache key, you can exclude certain headers that you expect to change from service cache entries. Learn more in Connecting a service and Service caching.

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. Learn more in Specifying custom configuration with an options file.

When using embedded PDP mode, you can declare local file-based trust stores and key stores by assigning trust manager providers and key manager providers to the Policy Decision Service. Learn more in Deploy policies in a production environment.

Server (TLS)

Applies 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**

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> This option is the default for Server (TLS).

Uses the default trust store provided by the runtime environment.

Use this if you're 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:

  - **Truststore 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:

- **Keystore 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 certificate entry is password protected, it might need to provide the password.

  Attributes can be interpolated anywhere in the value.

HTTP authentication methods

HTTP service requests can use no authentication, basic authentication with a username and password, a bearer token, or the OAuth 2.0 Client Credentials flow.

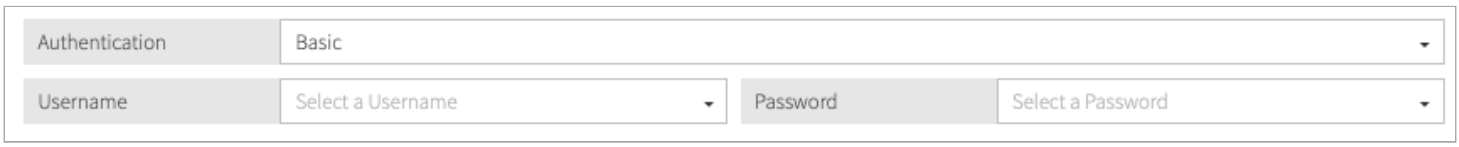
Copyright © 2025 Ping Identity Corporation

None

With this method, no authorization header is sent with the HTTP request. This is the default option.

Basic

For basic authentication, provide a username and password.

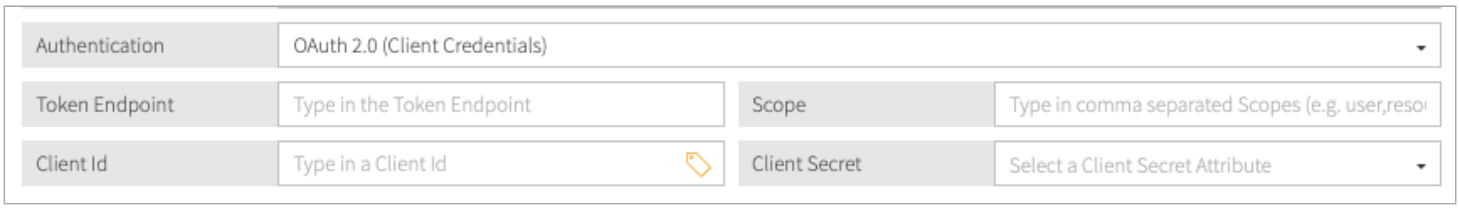


Bearer Token

For bearer authentication with a static token, select an attribute that stores the authorization token to send with the HTTP request.

OAuth 2.0 (Client Credentials)

To reduce configuration complexity and time to production when connecting to HTTP services that require OAuth authentication, configure client credentials to exchange for an access token.



PingAuthorize manages this authentication process as follows:

1. PingAuthorize requests an access token for the external service by sending client credentials, including any scopes, in a POST request to the authorization server's token endpoint. Credentials are sent in the request body.

Copyright © 2025 Ping Identity Corporation

PingAuthorize always sends credentials in the HTTP request body (`client_secret_post`) and doesn't support using the `Authorization` header (`client_secret_basic`).

2. The authorization server validates the client credentials and provides an access token, which PingAuthorize caches. When the external service no longer accepts this token, PingAuthorize sends another POST request to the token endpoint to obtain a new one.
3. In an HTTP service request, PingAuthorize provides the access token in the `Authorization` header, authenticates with the HTTP service, and retrieves the requested information from the service.

Before you configure the following settings, add an attribute that stores the client secret. You can also add attributes that store other client credentials.

- **Token Endpoint**: The token endpoint URL for the authorization server that grants an access token. For example, `https://{domain}/oauth*/token`.
- **Client Id**: A unique string representing the client identifier. This value can be either a constant or an interpolated attribute to resolve the value at runtime.
- **Client Secret**: A Trust Framework attribute that resolves to the client secret.
- **Scope** (optional): A comma-separated list of scopes representing the scope of the access request.
- **Advanced OAuth Settings** (optional): A list of custom key-value pairs representing additional parameters sent in the body of the token endpoint request. This level of customization is useful when integrating with authorization servers that enforce specific configuration constraints.

  The keys are fixed strings, but their values can be constants or attribute values. To switch between constant and attribute, toggle **C / A**.

  > **Note**
  >
  > Advanced OAuth settings have the following constraints:
  > - You can define up to 10 key-value pairs.
  > - Keys cannot contain spaces in any position.
  > - You cannot use any of the following values as a key (case-sensitive):
  >   - `client_id`
  >   - `client_secret`
  >   - `scope`
  >   - `grant_type`

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.

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:

Copyright © 2025 Ping Identity Corporation

```
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.

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. For example, the following XPath processor extracts the set of unique members:

```
//searchResponse/searchResultEntry/attr[@name='uniqueMember']/text()
```

Applying this processor to the above XML document produces the following result:

Copyright © 2025 Ping Identity Corporation

```
uid=euclid,dc=example,dc=com
uid=riemann,dc=example,dc=com
uid=euler,dc=example,dc=com
uid=gauss,dc=example,dc=com
uid=test,dc=example,dc=com
```

Enabling Camel service connections

If your PingAuthorize implementation relies on Apache Camel, you can enable Camel for both embedded and external policy decision point (PDP) modes.

To ensure that Camel is used with appropriate permissions and security controls, Camel services are disabled by default in the Policy Editor.

Starting with PingAuthorize 10.3, you can use Camel version 4.0.

- Learn how to upgrade PingAuthorize versions 10.0-10.2 using Camel 3.x in the Apache Camel 3.x to 4.0 migration guide.

> **Warning**
>
> If you upgrade an existing installation of PingAuthorize to version 9.3.0.0 or later, and the earlier version was configured in embedded PDP mode with a deployment package containing Camel services, the upgraded PingAuthorize server will fail to start. The following steps also resolve this issue.

Before you begin

You must install the Policy Editor and the PingAuthorize server before enabling Camel services.

Steps

1. In the PingAuthorize Server distribution, copy the `symphonic-camel-pdp-<version>.jar` file from `resource/camel` to the `lib` folder.
2. In the Policy Editor distribution, copy the `symphonic-camel-pap-<version>.jar` file from `resource/camel` to the `admin-point-application/lib` folder.
3. Enable the Camel service.

   Choose from:

   - Run `setup` in non-interactive mode using the `--enableCamelService` 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 \
  --enableCamelService
```

Copyright © 2025 Ping Identity Corporation

> **Warning**
>
> This choice will override the existing configuration.

- In your Policy Editor distribution, add the `- "camel-service"` flag to `enabledFeatures` in the `config/configuration.yml` file.

```
enabledFeatures:
  - "test-suite"
  - "entity-dependents"
  - "camel-service"
```

4. Save your changes and restart the Policy Editor.

Result

Camel service connections are enabled in both external and embedded PDP modes.

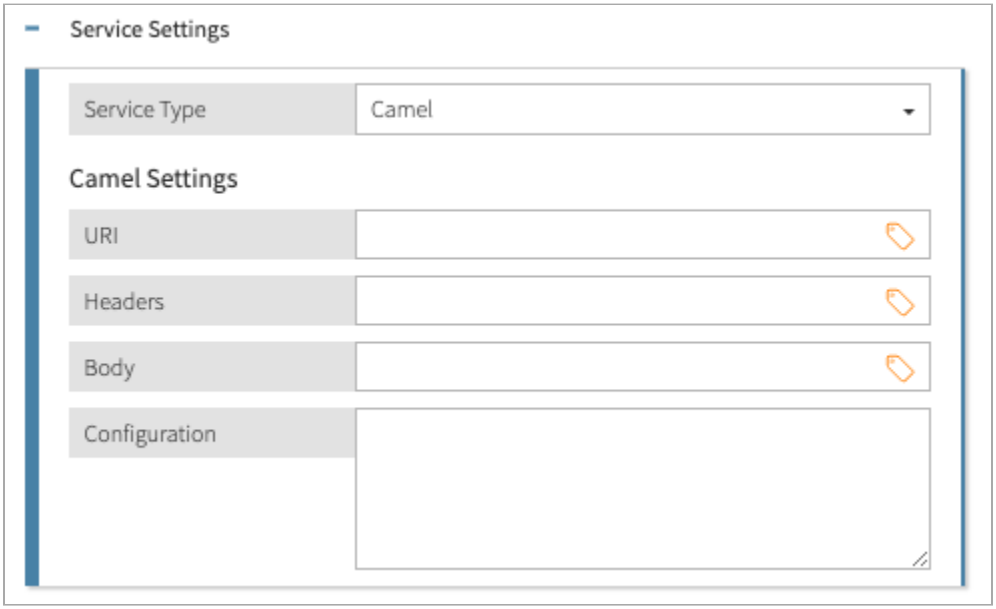
Camel services

You can retrieve information from any endpoint that the Apache Camel enterprise integration platform supports. The list of Camel components in the Camel documentation provides the full list of supported systems.

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 being used.

You can make requests dynamic by interpolating attribute values into different parameters. Learn more in Attribute interpolation.

Camel settings



Copyright © 2025 Ping Identity Corporation

Setting	Description
URI	The identifier for the Camel endpoint. PingAuthorize can interpolate attribute values into this field.
Headers	Additional information to send with the Camel service request. PingAuthorize can interpolate attribute values into this field.
Body	The body to send with the Camel service request. PingAuthorize can interpolate attribute values into this 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. Learn more about writing configurations in Class GroovyBeanDefinitionReader. PingAuthorize cannot interpolate attributes into this field.

> **Note**
>
> The Camel JDBC component uses the **Headers** and **Body** settings and requires a JDBC data source to be set up in the **Configuration** setting.

Value processors

You can define value processors to transform data returned by the Camel service. Learn more in Processors.

Service caching

Caching improves decision evaluation performance and reduces latency by storing data retrieved from services for faster retrieval on subsequent service requests.

A service is a good candidate for caching if service response values don't change very often.

How service caching works

When the decision engine requests information from a service, it caches the service response if the data has not already been cached. On subsequent service requests, instead of invoking the service again, the decision engine uses the cached data until the cache expires.

The cached value is used as long as none of the values defined in **Service Settings** have changed. The decision service invokes the service and caches the service response under a new key when a value in the service request changes, such as when there's a new parameter in the request body, or an attribute value is resolving differently.

To prevent changes in request header values from invalidating an HTTP service response cache, you can exclude such headers from the cache key. Learn more in Connecting a service.

PingAuthorize maintains the service response in the cache for the period of time that you define in the **Time to live** setting for a service.

Copyright © 2025 Ping Identity Corporation

> **Important**
>
> Cache entries are stored in memory and do not persist during a restart or outage unless you have defined your own data store.

Service caching configuration

You can configure cache settings for individual services when you add or edit a service in the **Trust Framework**. Learn more in Connecting a service.

If you've already set up external attribute caching in PingAuthorize and enable caching for a service:

- Attributes are cached in an external Redis instance.
- Services are cached with the same configuration object used for external attribute caches and are also cached locally in memory.

You cannot configure external caching just for services. If you are not seeing cache hits in the external cache instance, this might be because the cache entry is also stored locally. Storing cache entries both locally and externally is unique to services. Attributes configured with external caching are not stored locally.

Service caching behavior

Unlike attribute caching, which uses a one-level approach where cache entries are stored and retrieved from the single configured cache type, service caching uses a two-level approach to improve performance and minimize redundancy. For example, if you configure an external Redis cache, a local cache acts as the first-level cache, and the Redis instance acts as the second-level cache.

When making a call to a service, the decision engine attempts to load the first-level cache entry, and if the entry cannot be found, an attempt is made to load the second-level cache. If neither attempt succeeds, the decision engine makes a call to the service itself. If the second-level cache entry is unreachable, the first-level cache entry is stored for a maximum of 2 minutes.

You cannot clear the cache manually.

> **Important**
>
> Service cache settings in PingAuthorize override any cache control headers in the HTTP response. This can result in:
>
> - Caching when the resource server cache control headers indicate that the response shouldn't be cached
> - A delay in retrieving data from the resource service because the service is not called until the time to live has expired

Verifying use of the service cache

By enabling the file-based Debug Trace logger and inspecting service call messages in `PingAuthorize/logs/debug-trace.log`, you can determine whether the policy decision point (PDP) retrieved service data from the cache.

Learn more in Service cache logging.

Attributes

Attributes provide the context that enables fine-grained policies.

Copyright © 2025 Ping Identity Corporation

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, Adding a named condition, 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.

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.

Copyright © 2025 Ping Identity Corporation

Resolvers

Resolvers define where attributes pull information from.

An attribute can have one or more resolvers. Resolvers can be conditional. In addition, you can add a processor to a resolver to modify the attribute value.

PingAuthorize resolves an attribute when the attribute is used in a decision evaluation. Multiple resolvers are resolved in order of appearance in the attribute definition, until one of them produces a value. You can drag collapsed resolvers to change their order. If a resolver fails to resolve successfully, processing moves on to the next resolver.

Resolver types

You can use the following types of resolvers:

Resolver type	Description
Request	Looks inside the decision request 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	Takes a constant value defined on the resolver itself. The type and value of the constant are required. **Note** Constants undergo any value processing defined for the attribute.
Service	Resolves the attribute by using a **Trust Framework** service endpoint to invoke the service at runtime. The service might rely on other attributes being suplpied to invoke the service.
Attribute	Resolves an attribute form other attributes. This enables you to extract a child attribute or subset from an attribute that contains multiple pieces of information. For example, consider a `Customer.Name` attribute that returns the following JSON representation: `{ "firstname": "Andrew", "middlename": "James", "surname": "Martin" }` You can use an **Attribute** resolver to extract the surname. Create a **Customer.Name.Surname** attribute that resolves against the **Customer.Name** attribute. Then, add a JSON Path processor to extract the **Surname** property of the JSON using the expression `$.fullname[0].surname`. When an attribute has a parent, you can click the **+ Add Parent Resolver** button to resolve the child attribute against its parent.

Copyright © 2025 Ping Identity Corporation

Resolver type	Description
System	Use standard system attributes without the need for any additional configuration: **`CurrentDateTime`** This returns the current system `datetime` according to the type defined for the attribute. **`Null`** This returns no value. For example, you can use `Null` to compare the value of an attribute to an empty string. To check against a null value, create an attribute with a **System** resolver and a value of **Null**.
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. Learn more in Specifying custom configuration with an options file. When using embedded PDP mode, you do this by creating Policy Configuration Keys in the Policy Decision Service. Learn more in Configuring embedded PDP mode.

Conditional resolvers

Resolvers can have conditional logic that invokes the resolver only under defined conditions. As with other in PingAuthorize, you can add comparisons or named conditions, combine multiple conditions using **All**, **Any**, or **None**, and add subgroups.

In the following example of a conditional resolver, the game player's email address is resolved only when the **User is over 18** attribute has a value of `true`.

Value processing for a resolver

Resolvers can have value processors that extract details and modify data when the attribute value is being resolved. As with other processors, value processors for resolvers are processed in order, with each processor receiving the output of the previous processor.

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`

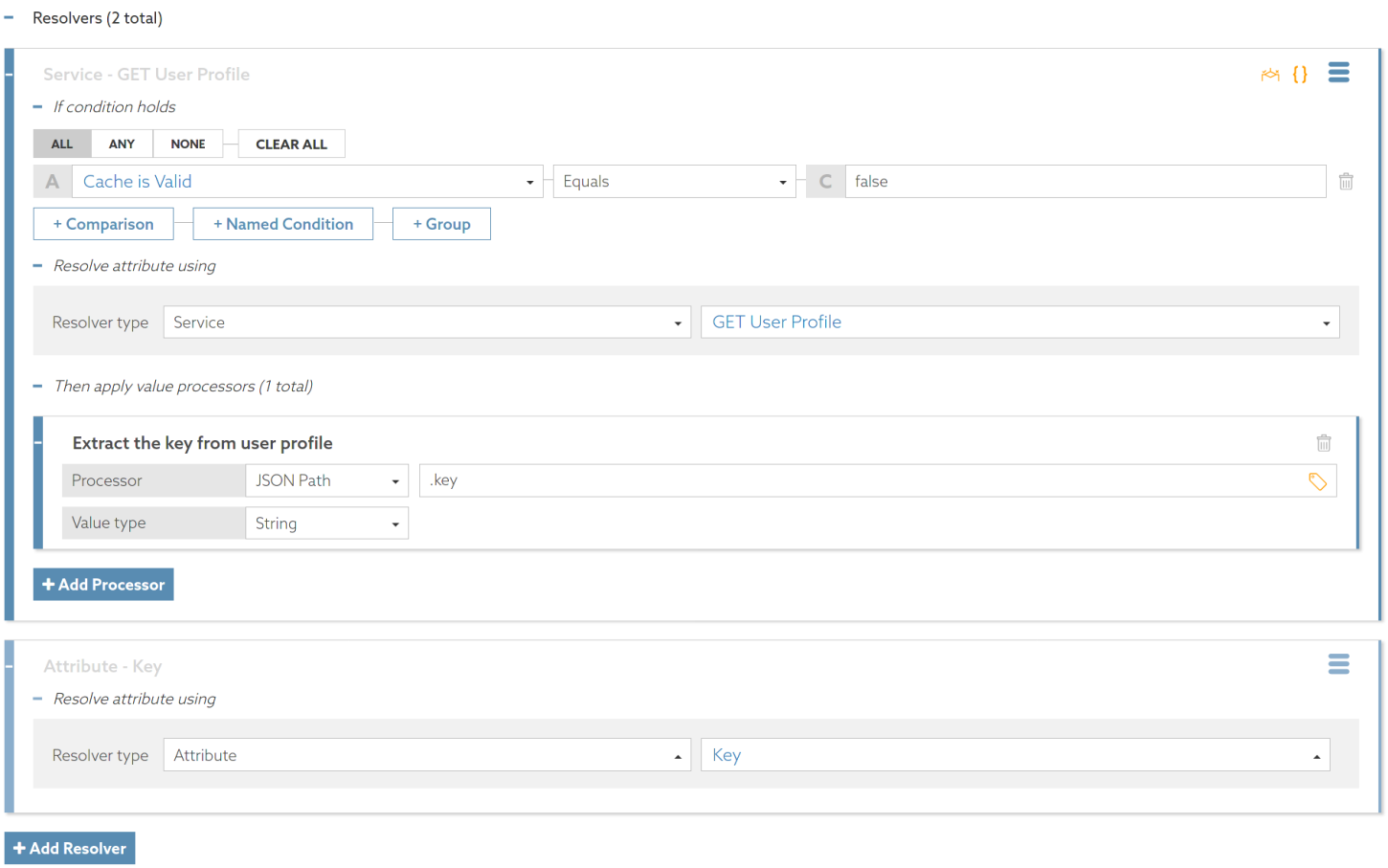
  With this resolver, if the `Cache is Valid` attribute is false, the resolver calls the `GET User Profile` service and uses a JSON Path processor to extract the key from the profile JSON.

- An attribute named `Key`

  In the second resolver, the attribute value comes from the `Key` attribute, and the value requires no processing.

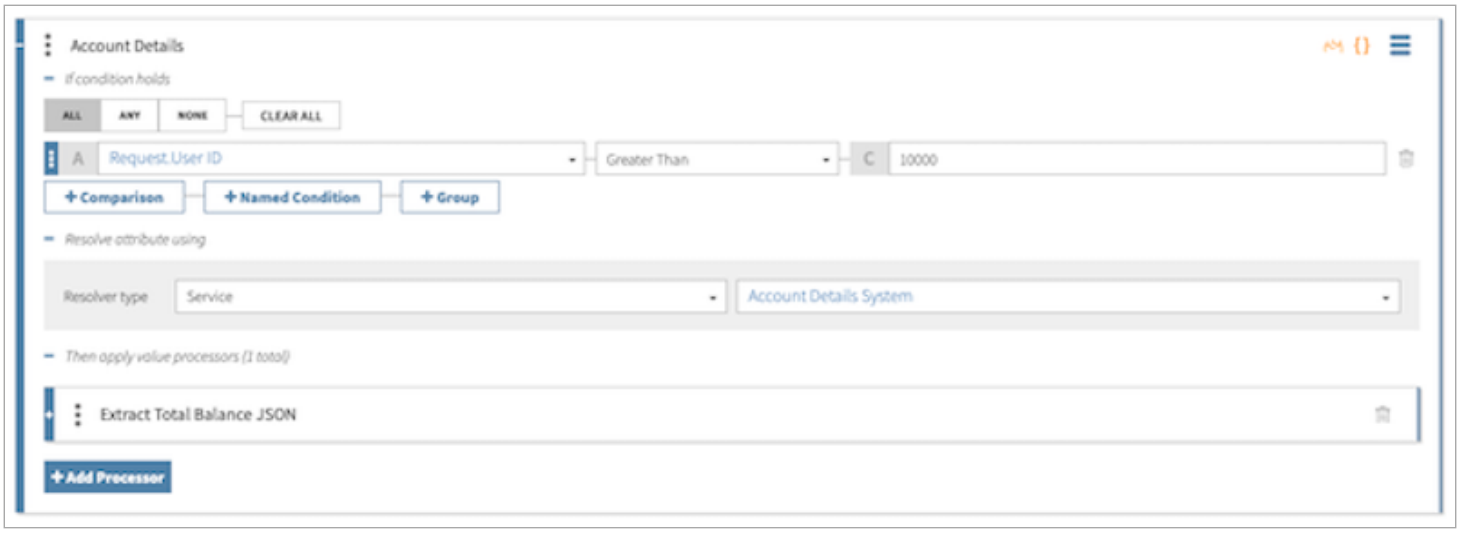
The following image shows the resolvers. The resolvers apply in the order shown.

Copyright © 2025 Ping Identity Corporation

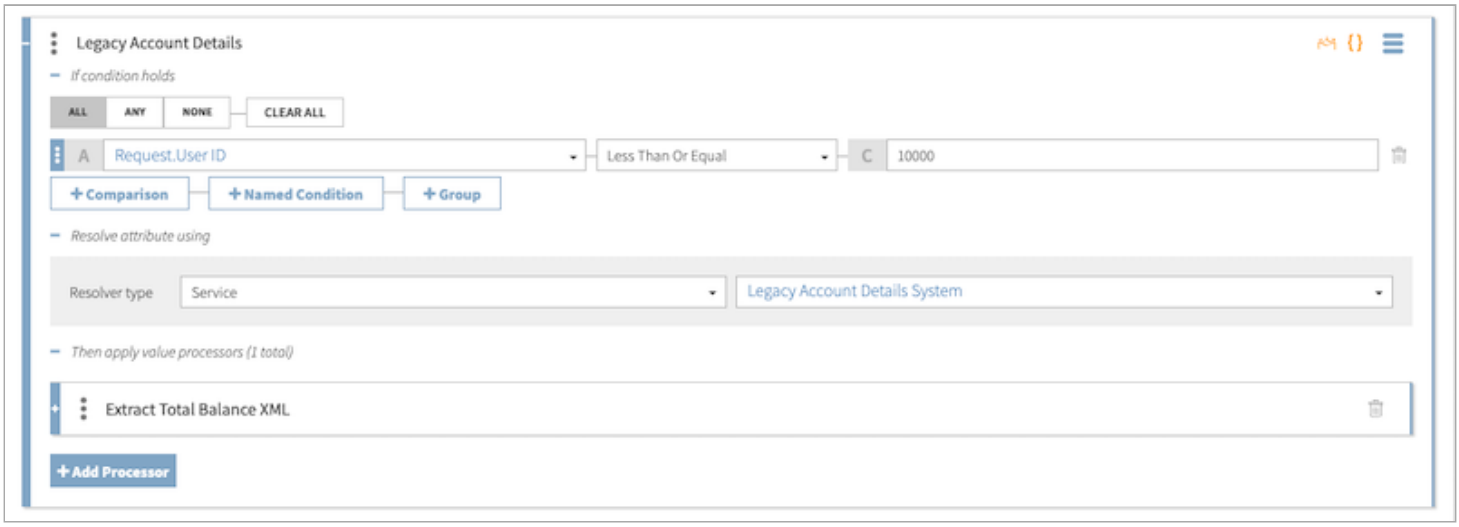


Adding value processors directly to resolvers, rather than to the attribute itself, can be useful when pulling data from distinct sources that require different formatting.

For example, consider a bank that wants to pull in user information either from a modern system or a legacy system to authorize transactions. These systems are represented by the **Account Details System** and **Legacy Account Details System** HTTP services, respectively. With the following resolver configuration, the decision service calls to the **Account Details System** service if the user ID is above `10000` (indicating a new user), or to the **Legacy Account Details System** service if the user ID is less than or equal to `10000` (indicating a legacy user):



Copyright © 2025 Ping Identity Corporation



Data collected from the **Account Details System** service is formatted as JSON and requires a JSON Path processor defined on the **Account Details** resolver to extract a new user's total balance:

Data collected from the **Legacy Account Details System** service is formatted as XML and requires an X Path processor defined on the **Legacy Account Details** resolver to extract a legacy user's total balance:

After the decision service pulls in user data from either service, common value processing might be required regardless of where the data comes from. For example, suppose the bank has a 500 USD minimum for an active account and anything above that is available for transfers. The following SpEL processor subtracts 500 from the result of resolver processing to obtain the user's available balance for transfers:

Copyright © 2025 Ping Identity Corporation

Attribute caching

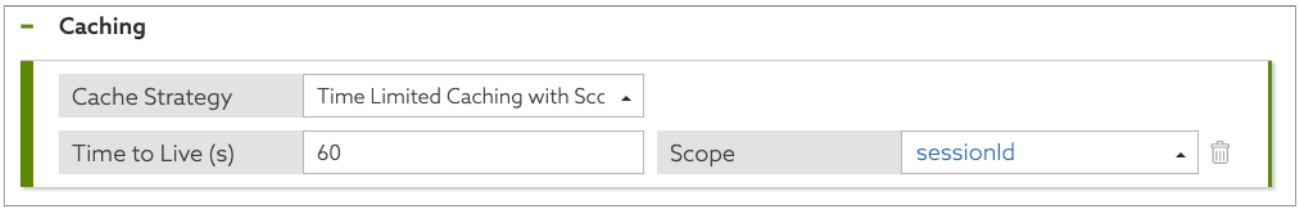
The policy decision point (PDP) and the PingAuthorize Policy Editor support caching for attributes. The ability to cache resolved attributes can deliver significant performance gains for the PDP.

Carefully consider this concept to ensure optimum configuration.

This section focuses on the individual cache options that you can set at the attribute level. See Configuring Trust Framework attribute caching for development or Configuring Trust Framework attribute caching for production for more information.

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.



Attribute caching uses a one-level approach where cache entries are stored and retrieved from the single configured cache type. 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 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 helps with 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 the 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. Learn more in Processors.

Copyright © 2025 Ping Identity Corporation

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 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. In both cases, the system uses the canonical conversion to a String format.
- Adding attribute data to statements, 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)

Copyright © 2025 Ping Identity Corporation

Data type	Description
Time	A time of day, such as 4:15 pm and 30 seconds. The textual representation is ISO-8601. The maximum resolution is microseconds. 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.
Date Time	A date and time of day, such as 4:15 pm and 30 seconds on April 23, 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 the appropriate information in each case: • **Date Time** • **Date** • **Time**
Duration	A time duration expressible in seconds or fractions of a second. 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.

Copyright © 2025 Ping Identity Corporation

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 always resolving to the default value.

Secrets

To encrypt an attribute's values in PingAuthorize logs, you can enable secrets for that attribute.

Depending on which mode you have configured PingAuthorize in, these secrets are recorded in one of two logs:

- Embedded PDP mode: The attributes are encrypted in `PingAuthorize/policy-decision.log`.
- External PDP mode: The attributes are encrypted in the `decision-audit.log` file distributed with the Policy Editor, but not `PingAuthorize/policy-decision.log`.

To decrypt an attribute's value, run the following command. In this example, `RSNH/SPsNJSFQyyLSxdKsw==` represents the encrypted attribute string, and `546553745061537350685261536539 39` represents the encryption key in hexadecimal. By default, the encryption key is `TeStPaSsPhRaSe99`, and cannot be changed.

Copyright © 2025 Ping Identity Corporation

```
'echo -n "RSNH/SPsNJSFQyyLSxdKsw==" | base64 -d | openssl enc -aes-128-ecb -d -K
"5465537450615373506852615365393 9"
```

Attribute interpolation

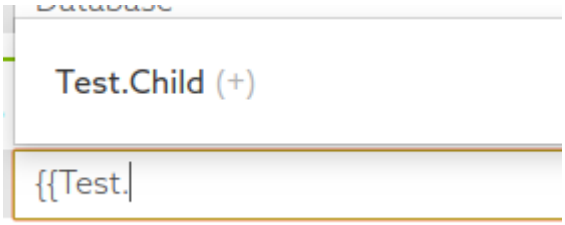
With attribute interpolation, you reference an attribute in a field. The system resolves the value of the referenced attribute, replacing the reference with the value itself.

About this task

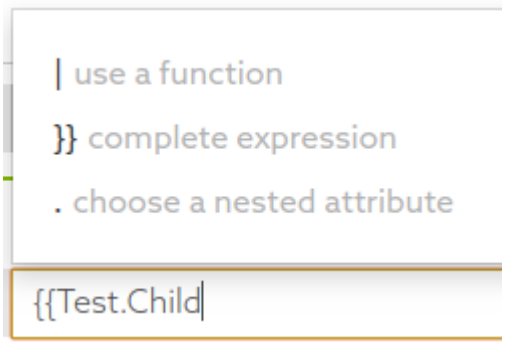
You can use attribute interpolation in any field that has the label icon, shown below.

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.

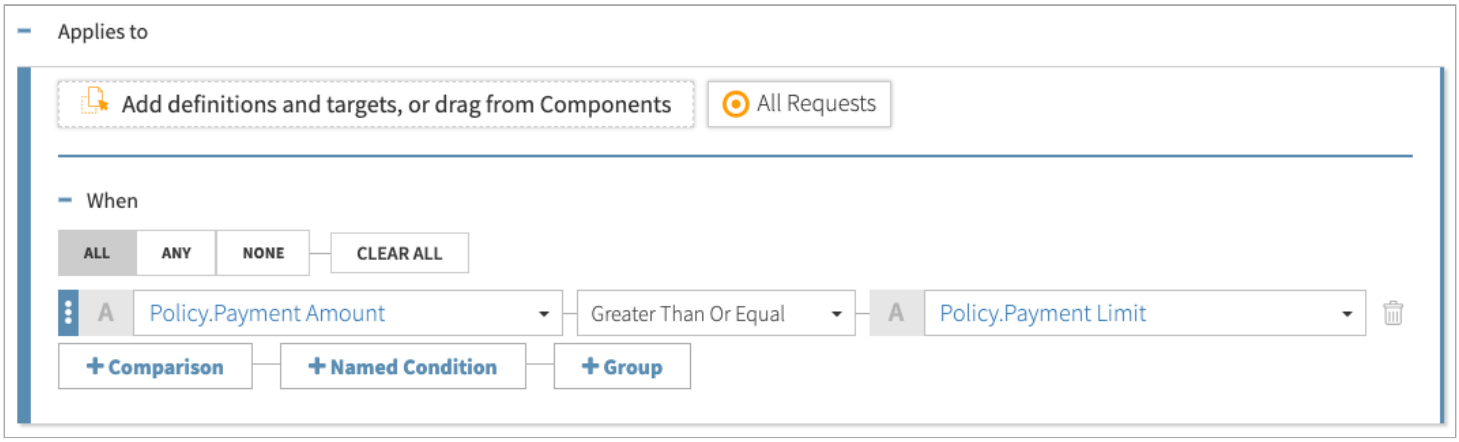


Conditions

Use conditions in PingAuthorize attributes, rules, and policies to define authorization logic by comparing one thing to another. Conditions evaluate to either `true` or `false`.

You can compare requests, attributes, constant values, and regular expressions in conditions. Conditions can also serve as targets that define when a policy or rule applies to a decision request. For example, you can target a rule so that it applies when a payment amount is greater than or equal to a payment limit.

Copyright © 2025 Ping Identity Corporation



When you define a condition, on the left side, select an attribute or request type that represents unknown or variable information to be validated. On the right side, enter known or predefined criteria in the form of an attribute, request, or constant value. This keeps logical statements consistent regardless of what's being compared. You can drag collapsed conditions to change their order, as needed.

You can add conditions directly to resolvers and rules or define them on the **Components** tab of the **Library** as reusable named conditions.

Condition comparators

You can use the following comparators in condition comparisons.

> **Note**
>
> For simplicity, the table groups logical comparator pairs together, but you can only use one comparator at a time in a condition.

Attribute comparators

Comparator	Supported data types	Description
Contains **Does Not Contain**	Collection, String	Checks whether a string or collection contains, or doesn't contain, another string. Use this comparator when you know part of a value that you want to check. **Note** Matches for strings can differ from matches for collections. For example, the string `1234` contains the constant `23`, but the collection `[1234]` does not contain this constant. One possible matching collection for the constant `23` is `[21, 22, 23]`.

Copyright © 2025 Ping Identity Corporation

Comparator	Supported data types	Description
Contains Claim Pair **Does Not Contain Claim Pair**	String	Add defense in depth by validating token claims as part of the authorization layer. Checks whether a JSON Web Token (JWT) contains a given claim pair. Encrypted tokens are not supported. To create a comparison: 1. Select an attribute of type String that resolves to a JWT. 2. Select the **Contains Claim Pair** or **Does Not Contain Claim Pair** comparator. 3. Enter a claim pair using the syntax `<claim-name>=<claim-value>.` **Note** The claim value has to exactly match one of the possible values of that claim contained in the token. For example, `aud=api1` will match against a token containing the claim `"aud": "api1"` or `"aud"=["api1", "api2"]`. To check a multivalued claim, create a comparator for each value you want to check. Multivalued `scope` claims in the token must be expressed as a list of space-delimited, no-space strings.
Ends With **Does Not End With**	String	Checks whether a string ends with, or doesn't end with, another string.
Equals **Does Not Equal**	Boolean, Collection, Date, Date Time, Duration, JSON, Number, Period, String, Time, XML, Zoned Date Time	Checks whether two values are equal or not equal.
Greater Than **Less Than**	Boolean, Date, Date Time, Duration, Number, String, Time, Zoned Date Time	Checks whether a value is greater than, or less than, another value.
Greater Than Or Equal **Less Than Or Equal**	Boolean, Date, Date Time, Duration, Number, String, Time, Zoned Date Time	Checks whether a value is greater than or equal to, or less than or equal to, another value.

Copyright © 2025 Ping Identity Corporation

Comparator	Supported data types	Description
Has Valid Signature For JWKS **Has Invalid Signature For JWKS**	String, JSON	Add defense in depth by validating tokens as part of the authorization layer. Checks the following for a JWT: • Whether the signature can be verified using one of the public keys in the JSON web key set (JWKS) • If a token expiry was set, whether the token is expired A valid token must have a verified signature and not be expired. **Note** PingAuthorize supports both RSA-encoded and EDSCA-encoded signatures. Encrypted tokens are not supported. To create a comparison: 1. Select an attribute of type String that resolves to a JWT. 2. Select the **Has Valid Signature For JWKS** or **Has Invalid Signature For JWKS** comparator. 3. Select a JSON attribute that resolves to a JWKS.
In CIDR Block **Not In CIDR Block**	String	Simplify adding network checks to support your zero trust policies. Verifies whether a user's IP address is in, or not in, an IP subnet range. IPv4 and IPv6 addresses are supported. To create a comparison: 1. Select an attribute that resolves to a valid IP address. 2. Select the **In CIDR Block** or **Not In CIDR Block** comparator. 3. Enter the IP address range as a constant or select an attribute that resolves to the IP address range. You must express the IP address range in Classless Inter-Domain Routing (CIDR) notation (the bitmask indicates the size of the routing prefix): `IP address/bitmask` For example, consider a condition that checks for IP addresses between 192.0.2.0 - 192.0.2.15. CIDR notation for this range is `192.0.2.0/28`. If the IP address attribute resolves to `192.0.2.1`, for example, the condition evaluates to `true`. **Tip** For help expressing an IP address range in CIDR notation, use a CIDR calculator.
Is In **Is Not In**	Collection, String	Checks whether a string or a collection is in, or not in, another collection.
Regular Expression	String	Checks whether a string matches a regular expression.

Copyright © 2025 Ping Identity Corporation

Comparator	Supported data types	Description
Starts With **Does Not Start With**	String	Checks whether a value starts with, or doesn't start with, another value.

Request comparators

Comparator	Description
Matches **Does Not Match**	Checks whether the inbound request name starts with, or does not start with, the conditional request name. **Note** For **Matches** to be true, the inbound request name must contain the entirety of the conditional request name. For example, if the conditional domain is `BankingChannels.OnlineBanking`, a request domain of `BankingChannels` evaluates to `false`.
Equals **Does Not Equal**	Checks whether the inbound request equals, or does not equal, the conditional request.

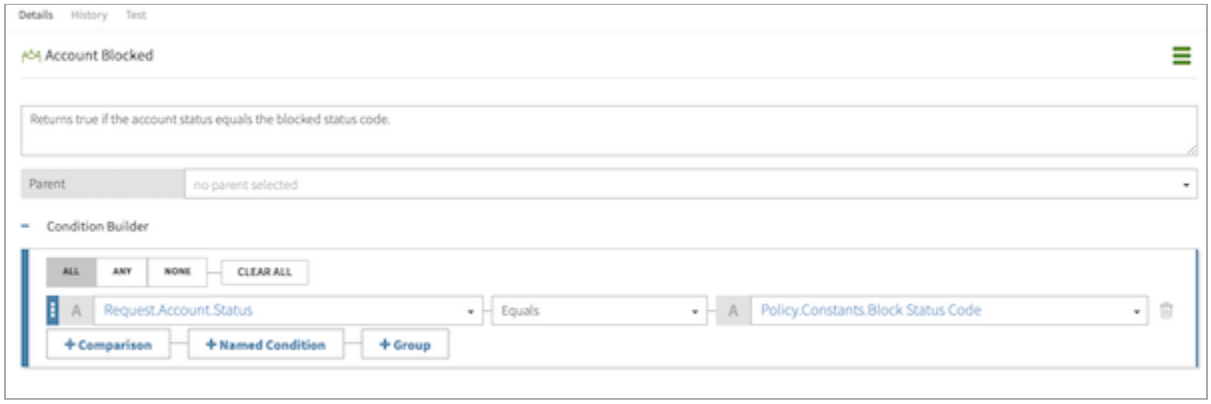
Adding a named condition

Named conditions provide a way to reuse conditional logic across attributes and policies.

About this task

Named conditions can help provide consistency in authorization logic and minimize repetition throughout policies. You can use named conditions as components in more complicated condition expressions.

For example, consider a named condition that compares the account status received in a decision request to a status code to determine if the account is blocked. You can use this condition in multiple policies to check a user's account status.



Steps

1. Click **Trust Framework** and click the **Conditions** tab.

Copyright © 2025 Ping Identity Corporation

2. Click the + icon and select **Add new Condition**.
3. Define general information for the named condition:
    1. Enter a unique **Name** for the condition.
    2. **Optional:** For **Description**, enter information that describes the condition's purpose.
    3. **Optional:** To nest the condition under a parent in the tree, in the **Parent** list, select a parent condition.

       Nesting helps group related conditions together. You can move the condition to another location in the tree by selecting a different parent condition.

       To remove nesting, click the **Delete** icon and leave the **Parent** blank.
4. To add a comparison to the condition, click + **Comparison**.

5. Select an attribute to use in the comparison, select a comparator, and then enter a constant or click the **C** icon to select an attribute.
6. To next a comparison under another comparison, click + **Group**.

   Subgroups allow more permutations in comparisons. To remove nesting while keeping the comparison, click **Ungroup**.
7. To add a named condition, click + **Named Condition**, select a named condition, and then select **is True or False**.
8. To combine multiple conditions, named conditions, or groups, select one of the following options.

   Choose from:

    - **All**: Invokes the condition when all of the conditions are true. If one condition evaluates to false, evaluation stops and the remaining conditions are not executed. This is like adding an `AND` Boolean operator between conditions.
    - **Any**: Invokes the condition when at least one of the conditions is true. If one condition evlautes to true, evaluation stops and the remaining conditions are not executed. This is like adding an `OR` Boolean operator between conditions.
    - **None**: Invokes the condition when none of the conditions are true. This like adding a `NOT` Boolean operator.
9. Click **Save Changes**.

> **Note**
>
> You can copy a named condition for reuse by selecting **Make Copy** from the hamburger menu of that condition. If you copy a named condition with children, only the parent is duplicated. You cannot copy a named condition at its point of use in a rule or policy.

Copyright © 2025 Ping Identity Corporation

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

Use the **Identity Properties** window to define objects and elements to attach to specific IdPs.

You use these properties later to map IdPs to specific identity classification levels.

Identity providers

Use the **Identity Providers** window to define different 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.

Copyright © 2025 Ping Identity Corporation

Identity classifications

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

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

Processors

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, and 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`.

Copyright © 2025 Ping Identity Corporation

```
$.score
```

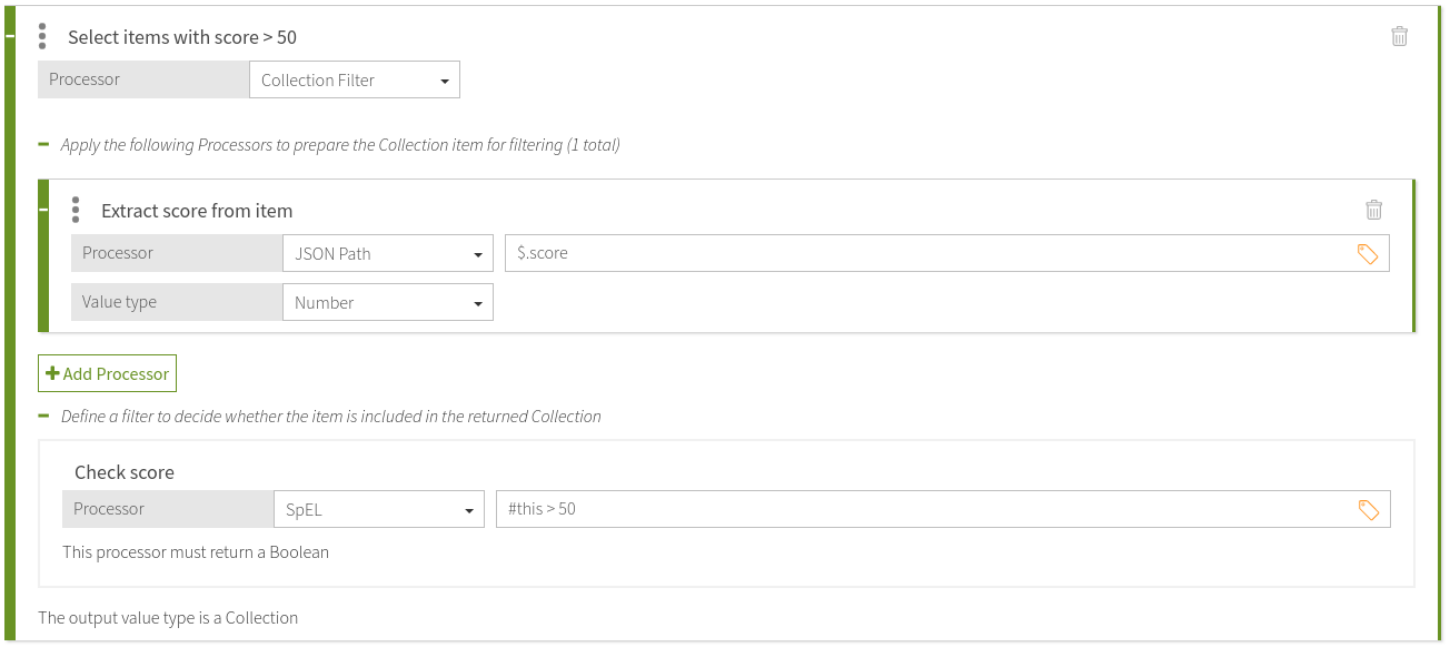
The following SpEL condition yields a true or false decision for each item.

```
#this > 50
```

Each list item is, in turn, passed through the preprocessing and the condition. The first item has a score of 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. The following image shows 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 is 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:

Copyright © 2025 Ping Identity Corporation

```
[
    { "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
```

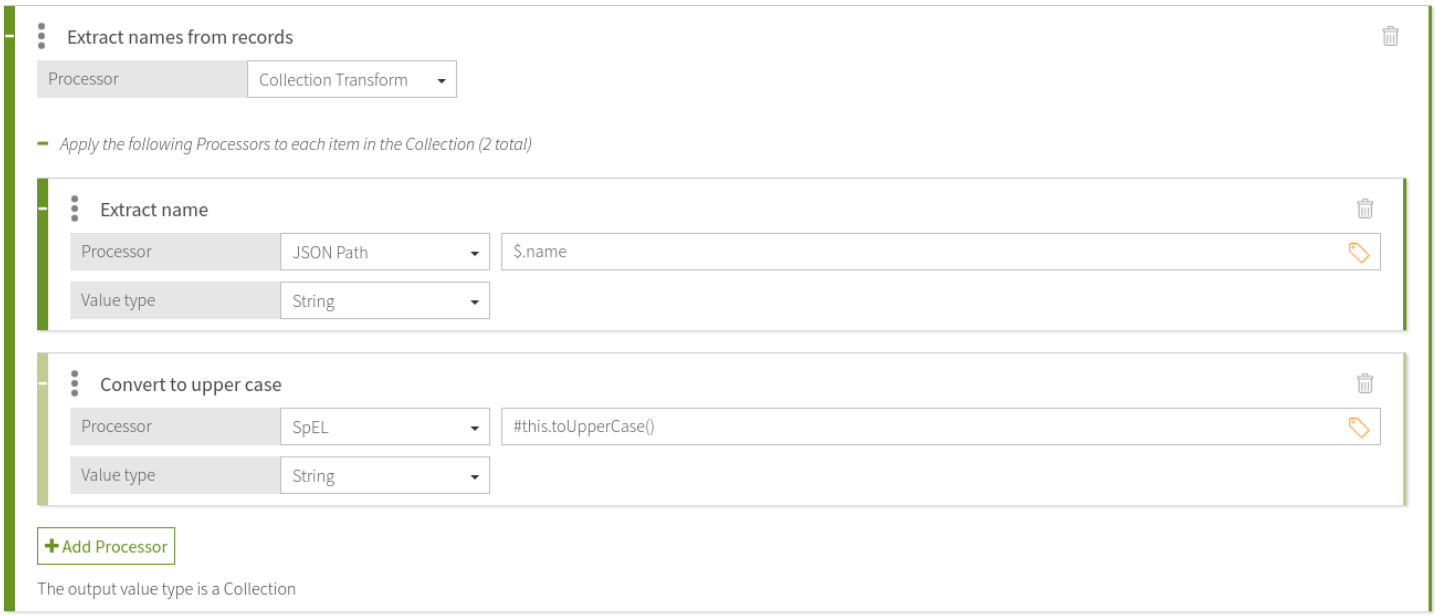
The following SpEL processor converts each `name` to upper case:

```
#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" ]
```

The following images shows 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:

Copyright © 2025 Ping Identity Corporation

```
{
   "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`.

Learn more about JSON Path expressions at https://github.com/json-path/JsonPath.

X Path

XPath is the XML-equivalent of JSON Path and follows a very similar syntax. Learn more about XPath expressions in 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 SpEL, 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:

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

Set the processor to **SpEL** and set the expression to the following text:

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

Use curly brackets to interpolate attributes directly into the SpEL expression, which can be useful if you want to combine multiple attribute values into a single value. Learn more in Attribute interpolation.

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

You can concatenate interpolated attributes with strings in SpEL expressions. In the following example, a SpEL expression is used to dynamically concatenate a string with a user's account number to create a transaction verification message:

```
"Account " + {{Customer.AccountNumber}} + " has been charged."
```


Copyright © 2025 Ping Identity Corporation

> **Note**
>
> Be careful not to wrap an interpolated attribute with quotes.

Learn more about SpEL in the official Spring Framework docs.

Learn more about the Java classes available for SpEL processing in Configuring SpEL Java classes for value processing.

You can find examples of custom SpEL functions in PingAuthorize in SpEL processing examples.

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**.

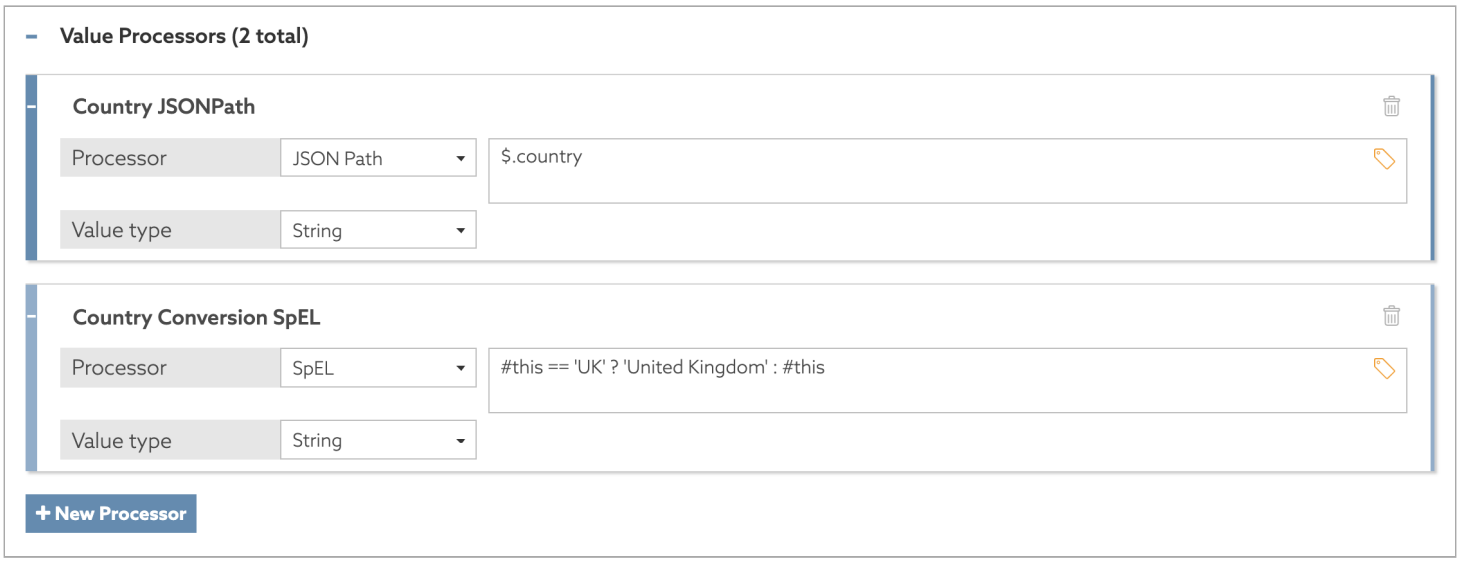
Chained value processors

You can chain processors together to combine data preprocessing steps.

For example, you can extract data using JSON Path and then apply a SpEL processor to the extracted data. Suppose you have a service response that resolves to the following JSON:

```
{
   "name": "Joe Bloggs",
   "city": "London",
   "country": "UK"
}
```

Suppose you want to extract the `country` data and transform the value to `United Kingdom`, whenever the current value is `UK`. You would add a JSON Path processor to extract the `country` value, followed by a SpEL expression to transform the value:



Copyright © 2025 Ping Identity Corporation

You can make a chained processor reusable by creating it as a named processor. That way, you can implement even more complex processing by using named processors as the building blocks.

SpEL processing examples

PingAuthorize enhances SpEL expressions with a rich library of custom functions designed to simplify complex tasks and streamline your processor configurations. The following sections introduce some of these functions with examples, focusing on common use cases for working with collections, dates and times, and Uniform Resource Identifiers (URIs).

Collection functions

To help you create and manipulate collections and maps dynamically, PingAuthorize provides a set of functions for common collection transformations in SpEL expressions.

Creates a new collection containing the specified items.

Syntax

```
#CreateList(item1, item2, …, itemN)
```

Arguments

- `item1, item2, …, itemN` (Type: Any): One or more items of any data type to include in the new collection. These items appear in the new collection in the order they're provided.

Examples

Creating a collection of strings

- SpEL: `#CreateList("view_profile", "edit_settings", "delete_account")`
- Result: `["view_profile", "edit_settings", "delete_account"]`

Creating a collection of numbers

- SpEL: `#CreateList(1, 2, 3, 4, 5)`
- Result: `[1, 2, 3, 4, 5]`

Creating a collection with mixed data types

- SpEL: `#CreateList("config_A", 100, true)`
- Result: `["config_A", 100, true]`

Combines two existing collections into a single new collection.

Syntax

```
#ConcatLists(list1, list2)
```

Arguments

- `list1` (Type: Collection): The first collection

Copyright © 2025 Ping Identity Corporation

- `list2` (Type: Collection): The second collection

The items in `list2` are appended to the end of `list1`.

Examples

Combining two collections of permissions

- SpEL: `#ConcatLists(["read_basic"], ["read_extended", "write_comments"])`
- Result: `["read_basic", "read_extended", "write_comments"]`

Combining dynamically created collections

- SpEL: `#ConcatLists(#CreateList(1, 2, 3), #CreateList(4, 5, 6))`
- Result: `[1, 2, 3, 4, 5, 6]`

Creates a new map from two separate collections: one containing the keys and the other containing the corresponding values. Each key is paired with the value at the same index.

Syntax

```
#CreateMap(keysList, valuesList)
```

Arguments

- `keysList` (Type: Collection): A collection of items used as keys in the map
- `valuesList` (Type: Collection): A collection of items used as values in the map

The item at `valuesList[i]` becomes the value for the key at `keysList[i]`.

> **Note**
>
> If `keysList` and `valuesList` have different lengths, the resulting map will only contain key-value pairs up to the length of the shorter collection. Each key in the map must be unique.

Examples

Creating a map of user attributes

- SpEL: `#CreateMap(#CreateList("username", "email", "status"), #CreateList("jdoe", "jdoe@example.com", "active"))`
- Result: `{"username": "jdoe", "email": "jdoe@example.com", "status": "active"}`

Creating a map with numeric keys and values

- SpEL: `#CreateMap(#CreateList(1, 2, 3), #CreateList(10, 20, 30))`
- Result: `{1: 10, 2: 20, 3: 30}`

Merges two existing maps into a single new map.

Syntax

```
#MergeMaps(map1, map2)
```

Copyright © 2025 Ping Identity Corporation

Arguments

- `map1` (Type: Map): The first map
- `map2` (Type: Map): The second map

> **Note**
>
> If both `map1` and `map2` contain the same key, the value from `map2` overrides the value from `map1` in the resulting map.

Example: Merging maps with distinct keys

- SpEL: `MergeMaps({#CreateMap(#CreateList(1, 2), #CreateList(10, 20))}, {#CreateMap(#CreateList(3, 4), #CreateList(30, 40))})`
- Result: `{1: 10, 2: 20, 3: 30, 4: 40}`

Date and time functions

PingAuthorize provides a set of functions for common date and time manipulations in SpEL expressions. These functions leverage Java's `java.time` API and help you parse, convert, and calculate durations involving dates and times. Learn more in the `java.time` API specification in the Java documentation.

Parses a date-time string using a specified custom format. This function can parse strings into `LocalDateTime` (date-time without time zone) or `ZonedDateTime` (date-time with time zone), depending on the input string and format.

Syntax

```
#DateTime(dateString, formatPattern)
```

Arguments

- `dateString` (Type: String): The date-time string to be parsed
- `formatPattern` (Type: String): A pattern string that defines the format of the `dateString`. The pattern should conform to Java's DateTimeFormatter patterns (`"uuuu-MM-dd HH:mm:ss"` , `"MM/dd/uuuu'T'HH:mm:ssXXX"`).

Example: Parsing a date-time with a custom format

- SpeL: `#DateTime("11:09:08 07-22-2018", "HH:mm:ss MM-dd-uuuu")`
- Result: A `LocalDateTime` object representing July 22, 2018, 11:09:08

Parses a date-time string that conforms to the ISO 8601 standard. This function can parse strings into `LocalDateTime` or `ZonedDateTime` .

Syntax

```
#ISO8601DateTime(dateString)
```


Copyright © 2025 Ping Identity Corporation

Arguments

- `dateString` (Type: String): The date-time string in ISO 8601 format (`"2019-01-01T10:01:01"` , `"2019-01-01T10:01:01Z"` , `"2019-01-01T10:01:01+02:00"`)

Example: Parsing an ISO 8601 local date-time string

- SpEL: `#ISO8601DateTime("2024-05-30T15:45:00")`
- Result: A `LocalDateTime` object representing May 30, 2024, 15:45:00

Parses a date string using a specified custom format into `LocalDate` (a date without time-of-day and time zone).

Syntax

```
#Date(dateString, formatPattern)
```

Arguments

- `dateString` (Type: String): The date string to be parsed
- `formatPattern` (Type: String): A pattern string that defines the format of the `dateString` (`"uuuu-MM-dd"` , `"MM/dd/uuuu"`)

Example: Parsing a date with "year-month-day" format

- SpEL: `#Date("2024-12-25", "uuuu-MM-dd")`
- Result: A `LocalDate` object representing December 25, 2024

Parses a date string that conforms to the ISO 8601 standard (YYYY-MM-DD) into `LocalDate` .

Syntax

```
#ISO8601Date(dateString)
```

Arguments

- `dateString` (Type: String): The date string in ISO 8601 format (`"2019-01-01"`)

Example: Parsing an ISO 8601 date string

- SpEL: `#ISO8601Date("2019-01-01")`
- Result: A `LocalDate` object representing January 1, 2019

Converts a Unix timestamp (seconds since the 1970-01-01T00:00:00Z epoch) to `LocalDateTime` , representing the date and time in the system's default time zone.

Syntax

```
#LocalDateTimeFromUnix(unixTimestamp)
```

Arguments

`unixTimestamp` (Type: Long): The Unix timestamp value in seconds

Copyright © 2025 Ping Identity Corporation

Example: Converting a Unix timestamp

- SpEL: `#LocalDateTimeFromUnix(1572348852L)`
- Result: A `LocalDateTime` object. For the specified timestamp, this corresponds to `2019-10-29T11:34:12`.

Converts `LocalDateTime` to a Unix timestamp (seconds since the 1970-01-01T00:00:00Z epoch). The conversion assumes the `LocalDateTime` is in Coordinated Universal Time (UTC).

Syntax

```
#UnixFromLocalDateTime(localDateTime)
```

Arguments

`localDateTime` (Type: `LocalDateTime`): The `LocalDateTime` to be converted

Example: Converting a specific LocalDateTime to Unix time

- SpEL: `#UnixFromLocalDateTime(#ISO8601DateTime('2019-10-29T11:34:12'))`
- Result: 1572348852L

Calculates the time period (in years, months, and days) between two `LocalDate` objects.

Syntax

```
#GetPeriodBetween(startDate, endDate)
```

Arguments

- `startDate` (Type: `LocalDate`): The start date of the period (inclusive)
- `endDate` (Type: `LocalDate`): The end date of the period (exclusive)

Returns

A `Period` object representing the duration between `startDate` and `endDate`. You can access the result's components using properties like `.years`, `.months`, and `.days`.

Example: Calculating the period between two dates

- SpEL: `#GetPeriodBetween(#ISO8601Date("2018-02-15"), #ISO8601Date("2019-04-25"))`
- Result:
  - `periodObject.years` = 1
  - `periodObject.months` = 2
  - `periodObject.days` = 10

URI functions

PingAuthorize provides a set of functions for parsing and constructing URIs in SpEL expressions. These functions leverage Java's `java.net.URI` capabilities to simplify the manipulation of URI components.

Copyright © 2025 Ping Identity Corporation

Parses a URI string into its components and returns a map that associates each component name with its corresponding value.

Syntax

`#URIParse(uriString)`

Arguments

- `uriString` (Type: String): The complete URI string to be parsed

Examples

Parsing a complex URI

- SpEL: `#URIParse('https://jane.doe:s3cr3t@api.example.com:8443/v1/resources?type=widget&id=42#details')`
- Result:

```
{
  "scheme": "https",
  "userInfo": "jane.doe:s3cr3t",
  "host": "api.example.com",
  "port": 8443,
  "path": "/v1/resources",
  "query": {
    "type": "widget",
    "id": "42"
  },
  "fragment": "details"
}
```

Parsing a URI with a valueless query parameter

- SpEL: `#URIParse('http://example.com/search?q=ping&enabled')`
- Result:

```
{
  "scheme": "http",
  "host": "example.com",
  "path": "/search",
  "query": {
    "q": "ping",
    "enabled": ""
  }
}
```

Constructs a URI string from a map containing the URI's individual components.

Syntax

`#URIBuild(componentsMap)`

Copyright © 2025 Ping Identity Corporation

Arguments

- `componentsMap` (Type: Map<String: Any>): A map that associates URI component names with the corresponding values used to construct the URI

  The following table describes the allowable URI component names:

Component	Data type	Description	Required
`scheme`	String	The URI scheme.	Required
`host`	String	The host name or IP address.	Required
`userInfo`	String	Information about the user.	Optional
`port`	Integer	The port number. If this value is `null` or not provided, the port isn't included in the URI string.	Optional
`path`	String	The path component. For a root path, use `/`. If this value is empty or `null`, the URI will have no path component after the authority.	Optional
`query`	Map<String, Any>	A map of query parameter names and their corresponding values.	Optional
`fragment`	String	The fragment identifier.	Optional

Example: Building a simple URI

- SpEL: `#URIBuild(#{'scheme': 'http', 'host': 'localhost', 'path': '/status'})`
- Result: `"http://localhost/status"`

Copying elements

To build your authorization logic more quickly and accurately, you can make editable copies of many of your Policy Editor entities.

About this task

Within the **Policies** and **Trust Framework** pages, you can copy whole definitions, policies, rules, statements, and **Library** entities. See the following table for page-specific capabilities and limitations.

Copyright © 2025 Ping Identity Corporation

Policy Editorpage	Capabilities and limitations
Trust Framework	• You can copy any definition. • With the exception of resolvers, you can't copy a portion of a definition. ◦ Copy the whole definition to duplicate any of its content, or manually copy the element between definitions. • When you copy a definition that contains child definitions, only the parent definition is duplicated. • You can't copy definition hierarchy trees.
Policies	• You can copy any policy, rule, or statement. • You can't copy policy sets. • You can't copy targets, conditions, or properties within a policy or rule. • When you copy policies containing rules, targets, or statements belonging to the **Library**, these entities get reused in the new policy, not copied. • When you copy rules or statements within a policy, you must save the policy.

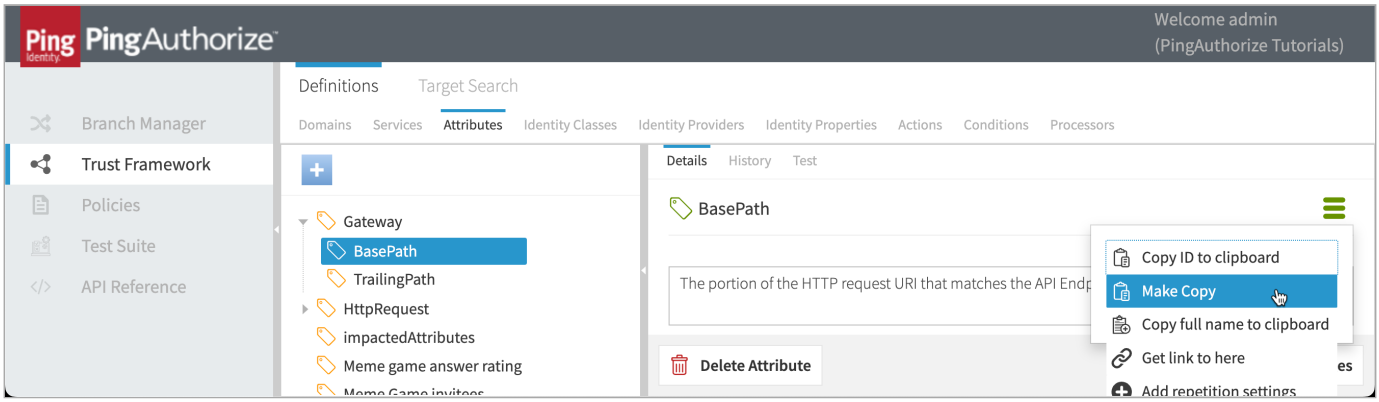
Library	• On the **Library** tab, you can copy top-level **Library** rules, targets, and statements.
	  ◦ You can also copy statements within rules.
	  ◦ You can't copy targets within rules.
	• You can copy **Library** rules and statements at their point of use.
	  ◦ When you make copies of **Library** entities within rules or policies, the copies display at the point of use and on the **Library** tab.
	• Where relevant, the copying capabilities and limitations listed for the **Policies** page apply to **Library** entities.

Trust Framework, **Policies**, and **Library**	• You can't use **Make Copy** in the tree structure view. • You must save any unsaved changes to an entity before making a copy.

Steps

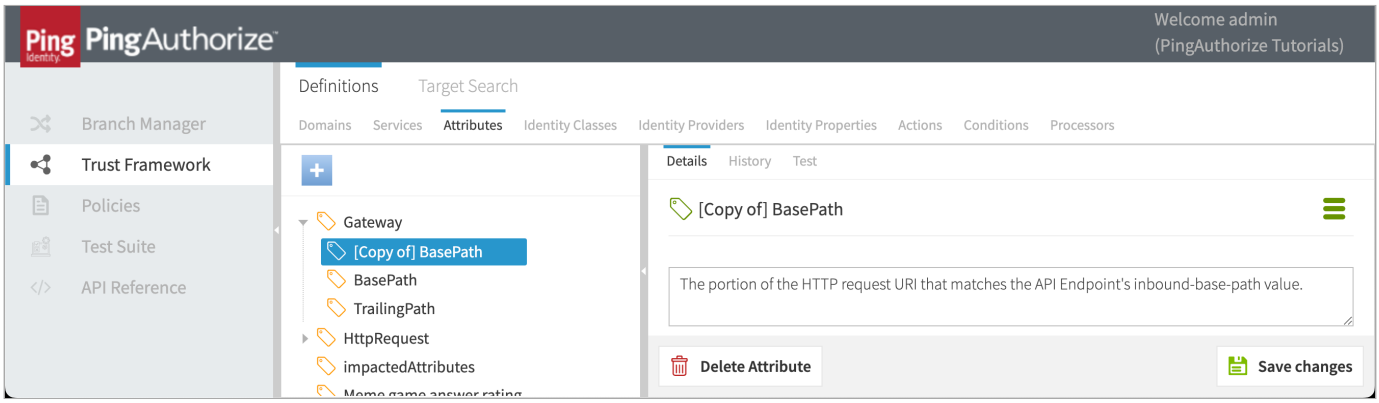
1. To create a copy of a Policy Editor entity, click **Make Copy** in the hamburger menu.

Copyright © 2025 Ping Identity Corporation



Result:

The Policy Editor creates a duplicate of the entity and its contents. The new duplicate's name is formed by adding the prefix **[Copy of]** to the original name, as in the following image.



2. **Optional:** Make edits to the new entity.

Deleting persistent copies

About this task

If you've upgraded from a version of PingAuthorize earlier than 10.1, copies of policy elements can persist in the UI when you try to delete them. Resolve this problem by deleting the original elements after you've made a backup copy.

Steps

1. Back up your policy database.
2. Make a new copy of the original element.
3. Delete the original element.

   Result:

   The original element and its persistent copies are deleted from the Policy Editor.
4. Rename the copy that you created in step 2 and remove **[Copy of]**.

You can now delete copies of the newly created policy element without experiencing persistence issues.

Copyright © 2025 Ping Identity Corporation

Testing Trust Framework elements

Test Trust Framework elements from end-to-end with tools that show the complete processing flow.

About this task

Testing enables you to validate the type conversion of policy information points (PIP) included in decision requests and the logical behavior of conditions.

Although policy testing indirectly tests your Trust Framework elements, testing each Trust Framework element individually allows you to immediately view the output of element processing without parsing policy test results.

Use Trust Framework testing to:

- Verify attribute or service value settings.

  If you are testing an attribute with a value type other than `String`, the decision engine must convert this attribute from a string provided in the request to the correct type. Testing an attribute or service allows you to verify that the formatting required for this conversion is correct.

- Test the behavior of processors.

  The decision engine's conversion of request parameters from string to other data types, such as `Number`, might involve a chain of processors configured in the attribute definition. Testing an attribute allows you to determine before runtime whether such processing produces the desired outcome.

- Test the behavior of resolvers.

  If you define multiple resolvers on an attribute, you can use testing to confirm the order in which the decision service checks these resolvers for a value.

- Test the logic of conditions.

  With testing, you can confirm that your conditional logic produces the desired outcome before implementing that logic in your authorization policies.

Steps

1. Go to **Trust Framework**.
2. Click the tab of the appropriate Trust Framework element type, select the element that you want to test, and then click the **Test** tab.
3. Define the test scenario:
    1. In the **Domain** list, select any domains you want to include as request parameters.
    2. In the **Service** list, select any services you want to include as request parameters.
    3. In the **Identity Provider** list, select any identity providers (IdPs) you want to include as request parameters.
    4. In the **Action** list, select any actions you want to include as request parameters.
    5. In the **Attributes** list, select the attribute that you want to test and any other attributes that you want to include as request parameters, and provide sample values.

Copyright © 2025 Ping Identity Corporation

> **Tip**
>
> Click the angle brackets next to an attribute to display a JSON text editor.

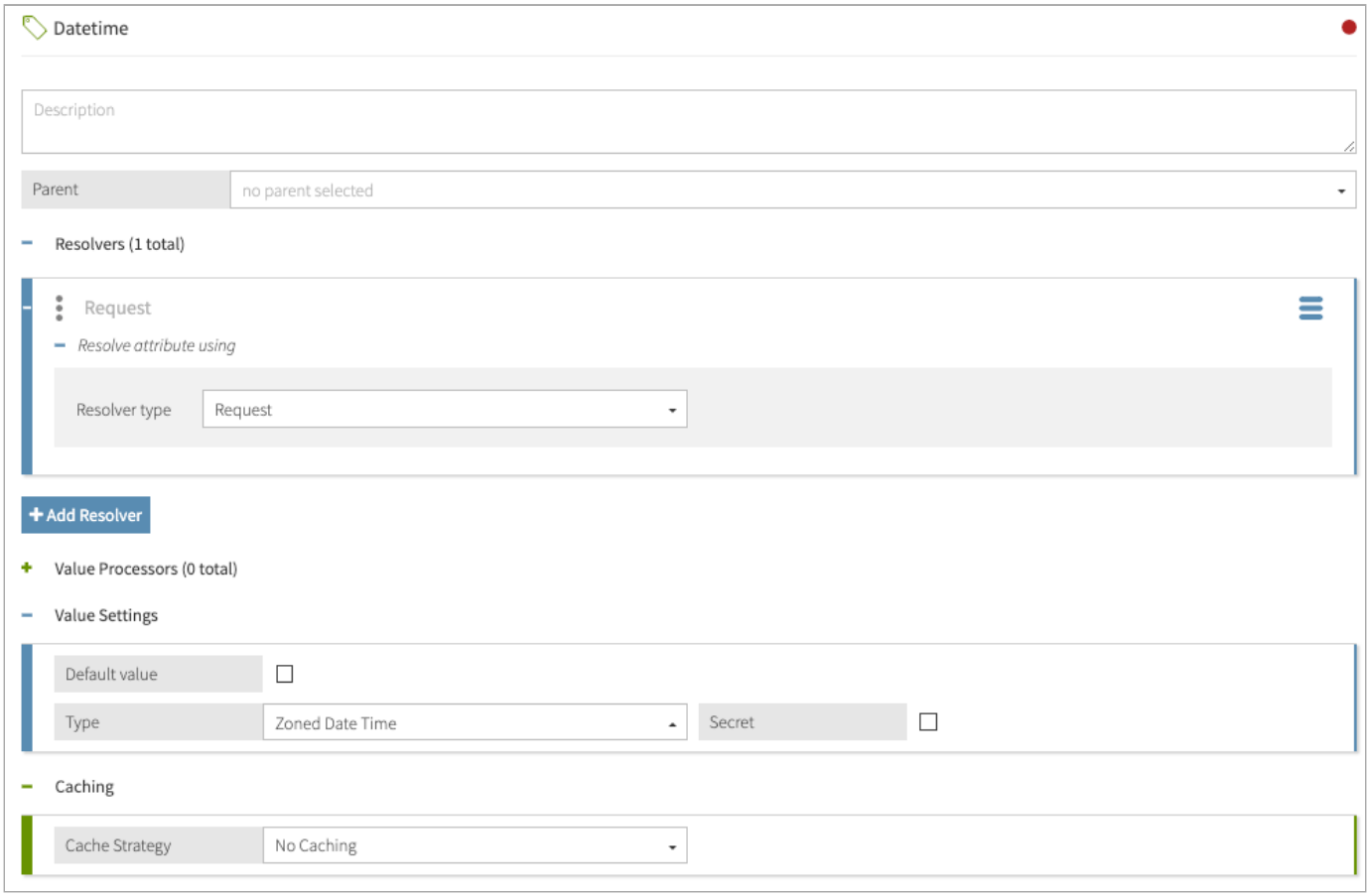
6. In the **Overrides** section, configure attribute and service values if you want to override those elements' default behavior.

   For example, if an attribute is defined with a request parameter resolver and no value is specified in the test request, the decision engine resolves that attribute from the **Overrides** configuration.

   > **Note**
   >
   > You can override any attribute's value, regardless of its resolution or processing details.

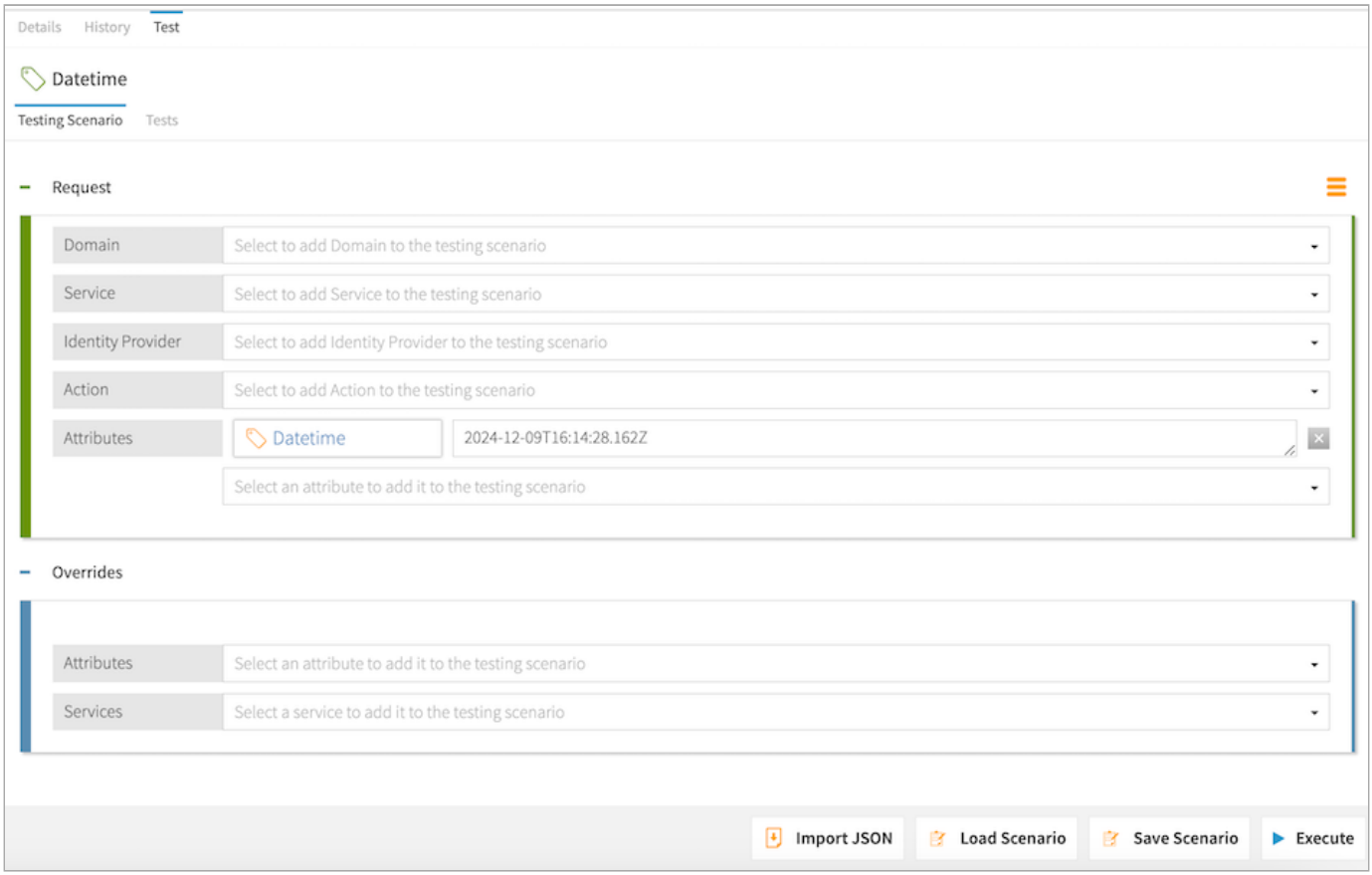
Example:

The **Datetime** attribute defined in the following example has a value type of **Zoned Date Time**, resolves from a decision request parameter, and has no processors defined. Testing this attribute verifies whether the decision engine can directly convert this attribute from a string.



The following scenario tests the **Datetime** attribute with an ISO 8601-formatted string as input:

Copyright © 2025 Ping Identity Corporation

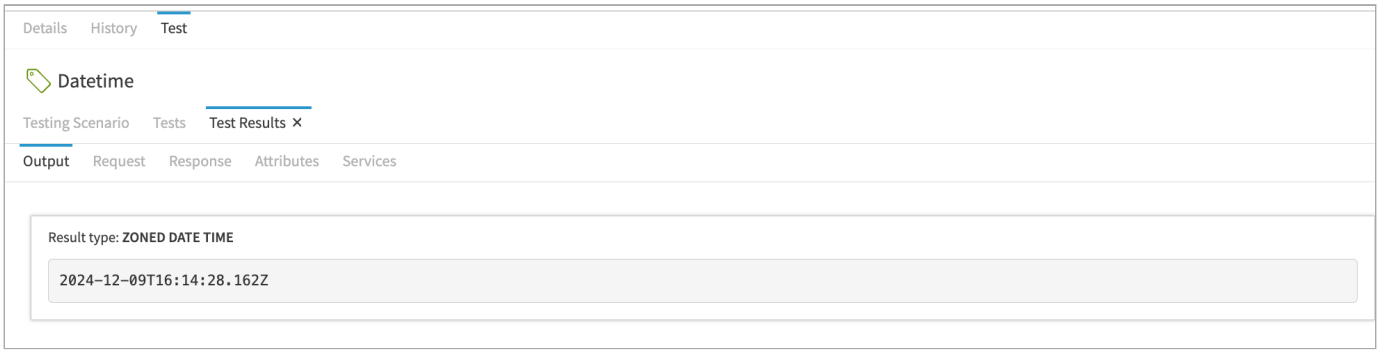


4. Click **Execute**.

> **Note**
>
> If the test scenario exceeds 50 kB in size, sample values will not be stored in the local cache after executing the test request.

Result:

The **Output** tab displays the value and type of the attribute being tested.



5. Click the other tabs for additional details:
   - **Request**: Shows the JSON request sent to the decision, allowing you to confirm the expected information was sent.

Copyright © 2025 Ping Identity Corporation

- **Response**: Shows the complete, high-verbosity response for the decision, including expanded errors and other helpful information.

  This tab can be helpful when troubleshooting value processing failures.

- **Attributes**: Shows details about the attributes used in the decision.

- **Services**: Shows details about the services used in the decision.

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. Additionally, 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 standard, 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.

Learn more about these variables in Repeating policies and attributes.

Viewing dependents

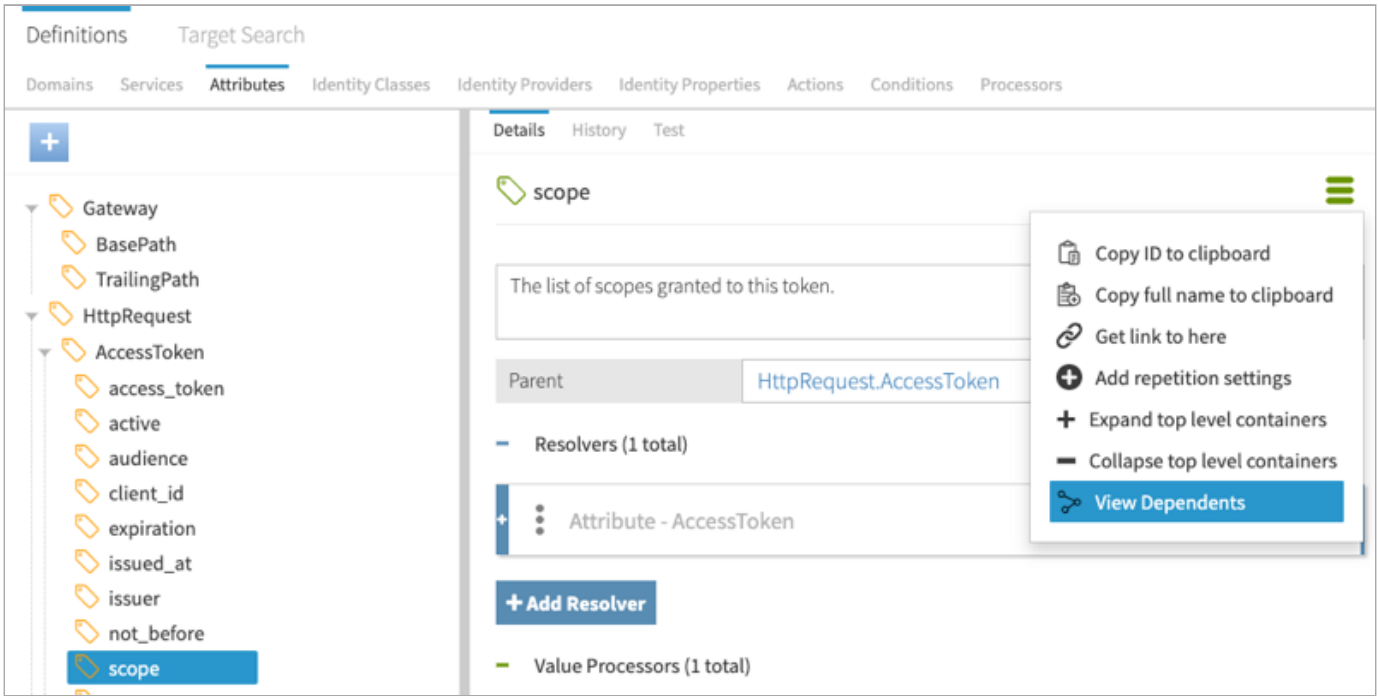
Before you change an entity, check what depends on that entity so that you do not introduce unintended consequences.

Steps

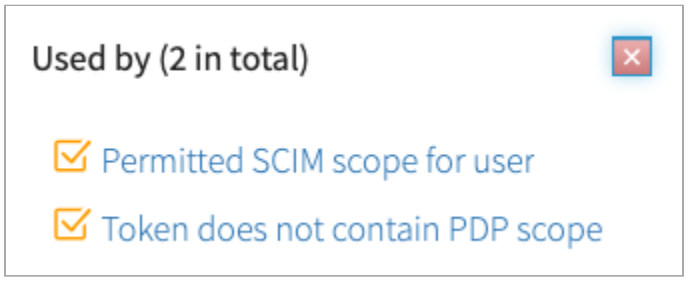
1. Select the Trust Framework entity.
2. Click the hamburger menu to the right of the entity name.
3. Click **View Dependents**.

   The following image shows the `scope` attribute.

Copyright © 2025 Ping Identity Corporation



If the entity has dependents, they are displayed in a new window. For example, the following image shows the dependents for `scope`.



> **Note**
>
> Any entity restricted by self-governance will be omitted from the list of dependents. See Self-governance for more information.

Policy management

The PingAuthorize Policy Editor 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.

The Policy Manager, located on the **Policies** page, enables you to build 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:

Copyright © 2025 Ping Identity Corporation

> 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 that 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 Policy Manager reflects the structure of grouping rules for attribute-based access control (ABAC) 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 protected resource.

You can group these rules together naturally, so that 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. The Policy Manager allows you to organize these business rules into a tree structure, with a root policy set that contains policy sets for each business case. The policy sets contain related policies, and each policy contains one or more rules that help define its behavior.

> **Note**
>
> A root policy set is not required, but it is useful for building a deployment package from the entire policy tree. Individual policies often belong to policy sets but can also stand on their own.

Creating policies and policy sets

Create policies and policy sets to define the circumstances under which users access specific resources.

Steps

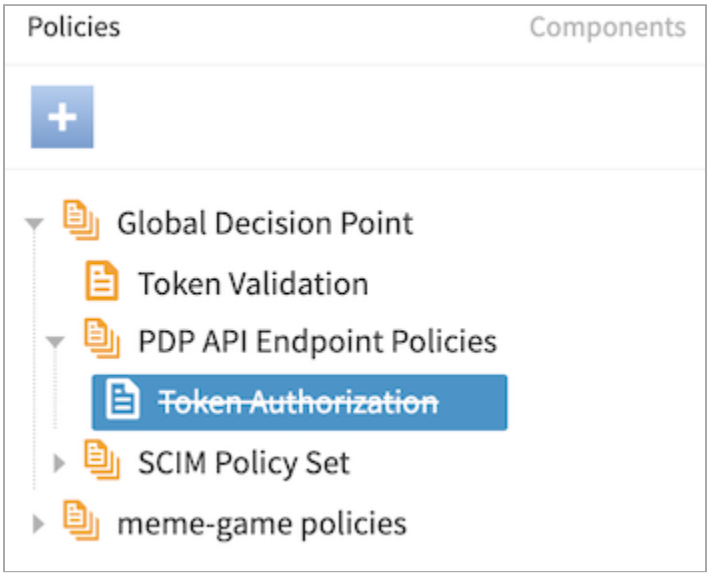
1. Click **Policies**.
2. Click **+**.
3. Select **Add Policy Set** or **Add Policy**, as appropriate.

   You can name policies and policy sets anything you like. However, you should 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 represent the operational policies of the organization.
4. Update the policy to include targets, statements, and other changes.

Copyright © 2025 Ping Identity Corporation

5. **Optional:** Use **Properties** to add metadata to the policy or policy set in the format of a key-value pair.
    1. Click **+** next to **Properties**.
    2. Click **Add Property** and enter a key-value pair.
6. **Optional:** Select the **Disable** check box to disable your policy or policy set.

    If you disable the policy, it is not evaluated and produces a `Not Applicable` decision. When you disable it, the policy is shown as crossed out in the policy tree.



> **Note**
>
> You can also disable rules. If a rule is unreachable because of the rule structure and combining algorithm, disabling that rule has no effect on the final decision.

7. 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.

Example

In the following example, the policy name is `My Basic Policy`. Because the name has been changed, you see a red dot in the upper-right corner. This dot indicates that the policy contains unsaved changes. If you try to leave the page, a modal opens and prompts you to save your changes.

Copyright © 2025 Ping Identity Corporation

Adding targets to a policy

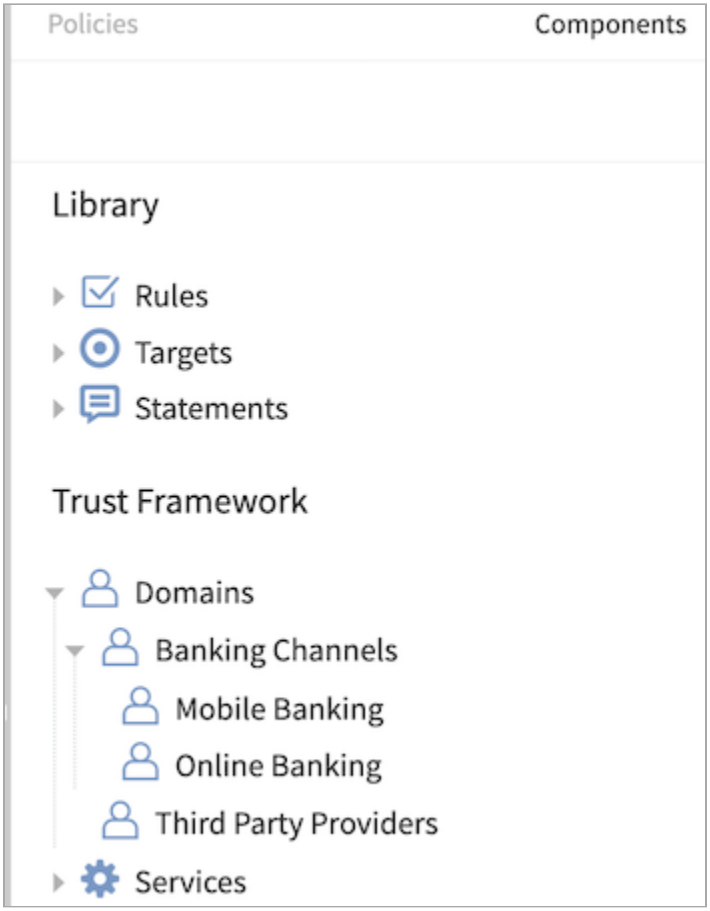
Add targets to identify the requests to which the policy applies its fine-grained authorization logic. If no targets are attached to a policy, the policy applies to all requests. To make a policy only apply for all requests to a certain database, for example, add the database domain as a target.

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** and **Mobile Banking** domains. Because the top level is also a target, this step adds a total of three targets.



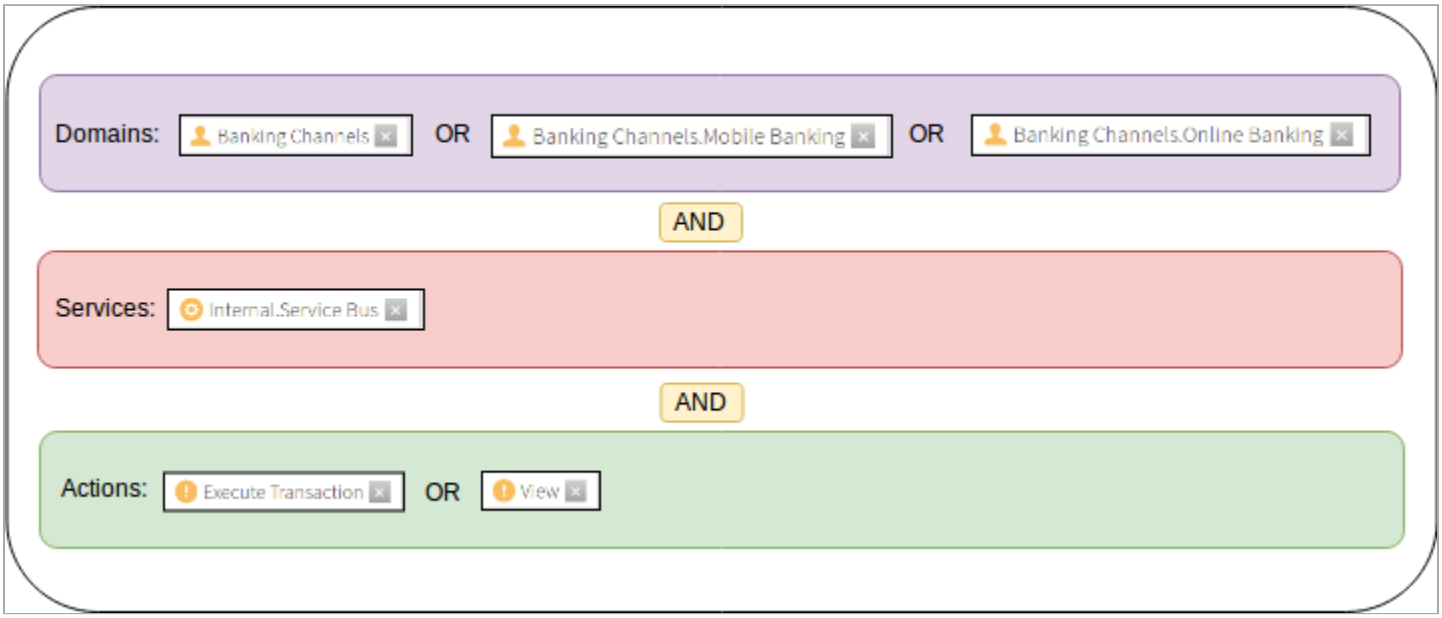
4. Click **Save changes**.

Example

Copyright © 2025 Ping Identity Corporation

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 to)

You can use conditional targets to extend the capability of the "Applies to" concept when creating attribute-based access control (ABAC) 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.

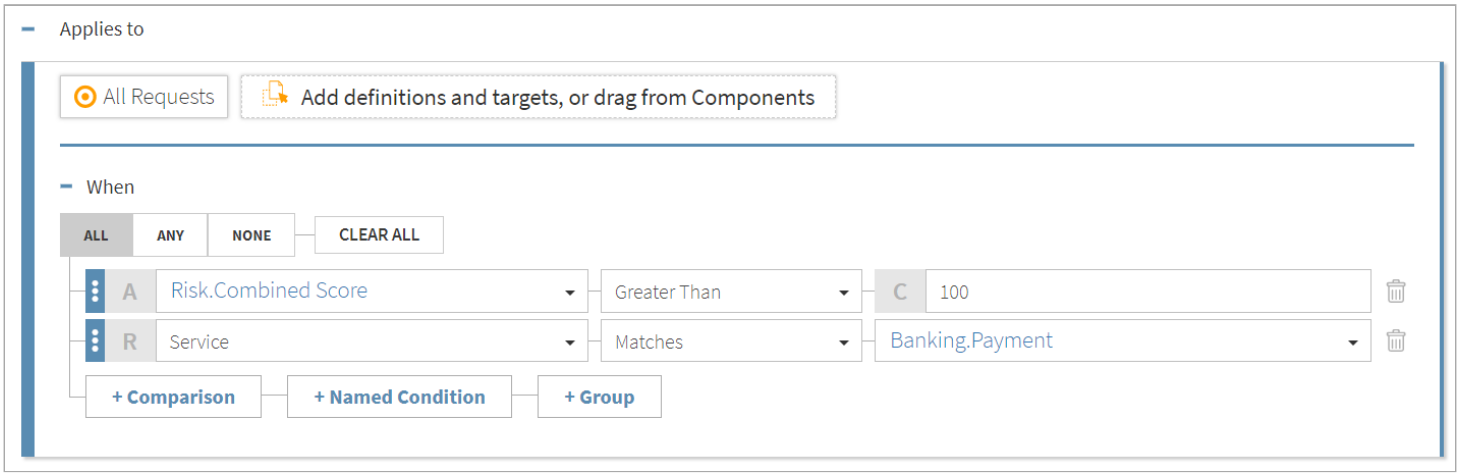
To enable this functionality, click **Applies to** and then **When**.

You can include the following types of conditions in a logical expression:

- Attribute comparison – Allows the comparison of an attributes 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. See Conditions for more information.

Copyright © 2025 Ping Identity Corporation



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, and 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.

Statements

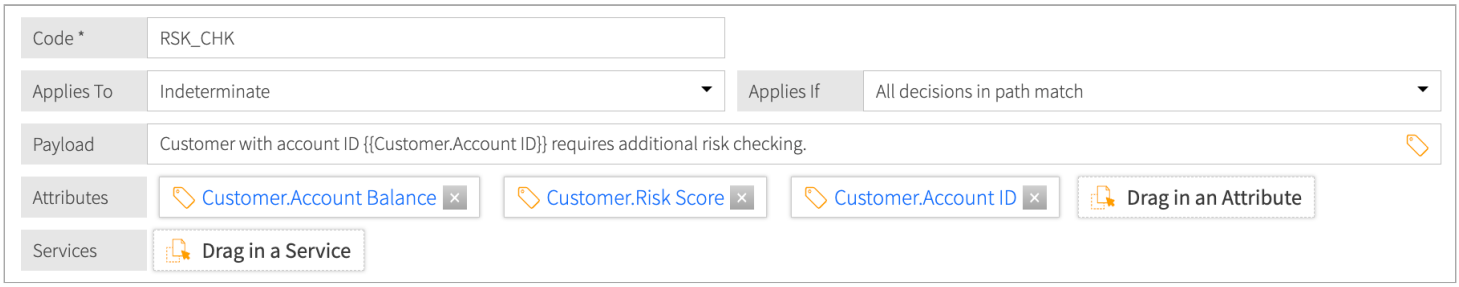
Statements are directives that instruct the policy enforcement point (PEP) to perform additional processing in conjunction with an authorization decision.

When a policy is applied to a request or response, the policy result might include one or more statements. Statements enable the PEP to do more than allow or deny access to an API resource. For example, a statement can:

- Cause the removal of a specific set of fields from a response
- Provide details about the reason for denying access to a user

You can add a statement directly to a single policy or rule and modify that statement as part of a policy definition. You can also add a statement in **Components** for use with multiple policies or rules.

Copyright © 2025 Ping Identity Corporation



Statements are sometimes called obligations. An obligation is a statement that must be fulfilled as a condition of authorizing the decision request.

Each statement contains the following mandatory fields:

Name

Human-readable label for reference in the Policy Manager

Code

Identifier that distinguishes between different types of statements

Applies To

Type of decision to which the statement is attached

Applies If

Condition under which the statement is returned in the decision response

> **Note**
>
> If the **Applies To** criteria for a statement is met by its associated rule or policy decision, and that decision contributes to the final result, PingAuthorize uses the statement in its final response if the statement's associated **Applies If** condition is satisfied. Select an option to exercise precise control over when a qualifying statement gets returned in a decision response, which can make it easier to provide reasons for both `permit` and `deny` decisions and risk evaluation feedback.

Statements carry additional data in the form of payloads and attributes:

- 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 statement.

To indicate that the final decision applies only if a statement can be fulfilled, mark the statement as **Obligatory**. Typically, the service that calls PingAuthorize Server handles this responsibility.

You can reorder collapsed statements by dragging the handles on the left. To reorder using the keyboard, press Tab to go to the statement, press Enter to select the statement, press the Up Arrow key or Down Arrow key to go to the desired location, and press Enter to drop the statement in the new location.

The following table identifies significant statement properties.

Copyright © 2025 Ping Identity Corporation

Property	Description
`Name`	Friendly name for the statement.
`Obligatory`	When marked as **Obligatory**, the statement must be fulfilled as a condition of authorizing the request: • If PingAuthorize can't fulfill an obligatory statement, it fails the operation and returns an error to the client application. • If PingAuthorize can't fulfill a non-obligatory statement, the server logs an error, but the client's requested operation continues.
`Code`	Identifies the statement type. This value corresponds to a statement ID that the PingAuthorize configuration defines.
`Applies To`	Specifies the decision types that should include the associated statement with the result. Available types include: • **Applicable** ◦ This is the default option. Select **Applicable** if the statement should apply to any of the following decision types. • **Permit** • **Deny** • **Permit or Deny** • **Indeterminate**
`Applies If`	Specifies how the statement propagates through the decision tree and whether it is returned in the overall decision response. Available options include: • **All decisions in path match**: The statement is returned when the decision for the rule or policy with which the statement is associated matches all decisions in the path. For example, when the decision for a rule with which the statement is associated is `permit`, and all decisions in the path are `permit`, the statement is returned. This is the default option. • **Final decision in path matches**: The statement is returned when the decision for the rule or policy with which the statement is associated matches the overall decision. For example, when the decision for the rule with which the statement is associated is `permit`, and the overall decision is `permit`, the statement is returned even if there are `deny` decisions in between. • **All decisions in path are applicable**: The statement is always returned, unless an error occurs in the associated decision.
`Payload`	Set of parameters governing the actions that the statement performs when PingAuthorize applies the statement. The appropriate payload value depends on the statement type.

PingAuthorize Server supports all of the provided statement types except for custom statements. To develop custom statement types, use the PingAuthorize Server SDK.

Copyright © 2025 Ping Identity Corporation

> **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 statements

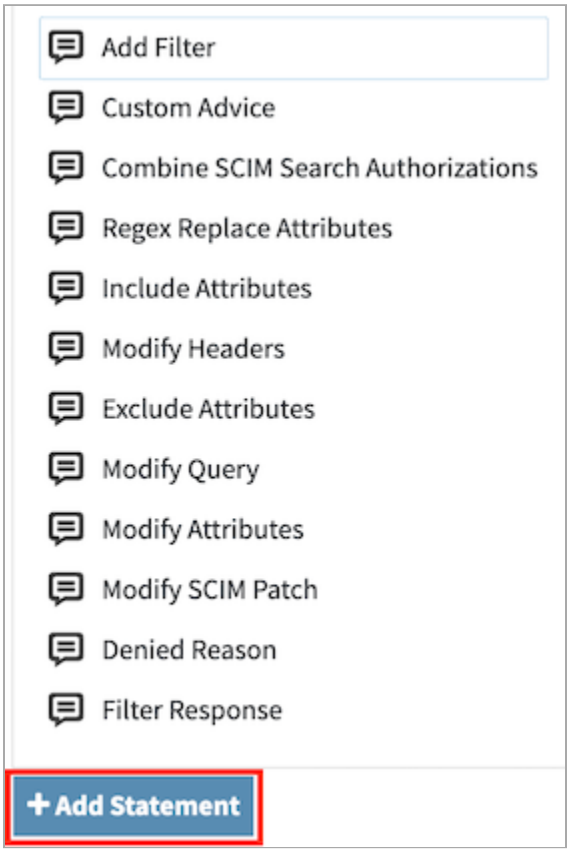
The PingAuthorize Policy Editor comes with preconfigured statement types that are also in PingAuthorize Server.

Policy writers can use the following provided statements out of the box, and PingAuthorize Server fulfills the statements as documented:

- Add Filter
- Combine SCIM Search Authorizations
- Denied Reason
- Exclude Attributes
- Filter Response
- Include Attributes
- Modify Attributes
- Modify Headers
- Modify Query
- Modify SCIM Patch
- Regex Replace Attributes

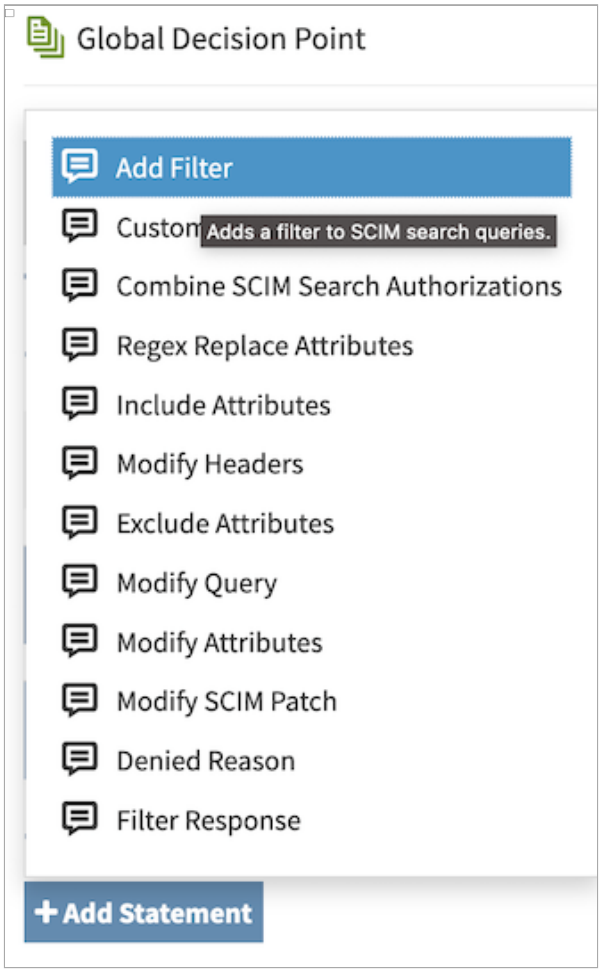
To view the set of provided statement types within the Policy Editor, click **+ Add Statement**.

Copyright © 2025 Ping Identity Corporation

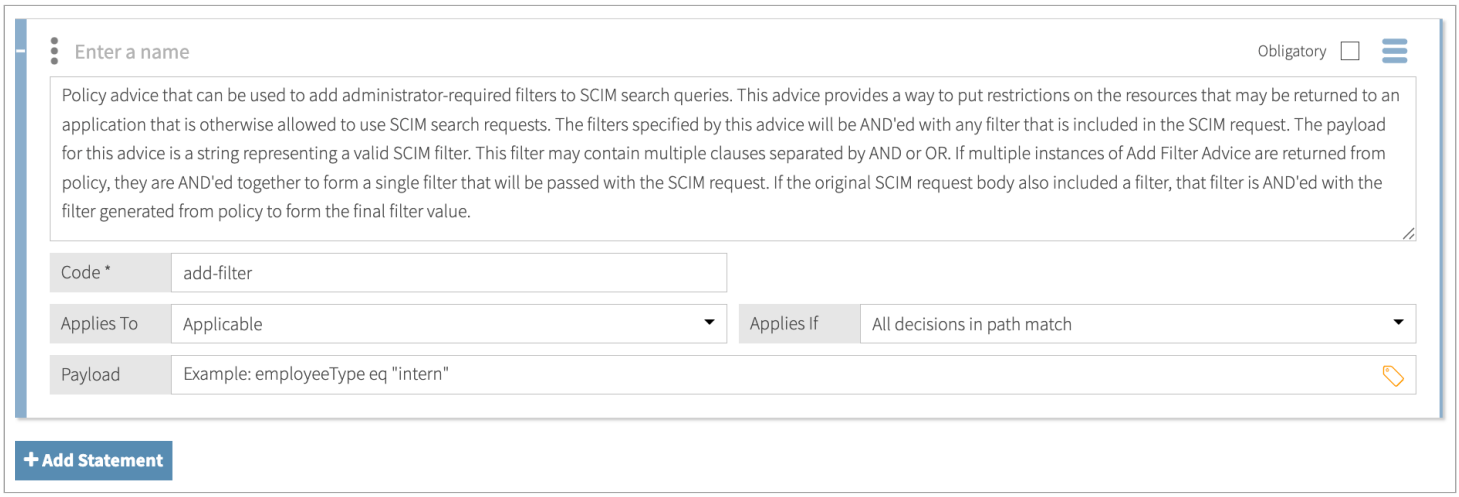


You can hover over a statement type to view its description.

Copyright © 2025 Ping Identity Corporation



Selecting a statement 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.



The following sections describe the statement 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.

Copyright © 2025 Ping Identity Corporation

Description	Details
Applicable to	SCIM
Additional information	The Add Filter statement places restrictions on the resources returned to an application that can otherwise use SCIM search requests. The filters that the statement specifies are `AND`ed with any filter that the SCIM request includes. The payload for this statement 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 the Add Filter statement, 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 SCIM search responses.

Description	Details
Applicable to	SCIM
Additional information	By default, SCIM search responses are authorized by generating multiple policy decision requests with the `retrieve` action, one for each member of the result set. The default mode enables policy reuse but might result in greater overall policy processing time. When you use this statement 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. Statements that the policy result returns are applied iteratively against each matching SCIM resource, allowing for the modification or removal of individual search results. This statement type does not use a payload. For more information about SCIM search handling, see About SCIM searches.

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 statement only applies to SCIM searches using the optimized search response authorization mode.

Copyright © 2025 Ping Identity Corporation

Description	Details
Additional information	The payload for Denied Reason statements 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 statements directly to a SCIM search. Also, do not use this statement type with SCIM modifies. Instead, use the Modify SCIM Patch statement type.
Additional information	The payload for this statement 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"]` Learn more about the processing of SCIM searches in Filter Response.

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 the gateway, although you cannot apply Filter Response statements directly to a SCIM search. However, the SCIM service performs similar processing automatically when it handles a search result. For every candidate resource in a search result, the SCIM service makes a policy request for the resource with an `Action` value of `retrieve`.

Copyright © 2025 Ping Identity Corporation

Description	Details
Additional information	When presented with a request to permit or deny a multivalued response body, Filter Response statements allow 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 list identifies the fields of the JSON object that represents the payload for this statement. **`Path`** JSONPath to an array within the API's response body. The statement implementation iterates over the nodes in this array and makes a policy request for each node. This field is required. **`Action`** Value to pass as the `action` parameter on subsequent policy requests. If no value is specified, the action from the parent policy request is used. This field is optional. **`Service`** Value to pass as the `service` parameter on subsequent policy requests. If no value is specified, the service value from the parent policy request is used. This field is optional. **`ResourceType`** Type of object contained by each JSON node in the array, selected by the `Path` field. On each subsequent policy request, the contents of a single array element pass to the policy decision point as an attribute with the name that this field specifies. If no value is specified, the resource type of the parent policy request is used. This field is optional. On each policy request, if policy returns a `deny` decision, the relevant array node is removed from the response. If the policy request returns a `permit` decision with additional statements, the statements are fulfilled within the context of the request. For example, this statement allows the 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 statement type can descend through the JSON content of the response. **Note** Performance might degrade as the total number of policy requests increases.

Include Attributes

Use `include-attributes` to limit the attributes that a JSON response can return.

Description	Details
Applicable to	PERMIT decisions, although you cannot apply Include Attributes statements directly to a SCIM search. Also, do not use this statement type with SCIM modifies. Instead, use the Modify SCIM Patch statement type.

Copyright © 2025 Ping Identity Corporation

Description	Details
Additional information	The payload for this statement 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 statements, the response includes the union of all selected attributes. Learn more about the processing of SCIM searches in Filter Response.

Modify Attributes

Use `modify-attributes` to modify the values of attributes in the JSON request or response.

Description	Details
Applicable to	All, although you can't apply the Modify Attributes statement directly to a SCIM search. Also, do not use this statement type for SCIM modify operations. Instead, use the Modify SCIM Patch statement type.
Additional information	The payload for this statement 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 expression 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 such as an object or array. A `null` value removes the attribute from the body. Key-value pairs that do not already exist are added to the body, unless the value is `null`. Format: `{ "$.jsonPath": "attributeValue" }`

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 statement directly to a SCIM search.

Copyright © 2025 Ping Identity Corporation

Description	Details
Additional information	The payload for this statement 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

Copyright © 2025 Ping Identity Corporation

Description	Details

Additional information	The payload for this statement 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`
	The query parameter is removed from the request.
	String
	The parameter is set to that specific value.
	Array of strings
	The 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, consider the following request made to a proxied API:
	`curl --location --request GET '{{apiPath}}/directory/v1/{{dn}}/subtree?searchScope=wholeSubtree&limit=1000' \ --header 'Authorization: Bearer {{jwtToken}}'`
	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 request to be rewritten as follows:
	`curl --location --request GET '{{apiPath}}/directory/v1/{{dn}}/subtree?searchScope=wholeSubtree&limit=20' \ --header 'Authorization: Bearer {{jwtToken}}'`

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

Copyright © 2025 Ping Identity Corporation

Additional information	The payload for this statement 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 statement 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 statement payload is as follows.
	`{"name.first": "John", "name.last": "Doe"}`
	Then the resulting request to the store adapter looks like this.

Copyright © 2025 Ping Identity Corporation

Description	Details

	<pre>{
	  "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"}
	  ]
	}</pre>

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.

Description	Details
Applicable to	All, although you cannot apply the Regex Replace Attributes statement directly to a SCIM search.
Additional information	The payload for this statement is either a JSON object or an array of JSON objects. Each object represents a single replacement operation and has up to four keys. The following list describes these keys: **`"regex"`** Represents the regular expression to use to find the attribute values to replace. This key is required. **`"replace"`** Represents the regex replacement string to use to replace the attribute values with a new value. This key is required. **`"path"`** A JSONPath expression that represents the nodes to start searching under. The expression can point to objects, arrays, or strings in the body. This key is optional. **`"flags"`** A string that contains the regex flags to use. Recognized flags include: • `"i"` Performs case-insensitive matching. • `"l"` Treats the `"regex"` value as a literal string. • `"c"` Performs "canonical equivalence" matching. PingAuthorize replaces any portion of the attribute value that matches the regular expression in the `"regex"` value in accordance with the `"replace"` replacement string. If multiple substrings within the attribute value match the regular expression, 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.

Example

Copyright © 2025 Ping Identity Corporation

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 statement 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'."
   }
}
```

Custom statements

In addition to the statement types that are available out of the box in the PingAuthorize Policy Editor, policy writers can use a custom statement that leverages the PingAuthorize Server SDK.

For information about the implementation and configuration of such statements, see the PingAuthorize Server Administration Guide.

After configuring the statement properly, you can use it in a policy by selecting **Custom Statement** from the **Create new Statement** drop-down list.

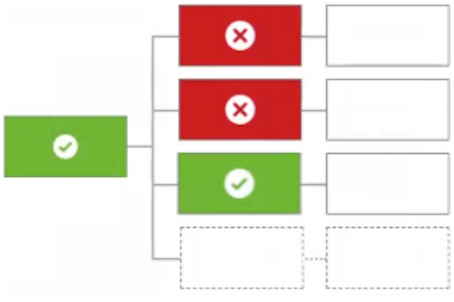
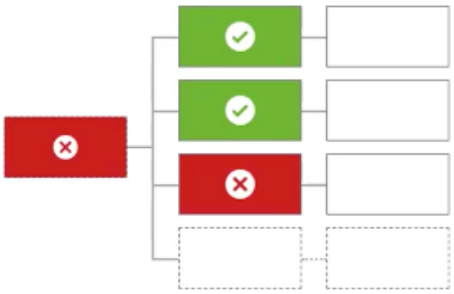
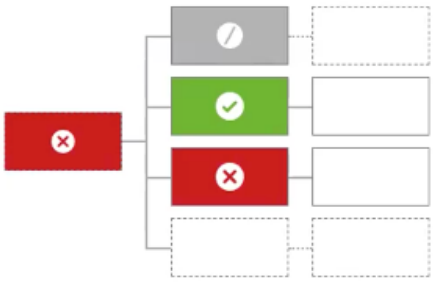
Combining algorithms

Policies can combine multiple rules to produce a `Permit`, `Deny`, `Indeterminate`, or `Not Applicable` decision.

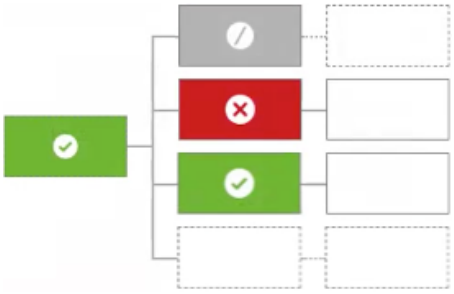
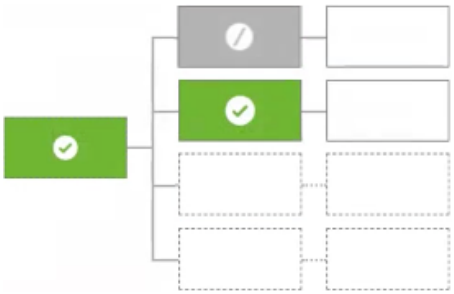
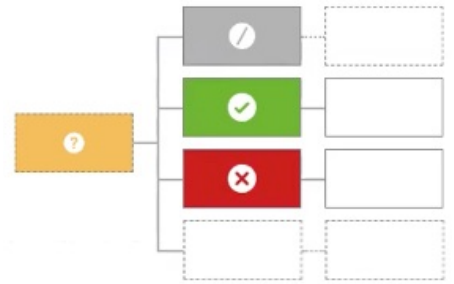
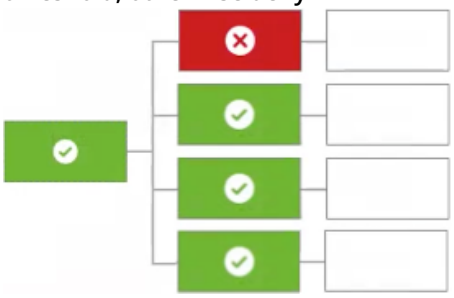
To evaluate the overall decision of a policy, the policy decision point (PDP) applies a combining algorithm. The default algorithm that is set on a new policy is **The first applicable will be the final decision**. If the evaluate all option isn't selected, this algorithm stops evaluating as soon as it reaches a decision that is not `Not Applicable`.

Copyright © 2025 Ping Identity Corporation

The following table identifies available combining algorithms and describes their effects. The diagrams show one example of a decision evaluation for each combining algorithm; other evaluation paths are possible. The diagram legend is displayed before the table. The first column in each diagram represents the overall decision returned by the policy. The second column represents child decisions that produce the resulting policy decision.

Combining algorithm	Details
A single permit will override 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. 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`.
A single deny will override 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. The policy returns `Indeterminate` if a child produces `Indeterminate`. Otherwise, the policy returns `Permit` if a child produces `Permit`. If none of the previous situations occur, the policy returns `Not Applicable`.
Unless one decision is deny, the decision will be 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.

Copyright © 2025 Ping Identity Corporation

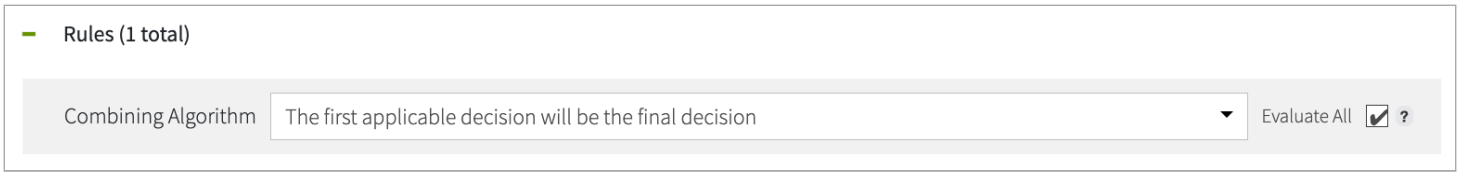
Combining algorithm	Details
Unless one decision is permit, the decision will be deny 	The policy defaults to `Deny` unless any of its children produce the decision `Permit`. The evaluation of rules stops as soon as a `Permit` is produced.
The first applicable decision will be the final decision 	Evaluates 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`.
Only one child may produce a decision. If more than one is produced, the result will be indeterminate 	Evaluates the children in turn. If at any point two children produce a decision other than `Not Applicable`, evaluation stops, and the policy returns `Indeterminate`. If at any point a single child produces an `Indeterminate` decision, evaluation stops, and the policy returns `Indeterminate`. If exactly one child produces an applicable decision, the policy uses it. If the evaluation produces no applicable decisions, the policy returns `Not Applicable`.
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`.

Copyright © 2025 Ping Identity Corporation

Evaluate all

By default, combining algorithms stop evaluating a policy's child elements as soon as a final decision is reached. For example, when using the **The first applicable decision will be the final decision** combining algorithm, evaluation ends immediately if any child returns a `Permit` or `Deny`.

To override this behavior and ensure that all child policies or rules are evaluated, select the **Evaluate All** checkbox next to the **Combining Algorithm** list.

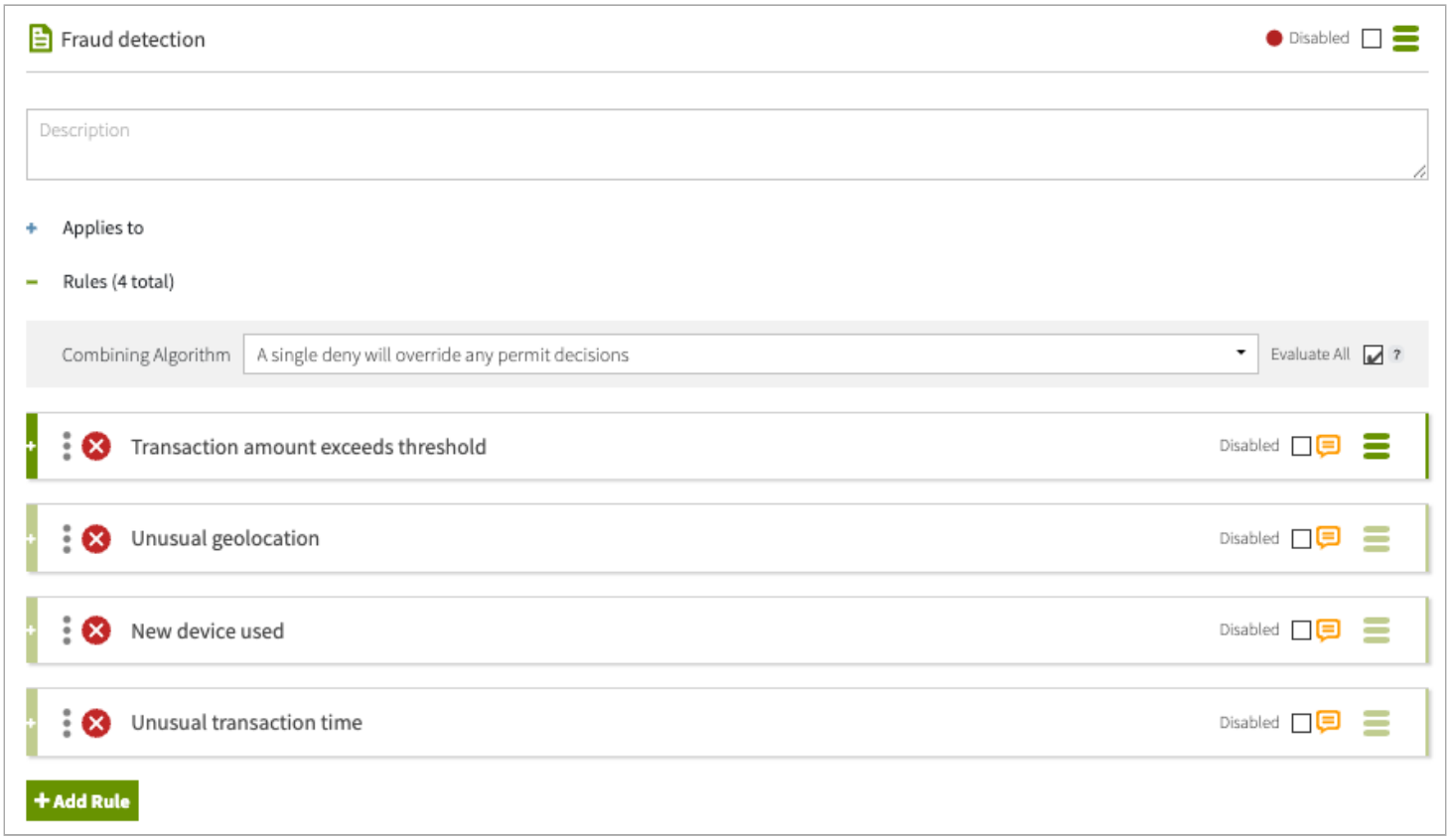


Selecting **Evaluate All** ensures that every child element is evaluated, even if a final decision has already been reached. Although this behavior doesn't affect the final decision, it allows statement information, such as the reason for transaction denial, to propagate up to the parent policy. This information is useful for auditing, debugging, and understanding policy outcomes.

> **Note**
>
> Even with **Evaluate All** selected, targets still determine whether a policy or rule is evaluated for a given request.

Example: Detecting fraud with full rule visibility

Consider a bank that approves or denies transactions based on multiple risk signals. These signals are modeled as child rules under a **Fraud detection** policy, which uses the **A single deny will override any permit decisions** combining algorithm.



Copyright © 2025 Ping Identity Corporation

The policy includes the following rules:

- **Transaction amount exceeds threshold**: Denies if the transaction amount exceeds 10,000 USD.
- **Unusual geolocation**: Denies if the transaction originates from an IP or country not associated with the account.
- **New device used**: Denies if the transaction comes from a device not associated with the account.
- **Unusual transaction time**: Denies if the transaction occurs at an unusual time.

Each rule includes a `denied-reason` statement explaining why the rule denied the transaction.

With **Evaluate All** selected, the policy evaluates every rule, regardless of whether an earlier rule has returned a `Deny`. This provides analysts full visibility into every risk signal, helping detect fraud patterns, meet audit requirements, and enforce internal risk policies more effectively.

Without **Evaluate All** selected, the combining algorithm stops evaluating when the first `Deny` occurs, skipping evaluation of other potentially relevant rules. For example, if the transaction amount exceeds the threshold, the policy doesn't evaluate the geolocation or device rules, and their corresponding denial reasons aren't included in the final decision.

Rule structure

Policy rules power the fine-grained access control capability of 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.

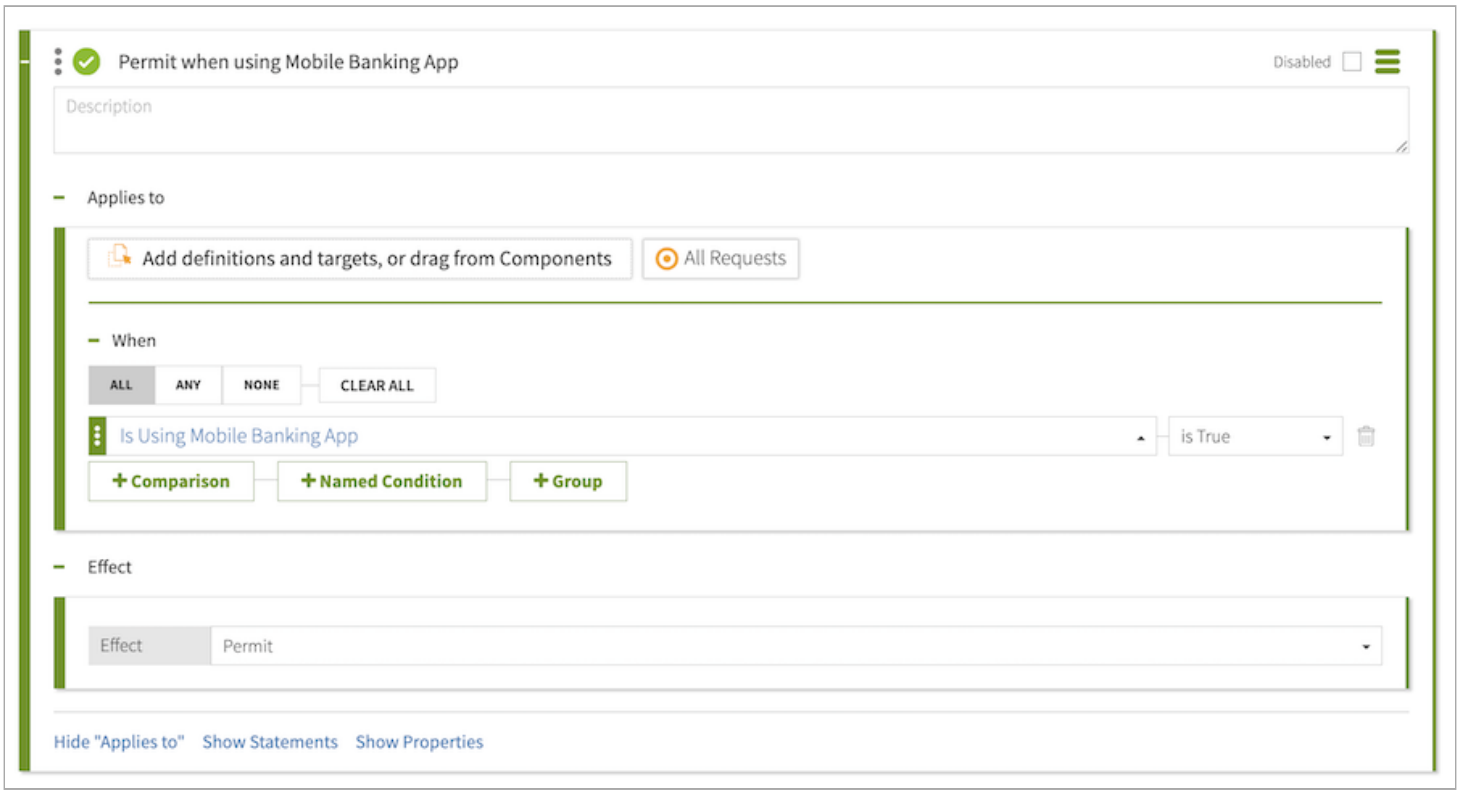
The following example includes an **Applies to** condition and a **Permit** effect:

- If the condition evaluates to `true`, the rule permits.

Copyright © 2025 Ping Identity Corporation

- 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

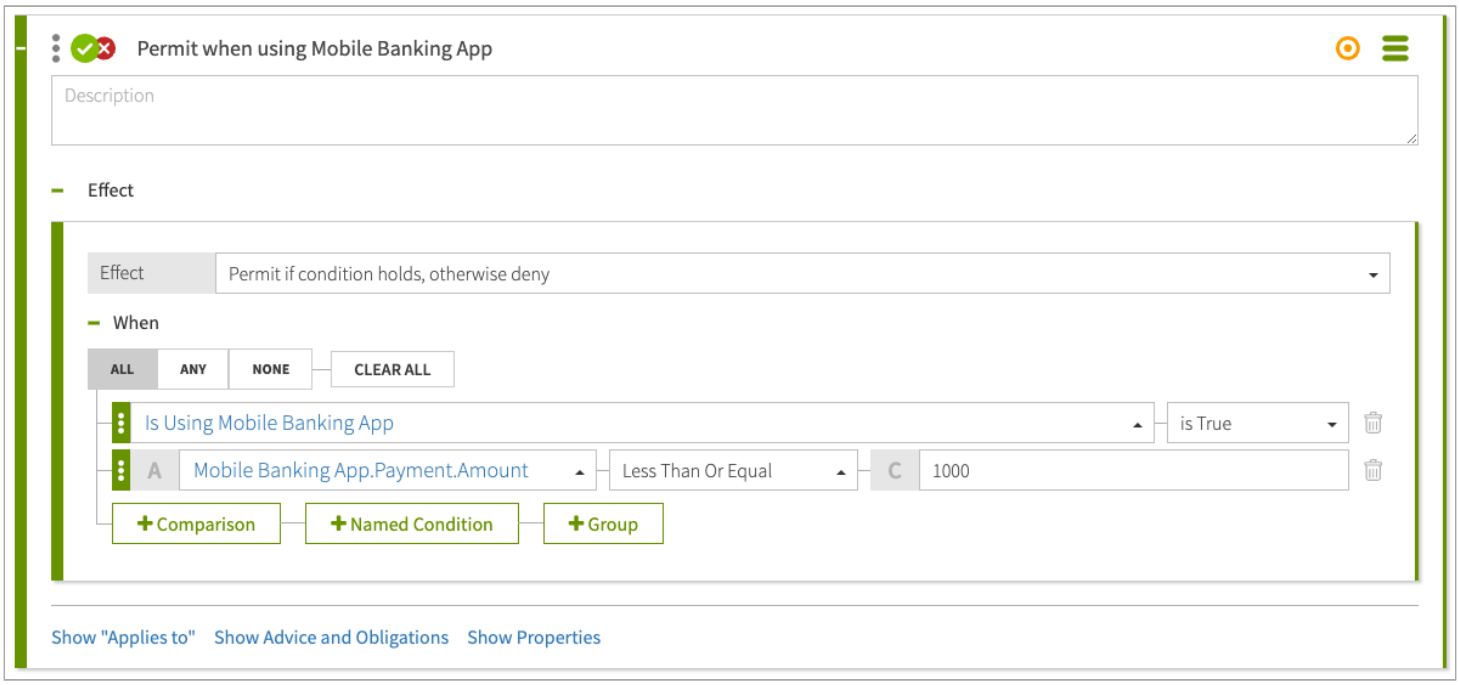
- When a logical condition involves comparing two attributes, try to ensure the attributes have the same data type. Comparing different data types requires an implicit conversion that might not always yield the intended result.
- Just as with Trust Framework entities, you can check which entities depend 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.

Copyright © 2025 Ping Identity Corporation

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.

Testing policies

Test PingAuthorize policies from end-to-end with visualization tools that show the complete decision flow.

Steps

1. Go to **Policies**.
2. Select the policy you want to test and then click the **Test** tab.

Copyright © 2025 Ping Identity Corporation

3. Define the test scenario:

    1. In the **Domain** list, select any domains you want to include as request parameters.
    2. In the **Service** list, select any services you want to include as request parameters.
    3. In the **Identity Provider** list, select any identity providers you want to include as request parameters.
    4. In the **Action** list, select any actions you want to include as request parameters.
    5. In the **Attributes** list, select any attributes you want to include as request parameters and provide sample values.
    6. In the **Overrides** section, configure attribute and service values to override those elements' default behavior.

       For example, if an attribute is defined with a request parameter resolver and no value is specified in the test request, the decision service resolves that attribute from the **Overrides** configuration.

       > **Note**
       >
       > You can override any attribute's value, regardless of its resolution or processing details.

Example:

Using the **Users starting a new game** policy from the tutorials as an example, the following testing scenario uses the **HttpRequest.AccessToken** attribute in a request to test whether the policy returns a deny decision when the token's subject claim ends with `@example.com`.

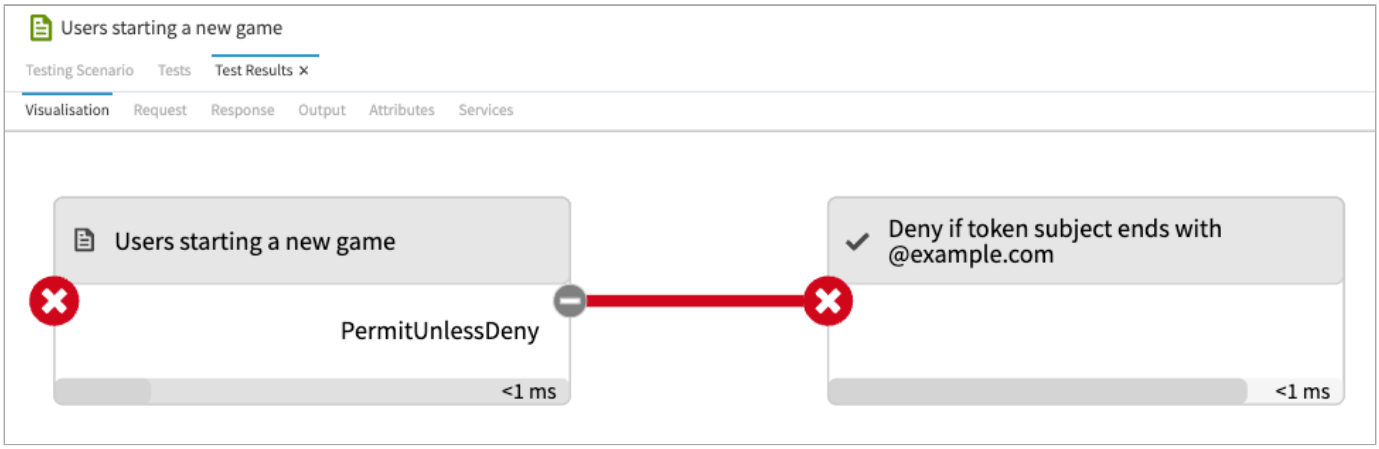
4. Click **Execute**.

> **Note**
>
> If the test scenario exceeds 50 kB in size, sample values will not be stored in the local cache after executing the test request.

Result:

The **Visualization** tab shows test results. As expected, the request is denied.

Copyright © 2025 Ping Identity Corporation



5. Examine the decision flow to make sure decisions are evaluated according to your expectations.

   You can click any box in the flow to show more details.

6. Click the other tabs for additional details.

   - **Request** tab: Shows the JSON request sent to the decision engine, allowing you to confirm that the expected information was sent.
   - **Response** tab: Shows the complete, high-verbosity response for the decision.

     > **Note**
     >
     > If the same comparison condition is attached to more than one rule in the policy subtree, the decision response only includes the evaluation of the first occurrence of this condition. Although the condition is only included once in the response, the decision engine evaluates this condition wherever it is needed to make a decision.
     > If the parent policy of the first instance of this condition is not applicable to the request, the decision response does not include evaluation of any rule containing this condition. This behavior is the same regardless of the rule's outcome (`Permit` , `Deny` , `Not Applicable`).

   - **Output** tab: Shows details about the decision, including the time it took to evaluate policies and rules.
   - **Attributes** tab: Shows details about the attributes used in the decision.
   - **Services** tab: Shows details about the services used in the decision.

7. To repeat the test using a different scenario, click the **Testing Scenario** tab, change the parameters and then click **Execute**.

Visualizing a policy decision response

When you develop and test policies in PingAuthorize, examine the decision flow and other details about recent decisions to make sure the decision service is evaluating policies according to your expectations.

Steps

1. In the Policy Editor, go to **Policies** > **Decision Visualizer**.
2. Select a decision to visualize.

Copyright © 2025 Ping Identity Corporation

Choose from:

- Select a recent decision.

  1. In the **Decision Visualizer**, click the **Recent Decisions** tab.
  2. Select a recent decision from the list.

  > **Note**
  >
  > You can control the number of decisions that appear in the **Recent Decisions** list. Learn more in Setting the request list length for Decision Visualizer.
  > To visualize self-governance decisions, sign on as a self-governance administrator and click **Self Governance** instead of **Recent Decisions**.

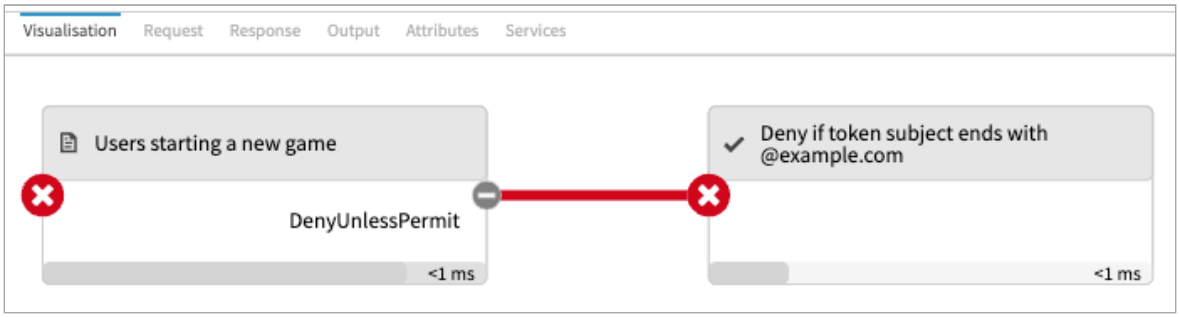
- Copy and paste a decision response from the policy decision log.

  > **Important**
  >
  > To visualize a policy decision log entry, you must add either the decision-tree or the evaluation-log view to the Decision Response View. Learn more in Configuring the Decision Response View.

  1. In the `<PingAuthorize>/logs/policy-decision.log` file, copy the decision response JSON object.
  2. In the **Decision Visualizer**, click the **Paste Logs** tab.
  3. Paste the decision response JSON object.
  4. Click **Visualise**.

     Result

     An interactive decision tree of your policies is displayed.

3. Examine the decision flow to make sure decisions are evaluated according to your expectations.

   You can click any box in the flow to show more details.

4. Click the other tabs for additional details.

   - **Request** tab: Shows the JSON request sent to the decision service, allowing you to confirm that the expected information was sent.
   - **Response** tab: Shows the complete, high-verbosity response for the decision.

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> If the same comparison condition is attached to more than one rule in the policy subtree, the decision response includes only the evaluation for the first instance of this comparison. Although the condition is included only once in the response, the decision service evaluates the condition wherever it is needed to make a decision.
> If the parent policy of the first instance of this condition isn't applicable to the request, the decision response doesn't include evaluation of any rule containing this condition. This behavior is the same regardless of the rule's outcome (`Permit` , `Deny` , `Not Applicable`).

- **Output** tab: Shows details about the decision, including the time it took to evaluate policies and rules.
- **Attributes** tab: Shows details about the attributes used in the decision.
- **Services** tab: Shows details about the services used in the decision.

Policy queries

Policy queries enable you to dynamically determine user entitlements by posing open-ended authorization questions.

Why use policy queries?

Authorization can be represented as a relationship between a subject, action, and a resource. For example, "Is John (subject) authorized to transfer (action) money from this savings account (resource)?"

Instead of executing a batch request to obtain a list of accounts or actions a user is authorized to access or perform, policy queries enable you to construct a single decision request that checks which combinations of subject, action, and resource produce a `PERMIT` decision in a specified context. This enables your applications to more efficiently retrieve and enforce decisions for real-time events, such as generating dynamic web content.

A typical decision request forwarded to the JSON PDP API in PingAuthorize includes a single value for subject, action, and resource. The Policy Decision Service uses these explicitly provided values, together with your access control policies, to compute a `PERMIT` or `DENY` decision result. For example, the following decision request asks, "Is John Smith (subject) authorized to edit (action) the account (resource)?":

```
{
    "domain": "",
    "service": "",
    "action": "",
    "identityProvider": "",
    "attributes": {
    "Subject": "John Smith",
    "Resource": "account",
    "Action": "edit",
    "RegionInfo": "EU",
    "RequestType": "WEB"
    }
}
```


Copyright © 2025 Ping Identity Corporation

However, instead of limiting each authorization attribute to a single, fixed value, you might want to ask authorization questions such as:

- Which accounts can this customer withdraw funds from?
- Which users can update account preferences?

For example, the following decision request sent to the `governance-engine/query` endpoint asks, "Which users are authorized to edit the account?":

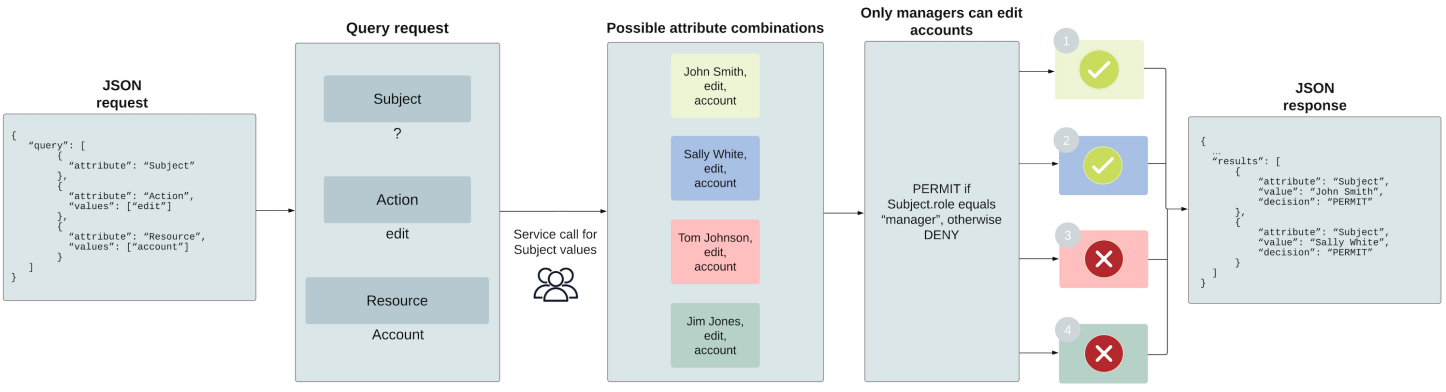
```
{
  "query": [
    {
      "attribute": "Subject"
    },
    {
      "attribute": "Action",
      "values": ["edit"]
    },
    {
      "attribute": "Resource",
      "values": ["account"]
    }
  ]
}
```

The following section walks through the necessary steps to successfully send such a request.

Querying entitlements based on user role

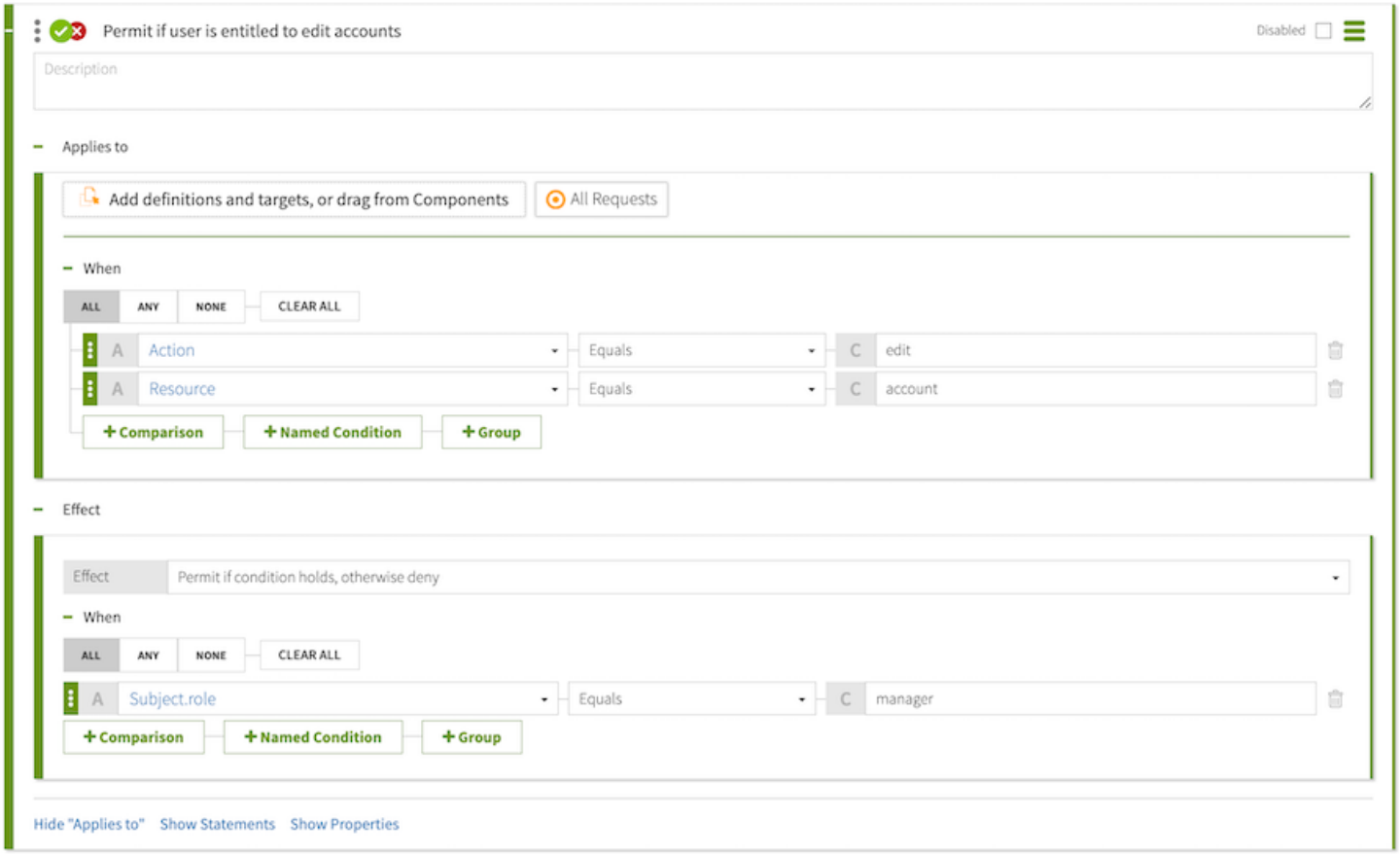
Suppose you want to implement access controls in your banking application around users' interaction with accounts, based on those users' organizational role.

The following diagram illustrates the process of using policy queries to enforce such access controls, from request to response:

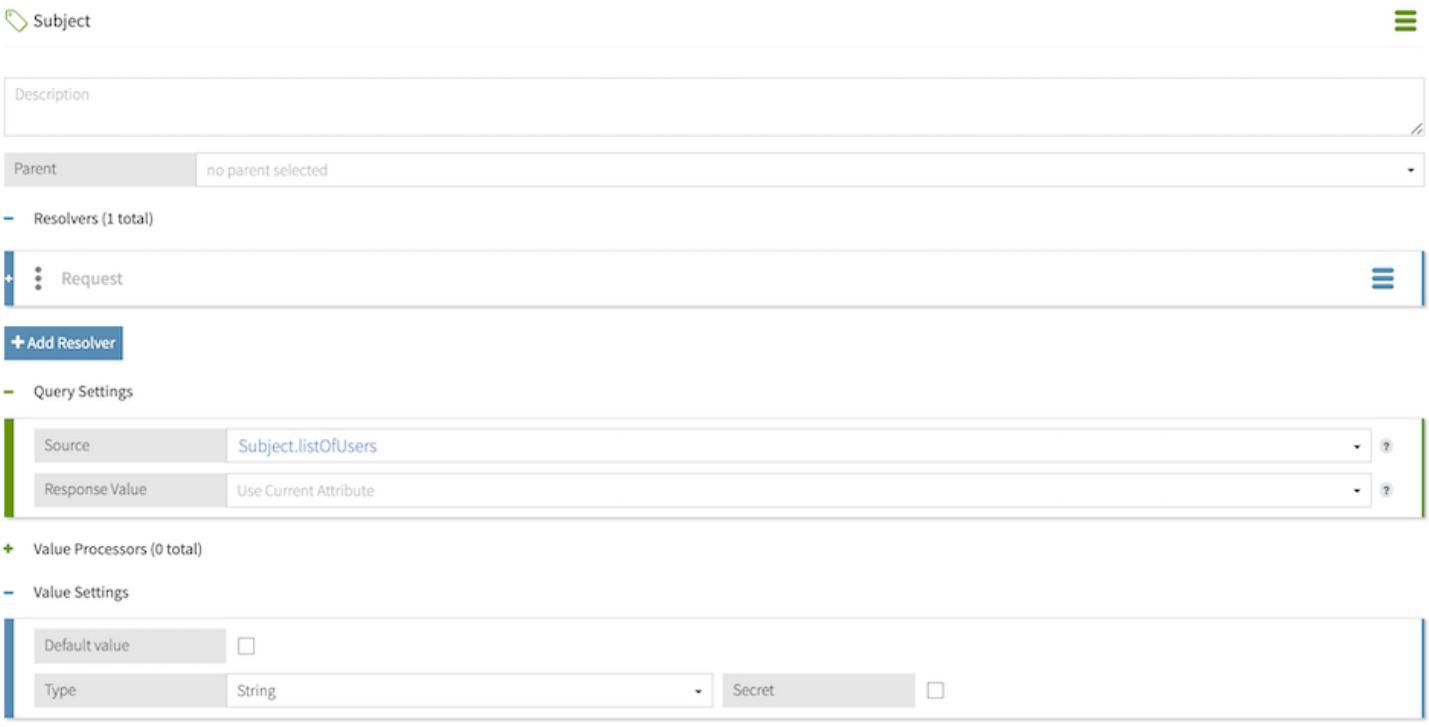


The following rule checks whether the user requesting to edit an account has the role of `manager`. If the user is assigned this role, the decision engine returns a `PERMIT`, and denies otherwise.

Copyright © 2025 Ping Identity Corporation

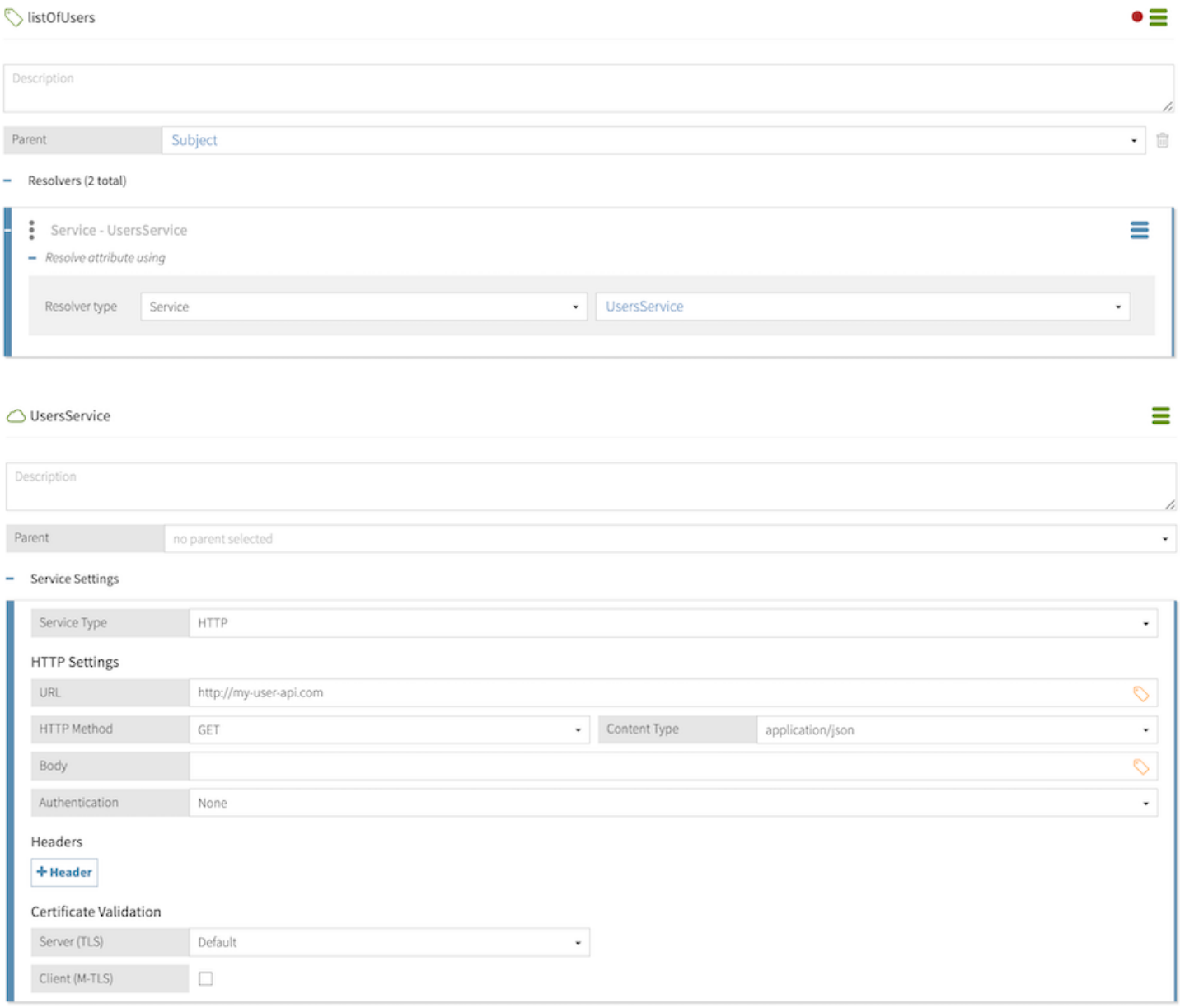


To dynamically populate values for a request attribute with an external service call, rather than providing a static value in the request, enable query settings for that attribute. In this example, to query which users can edit the account, enable query settings for the `Subject` attribute.



Copyright © 2025 Ping Identity Corporation

When enabling query settings for a request attribute, you must specify a `Source` attribute. This attribute provides a collection of possible attribute values for the decision engine to check your policies against. In this example, the `Subject.listOfUsers` attribute resolves from a call to an external information point made by `UsersService` and provides a list of possible `Subject` values.



> **Note**
>
> Learn more about the **Response Value** query setting in Enabling query settings.

Based on the request and policy logic, the JSON PDP API returns the following:

Copyright © 2025 Ping Identity Corporation

```
{
   "requestId": "f1c536bd-a5e0-4b14-80f8-25a71d45774c",
   "timeStamp": "2024-08-16T14:32:25.417092Z",
   "deploymentPackageId": "a2435f66-3669-4d2c-960c-d01dc9c39bfa",
   "elapsedTime": 60,
   "results": [
       {
           "attribute": "Subject",
           "value": "John Smith",
           "results": [
               {
                   "attribute": "Action",
                   "value": "edit",
                   "results": [
                       {
                          "attribute": "Resource",
                          "value": "account",
                          "decision": "PERMIT"
                       }
                    ]
               }
          ]
       },
       {
           "attribute": "Subject",
           "value": "Sally White",
           "results": [
               {
                   "attribute": "Action",
                   "value": "edit",
                   "results": [
                       {
                           "attribute": "Resource",
                           "value": "account",
                           "decision": "PERMIT"
                       }
                   ]
                }
           ]
       }
   ]
}
```


The top-level `results` array contains each subject that produced a `PERMIT` decision, and each of the subjects contains the action and resource that the subject is authorized to perform and access.

Learn more about the structure of query requests and responses in Query requests.

Enabling query settings

To use an attribute without explicitly providing its value in an open-ended decision request, you must enable query settings for that attribute.

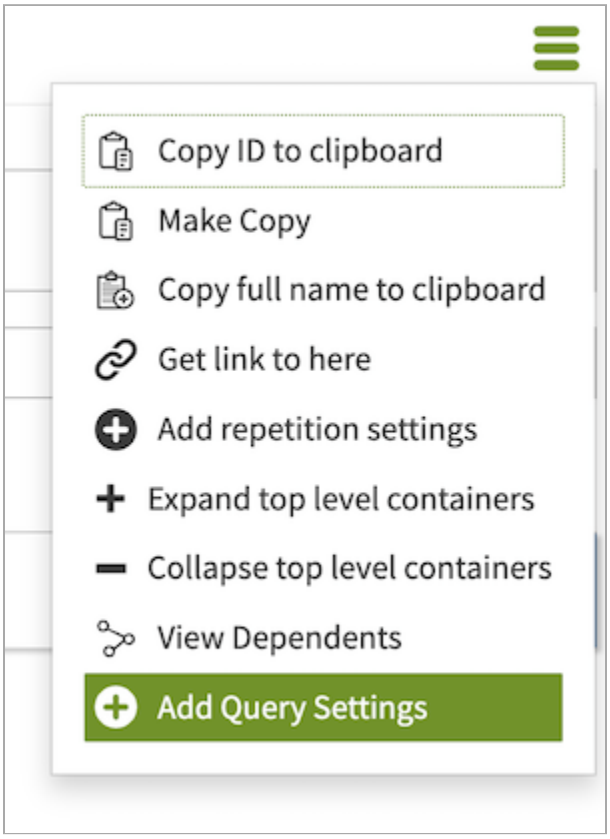
About this task

To enable query settings:

Copyright © 2025 Ping Identity Corporation

Steps

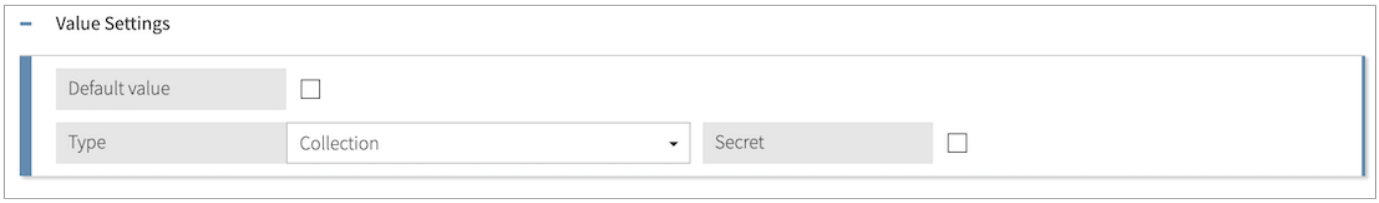
1. In the **Trust Framework**, navigate to the attribute for which you want to enable query settings.
2. In the attribute's hamburger menu, select **Add Query Settings**.



> **Note**
>
> Before enabling query settings, you must save the query attribute.

3. In the **Source** list of the **Query Settings** section that you created, select a source attribute of the **Collection** type.

   This attribute defines the collection of possible query attribute values that the decision engine retrieves to test against the appropriate policy.



> **Caution**
>
> Using a source collection attribute with a significant number of values could strain the system and cause performance issues.

4. **Optional:** In the **Response Value** list, select an attribute with which to filter the decision response.

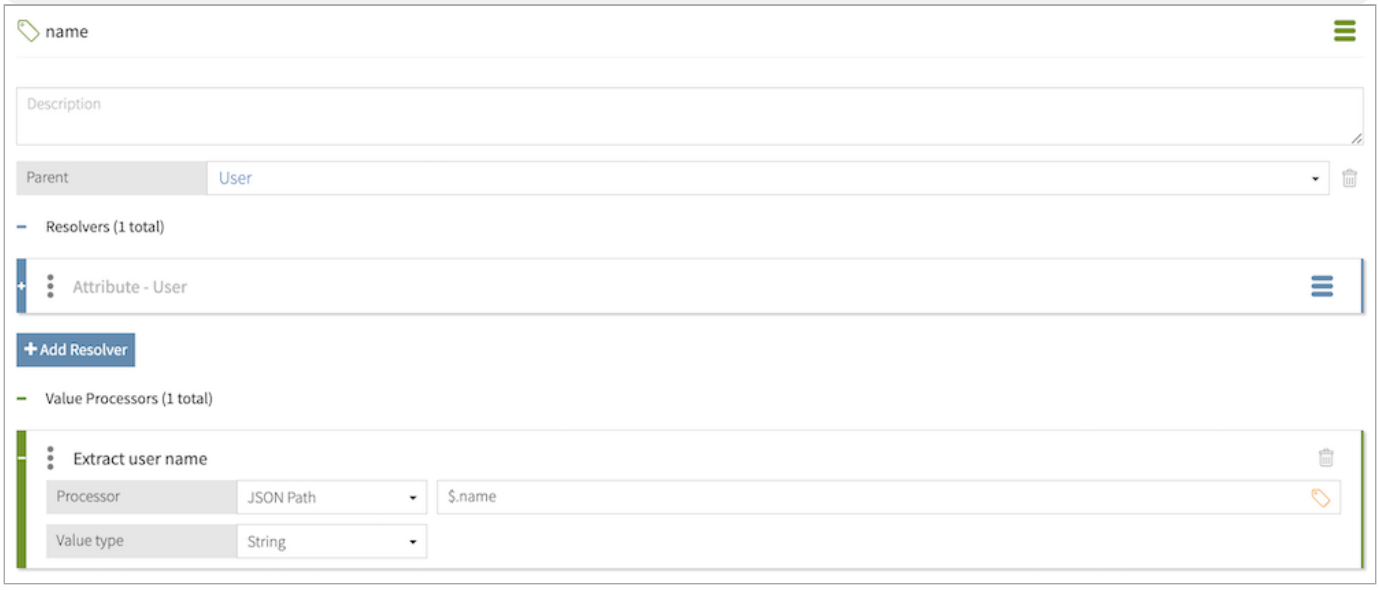
Copyright © 2025 Ping Identity Corporation

> **Note**
>
> You cannot use the same value for **Source** and **Response Value**.

Example:

In this example, **UsersService** is an attribute that resolves from an HTTP service call retrieving a collection of `User` objects. Given a JSON `User` object such as the following, the **User.name** sub-attribute resolves from the **User** attribute and is configured with a JSONPath value processor to extract the `name` field:

```
{
  "name": "Joe Boggs",
  "id": "b5f963fa-111e-49ff-994b-b89a20a2c1d5",
  "age": 27
}
```



With the **User.name** attribute selected as the **Response Value**, requests that query the **User** attribute will only return the name of the relevant user, rather than the entire user object.

```
"results": [
  {
    "attribute": "User",
    "value": "Joe Boggs",
    "decision": "PERMIT"
  }
]
```

To return the entire value of the queried attribute, leave **Response Value** blank.

Copyright © 2025 Ping Identity Corporation

5. Click **Save**.

Result

You can now include this attribute in the `query` array of a query decision request without assigning it any values.

Policy query request and response examples

The following examples highlight different policy query request and response formats supported by the JSON PDP API. Learn more about the structure of such requests and responses in JSON PDP API request and response flow.

Using a query attribute with no values specified

The following request asks, "Which actions can the specified user perform on accounts?":

```
{
    "query": [
        {
            "attribute": "Action"
        },
        {
            "attribute": "Subject",
            "values": ["John Smith"]
        },
        {
            "attribute": "Resource",
            "values": ["account"]
        }
    ]
}
```

The response returns each action that produced a `PERMIT` decision, given the specified subject and resource:

Copyright © 2025 Ping Identity Corporation

```
{
  "requestId": "4da494e4-2f50-4165-b1b3-644981564196",
  "timeStamp": "2024-09-19T21:44:51.905443Z",
  "deploymentPackageId": "ed614a98-f4d0-483a-b9dd-574aa327ad11",
  "elapsedTime": 7,
  "result": [
     {
       "attribute": "Action",
       "value": "edit",
       "results": [
           {
               "attribute": "Subject",
               "value": "John Smith",
               "results": [
                   {
                       "attribute": "Resource",
                       "value": "account",
                       "decision": "PERMIT"
                   }
               ]
           }
       ]
     },
     {
       "attribute": "Action",
       "value": "view",
       "results": [
           {
               "attribute": "Subject",
               "value": "John Smith",
               "results": [
                   {
                       "attribute": "Resource",
                       "value": "account",
                       "decision": "PERMIT"
                   }
               ]
           }
       ]
     }
  ]
}
```


Using a query attribute with multiple values specified

The following request uses a request attribute with multiple values to ask, "Can the specified user edit or view account information?":

Copyright © 2025 Ping Identity Corporation

```
{
    "query": [
        {
            "attribute": "Action",
            "values": ["edit", "view"]
        },
        {
            "attribute": "Subject",
            "values": ["Tom Johnson"]
        },
        {
            "attribute": "Resource",
            "values": ["account"]
        }
    ]
}
```

The response returns each action that produced a `PERMIT` decision, given the specified subject and resource:

Copyright © 2025 Ping Identity Corporation

```
{
   "requestId": "af52d214-6dbb-4699-9fe1-74ec88ccebac",
   "timeStamp": "2024-09-20T01:40:04.381703Z",
   "deploymentPackageId": "292863fe-2cde-440f-9c7b-9aee4a8dc94e",
   "elapsedTime": 4,
   "results": [
      {
         "attribute": "Action",
         "value": "edit",
         "results": [
             {
                "attribute": "Subject",
                "value": "John Smith",
                "results": [
                    {
                       "attribute": "Resource",
                       "value": "account",
                       "decision": "PERMIT"
                    }
               ]
             }
         ]
      },
      {
        "attribute": "Action",
        "value": "view",
        "results": [
            {
               "attribute": "Subject",
               "value": "John Smith",
               "results": [
                   {
                     "attribute": "Resource",
                     "value": "account",
                     "decision": "PERMIT"
                   }
               ]
            }
        ]
      }
   ]
}
```


Using a query attribute with no values specified and a query attribute with multiple values specified

The following request asks, "Which users can either edit or view bank accounts?":

Copyright © 2025 Ping Identity Corporation

```
{
    "query": [
        {
            "attribute": "Subject"
        },
        {
            "attribute": "Action",
            "values": ["edit", "view"]
        },
        {
            "attribute": "Resource",
            "values": ["account"]
        }
    ]
}
```

The response returns each user that produced a `PERMIT` decision on either of the specified actions, given the specified resource:

Copyright © 2025 Ping Identity Corporation

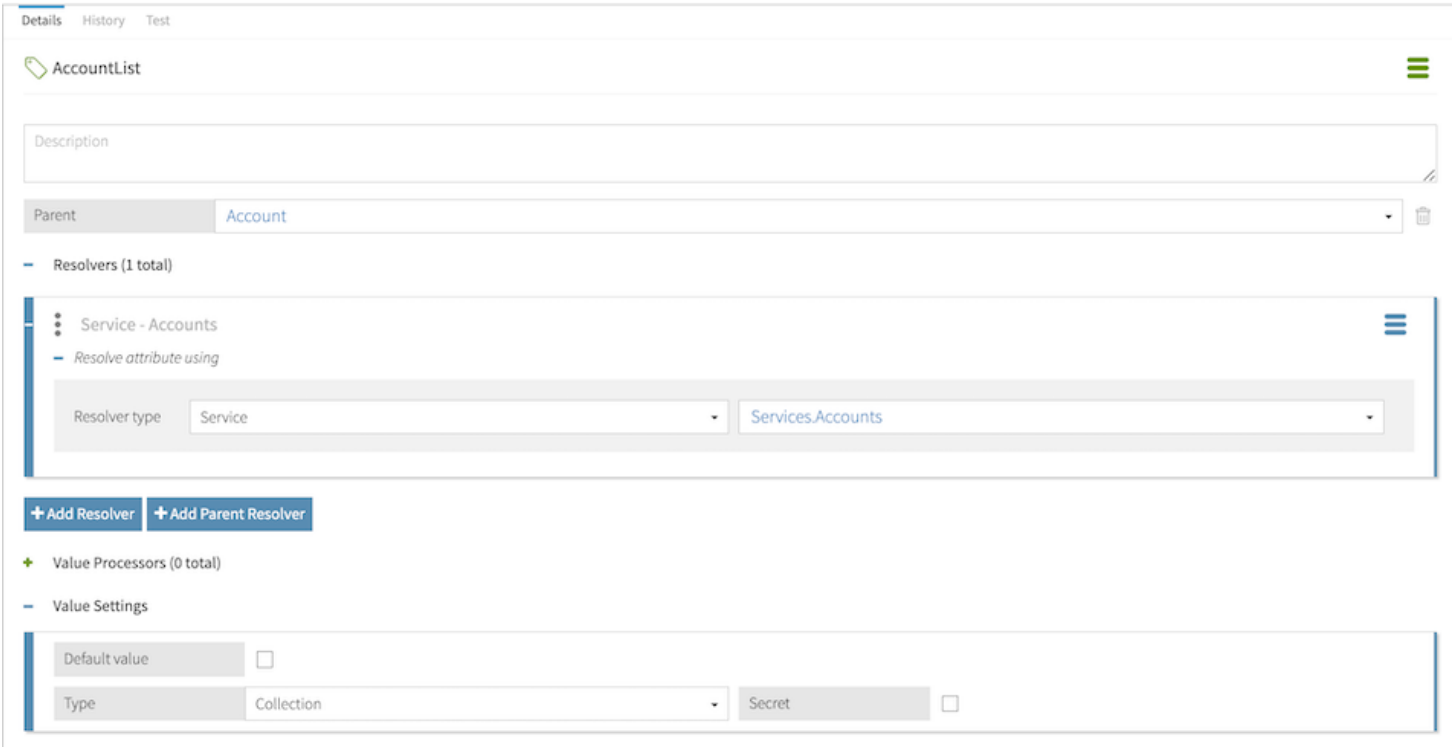
```
{
   "requestId": "2d3fe162-7490-43a4-abdf-56c978a35abf",
   "timeStamp": "2024-09-20T01:53:06.102542Z",
   "deploymentPackageId": "292863fe-2cde-440f-9c7b-9aee4a8dc94e",
   "elapsedTime": 4,
   "results": [
      {
         "attribute": "Subject",
         "value": "John Smith",
         "results": [
             {
                "attribute": "Action",
                "value": "edit",
                "results": [
                    {
                       "attribute": "Resource",
                       "value": "account",
                       "decision": "PERMIT"
                    }
                ]
             },
             {
                "attribute": "Action",
                "value": "view",
                "results": [
                    {
                       "attribute": "Resource",
                       "value": "account",
                       "decision": "PERMIT"
                    }
                ]
             }
         ]
      },
      {
        "attribute": "Subject",
        "value": "Sally White",
        "results": [
            {
               "attribute": "Action",
               "value": "edit",
               "results": [
                   {
                     "attribute": "Resource",
                     "value": "account",
                     "decision": "PERMIT"
                   }
               ]
            }
        ]
      }
   ]
}
```


Using query attributes to resolve other query attributes

When building the Trust Framework around your policy query use case, you can use resolvers to create chains of dependence between query attributes.

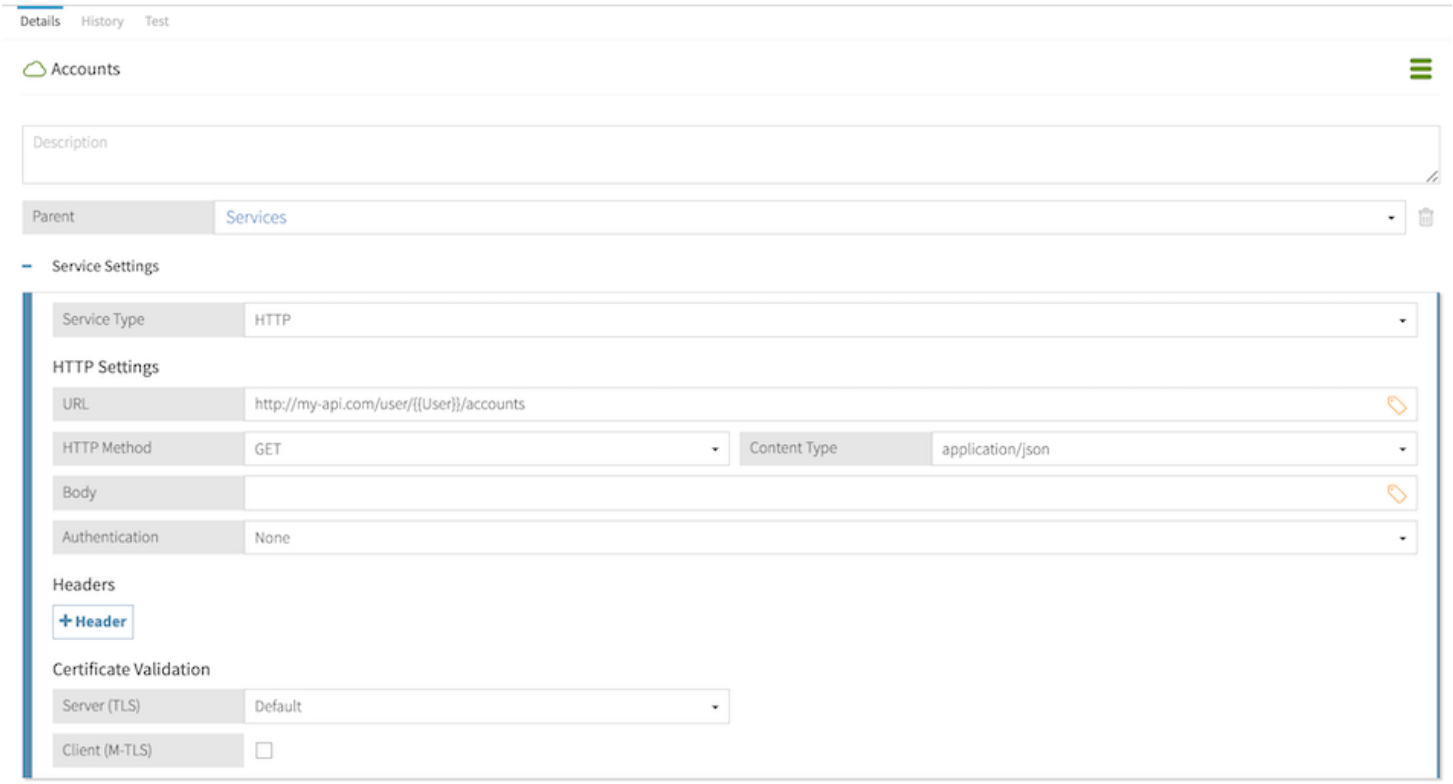
Copyright © 2025 Ping Identity Corporation

In this example, the `Account` attribute is configured with query settings to enable authorization questions, such as "Which accounts can this user update? The `AccountList` attribute is configured as the `Account` attribute's source collection:



The `AccountList` attribute uses the `Accounts` service as its resolver, and this service interpolates the `User` attribute in its endpoint URL definition:

Copyright © 2025 Ping Identity Corporation



The following query requests asks, "Which accounts can John Smith edit?":

```
{
    "query": [
        {
            "attribute": "Account"
        },
        {
            "attribute": "Subject",
            "values": ["John Smith"]
        },
        {
            "attribute": "Action",
            "values": ["edit"]
        }
    ]
}
```

In resolving the `Account` attribute, the `Accounts` service uses the single-valued `User` attribute included in the request to make an HTTP call and retrieve a list of accounts. The response then returns an array of accounts that produced a `PERMIT` decision, given the specified user and action.

> **Note**
>
> Using multivalued or unbounded query attributes to resolve other query attributes is not currently supported.

Copyright © 2025 Ping Identity Corporation

Troubleshooting policy queries

The following resources can help you solve issues with policy query requests and responses.

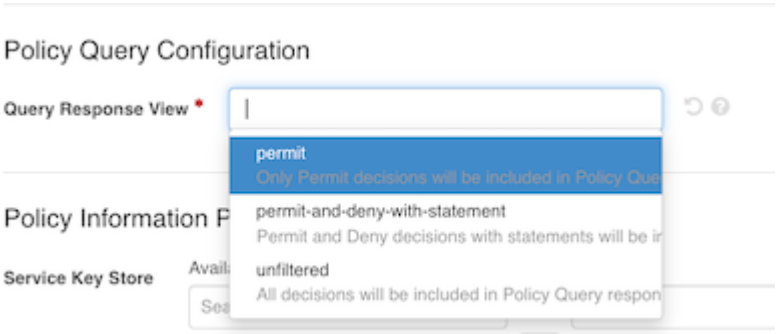
Configuring query response granularity

A query response includes a set of decisions. Each decision is executed against a possible combination of request attributes. By default, query responses return two types of decisions:

- `PERMIT`
- `DENY` with statements attached

Adjust the granularity of query responses by configuring the query response view:

1. In the administrative console, go to **Authorization and Policies** and click **Policy Decision Service**.
2. In the **Policy Query Configuration** section, select the response granularity in the **Query Response View** list.



The following table describes the behavior of each available query response view:

Response view	Description
`permit`	Only includes `PERMIT` decisions in the query response.
`permit-and-deny-with-statement`	Includes `PERMIT` decisions and `DENY` decisions with statements attached in the query response.
`unfiltered`	Includes all decisions in the query response, regardless of the outcome or the presence of statements. You can use this view can help diagnose unexpected decision outcomes for specific combinations of attributes.

Alternatively, you can add the `x-respond-with` header to your query requests and assign it one of the following values:

- `permit`
- `permitAndDenyWithStatement`
- `unfiltered`

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> The configured query response view will apply to responses in both embedded and external policy decision point (PDP) mode.

Visualizing policy query decisions

As part of the policy development and debugging process, you can examine recent decisions returned in a query response.

About this task

When you develop and test policies, you can examine the decision flow and other details about recent decisions to make sure the decision engine is evaluating policies according to your expectations.

Steps

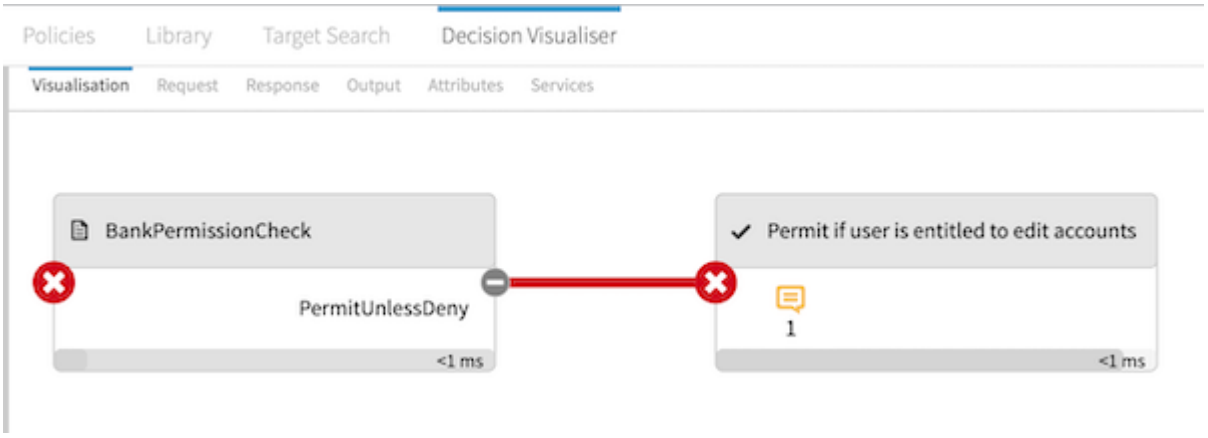
1. In the `policy-query.log` file, copy the `response` field for the individual query permutation for which you want to visualize the corresponding decision.


```
"response":{"id":"38be8a7e-b552-4fd8-9311-4b742dc71280","deploymentPackageId":"38288f2d-f7ca-4c1c-
ada4-0d6ef878de38","timestamp":"2024-09-09T14:44:19.687068Z","elapsedTime":42514,"request":
{"domain":"","service":"","identityProvider":"","action":"","attributes":{"User":"{\"userID\":
789,\"role\":\"financial advisor\",\"name\":\"Jim
Jones\"}","Action":"view","BankAccount.resource":"balance"}},"externallyResolvedAttributes":
{},"decision":"DENY","authorised":false,"statements":[{"id":"1290c587-bfdc-434b-
b181-43f542207235","name":"Denial reason","code":"denied-reason","payload":"Example: { \"status\":
403, \"message\": \"error_code\", \"detail\":\"Role does not equal teller or customer service
rep\"}","obligatory":false,"fulfilled":false,"attributes":{}}],"status":{"code":"OKAY","messages":
[],"errors":[]},"attributes":{...},"services":{...},"decisionTree":{...},"evaluationLog":[{...},
{...}]}
```


2. In the Policy Editor, go to **Policies > Decision Visualiser**.
3. In the Paste Logs field, paste the log data you copied in step 1.

Copyright © 2025 Ping Identity Corporation

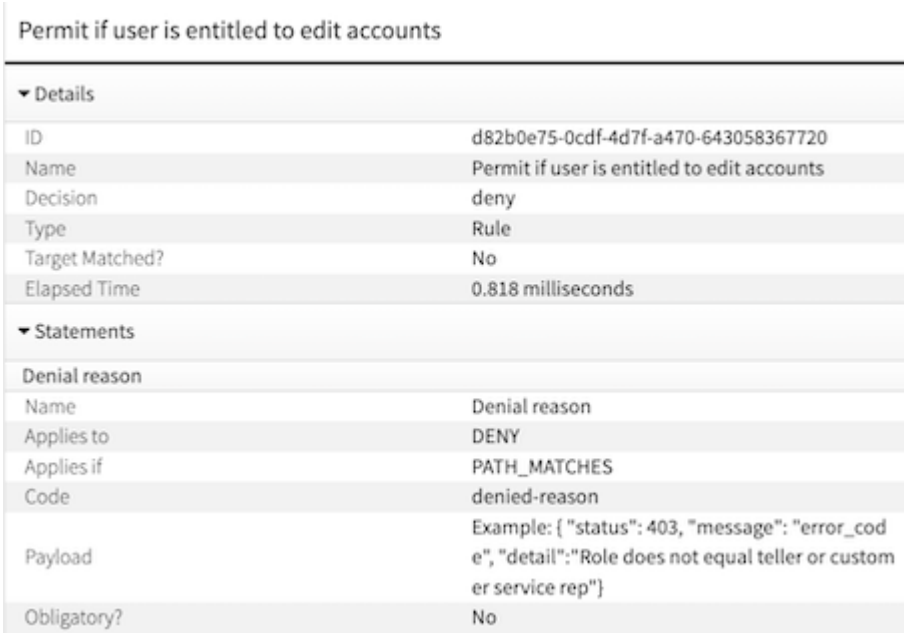
4. Click Visualise.



On the Visualisation tab, examine the decision flow to make sure decisions are evaluated according to your expectations.

5. Click a box in the flow to show more details.

Copyright © 2025 Ping Identity Corporation



Permit if user is entitled to edit accounts

Details	
ID	d82b0e75-0cdf-4d7f-a470-643058367720
Name	Permit if user is entitled to edit accounts
Decision	deny
Type	Rule
Target Matched?	No
Elapsed Time	0.818 milliseconds
Statements	
Denial reason	
Name	Denial reason
Applies to	DENY
Applies if	PATH_MATCHES
Code	denied-reason
Payload	Example: { "status": 403, "message": "error_code", "detail":"Role does not equal teller or customer service rep"}
Obligatory?	No

This example represents a rule that denies requests to view account balances if the user's role is not `teller` or `customer service rep` and permits otherwise. The decision evaluated as `deny` and took 0.818 milliseconds. Additionally, there is a `denied-reason` statement attached to the decision with additional context for the denial.

6. Click the other tabs for additional details:
    - Request tab: Shows the JSON request sent to the decision engine, allowing you to confirm that the expected information was sent.
    - Response tab: Shows the response for the individual permutation's decision.

      This tab includes detailed information about the attributes and policy elements used to produce the decision response.
    - Output tab: Shows the time taken to evaluate each policy set, policy, and rule used in the final decision.
    - Attributes tab: Shows resolution, processing, and policy dependency details for each attribute used in the decision.
    - Services tab: Shows details about the services used in the decision.

Policy query logging

The PingAuthorize Server and Policy Editor provide a set of configurable audit logs for debugging policy query requests. Learn more in Policy query logging.

Policy query logging

PingAuthorize writes policy query information to the following logs:

`policy-query.log`

Similar to the Policy Decision logger, this file-based log publisher records query responses in embedded PDP mode.

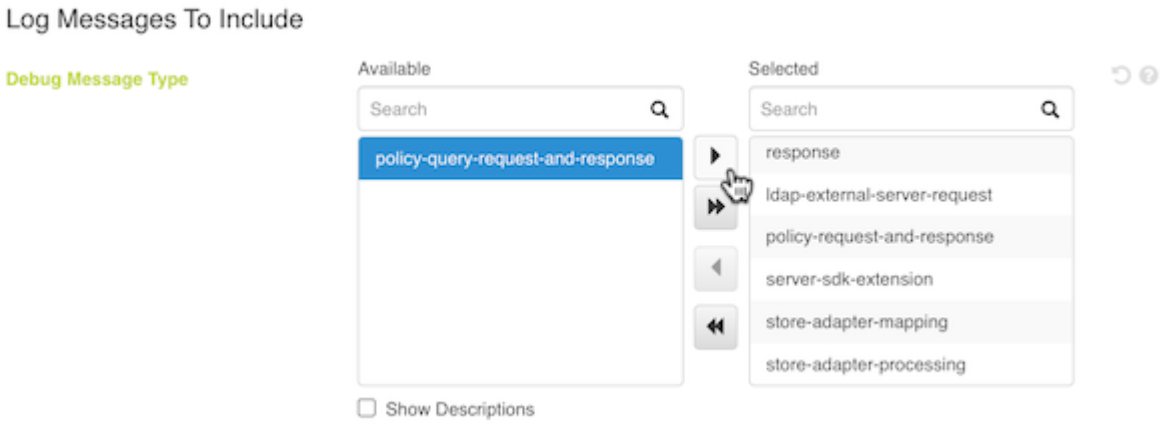
Copyright © 2025 Ping Identity Corporation

`debug-trace.log`

Records detailed information about the processing of HTTP requests and responses in embedded PDP mode.

To include query requests and responses in `debug-trace.log`:

1. In the administrative console, go to **Logging, Monitoring, and Notifications** and click **Log Publishers**.
2. Click **Debug Trace Logger**.
3. In the **Log Messages to Include** section, on the **Debug Message Type** row, select **policy-query-request-and-response** and click the arrow button.



`query-audit.log`

Records query responses in external PDP mode.

Enabling debug logging in embedded PDP mode

Enable debug logging to provide detailed information when query requests produce errors or unexpected results. In addition to logging the full query request and response, debug logging records details about the resolution and policy dependencies of each attribute included in the query request. This level of detail can be necessary to troubleshoot the individual decision requests that make up a single query request.

> **Warning**
>
> Debug logging could log sensitive and personally identifiable information (PII). Enable debug logging only during troubleshooting and disable it afterward.

In debug mode, a policy query audit log entry includes the following fields:

- `requestId`: A unique identifier for the query request
- `permutationId`: A unique identifier for the query permutation

  A query permutation is a combination of query attributes used for a decision in the final query response. Use this identifier and the `requestId` for increased visibility of query request information across your logging system. For example, a logged call to an external information point would include identifiers for the request and permutation that invoked that service.

Copyright © 2025 Ping Identity Corporation

- `permutation`: A query permutation as an array of JSON objects containing each query attribute and its value
- `response`: The complete, high-verbosity response for a query permutation's associated decision, including expanded errors and other helpful information

  By default, this field includes details about the resolution and policy dependencies of each attribute involved in the permutation's corresponding decision, along with details about any external service used in that decision.

> **Note**
>
> You can increase the `response` field's level of detail by controlling the query permutation view. Learn more in Configuring the query permutation view.

The following is an example of a policy query request body and its associated audit log messages. The `response` field is shortened for the sake of brevity.

```
{
    "query": [
        {
            "attribute": "Account"
        },
        {
            "attribute": "Action",
            "values": ["read"]
        },
        {
            "attribute": "User",
            "values": ["Bob"]
        }
    ]
}
```

```
[06/Jun/2025:15:28:08 +0100] QUERY requestID=103 correlationID="d08fe710-7394-41f9-9a80-26009f2bbd6e"
queryId=072a9fbf-528a-4f71-9045-ad541e7c2d73 {"requestId":"072a9fbf-528a-4f71-9045-
ad541e7c2d73","permutationId":"89f37761-8b9f-4b2a-91e0-392e64da84cd","permutation":
[{"attribute":"User","value":"Bob"},{"attribute":"Action","value":"read"},
{"attribute":"Account","value":"{\"owner\":\"Bob\",\"id\":2,\"type\":\"checking\"}"}],"response":
{"id":"89f37761-8b9f-4b2a-91e0-392e64da84cd"}}
```

```
[06/Jun/2025:15:28:08 +0100] QUERY requestID=103 correlationID="d08fe710-7394-41f9-9a80-26009f2bbd6e"
queryId=072a9fbf-528a-4f71-9045-ad541e7c2d73 {"requestId":"072a9fbf-528a-4f71-9045-
ad541e7c2d73","permutationId":"010c4062-92da-432f-a5f6-66b0a901c480","permutation":
[{"attribute":"User","value":"Bob"},{"attribute":"Action","value":"read"},
{"attribute":"Account","value":"{\"owner\":\"John\",\"id\":3,\"type\":\"savings\"}"}],"response":
{"id":"010c4062-92da-432f-a5f6-66b0a901c480",…}}
```

A single query request is separated into permutations in the log:

- Each possible combination of query attributes becomes a separate `permutation` with its own unique identifier and decision details.
- Each `permutation` is recorded as its own distinct log message.

Copyright © 2025 Ping Identity Corporation

- All permutations from the same query request are linked together by a shared `requestId`, `correlationID`, and `queryId`.

You can enable debug logging for the policy query audit log in the administrative console or with **`dsconfig`**.

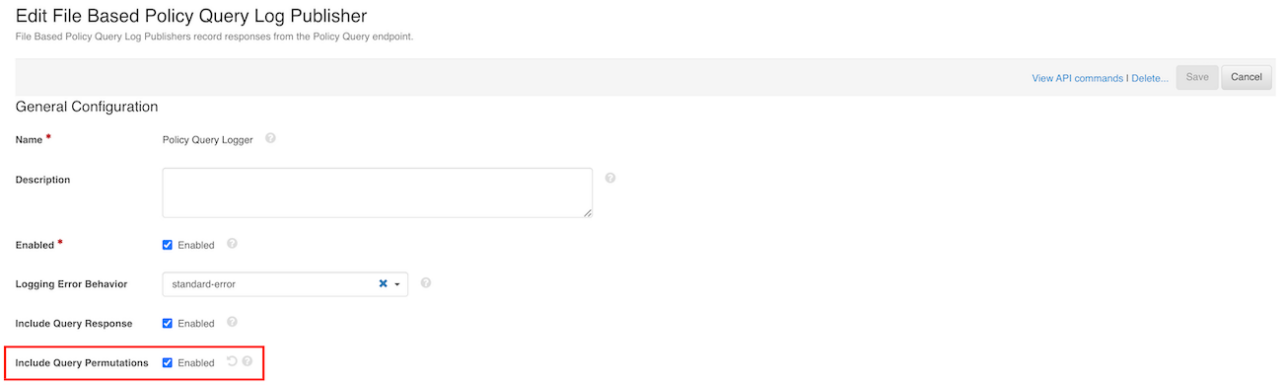
> **Note**
>
> By default, the policy query audit log is located at `PingAuthorize/logs/policy-query.log`.

Admin console

Enabling policy query debug logging in the administrative console

Steps

1. Go to **Logging, Monitoring, and Notifications** and click **Log Publishers**.
2. Click **Policy Query Logger**.
3. Select the **Include Query Permutations** checkbox.



4. Click **Save to PingAuthorize Server Cluster**.

Copyright © 2025 Ping Identity Corporation

dsconfig

Enabling policy query debug logging with `dsconfig`

Steps

1. Enable the file-based Policy Query Logger.

```
dsconfig set-log-publisher-prop
  --publisher-name "Policy Query Logger"
  --set enabled:true
```

2. Use the **`dsconfig set-log-publisher-prop`** command with the following arguments:

```
dsconfig set-log-publisher-prop
  --publisher-name "Policy Query Logger"
  --set include-query-permutations:true
```

Configuring the query permutation view

In addition to enabling query permutations in the Policy Query logger, you can specify additional levels of detail to include in each permutation's `response` field. The Policy Query logger provides the following additional views:

- `request`: Includes the decision request object
- `decision-tree`: Includes details of the policy tree's evaluation flow
- `attributes`: Includes details of attributes used during policy evaluation, including the attribute's value and type

> **Note**
>
> Specifying this view overrides any attribute logging configured for embedded mode.

- `services`: Includes details of services invoked during policy evaluation

> **Note**
>
> Specifying additional views will impact performance. Use these views for troubleshooting purposes and disable them afterward.

You can configure the query permutation view in the administrative console or with **`dsconfig`**.

Copyright © 2025 Ping Identity Corporation

Admin console

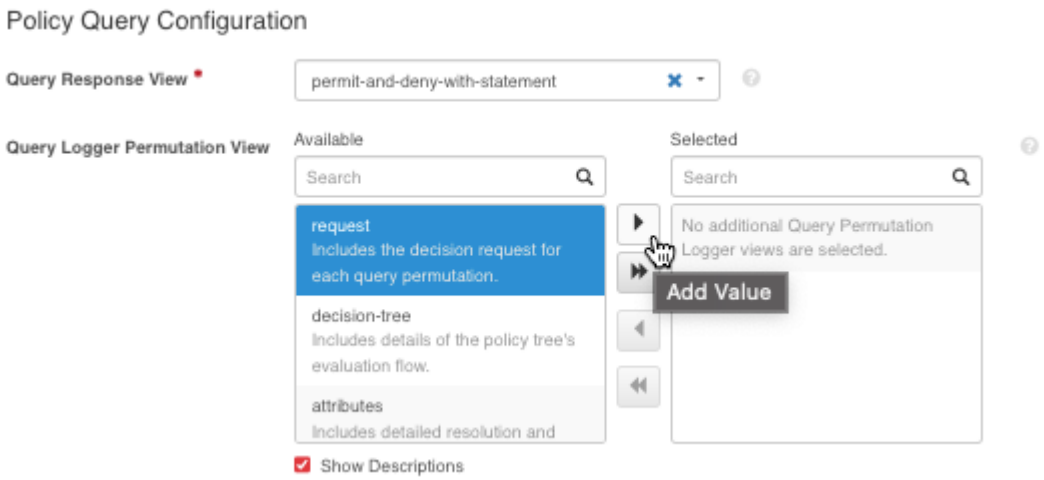
Configuring the query permutation view in the administrative console

Before you begin

Create a Policy Query Logger with debug logging enabled.

Steps

1. Go to **Authorization and Policies** and click **Policy Decision Service**.
2. In the **Policy Query Configuration** section, change the **Selected** views included for **Query Logger Permutation View** and click the arrow button.



3. Click **Save to PingAuthorize Server Cluster**.

Copyright © 2025 Ping Identity Corporation

dsconfig

Configuring the query permutation view with `dsconfig`
Before you begin

Create a Policy Query Logger with debug logging enabled.

Steps

- Use the **`dsconfig set-policy-decision-service-prop`** command with the `--add query-logger-permutation-view` argument to add query permutation views:

```
dsconfig set-log-publisher-prop
  --add query-logger-permutation-view:attributes
  --add query-logger-permutation-view:services
```

- Use the **`dsconfig set-policy-decision-service-prop`** command with the `--remove query-logger-permutation-view` argument to remove query permutation views:

```
dsconfig set-log-publisher-prop
  --remove query-logger-permutation-view:attributes
```

You can also configure policy query debug logging for policy development and testing in the Policy Editor. Learn more in Configuring policy query debug logging in the Policy Editor.

Repeating policies and attributes

Use repeating policies and attributes to evaluate a policy multiple times—once for each item in a collection.

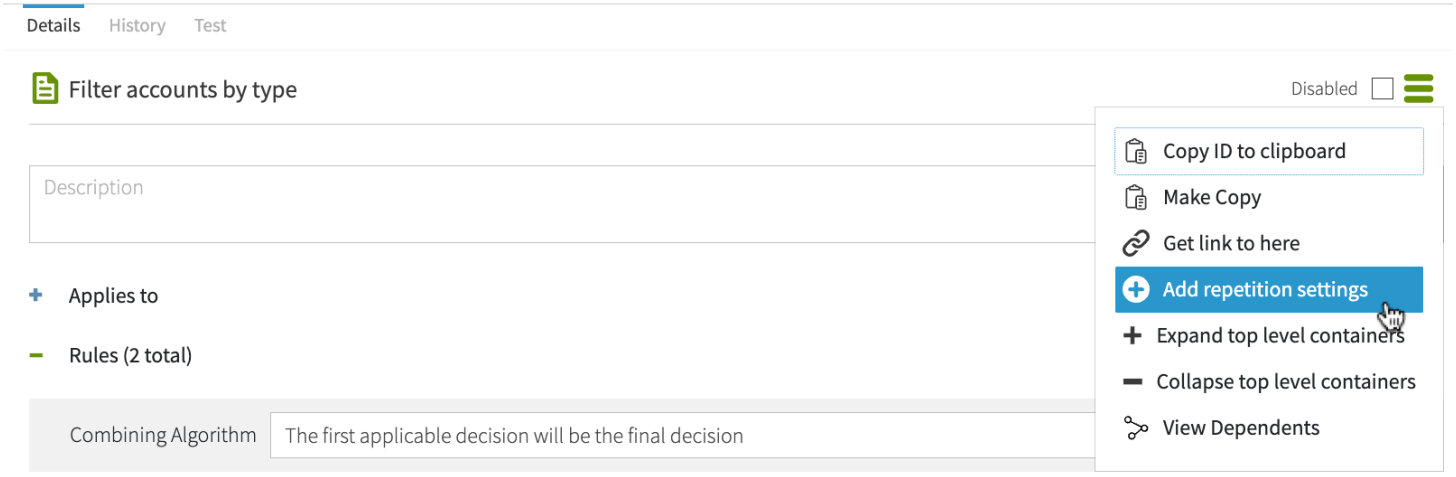
For example, assume the `Accounts` attribute contains a list of accounts associated with a customer. You want to filter access to the accounts based on the account type. With repeating policies, a decision is made for each item in the `Accounts` attribute, returning statements for each account that is permitted.

Repeating policies

To make a policy repeat, from the hamburger menu, select **Add repetition settings**.

> **Note**
>
> You can only add repetition settings to an existing policy. The hamburger menu to add these settings does not appear when you are creating a new policy.

Copyright © 2025 Ping Identity Corporation



The policy repetition settings are as follows:

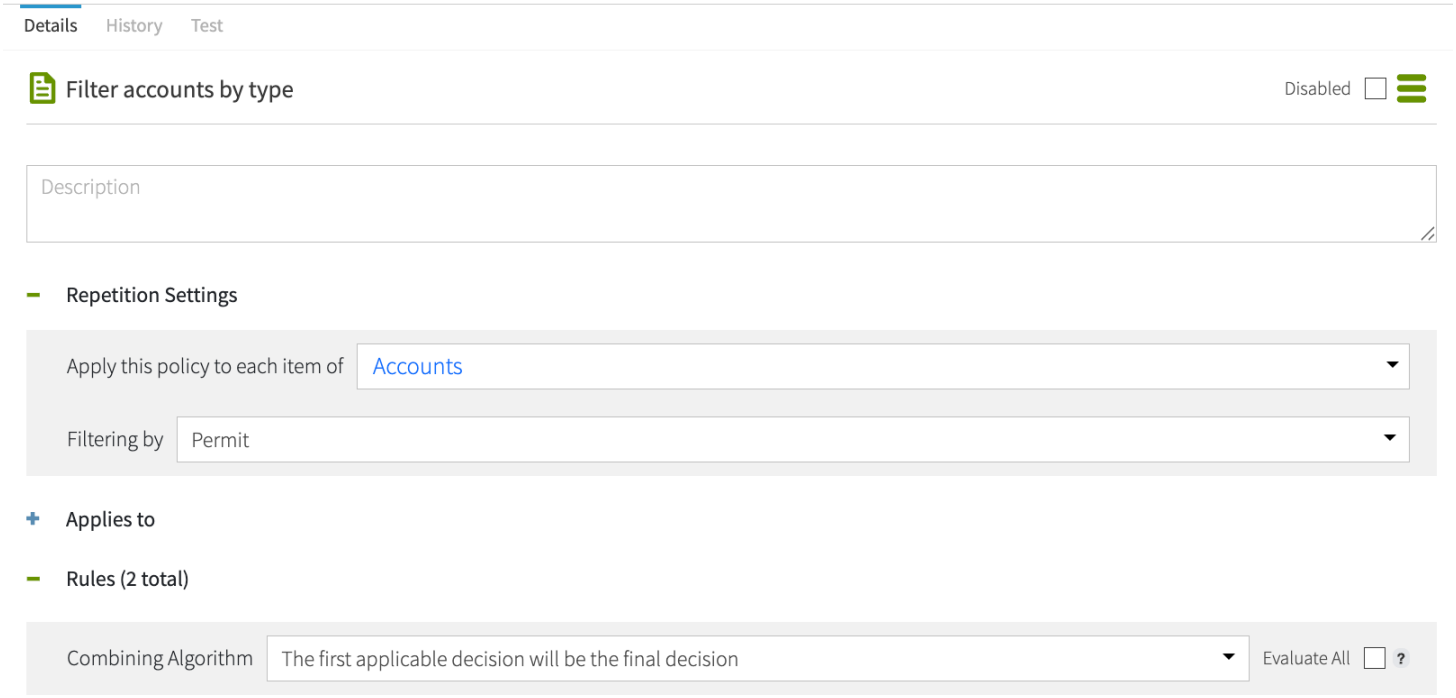
- **Apply this policy to each item of**

  This specifies the collection attribute to repeat over, referred to as the repetition source.

- **Filtering by**

  This represents the decision and any attached statement 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 statements for a repeating policy, you can use:

- Attributes with no repetition source

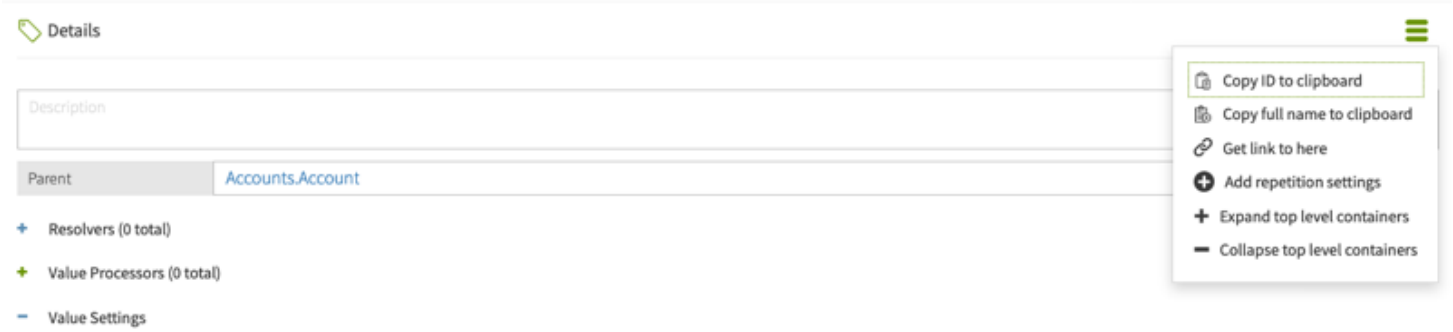
Copyright © 2025 Ping Identity Corporation

- Attributes with the same repetition source as the policy

Repeating attributes

To make an attribute repeat, from the hamburger menu, select **Add repetition settings**.

> **Note**
>
> You can only add repetition settings to an existing attribute. The hamburger menu to add these settings does not appear when you are creating a new attribute.



The attribute repetition settings are as follows:

- **Repeat for each item of**

  This specifies the attribute to repeat over, referred to as the repetition source.

  > **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.

- **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, Processors, and Attribute caching.

  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.

Self-governance

Self-governance enables you to regulate who can view or change your authorization controls within the Policy Editor.

Just as you build application policies to manage access to your API resources and functions, you build self-governance policies to manage access to your Policy Editor entities and operations. This allows you to protect against unauthorized or accidental application policy changes.

Copyright © 2025 Ping Identity Corporation

When you install the Policy Editor with self-governance enabled, the application automatically creates the **Admin Point Governance** branch, which contains the **System Policy Set** and several self-governance Trust Framework definitions.

> **Important**
>
> While self-governance controls are no different than standard Policy Editor controls, all of your self-governance controls must be contained within the **Admin Point Governance** branch. By default, this branch is only visible to the self-governance administrator. Any policy sets, policies, and rules you build must be children of the **System Policy Set**.

How does self-governance manage access?

When self-governance is enabled and a user attempts to perform an action within the Policy Editor, this action is subject to the authorization policies defined in the **System Policy Set** of the **Admin Point Governance** branch. These access controls are applied at the level of the Policy Editor REST API and are therefore applicable to users of the Policy Editor web application and clients of the REST API.

The Policy Editor automatically turns the attempted action into an internal decision request against these self-governance policies. This request includes:

- A service, which indicates the type of entity the user is interacting with (for example, policy set, policy, attribute, or branch)
- An action, which indicates what the user is trying to do (for example, read, update, or delete)
- Attributes that describe the existing state of the object being acted upon and the intended state of the object after the user's action
- Attributes that describe the current branch upon which the user is performing the action
- Attributes that describe the Policy Editor user performing the action

You can find more information about Self-governance Trust Framework for all of the relevant services, actions, attributes, and conditions that you can use to build policies for your Self-governance use cases.

Enable self-governance

Self-governance is not part of the baseline configuration for the Policy Editor and must be enabled during installation. Use the non-interactive setup for self-governance in either demo mode or OIDC mode, supplying the appropriate arguments. You can find more information about enabling self-governance in Installing the Policy Editor non-interactively.

> **Note**
>
> **Best Practice**: Self-governance is intended for use with a single Policy Editor instance. In an environment with multiple Policy Editor instances connected to a single database, restart each instance after committing your self-governance policy updates.

As part of setup, the Policy Editor creates a self-governance administrator account. This is a special user who is never subject to self-governance policies and always has full control over the entire Policy Editor.

> **Important**
>
> It is possible, through misconfiguration of self-governance policies, to completely lock regular users out of the Policy Editor. To address this scenario, the self-governance administrator can update or roll back self-governance policy updates to restore Policy Editor access.

HTTP caching

Copyright © 2025 Ping Identity Corporation

HTTP caching is enabled by default as part of a standard Policy Editor configuration, but can be permanently disabled. When using HTTP caching in combination with self-governance, the time-to-live value of the cache plays an important role in self-governance decisions. After the cache expires, the Policy Editor verifies that the user still has access to the resource, even if the resource is unchanged. You can find more information in HTTP caching.

Logging

Self-governance logging is enabled by default and all related logs are stored in the `logs` directory. Self-governance information is written to the following logs:

`decision-audit.log`

This log contains a list of policy decisions made by the policy decision point (PDP).

`management-audit.log`

The log contains a list of any actionable self-governance requests.

`debug.log`

This is the Policy Editor application log. You can use it to debug any application-related issues.

Self-governance use cases

You can use self-governance to control access to Policy Editor entities and operations in a wide variety of ways.

Controls that you can configure using self-governance include:

- Protecting a policy set from deletion
- Ensuring a policy can never be updated
- Preventing policies from being added or created in a policy set
- Blocking a user's ability to delete attributes
- Restricting a user's ability to read policies or policy sets
- Allowing attributes to be elevated to secret status while forbidding secret attributes from moving to non-secret status

The following use cases demonstrate how to build and deploy some common self-governance policies.

> **Note**
>
> To view a visual flow of your self-governance policy decisions, see Visualizing a policy decision response.

Use case: Preventing a user from viewing a branch

About this task

The following are the general steps needed to prevent a user from viewing a branch:

Copyright © 2025 Ping Identity Corporation

Steps

1. Create a policy named `Prevent a user from viewing a branch` in the **System Policy Set** of the **Admin Point Governance** branch.
2. Add a rule and name it `Deny if branch is Admin Point Governance and user is Steve Smith`.
3. Add a condition that returns true when **Branch.Name** equals `Admin Point Governance`.
4. Add a condition that returns true when **user.name** equals `Steve Smith`.
5. Select **Components** and drag the **Core.Branch** service to the **Applies to** section of the rule.
6. Drag the **Read** action to the **Applies to** section of the rule.
7. Save the policy.
8. Commit the changes to the **Admin Point Governance** branch in **Branch Manger > Version Control**.

Result

This self-governance policy returns a `Deny` response if a user named Steve Smith tries to view the **Admin Point Governance** branch.

Use case: Preventing users from updating a policy set

About this task

The following are the general steps needed to prevent users from updating a policy set:

Steps

1. Create a policy named `Prevent users from updating a policy set` in the **System Policy Set** of the **Admin Point Governance** branch.
2. Add a rule that returns a `Deny` if **Object.Existing.Name** equals `examplePolicySet`.
3. Select **Components** and drag the **Core.PolicySet** service to the **Applies to** section of the rule.
4. Drag the **Modify.Update** action to the **Applies to** section of the rule.
5. Save the policy.
6. Commit the changes to the **Admin Point Governance** branch in **Branch Manger > Version Control**.

Result

This self-governance policy returns a `Deny` response when a user tries to modify a policy set with the name **examplePolicySet**.

Use case: Preventing users from deleting policies

About this task

The following are the general steps needed to prevent users from deleting policies:

Copyright © 2025 Ping Identity Corporation

Steps

1. Create a policy named `Prevent users from deleting policies` in the **System Policy Set** of the **Admin Point Governance** branch.
2. Add a rule that always returns a `Deny` response.
3. Select **Components** and drag the **Core.Policy** service to the **Applies to** section of the rule.
4. Drag the **Modify.Delete** action to the **Applies to** section of the rule.
5. Save the policy.
6. Commit the changes to the **Admin Point Governance** branch in **Branch Manger > Version Control**.

Result

This self-governance policy returns a `Deny` response when a user tries to delete a policy within the system.

Self-governance Trust Framework

To make it easier to get started developing self-governance policies, the **Admin Point Governance** branch initializes with a default set of Trust Framework definitions.

Use the self-governance attributes and conditions to build your policy logic and then include self-governance actions and services to target when your policies will apply. The following tables describe the included Trust Framework definitions for self-governance.

> **Note**
>
> Avoid using Trust Framework definitions related to self-governance permissions. While some of these items are visible and exposed within the Policy Editor, the permissions system is not enabled or supported.

Attribute Name	Scope	Description
Branch	All operations	JSON data regarding the branch on which the current operation is being performed.
Branch.Id	All operations	GUID of the branch on which the current operation is being performed.
Branch.Name	All operations	Name of the branch on which the current operation is being performed.
Branch.ParentId	All operations	GUID of the parent branch for the branch on which the current operation is being performed.
DeploymentPackage	Deployment package operations	A folder to contain nested attributes. It has no value of its own.

Copyright © 2025 Ping Identity Corporation

Attribute Name	Scope	Description
DeploymentPackage.Decision Node ID	Deployment package operations	GUID of the decision node referred to by the deployment package that the user is acting upon.
DeploymentPackage.Snapshot Id	Deployment package operations	GUID of the snapshot referred to by the deployment package that the user is acting upon.
fromId	Diff or merge operations	The `from` ID argument passed to the service, if `from` and `to` arguments are required.
id	Diff or merge operations	The ID argument passed to the service.
name	Diff or merge operations	The string argument passed to the service.
Object	None	A folder to contain nested attributes. It has no value of its own.
Object.Existing	All operations	JSON data containing details of the current state of the object that the user is acting upon. Nested attributes below this attribute in the hierarchy extract specific values from this JSON data using JSONPath.
Object.Existing.Approvals	None	Collection of all the approvals for the deployment package that the user is acting upon.
Object.Existing.Approvals.UserIds	Deployment package operations	Collection of user IDs for all approvals on the deployment package that the user is acting upon.
Object.Existing.Approvals.Count	Deployment package operations	Total number of approvals on the deployment package that the user is acting upon.
Object.Existing.AttributeResolvers	Attributes	Collection of all the resolvers for the attribute that the user is acting upon.
Object.Existing.BranchId	Attributes	The branch ID of the object being acted upon by the user.
Object.Existing.CacheConfig	Attributes	JSON data detailing the cache settings for the attribute that the user is acting upon.

Copyright © 2025 Ping Identity Corporation

Attribute Name	Scope	Description
Object.Existing.Children	All operations	The direct first level children of the object that the user is acting upon.
Object.Existing.CombiningAlgorithm	Policy sets, policies	The combining algorithm of the policy or policy set that the user is acting upon.
Object.Existing.CustomProperties	All operations	JSON data of the custom properties set for the object that the user is acting upon.
Object.Existing.DefaultValue	Attributes	The default value of the attribute that the user is acting upon.
Object.Existing.DefinitionId	Trust Framework definitions	The GUID of the Trust Framework definition that the user is acting upon.
Object.Existing.Description	All operations	The description of the object that the user is acting upon.
Object.Existing.Disabled	Policy sets, policies, rules	Boolean value indicating whether or not the policy node that the user is acting upon is disabled.
Object.Existing.FullName	Trust Framework definitions	The full name, including parent names, of the Trust Framework definition.
Object.Existing.Id	All operations	The GUID of the object that the user is acting upon.
Object.Existing.IdentityProperties	Identity classes, identity providers	Collection of all the identity properties for the definition that the user is acting upon.
Object.Existing.IsPresent	All operations	Boolean that is true if the object that the user is acting upon is present.
Object.Existing.Name	All operations	The name of the object being acted upon by the user.
Object.Existing.Name.IsPresent	All operations	Boolean that is true if the **Object.Existing.Name** attribute is present in the self-governance decision request.
Object.Existing.ObjectType	Trust Framework	The type of Trust Framework definition that the user is acting upon.

Copyright © 2025 Ping Identity Corporation

Attribute Name	Scope	Description
Object.Existing.ParentId	All operations	The GUID of the direct parent of the object being acted upon by the user.
Object.Existing.Resolvers	Attributes	JSON value detailing the resolvers of the attribute that the user is acting upon.
Object.Existing.Secret	Attributes	Boolean indicating whether the attribute that the user is acting upon has been marked as secret.
Object.Existing.ServiceSettings	Services	JSON value detailing all of the service settings for the service that the user is acting upon.
Object.Existing.ServiceType	Services	The type of the service that the user is acting upon.
Object.Existing.Shared	Rules, targets, statements	Boolean that is true if the object that the user is acting upon is shared (appears in the **Library**).
Object.Existing.Statements	Policy nodes	Collection containing the list of statements for the policy node that the user is acting upon.
Object.Existing.Targets	Policy nodes	Collection containing the list of targets for the policy node that the user is acting upon.
Object.Existing.TestCase	Test case definitions	JSON representation of the test case associated with this definition.
Object.Existing.TestScenario	Test scenario definitions	JSON representation of the test scenario associated with this definition.
Object.Existing.Type	Definition	The type of definition (Trust Framework or Test Suite entity) that the user is acting upon.
Object.Existing.Version	All operations	Version of the entity that the user is acting upon.
Object.Intended	All operations	JSON data containing details of the intended state of the object after the action the user is trying to perform. Nested attributes below this attribute in the hierarchy extract specific values from this JSON data using JSONPath.

Copyright © 2025 Ping Identity Corporation

Attribute Name	Scope	Description
Object.Intended.*	All operations	**Object.Intended** has the same child attribute structure as **Object.Existing**.
Snapshot	Snapshots	JSON data regarding the commit that the user is acting upon.
Snapshot.Approval Count	Snapshots	Number of approvals on the commit that the user is acting upon.
Snapshot.Approvals	Snapshots	Collection of names of all users who have approved the commit that the user is acting upon.
Snapshot.BranchId	Snapshots	GUID of the branch of the commit that the user is acting upon.
Snapshot.Id	Snapshots	GUID of the commit that the user is acting upon.
Snapshot.ParentId	Snapshots	GUID of the direct parent of the commit that the user is acting upon.
Snapshot.State	Snapshots	Current state of the commit that the user is acting upon. The value can be either `UNCOMMITTED` or `COMMITTED`.
toId	Diff or merge operations	ID of the entity the change is being merged to.
user	All operations	JSON data describing the user performing the action. User attributes like LDAP properties or OIDC claims are mapped as keys and values, where a single value is expressed as a JSON array. If OIDC claims have an attribute nickname with value `abc`, the JSON data will be `\{"nickname":["abc"]}`. To get the scalar value of an attribute, use a processor and access the zeroth value of the JSON array. Any derived user attribute that contains sensitive information must be marked as secret in **Value Settings**.
user.name	All operations	Name of the user performing the action.

Copyright © 2025 Ping Identity Corporation

Attribute Name	Scope	Description
user.name.lowercase	All operations	Lowercase value of the name.

Service Name	Description
Core	A folder containing child services. It has no other function.
Core.Branch	Targets operations involving branches.
Core.Definition	Targets operations involving Trust Framework definitions.
Core.Delta	Targets operations involving **Version Control** deltas.
Core.DeploymentPackage	Targets operations involving deployment packages.
Core.DiffMerge	Targets **Version Control** diff or merge operations.
Core.Entity.Change	Targets operations involving entity changes.
Core.Policy	Targets operations involving policies.
Core.PolicySet	Targets operations involving policy sets.
Core.RecentDecisions	Targets operations involving the recent decisions diagnostics buffers.
Core.RecentDecisions.Configuration	Targets operations involving the configuration of a recent decisions buffer.
Core.RecentDecisions.Content	Targets operations involving the content of a recent decisions buffer.
Core.Rule	Targets operations involving rules.
Core.Snapshot	Targets operations involving snapshots.
Core.Statement	Targets operations involving statements (obligations or advice).
Core.Target	Targets operations involving targets.
Test.Scenario	Targets operations involving test scenarios and scenario groups.
Test.TestCase	Targets operations involving test cases and test groups.

Copyright © 2025 Ping Identity Corporation

Action Name	Scope	Description
Modify	All operations	Targets any modification: commit, create, delete, import, roll back, or update.
Modify.Commit	**Version Control**	Targets commits within **Branch Manager → Version Control**.
Modify.Create	All operations	Targets the creation of any object, including branches, attributes, policies, and so on.
Modify.Delete	All operations	Targets the deletion of any object, including branches, attributes, policies, and so on.
Modify.Import	Snapshots	Targets importing snapshots.
Modify.Rollback	**Version Control**	Targets attempts to roll back deltas within **Version Control**.
Modify.Update	All operations	Targets the update of any object, including branches, attributes, policies, and so on.
Read	All operations	Targets reading any object, including branches, attributes, policies, and so on.
Read.Diff	**Version Control**	Targets reading a diff within **Version Control**.
Read.Export	Deployment packages and snapshots	Targets exporting a snapshot or deployment package.
Read.List	All operations	Targets any operations that read a listing (in other words, list all branches).
Read.History	All operations	Targets operations that attempt to read the history of an entity.

Condition Name	Scope	Description
Object.Is Root	All operations	True if the object is a root element (in other words, has no parent) but otherwise false.

Copyright © 2025 Ping Identity Corporation

Condition Name	Scope	Description
Object.Is Shared	All operations	True if the object is a shared element (in other words, it is in the **Library** and has no parent) but otherwise false.
Object.Is Shared.Existing	All operations	True if the object is already shared before being updated but otherwise false.
Object.Is Shared.Intended	All operations	True if the object will be shared after being updated but otherwise false.

Importing a self-governance policy snapshot

If you have existing self-governance policies in a policy snapshot, you can import them using the standard snapshot import process, with some minor adjustments.

Before you begin

Enable self-governance and sign on to the Policy Editor as the self-governance administrator.

Steps

1. Deactivate the previous self-governance policies:

   Choose from:

   - Delete the existing `Admin Point Governance` branch.
   - Rename the existing `Admin Point Governance` branch.

   > **Caution**
   >
   > After renaming or deleting the branch, all change requests subject to self-governance will be denied until the Policy Editor detects a branch named `Admin Point Governance`.

2. Import the new self-governance branch and name it `Admin Point Governance`.

   Learn more in Loading a policy snapshot.

3. Commit the changes in **Branch Manager > Version Control**.

Result

You've imported a new `Admin Point Governance` branch and activated its self-governance policies. If you didn't delete the previous branch, you've also created a backup of the previous self-governance policies, which you can roll back to if needed.

Copyright © 2025 Ping Identity Corporation

Policy solutions

This section recommends how to implement commonly needed business rules in policy.

- Use case: Using consent to determine access to a resource
- Use case: Using consent to change a response
- Use case: Using a SCIM resource type or a policy request action to control behavior
- Restricting the modification of attributes

Use case: Using consent to determine access to a resource

PingAuthorize can provide attribute-based access control to a specific protected 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 game API used in Getting started with PingAuthorize (tutorials). 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
2. Getting the requestor identifier from the access token
3. Searching for consent granted by resource owner to requestor
4. Getting consent status from the consent record
5. Creating a policy to check consent and then permit or deny access

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.

About this task

Copyright © 2025 Ping Identity Corporation

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.

   1. In the administrative console, click **Configuration** and then **Gateway API Endpoints**.
   2. Click **New Gateway API Endpoint**.
   3. For **Name**, specify `meme-game user_answers`.
   4. 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.

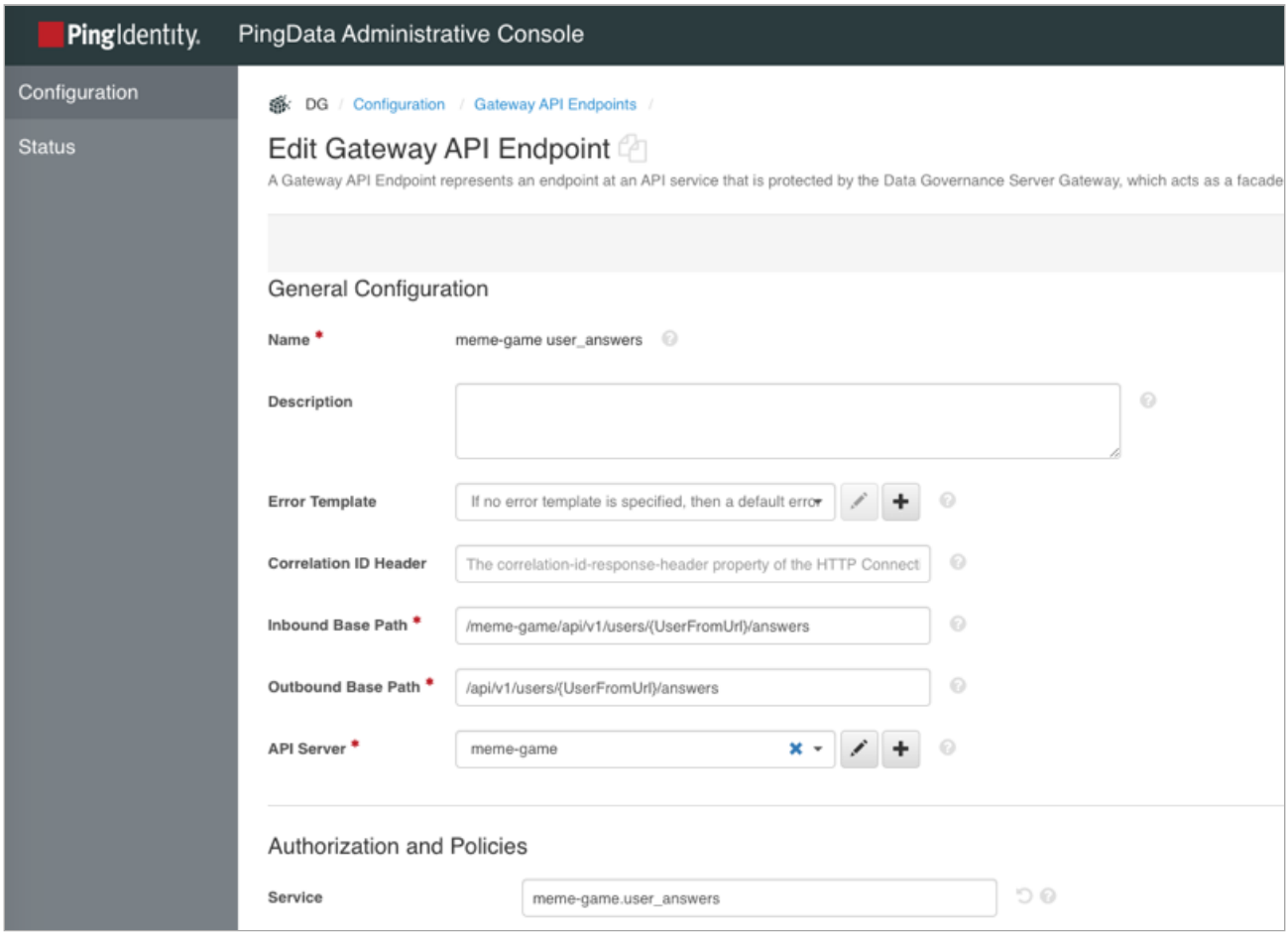
      Using the curly braces (`\{` and `}`) around a string creates an item with the name given by the string so that we can refer to it later. That notation also preserves the item to pass along in the next step.
   5. For **Outbound Base Path**, specify `/api/v1/users/{UserFromUrl}/answers`.

      The outbound base path defines the base request path for requests that PingAuthorize Server forwards to an API server.
   6. For **API Server**, specify `meme-game`. This is the API External Server you defined previously.
   7. 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.

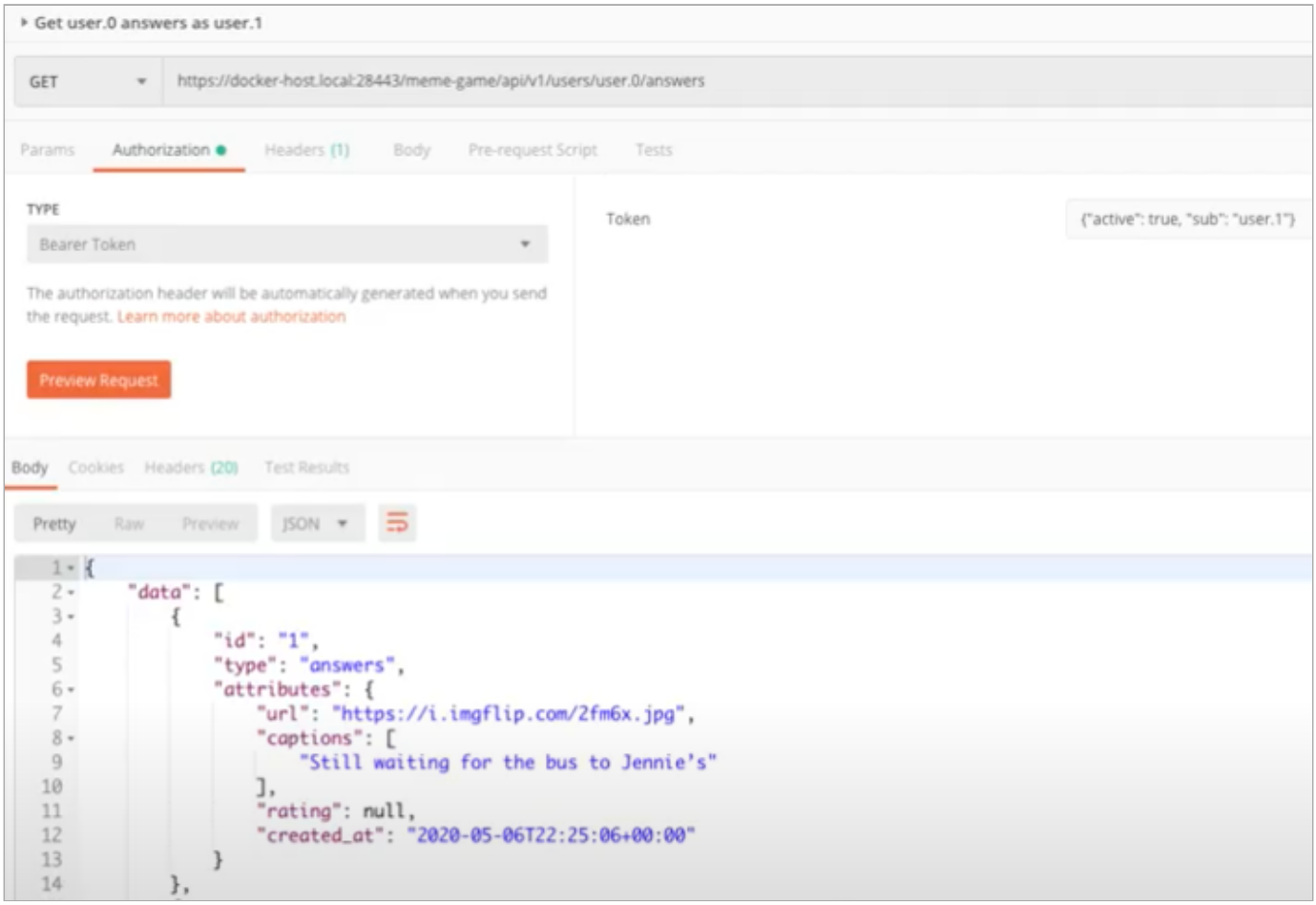
Copyright © 2025 Ping Identity Corporation



8. Save your changes.

2. Send a test request to the gateway to see how PingAuthorize handles the request. The following request uses Postman.

Copyright © 2025 Ping Identity Corporation

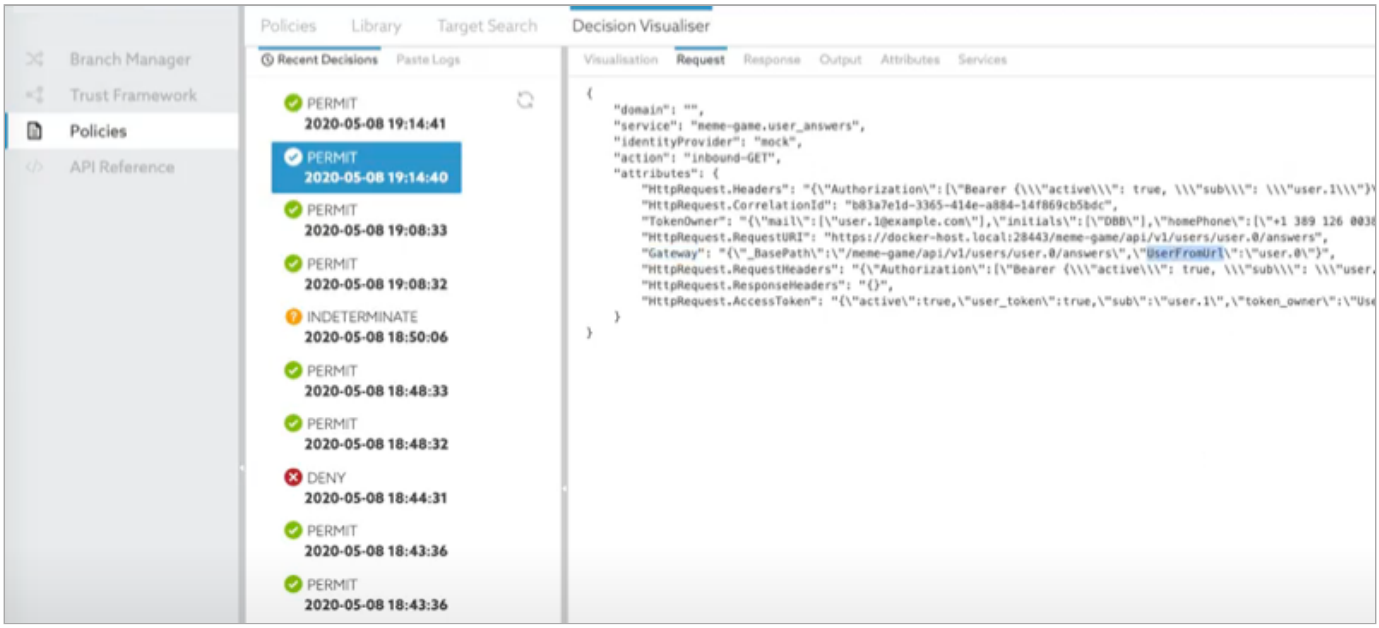


3. Check the request in the Policy Editor.

   Go to **Policies** in the left pane and then click **Decision Visualiser** along the top. Under **Recent Decisions**, click the **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.

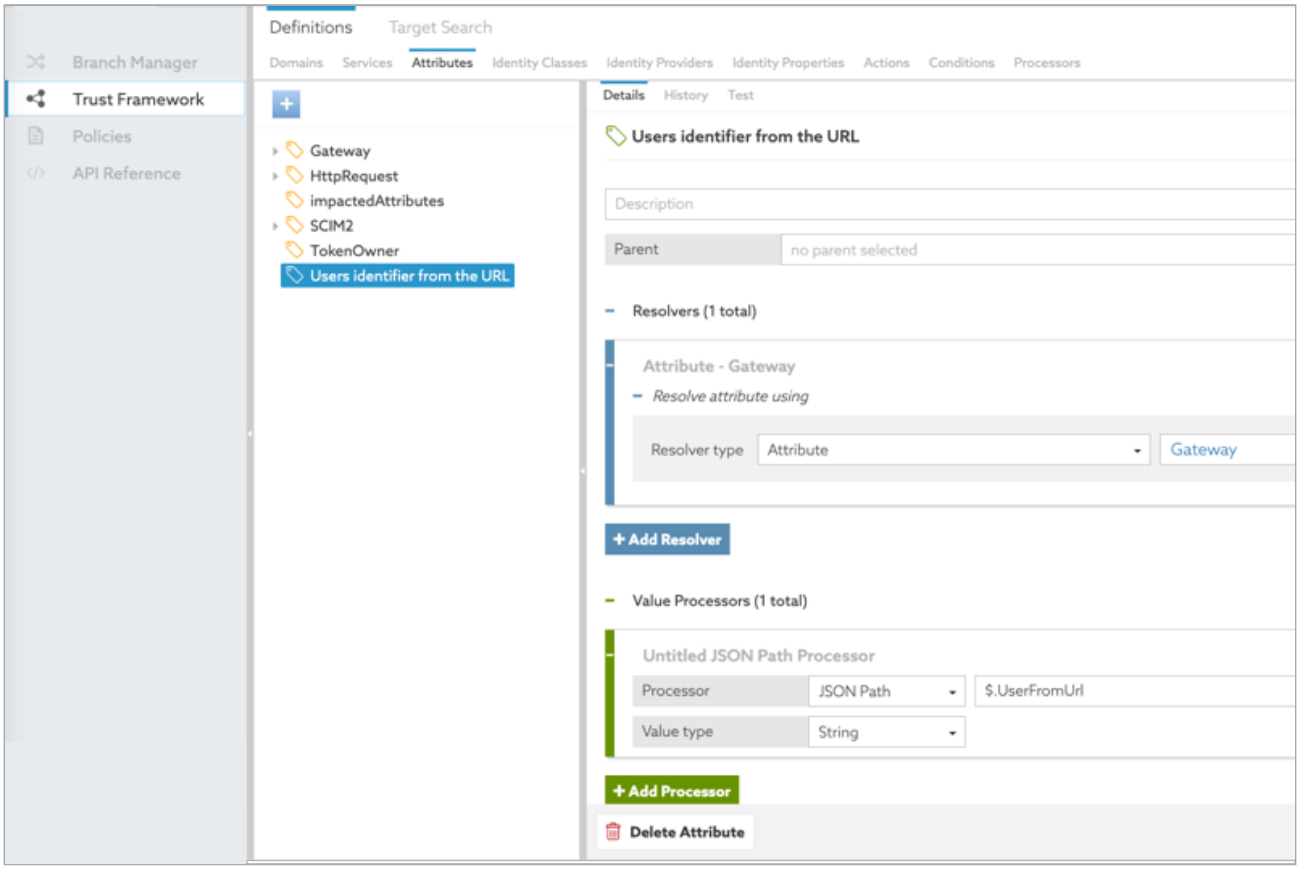
Copyright © 2025 Ping Identity Corporation



4. Create an attribute to pull UserFromUrl from the object.
    1. Go to **Trust Framework** and then click **Attributes** along the top.
    2. From the + menu, select **Add new Attribute**.
    3. For the name, replace **Untitled** with `Users identifier from the URL`.
    4. Click the + next to **Resolvers** and click + **Add Resolver**.
    5. Set **Resolver type** to **Attribute** and select the **Gateway** attribute.
    6. Click the + next to **Value Processors** and click + **Add Processor**.
    7. 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.

Copyright © 2025 Ping Identity Corporation



8. Click **Save changes**.

5. Test the new attribute.

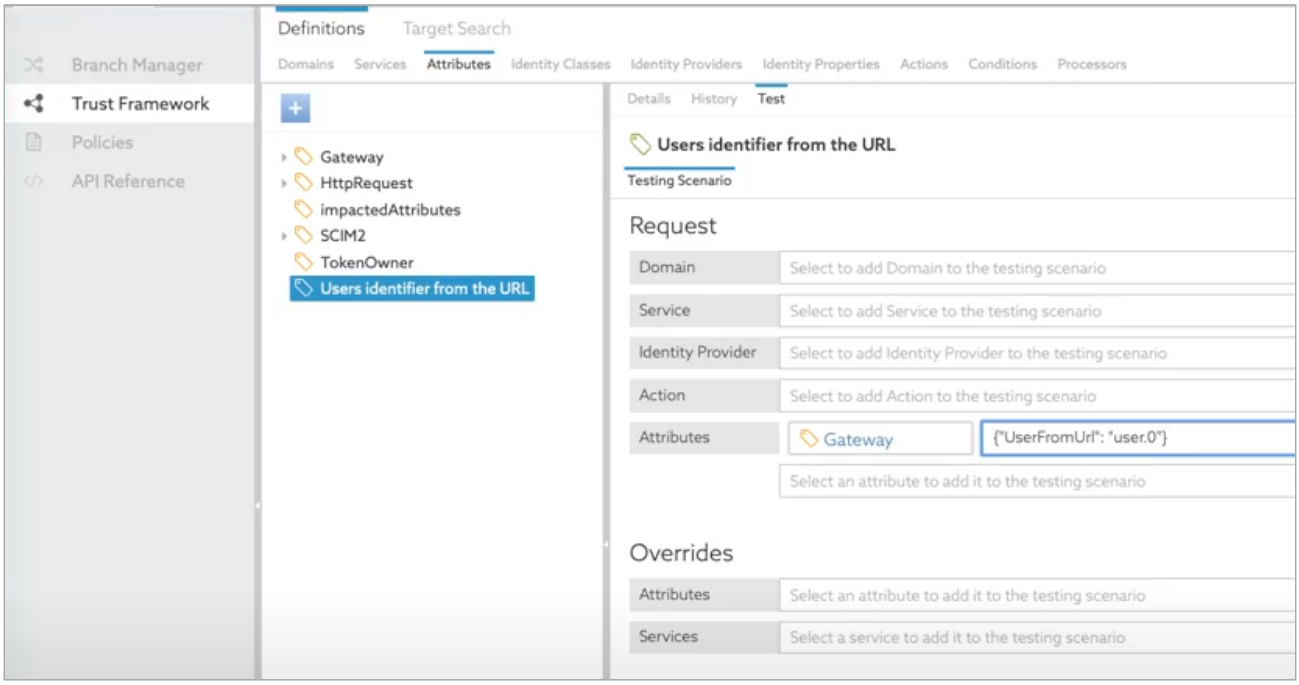
    1. Click **Test** just above the attribute name.

    2. 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.

Copyright © 2025 Ping Identity Corporation



3. Click **Execute**.

   The test result 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 granted 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)

Copyright © 2025 Ping Identity Corporation

- 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.

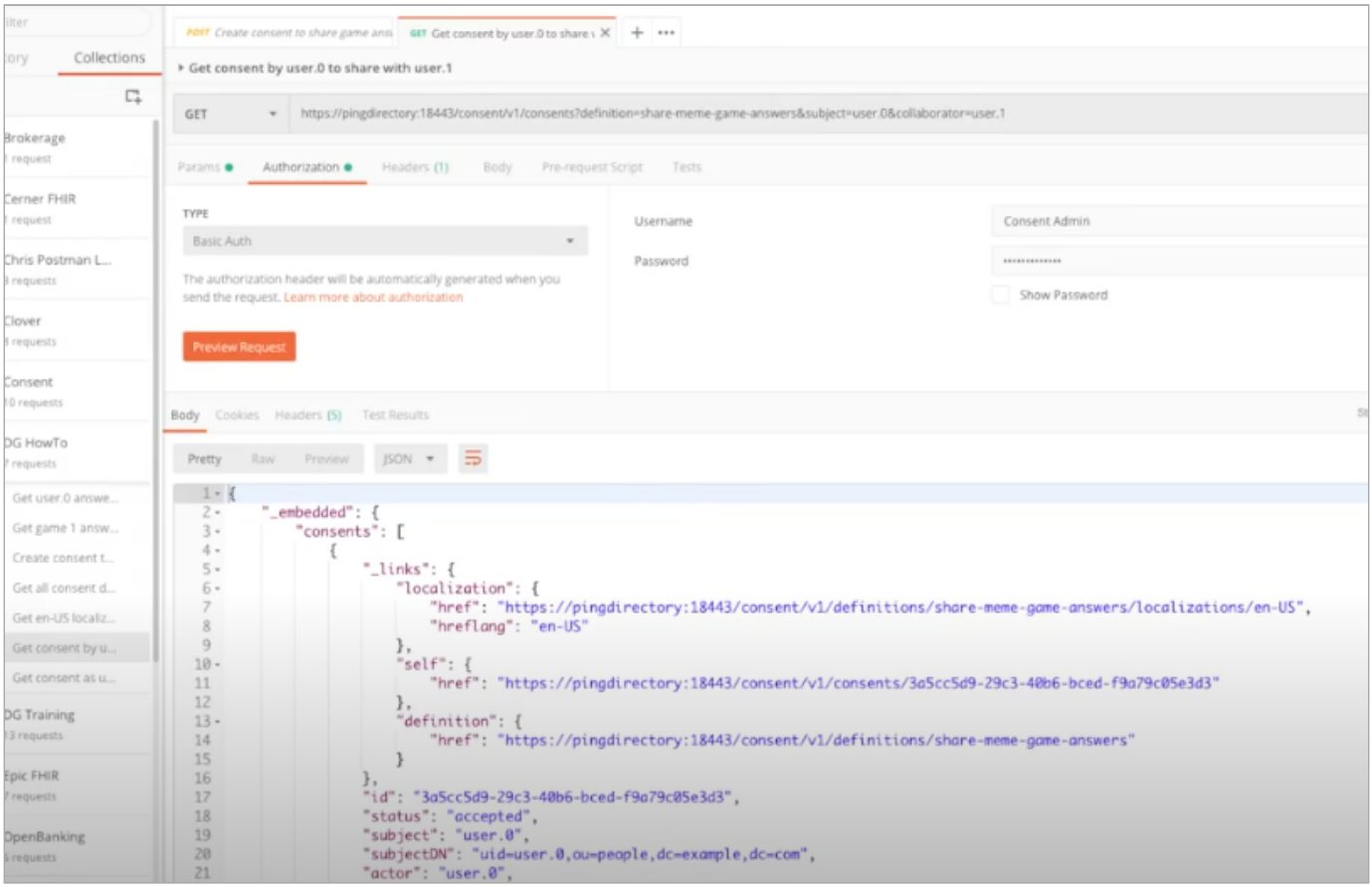
Steps

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 **Basic Auth** 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.

   1. Go to **Trust Framework** and click **Attributes**.

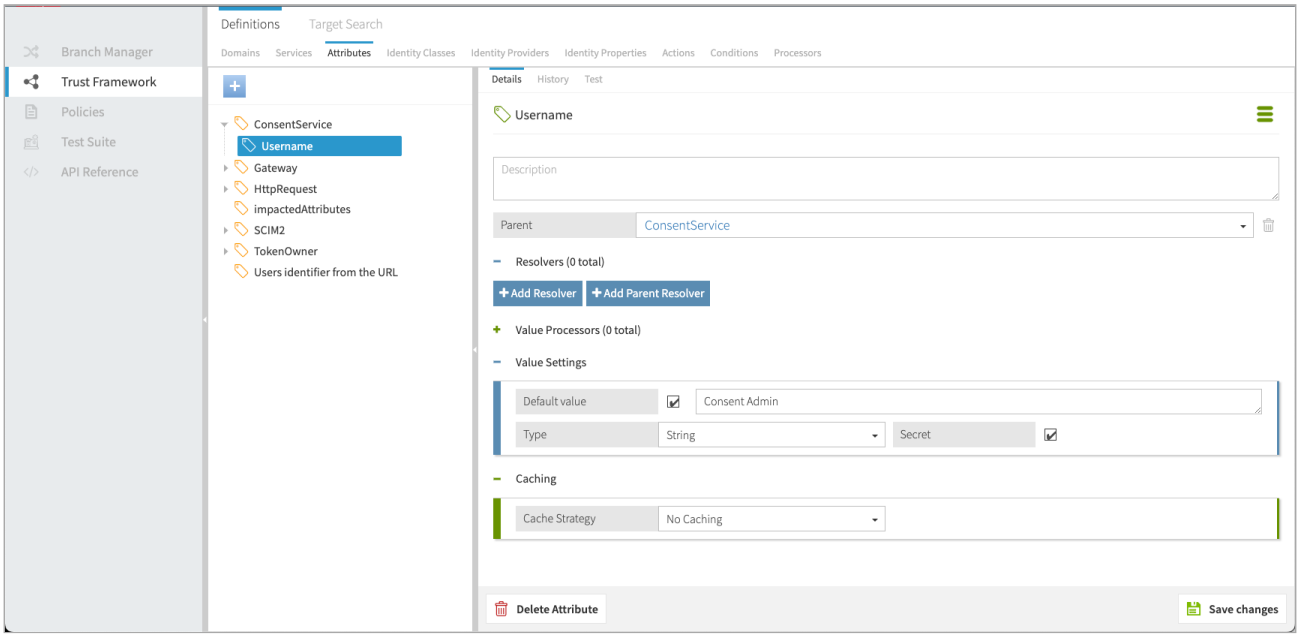
Copyright © 2025 Ping Identity Corporation

2. From the + menu, select **Add new Attribute**.
3. For the name, replace **Untitled** with `ConsentService` and click **Save changes**.

   This attribute will serve as a parent to the username and password attributes and will help organize the attributes.
4. From the + menu, select **Add new Attribute**.

   Because the **ConsentService** attribute is selected, the new attribute is a child to it.
5. 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.



6. From the + menu, select **Add new Attribute**.
7. 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.

   1. Click **Services** along the top.
   2. From the + menu, select **Add new Service**.
   3. For the name, replace **Untitled** with `Search for consent to share game answers`.
   4. Set **Service Type** to HTTP.
   5. 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.
   6. Set **Authentication** to **Basic**.

Copyright © 2025 Ping Identity Corporation

This setting requires a username and password. We will use the attributes we just created.

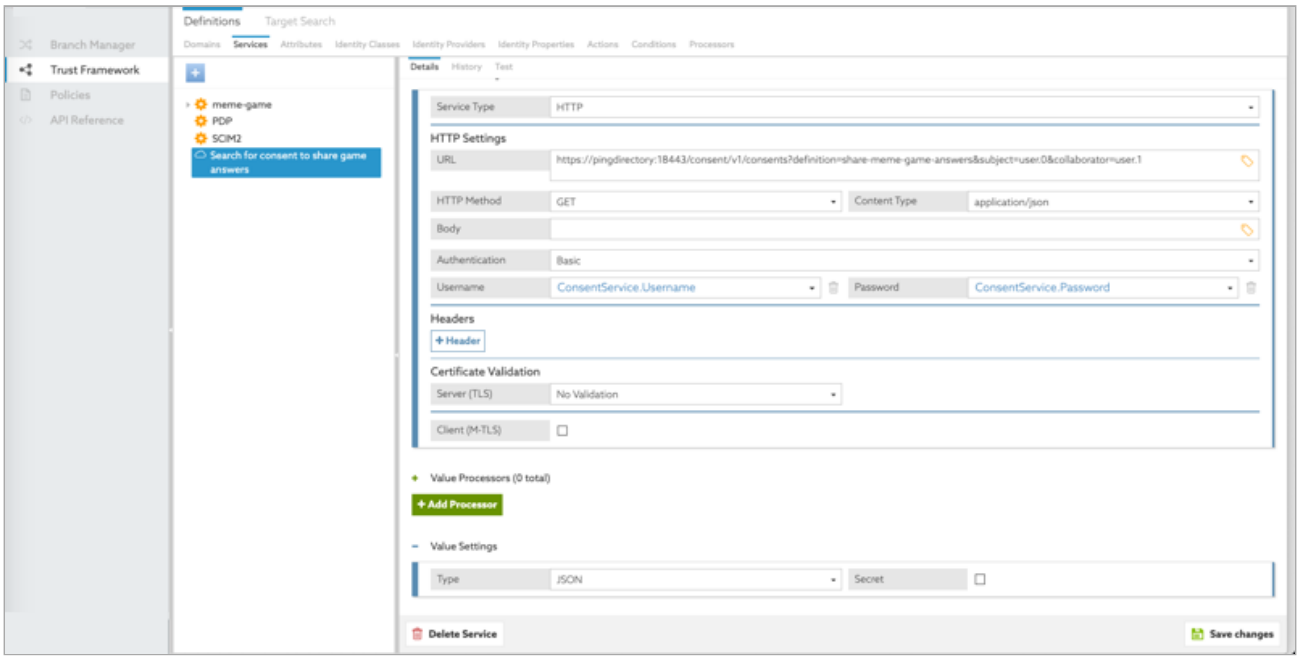
1. Set **Username** to **ConsentService.Username**.
2. Set **Password** to **ConsentService.Password**.

7. This setup uses a self-signed certificate, so set **Server (TLS)** to **No Validation**.

> **Caution**
>
> This case is for a development environment only. Do not use this setting for other environments.

8. Under **Value Settings**, set **Type** to **JSON**.

   The following image shows this configuration.



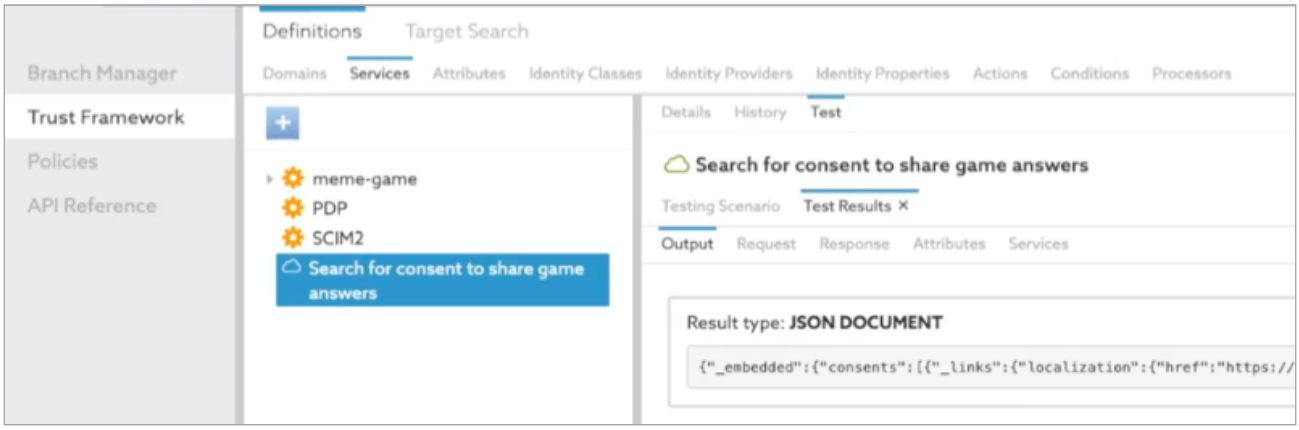
9. Click **Save changes**.

6. Test the service.

   1. Click **Test** above the **Search for consent to share game answers** service name.
   2. Click **Execute**.

      The results should include a `consents` array.

Copyright © 2025 Ping Identity Corporation



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.

    1. 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 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}}

    2. Click **Save changes**.

    3. 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.

    4. 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 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}}

    5. Click **Save changes**.

    6. Test this change the same way you tested the previous change: using two users, where one has a consent record and one does not.

Copyright © 2025 Ping Identity Corporation

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.

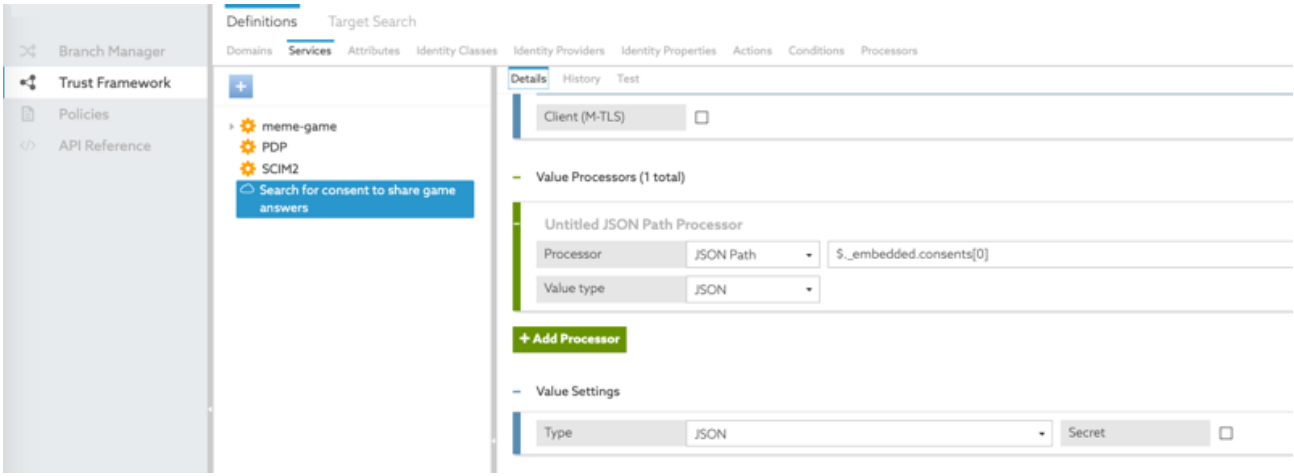
   The response starts with

   \{"_embedded":\{"consents":[\{_links":"localization":

   We want to pull the first consent record for the user, which starts after the square bracket (`[`).

   1. Click **Details** to return to the service definition.
   2. Click the + next to **Value Processors** and click + **Add Processor**.
   3. Set **Processor** to **JSON Path** with a value of `$._embedded.consents[0]`.
   4. Set **Value type** to **JSON**.

      For an example, see the following image.

   5. Click **Save changes**.
   6. 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, set **Attributes** to `Users identifier from the URL` with the value specifying user.0 again, and click **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

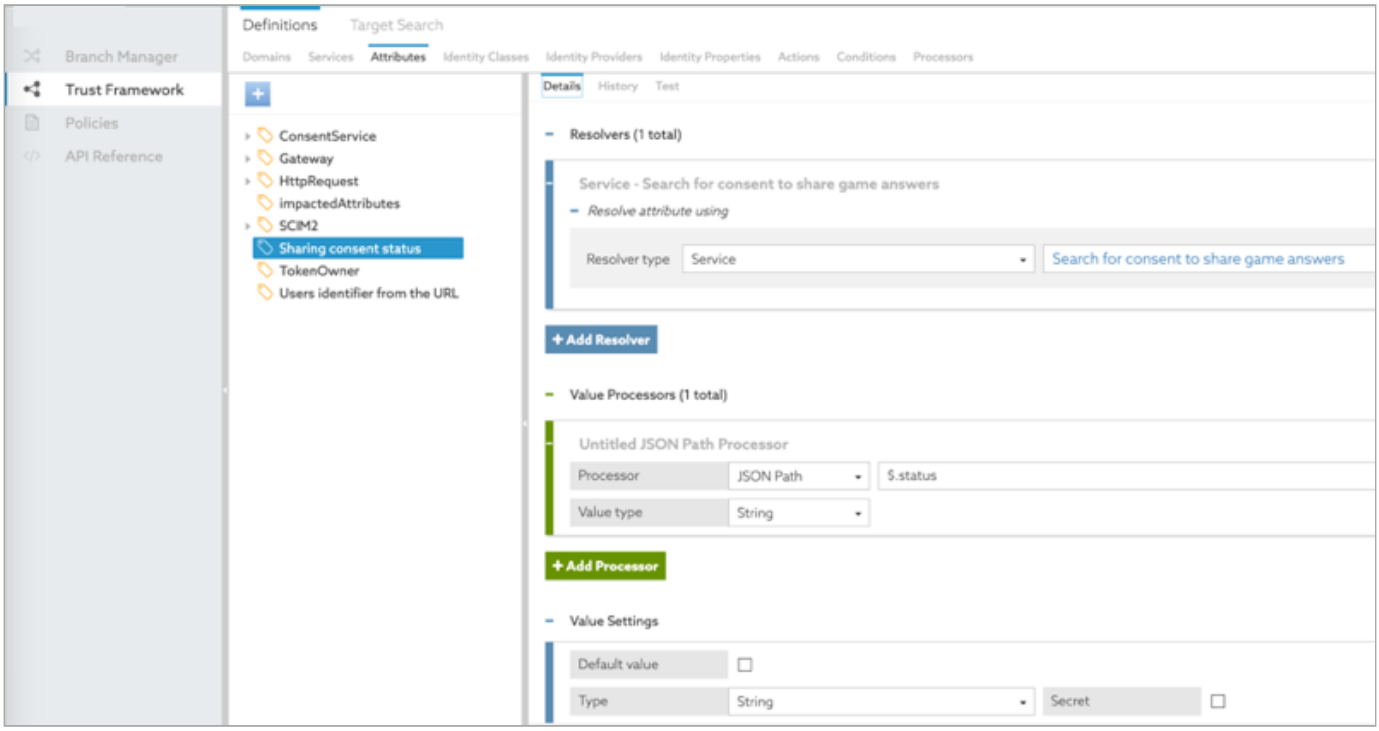
This task defines an attribute that uses a service to get a consent record and then uses a processor to pull the consent status from that record.

Copyright © 2025 Ping Identity Corporation

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.

Copyright © 2025 Ping Identity Corporation

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

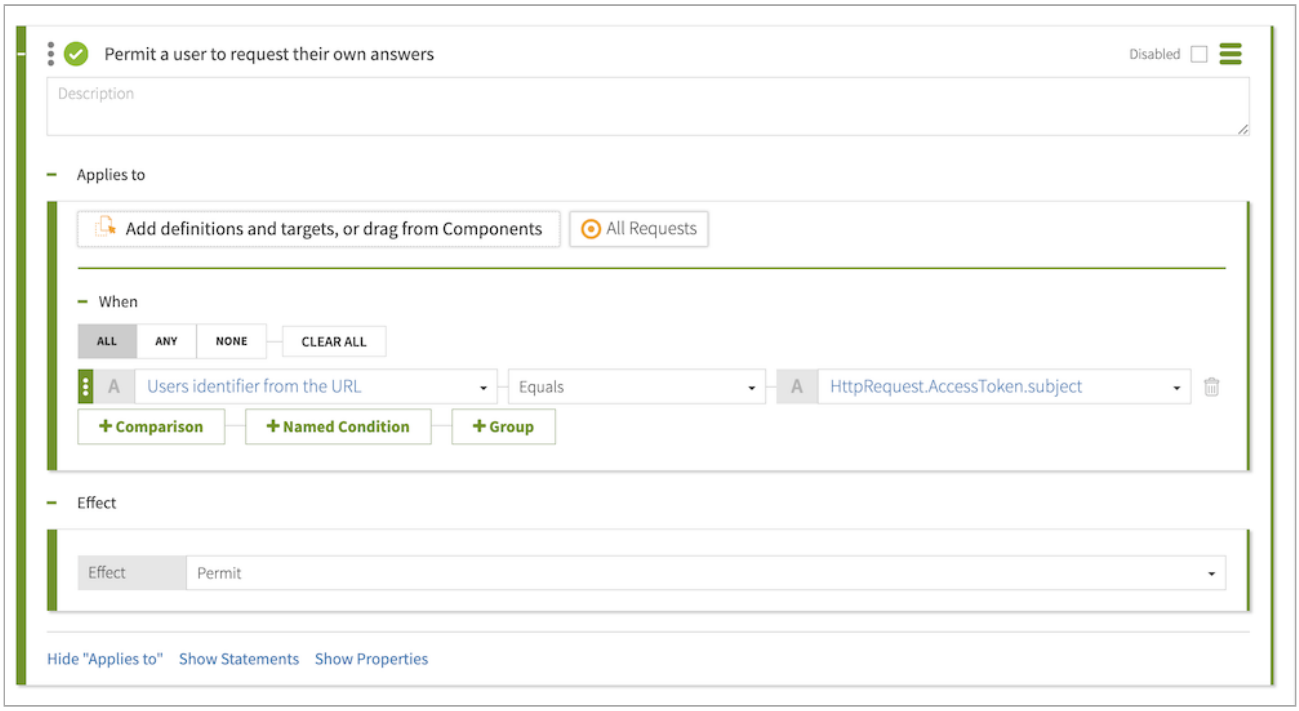
1. In the Policy Editor, go to **Policies** in the left pane and then click **Policies** along the top.

   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. 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.
   1. Click + **Add Rule.**
   2. For the name, replace **Untitled** with `Permit a user to request their own answers`.
   3. Click + **Comparison**.
   4. 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.
   5. In the second field, select **Equals**.
   6. 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.

Copyright © 2025 Ping Identity Corporation

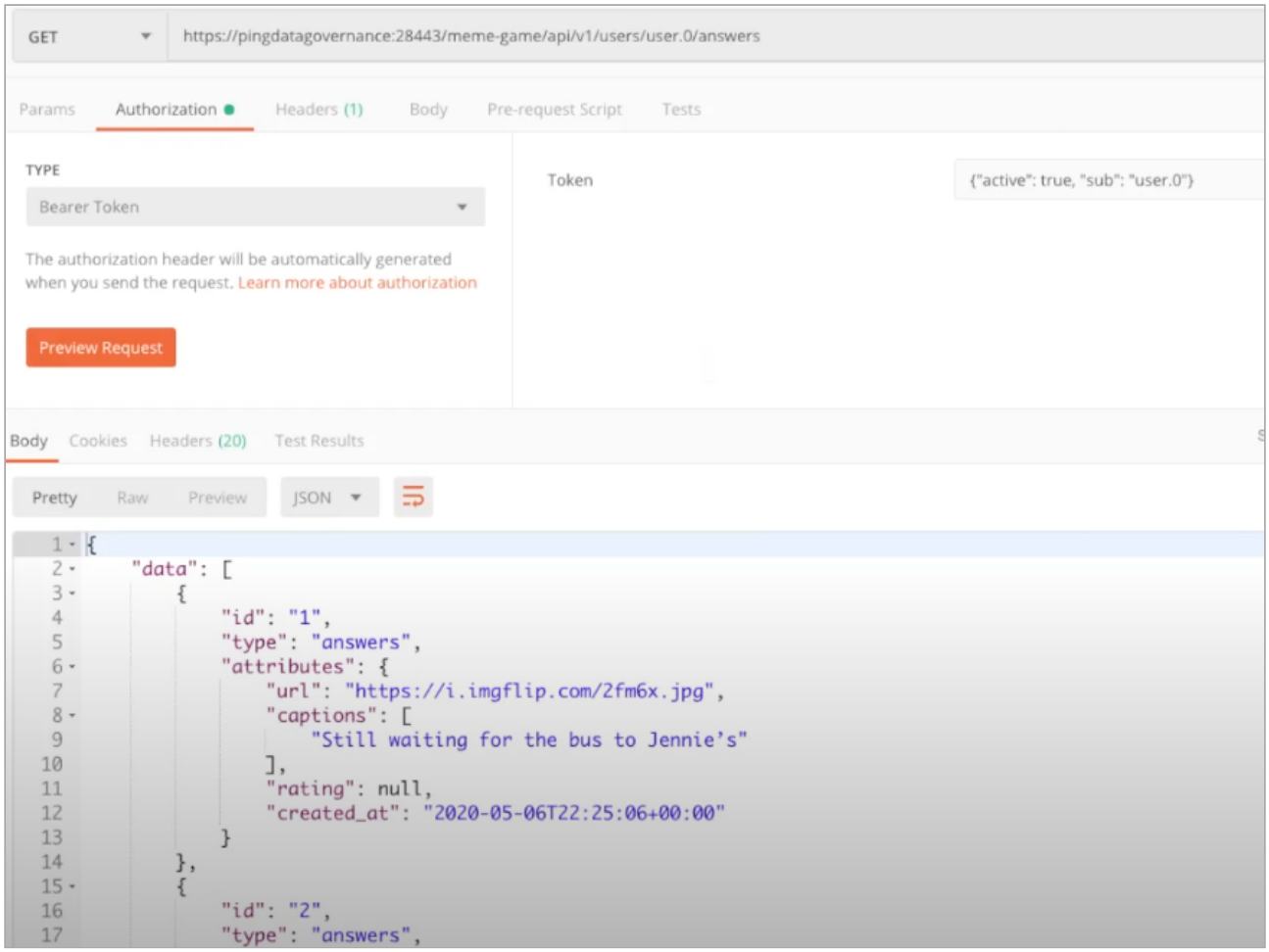


7. Click **Save changes**.

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.

Copyright © 2025 Ping Identity Corporation



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.

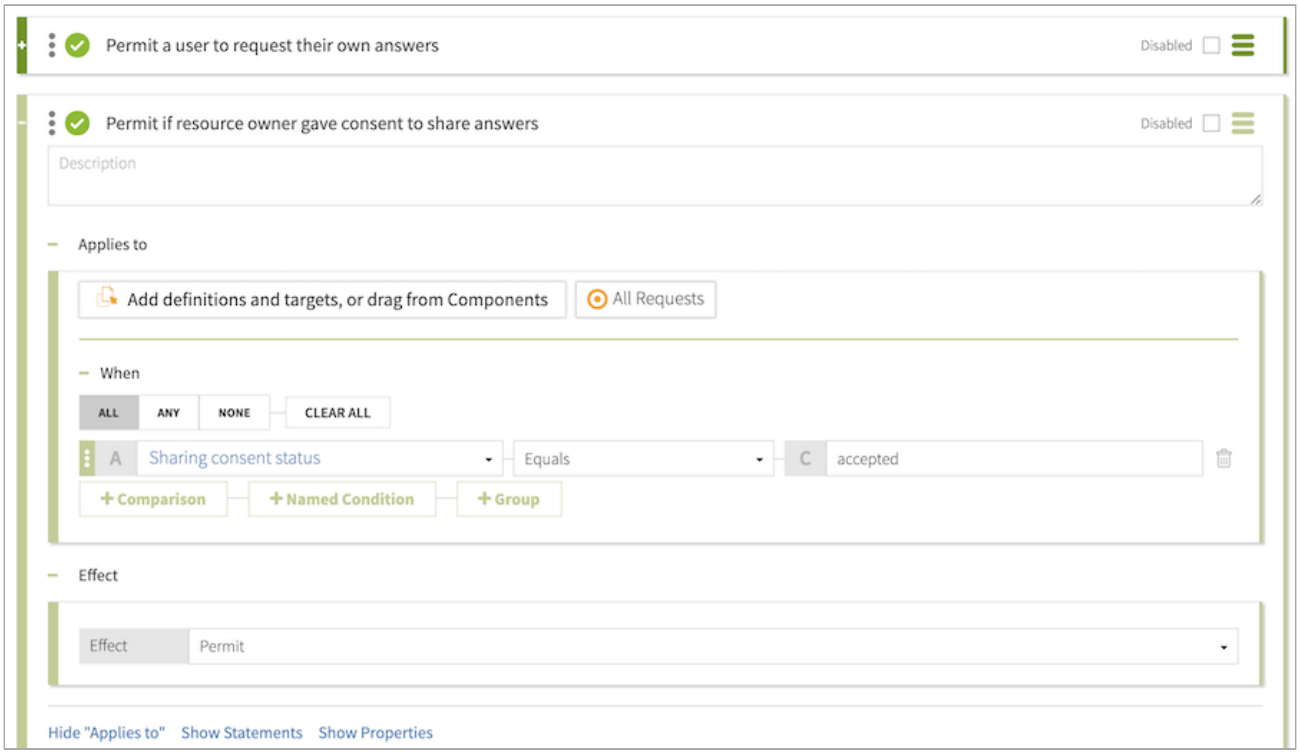
10. Add a rule to get status from a consent record.

    1. Click **+ Add Rule.**
    2. For the name, replace **Untitled** with `Permit if resource owner gave consent to share answers`.
    3. Click **+ Comparison**.
    4. From the **Select an Attribute** list, select **Sharing consent status**, which we set up in Getting consent status from the consent record.
    5. In the second field, select **Equals**.
    6. In the third field, type `accepted`.

       This value is the status to check against.

       The following image shows this rule.

Copyright © 2025 Ping Identity Corporation



7. 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. Because user.0 does not have a consent record for user.2, there is no consent status and 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).

Copyright © 2025 Ping Identity Corporation

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, 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
2. Getting the requestor identifier from the access token
3. Searching for consent granted by resource owner to requestor
4. Getting consent status from the consent record
5. What is different for this use case is the policy itself. The following topic explains how to add rules with statements to include, exclude, or modify attributes in the response.

   Creating a policy to check consent and then change the server response

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 statements based on the consent status.

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. 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**.

Copyright © 2025 Ping Identity Corporation

8. Add a rule to include attributes.
    1. Click **+ Add Rule**.
        1. For the name, replace **Untitled** with `If consent to share status is accepted then include attributes`.
        2. Specify the condition.
            1. Click **+ Comparison**.
            2. From the **Select an Attribute** list, select **Sharing consent status**, which we created in Getting consent status from the consent record.
            3. In the second field, select **Equals**.
            4. In the third field, type `accepted`.
        3. Specify the statement.
            1. Click **Show Statements**.
            2. Click **+** next to **Statements**.
            3. Click **+ Add Statement > Include Attributes**.

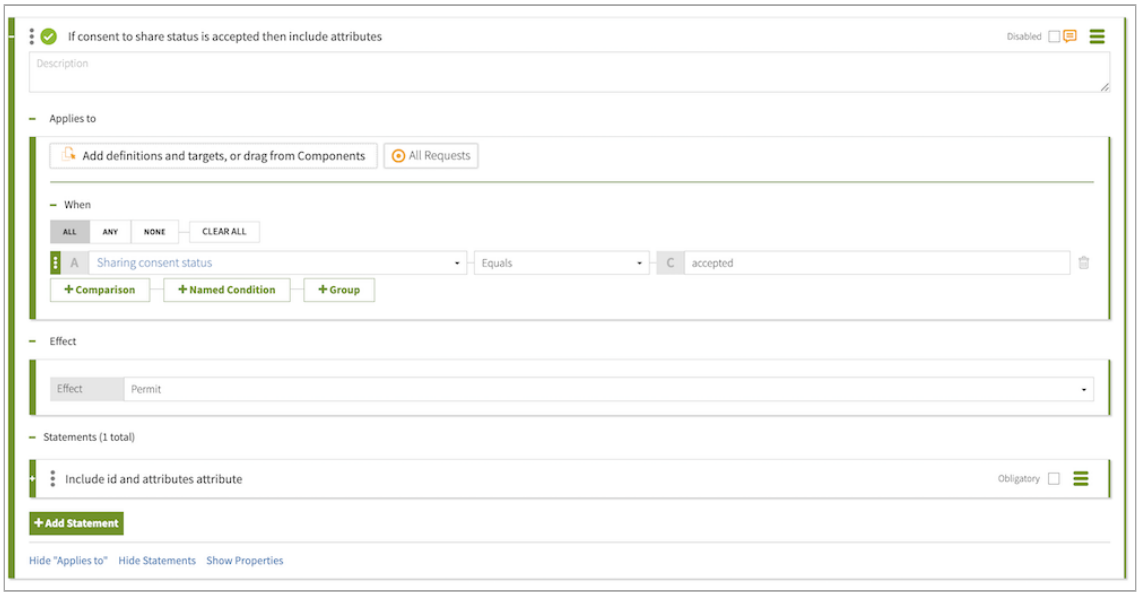
               Use this statement to be explicit about which attributes to keep, especially when you have a large set of attributes where you only need a small subset in the response.

               Learn more about this statement in Include Attributes.
            4. For the name, replace **Untitled** with `Include id and attributes attribute`.
            5. In the **Code** field, enter `include-attributes`.
            6. From the **Applies To** list, select **Permit**.
            7. 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.*"]
               ```
            8. Click **Save changes**.

               The following screen shows the rule.

Copyright © 2025 Ping Identity Corporation



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.
    1. Click **+ Add Rule**.
        1. For the name, replace **Untitled** with `If consent to share status is revoked then exclude attributes`.
        2. Specify the condition.
            1. Click **+ Comparison**.
            2. From the **Select an Attribute** list, select **Sharing consent status**, which we created in Getting consent status from the consent record.
            3. In the second field, select **Equals**.
            4. In the third field, type `revoked`.
        3. Specify the statement.
            1. Click **Show Statements**.

Copyright © 2025 Ping Identity Corporation

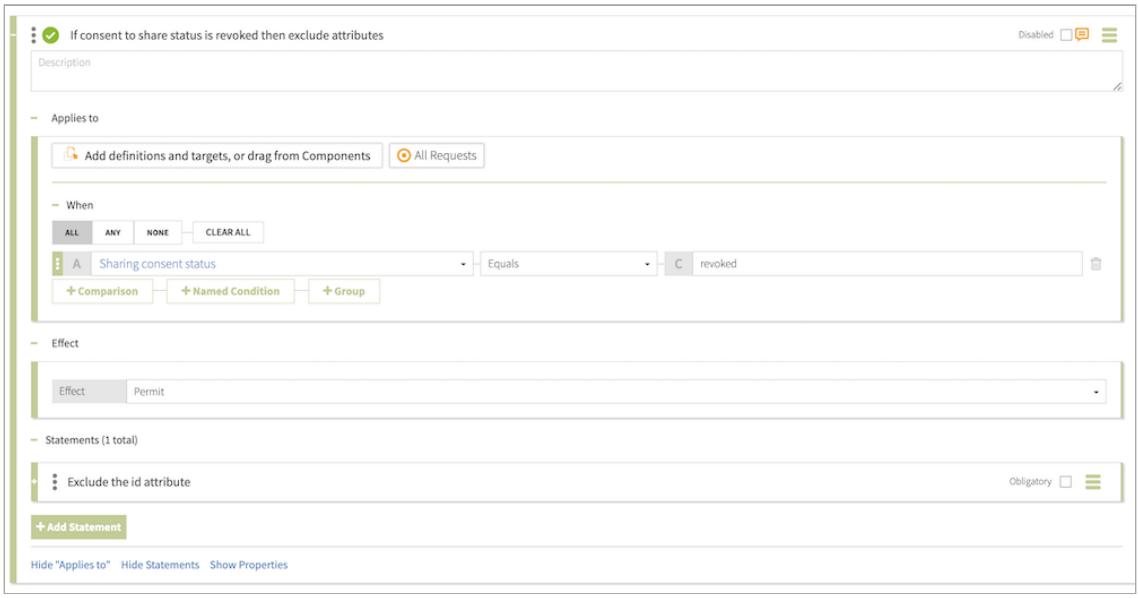
2. Click + next to **Statements**.
3. Click + **Add Statement** > **Exclude Attributes**.

   Use this statement to be explicit about which 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.

   Learn more about this statement in Exclude Attributes.
4. For the name, replace **Untitled** with `Exclude the id attribute`.
5. In the **Code** field, enter `exclude-attributes`.
6. From the **Applies To** list, select **Anything**.
7. In the **Payload** field, enter the following text to exclude the `id` attribute.

   ```
   ["data[*].id"]
   ```
8. Click **Save changes**.

   The following screen shows the rule.



   With the policy in place, trying the request again gets a response with the `id` attribute removed, as shown in the following snippet.

Copyright © 2025 Ping Identity Corporation

```
{
        "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 which policies 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.
    1. Click **+ Add Rule**.
       1. For the name, replace **Untitled** with `If consent to share status is restricted then modify attributes`.
       2. Specify the condition.
          1. Click **+ Comparison**.
          2. From the **Select an Attribute** list, select **Sharing consent status**, which we created in Getting consent status from the consent record.
          3. In the second field, select **Equals**.
          4. In the third field, type `restricted`.
       3. Specify the statement.
          1. Click **Show Statements**.
          2. Click **+** next to **Statements**.
          3. Click **+ Add Statement > Modify Attributes**.

             Use this statement 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.

             Learn more about this statement in Modify Attributes.
          4. For the name, replace **Untitled** with `Modify all the values in attributes`.
          5. In the **Code** field, enter `modify-attributes`.
          6. From the **Applies To** list, select **Permit**.

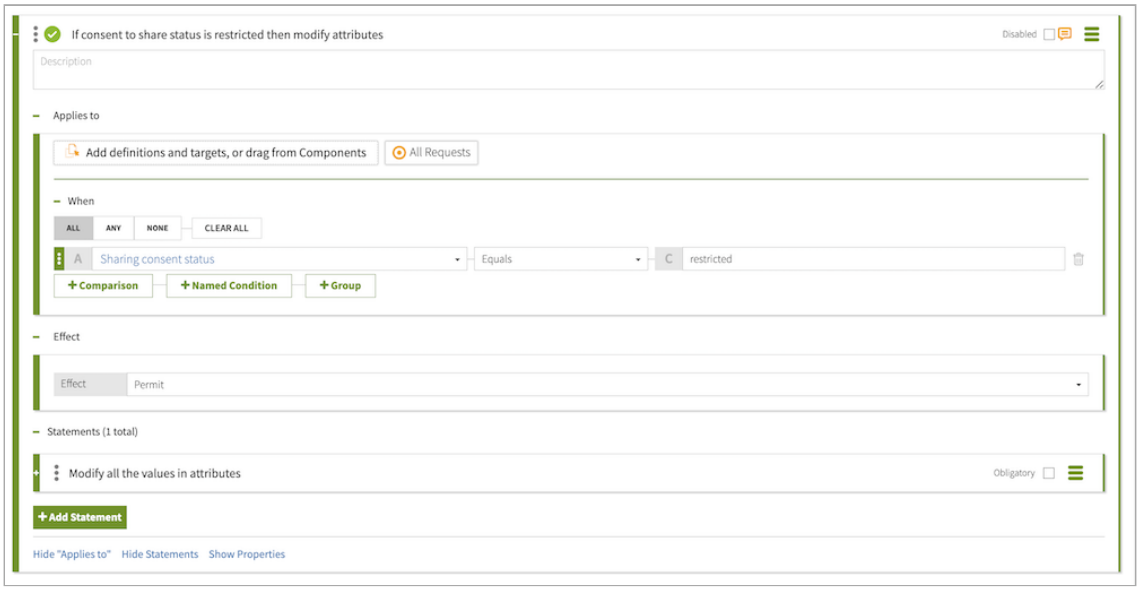
Copyright © 2025 Ping Identity Corporation

7. In the **Payload** field, enter the following text to replace all values in the `attributes` attribute with three dashes.

   ```
   \{"data[].attributes.":"---"}
   ```

8. Click **Save changes**.

   The following screen shows the rule.



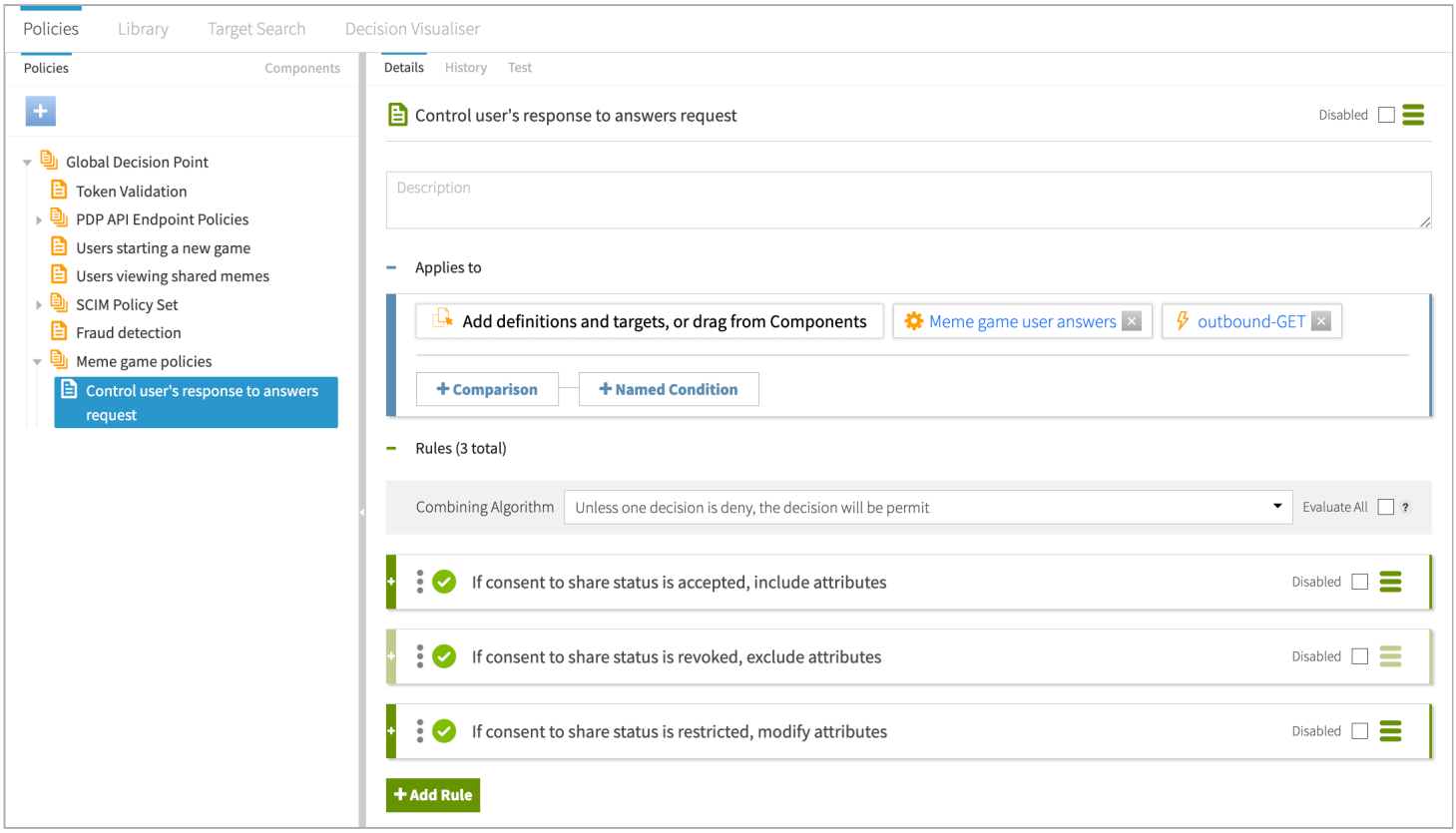
   With the policy in place, trying the request now gets a response with the `id` and `type` attributes unchanged but all the `attributes` values changed to dashes, as shown in the following snippet.

```
{
        "data": [
      {
        "id": "168",
        "type": "answers",
        "attributes": {
          "url": "---",
          "captions": "---",
          "rating": "---",
          "created_at": "---"
        }
      },
```

Result

The following image shows what the policy applies to and the three rules:

Copyright © 2025 Ping Identity Corporation



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.

For more information about actions, see SCIM policy requests.

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

Copyright © 2025 Ping Identity Corporation

- 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
2. Creating a policy to permit or deny the creation of resources
3. Creating a policy to control the set of actions for a specific resource
4. Creating a policy to restrict the ability to delete based on resource type
5. Creating a policy to dynamically modify a resource based on the SCIM resource type

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

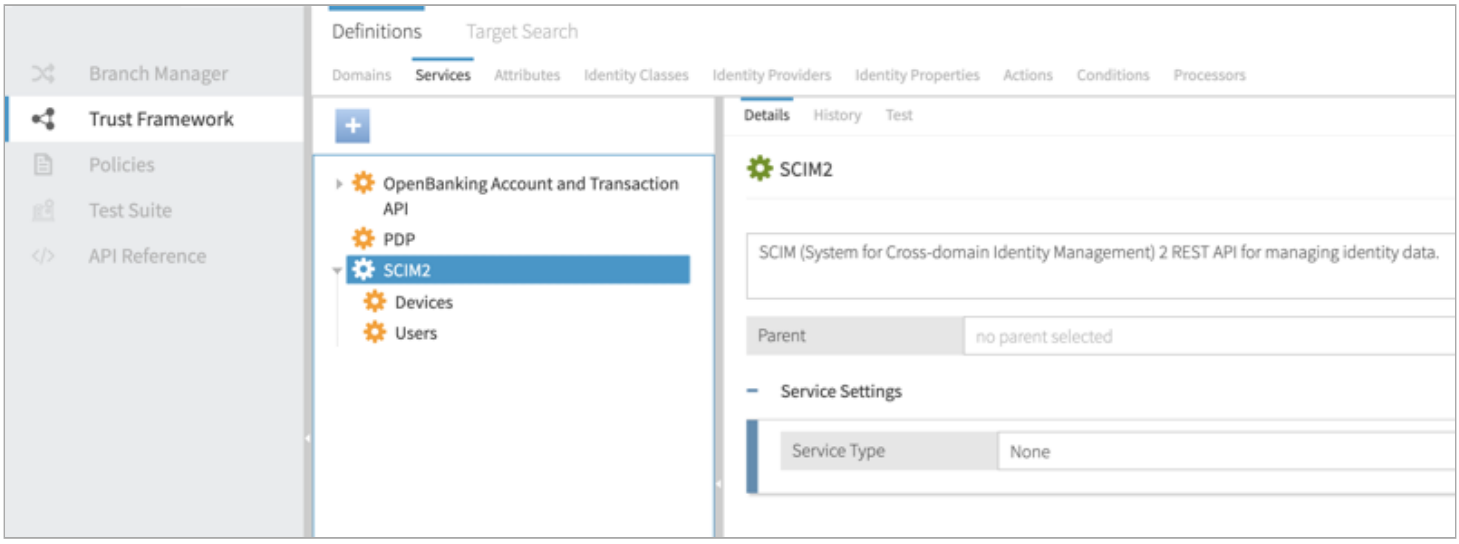
The PingAuthorize Policy Editor provides a SCIM2 service in the Trust Framework. This service is for the SCIM2 REST API and does not reference resource types. This task creates two services: Users and Devices.

Steps

1. Sign on to the Policy Editor.
2. Create the Users and Devices services.
    1. Go to **Trust Framework** and click **Services**.
    2. Click the **SCIM2** service so the service we create is listed under **SCIM2**.
    3. From the **+** menu, select **Add new Service**.
    4. For the name, replace **Untitled** with `Users`.
    5. Click **Save changes**.
    6. Click the **SCIM2** service again.
    7. From the **+** menu, select **Add new Service**.
    8. For the name, replace **Untitled** with `Devices`.
    9. Click **Save changes**.

With the services defined, you should have a screen similar to the following one.

Copyright © 2025 Ping Identity Corporation



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**.

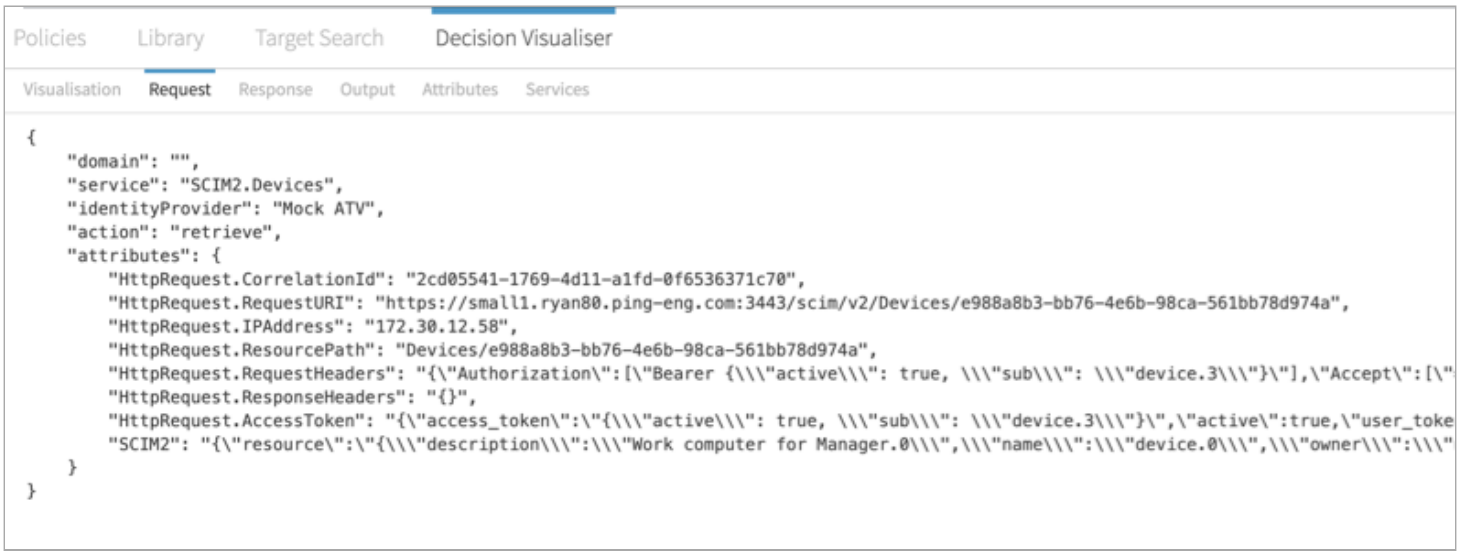
> **Note**
>
> The `SCIM2.resource` attribute is only available when the SCIM resource exists. For example, the search and create actions do not have this attribute. However, the search action does have a policy request with a retrieve action that does have the attribute.

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, and 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.



Copyright © 2025 Ping Identity Corporation

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 a retrieve action. For more information about actions, see SCIM policy requests.

- `RequestURI` line

  Verify that the endpoint belongs to the expected service.

- `SCIM2` line

  Scroll right to verify that the `resourceType` is as expected.

Creating a policy to permit or deny the creation of resources

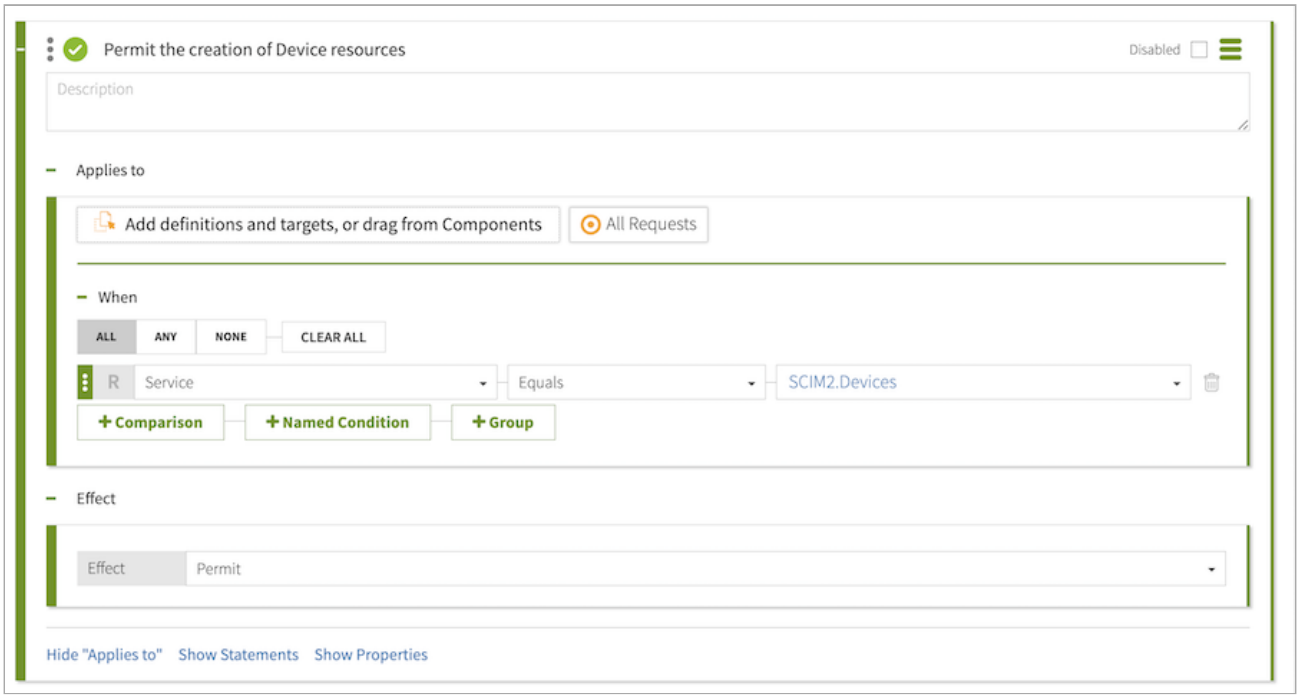
This policy allows the creation of one resource type but not another. In particular, the policy focuses on the create action and then allows the creation of Device resources but denies the creation of User resources.

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.
    1. Click + **Add Rule.**
    2. For the name, replace **Untitled** with `Permit the creation of Device resources`.
    3. Click + **Comparison**.
    4. In the first field, click the **A** to toggle to an **R**, and from that field's drop-down list, select **Service**.
    5. In the second field, select **Equals**.
    6. In the third field, select the **SCIM2.Devices** service.
    7. Click **Save changes**.

       You should have a screen similar to the following one for the policy and this rule.

Copyright © 2025 Ping Identity Corporation



8. Add a rule to deny the creation of User resources.
    1. Click **+ Add Rule.**
    2. For the name, replace **Untitled** with `Deny the creation of User resources`.
    3. Set **Effect** to **Deny**.
    4. Click **+ Comparison**.
    5. In the first field, click the **A** to toggle to an **R**, and from that field's drop-down list, select **Service**.
    6. In the second field, select **Equals**.
    7. In the third field, select the **SCIM2.Users** service.
    8. Add statements to provide a custom message.
        1. Within the rule, click **Show Statements**.
        2. Click **+** next to **Statements**.
        3. Click **+ Add Statement → Denied Reason**.
        4. For the name, specify `denied-reason`.
        5. Set **Applies To** to **Deny**.
        6. In the **Payload** field:
            - Remove

              `Example:`
            - Change

Copyright © 2025 Ping Identity Corporation

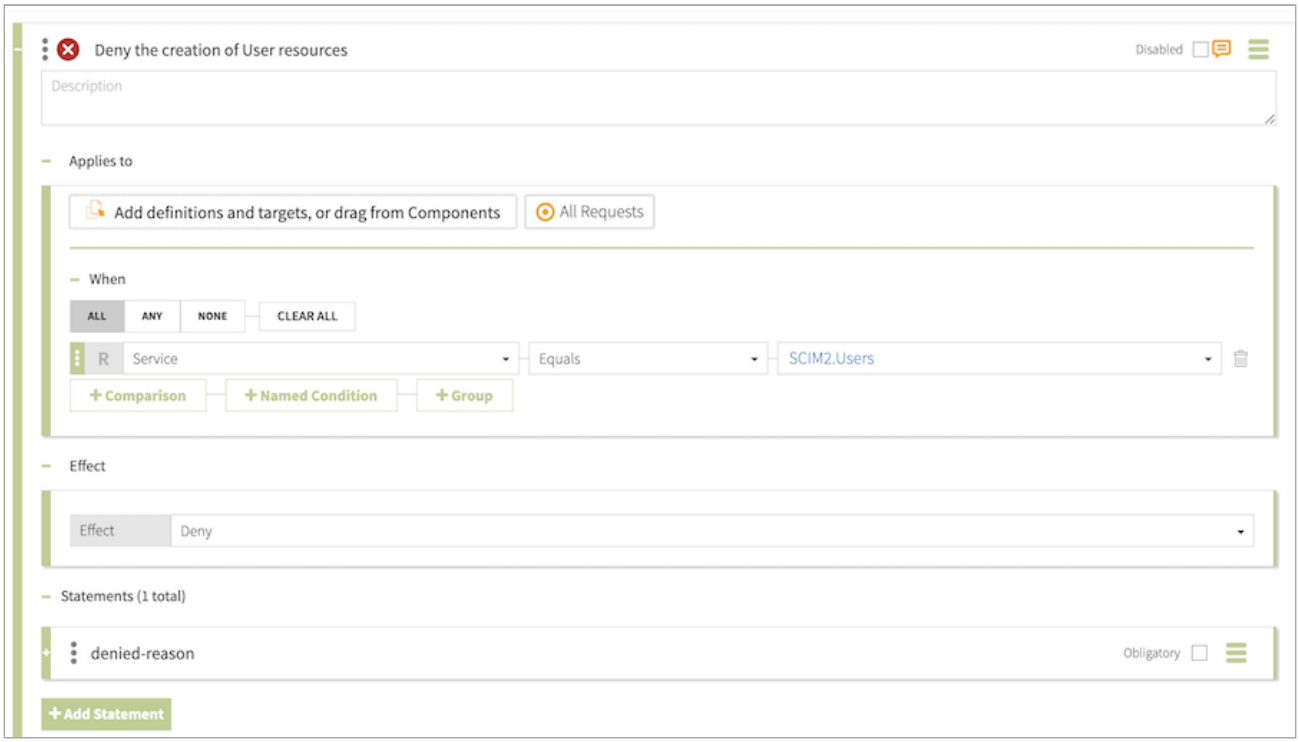
`Human-readable error message`

to

`System has restricted the ability to create User resources`

9. 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

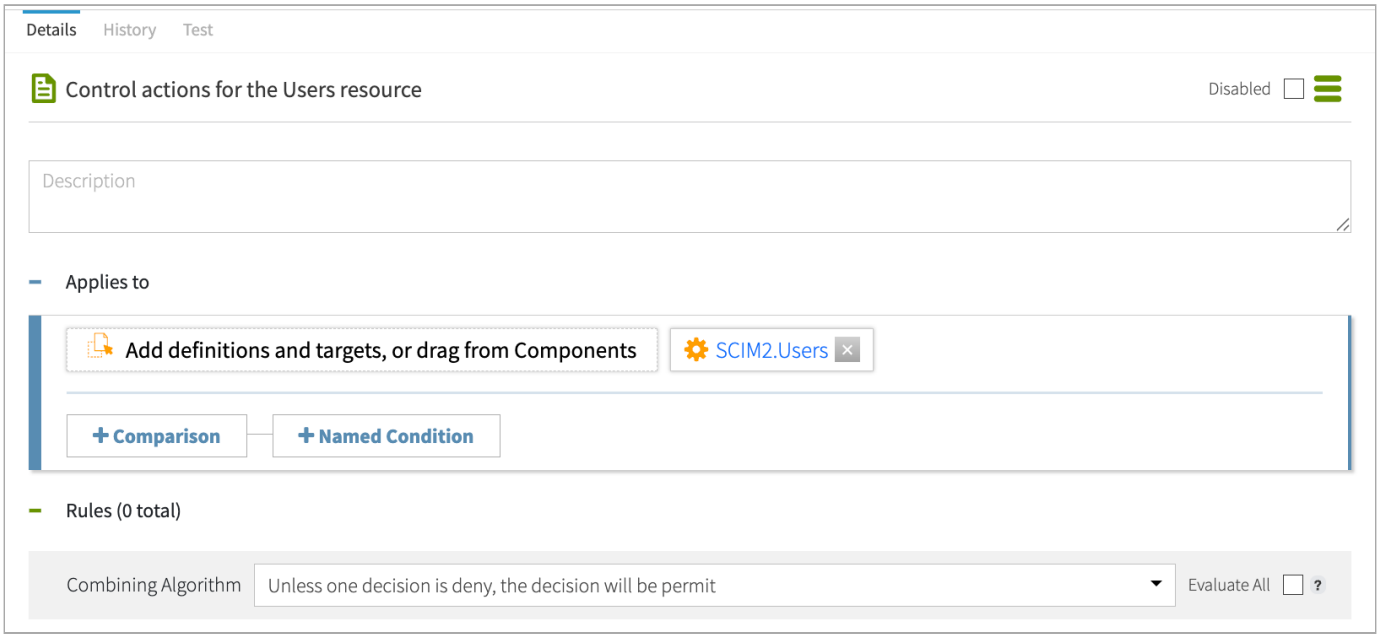
For a given resource, control the outcomes (deny or permit) of actions on the resource. In particular, the policy focuses on the Users resource and then denies deletes but permits retrieves.

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.

Copyright © 2025 Ping Identity Corporation



7. Add a rule to deny the deletion of User resources.
    1. Click **+ Add Rule**.
    2. For the name, replace **Untitled** with `Action: delete`.
    3. Set **Effect** to **Deny**.
    4. Click **+ Comparison**.
    5. In the first field, click the **A** to toggle to an **R**, and from that field's drop-down list, select **Action**.
    6. In the second field, select **Equals**.
    7. In the third field, select the **delete** action.
    8. Add a statement to provide a custom message.
        1. Within the rule, click **Show Statements**.
        2. Click **+** next to **Statements**.
        3. Click **+ Add Statement → 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`

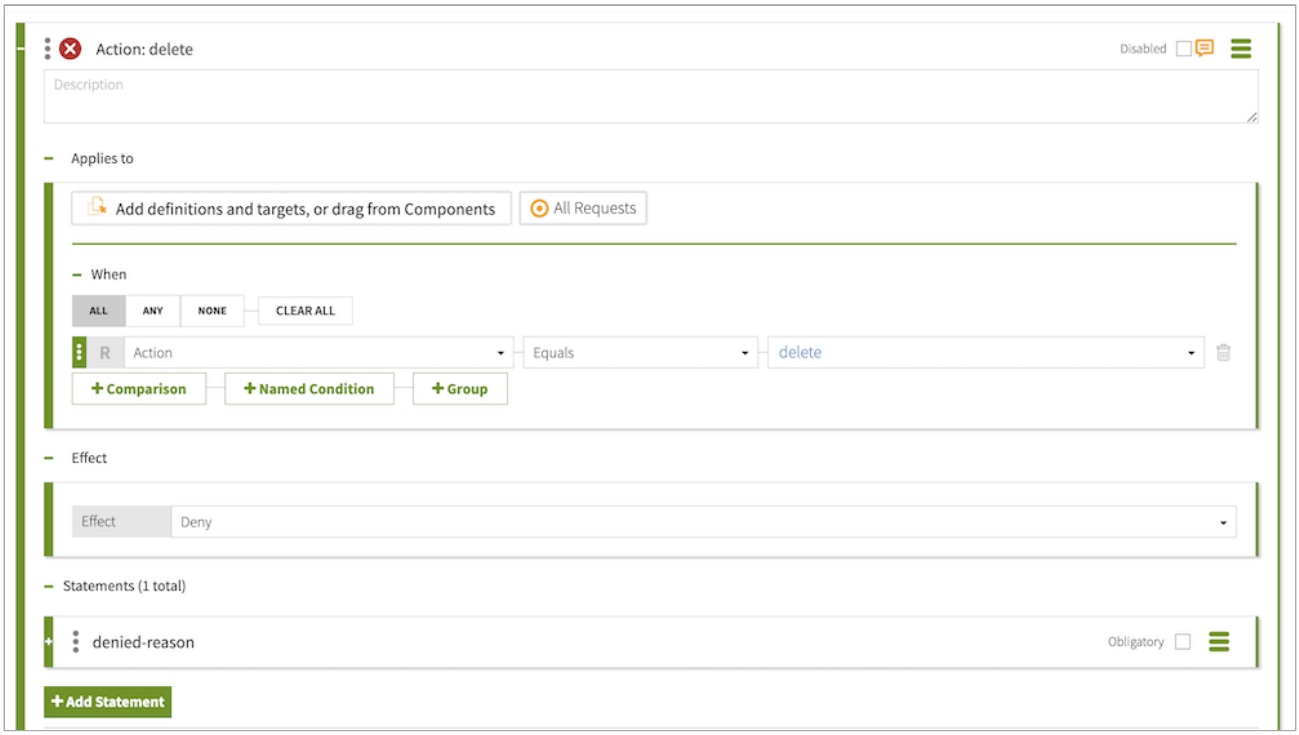
Copyright © 2025 Ping Identity Corporation

to

```
System has restricted the ability to delete User resources
```

9. Click **Save changes**.

   Your rule should be similar to the following one.

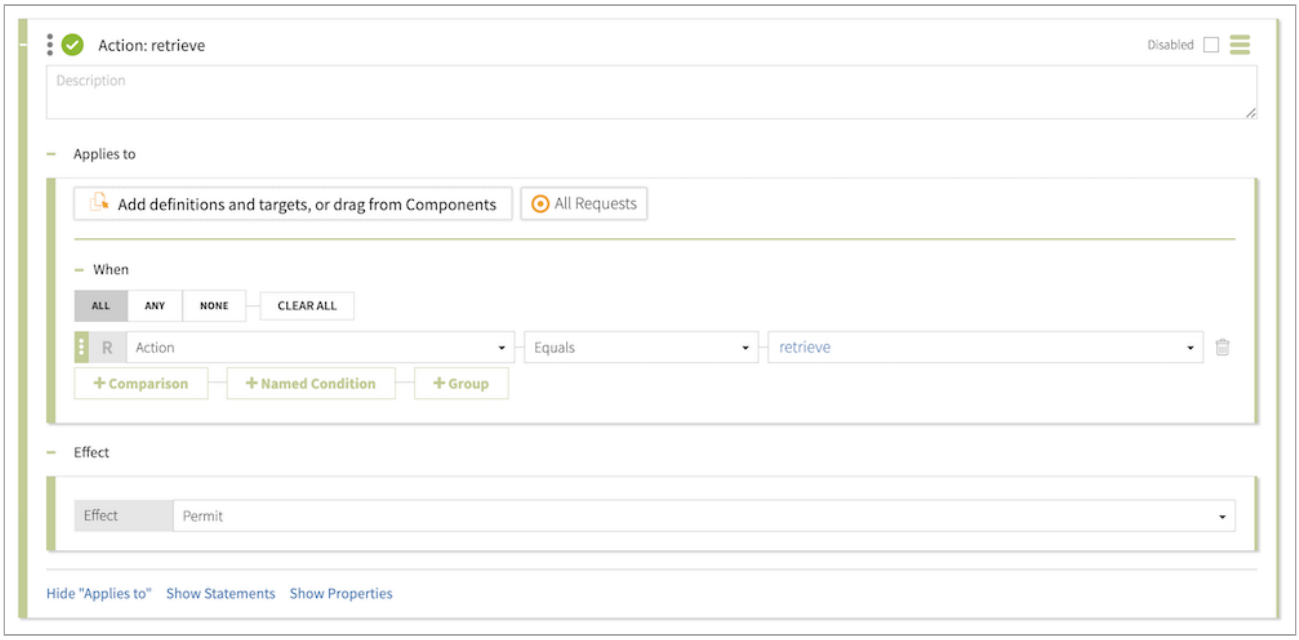


8. Add a rule to permit the retrieval of User resources.

   1. Click **+ Add Rule.**
   2. For the name, replace **Untitled** with `Action: retrieve`.
   3. Set **Effect** to **Permit**.
   4. Click **+ Comparison**.
   5. In the first field, click the **A** to toggle to an **R**, and from that field's drop-down list, select **Action**.
   6. In the second field, select **Equals**.
   7. In the third field, select the **retrieve** action.
   8. Click **Save changes**.

      Your rule should be similar to the following one.

Copyright © 2025 Ping Identity Corporation



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

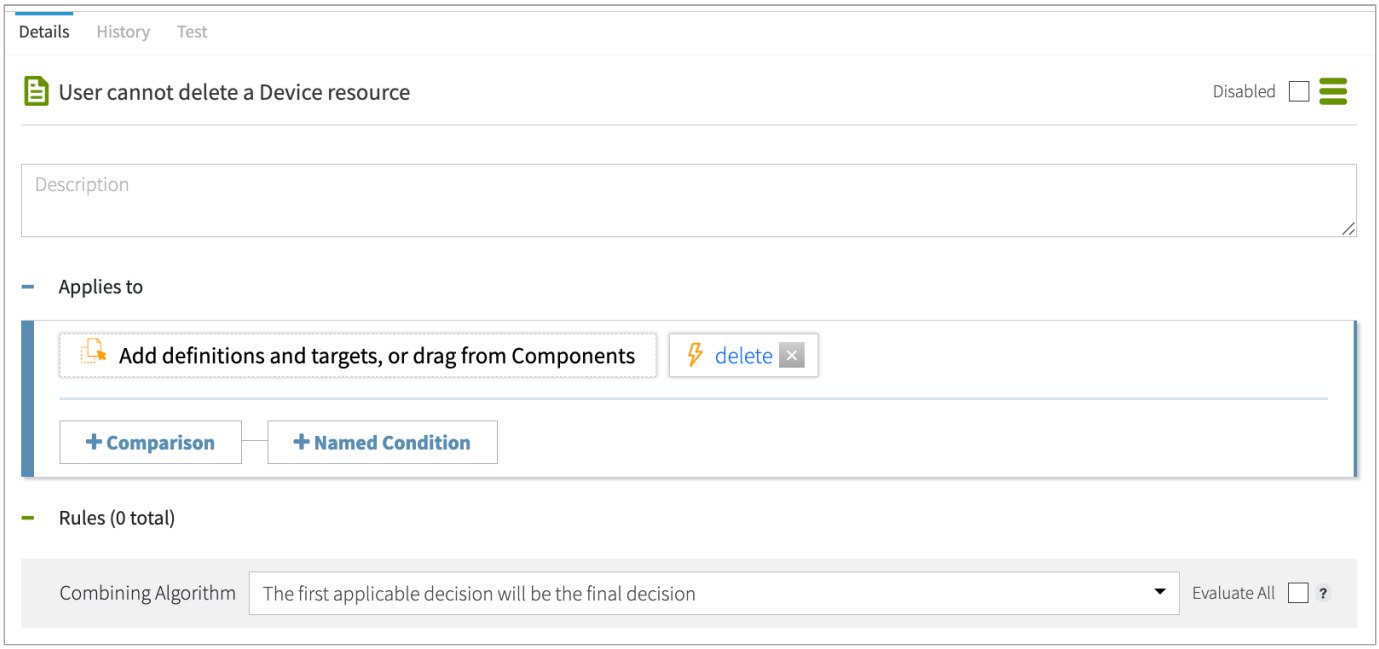
For a given resource type, restrict the ability to delete. In particular, the policy focuses on the delete action and then denies the action when the resource type is Devices.

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.

Copyright © 2025 Ping Identity Corporation



7. Add a rule to deny the deletion of Device resources.
    1. Click **+ Add Rule.**
    2. For the name, replace **Untitled** with `If the SCIM resource type is Device, then deny`.
    3. Set **Effect** to **Deny**.
    4. Click **+ Comparison**.
    5. In the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
    6. In the second field, select **Equals**.
    7. In the third field, specify `Devices` as the constant.
    8. Add a statement to provide a custom message.
        1. Within the rule, click **Show Statements**.
        2. Click **+** next to **Statements**.
        3. Click **+ Add Statement → 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`

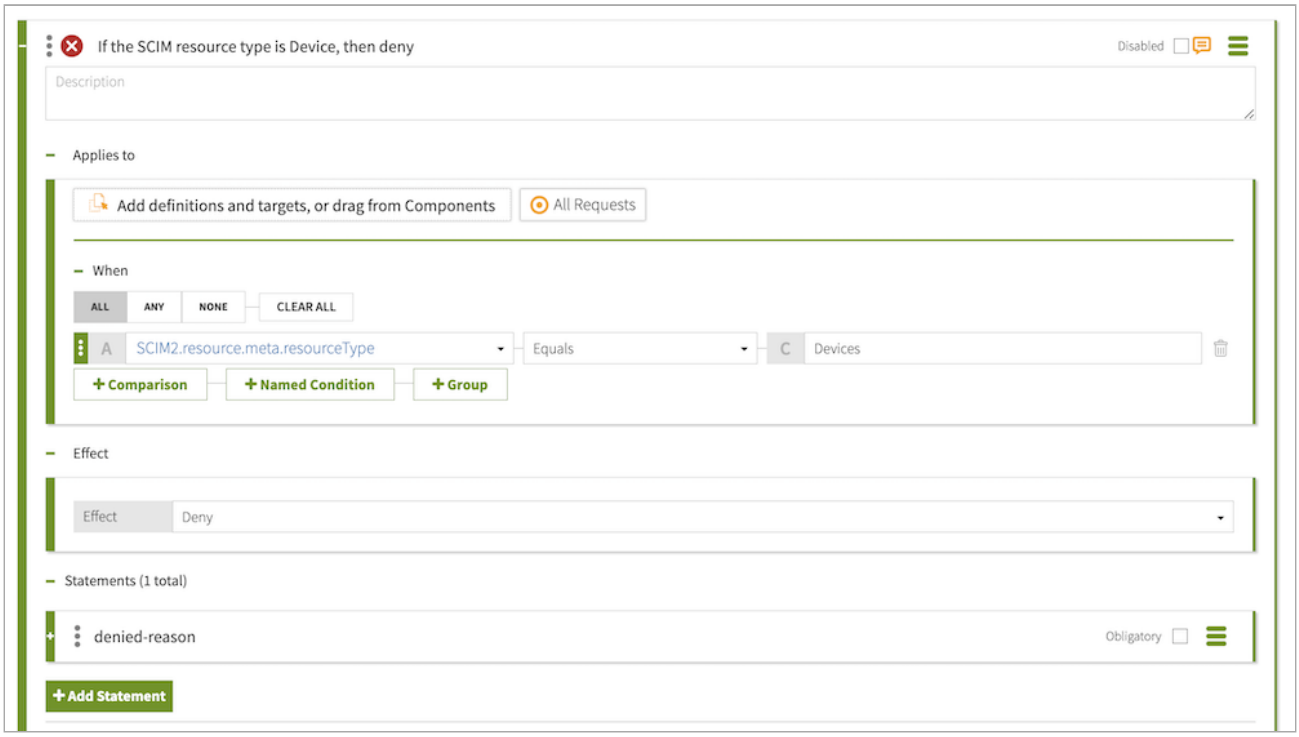
Copyright © 2025 Ping Identity Corporation

to

```
System has restricted the ability to delete Device resources
```

9. 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

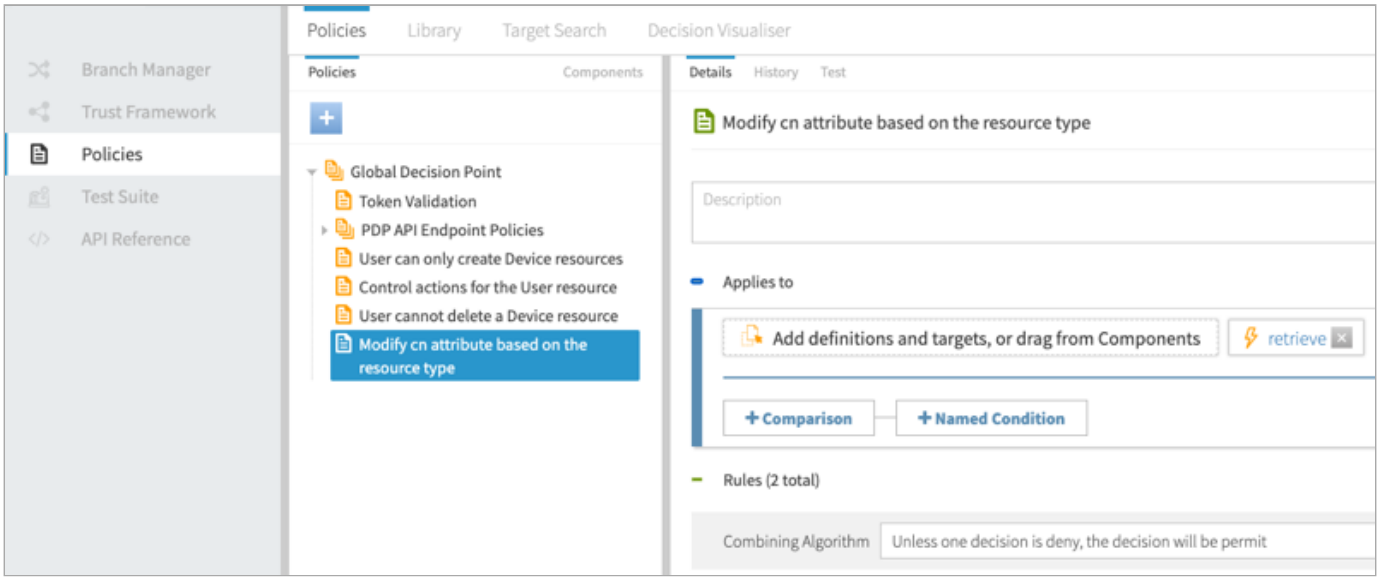
Given an attribute defined in multiple resource types, modify the attribute differently depending on the resource type. In particular, this policy focuses on the retrieve action and changes the `cn` attribute to one value for the Users resource type and to another value for the Devices resource type.

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.

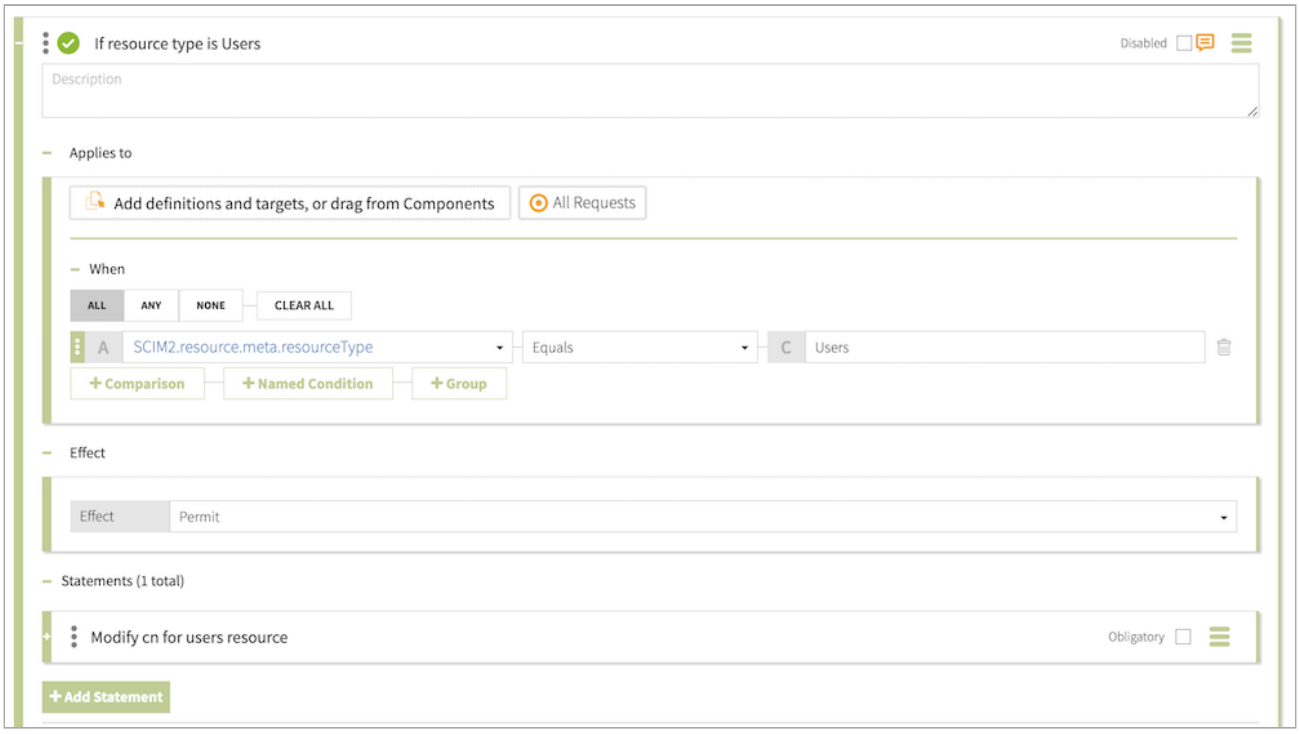
Copyright © 2025 Ping Identity Corporation



7. Add a rule for the Users resource.
    1. Click **+ Add Rule.**
    2. For the name, replace **Untitled** with `If resource type is Users`.
    3. Click **+ Comparison**.
    4. From the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
    5. In the second field, select **Equals**.
    6. In the third field, specify `Users` as the constant.
    7. Add statements to modify attributes.
        1. Within the rule, click **Show Statements**.
        2. Click **+** next to **Statements**.
        3. Click **+ Add Statement → 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"}`.
    8. Click **Save changes**.

    Your rule should be similar to the following one.

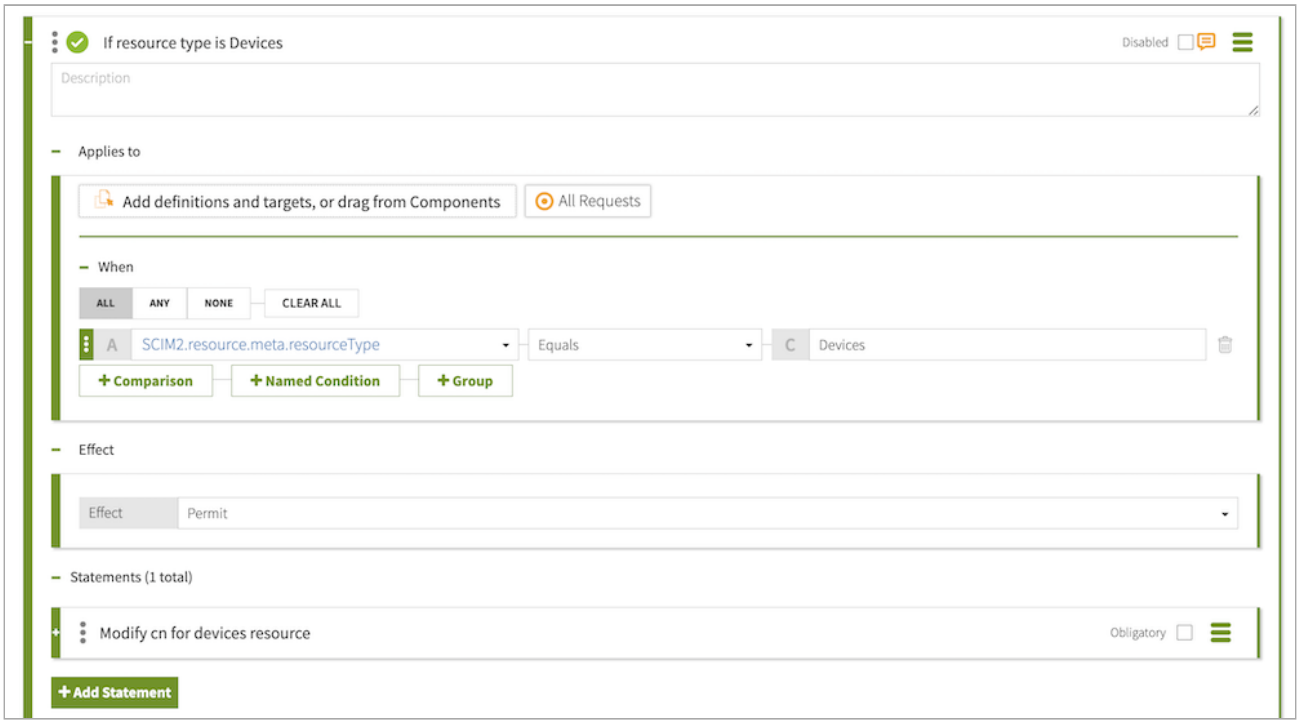
Copyright © 2025 Ping Identity Corporation



8. Add a rule for the Devices resource.
    1. Click **+ Add Rule.**
    2. For the name, replace **Untitled** with `If resource type is Devices`.
    3. Click **+ Comparison**.
    4. From the **Select an Attribute** list, select the `SCIM2.resource.meta.resourceType` attribute.
    5. In the second field, select **Equals**.
    6. In the third field, specify `Devices` as the constant.
    7. Add statements to modify attributes.
        1. Within the rule, click **Show Statements**.
        2. Click **+** next to **Statements**.
        3. Click **+ Add Statement → 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"}`.
    8. Click **Save changes**.

    Your rule should be similar to the following one.

Copyright © 2025 Ping Identity Corporation



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 statement and Prohibit Attributes statement are no longer supported. If you have policies that use those statements, 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 delete 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:
    1. Select the box next to **Default value** and specify square brackets with no space between them (`[]`) as the value.
    2. Set **Type** to **Collection**.
7. Click **Save changes**.

Copyright © 2025 Ping Identity Corporation

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, and then add a rule with the **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.
   1. Go to **Trust Framework** and then **Attributes**.
   2. From the **+** menu, select **Add new Attribute**.
   3. For the name, replace **Untitled** with `allowlistAttributes`.
   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.
   5. Click **+ Add Resolver** and set the **Resolver type** to **Constant**.
   6. 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]`.
   1. In the **Value Settings** section, set **Type** to **Collection**.
   2. Click **Save changes**.
2. Modify or create a policy to use that constant collection.
   1. Go to **Policies**.

Copyright © 2025 Ping Identity Corporation

2. Select a policy or create a new one.
3. 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.

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.

Copyright © 2025 Ping Identity Corporation

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 that avoids 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**.

Copyright © 2025 Ping Identity Corporation

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.

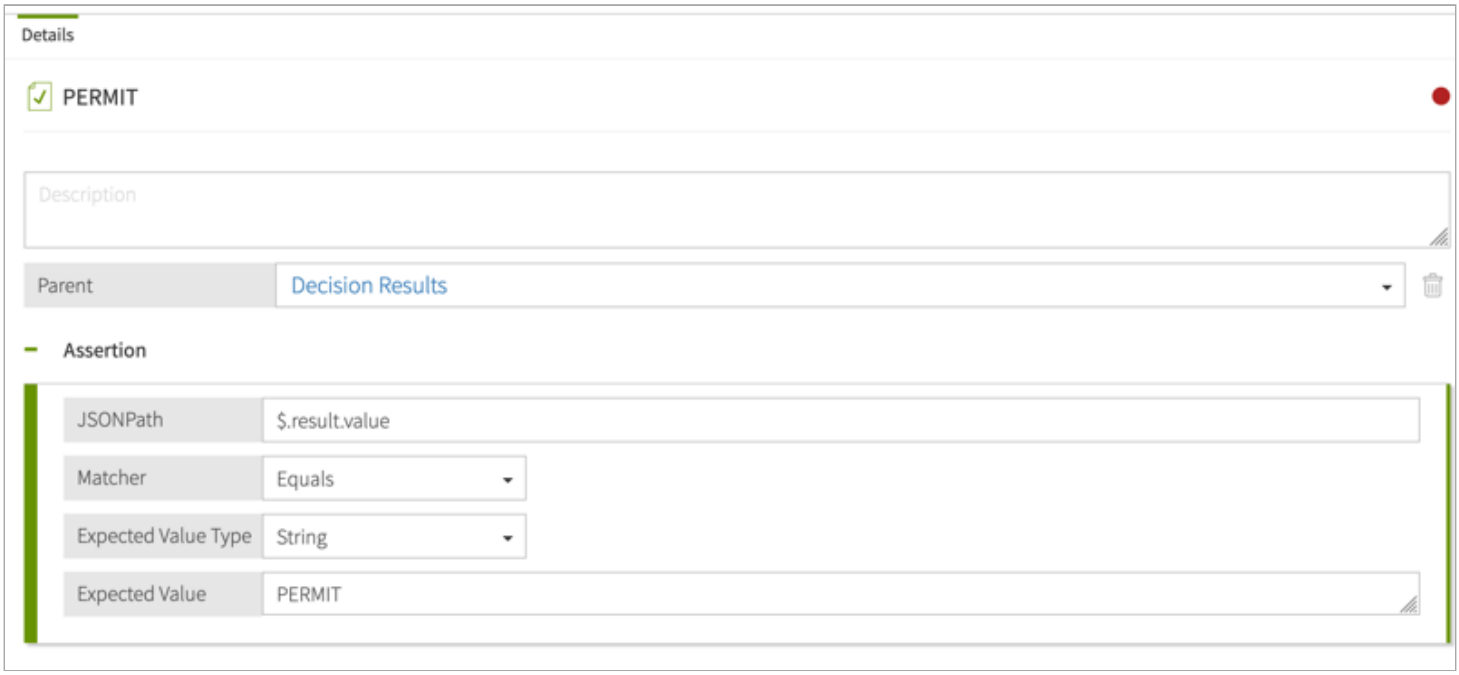
- Creating them inline when you define a test on the **Tests** tab.

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 the result value equals `PERMIT`:

Copyright © 2025 Ping Identity Corporation



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 your 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 now 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.

Management REST API documentation

The PingAuthorize Policy Editor provides a set of REST APIs for managing policies, snapshots, and deployment packages.

Documentation for these APIs will be made available at a future date.

Copyright © 2025 Ping Identity Corporation

Troubleshooting PingAuthorize Server

You can capture diagnostic data to help troubleshoot issues with PingAuthorize Server or a supporting component, such as the Java Virtual Machine (JVM), the operating system, or the hardware.

With this data, you can troubleshoot the problem quickly to determine the underlying cause and the best course of action to resolve it.

Learn about troubleshooting decision requests and responses in the following:

- Visualizing a policy decision response
- Policy Decision Logger
- Configuring the Decision Response View

Working with the collect-support-data tool

If a problem occurs with PingAuthorize Server, you should first run the **`collect-support-data`** tool in the server's `bin` directory.

The server provides detailed insights into its current state, including any processing issues. The **`collect-support-data`** tool compiles relevant support files and diagnostic data, such as outputs from the **`jps`**, **`jstack`**, and **`jstat`** utilities, into a `.zip` archive for administrators to share with their authorized support provider.

Although the **`collect-support-data`** tool tries to collect the same data across all systems for the target PingAuthorize Server, the resulting data might vary between operating systems. The collected data includes the configuration directory, summaries and snippets from the `logs` directory, monitor entry status, and a list of all files in the server root.

By default, the **`collect-support-data`** tool excludes log files that might contain sensitive customer information, including the debug logs described in Enable detailed logging. If you are using test data that doesn't include sensitive information, send the following log files along with **`collect-support-data`**'s compressed output file:

- `PingAuthorize/logs/policy-decision.log`
- `PingAuthorize/logs/debug-trace.log`
- `PingAuthorize/logs/debug.log`
- `PingAuthorize/logs/policy-query.log`

To prevent the support `.zip` archive from exceeding e-mail attachment size limits, the **`collect-support-data`** tool might only archive portions of certain log files.

Running the collect-support-data tool

Steps

1. Run the **`collect-support-data`** tool.

   Make sure to include the host, port number, bind DN, and bind password.

   Example:

Copyright © 2025 Ping Identity Corporation

```
<PingAuthorize>/bin/collect-support-data \
  --hostname <host> \
  --port <port> \
  --bindDN "cn=<dn>" --bindPassword <password>
```

2. Email the generated `.zip` archive to your authorized support provider.

Invoking the collect-support-data tool as a recurring task

To automatically invoke **`collect-support-data`** on a regular basis, create and configure a recurring task.

Admin console

Use the administrative console
Steps

1. Go to **System > Recurring Tasks**.
2. In the **New Recurring Task** list, select **Collect Support Data Recurring Task**.
3. In the **Name** field, enter a name for the recurring task.
4. In the **Output Directory** field, enter the path of the directory in which support data archive files will be placed.
5. Do at least one of the following:
   - In the **Retain Previous Support Data Archive Count** field, enter the number of previous support data archives that PingAuthorize Server should preserve after generating a new archive.
   - In the **Retain Previous Support Data Archive Age** field, enter the minimum age of previous support data archives that PingAuthorize Server should preserve after generating a new archive. Values for this property should consist of an integer followed by a time unit. For example, a value of `1w` specifies that PingAuthorize Server should preserve support data archives for one week.
6. Configure the rest of the recurring task to meet your business needs.
7. Click **Save**.

Copyright © 2025 Ping Identity Corporation

dsconfig

Use `dsconfig`

Steps

- Run `dsconfig` with the `create-recurring-task` subcommand and the following options and parameters:

```
<PingAuthorize>/bin/dsconfig create-recurring-task \
  --task-name <your-task-name> \
  --type collect-support-data \
  --set output-directory:<your-directory-path> \
  --set retain-previous-support-data-archive-count:<count>
```

Important

You must include at least one of the following parameters in your command:

- `retain-previous-support-data-archive-count` : Specifies the number of previous support data archives that PingAuthorize Server should preserve after generating a new archive.
- `retain-previous-support-data-archive-age` : Specifies the minimum age of previous support data archives that PingAuthorize Server should preserve after generating a new archive. Values for this property should consist of an integer followed by a time unit. For example, a value of `1w` specifies that PingAuthorize Server should preserve support data archives for one week.

Server commands used in the collect-support-data tool

The following presents a summary of the data collectors that the `collect-support-data` tool archives in `.zip` format. If an error occurs during processing, you can re-run the specific data collector command and send the results to your authorized support provider.

Data Collector	Description
`status`	Run `status -F` to show the full version information of PingAuthorize Server (Unix, Windows).
`server-state`	Run `server-state` to show the current state of the PingAuthorize Server process (Unix, Windows).

JDK commands used in the collect-support-data tool

Data Collector	Description
`jps`	Java Virtual Machine Process status tool. Reports information on the JVM (Linux, Windows, Mac OS).

Copyright © 2025 Ping Identity Corporation

Data Collector	Description
`jstack`	Java Virtual Machine Stack Trace. Prints the stack traces of threads for the Java process (Linux, Windows, Mac OS).
`jstat`	Java Virtual Machine Statistics Monitoring Tool. Displays performance statistics for the JVM (Linux, Windows, Mac OS).
`jinfo`	Displays the Java configuration information for the Java process (Linux, Windows, Mac OS).

Linux commands used in the collect-support-data tool

Data Collector	Description
`tail`	Displays the last few lines of a file. Tails the `/var/logs/messages` directory.
`uname`	Prints system, machine, and operating system information.
`ps`	Prints a snapshot of the current active processes.
`df`	Prints the amount of available disk space for file systems in 1024-byte units.
`cat`	Concatenates the following files and prints to standard output: • `/proc/cpuinfo` • `/proc/meminfo` • `/etc/hosts` • `/etc/nsswitch.conf` • `/etc/resolv.conf`
`netstat`	Prints the state of network interfaces, protocols, and the kernel routing table.
`ifconfig`	Prints information on all interfaces.
`uptime`	Prints the time the server has been up and active.
`dmesg`	Prints the message buffer of the kernel.
`vmstat`	Prints information about virtual memory statistics.
`iostat`	Prints disk I/O and CPU utilization information.
`mpstat`	Prints performance statistics for all logical processors.
`pstack`	Prints an execution stack trace on an active processed specified by the pid.
`top`	Prints a list of active processes and how much CPU and memory each process is using.

Copyright © 2025 Ping Identity Corporation

MacOS commands used in the collect-support-data tool

Data Collector	Description
`uname`	Prints system, machine, and operating system information.
`uptime`	Prints the time the server has been up and active.
`ps`	Prints a snapshot of the current active processes.
`system_profiler`	Prints system hardware and software configuration.
`vm_stat`	Prints machine virtual memory statistics.
`tail`	Displays the last few lines of a file. Tails the `/var/log/system.log` directory.
`netstat`	Prints the state of network interfaces, protocols, and the kernel routing table.
`ifconfig`	Prints information on all interfaces.
`df`	Prints the amount of available disk space for file systems in 1024-byte units.
`sample`	Profiles a process during an interval.

Available tool options

The following options are available for the `collect-support-data` tool:

`--noLdap`

Specifies that no information should be collected over LDAP. Use this option only if the server is completely unresponsive or won't start, and only as a last resort.

`--pid` *<pid>*

Specifies the ID of an additional process from which information should be collected. This is a useful option for troubleshooting external server tools, and you can specify it multiple times for each external server.

`--sequential`

Use this option to troubleshoot `Out of Memory` errors. By default, the `collect-support-data` tool collects data in parallel to minimize the collection time necessary for some analysis utilities. This option specifies that data collection should be run sequentially rather than in parallel. Specifying this option reduces the initial memory footprint of the `collect-support-data` tool but increases the time required for completion.

`--reportCount` *<count>*

Specifies the number of reports generated for commands that support sampling (for example, `vmstat`, `iostat`, or `mpstat`). If you specify a value of `0`, no reports are generated for these commands. If you don't provide this option, the number of reports generated defaults to `10`.

Copyright © 2025 Ping Identity Corporation

`--reportInterval` *<interval>*

Specifies the number of seconds between reports for commands that support sampling (for example, **`mpstat`**). You must specify a value greater than `0` for this option. If you don't provide this option, the number of seconds between reports defaults to `1`.

`--maxJstacks` *<number>*

Specifies the number of jstack samples to collect. If you don't provide this option, the number of samples collected defaults to `10`.

`--collectExpensiveData`

Specifies that data for expensive or long-running processes should be collected. To prevent negative impact on server performance, data for these processes are not collected by default.

`--comment` *<comment>*

Allows provisioning of additional information about the collected data set. The comment you provide will be added to the generated archive as a `README` file.

`--includeBinaryFiles`

Specifies that binary files should be included in the archive collection. By default, all binary files are excluded from data collection.

`--outputPath`

Specifies the path (and optionally, the name) for the support data archive file. If the path specifies a filename, the archive is written to that file. If the path specifies a directory, the file is written to that directory with a server-generated name.

`--useRemoteServer`

Invokes the **`collect-support-data`** tool against a remote server instance and streams the output and resulting support data archive back to the client. This is a useful option when the server instance is running in a container, as it might otherwise be difficult to invoke commands or access files in that container.

PingAuthorize Server logs for troubleshooting and monitoring

PingAuthorize Server has a comprehensive default set of log files and monitor entries that are useful when troubleshooting a particular server problem.

Error log

By default, this log file is available at `logs/errors` below the server install root, and it provides information about warnings, errors, and other significant events that occur within the server. A number of messages are written to this file on startup and shutdown, but while the server is running, there is normally little information written to it. In the event that a problem does occur, the server writes information about that problem to this file.

The following is an example of a message that might be written to the error log:

Copyright © 2025 Ping Identity Corporation

```
[11/Apr/2011:10:31:53.783 -0500] category=CORE severity=NOTICE msgID=458887 msg="The  {pingauthorize}
Server has started successfully."
```

The category field provides information about the area of the server from which the message was generated. Available categories include:

```
ACCESS_CONTROL, ADMIN, ADMIN_TOOL, BACKEND, CONFIG, CORE, DSCONFIG, EXTENSIONS, PROTOCOL, SCHEMA, JEB,
SYNC, LOG, PLUGIN, PROXY, QUICKSETUP, REPLICATION, RUNTIME_INFORMATION, TASK, THIRD_PARTY, TOOLS,
USER_DEFINED, UTIL, VERSION
```

The severity field provides information about how severe the server considers the problem to be. Available severities include:

DEBUG

Used for messages that provide verbose debugging information and do not indicate any kind of problem. Note that this severity level is rarely used for error logging, because the server provides a separate debug logging facility.

INFORMATION

Used for informational messages that can be useful from time to time but are not normally something that administrators need to see.

MILD_WARNING

Used for problems that the server detects, which can indicate something unusual occurred, but the warning does not prevent the server from completing the task it was working on. These warnings are not normally something that should be of concern to administrators.

MILD_ERROR

Used for problems detected by the server that prevented it from completing some processing normally but that are not considered to be a significant problem requiring administrative action.

NOTICE

Used for information messages about significant events that occur within the server and are considered important enough to warrant making available to administrators under normal conditions.

SEVERE_WARNING

Used for problems that the server detects that might lead to bigger problems in the future and should be addressed by administrators.

SEVERE_ERROR

Used for significant problems that have prevented the server from successfully completing processing and are considered important.

FATAL_ERROR

Used for critical problems that arise which might leave the server unable to continue processing operations normally.

Copyright © 2025 Ping Identity Corporation

The messages written to the error log can be filtered based on their severities in two ways. First, the error log publisher has a `default-severity` property, which can be used to specify the severity of messages logged regardless of their category. By default, this includes the `NOTICE`, `SEVERE_WARNING`, `SEVERE_ERROR`, and `FATAL_ERROR` severities.

You can override these severities on a per-category basis using the `override-severity` property. If this property is used, then each value should consist of a category name followed by an equal sign and a comma-delimited set of severities that should be logged for messages in that category. For example, the following override severity would enable logging at all severity levels in the `PROTOCOL` category:

```
protocol=debug,information,mild-warning,mild-error,notice,severe-warning,severe-error,fatal-error
```

Note that for the purposes of this configuration property, any underscores in category or severity names should be replaced with dashes. Also, severities are not inherently hierarchical, so enabling the `DEBUG` severity for a category will not automatically enable logging at the `INFORMATION`, `MILD_WARNING`, or `MILD_ERROR` severities.

The error log configuration can be altered on the fly using tools like **`dsconfig`**, the Administrative Console, or the LDIF connection handler, and changes will take effect immediately. You can configure multiple error logs that are active in the server at the same time, writing to different log files with different configurations. For example, a new error logger can be activated with a different set of default severities to debug a short-term problem, and then that logger can be removed after the problem is resolved, so that the normal error log does not contain any of the more verbose information.

server.out log

The `server.out` file holds any information written to standard output or standard error while the server is running. Normally, it includes a number of messages written at startup and shutdown, as well as information about any administrative alerts generated while the server is running. In most cases, this information is also written to the error log. The `server.out` file can also contain output generated by the JVM. For example, if garbage collection debugging is enabled, or if a stack trace is requested via `kill -QUIT` as described in a later section, then output is written to this file.

Debug log

The debug log provides a means of obtaining information that can be used for troubleshooting problems but is not necessary or desirable to have available while the server is functioning normally. As a result, the debug log is disabled by default, but it can be enabled and configured at any time.

Some of the most notable configuration properties for the debug log publisher include:

`enabled`

Indicates whether debug logging is enabled. By default, it is disabled.

`log-file`

Specifies the path to the file to be written. By default, debug messages are written to the `logs/debug` file.

Copyright © 2025 Ping Identity Corporation

`debug-level`

Specifies the minimum log level for debug messages that should be written. The default value is `error`, which only provides information about errors that occur during processing (for example, exception stack traces). Other supported debug levels include `warning`, `info`, and `verbose`. Note that unlike error log severities, the debug log levels are hierarchical. Configuring a specified debug level enables any debugging at any higher levels. For example, configuring the `info` debug level automatically enables the `warning` and `error` levels.

`debug-category`

Specifies the categories for debug messages that should be written. Some of the most useful categories include `caught` (provides information and stack traces for any exceptions caught during processing), `database-access` (provides information about operations performed in the underlying database), `protocol` (provides information about ASN.1 and LDAP communication performed by the server), and `data` (provides information about raw data read from or written to clients).

As with the error and access logs, multiple debug loggers can be active in the server at any time with different configurations and log files to help isolate information that might be relevant to a particular problem. Debug targets can be used to further pare down the set of messages generated. For example, you can specify that the debug logs be generated only within a specific class or package.

Caution

Enabling one or more debug loggers can have a significant impact on server performance. Enable debug loggers only when necessary, and scope them so that only pertinent debug information is recorded. If you need to enable the debug logger, you should work with your authorized support provider to best configure the debug target and interpret the output.

Config audit log and the configuration archive

The configuration audit log provides a record of any changes made to the server configuration while the server is online. This information is written to the `logs/config-audit.log` file and provides information about the configuration change in the form that can be used to perform the operation in a non-interactive manner with the **`dsconfig`** command. Other information written for each change includes:

- Time that the configuration change was made
- Connection ID and operation ID for the corresponding change, which can be used to correlate it with information in the access log
- DN of the user requesting the configuration change and the method by which that user authenticated to the server
- Source and destination addresses of the client connection
- Command that can be used to undo the change and revert to the previous configuration for the associated configuration object

In addition to information about the individual changes that are made to the configuration, the server maintains complete copies of all previous configurations. These configurations are provided in the `config/archived-configs` directory and are gzip-compressed copies of the `config/config.ldif` file in use before the configuration change was made. The file names contain timestamps that indicate when that configuration was first used.

Copyright © 2025 Ping Identity Corporation

Setup log

The `setup` tool writes a log file providing information about the processing that it performs. By default, this log file is written to `logs/setup.log` although a different name may be used if a file with that name already exists, because the `setup` tool has already been run. The full path to the setup log file is provided when the `setup` tool has completed.

Tool log

Many of the administrative tools provided with the server (for example, `import-ldif`, `export-ldif`, `backup`, `restore`, etc.) can take a significant length of time to complete writing information to standard output or standard error (or both) while the tool is running. They also write additional output to files in the `logs/tools` directory (for example, `logs/tools/ import-ldif.log`). The information written to these log files can be useful for diagnosing problems encountered while they were running. When running using the server tasks interface, log messages generated while the task is running can alternately be written to the server error log file.

Advanced HTTP event logging

The PingAuthorize Server provides advanced HTTP event logging through the file-based trace logger to help diagnose slowdowns in HTTP request processing.

By capturing the full lifecycle of HTTP requests, you can identify performance bottlenecks and fine-tune the server for your workload.

HTTP request lifecycle

The typical HTTP request lifecycle in the PingAuthorize Server is as follows:

1. The server accepts a low-level channel.

   This begins the request lifecycle at the lowest level of the network stack.

   - Example log:

     ```
     HTTP-EVENT CHANNEL channelID=89 msg="accepting channel"
     ```

2. The client establishes a TCP socket with the server.

   This creates the foundational network connection.

   - Example log:

     ```
     HTTP-EVENT SOCKET socketID=90 remotePort="63418" localPort="8443" msg="opened"
     ```

3. An HTTP connection is established over the socket.

   The server is now ready to listen for HTTP-specific traffic.

   - Example log:

Copyright © 2025 Ping Identity Corporation

```
HTTP-EVENT CONNECTION connectionID=90 channelID=90 socketID=89 msg="open"
```

4. The server begins processing the client request.

   The server receives the request and assigns it to a handler thread.

   - Example log:

```
HTTP-EVENT REQUEST-PROCESSING requestHash=1453742073 connectionID=90 channelID=90 socketID=89
method=POST URL="http://localhost:8443/governance-engine" msg="begin request"

HTTP-EVENT REQUEST-PROCESSING requestHash=1453742073 connectionID=90 channelID=90 socketID=89
method=POST URL="http://localhost:8443/governance-engine" msg="dispatching request"
```

5. The server finishes processing the request and sends a response.

> **Note**
>
> This log entry is especially important for performance analysis, as it contains the total request processing duration and the queue time.

   - Example log:

```
HTTP-EVENT REQUEST-PROCESSING requestID=146 correlationID="7fed277a-
c867-4571-9f6d-591722f8754c" requestHash=1453742073 connectionID=90 channelID=90 socketID=89
method=POST URL="https://localhost:8443/governance-engine" totalDurationMs=167
queueDurationMs=1 requestDurationMs=45 requestContentLength=8 responseDurationMs=0
responseContentLength=281 requestBeginTime="23/Jun/2025:20:46:52.037 -0500" dispatchTime="23/
Jun/2025:20:46:52.037 -0500" requestEndTime="23/Jun/2025:20:46:52.082 -0500"
responseBeginTime="23/Jun/2025:20:46:52.202 -0500" responseEndTime="23/Jun/2025:20:46:52.202
-0500" requestCompleteTime="23/Jun/2025:20:46:52.204 -0500" msg="complete processing"
```

6. The server closes the connection.

   After sending the response, the server can close the HTTP connection. For persistent connections, this step might be delayed until a timeout or until the client explicitly closes it.

   - Example log:

```
HTTP-EVENT CONNECTION connectionID=89 channelID=89 totalDurationMs=6 openTime="23/Jun/
2025:20:46:52.029 -0500" closeTime="23/Jun/2025:20:46:52.035 -0500" msg="close"
```

7. The server closes the underlying TCP socket.

   The server terminates the low-level network socket, freeing up the resource.

   - Example log:

Copyright © 2025 Ping Identity Corporation

```
HTTP-EVENT SOCKET socketID=90 remotePort="63418" localPort="8443" totalDurationMs=4
incomingBytes=0 outgoingBytes=0 socketOpenedTime="23/Jun/2025:20:46:52.029 -0500"
socketClosedTime="23/Jun/2025:20:46:52.033 -0500" msg="closed"
```

Each HTTP event includes a unique identifier that allows you to trace the full lifecycle of a request and pinpoint exactly where delays occur, whether in connection setup, thread queuing, or request handling.

Enabling HTTP event logging

To trace the complete lifecycle of HTTP requests, configure the `http-event` property in the file-based trace logger.

You can individually enable the following HTTP event types:

HTTP event type	Description
`socket`	Logs when a TCP socket is opened, closed, or reused. The close event includes the total duration the socket was open.
`connection`	Logs when an HTTP connection is opened, closed, or reused. The close event includes the total duration of the connection.
`channel`	Logs low-level channel events related to the connection.
`request-processing`	Logs when request processing starts and finishes. The finish event includes the total request processing duration and the queue time (the time the request spent waiting for a handler thread).

Note

The queue time calculation logged by `request-processing` events might be less accurate in high-throughput scenarios where sockets are reused for multiple requests.

You can use the administrative console or **`dsconfig`** to enable detailed HTTP event logging.

Tip

High-volume logging can impact performance, so you should only enable event types relevant to your troubleshooting goals.

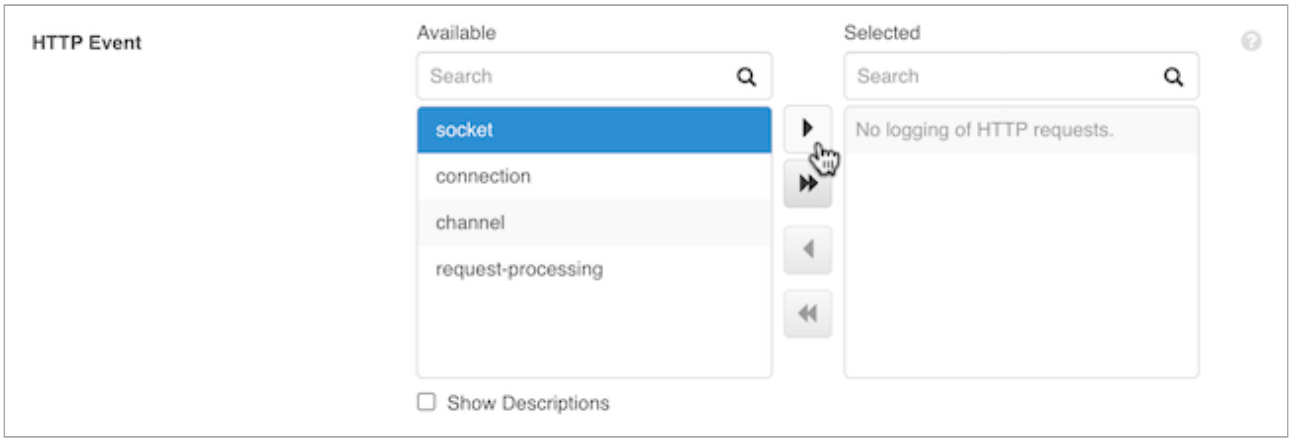
By default, the file-based trace logger writes messages to the `logs/trace.log` file.

Copyright © 2025 Ping Identity Corporation

Admin console

Steps

1. Go to **Logging, Monitoring, and Notifications > Log Publishers**.
2. Click **File-Based Trace Logger**.
3. Select the **Enabled** checkbox if the logger isn't already enabled.
4. Go to **Log Messages To Include > HTTP Event**.
5. In the **Available** list, select an HTTP event type and click the right arrow to move it to the **Selected** list.



6. Click **Save**.

Result

The file-based trace logger now logs messages for the selected event type.

dsconfig

Steps

- Use the `dsconfig set-log-publisher-prop` command with the `--add http-event` argument to enable specific HTTP event types.

  The following example enables logging for socket and request processing events:

```
dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Trace Logger" \
  --add http-event:socket \
  --add http-event:request-processing
```

Result

The file-based trace logger now logs messages for the specified event types.

Copyright © 2025 Ping Identity Corporation

Troubleshooting resources for Java applications

Because PingAuthorize Server is written entirely in Java, it is possible to use standard Java debugging and instrumentation tools when troubleshooting problems with the server.

> **Note**
>
> In many cases, obtaining the full benefit of these tools requires access to the server source code. These Java tools should be used under the advisement of your authorized support provider.

Java troubleshooting tools

The Java Development Kit provides a number of tools for obtaining information about Java applications and diagnosing problems. These tools are not included with the Java Runtime Environment, so the full Java Development Environment should always be installed and used to run the server.

jps

The `jps` tool is a Java-specific version of the UNIX `ps` tool. It can be used to obtain a list of all Java processes currently running and their respective process identifiers. When invoked by a non-root user, it will list only Java processes running as that user. When invoked by a root user, it lists all Java processes on the system.

This tool can be used to see if PingAuthorize Server is running and if a process ID has been assigned to it. This process ID can be used in conjunction with other tools to perform further analysis.

This tool can be run without any arguments, but some of the more useful arguments include:

`-v`

Includes the arguments passed to the Java Virtual Machine (JVM) for the processes that are listed.

`-m`

Includes the arguments passed to the main method for the processes that are listed.

`-l` *(lowercase L)*

Includes the fully qualified name for the main class rather than only the base class name.

jstack

The `jstack` tool is used to obtain a stack trace of a running Java process, or optionally from a core file generated if the JVM happens to crash. A stack trace can be extremely valuable when trying to debug a problem, because it provides information about all threads running and exactly what each thread is doing at the point in time that the stack trace was obtained.

Stack traces are helpful when diagnosing problems in which the server appears to be hung or behaving slowly. Java stack traces are generally more helpful than native stack traces, because Java threads can have user-friendly names (as do the threads used by the server), and the frame of the stack trace may include the line number of the source file to which it corresponds. This is useful when diagnosing problems and often allows them to be identified and resolved quickly.

To obtain a stack trace from a running JVM, use the command:

Copyright © 2025 Ping Identity Corporation

```
jstack  <processID>
```

Replace `<processID>` with the process ID of the target JVM as returned by the **`jps`** command.

To obtain a stack trace from a core file from a Java process, use the command:

```
jstack  <pathToJava>  <pathToCore>
```

Replace `<pathToJava>` with the path to the Java command from which the core file was created, and replace `<pathToCore>` with the path to the core file to examine. In either case, the stack trace is written to standard output and includes the names and call stacks for each of the threads that were active in the JVM.

In many cases, no additional options are necessary. The `-l` option can be added to obtain a long listing, which includes additional information about locks owned by the threads. The `-m` option can be used to include native frames in the stack trace.

jmap

The **`jmap`** tool is used to obtain information about the memory consumed by the JVM. It is very similar to the native **`pmap`** tool provided by many operating systems. As with the **`jstack`** tool, **`jmap`** can be invoked against a running Java process by providing the process ID or against a core file. For example:

```
jmap  <processID>
jmap  <pathToJava>  <pathToCore>
```

Some of the additional arguments include:

`-dump:live,format=b,file=filename`

Dumps the live heap data to a file that can be examined by the **`jhat`** tool.

`-heap`

Provides a summary of the memory used in the Java heap, along with information about the garbage collection algorithm in use.

`-histo:live`

Provides a count of the number of objects of each type contained in the heap. If the `live` option is included, then only live objects are included; otherwise, the count includes objects that are no longer in use and are garbage collected.

jhat

The **`jhat`** (Java Heap Analysis Tool) utility provides the ability to analyze the contents of the Java heap. It can be used to analyze a heap dump file, which is generated if PingAuthorize Server encounters an `out of memory` error (as a result of the `-XX:+HeapDumpOnOutOfMemoryError` JVM option) or from the use of the **`jmap`** command with the `-dump` option.

The **`jhat`** tool acts as a web server that can be accessed by a browser to query the contents of the heap. Several predefined queries are available to help determine the types of objects consuming significant amounts of heap space, and it also provides a custom query language (OQL, the Object Query Language) for performing more advanced types of analysis.

Copyright © 2025 Ping Identity Corporation

The **jhat** tool can be launched with the path to the heap dump file. For example:

```
jhat  </path/to/heap.dump>
```

This command causes the **jhat** web server to begin listening on port 7000. It can be accessed in a browser at http://localhost:7000 (or http://*<address>*:7000 from a remote system). An alternate port number can be specified using the **-port** option. For example:

```
jhat -port 1234  </path/to/heap.dump>
```

To issue custom OQL searches, access the web interface using the URL http://localhost:7000/oql/ (the trailing slash must be provided). Additional information about the OQL syntax may be obtained in the web interface at http://localhost:7000/oqlhelp/.

jstat

The **jstat** tool is used to obtain a variety of statistical information from the JVM, much like the **vmstat** utility, which can be used to obtain CPU utilization information from the operating system. The general manner to invoke it is as follows:

```
jstat  <type>  <processID>  <interval>
```

The *<interval>* option specifies the length of time in milliseconds between lines of output. The *<processID>* option specifies the process ID of the JVM used to run PingAuthorize Server, which can be obtained by running **jps** (as mentioned previously). The *<type>* option specifies the type of output that should be provided. Some of the most useful types include:

-class

Provides information about class loading and unloading.

-compile

Provides information about the activity of the JIT complex.

-printcompilation

Provides information about JIT method compilation.

-gc

Provides information about the activity of the garbage collector.

-gccapacity

Provides information about memory region capacities.

Java diagnostic information

In addition to the tools listed in the previous section, the JVM can provide additional diagnostic information in response to certain events.

Copyright © 2025 Ping Identity Corporation

JVM crash diagnostic information

If the JVM crashes, then it generates a fatal error log with information about the state of the JVM at the time of the crash. By default, this file is named `hs_err_pid<processID>.log` and is written into the base directory of the PingAuthorize Server installation. This file includes information on the underlying cause of the JVM crash, information about the threads running and Java heap at the time of the crash, the options provided to the JVM, environment variables that were set, and information about the underlying system.

Troubleshooting resources in the operating system

The underlying operating system also provides a significant amount of information that can help diagnose issues that impact the performance and stability of PingAuthorize Server.

In some cases, problems with the underlying system can be directly responsible for issues seen with the server, and in others, system tools can help narrow down the cause of the problem.

Identifying problems with the underlying system

If the underlying system itself is experiencing problems, it can adversely impact the function of applications running on it. To look for problems in the underlying system, review the system log file (`/var/log/messages` on Linux). Information about faulted or degraded devices or other unusual system conditions are written there.

Examining CPU utilization

Observing CPU utilization for the server process and the system as a whole provides clues as to the nature of the problem.

System-Wide CPU utilization

To investigate CPU consumption of the system as a whole, use the **`vmstat`** command with a time interval in seconds. For example:

```
vmstat 5
```

The specific output of this command varies between different operating systems, but it includes the percentage of the time the CPU was spent executing user-space code (user time), the percentage of time spent executing kernel-space code (system time), and the percentage of time not executing any code (idle time).

- If the CPUs are spending most of their time executing user-space code, the available processors are being well-utilized.
- If performance is poor or the server is unresponsive, it can indicate that the server is not optimally tuned. If there is a high system time, it can indicate that the system is performing excessive disk and/or network I/O, or in some cases, there can be some other system-wide problem, like an interrupt storm.
- If the system is mostly idle, but the server is performing poorly or is unresponsive, there can be a resource constraint elsewhere (for example, waiting on disk or memory access, or excessive lock contention), or the JVM can be performing other tasks, like stop-the-world garbage collection, that cannot be run heavily in parallel.

Per-CPU utilization

To investigate CPU consumption on a per-CPU basis, use the **`mpstat`** command with a time interval in seconds. For example:

Copyright © 2025 Ping Identity Corporation

```
mpstat 5
```

On Linux systems, it might be necessary to add `-P ALL` to the command. For example:

```
mpstat -P ALL 5
```

Among other things, this command shows the percentage of time each CPU has spent in user time, system time, and idle time. If the overall CPU utilization is relatively low but **`mpstat`** reports that one CPU has a much higher utilization than the others, there might be a significant bottleneck within the server, or the JVM might be performing certain types of garbage collection which cannot be run in parallel. On the other hand, if CPU utilization is relatively even across all CPUs, there is likely no such bottleneck, and the issue might be elsewhere.

Per-process utilization

To investigate CPU consumption on a per-process basis, use a command such as the **`top`** utility on Linux. If a process other than the Java process used to run PingAuthorize Server is consuming a significant amount of available CPU, it might be interfering with the ability of the server to run effectively.

Examining disk utilization

If the underlying system has a very high disk utilization, it can adversely impact server performance. It could delay the ability to read or write database files or write log files. It could also raise concerns for server stability if excessive disk I/O inhibits the ability of the cleaner threads to keep the database size under control.

The **`iostat`** tool can be used to obtain information about the disk activity on the system.

On Linux systems, **`iostat`** should be invoked with the `-x` argument. For example:

```
iostat -x 5
```

Several different types of information will be displayed, but to obtain an initial feel for how busy the underlying disks are, look at the **%util** column on Linux. This field shows the percentage of time that the underlying disks are actively servicing I/O requests. A system with a high disk utilization likely exhibits poor server performance.

If the high disk utilization is on one or more disks that are used to provide swap space for the system, the system might not have enough free memory to process requests. As a result, it might have started swapping blocks of memory that have not been used recently to disk. This can cause very poor server performance. It is important to ensure that the server is configured appropriately to avoid this condition.

If this problem occurs on a regular basis, then the server is likely configured to use too much memory. If swapping is not normally a problem, but it does arise, then check to see if there are any other processes running that are consuming a significant amount of memory, and check for other potential causes of significant memory consumption (for example, large files in a **`tmpfs`** file system).

Examining process details

There are a number of tools provided by the operating system that can help examine a process in detail.

Copyright © 2025 Ping Identity Corporation

ps

The standard `ps` tool can be used to provide a range of information about a particular process. For example, the command can be used to display the state of the process, the name of the user running the process, its process ID and parent process ID, the priority and nice value, resident and virtual memory sizes, the start time, the execution time, and the process name with arguments. For example:

```
ps -fly -p  <processID>
```

Note that for a process with a large number of arguments, the standard `ps` command displays only a limited set of the arguments based on available space in the terminal window.

pstack

The `pstack` command can be used to obtain a native stack trace of all threads in a process. While a native stack trace might not be as user-friendly as a Java stack trace obtained using `jstack`, it includes threads that are not available in a Java stack trace. For example, the command displays those threads used to perform garbage collection and other housekeeping tasks. The general usage for the `pstack` command is:

```
pstack  <processID>
```

dbx / gdb

A process debugger provides the ability to examine a process in detail. Like `pstack`, a debugger can obtain a stack trace for all threads in the process, but it also provides the ability to examine a process (or core file) in much greater detail, including observing the contents of memory at a specified address and the values of CPU registers in different frames of execution. The GNU debugger `gdb` is widely-used on Linux systems.

> **Warning**
>
> Using a debugger against a live process interrupts that process and suspends its execution until it detaches from the process. In addition, when running against a live process, a debugger has the ability to actually alter the contents of the memory associated with that process, which can have adverse effects. As a result, it is recommended that the use of a process debugger be restricted to core files and only used to examine live processes under the direction of your authorized support provider.

pfiles / lsof

To examine the set of files that a process is using (including special types of files, like sockets), you can use a tool such as `lsof` on Linux systems. For example:

```
lsof -p  <processID>
```

Tracing process execution

If a process is unresponsive but is consuming a nontrivial amount of CPU time, or if a process is consuming significantly more CPU time than is expected, it might be useful to examine the activity of that process in more detail than can be obtained using a point-in-time snapshot.

Copyright © 2025 Ping Identity Corporation

For example, if a process is performing a significant amount of disk reads or writes, it can be useful to see which files are being accessed. Similarly, if a process is consistently exiting abnormally, starting a trace for that process just before it exits can help provide additional information that cannot be captured in a core file (and if the process is exiting rather than being terminated for an illegal operation, then no core file may be available).

To perform this trace on Linux, use the **`strace`** tool. For example:

```
strace -f -p  <processID>
```

Consult the **`strace`** manual page for additional information.

Problems with SSL communication

Enable TLS debugging in the server to troubleshoot SSL communication issues. For example:

```
$ dsconfig create-debug-target \
  --publisher-name "File-Based Debug Logger" \
  --target-name com.unboundid.directory.server.extensions.TLSConnectionSecurityProvider \
  --set debug-level:verbose \
  --set include-throwable-cause:true
```

```
$ dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Debug Logger" \
  --set enabled:true \
  --set default-debug-level:disabled
```

In the `java.properties` file, add `-Djavax.net.debug=ssl` to the `start-server` line, and run `bin/dsjavaproperties` to make the option take effect on a scheduled server restart.

Examining network communication

Because the server is a network-based application, it can be valuable to observe the network communication that it has with clients. The server itself can provide details about its interaction with clients by enabling debugging for the protocol or data debug categories, but there can be a number of cases in which it is useful to view information at a much lower level. A network sniffer, like the **`tcpdump`** tool on Linux, can be used to accomplish this.

There are many options that can be used with these tools, and their corresponding manual pages provide a more thorough explanation of their use. However, to perform basic tracing to show the full details of the packets received, for example, on port 389 with remote host 1.2.3.4, the following command can be used on Linux:

```
tcpdump -i  <interface>  -n -XX -s 0 host 1.2.3.4 and port 389
```

It does not appear that the **`tcpdump`** tool provides support for LDAP parsing. However, it is possible to write capture data to a file rather than displaying information on the terminal (using `-w` *<path>* with **`tcpdump`**), so that information can be later analyzed with a graphical tool like Wireshark, which provides the ability to interpret LDAP communication on any port.

Copyright © 2025 Ping Identity Corporation

> **Note**
>
> Enabling network tracing generally requires privileges that are not available to normal users and therefore can require root access.

Common problems and potential solutions

This section describes a number of different types of problems that can occur and common potential causes for them.

General troubleshooting methodology

When you detect a problem, use the following general methodology to isolate the problem.

1. Run the `bin/status` tool or look at the server status in the Administrative Console. The `status` tool provides a summary of the server's current state with key metrics and a list of recent alerts.
2. Look in the server logs. In particular, view the following logs:
   - `logs/errors`
   - `logs/failed-ops`
   - `logs/expensive-ops`
3. Use system commands such as `vmstat` and `iostat` to determine if the server is bottlenecked on a system resource like CPU or disk throughput.
4. For server performance issues (especially intermittent ones like spikes in response time), enable the `periodic-stats-logger` to help isolate problems, because it stores important server performance information on a per-second basis. The `periodic-stats-logger` can save the information in a `.csv` file that can be loaded into a spreadsheet.

   The information this logger makes available is very configurable. You can create multiple loggers for different types of information or a different frequency of logging (for example, hourly data in addition to per-second data). For more information, see Profiling server performance using the Stats Logger.
5. For more advanced users, run the `collect-support-data` tool on the system, unzip the archive, and look through the collected information. This is often useful when administrators most familiar with the data platform do not have direct access to the systems where the production servers are running. They can examine the `collect-support-data` archive on a different server. For more information, see Working with the collect-support-data tool.

> **Important**
>
> Run the `collect-support-data` tool whenever you can't easily identify the cause of a problem. You can pass this information to your authorized support provider for assistance in identifying and addressing the root cause of the issue.

The server will not run setup

If the `setup` tool does not run properly, some of the most common reasons include the following:

Copyright © 2025 Ping Identity Corporation

A suitable Java environment is not available

The server requires that Java be installed on the system and made available to the server, and it must be installed prior to running `setup`. If the `setup` tool does not detect that a suitable Java environment is available, it will refuse to run.

To ensure that this does not happen, the `setup` tool should be invoked with an explicitly defined value for the *JAVA_HOME* environment variable that specifies the path to the Java installation that should be used. For example:

```
env JAVA_HOME=/ds/java ./setup
```

If the previous command doesn't solve the issue, the value specified in the provided *JAVA_HOME* environment variable might have been overridden by another environment variable. In that case, try the following command, which should override any other environment variables that can be set:

```
env UNBOUNDID_JAVA_HOME="/ds/java" UNBOUNDID_JAVA_BIN="" ./setup
```

Unexpected arguments provided to the JVM

If the `setup` script attempts to launch the `java` command with an invalid set of Java arguments, it might prevent the JVM from starting. By default, no special options are provided to the JVM when running `setup`, but this might not be the case if either the *JAVA_ARGS* or *UNBOUNDID_JAVA_ARGS* environment variable is set. If the `setup` tool displays an error message that indicates that the Java environment could not be started with the provided set of arguments, then invoke the following command before trying to re-run `setup`:

```
unset JAVA_ARGS UNBOUNDID_JAVA_ARGS
```

The server has already been configured or used

The `setup` tool is only intended to provide the initial configuration for the server. It refuses to run if it detects that the `setup` tool has already been run, or if an attempt has been made to start the server prior to running the `setup` tool. This protects an existing server installation from being inadvertently updated in a manner that could harm an existing configuration or data set.

If the server has been previously used, and if you want to perform a fresh installation, it is recommended that you first remove the existing installation, create a new one, and run `setup` in that new installation. However, if you are confident that there is nothing of value in the existing installation (for example, if a previous attempt to run `setup` failed to complete successfully, but it will refuse to run again), follow these cleanup steps to allow the `setup` program to run:

- Remove the `config/config.ldif` file and replace it with the `config/update/config.ldif.<revision>` file containing the initial configuration.
- Remove any files or subdirectories below the `db` directory if they exist.
- Remove the `config/java.properties` file if it exists.
- Remove the `lib/setup-java-home` script (or `lib\setup-java-home.bat` file on Microsoft Windows) if it exists.

The server will not start

If the server does not start, then there are a number of potential causes.

Copyright © 2025 Ping Identity Corporation

The server or other administrative tool is already running

Only a single instance of the server can run at any time from the same installation root. If an instance is already running, then subsequent attempts to start the server will fail. Similarly, some other administrative operations can also prevent the server from being started. In such cases, the attempt to start the server should fail with a message like the following:

```
{pingauthorize}  Server could not acquire an exclusive lock on file
/{pingauthorize}/locks/server.lock: The exclusive lock requested for file
/{pingauthorize}/locks/server.lock was not granted, which indicates
that another process already holds a shared or exclusive lock on that
file. This generally means that another instance of this server is already
running
```

If the server is not running (and is not in the process of starting up or shutting down), and there are no other tools running that could prevent the server from being started, and the server still believes that it is running, then it is possible that a previously held lock was not properly released. In that case, you can try removing all of the files in the `locks` directory before attempting to start the server.

> **Tip**
>
> If you wish to have multiple instances running at the same time on the same system, then you should create a completely separate installation in another location on the file system.

There is not enough memory available

When the server is started, the JVM attempts to allocate all memory that it has been configured to use. If there is not enough free memory available on the system, then the server generates an error message that indicates that the server could not be started with the specified set of arguments.

> **Note**
>
> It's possible that an invalid option was provided to the JVM (as described in the following potential causes), but if that same set of JVM arguments has already been used successfully to run the server, then it is more likely that the system does not have enough memory available.

There are a number of potential causes for this issue:

- If the amount of memory in the underlying system has changed (for example, system memory has been removed, or the server is running in a zone, or other type of virtualized container, and a change has been made to the amount of memory that container is allowed to use), then the server might need to be re-configured to use a smaller amount of memory than had been previously configured.
- Another process running on the system is consuming a significant amount of memory, so that there is not enough free memory available to start the server. If this is the case, then either terminate the other process to make more memory available for the server, or reconfigure the server to reduce the amount of memory that it attempts to use.
- The server was just shut down and an attempt was made to immediately restart it. In some cases, if the server is configured to use a significant amount of memory, then it can take a few seconds for all of the memory that had been in use by the server, when it was previously running, to be released back to the operating system. In that case, run the **`vmstat`** command and wait until the amount of free memory stops growing before attempting to restart the server.

Copyright © 2025 Ping Identity Corporation

- If the system is configured with one or more memory-backed file systems, verify whether any large files might be consuming a significant amount of memory in any of those locations. If so, remove them or relocate them to a disk-based file system.
- For Linux systems only, there could be a mismatch between the huge pages setting for the JVM and the huge pages reserved in the operating system.

> **Tip**
>
> If nothing else works, and there is still not enough free memory to allow the JVM to start, then as a last resort, try rebooting the system.

An invalid Java Environment or JVM option was used

If an attempt to start the server fails with an error message indicating that no valid Java environment could be found, or indicates that the Java environment could not be started with the configured set of options, then you should first ensure that enough memory is available on the system as described previously. If there is a sufficient amount of memory available, then other causes for this error can include the following:

- The Java installation that was previously used to run the server no longer exists (for example, an updated Java environment was installed and the old installation was removed). In that case, update the `config/java.properties` file to reference to path to the new Java installation and run the `bin/dsjavaproperties` command to apply that change.
- The Java installation used to run the server has been updated, and the server is trying to use the correct Java installation, but one or more of the options that had worked with the previous Java version no longer work with the new version. In that case, it is recommended that the server be re-configured to use the previous Java version, so that it can be run while investigating which options should be used with the new installation.
- If an *UNBOUNDID_JAVA_HOME* or *UNBOUNDID_JAVA_BIN* environment variable is set, then its value might override the path to the Java installation used to run the server as defined in the `config/java.properties` file. Similarly, if an *UNBOUNDID_JAVA_ARGS* environment variable is set, then its value might override the arguments provided to the JVM. If this is the case, then explicitly unset the *UNBOUNDID_JAVA_HOME*, *UNBOUNDID_JAVA_BIN*, and *UNBOUNDID_JAVA_ARGS* environment variables before trying to start the server.

> **Note**
>
> Any time the `config/java.properties` file is updated, the **`bin/dsjavaproperties`** tool must be run to apply the new configuration. If a problem with the previous Java configuration prevents the **`bin/dsjavaproperties`** tool from running properly, then it can be necessary to remove the `lib/set-java-home` script (or `lib\set-java-home.bat` file on Microsoft Windows) and invoke the **`bin/dsjavaproperties`** tool with an explicitly-defined path to the Java environment, as in the following example:
>
> ```
> env UNBOUNDID_JAVA_HOME=/{pingauthorize}/java bin/dsjavaproperties
> ```

An invalid command-line option was provided

You might provide a small number of arguments when running the **`bin/start-server`** command, but in most cases, none are required. If you provided one or more command-line arguments for the **`bin/start-server`** command, and any one of them is not recognized, then the server provides an error message indicating that an argument was not recognized and displays version information. In that case, correct or remove the invalid argument and try to start the server again.

Copyright © 2025 Ping Identity Corporation

The server has an invalid configuration

If you make a change to the server configuration using an officially-supported tool like **`dsconfig`** or the administrative console, the server should validate that configuration change before applying it. However, it is possible that a configuration change can appear to be valid at the time that it is applied but does not work as expected when the server is restarted. Alternately, a change in the underlying system can cause a previously-valid configuration to become invalid.

In most cases involving an invalid configuration, the server displays (and writes to the error log) a message that explains the problem. This might be sufficient to identify the problem and understand what action needs to be taken to correct it. If the startup failure does not provide enough information to identify the problem with the configuration, then look in the `logs/config-audit.log` file to see what recent configuration changes have been made with the server online, or in the `config/archived-configs` directory to see if there might have been a recent configuration change resulting from a direct change made to the configuration file itself, rather than through a supported configuration interface.

If the server does not start as a result of a recent invalid configuration change, then it can be possible to start the server using the configuration that was in place the last time that the server started successfully (for example, the last known-good configuration). This can be achieved using the `--useLastKnownGoodConfig` option. For example:

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

If it has been a long time since the server was started and a number of configuration changes have been made since that time, then the last known-good configuration can be significantly out of date. In such cases, it can be preferable to manually repair the configuration.

If there is no last known-good configuration, if the server no longer starts with the last known-good configuration, or if the last known-good configuration is significantly out of date, then manually update the configuration by editing the `config/config.ldif` file.

> **Important**
>
> Before editing the `config/config.ldif` file, ensure that the server is offline and make a copy of the existing configuration. You might wish to discuss the configuration change with your authorized support representative before applying it to ensure that you understand the correct change that needs to be made.
> In addition to manually editing the configuration file, you can look at previous archived configurations to see if the most recent one works. You can also use the **`ldif-diff`** tool to compare the configurations in the archive to the current configuration and review the differences.

You do not have sufficient permissions

The server should only be started by the user or role used to initially install the server. In most cases, if an attempt is made to start the server as a user or role other than the one used to create the initial configuration, the server fails to start, because the user doesn't have sufficient permissions to access files owned by the other user, such as database and log files.

However, if a non-root user installs the server and then the root account starts the server, it becomes impossible for a non-root user to start the server. This is because newly created files are owned by root and can't be written to by other users.

If the server is unintentionally started by root, or if you wish to change the user account that should be used to run the server, change ownership on all files in the server installation so that they are owned by the user or role under which the server should run. For example, if the server should be run as the `ds` user in the `other` group, then run the following command as the root user:

Copyright © 2025 Ping Identity Corporation

```
chown -R ds:other /ds/the
```

The server has crashed or shut itself down

If the server has crashed or shut itself down, first check the current server state by using the **`bin/server-state`** command.

If PingAuthorize Server was previously running but is no longer active, then the potential causes include the following:

- PingAuthorize Server was shut down by an administrator. Unless the server was forcefully terminated (for example, using `kill -9`), then messages are written to the `error` and `server.out` logs explaining the reason for the shutdown.
- PingAuthorize Server was shut down when the underlying system crashed or was rebooted. If this is the case, then running the **`uptime`** command on the underlying system shows that it was recently booted.
- The PingAuthorize Server process was terminated by the underlying operating system for some reason (for example, the out-of-memory killer on Linux). If this happens, then a message is written to the system error log.
- PingAuthorize Server decided to shut itself down in response to a serious problem. At present, this should only occur if the server has detected that the amount of usable disk space has become critically low, or if significant errors during processing have left the server without any remaining worker threads to process operations. If this happens, then messages are written to the `error` and `server.out` logs (if disk space is available) to provide the reason for the shutdown.
- The JVM in which PingAuthorize Server was running crashed. If this happens, then the JVM should dump a fatal error log (an `hs_err_pid<processID>.log` file) and potentially a core file.

In the event that the operating system itself crashed or terminated the process, then you should work with your operating system vendor to diagnose the underlying problem. If the JVM crashed or the server shut itself down for a reason that is not clear, then contact your authorized support provider for further assistance.

Conditions for automatic server shutdown

All PingAuthorize Server instances will shut down in an out-of-memory condition, a low-disk-space error state, or if they run out of file descriptors.

The server will enter lockdown mode on unrecoverable database environment errors, but can be configured to shutdown instead with this setting:

```
$ dsconfig set-global-configuration-prop \
            --set unrecoverable-database-error-mode:initiate-server-shutdown
```

The server will not accept client connections

If the server does not appear to be accepting connections from clients, you can first check the current server state by using the **`bin/server-state`** command.

Copyright © 2025 Ping Identity Corporation

Potential reasons the server won't accept client connections include the following:

- The server is not running.
- The underlying system on which the server is installed is not running.
- The server is running but is not reachable as a result of a network or firewall configuration problem. If that is the case, then connection attempts should time out rather than be rejected.
- If the server is configured to allow secure communication via SSL or StartTLS, then a problem with the key manager and/or trust manager configuration can cause connections to be rejected. If that is the case, then messages should be written to the server access log for each failed connection attempt.
- If the server has been configured with a maximum allowed number of connections, then it can be that the maximum number of allowed client connections are already established. If that is the case, then messages should be written to the server access log for each rejected connection attempt.
- If the server is configured to restrict access based on the address of the client, then messages should be written to the server access log for each rejected connection attempt.
- If a connection handler encounters a significant error, then it can stop listening for new requests. If this occurs, then a message should be written to the server error log with information about the problem. You can also restart the server to address this condition.

The server is unresponsive

If the server process is running and appears to be accepting connections, but doesn't respond to requests received on those connections, first check the current server state by using the `bin/server-state` command.

> **Important**
>
> If it appears that the problem is with the server software or the JVM in which it is running, then you need to work with your authorized support provider to fully diagnose the problem and determine the best course of action to correct it.

If the server is unresponsive, then potential reasons for this behavior include:

- If all worker threads are busy processing other client requests, then new requests that arrive will be forced to wait in the work queue until a worker thread becomes available. If this is the case, then a stack trace obtained using the `jstack` command shows that all of the worker threads are busy and none of them are waiting for new requests to process.

  A dedicated thread pool can be used for processing administrative operations. This thread pool enables diagnosis and corrective action if all other worker threads are processing operations. To request that operations use the administrative thread pool, using the `ldapsearch` command for example, include the `--useAdministrativeSession` option. The requester must have the `use-admin-session` privilege (included for root users). By default, eight threads are available for this purpose. This can be changed with the `num-administrative-session-worker-threads` property in the work queue configuration.

  > **Note**
  >
  > If all of the worker threads are tied up processing the same operation for a long time, the server will also issue an alert that it might be deadlocked, which might not actually be the case. All threads might be tied up processing unindexed searches.

Copyright © 2025 Ping Identity Corporation

- If a request handler is stuck performing some expensive processing for a client connection, then other requests sent to the server on connections associated with that request handler are forced to wait until the request handler is able to read data on those connections. If this is the case, then only some of the connections can experience this behavior (unless there is only a single request handler, in which it will impact all connections), and stack traces obtained using the `jstack` command show that a request handler thread is continuously blocked rather than waiting for new requests to arrive. Note that this scenario is a theoretical problem and one that has not appeared in production.
- If the JVM in which the server is running is not properly configured, then it can be forced to spend a significant length of time performing garbage collection, and in severe cases, could cause significant interruptions in the execution of Java code. In such cases, a stack trace obtained from a `pstack` of the native process should show that most threads are idle, but at least one thread performing garbage collection is active. It is also likely that one or a small number of CPUs are 100% busy while all other CPUs are mostly idle. The server will also issue an alert after detecting a long JVM pause (due to garbage collection). The alert will include details of the pause.
- If the JVM in which the server is running has hung for some reason, then the `pstack` utility should show that one or more threads are blocked and unable to make progress. In such cases, the system CPUs should be mostly idle.
- If a network or firewall configuration problem arises, then attempts to communicate with the server cannot be received by the server. In that case, a network sniffer like `snoop` or `tcpdump` should show that packets sent to the system on which the server is running are not receiving TCP acknowledgement.
- If the system on which the server is running has become hung or lost power with a graceful shutdown, then the behavior is often similar to that of a network or firewall configuration problem.

The server is slow to respond to client requests

If the server is running and does respond to clients, but clients take a long time to receive responses, then the problem can be attributable to a number of potential problems.

> **Tip**
>
> In these cases, use the Periodic Stats Logger, which is a valuable tool to get per-second monitoring information on the server. The Periodic Stats Logger can save the information in `.csv` format for easy viewing in a spreadsheet. For more information, see Profiling server performance using the Stats Logger.

The potential problems that cause slow responses to client requests are as follows:

The server is not optimally configured for the type of requests being processed, or clients are requesting inefficient operations.

If this is the case, then the access log should show that operations are taking a long time to complete and they will likely be unindexed. Updating the server configuration to better suit the requests or altering the requests to make them more efficient could help alleviate the problem.

Review the expensive operations access log in `logs/expensive-ops`, which by default logs operations that take longer than 1 second. You can also run the `bin/status` command or view the status in the Administrative Console to see the server's `Work Queue` information (also see the following case).

Copyright © 2025 Ping Identity Corporation

The server is overwhelmed with client requests and has amassed a large backlog of requests in the work queue.

This can be the result of a configuration problem (for example, too few worker threads configured), or it can be necessary to provision more systems on which to run the server software. Symptoms of this problem appear similar to those experienced when the server is asked to process inefficient requests, but looking at the details of the requests in the access log show that they are not necessarily inefficient requests.

Run the `bin/status` command to view the `Work Queue` information. If everything is performing well, you should not see a large queue size or a server that is near 100% busy. The `% Busy` statistic is calculated as the percentage of worker threads that are busy processing operations. For example:

```
           --- Work Queue ---
           : Recent : Average : Maximum
-----------:--------:---------:--------
Queue Size : 10   : 1       : 10
% Busy     : 17     : 14      : 100
```

You can also view the expensive operations access log in `logs/expensive-ops`, which by default logs operations that take longer than 1 second.

The server is not configured to fully cache all of the data in the server, or the cache is not yet primed.

In this case, `iostat` reports a very high disk utilization. This can be resolved by configuring the server to fully cache all data and to load database contents into memory on startup. If the underlying system does not have enough memory to fully cache the entire data set, then it might not be possible to achieve optimal performance for operations that need data which is not contained in the cache. For more information, see Tuning for disk-bound deployments.

If the JVM is not properly configured, then it will need to perform frequent garbage collection and periodically pause execution of the Java code that it is running.

In that case, the server error log should report that the server has detected a number of pauses and can include tuning recommendations to help alleviate the problem.

If the server is configured to use a large percentage of the memory in the system, then it is possible that the system has gotten low on available memory and has begun swapping.

In this case, `iostat` should report very high utilization for disks used to hold swap space, and commands like `cat /proc/meminfo` on Linux can report a large amount of swap memory in use. Another cause of swapping is if swappiness is not set to `0` on Linux. For more information, see Disabling file system swapping.

If another process on the system is consuming a significant amount of CPU time, then it can adversely impact the ability of the server to process requests efficiently.

Isolating the processes (for example, using processor sets) or separating them onto different systems can help eliminate this problem.

Copyright © 2025 Ping Identity Corporation

The server returns error responses to client requests

If a large number of client requests are receiving error responses, then review the `logs/failed-ops` log, which is an access log only for failed operations.

The potential reasons for the error responses include the following:

- If clients are requesting operations that legitimately should fail (for example, they are targeting entries that do not exist, are attempting to update entries in a way that would violate the server schema, or are performing some other type of inappropriate operation), then the problem is likely with the client and not the server.
- If the PingAuthorize Server work queue is configured with a maximum capacity, and that capacity has been reached, then the server begins rejecting all new requests until space is available in the work queue. In this case, you might need to alter the server configuration, the client requests, or both so that they can be processed more efficiently. Alternatively, you might need to add additional server instances to handle some of the workload.
- If an internal error occurs within the server while processing a client request, then the server terminates the connection to the client and logs a message about the problem that occurred. This should not happen under normal circumstances. Contact your authorized support provider for help with diagnosing and correcting the problem.
- If the server has an issue while interacting with the underlying database (for example, an attempt to read from or write to disk failed because of a disk problem or lack of available disk space), then the server can begin returning errors for all attempts to interact with the database. This will continue until you close and re-open the back end and give the database a chance to recover itself. For information about this type of issue, review the `je.info.*` file in the database directory.

Problems with the administrative console

To troubleshoot problems with the administrative console, consider these potential causes:

- The web application container used to host the console is not running. If an error occurs while trying to start it, then consult the logs for the web application container.
- If a problem occurs while trying to authenticate to the web application container, then make sure that the target PingAuthorize Server is online. If it is online, then the access log may provide information about the reasons for the authentication failure.

JVM memory issues

For servers running Java 7 and hosting web applications like the administrative console, an inadequate **PermSize** setting might cause an `OutOfMemoryError` related to `PermGen space`, as shown in the following error log example:

Copyright © 2025 Ping Identity Corporation

```
[02/Mar/2016:07:50:27.017 -0600] threadID=2 category=UTIL
    severity=SEVERE_ERROR msgID=-1 msg="The server experienced an unexpected
    error. Please report this problem and include this log file.
    OutOfMemoryError: PermGen space
    ()\ncom.unboundid.directory.server.core.DirectoryServer.uncaughtException
    (DirectoryServer.java:15783)\njava.lang.ThreadGroup.uncaughtException
    (ThreadGroup.java:1057)\njava.lang.ThreadGroup.uncaughtException
    (ThreadGroup.java:1052)\njava.lang.ThreadGroup.uncaughtException
    (ThreadGroup.java:1052)\njava.lang.Thread.dispatchUncaughtException
    (Thread.java:1986)\nBuild revision: 22496\n"
```

Problems with the HTTP Connection Handler

When problems with the HTTP Connection Handler occur, first look at the HTTP connection handler log to diagnose the issue.

The following HTTP log examples detail various errors that can occur related to the HTTP Connection Handler:

Failed request due to a non-existent resource

The server receives a status code 404, which indicates the server could not match the URI:

```
[15/Mar/2012:17:39:39 -0500] RESULT requestID=0 from="10.2.1.113:52958"
method="GET" url="https://10.2.1.113:443/Aleph/Users/uid=user.1,ou=people,
dc=example,dc=com" requestHeader="Host: x2270-11.example.lab"
requestHeader="Accept: */* " requestHeader="User-Agent: curl/7.21.6
(i386-pc-centos2.10) libcurl/7.21.6 OpenSSL/1.0.0d zlib/1.2.5 libidn/1.22
libssh2/1.2.7" authorizationType="Basic" statusCode=404 etime=81.484
responseContentLength=103 responseHeader="Access-Control-Allow-Credentials:true"
responseContentType="application/json"
```

Failed request due to a malformed request body

The server receives a status code 400, which indicates that the request had malformed syntax in its request body:

```
[15/Mar/2012:17:47:23-0500] RESULT requestID=10 from="10.2.1.113:55284"
method="POST" url="https://10.2.1.113:443/Aleph/Users" requestHeader="Host:
x2270-11.example.lab" requestHeader="Expect: 100-continue"
requestHeader="Accept: */* " requestHeader="Content-Type: application/json"
requestHeader="User-Agent: curl/ 7.21.6 (i386-pc-centos2.10) libcurl/7.21.6
OpenSSL/1.0.0d zlib/1.2.5 libidn/1.22 libssh2/1.2.7" authorizationType="Basic"
requestContentType="application/json" requestContentLength=5564 statusCode=400
etime=15.272 responseContentLength=133 responseContentType="application/json"
```

Failed request due to an unsupported HTTP method

The server receives a status code 405, which indicates that the specified HTTP method (e.g., `"PATCH"`) in the request line is not allowed for the resource identified in the URI:

Copyright © 2025 Ping Identity Corporation

```
[15/Mar/2012:17:48:59-0500] RESULT requestID=11 from="10.2.1.113:55763"
method="PATCH" url="https://10.2.1.113:443/Aleph/Users" requestHeader="Host:
x2270-11.example.lab" requestHeader="Accept: / " requestHeader="Content-Type:
application/json" requestHeader="User-Agent: curl/7.21.6 (i386-pc-centos2.10)
libcurl/7.21.6 OpenSSL/1.0.0d zlib/1.2.5 libidn/1.22 libssh2/1.2.7"
authorization-Type="Basic" requestContentType="application/json" statusCode=405
etime=6.807 responseContentLength=0 responseHeader="Allow: POST, GET, OPTIONS, HEAD"
```

Failed request due to an unsupported media type

The server receives a status code 415, which indicates that the request entity is in a format that is not supported by the requested resource:

```
[15/Mar/2012:17:44:45-0500] RESULT requestID=4 from="10.2.1.113:54493"
method="POST" url="https://10.2.1.113:443/Aleph/Users" requestHeader="Host:
x2270-11.example.lab" requestHeader="Accept: / " requestHeader="Content-Type:
application/atom+xml" requestHeader="User-Agent: curl/7.21.6 (i386-pc-centos2.10)
libcurl/7.21.6 OpenSSL/1.0.0d zlib/1.2.5 libidn/1.22 libssh2/1.2.7"
authorizationType="Basic" requestContentType="application/atom+xml"
requestContentLength=3 statusCode=415 etime=6.222 responseContentLength=1402
responseHeader="Cache-Control: must-revalidate,no-cache,no-store"
responseContentType="text/html;charset=ISO-8859-1"
```

Failed request due to an authentication error

The server receives a status code 401, which indicates that the request requires user authentication:

```
[15/Mar/2012:17:46:06-0500] RESULT requestID=8 from="10.2.1.113:54899"
method="GET" url="https://10.2.1.113:443/Aleph/Schemas" requestHeader="Host:
x2270-11.example.lab" requestHeader="Accept: / " requestHeader="User-Agent:
curl/7.21.6 (i386-pc-centos2.10) libcurl/7.21.6 OpenSSL/1.0.0d zlib/1.2.5
libidn/1.22 libssh2/ 1.2.7" authorizationType="Basic" statusCode=401
etime=2.751 responseContentLength=63 responseHeader="WWW-Authenticate: Basic
realm=SCIM" responseHeader="Access-Control-Allow-Credentials: true"
responseContentType="application/json"
```

Detecting expensive HTTP threads

The HTTP Connection Handler can detect expensive threads that spend an unusually long time processing HTTP requests. This helps identify potential performance bottlenecks and responsiveness issues in the PingAuthorize Server.

When request processing takes longer than a configured threshold, the PingAuthorize Server can automatically generate a thread dump. A thread dump is a snapshot of all active application threads, indicating whether each thread is actively running, blocked waiting for a resource, or waiting for another task to complete. Thread dumps also include the sequence of method calls each thread is currently executing.

This diagnostic information is helpful for troubleshooting slowdowns caused by issues such as:

- Delays in connecting to external data sources

Copyright © 2025 Ping Identity Corporation

- Long-running policy evaluations
- Resource contention or deadlocks
- Infrastructure problems affecting responsiveness

Analyzing a thread dump enables you to pinpoint the specific operations responsible for performance degradation.

How it works

The PingAuthorize Server uses the following process to detect expensive threads:

1. At a regular interval, the server checks the status of all active HTTP threads.
2. If a thread is still handling the same request it was processing during the previous check, the server flags that thread as expensive.
3. If the number of concurrently expensive threads meets a configured threshold, the server creates a thread dump.
4. Each thread dump is saved as a separate file in the `/logs/thread-dumps` directory. To avoid excessive logging, the server waits for a configured interval before creating another thread dump.

   Thread dump files are named with the following format:

   `expensive-operation-dump-<timestamp>.log`

   > **Note**
   >
   > The thread dump file path and naming format aren't configurable.

Configuring expensive thread detection

The HTTP Connection Handler provides the following settings for configuring expensive thread detection:

Setting name	`dsconfig` parameter name	Description	Default value
Expensive Thread Check Interval	`expensive-thread-check-interval`	Specifies how frequently the server checks thread activity. If a thread is processing the same request for two consecutive checks, the server flags it as expensive. **Note** You must specify a non-zero value for this setting to enable expensive thread detection.	`0 ms`

Copyright © 2025 Ping Identity Corporation

Setting name	`dsconfig` parameter name	Description	Default value
Expensive Thread Minimum Concurrent Count	`expensive-thread-minimum-concurrent-count`	Specifies the minimum number of simultaneously expensive threads required to generate a thread dump.	`1`
Expensive Thread Hold Off Interval	`expensive-thread-hold-off-interval`	Specifies the cooldown period the server waits after creating a thread dump before it can create another. This setting helps prevent excessive disk usage.	`60000 ms`

You can configure expensive thread detection with the administrative console or with the **`dsconfig`** command.

Admin console

Steps

1. Go to **System > Connection Handlers**.
2. Click **HTTP Connection Handler** or **HTTPS Connection Handler**.
3. Scroll down and configure the following:
    - **Expensive Thread Check Interval**
    - **Expensive Thread Minimum Concurrent Count**
    - **Expensive Thread Hold Off Interval**

dsconfig

Steps

- Run the **`dsconfig set-connection-handler-prop`** command with the following arguments:

```
dsconfig set-connection-handler-prop \
  --handler-name "HTTP Connection Handler" \
  --set "expensive-thread-check-interval:<time-interval> ms" \
  --set "expensive-thread-minimum-concurrent-count:<thread-count>" \
  --set "expensive-thread-hold-off-interval:<time-interval> ms"
```


Copyright © 2025 Ping Identity Corporation

> **Important**
>
> You must restart the server or the HTTP Connection Handler for the expensive thread detection settings to take effect.

Analyzing thread dump files

Each thread dump file is organized into three main parts:

- Header summary: Provides context about why the thread dump was generated. It includes the name of the detector that triggered the dump, the number of expensive threads identified, and the time interval over which they were detected.
- Expensive thread list: Lists the threads flagged as expensive immediately after the summary. Each entry includes a thread ID, thread name, and correlation ID to help trace related request activity across other logs.

  > **Tip**
  >
  > Save the thread IDs for use during analysis.

- Full stack trace: Shows a complete snapshot of all threads running in the PingAuthorize Server's Java process at the time of the thread dump. This section provides detailed request context to help identify performance bottlenecks or blocked threads.

  Use the thread IDs from the expensive thread list to locate and analyze each expensive thread's state.

For example, the following header summary and expensive thread list identify a single expensive thread, detected over an interval of 5000 milliseconds:

```
HTTP Connection Handler :80 worker Expensive HTTP Thread Detector (10.3.0.0-20250611152739.000Z-364a3822)
detected that 1 threads were found to be processing the same HTTP request over an interval of at least 5000
ms:
* Thread id=176 name='HTTP Connection Handler :80 worker 6 (10.3.0.0-20250611152739.000Z-364a3822)
requestID=0 correlationID="3bf39de1-7bc4-4937-b99a-62785a41f55d"
```

The following is a truncated version of the expensive thread's stack trace entry:

```
"HTTP Connection Handler :80 worker 6 (10.3.0.0-20250611152739.000Z-364a3822)" priority=5 id=176
   java.lang.Thread.State: WAITING
        at java.base@17.0.10/jdk.internal.misc.Unsafe.park(Native Method)
        at java.base@17.0.10/java.util.concurrent.locks.LockSupport.park(LockSupport.java:211)
        at java.base@17.0.10/java.util.concurrent.CompletableFuture$Signaller.block(CompletableFuture.java:
1864)
        at java.base@17.0.10/java.util.concurrent.ForkJoinPool.unmanagedBlock(ForkJoinPool.java:3465)
        at java.base@17.0.10/java.util.concurrent.ForkJoinPool.managedBlock(ForkJoinPool.java:3436)
        at java.base@17.0.10/java.util.concurrent.CompletableFuture.waitingGet(CompletableFuture.java:1898)
        at java.base@17.0.10/java.util.concurrent.CompletableFuture.get(CompletableFuture.java:2072)
        at app//com.pingidentity.authorize.decisionengine2.evaluation.effects.Inner.get(Inner.java:32)
```

A thread state of `WAITING` indicates the thread is idle, waiting indefinitely for another thread to perform an action. Learn more about thread states in the `Enum Thread.State` reference in the Java documentation.

Copyright © 2025 Ping Identity Corporation

Thread dump file management

The PingAuthorize Server doesn't automatically manage the thread dump files it creates. You must implement your own retention policy for these files, including any log rotation, compression, or cleanup.

In a containerized deployment, make sure a mechanism is in place, such as a volume mount, to persist thread dump files and make them available for analysis outside the container.

Providing information for support cases

If you are unable to fully diagnose with the server, contact your authorized support provider for assistance.

To ensure that a problem can be addressed as quickly as possible, provide all of the information that the support personnel might need to understand the underlying cause. Run the `collect-support-data` tool, and then send the generated `.zip` file to your authorized support provider.

> **Tip**
>
> It is good practice to run this tool and send the `.zip` file to your authorized support provider before you take any corrective action.

Copyright © 2025 Ping Identity Corporation

PingAuthorize API Reference

The PingAuthorize Server and Policy Editor provide a collection of APIs for managing authorization resources and leveraging fine-grained authorization capabilities in your application or API service.

- The Authorization Policy Decision APIs support non-API use cases where the PingAuthorize Server acts as the policy decision point (PDP), and your application server acts as the policy enforcement point (PEP). Learn more in the Authorization Policy Decision API reference.
- The Policy Editor APIs enable programmatic management of authorization resources, including:
  - Authorization policies, rules, targets, and statements
  - Trust Framework elements such as attributes, conditions, processors, and services
  - Branches, snapshots, and deployment packages
  - Test scenarios, test cases, and assertions

  Learn more in the Policy Editor API reference.
- The 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. Learn more in the SCIM API reference.

Copyright © 2025 Ping Identity Corporation