PingDataGovernance™

Release 7.3.0.3

Policy Administration Guide

Notice

PingDataGovernance™ and Symphonic™ Product Documentation

Contents

Chapter 1

Install the PingDataGovernance Policy Administration GUI

Topics:

This section provides instructions for installing and upgrading the PingDataGovernance Policy Administration GUI.

System requirements and prerequisites

Operating system

Powered by Symphonic, the PingDataGovernance Policy Administration GUI product suite components are distributed as executable scripts that can be run on any system with a Java SE Runtime 8 or Java SE Runtime 11 environment. They have been specifically tested with both Oracle and OpenJDK JRE installations on the following operating systems:

- Ubuntu 16.04 LTS and 18.04 LTS
- Red Hat Enterprise Linux 7.5 and 7.6
- Microsoft Windows 2016 Server

Hardware

The hardware requirements for the PingDataGovernance Policy Administration GUI depend on the expected load. The recommended minimum for each installation is as follows:

- Modern x64-compatible processor
- 4GB RAM
- 100GB disk space

Java SE Runtime Environment

The PingDataGovernance Policy Administration GUI Administration Point application requires Java SE Runtime 8 or 11. (Either OpenJDK or Oracle is recommended). Where possible, install the Java SE Runtime Environment through the operating system's package manager. Alternatively, you can download the latest Oracle Java 8 JRE or Java 11 from the following location:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

After the environment is installed, ensure that the `JAVA_HOME` environment variable is set to the Server JRE installation directory path, and ensure that its `bin` directory is added to the `PATH` environment variable.

Installation

Distribution .zip file

The PingDataGovernance Policy Administration GUI components are distributed as a `.zip` file. The following directories and files are included in this distribution:

- `PingDataGovernance-PAP/`
  - `admin-point-application/` – Administration Point executable scripts and example configuration files.
  - `admin-point-installer/` – Configurator application, which produces the configuration file `configuration.yml` after prompting the user.
  - `bin/` – Contains helper scripts to set up, start, and stop the server.
  - `config/` – Contains the configuration file `configuration.yml`, which is read when the helper scripts are used to start the server.
    - `templates/` – Contains templates from which the system generates `configuration.yml`.
    - `configuration.yml` – Contains the server configuration. This file is generated after `bin/setup` is run.
  - `logs/` – Contains the log files that are populated during server execution.
  - `resource/` – Contains Ping policy files that can be loaded into the server.

- `INSTALL` – README file.

Install and run the administration GUI

About this task

The PingDataGovernance PAP is distributed with executable scripts to run the application in Microsoft Windows or Linux/UNIX operating systems. After extracting the contents of the distribution `.zip` file, locate the scripts for starting the Administration Point in the `PingDataGovernance-PAP/bin` folder.

Regardless of whether you install the PAP on Windows or Linux/UNIX, the script runs the PAP by using an embedded H2 database. An embedded UI is served on `http://localhost:8080/` by default, or at the URL that was configured during setup. The REST API is available at `http://localhost:8080/api/` by default, or at the URL that was configured during setup, with Swagger documentation available through the UI.

Log files are generated in the working directory.

Windows

About this task

To generate the `configuration.yml` file, run the following script:

```
PingDataGovernance-PAP\admin-point-installer\bin\admin-point-
configurator.bat
```

To run the PingDataGovernance Policy Administration GUI as a Java application, run the following script:

```
PingDataGovernance-PAP\admin-point-application\bin\admin-point-
application.bat
```

Install as a Windows service

About this task

To install the PAP as a Windows service, extract the contents of the distribution `.zip` file to the appropriate directory, such as `PingDataGovernance-PAP`. The directory `PingDataGovernance-PAP\admin-point-application\bin\` contains the following batch scripts:

- `admin-point-application.bat` – Runs the PAP as a java Application.
- `InstallSymphonicAdminPoint.bat` – Installs the PAP as a Windows service named *SymphonicAdminPoint*. The life cycle of the PAP – start, stop, and restart – can be managed through `services.msc`, the standard tool that is provided with Windows.

  To start the PAP automatically during system startup, use **Windows Services Manager** to set the **Startup Type** to `Automatic`.
- `UninstallSymphonicAdminPoint.bat` – Uninstalls the PAP service from the system.

Linux and UNIX

About this task

To run the PAP with the default configuration, run the `start-server` script, as follows:

```
PingDataGovernance-PAP/bin/start-server
```

By default, the `start-server` script runs the application as a server and uses the `configuration.yml` file that is found in the `PingDataGovernance-PAP/config` folder. By using the `setup` script in

`PingDataGovernance-PAP/bin`, a new version of `configuration.yml` is generated, reflecting the appropriate configuration changes.

Upgrade the PingDataGovernance Policy Administration GUI

About this task

This section describes the steps by which a server administrator can upgrade the PingDataGovernance Policy Administration GUI when a new version is released. In the example scenario, the administrator transfers an existing server's configuration to an upgraded version.

The server uses a configuration YAML file and an H2 database, both of which utilize regular file-system files that can be transferred between installations. Similarly, the product license, along with any SSL certificate keystores that the configuration uses, can also be transferred between installations.

Important: Log files that are associated with the original installation are not transferred to the new server.

Steps

1. Navigate to the PingDataGovernance Policy Administration GUI installation directory.

   ```
   $ cd /opt/dg-prev/PingDataGovernance-PAP
   ```
2. Stop any servers that might be running.

   ```
   $ bin/stop-server
   ```
3. Copy the relevant server files to a temporary location for transferring.

   ```
   $ cp config/configuration.yml /tmp
   $ cp keystore /tmp
   $ cp Symphonic.mv.db /tmp
   $ cp PingDataGovernance.lic /tmp
   ```

   Note: If the original installation was not using SSL, or if it was using an existing SSL certificate in a different location,the keystore file might be absent.
4. Extract the contents of the PingDataGovernance PAP distribution `.zip` file to the appropriate location.

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

   This step assumes that the distribution file is located in the user's `HOME` directory, and that the destination folder, `/opt/dg-next`, already exists and is writable.
5. Navigate to the newly created directory.

   ```
   $ cd /opt/dg-next/PingDataGovernance-PAP
   ```
6. Move the original files to the new location.

   ```
   $ mv /tmp/configuration.yml config/
   $ mv /tmp/keystore .
   $ mv /tmp/Symphonic.mv.db .
   $ mv /tmp/PingDataGovernance.lic .
   ```
7. Start the new server.

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

Chapter

2

Configure the PingDataGovernance Policy Administration GUI

Topics:

The Administration Point application is built by using the Dropwizard framework (*https://www.dropwizard.io/*), which features a YAML configuration that is normally called `configuration.yml`. For information about setting the configuration file, see *Install and run the administration GUI* on page 6.

The distribution bundle for the PingDataGovernance Policy Administration GUI, which is powered by Symphonic, contains example Dropwizard configuration files. To gain an understanding of the configuration file structure and the manner in which the various configuration options fit together, refer to these files.

Common configuration

Authentication

PingDataGovernance Policy Administration GUI supports single sign-on (SSO) authentication by using identity provider solutions that are compatible with OpenID Connect (OIDC) or Lightweight Directory Access Protocol (LDAP). Each option defines a sign-in flow that enables the PAP to authenticate a user and obtain the user name and other information about him or her.

Enable authentication by selecting **OpenID Connect** when running the setup script.

OIDC

OIDC, or implicit flow, is an authentication layer that sits on top of the OAuth 2.0 open standard.

Enable and set up OIDC authentication

When configuring the identity provider (IdP), make certain that implicit flow is enabled. Additionally, the list of valid redirect URIs must include the callback URI for the UI, `http://ui-server:<port>/idp-callback`.

The PAP requires the following pieces of information from the IdP:

- `client_id` – Represents the ID that is referenced in the URI and tokens.
- OIDC configuration endpoint (`http://idp-server:<port>/.well-known/openid-configuration`) – Used to obtain information like the IdP's OpenID authentication workflow endpoints and token signing keys. The IdP must have CORS enabled on the OIDC discovery endpoint and userinfo endpoint.
- Authentication scope – Can be set explicitly in the configuration. The minimum required value is `openid`, and the default value is `openid email profile`.

The following table identifies the key authentication configuration properties in the file `configuration.yml`:

Property	Description	Notes
	Core	
`Authentication.Protocol`	Sets the authorization type for the backend.	Set to `oidc` for OpenID Connect.
`Authentication.oidc ConfigurationEndpoint`	Points to the IdP's OIDC configuration endpoint.	The PingDataGovernance Policy Administration GUI automatically fetches the authorization endpoints and signing keys from this location.
`Authentication.oidc ClientId`	OIDC client name.	Client name of PingDataGovernance Policy Administration GUI; set by the IdP.
`Authentication.redirectUri`	OIDC redirect URL back to the application.	Set this value in `configuration.yml` to the frontend host and post number, followed by `/idp-callback`.

Property	Description	Notes
`Authentication.scope`	OIDC identity server scopes to request.	Contains mandatory scopes that include the information about authenticated users that the PingDataGovernance Policy Administration GUI requires, like `openid email profile`.
	UI	
`authType`	Specifies the type of authorization that the front end uses.	Set to `oidc` for OIDC, and to `credentials` for a test logon attempt.
`configurationEndpoint`	Specifies the location from which the front end obtains its user information.	Same value as `Authentication.oidc ConfigurationEndpoint`.

The following code shows an example backend PAP configuration:

```
#OpenID Connect authentication – Example backend configuration
core:
  Authentication.Protocol: "oidc"
  Authentication.redirectUri: "http://ui-server:<port>/idp-callback"
  Authentication.oidcConfigurationEndpoint: "https://identity-
server:<port>/.well-known/openid-configuration"
  Authentication.oidcClientId: "pingdatagovernance-pap"
  Authentication.scope: "openid email profile"
```

Similarly, the following code shows an example frontend PAP configuration:

```
#OpenID Connect authentication – Example frontend configuration
ui:
  authType: oidc
  configurationEndpoint: "https://identity-server::<port>/.well-known/
openid-configuration"
  clientId: "pingdatagovernance-pap"
```

Log on and off

When an unauthenticated user attempts to access the PingDataGovernance Policy Administration GUI frontend application endpoint, he or she receives a response status code of 401 (unauthorized), and the front end is directed automatically to the IdP's logon page. After the IdP's logon mechanism authenticates a valid user, he or she is redirected back to PingDataGovernance Policy Administration GUI. When using the REST endpoint unauthenticated, the response status code is again 401 (unauthorized), and the `Location` header contains the identity server's authentication URI, decorated with everything the user needs to authenticate, including the specified `redirect_uri`, which is performed with the `id_token` and `access_token` in the fragment portion of the URI.

When a user logs off from PingDataGovernance Policy Administration GUI, a call is also made to log off the user from the IdP's end.

LDAP

LDAP is a protocol for querying items in a directory-service provider like Active Directory.

Enable and set up LDAP authentication

When using LDAP over TLS, add the required keys to the Java truststore. The file `configuration.yml` contains the information necessary to authenticate a user and to decorate the request with user properties through LDAP.

Property	Description	Notes
	Core	
`Authentication.Protocol`	Sets the authentication type for the backend.	Set to `ldap` for LDAP.
`Authentication.ldap.uri`	URI of the LDAP identity server.	Address of the LDAP identity server, such as `ldap://ldap.forumsys.com`.
`Authentication.ldap.connectionTimeout`	Number of milliseconds to wait on a connection to an LDAP server before failing.	Defaults to a value of `3000` (3 seconds).
`Authentication.ldap.readTimeout`	Number of milliseconds to wait on a query to an LDAP server before failing.	Defaults to a value of `3000` (3 seconds).
`Authentication.ldap.negotiateTls`	TLS strategy	Set to none for no SSL, and to `strict` or `fail` if SSL is unavailable.
`Authentication.ldap.userKey`	LDAP key that indicates the user name.	Set to the key that identifies the user name in the LDAP identity server, like `cn` or `uid`.
`Authentication.ldap.BaseUserDn`	Base DN for users on the LDAP server.	DN prefix for all users, such as `ou=users,dc=example,dc=com`.
	UI	
`authType`	Specifies the type of authorization that the front end uses.	Set to `ldap` for LDAP, and to `credentials` for a test logon attempt.

The following code shows an example backend LDAP configuration:

```
#LDAP authentication – Example backend configuration
core:
  Authentication.Protocol: "ldap"
  Authentication.ldap.uri: "ldap://ldap-server"
  Authentication.ldap.connectionTimeout: 3000
  Authentication.ldap.readTimeout: 3000
  Authentication.ldap.negotiateTls: "strict"
  Authentication.ldap.userKey: "uid"
  Authentication.ldap.BaseUserDn: "dc=example,dc=com"
```

Similarly, the following code shows an example frontend LDAP configuration:

```
#LDAP authentication – Example frontend configuration
ui:
  authType: ldap
```

Log on and off

Because basic authentication is used to authenticate through LDAP, we recommend using a TLS connection.

When a user attempts to log on with invalid credentials or without basic authentication, he or she receives a response status code of 401 (unauthorized). Credentials must be valid LDAP credentials that use the key `Authentication.ldap.userKey`, which resides in `Authentication.ldap.BaseUserDn`. To log off a user, remove the basic authorization that is attached to all requests.

Chapter

3

Policy administration

Topics:

About this task

This section introduces the features of the PingDataGovernance Policy Administration Point (PAP), which is powered by Symphonic, and provides instructions for creating access-control policies that reflect your business requirements. It also includes a tour of the various concepts that are involved in modelling policies. To get started with the PingDataGovernance PAP, complete the following tasks:

Steps

1. Log on to the system.

   For demonstration environments, use the following default credentials:

   - User name – `admin`
   - Password – `password123`
2. Create a branch.

   This branch functions as the base store for your policies and other entities.
3. Define the trust framework.

   This step lets you define the elements that form the building blocks of your policies, such as the WHO, WHAT, WHERE, WHY, and WHEN.
4. Define your policies and policy sets.

   Build your policies to reflect your business needs.
5. Test the policies and policy sets.

   Run scenario-based unit, system, and regression tests on your policies.
6. Conduct an analysis.

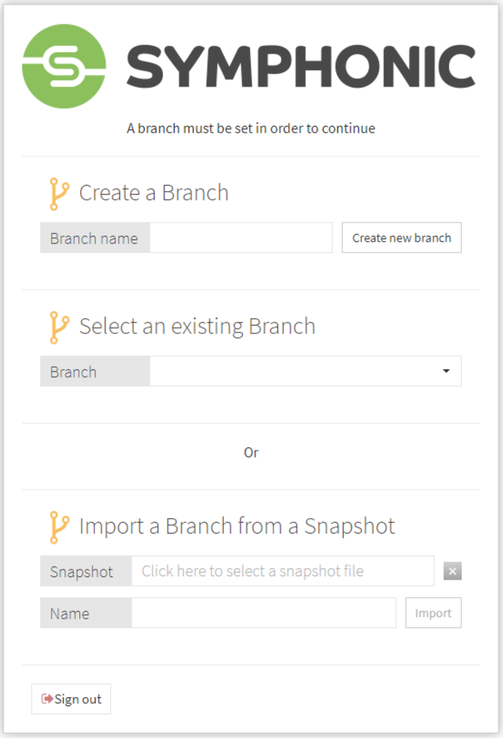
   Carry out detailed, cross-policy analysis to identify potential conflicts, shadows, redundancies, and failure impacts.
7. Commit your changes.

   This step creates a *snapshot*, which provides an immutable representation of your policy map at a specific moment.
8. Create a deployment package.

   This step creates a deployment file that can be deployed to PingDataGovernance Server for use in production environments.
9. Deploy the package.

Branches and snapshots

The PingDataGovernance Administration Point embraces similar principles to general software source control, and begins with the creation of a root branch. When you first deploy the PingDataGovernance Policy Administration GUI, the `Branches` repository is empty, and the application forces you to create or import a branch to continue using the product.


A branch must be set in order to continue

SYMPHONIC
Create a Branch
Branch name
Create new branch
Select an existing Branch
Branch
Or
Import a Branch from a Snapshot
Snapshot
Click here to select a snapshot file
Name
Import
Sign out

Create a new branch

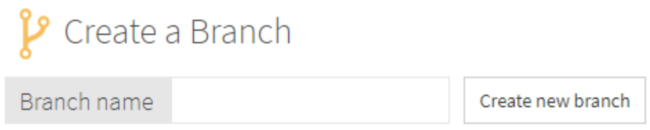
About this task

Branch names must be unique within the PAP.

Create a branch during startup

Steps

1. On startup, type a unique branch name in the **Create a Branch** text box.



2. Click **Create new branch**.

Create a branch from the Version Control tab

About this task

On the **Version Control** tab, you can create either of the following branch types:

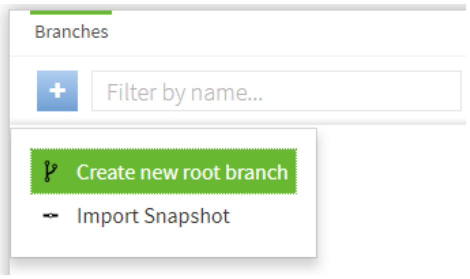
- Root branch.
- Child branch that is created from an existing branch.

Root branch

To create a root branch, perform the following steps:

Steps

1. Click **+** and select `Create new root branch`.



2. In the **Name** text box, type a unique name for the branch.
3. Click **Save Branch**.

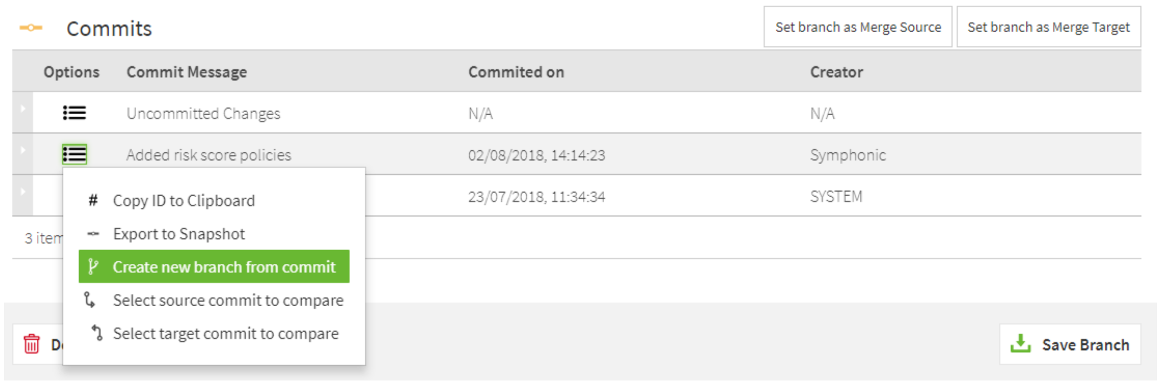
Next steps

Child branch

To create a child branch from an existing branch, perform the following steps:

1. Select the snapshot from which to create the branch.

   The snapshot must be committed. To branch from uncommitted changes, make certain to commit them before proceeding.
2. Click **+** and select `Create new branch from commit`.


Commits
Set branch as Merge Source
Set branch as Merge Target
Copy ID to Clipboard
Export to Snapshot
Create new branch from commit
Select source commit to compare
Select target commit to compare
3 item
Save Branch

Options	Commit Message	Commited on	Creator
	Uncommitted Changes	N/A	N/A
	Added risk score policies	02/08/2018, 14:14:23	Symphonic
		23/07/2018, 11:34:34	SYSTEM

3. In the **Name** text box, type a unique name for the branch.
4. Click **Save Branch**.

A new sub-branch is created from the selected snapshot.

Import a branch from a snapshot

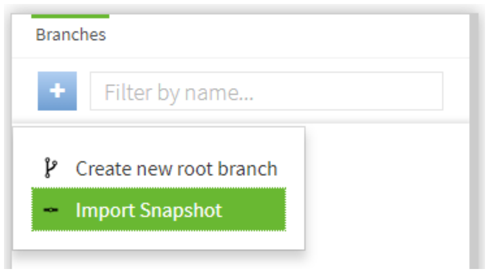
About this task

Branches can be imported from existing snapshot files, which contain the entities and policies from an existing branch, and can be shared like other files. Snapshots provide a useful method for sharing and restoring trust framework definitions and policies across users and environments.

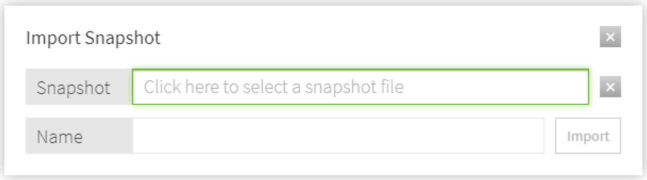
To import a branch from a snapshot, perform the following steps:

Steps

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



2. Select the appropriate snapshot file.
3. In the **Name** text box, type a unique name for the snapshot.
4. Click **Import**.



Delete a branch

About this task

To delete a branch, perform the following steps:

Steps

1. Select the branch to delete.
2. Click **Delete Branch**.
3. Confirm your choice to delete the branch.

   Unless a snapshot has been taken, deleting a branch is an irreversible process. All changes made after the snapshot was exported will be permanently lost.

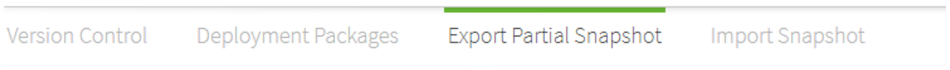
Next steps

To recover data from a deleted branch, load a snapshot that has been exported from the branch, if one exists.

Partial import and export

About this task

The partial import/export feature enables the packaging of a subset of the policies or trust framework entities for export. The policies and entities can then be imported as a new branch or into an existing branch.



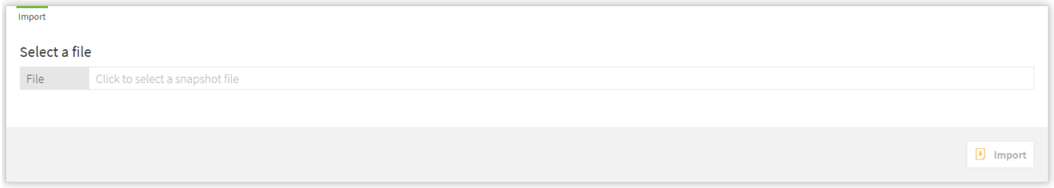
Partial import

About this task

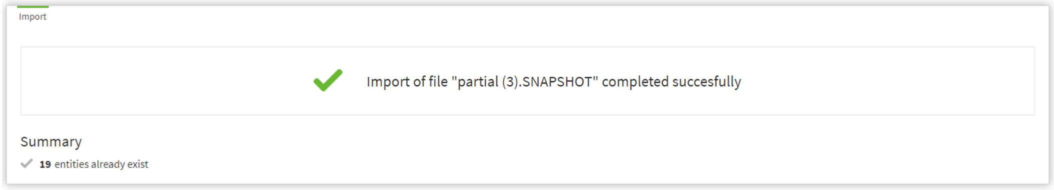
The process of importing a partial snapshot adds or updates all of the entities into the selected branch. To import a partial snapshot, perform the following steps:

Steps

1. Select the snapshot file to import.

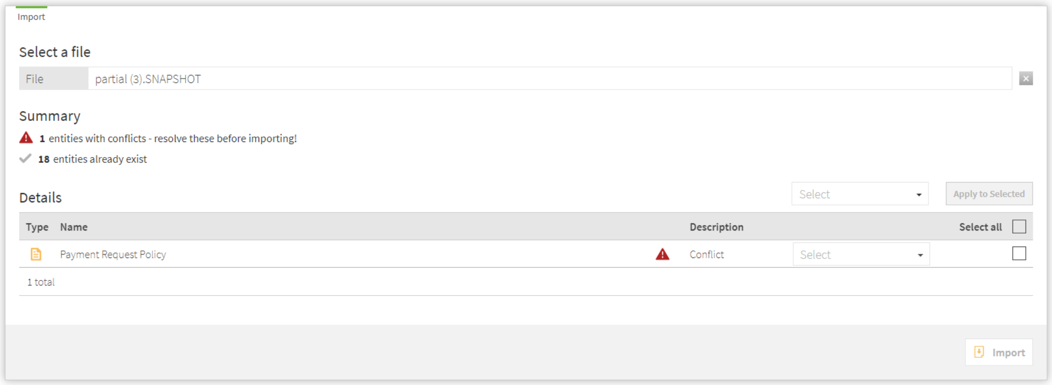


2. Click **Import**.
The **Summary** page details the results of the import.


Import of file "partial (3).SNAPSHOT" completed succesfully

Summary
19 entities already exist

If the import function detects conflicts between the current branch version and the snapshot version of the same entity, the **Merge Conflict Resolution** page opens.


Select a file
File: partial (3).SNAPSHOT

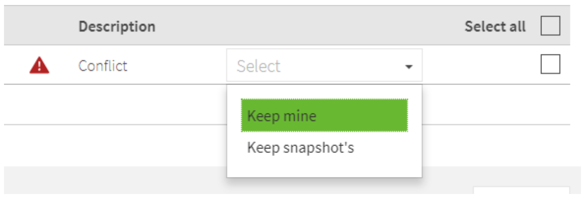
Summary
1 entities with conflicts - resolve these before importing!
18 entities already exist

Details

Type	Name		Description		Select all
	Payment Request Policy		Conflict	Select	

1 total

3. If the **Merge Conflict Resolution** page opens, select for each conflict whether to keep your local changes or to overwrite them with the values from the imported snapshot.

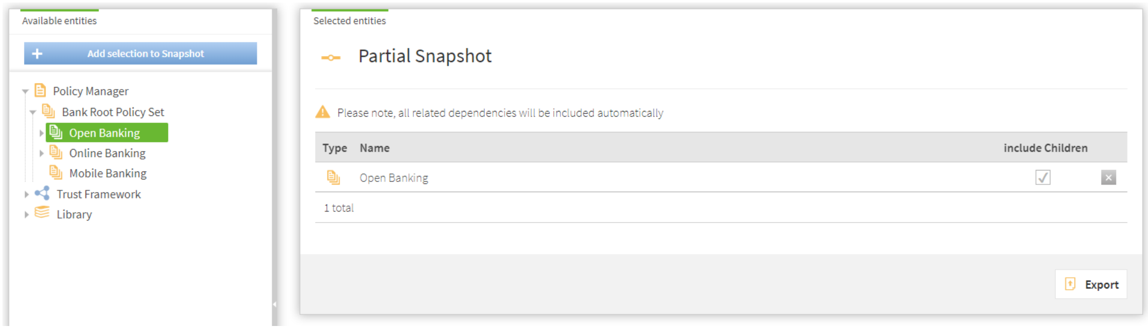


4. Click **Import**.

Partial export

About this task

A partial export lets users build an export snapshot of selected entities from a combination of the trust framework, policy sets, and library set.



Please note, all related dependencies will be included automatically

Type	Name	include Children
	Open Banking	✓

1 total

To export a partial snapshot, perform the following steps:

Steps

1. On the **Partial Export** page, select the snapshot file to export.
2. Click **Add selection to snapshot**.
   The entity is added to the table.

   Note: Because all dependencies are included automatically in the exported snapshot, you do not need to select each dependency individually.
3. Click **Export**.

Trust framework

The trust framework tool lets you define the entities within your organizations so that you can build policies about them at a later time. The information that your policies will express must be defined in the trust framework. As a result, the policies that you create will be strongly typed to the definitions in your trust framework.

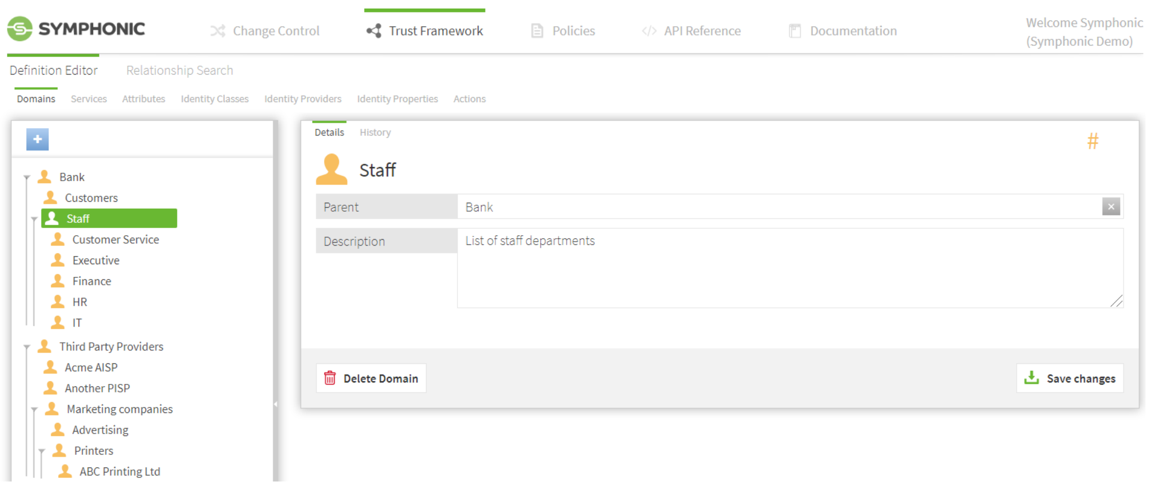
Definitions are broken into the following types:

- *Domains*
- *Services*
- *Actions*
- *Attributes*
- *Identity properties*
- *Identity providers*
- *Identity classifications*
- *Named conditions*

The following sections describe these definitions in more detail.

Domains

Use the **Domains** section to define the organizational structure as well as any other organizations with which you intend to interact, and on which you want to specify authorization policies. Keep the domain ontology clean and simple. If you want additional levels of granularity, you can extend it later.



Staff

Parent	Bank
Description	List of staff departments

When importing domains from an existing organizational directory, like Active Directory, make certain to exclude all unnecessary and redundant entities.

Services

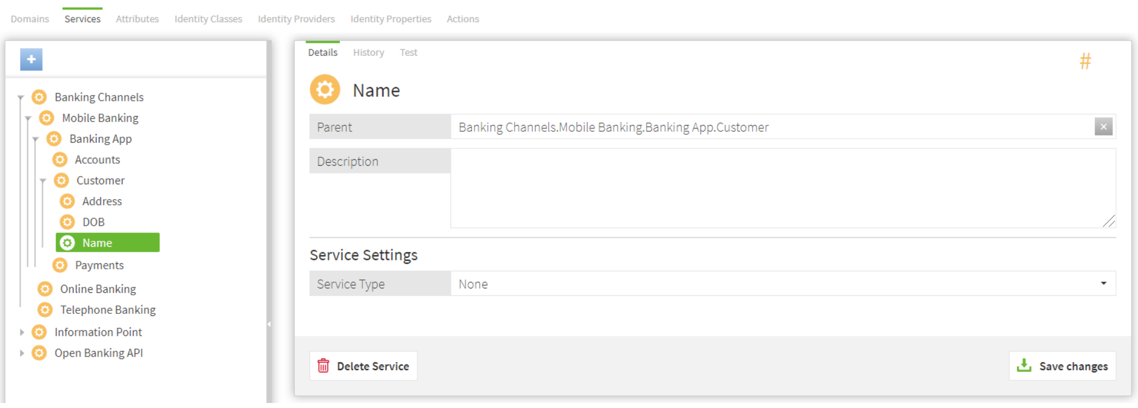
Use the **Services** section to define the following types of services:

- Resources that your policies protect by regulating access to them.
- Information points that provide data for attributes that inform policy decisions.

The following sections describe these services in more detail.

Resources

For a resource, define only the top-level fields, like **Name**, **Parent**, and **Description**.



Name

Parent	Banking Channels.Mobile Banking.Banking App.Customer
Description	

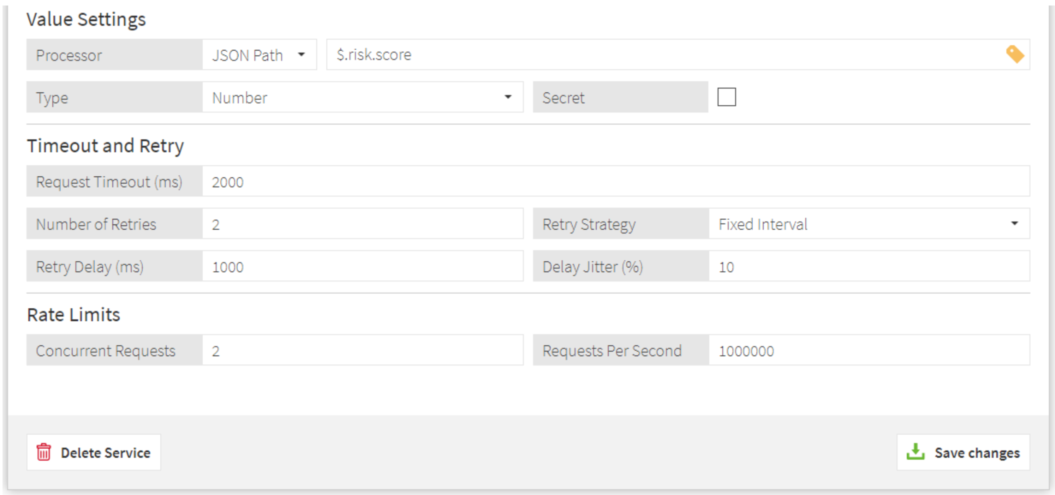
Service Settings

Service Type	None

Unless you plan to use the service as an information point, leave the **Service Type** as none.

Information points

The process of setting up services as information points makes use of the PingDataGovernance Policy Administration GUI's service connectors. Make your selection from the **Service Type** drop-down list and specify the relevant configuration values in the predefined fields.


Value Settings

Processor	JSON Path	$.risk.score	
Type	Number	Secret	☐

Timeout and Retry

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

Rate Limits

Concurrent Requests	2	Requests Per Second	1000000

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

When existing services return more information than is required, or when you need to convert information to a different format, support is provided for JSON Path, XPath, and the Spring Expression Library (SpEL).

For examples, see the following sections.

Common settings

The following settings apply to all service types:

- Request Timeout – Number of milliseconds that the PAP waits for the request to complete. If this value is met before a successful response is received, the request is canceled. If retries are configured, the request is attempted again. If all requests fail to complete in time, the result is an error that represents the timeout.
- Number of Retries – Number of times that the PAP attempts a request again, after the initial request fails or times out. This number is added to the initial attempt. For example, if you set this value to 2, the PAP attempts a maximum of three times. Set to `0` to try a request only once.
- Retry strategy:
  - Fixed Interval – Between each attempt to perform a service request, the PDP waits the amount of time that is specified by the Retry Delay. This option is the default retry strategy.
  - Exponential Backoff – The PDP waits for an exponentially increasing amount of time between attempts to perform a service request.
- Retry Delay – If the retry strategy is Fixed Interval, the Retry Delay value specifies the number of milliseconds that the PDP waits between request attempts.

  If the retry strategy is Exponential Backoff, the PDP multiplies the Retry Value by 2^n, where *n* represents the number of retries that have already been made. For example, if the Retry Delay is set to `1000`, and if the retry strategy is Exponential Backoff, the PDP makes the initial request, then waits 1000ms before making a second attempt, 2000ms before a third attempt, 4000ms before a fourth attempt, and so on.
- Delay Jitter – Percentage value that indicates the amount of variability to apply to the Retry Delay on each attempt. For example, if the Delay Jitter is set to `10%`, the delays in the previous example are 1000±100ms, 2000±100ms, 4000±100ms, and so on.

RESTful services

The PDP can send and receive text, JSON, and XML content with `GET`, `POST`, `PUT`, `DELETE`, and `HEAD` requests to HTTP services. HTTP authentication is supported by using a simple user name and password or by using an OAuth2 token.

Custom headers can be sent with any request. Additionally, requests can be made dynamic in various ways by interpolating attribute values into various parameters.

Core settings are as follows:

- URL Format – URL for the REST endpoint that will be accessed. Attributes can be interpolated anywhere in the URL, but attribute values are not escaped. If necessary, specify this value in the attribute definition.
- Http Method – Method to send in the HTTP request.
- Content Type – `Content-Type` header to send, which relates to the body of the request.
- Body – Body to send with the request. Attributes can be interpolated anywhere in the body, with no escaping.

Authentication

The **Authentication** picker selects the HTTP authentication type, which corresponds to an authorization header that is sent with the request:

- None – No authorization header is sent. This value is the default HTTP authentication type.
- Basic – Provides options for attributes whose values are sent as the user name and password of an HTTP request with Basic authentication.
- OAuth2 – Reveals a token selector. The selected attribute is sent as the authorization token in an HTTP request with Bearer authentication.

Headers

Any number of custom headers can be added to a request. Although header names are fixed strings, their values can be constants or attribute values. To toggle between a constant and an attribute, click `C/A` next to the appropriate header value.

Value settings

For a RESTful service, value settings describe the expected response from a request. If the response does not require preprocessing, leave **Processor** set to `None`, and set the **Type** value as follows:

- For plain text, specify `String`.
- For JSON, specify `JSON`.
- For XML, specify `XML`.

If the response requires preprocessing, perform the following steps:

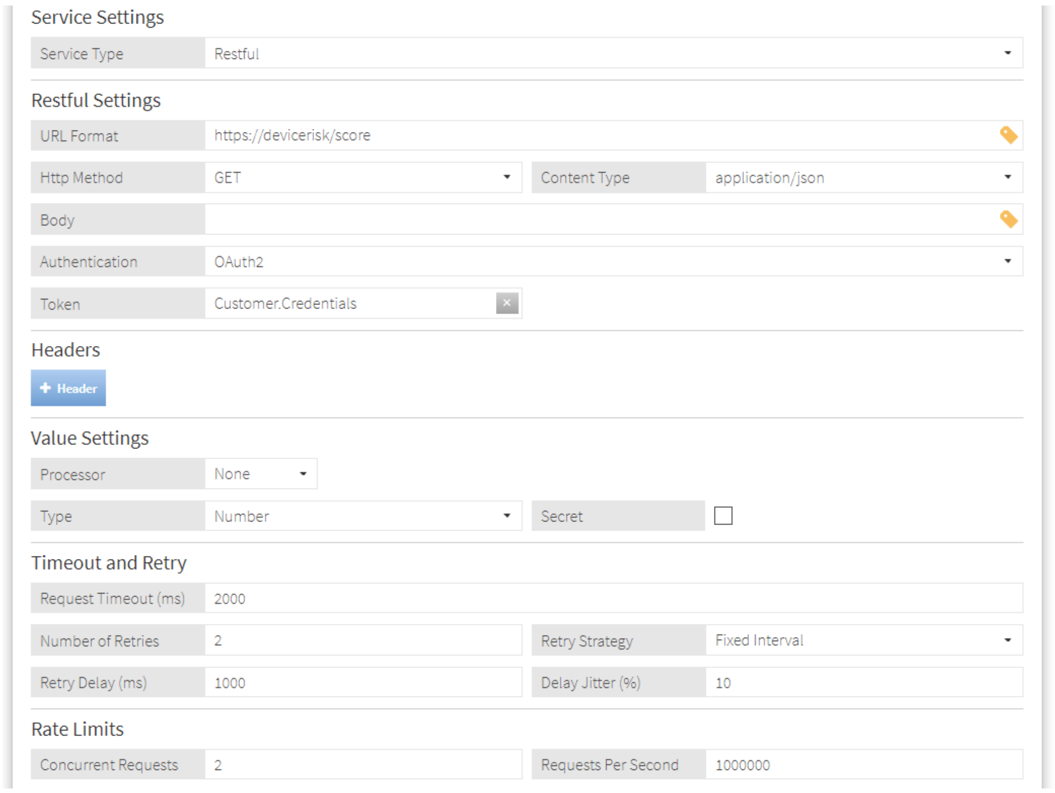
1. Select the required SpEL, XPath, or JSONPath processor.
2. Enter the appropriate expression.
3. In the **Type** field, specify the result type of the expression.

For example, if the RESTful service returns the following JSON body, and if a JSONPath processor is selected with the expression `$.name`, the **Type** must be `String`, and the final value for the service is `John Smith`.

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

Secret

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



Service Settings

Service Type	Restful

Restful Settings

URL Format	https://devicerisk/score		
Http Method	GET	Content Type	application/json
Body			
Authentication	OAuth2		
Token	Customer.Credentials		

Headers

Value Settings

Processor	None		
Type	Number	Secret	☐

Timeout and Retry

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

Rate Limits

Concurrent Requests	2	Requests Per Second	1000000

Database information points

The PDP can connect directly to relational databases and can perform SQL queries.

S=Database connection string

This setting defines the JDBC connection string that is used to connect to one of the following databases:

- H2
- Oracle
- PostgreSQL
- Microsoft SQL Server

To use a particular database, make certain the appropriate JDBC driver is included in the classpath. To use a database that is not included in the previous list, add the appropriate JDBC driver JAR file to the PingDataGovernance Policy Administration GUI classpath.

Example JDBC connection strings

```
jdbc:h2:~/database
jdbc:postgresql://dbhost/dbname
jdbc:oracle:thin:@localhost:1521:SID
```

Attributes can be interpolated anywhere in the connection string.

SQL

The **SQL** setting contains the SQL query that is sent to the database. To prevent SQL injection attacks, attribute interpolation is supported in value positions by default. In the following example, the attribute placeholder becomes a SQL placeholder when the query is prepared, with the attribute value provided as a parameter:

```
SELECT user,email FROM people WHERE userid == {{UserId}}
```

Attributes can be interpolated anywhere in a SQL string by using the `unsafe` modifier. For example, if the Attribute Limit evaluates to a SQL snippet that limits the results of a query, such as `LIMIT 20`, the following SQL could be used in a database service:

```
SELECT user,email FROM people WHERE dateOfBirth > {{StartDate}} {{Limit |
 unsafe}}
```

Warning: Depending on the source of the attribute value, this approach might open the SQL query to injection attacks.

Results

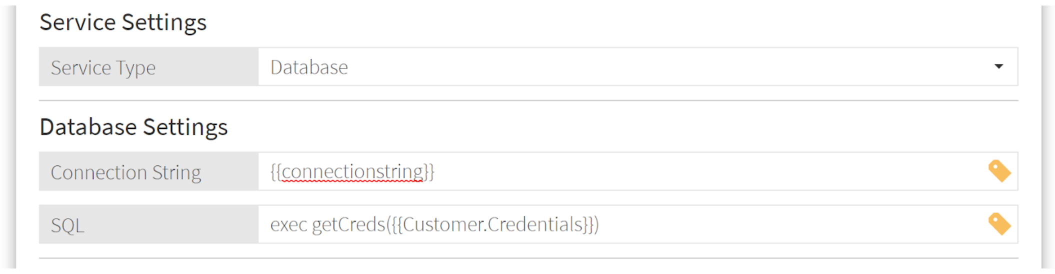
Results from database queries are converted into a two-dimensional table and rendered in XML format inside a `root` node. The following table features columns named **firstname** and **lastname**, which are populated with example data that has been queried from a hypothetical database.

firstname	lastname
Fred	Flintstone
Barney	Rubble

The following XML document shows the service result for this example:

```
<root>
  <row>
    <column name="firstname">Fred</column>
    <column name="lastname">Flintstone</column>
  </row>
  <row>
    <column name="firstname">Barney</column>
    <column name="lastname">Rubble</column>
  </row>
</root>
```

If the result type of such a service definition is set to `XML`, attributes can use XPath queries to pick individual or collections of values from the XML data.



Service Settings

Service Type	Database

Database Settings

Connection String	{{connectionstring}}
SQL	exec getCreds({{Customer.Credentials}})

LDAP services

The PDP can make LDAP queries to resolve attribute values.

Configuration

Multiple settings are required to configure an LDAP service. This section uses a publicly available LDAP service as an example.

Host and port

The following values provide an example hostname and port number for an LDAP server:

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

User name / Bind DN and password

The following values provide an example bind DN and password for an LDAP server:

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

Search base DN / LDAP filter

The following settings define an LDAP query that can be made:

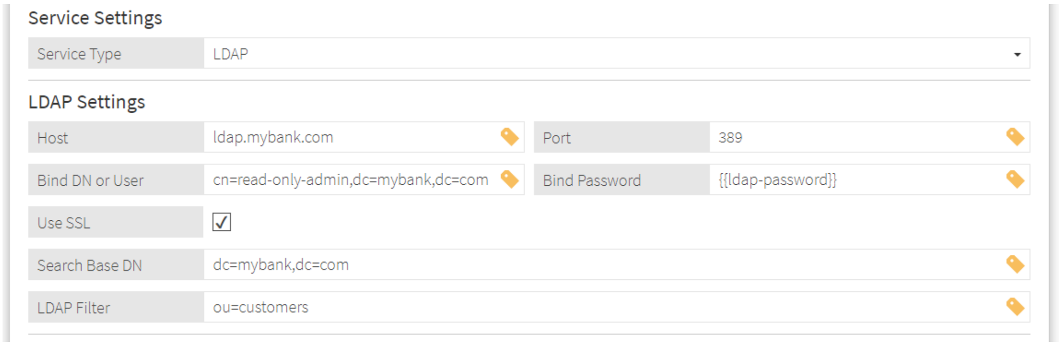
```
Search Base DN: dc=example,dc=com
LDAP Filter: ou=mathematicians
```

Results

Because the result of an LDAP query is converted to an XML document, set the **Service Value Type** to XML. The following example shows the resulting document for the previous query:

```
<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>
```

Use XPath processors to extract individual parts or collections of data from a resulting XML document.



Service Settings

Service Type	LDAP

LDAP Settings

Host	ldap.mybank.com	Port	389
Bind DN or User	cn=read-only-admin,dc=mybank,dc=com	Bind Password	{{ldap-password}}
Use SSL	✓		
Search Base DN	dc=mybank,dc=com		
LDAP Filter	ou=customers		

Camel services

In addition to retrieving information from HTTP, LDAP, and SQL information points, the PAP also supports retrieving information from endpoints that are supported by the *Apache Camel* enterprise integration platform.

To view the full list of systems supported, visit the *Components* page on the Apache Camel website. Camel components are configured by using a combination of URI, headers, body, and configuration settings. The appropriate values for each setting depend on the component that is being used. Consult the documentation on the Camel website for the particular component that you plan to use.

URI

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

Attribute values can be interpolated into a URI.

Headers

Further information can be sent to an external information point by using Camel headers. If the component to which you want to connect uses headers, locate the instructions for the component on the Apache Camel website.

Attribute values can be interpolated into headers.

Body

Some Camel components operate on a message body, which can be provided by using this setting. If the component to which you want to connect to requires a message body, locate the instructions for the component on the Apache Camel website.

Attribute values can be interpolated into the body.

Configuration

Some Camel components require you to configure helper components, which are specified by using the *Groovy* scripting language to write a *Spring Bean* configuration block. For information about writing such configurations, view this *document*.

Attribute values cannot be interpolated into the configuration.

Note: The Camel JDBC component makes use of the headers and body settings, and requires that a JDBC data source be set up in the Camel configuration setting.

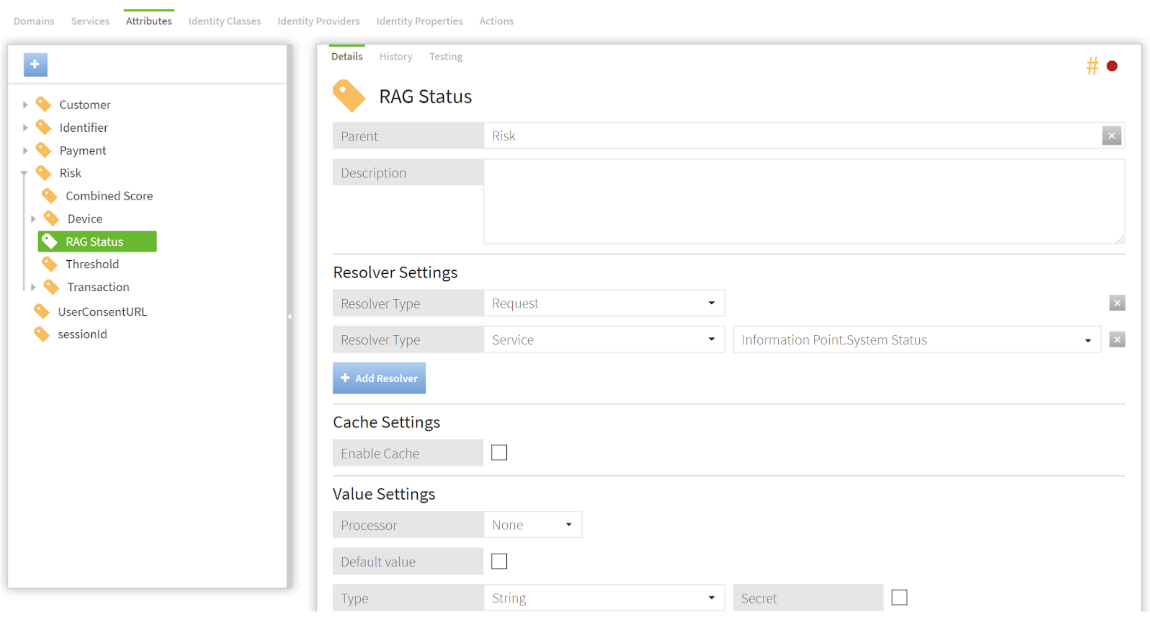
Attributes

Attributes provide the context that enables fine-grained policies. They retrieve data from multiple information endpoints and permit the inclusion of values and results inside the rules of a policy. We recommend contemplating the structure and naming conventions of your attributes so that policy writers and editors can build policies without a deep understanding of the underlying data endpoints. The intention is to abstract any complexity and expose the attributes in terms that business users and policy builders can understand.

Create an attribute

About this task

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



RAG Status

Parent: Risk

Description:

Resolver Settings

Resolver Type: Request

Resolver Type: Service — Information Point.System Status

Cache Settings

Enable Cache: ☐

Value Settings

Processor: None

Default value: ☐

Type: String — Secret: ☐

Resolver settings

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

Each attribute can feature multiple resolver types that are resolved in the order in which they are defined. These can be reordered by dragging and dropping.

The following resolver types are supported:

- Request – Looks inside an authorization request and determines whether the caller is providing the attribute.
- Service – Uses a trust framework or service endpoint to invoke the service at runtime and to resolve the attribute. To invoke the service, the service might rely on other attributes that are supplied. The PDP handles this resolver type automatically.
- Attribute – Attributes can also be resolved from other attributes, which is useful when your attributes contain multiple pieces of information and you want to create nested or children attributes as subset extracts from them. For example, the `Customer.Name` attribute might return the following JSON representation:

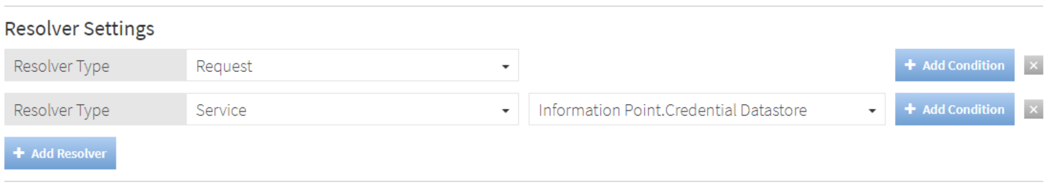
  ```
  { "firstname": "Joe", "middlename": "Bod", "surname": "Bloggs" }
  ```

  The `Customer.Name.Surname` attribute can be created to resolve against the `Customer.Name` attribute, and to use a JSON parser to extract only the `Surname` property of the JSON object.
- System – The PingDataGovernance PAP provides a number of default system attributes that can be used without any additional configuration. For example, the `CurrentDateTime attribute` returns the current system `datetime` value.
- Configuration key – Attributes can be resolved against configuration key items. These key-pair values are defined in the file `configuration.yml` as part of the initial PAP configuration.

Conditional resolvers

All resolver types support the ability to add conditional logic. As a result, the resolver is invoked only under certain defined conditions.

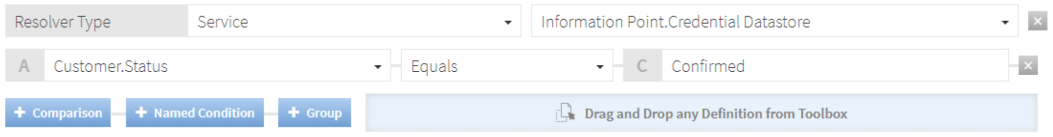
To add a conditional logic builder to a particular resolver, click **Add Condition** next to the resolver item.


Resolver Settings

Resolver Type: Request

Resolver Type: Service — Information Point.Credential Datastore

The following example implies that the Service Resolver `[Information Point.Credential Datastore]` is used only when `Customer.Status = "Confirmed"`.


Resolver Type: Service — Information Point.Credential Datastore

A Customer.Status — Equals — C Confirmed

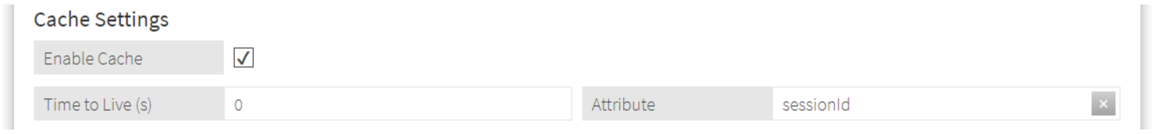
Attribute caching

The PingDataGovernance PAP supports caching for attributes. The ability to cache resolved attributes delivers significant performance gains for the overall decision engine. To ensure optimum configuration, we recommend that you consider this concept carefully.

The cache settings for attributes offer the following approaches:

- Time-based – Allows you to set the amount of time that the cache lives before it expires. This value is also known as the *time to live (TTL)*. If an attribute does not exist in the cache, the decision engine uses the appropriate attribute resolvers to resolve it automatically. After the attribute is resolved, it is added to the cache.

  All subsequent attribute usages leverage the cached value until it expires from the cache, which results in another attribute resolution.
- Attribute-based – Allows the application of an additional scope to the cache. This setting lets you attach the value of an attribute to the scope of the cache, such as `customerId` or, as in the previous example, `sessionid`. The cache is returned only when the `scope` is matched. The following example saves and reuses the cache for a specific `sessionid` only.


Cache Settings

Enable Cache: ✓

Time to Live (s): 0

Attribute: sessionId

Tip: The cache key for a trust framework attribute value includes a hash of the values that are required for it to resolve. The cache key is invalidated automatically whenever one of these values changes. This aggregation of scope parameters guards against inconsistencies between cached values.

Value settings

The **Value Settings** section of services and attributes allows you to dictate the value type and optional processor steps to transform a resolved value. The PingDataGovernance PAP supports the following value processors:

- JSONPath
- XPath
- Spring Expression language (SpEL)

The following sections describe these value processors in more detail.

JSONPath

Use JSONPath to extract data from JSON objects. A service might resolve to the following example:

```
{
   "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, create an attribute that resolves against the service, and set the value processor to `JSONPath` with the following expression:

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

This attribute can be used in conditions or in further attribute resolution. For more reference about JSONPath expressions, visit *JSONPath – XPath for JSON*.

XPath

As the XML equivalent of JSONPath, XPath follows a similar syntax. For more information about XPath expressions, refer to the *W3Schools XPath Tutorial*.

The PingDataGovernance Policy Administration GUI supports *XPath 1.0* only. Later functionality might be unavailable.

SpEL

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

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

To locate the substring, set the processor to `SpEL`, and set the expression as follows:

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

To combine multiple attribute values into a single value, insert attribute values directly into the SpEL expression by wrapping its full name in double curly brackets, as the following example shows:

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

Note:

When interpolating an attribute into a SpEL expression, you can interpret only at a location where the symbol is valid. For example, the following expression is valid because it uses string concatenation:

```
"Hello " + {{User.Name}} + "!"
```

The following expression is invalid because a symbol cannot appear inside a string literal:

```
"Hello {{User.Name}}!"
```

For more information about the Spring Expression language, refer to the *Language Reference* documentation for the Spring Framework.

Default value

Attributes can be given an optional default value in the event that they cannot be resolved. A default value can also be used to encode constant attributes within the trust framework by not setting any resolvers and, consequently, always resolving to the default value.

Actions

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

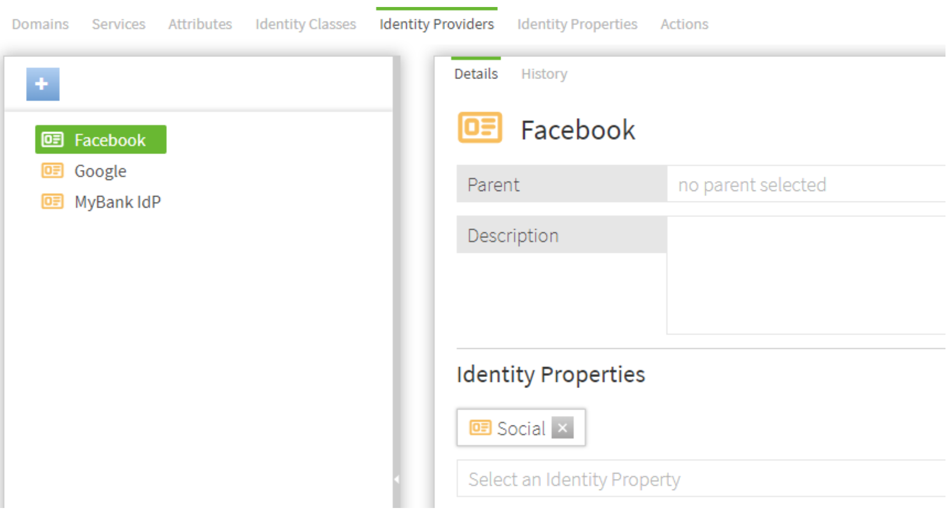
- HTTP:
  - GET
  - PUT
  - POST
  - DELETE
  - PATCH
  - HEAD
  - OPTIONS
- CRUD:
  - Create
  - Read
  - Update
  - Delete

Identity classifications and IdP support

The PingDataGovernance PAP lets you generate intelligent identity classifications that abstract the underlying *identity providers* (IdPs) from their presumed levels of trust. As a result, policies can target levels of trust instead of specific identity providers.

The following elements collectively define these trust levels:

- Identity providers – Define the different IdPs and attach identity properties to them. This approach provides significant abstraction for complicated ecosystems that feature tens or hundreds of participating IdPs.



Facebook

Parent: no parent selected

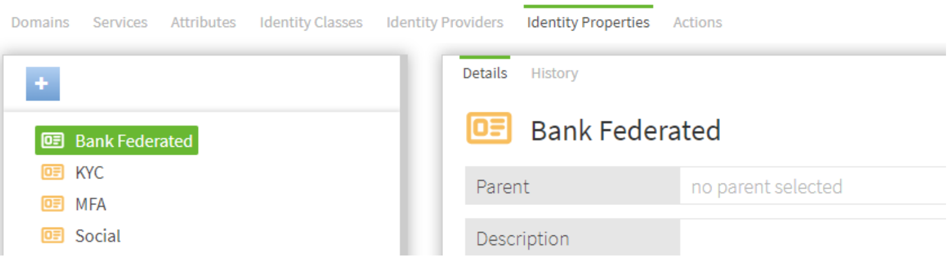
Description

Identity Properties

Social

Select an Identity Property

- Identity properties – Define the items to attach to specific identity providers. These properties map the identity providers to specific Identity Classification levels. For example, a `Social` property can be attached to any IdPs that are classified as *social*.

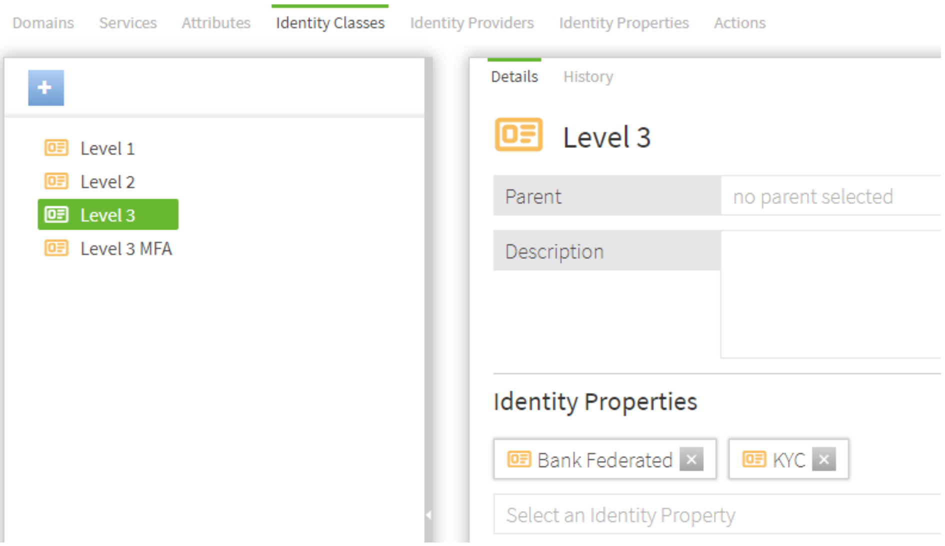


Bank Federated

Parent: no parent selected

Description

- Identity classifications – Create different levels of classification by attaching the properties that are required for an IdP to be considered inside the classification level.



Level 3

Parent: no parent selected

Description

Identity Properties

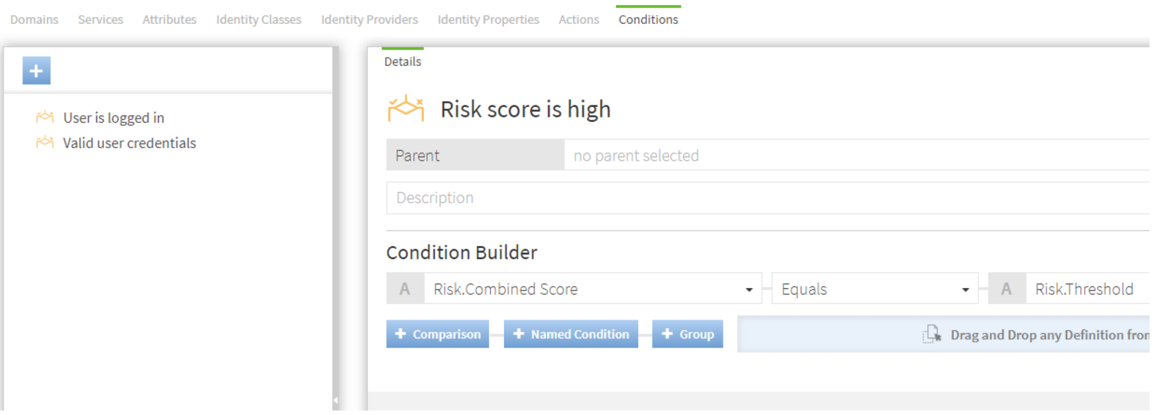
Bank Federated ✕ KYC ✕

Select an Identity Property

Named conditions

Named conditions let you create reusable, conditional logic that helps abstract some of the logical complexity from the people who build policies. Named conditions also minimize repetition throughout policies.

Use named conditions to replace entire conditions, and also as components of complicated condition expressions. To add a named condition within the condition builder, click **+ Named Condition**.



Risk score is high

Parent: no parent selected

Description

Condition Builder

A Risk.Combined Score | Equals | A Risk.Threshold

Policy builders can create their own conditions that coexist with named conditions.

Policy management

The Policy Manager provides the tools for implementing the dynamic, fine-grained access-control policies that govern the use of services and data in your organization. Use the Policy Manager to create policies that answer the question, "Will this resource access request be permitted or denied?” In a traditional role-based access control (RBAC) system, this question can be rephrased as, "Who is the user making the access request, and has the user been assigned a role that can access the resource?"

Although such a policy can be modeled in the PingDataGovernance Policy Administration GUI, the application is essentially an attribute-based access-control (ABAC) system. The question can be rephrased yet again as, "Given the facts that I know about the user, the resource being accessed, what the user wants to do with the resource, my confidence that the user is who she says she is, and any other pertinent facts about the world at this point in time, should the user’s access request be permitted, and must anything else be done besides permitting or denying access?" This lengthy question speaks directly to the power of the PingDataGovernance Policy Administration GUI, which the Policy Manager provides a straightforward way to harness.

Policy sets, policies, and rules

A typical large organization might utilize hundreds or thousands of conditions and constraints around access control. Collectively, these conditions and constraints comprise the business rules that define the circumstances under which certain resources are accessed.

Rules can be grouped together so that people can reason about them without needing to hold them in their heads all at one time. For example, a set of authentication policies might require a user to authenticate to a certain level before he or she can access a particular resource. Another set of policies might gather all of the business rules around accessing the resources of a particular business unit. Yet another set of policies might define the audit processes that are triggered whenever a user attempts to access a set of restricted resources.

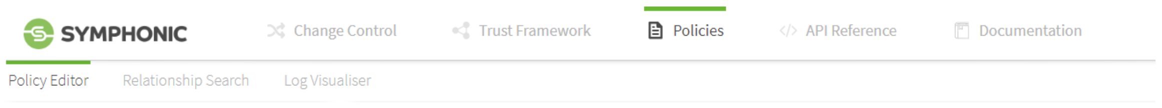
This structure is inherent in the problem domain of resource access control, and is reflected in the following PingDataGovernance Policy Administration entities:

- Policy sets
- Policies
- Rules

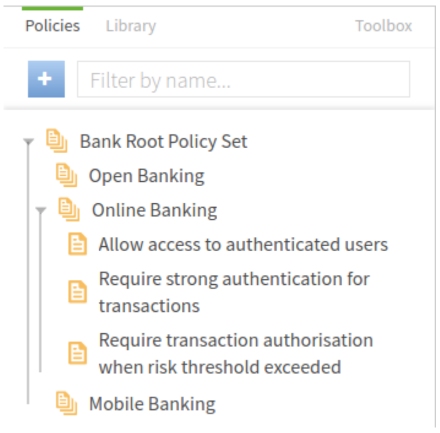
The following sections examine these entities, their properties, and the manner in which they are composed.

Policies and policy sets

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



The left side of the page features the navigation panel's tree structure, which lists the existing policy nodes.


Policies
Library
Toolbox
+
Filter by name...
Bank Root Policy Set
Open Banking
Online Banking
Allow access to authenticated users
Require strong authentication for transactions
Require transaction authorisation when risk threshold exceeded
Mobile Banking

We recommend adding a root policy set to hold all other policy sets. A root policy set is useful when you build a deployment package from the entire Policy tree.

Create policies and policy sets

About this task

To create a policy or policy set, perform the following steps:

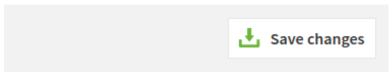
Steps

1. In the navigation panel, click the **Policies** tab.
2. Click **Creation** (blue button).
3. In the pop-up window that opens, select **Policy** or **Policy Set** to start the creation process.
4. Specify a name.

   Although you can give policies and policy sets almost any name that you like, we recommend using relevant and contextual names, especially as the Policy tree grows in size and complexity. Consider the business rule that the name is trying to model, and check whether it adequately represents the operational policies of the organization.

   Inspector
My Basic Policy
Disabled

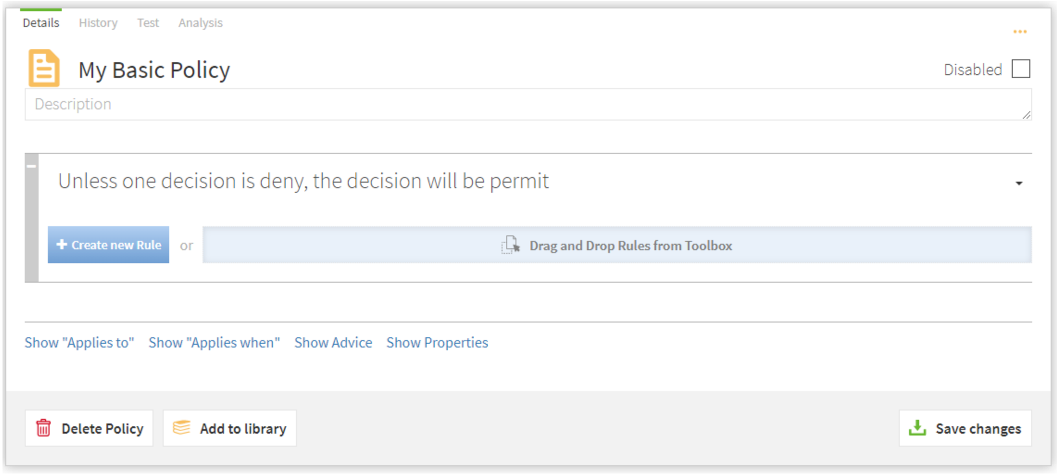
   For example purposes, this document uses a policy named `My Basic Policy`. A red dot in the upper-right corner signifies that, after changing the name, the policy contains unsaved changes. If you try to navigate away from this page, the system prompts you to save or discard your changes.
5. Click **Save Changes**.

Add targets to a policy

About this task

A *target* defines a set of access requests to which the policy applies.

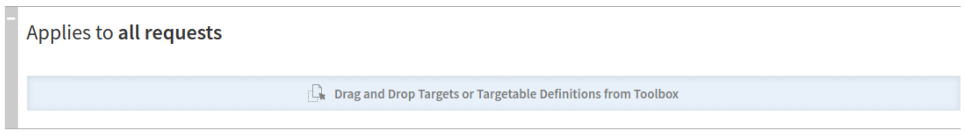


After you name a policy, perform the following steps to add targets:

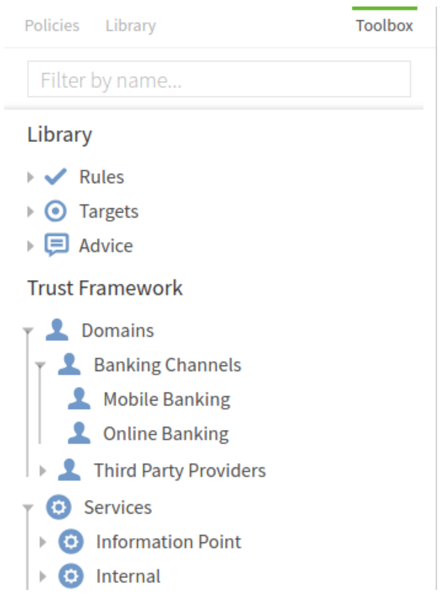
Steps

1. Click **Show "Applies to"**.

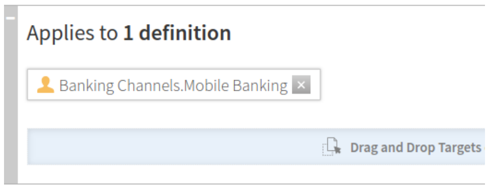
   The **Target** section expands.

   
   Applies to all requests
   Drag and Drop Targets or Targetable Definitions from Toolbox

   Note: This policy applies to all requests because no targets are attached. To apply a policy exclusively to all requests to a particular database, add the database domain as a target.
2. Drag the appropriate domains, services, identity classes, and actions from the **Toolbox** to the policy's **Target** box.

   These elements were created in the trust framework. To target Mobile Banking requests, for example, drag the corresponding domain to the **Target** box. To target all banking groups, including mobile and online groups, add the `Banking Channel` domain, which functions as a parent for both types of requests. This step adds a total of three targets because the top level also serves as a target.
3. In this example, the domain `Mobile Banking` is dragged to the policy's **Target** box.

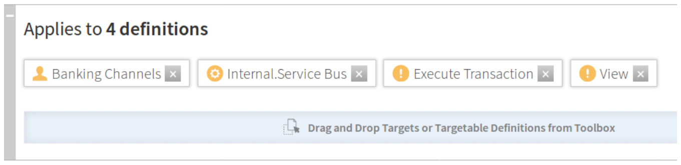

Applies to **1 definition**

Banking Channels.Mobile Banking

Drag and Drop Targets

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

4. This example adds four definitions as targets to the policy.

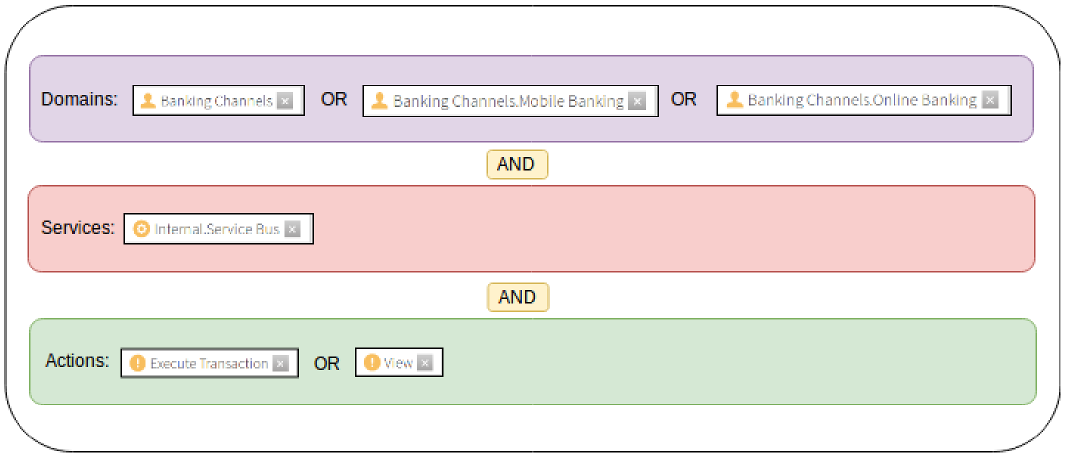

Applies to **4 definitions**

Banking Channels | Internal.Service Bus | Execute Transaction | View

Drag and Drop Targets or Targetable Definitions from Toolbox

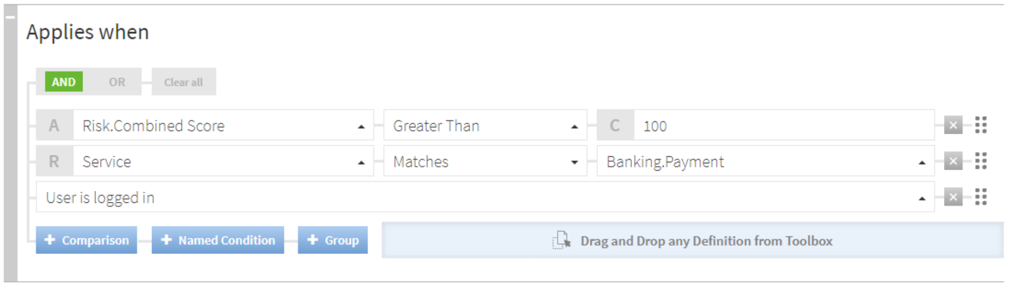
Three domains are involved in this example because the `Banking Channel` definition is a parent for two other definitions. Logically, one of the definitions is picked by applying an OR operation within the definition type.

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



Conditional targets

Conditional targets extend the capability of the "Applies to" concept by interweaving targets with other conditional logic, and by allowing standalone logic to determine whether and when a policy or rule applies. To enable this functionality, click **Show "Applies When"**.


Applies when

AND | OR | Clear all

A Risk.Combined Score | Greater Than | C 100

R Service | Matches | Banking.Payment

User is logged in

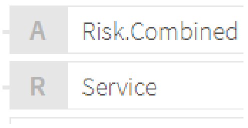
+ Comparison | + Named Condition | + Group | Drag and Drop any Definition from Toolbox

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

- Attribute comparison – Allows the comparison of an attribute with another attribute or a constant.
- Request comparison – Allows the matching of an incoming request to determine, for example, whether the requested service equals `Banking.Payment`.

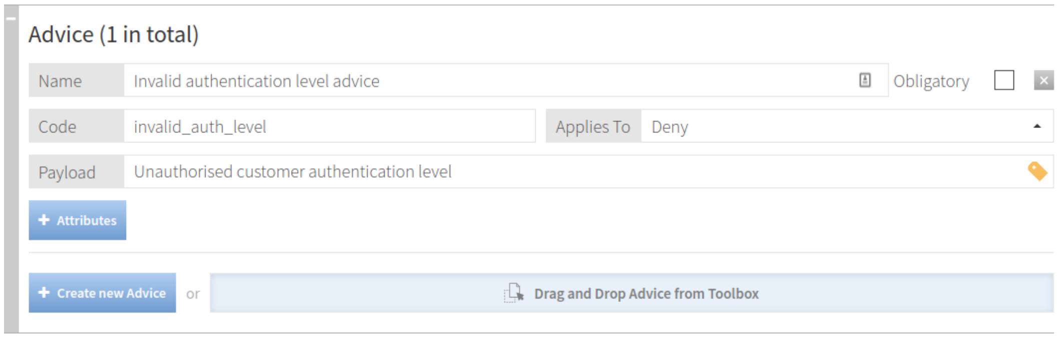
- Named condition – Click **Add Named Condition** to add a **Named Condition** drop-down list that provides a selection of named, presaved conditions.

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



Advice

Advice is additional information that can be attached to a decision response. It is sent back to the governance engine so that the appropriate action can be taken, depending on the response that was evaluated from the policy itself. If a policy is set up to verify the authentication level of a user, and if the policy evaluation determines that the user does not possess the correct access privileges, information can be sent to explain the reason for denying access.



Advice (1 in total)

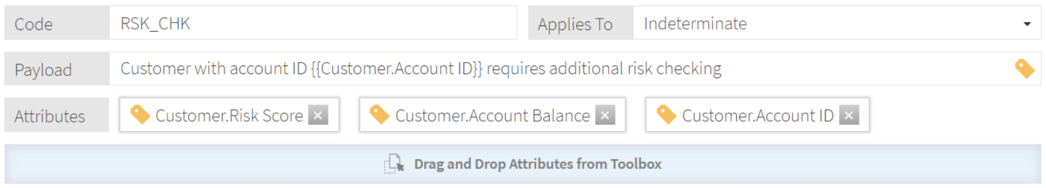
Name	Invalid authentication level advice
Obligatory	☐
Code	invalid_auth_level
Applies To	Deny
Payload	Unauthorised customer authentication level

To indicate that the final decision applies only when the advice can be fulfilled, mark the advice as Obligatory. The service that calls the PDP normally handles this responsibility, but you can also use the PingDataGovernance Policy Administration Obligation Fulfillment Service to handle it.

Each advice features the following mandatory fields:

- **Name** – Human-readable label for reference in the Policy Manager.
- **Code** – Identifier used to distinguish between different types of advice.
- **Applies To** – Type of decision to which the advice attaches.

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



Code	RSK_CHK
Applies To	Indeterminate
Payload	Customer with account ID {{Customer.Account ID}} requires additional risk checking
Attributes	Customer.Risk Score, Customer.Account Balance, Customer.Account ID

Advice carries additional data in the form of payloads and attributes. A *payload* is an optional field that can consist of static or interpolated data. *Attributes* allow you to return a key-value mapping of any attributes that might be relevant to the advice.

Properties

Use *properties* to add metadata to a policy in the format of a key-value pair.

Rules and combining algorithms

Each policy can feature multiple rules that produce one of the following responses:

- `Permit`
- `Deny`
- `Indeterminate`
- `Not applicable`

To evaluate the overall decision of a policy, a combining algorithm is applied. The default algorithm set on a new policy is *Unless One Decision is Deny, the Decision will be Permit*. This algorithm always evaluates to `permit` unless the decision is `deny`.

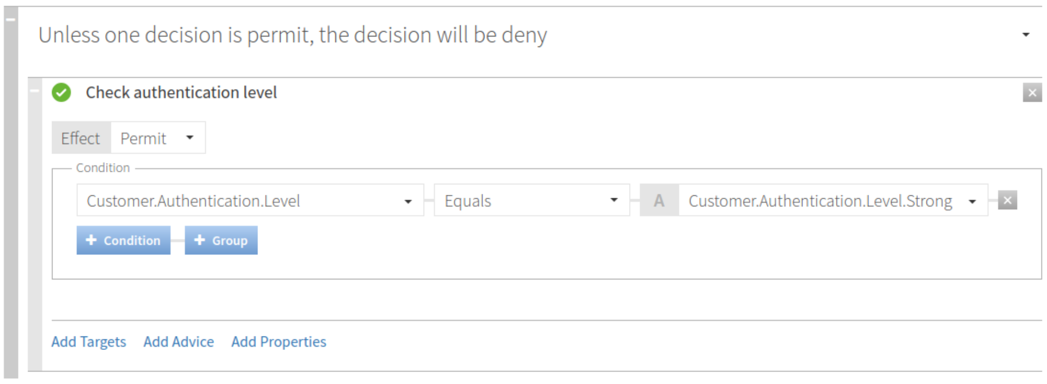
The following list identifies these algorithms and describes their effects:

- *PermitUnlessDeny (Unless one decision is deny, the decision will be permit)* – Policy defaults to `Permit` unless any of its children produce the decision `Deny`.
- *DenyUnlessPermit (Unless one decision is permit, the decision will be deny)* – Policy defaults to `Deny` unless any of its children produce the decision `Permit`.
- *PermitOverrides (A single permit will override any deny decisions)* – If any children produce the decision `Permit`, the policy returns `Permit`. If this step does not occur, the policy returns `Indeterminate` if a child produces `Indeterminate`. Otherwise, the policy returns `Deny` if a child produces `Deny`, and returns `Not Applicable` if all children are `Not Applicable`.
- *DenyOverrides (A single deny will override any permit decisions)* – If any children produce the decision `Deny`, the policy returns `Deny`. If this step does not occur, the policy returns `Indeterminate` if a child produces `Indeterminate`. Otherwise, the policy returns `Permit` if a child produces `Permit`, and returns `Not Applicable` if all children are `Not Applicable`.
- *FirstApplicable (The first applicable decision will be the final decision)* – Children are evaluated in turn until one produces an applicable value of `Permit`, `Deny`, or `Indeterminate`. If no applicable decisions are present, the policy returns `Not Applicable`.
- *OnlyOneApplicable (Only one child may produce a decision. If more than one is produced, the result will be indeterminate)* – Children are evaluated in turn. If at any point two children produce a decision other than `Not Applicable`, the policy returns `Indeterminate`. Otherwise, if precisely one child produces an applicable decision, the policy uses this value, and if no children produce applicable decisions, the policy returns `Not Applicable`.
- *DenyUnlessThreshold (Permit if the weighted average of applicable child decisions meets the threshold, otherwise deny)* – The policy's children are assigned weights between 0 and 100. If a child returns `Permit`, the weight is added to a running total. If a child returns `Deny`, the weight is subtracted from the running total. After all children have been evaluated, the total is divided by the number of children and is compared against the threshold. If the average value is greater than or equal to the threshold, the policy returns `Permit`. Otherwise, it returns `Deny`.

Rule structure

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

Like policies and policy sets, rules can have targets that work in the same way. Apply targets to achieve a more fine-grained approach. For example, one rule might target the mobile banking channel, and another rule might target the online banking channel.


Unless one decision is permit, the decision will be deny

Check authentication level

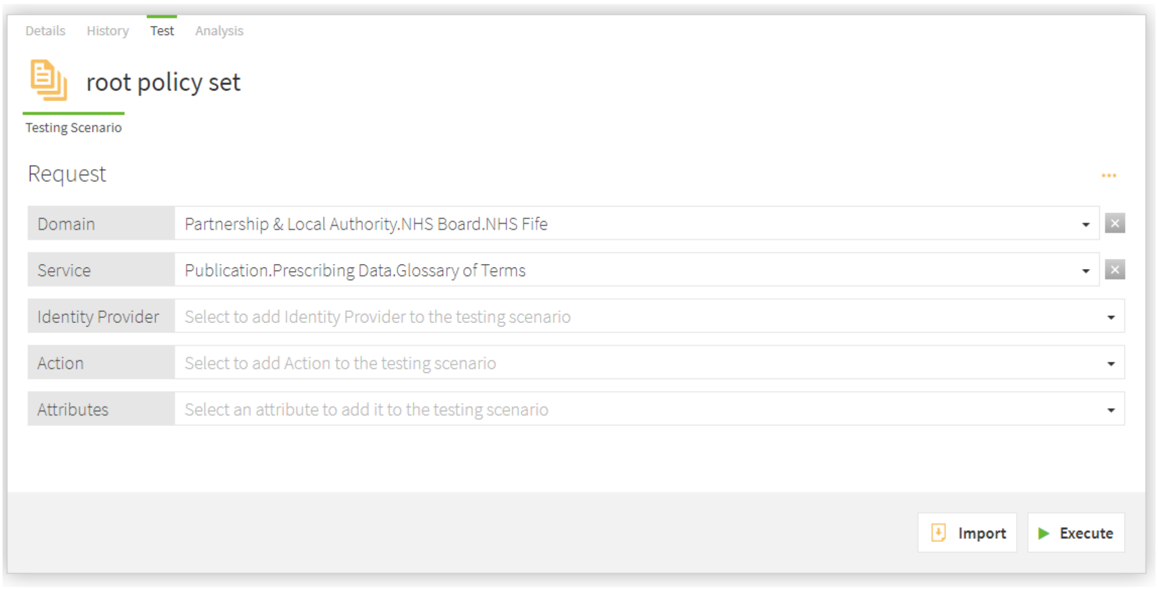
Effect: Permit

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

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

Testing

The PingDataGovernance Policy Administration GUI provides testing capabilities that allow for the evaluation of *test authorization requests* against any or all policy nodes. To identify the node against which policies are tested, select it as the root node from the tree on the left side of the page. In this example, the evaluation is run against all policies because the root policy set is selected.



root policy set

Testing Scenario

Request

Domain	Partnership & Local Authority.NHS Board.NHS Fife
Service	Publication.Prescribing Data.Glossary of Terms
Identity Provider	Select to add Identity Provider to the testing scenario
Action	Select to add Action to the testing scenario
Attributes	Select an attribute to add it to the testing scenario

To prepare a test request, perform the following steps:

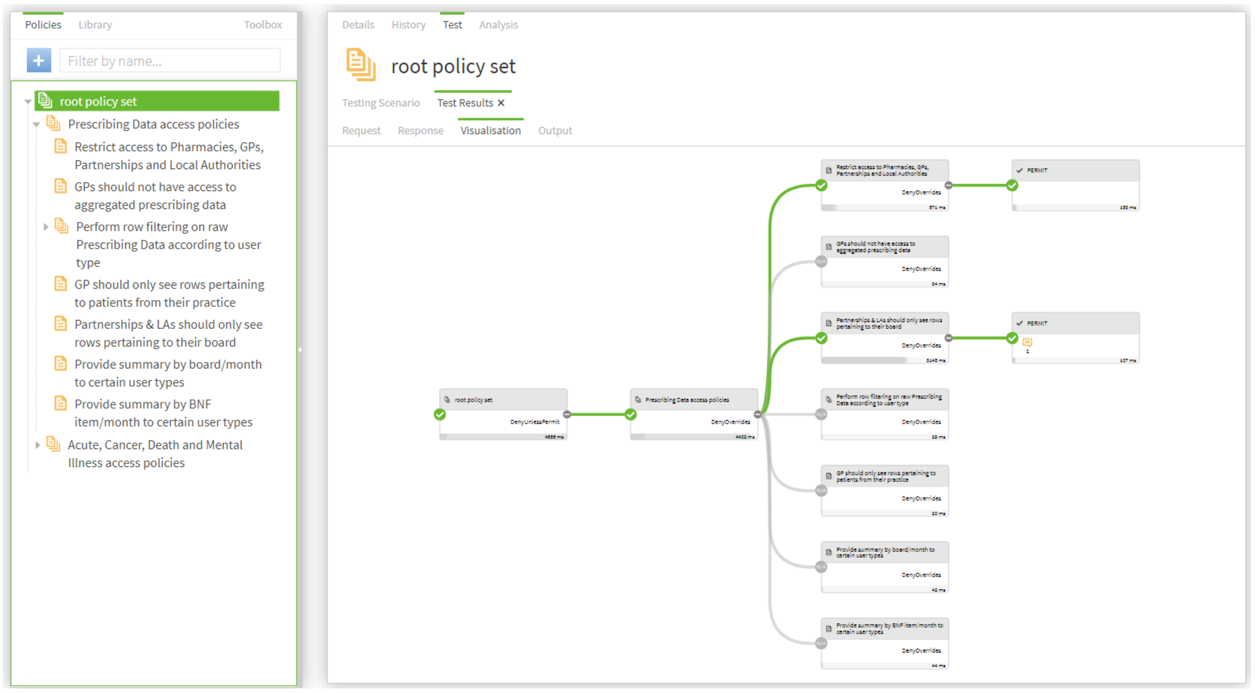
1. Select a definition of type `Attribute` or `Service`.
2. Navigate to the **Test** tab.
3. To form the request, select the following elements:
   - Domain
   - Service
   - IdP
   - Action
   - Attributes
4. If the information endpoints that your attribute resolvers require are running, click **Execute**.

   If your endpoints are not running or are otherwise unavailable, use the **Overrides** section to provide stubbed values for any attributes and services that might be required during evaluation.

This step overrides the attribute and service resolution, and uses the stubbed values instead.

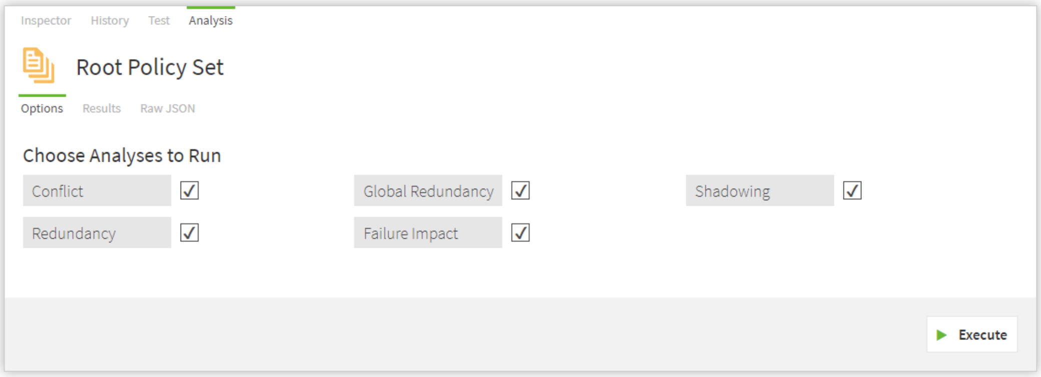
After the request is evaluated, the following Results tabs are displayed:

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


Policies
Library
Toolbox
Filter by name...
root policy set
Prescribing Data access policies
Restrict access to Pharmacies, GPs, Partnerships and Local Authorities
GPs should not have access to aggregated prescribing data
Perform row filtering on raw Prescribing Data according to user type
GP should only see rows pertaining to patients from their practice
Partnerships & LAs should only see rows pertaining to their board
Provide summary by board/month to certain user types
Provide summary by BNF item/month to certain user types
Acute, Cancer, Death and Mental Illness access policies
Details
History
Test
Analysis
root policy set
Testing Scenario
Test Results ×
Request
Response
Visualisation
Output
DenyUnlessPermit
DenyOverrides
PERMIT

Analysis

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

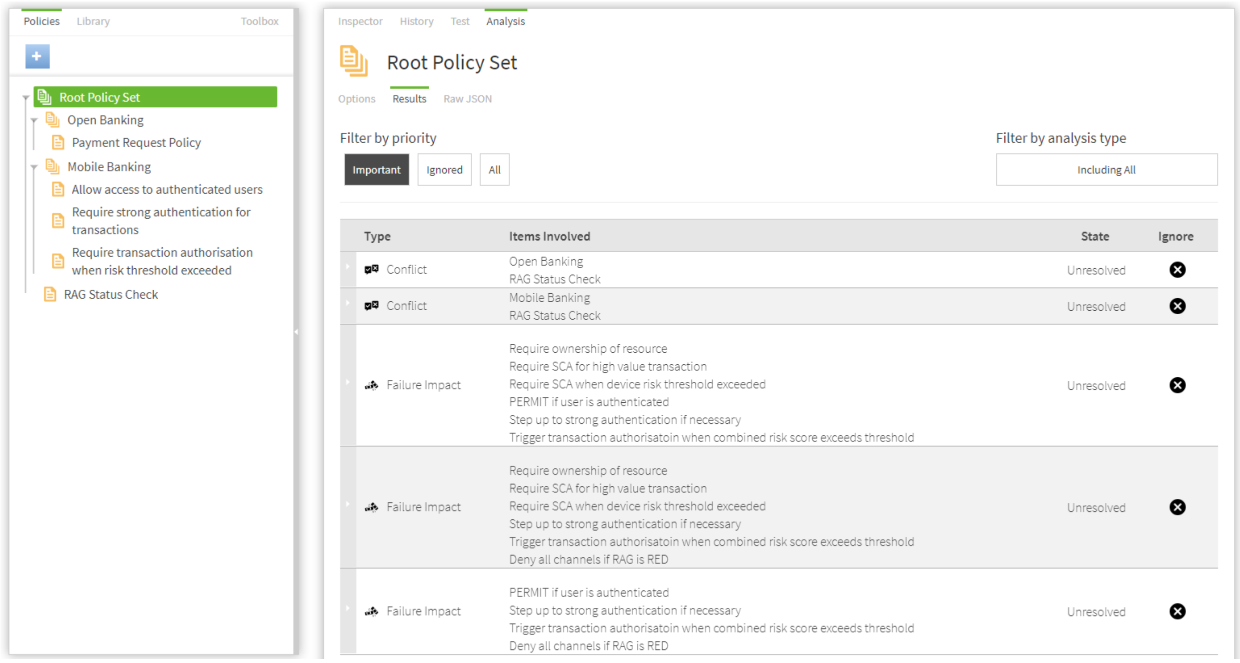


Choose Analyses to Run

Conflict ✓	Global Redundancy ✓	Shadowing ✓
Redundancy ✓	Failure Impact ✓	

The following options provide strong analytical value:

- Conflicts – Highlights real policy conflicts, such as a policy permitting access to a resource when, under the same conditions, another policy denies access to the resource.
- Redundancy – Highlights policies that are redundant, based on one or more policies, and whose presence make no difference to the response.
- Shadows – Highlights policies that another policy can replace.
- Global redundancy – Similar to Redundancy but applies to library policies that are used in multiple places.
- Failure impact – Highlights information points whose failure might alter the decision.



- Root Policy Set
  - Open Banking
    - Payment Request Policy
  - Mobile Banking
    - Allow access to authenticated users
    - Require strong authentication for transactions
    - Require transaction authorisation when risk threshold exceeded
  - RAG Status Check

Root Policy Set

Filter by priority: Important | Ignored | All

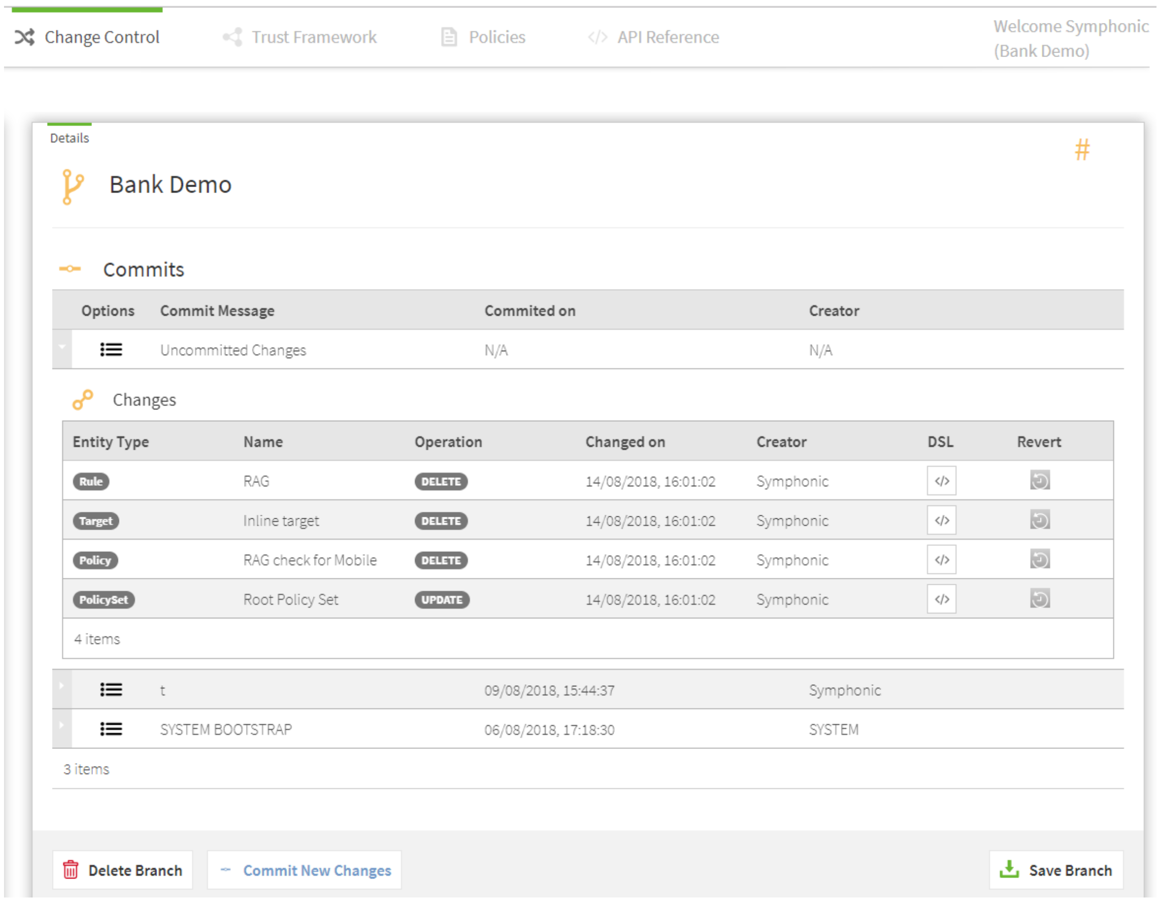
Filter by analysis type: Including All

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

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

Change control

After you finish building, testing, and analyzing your policies, commit your changes so that you can move the policies into a state where they can be deployed to PingDataGovernance Server.


Change Control | Trust Framework | Policies | API Reference | Welcome Symphonic (Bank Demo)

Details

Bank Demo

Commits

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

Changes

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

4 items

Options	Commit Message	Commited on	Creator
	t	09/08/2018, 15:44:37	Symphonic
	SYSTEM BOOTSTRAP	06/08/2018, 17:18:30	SYSTEM

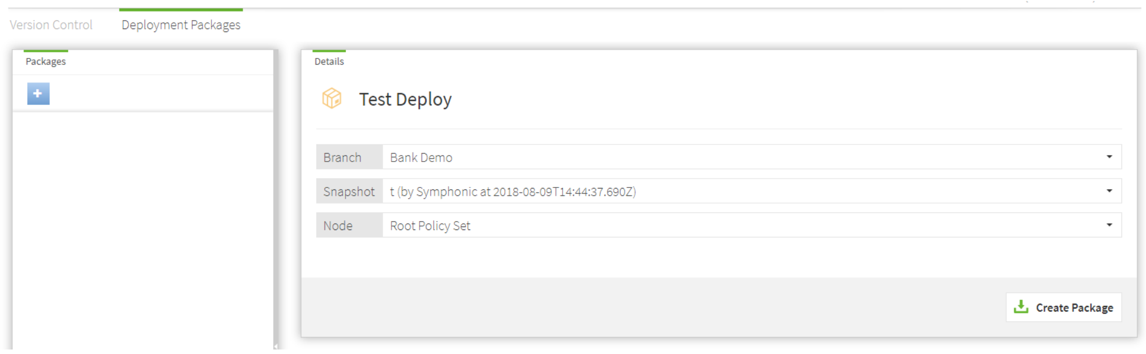
3 items

Delete Branch | Commit New Changes | Save Branch

A list of previously committed snapshots, as well as any uncommitted changes, appears under **Change Control**. To commit these changes and create a snapshot that forms the starting point of your deployment package, click **Commit New Changes**.

Deployment packages

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


Version Control | Deployment Packages

Packages

Details

Test Deploy

Branch	Bank Demo
Snapshot	t (by Symphonic at 2018-08-09T14:44:37.690Z)
Node	Root Policy Set

Create Package

For more information, see *Configure the PingDataGovernance Policy Administration GUI* on page 8.

Chapter 4

REST API

Swagger documentation is available through the PingDataGovernance Policy Administration GUI and has full testing capabilities. To view the Swagger documents, click **API Reference** in the GUI.