PingAccess

Contents

PingAccess

PingAccess protects web applications and APIs by applying security policies to client requests.

Release Notes

- *Current*

Get Started With PingAccess

- *Introduction to PingAccess*
- *Installing and Uninstalling PingAccess*
- *Backing up and Restoring PingAccess*
- *Upgrading PingAccess*

Use PingAccess

- *PingAccess Use Cases*
- *Configuring and Customizing PingAccess*
- *PingAccess User Interface Reference Guide*
- *Reference Guides* on page 152
- *PingAccess Integrations*
- *PingAccess Monitoring Guide*

Troubleshoot PingAccess

- *Administrative SSO lockout* on page 544
- *Collecting support data* on page 546
- *Minimizing the PingAccess cookie size* on page 547

Learn More

- *PingAccess Community*
- *Support Portal*
- *PingAccess Customer Training (existing customers only)*
- *Partner Portal (partners)*

Release Notes

These release notes summarize the changes in current and previous product updates. Updated August 16, 2024.

PingAccess is a centralized point of security and access control for Web applications and APIs, serving applications and other resources to clients outside an organization while still protecting internal interfaces from unauthorized access. PingAccess protects applications and APIs, enabling access control and identity-based auditing on incoming requests. Featuring a lightweight, highly scalable architecture, PingAccess complements PingFederate with centralized session management and URL-level authorization.

PingAccess 7.0.8 (August 2024)

Fixed a security vulnerability with URL-encoded characters PA-15776 **Security**

Added the **`pa.uri.canonicalize`** parameter to the *Configuration file reference* on page 163 to fix a security vulnerability. Learn more in an upcoming security advisory.

Opt out of automatic URL encoding PA-15697 **Improved**

By default, redirect rules and rejection handlers automatically URL encode the admin input redirect URL. This could cause unexpected behavior if an application targeted by a redirect requires the URL to follow a specific format.

You can now opt out of automatic URL encoding by deselecting the **Encode URL** check box on a specific application resource logout or redirect response generator, redirect rule, redirect authentication challenge response generator, or redirect rejection handler. Learn more in:

- *Adding application resources* on page 235
- *Adding redirect rules* on page 273
- *Creating rejection handlers* on page 290

Set response headers for OAuth errors PA-15764 **Improved**

Added the **`oauth.error.headers`** and **`oauth.error.header.Content-Security-Policy`** parameters to the *Configuration file reference* on page 163.

Fixed issues with query parameter behavior due to automatic URL encoding PA-15696 **Fixed**

Fixed an issue with automatically URL encoding target redirect URLs that sometimes disrupted query parameter sort order or added a trailing `=` to the end of single value query parameters. This issue affected redirect rules, redirect rejection handlers, redirect virtual resources, logout virtual resources, and redirect authentication challenge policy response generators.

Fixed potential infinite loop issue with PingAccess clusters PA-15741 **Fixed**

Fixed a potential infinite loop issue that could prevent an engine node or replica administrative node from applying configuration changes.

PingAccess 7.0.7 (March 2024)

Improved request header security PA-15610 **Security**

Fixed an issue with connection request header handling. Learn more in *SECADV045*.

Updated PingAccess client authentication logic PA-15229 **Improved**

Updated the PingAccess logic that determines which authentication method to use at runtime. This update prevents errors when Private Key JWT and Mutual TLS are the only client authentication methods that a token provider can support.

Fixed `NullPointerException` with the rewrite content rule PA-15612 **Fixed**

Fixed an issue that caused a `NullPointerException` error when the *rewrite content rule* was used on a resource that returned an empty chunked response body.

PingAccess 7.0.6 - September 2022

These enhancements and issue fixes are included in PingAccess 7.0.6, released in September 2022.

Resolved issues

Ticket ID	Description
PA-14974	Added handling to recover from deadlocks encountered during configuration import and other asynchronous Admin API actions.

PingAccess 7.0.5 - August 2022

These enhancements and issue fixes are included in PingAccess 7.0.5, released in August 2022.

Resolved issues

Ticket ID	Description
PA-14875	Fixed an exchange processing issue that could cause a memory leak.
PA-14772	Fixed a potential security issue with basic authentication.

PingAccess 7.0.4 - May 2022

These enhancements and issue fixes are included in PingAccess 7.0.4, released in May 2022.

Resolved issues

Ticket ID	Description
PA-14751	Fixed handling of PingAccess add-on SDK function `com.pingidentity.pa.sdk.http.Body#toString` to maintain the same behavior as seen prior to 6.3.
PA-14645	Fixed an issue where custom load balancing strategies that returned custom TargetHosts would result in runtime exceptions.
PA-14580	Added a default memory limit on the CSD tool.
PA-14638	Fixed handling of non-ASCII characters in the request body of PingAuthorize rules.
PA-14640	Fixed handling of non-ASCII characters in sideband API request bodies.

PingAccess 7.0.3 - January 2022

These enhancements and issue fixes are included in PingAccess 7.0.3, released in January 2022.

Resolved issues

Ticket ID	Description
PA-14557	PingAccess upgraded to Log4j version 2.17.1
PA-14414	Fixed an issue where PingAccess could not use non-FIPS HSM key pairs at runtime.

PingAccess 7.0.2 - December 2021

These enhancements and issue fixes are included in PingAccess 7.0.2, released in December 2021.

Resolved issues

Ticket ID	Description
PA-14555	PingAccess upgraded to Log4j version 2.17.

PingAccess 7.0.1 - December 2021

These enhancements and issue fixes are included in PingAccess 7.0.1, released in December 2021.

Resolved issues

Ticket ID	Description
PA-14549	PingAccess upgraded to Log4j version 2.16.

PingAccess 7.0 - December 2021

These enhancements and issue fixes are included in PingAccess 7.0, released in December 2021.

Enhancements

Added Logout virtual resource

Added a new Logout response generator for virtual resources, enabling you to customize logout behavior for each application. See *Adding application resources* on page 235 for more information.

CRL processing improvements

PingAccess now supports trace-level logging to help troubleshoot certification revocation issues and provides an option to bypass trust anchor validation. This helps improve interoperability with CA infrastructure. See *Creating trusted certificate groups* on page 310 for more information.

Added support for web session access token identity mappings

PingAccess now supports creating web session access token identity mappings. This helps ease integration with existing APIs, in particular in the context of Single Page Applications (SPAs). See *Creating web session access token identity mappings* on page 297 for more information.

Added support for reversed trust chain certificate validation

PingAccess now supports validation for client certificate chains that are not in the standard order, such as a reversed certificate chain of `[root, intermediate, leaf]`. See *Creating trusted certificate groups* on page 310 for more information.

Removed features

Runtime state clustering

PingAccess no longer supports runtime state clustering. Clustered environments that do not use runtime state clustering are not affected.

Resolved issues

Ticket ID	Description
PA-14403	Fixed a potential security issue.
PA-14296	Fixed a potential security issue.

Ticket ID	Description
PA-14284	Fixed a potential security issue.
PA-14279	Fixed a potential security issue.
PA-14287	Fixed a potential security issue.
PA-14331	Fixed a potential security issue.
PA-14302	Fixed a potential security issue.
PA-14134	Fixed a potential security issue.
PA-14135	Fixed a potential security issue.
PA-14143	Fixed a potential security issue.
PA-14542	Fixed a typo in the Content-Security-Policy header that prevented PingAccess from loading external scripts from HTML responses.
PA-14541	Fixed an issue in the CRL client certificate authentication flow that returned a 500 error code when PingAccess is in FIPS mode.
PA-14421	Updated the PingAccess UI to display the alias of the selected certificates in the Trusted Certificate Group List.
PA-14433	Fixed an issue that limited the host field for the Primary Administrative Node to 64 characters, instead of the standard 255 characters.
PA-14083	Added handling to URL encode client secrets with special characters per RFC 6749.
PA-14445	Fixed an issue where upon detecting a revoked certificate in a chain, PingAccess incorrectly assumes it is always the first cert in the chain.
PA-14304	Fixed an issue that returned a 500 error when requesting keypairs endpoints with special characters in the chain certs field.
PA-14467	Fixed an issue that caused key rolling to result in Admin Token Provider and System Token Provider being switched.
PA-14477	Fixed a typo that could cause warnings when running PingAccess as a Windows Service.
PA-14402	Fixed an issue that prevented PingAccess from encoding non-ASCII characters when they are in the domain only.
PA-14468	Fixed an issue that caused PingAccess to trigger an error when using the PingAuthorize Access Control rule and the target Sideband provider returns a response that omits the **`response.body`** parameter.
PA-14392	Fixed an issue that caused PingAccess Admin UI to incorrectly initialize an application with the state of another application leading to scenarios where an administrator could mistakenly update an application with the data of another application.
PA-14314	Fixed an issue that prevented header warnings from being sent for PEM key pairs with a single duplicate chain certificate.
PA-14258	Added INFO level logging at the start of configuration import.
PA-14280	Fixed an issue that prevented an ACME request with an INVALID state and an empty problem description from displaying correctly.

Ticket ID	Description
PA-14290	Fixed an issue that caused the PingAccess Sideband transport to only use fixed ports when performing resource matching against incoming sideband API requests.
PA-14238	Fixed an issue that caused disabled algorithms to appear on the Signing Algorithm drop down menu on the Auth Token Management page.
PA-14265	Fixed an issue that prevented the SSO Admin Authentication method in the PingAccess admin console from functioning in clustered PingAccess deployments when Private Key JWT client authentication is used.
PA-14029	Fixed an issue that caused PingAccess Sideband API to return an error when no scope claim is configured in the access token.
PA-14305	Fixed an issue where the 'Transfer-Encoding' request header is dropped from inbound PingAccess Sideband API request results.
PA-14472	Improved error message when supplying an empty string to fields that expect a charset.

Known issues and limitations

This list details the known issues and limitations of PingAccess.

Known issues

- PingAccess Cloud HSM functionality works in FIPS mode but not in regular mode for `Java8u261` and later. `RSASSA-PSS` signing algorithms fail with `Java8u261` or later, and HSM vendors and core Java use different naming conventions for the `RSASSA-PSS` algorithm. There is a documented workaround in *Adding an AWS CloudHSM provider* on page 318.
- Due to an outstanding defect in the Kong API Gateway, the `ping-auth` plugin currently does not support requests that utilize the `Transfer-Encoding` header. If PingAccess is used as the external authorization server, the Rewrite Content rule can prevent the page from displaying.
- PingAccess 6.3 deployments that use the Sideband API feature cannot be upgraded using the zero downtime upgrade procedure. You must use a planned outage to upgrade such an environment.
- Depending on the source version, the upgrade process may change the default settings for the SameSite cookie attribute to make PingAccess cookies work on all browsers. Review the settings for each web session in **Access** # **Web Sessions** to verify that your SameSite cookie attribute values are set to None or Lax, depending on the third-party context needs for PingAccess cookies.
- PingAccess may have difficulty maintaining TLS 1.3 connections when using JDK 11.0.0, 11.0.1, or 11.0.2 because of *a defect* in those versions. This might cause upgrades to fail on systems using these versions.
- Engines and admin replicas do not connect to admin console if a combination of IP addresses and DNS names are used.
- The token processor can't connect to a JWKS endpoint via SSL when an IP is used rather than a hostname. To workaround this issue, add the hostname as the subject alt name on the key pair.
- If you create multiple virtual hosts with a shared hostname and associate the hostname with a server key pair, the virtual hosts retain the connection with the server key pair even if they are subsequently renamed. The virtual host must be deleted and recreated to remove the association.

- Upgrades will fail with a risk-based authorization rule if a third-party service is not used in the rule.
- Log files may contain excessive warnings issued by Hibernate during startup.
- Asynchronous front-channel logout might fail in some browsers depending on end-user settings. See *https://support.pingidentity.com/s/article/Managing-Single-Log-Out-in-different-browsers* for browser-specific workarounds.
- Assigning a new key pair to the Admin HTTPS listener if the browser does not trust the new key pair can prevent the UI from functioning. The workaround is to close the browser and re-open it so that all connections to the admin node use the new certificate.
- Invalid special characters (`(),/;<=>?@[\]{}"`) can be added to the Certificate to Header Mapping field in an identity mapping. Adding this identity mapping to an application will cause 400 errors when the application is accessed.
- If PingAccess is repeatedly stopped and restarted in FIPS mode, subsequent restarts can take up to 5 minutes to complete. The workaround is to use a tool such as rng-tools to refresh /dev/random and make more entropy available faster. For example:

  ```
  sudo yum install rng-tools
  sudo rngb -b
  ```
- After starting PingAccess for the first time on a Windows system or upgrading PingAccess on a Windows system, a warning message is logged reporting that the `pa.jwk` file was not made non-executable. This message can be ignored.

Known limitations

- Firefox does not correctly support the HTML5 time tag. When using the Time Range rule, enter time in 24-hour format.
- When installing PingAccess as a Windows service using Windows PowerShell and Java 8, the error message "Could not find or load main class" can be safely ignored.
- Request Preservation is not supported with Safari Private Browsing.
- Incorrect handling for IPv6 literals in Host header. Note that IPv6 is not currently supported.

Previous Releases

The release notes linked here show the changes in previous versions of PingAccess.

- *PingAccess 6.3*
- *PingAccess 6.2*
- *PingAccess 6.1*
- *PingAccess 6.0*
- *PingAccess 5.3*
- *PingAccess 5.2*
- *PingAccess 5.1*
- *PingAccess 5.0*
- *PingAccess 4.3*
- *PingAccess 4.2*
- *PingAccess 4.1*
- *PingAccess 4.0*
- *PingAccess 3.2*

Deprecated Features

These features of previous versions of PingAccess have been removed.

Runtime State Clustering

Beginning with PingAccess 7.0, runtime state clustering is no longer supported. Clustered environments that do not use runtime state clustering are not affected.

PingAccess Use Cases

This section provides example uses cases for PingAccess deployments.

- *Protecting a web application with PingAccess in a gateway deployment*
- *Protecting an API with PingAccess in a gateway deployment*
- *Protecting a web application with PingAccess in an agent deployment*
- *Protecting an API with PingAccess in an agent deployment*
- *Protecting an API with PingAccess in a sideband deployment*

Protecting a web application with PingAccess in a gateway deployment

Protect a web application from unwanted access using PingAccess.

Prerequisites

Before configuring your PingAccess deployment to protect a web application:

- Install and run PingAccess. See *Installing and Uninstalling PingAccess* on page 50 for the full procedure.
- Configure a token provider. The procedures vary depending on the token provider. For more information, see:
  - *PingFederate* on page 343.
  - *PingOne* on page 352.
  - *Common token provider* on page 352.

Steps

Protecting a web application involves:

1. Configure a virtual host – A virtual host represents the external face of the site you will protect.
2. Configure a site – A site contains the internal details of the site you will protect, including its actual location.
3. Configure a web session – A web session defines the details of how user credential information is retained. This lets the token provider authenticate the user when it is required for a protected application.
4. Configure a rule – Rules control who can access what content under what circumstances.
5. Configure an identity mapping – An identity mapping lets you share identity information with the protected application as headers.
6. Configure an application – An application joins the other pieces together, giving users access to the site according to the configured rules.

Configuring a virtual host

About this task

The virtual host is the external-facing portion of a web application. In a gateway deployment, the virtual host contains the host name and port that your users use to reach the protected web application.

For more information about this procedure, including optional steps that aren't included here, see *Creating new virtual hosts* on page 245.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the name for the virtual host.

   This is the host name used by end users to reach the site, such as`myHost.com`.

   > **Note:**
   >
   > You can use a wildcard (*) for part or all of the host name. For example, `*.example.com` matches all host names ending in .example.com, and `*` matches all host names.
4. In the **Port** field, enter the port number for the virtual host, such as `443`.
5. Click **Save**.

Configuring a site

About this task

A site is only used in a gateway deployment. It contains the target address for the protected web application and any other information necessary to access the application.

For more information about this procedure, including optional steps that aren't included here, see *Adding sites* on page 247 .

Steps

1. Click **Applications** and then go to **Sites** # **Sites**.
2. Click **+ Add Site**.
3. In the **Site Name** field, enter a unique name of up to 64 characters, including special characters and spaces.

   This name is used internally.
4. In the **Targets** field, enter one or more targets.

   These targets are the actual locations of the site. The format for this is `hostname:port` or `IP address:port`, such as `www.example.com:80`.
5. Select the **Secure** check box if the site is expecting HTTPS connections.

   > **Note:**
   >
   > This decision depends on whether the target expects an HTTPS connections from the PingAccess system to the protected web application.
   >
   > If you select **Secure**, you must also select a **Trusted Certificate Group** from the list, or select **Trust Any** to trust any certificate presented by the listed targets. The trusted certificate group defines what

certificates or issuing certificate authorities PingAccess will trust when acting as a client to the backend server.

For information about importing a certificate and creating a trusted certificate group, see *Importing certificates* on page 309 and *Creating trusted certificate groups* on page 310.

6. Click **Save**.

Note:

If the target site can't be contacted, PingAccess saves the site and displays a warning indicating the reason the site couldn't be reached.

Configuring a web session

About this task

A web session specifies the details of how user information is stored.

For more information about this procedure, including optional steps that aren't included here, see *Creating web sessions* on page 300.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
2. Click **+ Add Web Session**.
3. In the **Name** field, enter a unique name for the web session, up to 64 characters, including special characters and spaces.
4. In the **Cookie Type** list, select **Encrypted JWT**.
5. In the **Audience** field, enter the audience that the PingAccess token is applicable to, represented as a short, unique identifier between 1 and 32 characters.

Note:

PingAccess rejects requests that contain a PingAccess token with an audience that differs from what is configured in the web session associated with the target application.

6. In the **OpenID Connect Login Type** list, select **Code**.

Note:

The **Code** login type is recommended for maximum security and standards interoperability, but other options are available. For information on the available profiles, see *OpenID Connect login types*.

7. In the **Client ID** field, enter the unique identifier (client ID) that was assigned when you created the OAuth Relying Party client within the token provider.

   For more information, see *Configuring a Client* in the PingFederate documentation.

8. In the **Client Credentials Type**list, select a client credentials type.

   Selecting a client credentials type is required when configuring the **Code** login type.

   Choose from:

   - **Secret**
   - **Mutual TLS**
   - **Private Key JWT**

> **Note:**
>
> The OAuth client you use with PingAccess web sessions must have an OpenID Connect policy specified.
>
> For more information see *Configuring OpenID Connect Policies*.

9. Provide the information required for the selected credential type.
   Choose from:

   - **Secret** – Enter the **Client Secret** assigned when you created the OAuth relying party client in the token provider.
   - **Mutual TLS** – Select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – No additional information is required.
10. In the **Idle Timeout** field, specify the amount of time, in minutes, that the PingAccess token remains active when no activity is detected by the user.

    The default is `60` minutes.

> **Note:**
>
> If there is an existing valid PingFederate session for the user, an idle timeout of the PingAccess session might result in its re-establishment without forcing the user to sign on again.

11. In the **Max Timeout** field, specify the amount of time, in minutes, that the PingAccess token remains active before expiring.

    The default is `240` minutes.
12. Click **Save**.

Configuring a rule

About this task

Rules are used to control the circumstances under which users can access the protected web server. Rules can grant or deny access based on criteria such as user parameters from the token provider, header values, network ranges, or web session attributes. You can configure any number of rules in your environment.

You can combine rules into rule sets, which combine multiple rules. You can configure rule sets to allow access to a resource if at least one rule's criteria is met or to only allow access if all rules have their criteria met. Access control rules are processed before processing rules. Each type of rule is otherwise processed in the order you specify when you create the rule set.

You can further combine rule sets into rule set groups, which combine multiple rule sets. As with rule sets, rule set groups can allow access if any one rule set's criteria are met or only if all rule sets' criteria are met. Rule sets are processed in the order you specify when you create the rule set group.

This example uses an HTTP request header rule to demonstrate how rules are created and used. Each environment has different requirements, and you can use any of the rules explained in the *Rule Management* on page 260 section according to your needs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name.

   The name can be up to 64 characters long. Special characters and spaces are allowed.
4. In the **Type** list, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter the header name you want to match in order to grant or not grant the client access.
6. In the **Value** field, enter the for the header you want to match in order to grant or not grant the client access.

   The wildcard (*) character is supported.

   > **Tip:**
   >
   > If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the host name (`host*` or `host:*`) to match what's in the HTTP request.
7. If you need additional header pairs, click **Add Row** to add an additional row, then repeat steps 5-6.
8. Click **Save**.

Configuring an identity mapping

About this task

A header identity mapping can expose one or more attribute values to the protected application in HTTP request headers.

For more information about this procedure, including optional steps that aren't included here, see *Creating header identity mappings* on page 295.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. In the **Type** list, select **Header Identity Mapping**.
5. In the **Attribute to Header Mapping** section, in the **Attribute Name** field, enter the name of the attribute to retrieve from the user web session, such as `sub`.
6. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

   > **Note:**
   >
   > The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.

7. In the **Certificate to Header Mapping** section, enter the header name included in a PEM-encoded client certificate.

   The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **+ Add Row** to add another row.
8. Click **Save**.

Configuring an application

About this task

The application represents the protected web application as a whole. Including the virtual host and the site allows PingAccess to route requests directed at the front-end name to the correct back-end resource. By including a web session, you specify how user credential data is stored and for how long. After you create the application and its root resource, you can add one or more rules to control access to the protected web application.

Within the application, you can add one or more resources. Resources are specific components that require a different degree of security. You can apply different rules to a resource, letting you apply specific controls to portions of an application. This example procedure does not include resources. For information about adding resources to an application, see *Configuring resource ordering in PingAccess* on page 234.

For more information about this procedure, including optional steps that aren't included here, see *Adding an application* on page 228.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. In the **Name** field, enter a unique name for the application, up to 64 characters, including special characters and spaces.
4. In the **Context Root** field, enter the context root for the application.

   The context root represents the context at which the application is accessed at the site and must meet the following criteria:

   - It must start with `/`.
   - It can contain additional `/` path separators.
   - It must not end with `/`.
   - It must not contain wildcards or regular expression strings.
   - The combination of the **Virtual Host** and **Context Root** must be unique.
5. In the **Virtual Host** list, select the virtual host you created.
6. In the **Application Type** section, click **Web**.
7. In the **Web Session** list, select the web session you created.
8. In the **Destination** section, click **Site**, then select the site you created.
9. Click **Save**.
10. Click **Applications** and then go to **Applications** # **Applications**.
11. Expand the new application in the list and click the **Pencil** icon.
12. Click the **Web Policy** tab.
13. Drag your rule from **Available Rules** onto the policy bar.
14. Click **Save**.

Protecting an API with PingAccess in a gateway deployment

You can protect an API from unwanted access using PingAccess.

Prerequisites

Before configuring your PingAccess deployment to protect an API:

- PingAccess must be installed and running. See *Installing and Uninstalling PingAccess* on page 50 for the full procedure.
- You must have a configured token provider. The procedures vary depending on the token provider. For more information, see:
  - *PingFederate* on page 343.
  - *PingOne* on page 352.
  - *Common token provider* on page 352.

Steps

After you have completed the following steps, your API is protected.

1. Configure a virtual host – A virtual host represents the external face of the API you will protect.
2. Configure a site – A site contains the internal details of the API you will protect, including its actual location.
3. Configure a rule – Rules control who can access what content under what circumstances.
4. Configure an identity mapping – An identity mapping lets you share identity information with the protected API as headers.
5. Configure an application – An application joins the other pieces together, giving users access to the API according to the configured rules.
6. Configure a resource – A resource specifies an API endpoint and the methods that can be used to access it.

Configuring a virtual host

About this task

The virtual host is the external-facing portion of an API. In a proxy deployment, the virtual host contains the host name and port that your users use to reach the protected API.

For more information about this procedure, including optional steps that are not included here, see *Creating new virtual hosts* on page 245.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the name for the virtual host.

   This is the host name used by end users to reach the site. For example, myHost.com. You can use a wildcard (*) for part or all of the host name. For example, `*.example.com` matches all host names ending in .example.com, and `*` matches all host names.
4. In the **Port** field, enter the port number for the virtual host. For example, `443`.
5. Click **Save**.

Configuring a site

About this task

A site is only used in a proxy deployment. It contains the target address for the protected web application and any other information necessary to access the application.

For more information about this procedure, including optional steps that are not included here, see *Adding sites* on page 247.

Steps

1. Click **Applications** and then go to **Sites** # **Sites**.
2. Click **+ Add Site**.
3. In the **Site Name** field, enter a unique name of up to 64 characters, including special characters and spaces. This name is used internally.
4. In the **Targets** field, enter one or more targets.

   These targets are the actual locations of the site. The format for this is `hostname:port` or `IP address:port`. For example, `www.example.com:80`.
5. Select the **Secure** check box if the site is expecting HTTPS connections.

   Note:

   This decision depends on whether the target expects an HTTPS connections from the PingAccess system to the protected web application.

   If you select **Secure**, you must also select a **Trusted Certificate Group** from the list, or select **Trust Any** to trust any certificate presented by the listed targets. The trusted certificate group defines what certificates or issuing certificate authorities PingAccess will trust when acting as a client to the backend server. For information about importing a certificate and creating a trusted certificate group, see *Importing certificates* on page 309 and *Creating trusted certificate groups* on page 310.
6. Click **Save**.

   Note:
   If the target site cannot be contacted, PingAccess saves the site and a displays a warning indicating the reason the site could not be reached.

Configuring a rule

About this task

Rules are used to control the circumstances under which users can access the protected API. Rules can grant or deny access based on criteria such as user parameters from the token provider, header values, network ranges, or web session attributes. You can configure any number of rules in your environment.

You can combine rules into rule sets, which combine multiple rules. You can configure rule sets to allow access to a resource if at least one rule's criteria is met, or to only allow access if all rules have their criteria met. Access control rules are processed before processing rules. Each type of rule is otherwise processed in the order you specify when you create the rule set.

You can further combine rule sets into rule set groups, which combine multiple rule sets. As with rule sets, rule set groups can allow access if any one rule set's criteria are met, or only if all rule sets' criteria are met. Rule sets are processed in the order you specify when you create the rule set group.

This example uses an HTTP request header rule to demonstrate how rules are created and used. Each environment has different requirements, and you can use any of the rules explained in the *Rule Management* on page 260 section according to your needs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** menu, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter the header name you want to match in order to grant or not grant the client access.
6. In the **Value** field, enter the for the header you want to match in order to grant or not grant the client access. The wildcard (*) character is supported.

> **Tip:**
> If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the host name (`host*` or `host:*`) to match what is in the HTTP request.

7. If you need additional header pairs, click **Add Row** to add an additional row, then repeat steps 5-6.
8. Click **Save**.

Configuring an identity mapping

About this task

A header identity mapping can expose one or more attribute values to the protected API in HTTP request headers.

For more information about this procedure, including optional steps that are not included here, see *Creating header identity mappings* on page 295.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. From the **Type** list, select **Header Identity Mapping**.
5. In the **Attribute to Header Mapping** section, in the **Attribute Name** field, enter the name of the attribute to retrieve from the user web session. For example, `sub`.
6. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

> **Note:**
>
> The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.

7. In the **Certificate to Header Mapping** section, enter the header name included in a PEM-encoded client certificate.

   The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **+ Add Row** to add another row.

8. Click **Save**.

Configuring an application

About this task

The application represents the protected API as a whole. By including the virtual host and the site, it allows PingAccess to route requests directed at the front-end name to the correct back-end resource. After you create the application and its root resource, you can add one or more rules to control access to the protected API.

Within the application, you can add one or more resources. Resources are specific components that require a different degree of security. You can apply different rules to a resource, letting you apply specific controls to portions of an application. This example procedure does not include resources. For information about adding resources to an application, see *Configuring resource ordering in PingAccess* on page 234.

For more information about this procedure, including optional steps that are not included here, see *Adding an application* on page 228.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. In the **Name** field, enter a unique name for the application, up to 64 characters, including special characters and spaces.
4. In the **Context Root** field, enter the context root for the API. This represents the context at which all of the API endpoints are accessed at the site.

   The context root must meet the following criteria:

   - It must start with `/`.
   - It can contain additional `/` path separators.
   - It must not end with `/`.
   - It must not contain wildcards or regular expression strings.
   - The combination of the **Virtual Host** and **Context Root** must be unique.
5. From the **Virtual Host** list, select the virtual host you created.
6. In the **Application Type** section, select **API**.
7. In the **Destination** section, select **Site**, then select the site you created.
8. Click **Save**.
9. Click **Applications** and then go to **Applications** # **Applications**.
10. Expand the new application in the list and click the pencil icon.
11. Click the **Web Policy** tab.
12. Drag your rule from **Available Rules** onto the policy bar.
13. Click **Save**.

Configuring a resource

About this task

An application resource is a component within an application that requires a different level of security. By configuring resources for your API endpoints, you can add different rules for different endpoints and specify which methods are allowed.

For more information about this procedure, including optional steps that are not included here, see *Adding application resources* on page 235.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand the application you created in the previous procedure.
3. Click the **Pencil** icon.
4. Click the **Resources** tab.
5. Click **+ Add Resource**.
6. In the **Name** field, enter a unique name up to 64 characters, including special characters and spaces.
7. In the **Path Patterns** field, enter a list of URL path patterns, within the context root, that identify this resource.

   Info:

   The path pattern must start with a forward slash (/). It begins after the application context root and extends to the end of the URL.

   When automatic path pattern evaluation ordering is in use (default), patterns can contain one or more wildcard characters (*). No use of wildcards is assumed. For example, there is a difference between `/app/` and `/app/*`. If a request matches more than one resource, the most specific match is used.
8. In the **Resource Authentication** section, select **Standard**, using the same authentication for the resource as for the root application.
9. In the **Methods** field, enter the methods supported by the resource.

   Leave the asterisk default if the resource supports all HTTP methods, including custom methods.

   Tip:
   Defining methods for a resource allows more fine-grained access control policies on API resources. If you have a server optimized for writing data (POST, PUT) and a server optimized for reading data (GET), you might want to segment traffic based on the operation being performed.
10. To log information about the transaction to the audit store, select the **Audit** check box.
11. To enable the resource, select the **Enabled** check box.
12. Click **Save**.

Protecting a web application with PingAccess in an agent deployment

You can protect a web application from unwanted access using PingAccess.

Prerequisites

Before configuring your PingAccess deployment to protect a web application:

- PingAccess must be installed and running. See *Installing and Uninstalling PingAccess* on page 50 for the full procedure.
- You must have a configured token provider. The procedures vary depending on the token provider. For more information, see:
  - *PingFederate* on page 343.
  - *PingOne* on page 352.
  - *Common token provider* on page 352.

- You must have installed an agent on the web server or servers that host the site you want to protect. For more information, see:
  - *PingAccess Agent for Apache (RHEL)* on page 357
  - *PingAccess Agent for Apache (SLES)* on page 380
  - *PingAccess Agent for Apache (Windows)* on page 393
  - *PingAccess Agent for IIS* on page 403
  - *PingAccess Agent for NGINX* on page 422

Steps

After you have completed the following steps, your web application is protected.

1. Configure a virtual host – A virtual host represents the site you will protect and contains information about its location.
2. Configure a web session – A web session defines the details of how user credential information is retained. This lets the token provider authenticate the user when it is required for a protected application.
3. Configure a rule – Rules control who can access what content under what circumstances.
4. Configure an identity mapping – An identity mapping lets you share identity information with the protected application as headers.
5. Configure an application – An application joins the other pieces together, giving users access to the application according to the configured rules.

Configuring a virtual host

About this task

The virtual host is the external-facing portion of a web application. In an agent deployment, the virtual host contains the actual host name and port for the protected web application.

For more information about this procedure, including optional steps that are not included here, see *Creating new virtual hosts* on page 245.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the name for the virtual host.

   This is the host name of the protected web application. For example, myHost.com. You can use a wildcard (*) for part or all of the host name. For example, `*.example.com` matches all host names ending in .example.com, and `*` matches all host names.
4. In the **Port** field, enter the port number for the virtual host. For example, `443`.
5. Click **Save**.

Configuring a web session

About this task

A web session specifies the details of how user information is stored.

For more information about this procedure, including optional steps that are not included here, see *Creating web sessions* on page 300.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Sessions**.

2. Click **+ Add Web Session**.
3. In the **Name** field, enter a unique name for the web session, up to 64 characters, including special characters and spaces.
4. From the **Cookie Type** list, select **Encrypted JWT**.
5. In the **Audience** field, enter the audience that the PA token is applicable to, represented as a short, unique identifier between one and 32 characters.

   > **Note:**
   >
   > PingAccess rejects requests that contain a PA token with an audience that differs from what is configured in the web session associated with the target application.

6. From the **OpenID Connect Login Type** list, select **Code**.

   > **Note:**
   >
   > The **Code** login type is recommended for maximum security and standards interoperability, but other options are available. For information on the available profiles, see *OpenID Connect login types*.

7. In the **Client ID** field, enter the unique identifier (client ID) that was assigned when you created the OAuth Relying Party client within the token provider (for more information, see *Configuring a Client* in the PingFederate documentation).
8. Select a **Client Credentials Type**. This is required when configuring the **Code** login type.
   Choose from:

   - **Secret**
   - **Mutual TLS**
   - **Private Key JWT**

   > **Info:**
   > The OAuth client you use with PingAccess web sessions must have an OpenID Connect policy specified (for more information see *Configuring OpenID Connect Policies*).

9. Provide the information required for the selected credential type.
   Choose from:

   - **Secret** – Enter the **Client Secret** assigned when you created the OAuth relying party client in the token provider.
   - **Mutual TLS** – Select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – No additional information is required.
10. In the **Idle Timeout** field, specify the amount of time, in minutes, that the PA token remains active when no activity is detected by the user (the default is `60` minutes).

    > **Info:**
    > If there is an existing valid PingFederate session for the user, an idle timeout of the PingAccess session might result in its re-establishment without forcing the user to sign on again.

11. In the **Max Timeout** field, specify the amount of time, in minutes, that the PA token remains active before expiring (the default is `240` minutes).
12. Click **Save**.

Configuring a rule

About this task

Rules are used to control the circumstances under which users can access the protected web server. Rules can grant or deny access based on criteria such as user parameters from the token provider, header values, network ranges, or web session attributes. You can configure any number of rules in your environment.

You can combine rules into rule sets, which combine multiple rules. You can configure rule sets to allow access to a resource if at least one rule's criteria is met, or to only allow access if all rules have their criteria met. Access control rules are processed before processing rules. Each type of rule is otherwise processed in the order you specify when you create the rule set.

You can further combine rule sets into rule set groups, which combine multiple rule sets. As with rule sets, rule set groups can allow access if any one rule set's criteria are met, or only if all rule sets' criteria are met. Rule sets are processed in the order you specify when you create the rule set group.

This example uses an HTTP request header rule to demonstrate how rules are created and used. Each environment has different requirements, and you can use any of the rules explained in the *Rule Management* on page 260 section according to your needs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** menu, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter the header name you want to match in order to grant or not grant the client access.
6. In the **Value** field, enter the for the header you want to match in order to grant or not grant the client access. The wildcard (*) character is supported.

   > **Tip:**
   > If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the host name (`host*` or `host:*`) to match what is in the HTTP request.

7. If you need additional header pairs, click **Add Row** to add an additional row, then repeat steps 5-6.
8. Click **Save**.

Configuring an identity mapping

About this task

A header identity mapping can expose one or more attribute values to the protected application in HTTP request headers.

For more information about this procedure, including optional steps that are not included here, see *Creating header identity mappings* on page 295.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.

4. From the **Type** list, select **Header Identity Mapping**.
5. In the **Attribute to Header Mapping** section, in the **Attribute Name** field, enter the name of the attribute to retrieve from the user web session. For example, `sub`.
6. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

> **Note:**
>
> The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.

7. In the **Certificate to Header Mapping** section, enter the header name included in a PEM-encoded client certificate.

   The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **+ Add Row** to add another row.
8. Click **Save**.

Configuring an application

About this task

The application represents the protected web application as a whole. By including the virtual host and the agent, it allows PingAccess to match incoming traffic to the application with the agent managing that application. By including a web session, you specify how user credential data is stored and for how long. After you create the application and its root resource, you can add one or more rules to control access to the protected web application.

Within the application, you can add one or more resources. Resources are specific components that require a different degree of security. You can apply different rules to a resource, letting you apply specific controls to portions of an application. This example procedure does not include resources. For information about adding resources to an application, see *Configuring resource ordering in PingAccess* on page 234.

For more information about this procedure, including optional steps that are not included here, see *Adding an application* on page 228.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. In the **Name** field, enter a unique name for the application, up to 64 characters, including special characters and spaces.
4. In the **Context Root** field, enter the context root for the application. This represents the context at which the application is accessed at the site.

   The context root must meet the following criteria:

   - It must start with `/`.
   - It can contain additional `/` path separators.
   - It must not end with `/`.
   - It must not contain wildcards or regular expression strings.
   - The combination of the **Virtual Host** and **Context Root** must be unique.
5. From the **Virtual Host** list, select the virtual host you created.
6. In the **Application Type** section, select **Web**.
7. From the **Web Session** list, select the web session you created.

8. In the **Destination** section, select **Agent**, then select the agent that is installed on the same web server as the web application.
9. Click **Save**.
10. Click **Applications** and then go to **Applications** # **Applications**.
11. Expand the new application in the list and click the pencil icon .
12. Click the **Web Policy** tab.
13. Drag your rule from **Available Rules** onto the policy bar.
14. Click **Save**.

Protecting an API with PingAccess in an agent deployment

You can protect an API from unwanted access using PingAccess.

Prerequisites

Before configuring your PingAccess deployment to protect an API:

- PingAccess must be installed and running. See *Installing and Uninstalling PingAccess* on page 50 for the full procedure.
- You must have a configured token provider. The procedures vary depending on the token provider. For more information, see:
  - *PingFederate* on page 343.
  - *PingOne* on page 352.
  - *Common token provider* on page 352.
- You must have installed an agent on the web server or servers that host the API you want to protect. For more information, see:
  - *PingAccess Agent for Apache (RHEL)* on page 357
  - *PingAccess Agent for Apache (SLES)* on page 380
  - *PingAccess Agent for Apache (Windows)* on page 393
  - *PingAccess Agent for IIS* on page 403
  - *PingAccess Agent for NGINX* on page 422

Steps

After you have completed the following steps, your API is protected.

1. Configure a virtual host – A virtual host represents the API you will protect and contains information about its location.
2. Configure a rule – Rules control who can access what content under what circumstances.
3. Configure an identity mapping – An identity mapping lets you share identity information with the protected API as headers.
4. Configure an application – An application joins the other pieces together, giving users access to the API according to the configured rules.
5. Configure a resource – A resource specifies an API endpoint and the methods that can be used to access it.

Configuring a virtual host

About this task

The virtual host is the external-facing portion of an API. In an agent deployment, the virtual host contains the actual host name and port for the protected API.

For more information about this procedure, including optional steps that are not included here, see *Creating new virtual hosts* on page 245.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the name for the virtual host.

   This is the host name of the protected API. For example, myHost.com. You can use a wildcard (*) for part or all of the host name. For example, `*.example.com` matches all host names ending in .example.com, and `*` matches all host names.
4. In the **Port** field, enter the port number for the virtual host. For example, `443`.
5. Click **Save**.

Configuring a rule

About this task

Rules are used to control the circumstances under which users can access the protected API. Rules can grant or deny access based on criteria such as user parameters from the token provider, header values, network ranges, or web session attributes. You can configure any number of rules in your environment.

You can combine rules into rule sets, which combine multiple rules. You can configure rule sets to allow access to a resource if at least one rule's criteria is met, or to only allow access if all rules have their criteria met. Access control rules are processed before processing rules. Each type of rule is otherwise processed in the order you specify when you create the rule set.

You can further combine rule sets into rule set groups, which combine multiple rule sets. As with rule sets, rule set groups can allow access if any one rule set's criteria are met, or only if all rule sets' criteria are met. Rule sets are processed in the order you specify when you create the rule set group.

This example uses an HTTP request header rule to demonstrate how rules are created and used. Each environment has different requirements, and you can use any of the rules explained in the *Rule Management* on page 260 section according to your needs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** menu, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter the header name you want to match in order to grant or not grant the client access.
6. In the **Value** field, enter the for the header you want to match in order to grant or not grant the client access. The wildcard (*) character is supported.

   > **Tip:**
   > If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the host name (`host*` or `host:*`) to match what is in the HTTP request.
7. If you need additional header pairs, click **Add Row** to add an additional row, then repeat steps 5-6.
8. Click **Save**.

Configuring an identity mapping

About this task

A header identity mapping can expose one or more attribute values to the protected API in HTTP request headers.

For more information about this procedure, including optional steps that are not included here, see *Creating header identity mappings* on page 295.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. From the **Type** list, select **Header Identity Mapping**.
5. In the **Attribute to Header Mapping** section, in the **Attribute Name** field, enter the name of the attribute to retrieve from the user web session. For example, `sub`.
6. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

   Note:

   The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.
7. In the **Certificate to Header Mapping** section, enter the header name included in a PEM-encoded client certificate.

   The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **+ Add Row** to add another row.
8. Click **Save**.

Configuring an application

About this task

The application represents the protected API as a whole. By including the virtual host and the agent, it allows PingAccess to route requests directed at the front-end name to the correct back-end resource. After you create the application and its root resource, you can add one or more rules to control access to the protected API.

Within the application, you can add one or more resources. Resources are specific components that require a different degree of security. You can apply different rules to a resource, letting you apply specific controls to portions of an application. This example procedure does not include resources. For information about adding resources to an application, see *Configuring resource ordering in PingAccess* on page 234.

For more information about this procedure, including optional steps that are not included here, see *Adding an application* on page 228.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. In the **Name** field, enter a unique name for the application, up to 64 characters, including special characters and spaces.

4. In the **Context Root** field, enter the context root for the API. This represents the context at which all of the API endpoints are accessed at the site.

   The context root must meet the following criteria:

   - It must start with `/`.
   - It can contain additional `/` path separators.
   - It must not end with `/`.
   - It must not contain wildcards or regular expression strings.
   - The combination of the **Virtual Host** and **Context Root** must be unique.
5. From the **Virtual Host** list, select the virtual host you created.
6. In the **Application Type** section, select **API**.
7. In the **Destination** section, select **Agent**, then select the agent that is installed on the same web server as the API.
8. Click **Save**.
9. Click **Applications** and then go to **Applications** # **Applications**.
10. Expand the new application in the list and click the pencil icon.
11. Click the **Web Policy** tab.
12. Drag your rule from **Available Rules** onto the policy bar.
13. Click **Save**.

Configuring a resource

About this task

An application resource is a component within an application that requires a different level of security. By configuring resources for your API endpoints, you can add different rules for different endpoints and specify which methods are allowed.

For more information about this procedure, including optional steps that are not included here, see *Adding application resources* on page 235.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand the application you created in the previous procedure.
3. Click the **Pencil** icon.
4. Click the **Resources** tab.
5. Click **+ Add Resource**.
6. In the **Name** field, enter a unique name up to 64 characters, including special characters and spaces.
7. In the **Path Patterns** field, enter a list of URL path patterns, within the context root, that identify this resource.

   > **Info:**
   >
   > The path pattern must start with a forward slash (/). It begins after the application context root and extends to the end of the URL.
   >
   > When automatic path pattern evaluation ordering is in use (default), patterns can contain one or more wildcard characters (*). No use of wildcards is assumed. For example, there is a difference between `/app/` and `/app/*`. If a request matches more than one resource, the most specific match is used.

8. In the **Resource Authentication** section, select **Standard**, using the same authentication for the resource as for the root application.

9. In the **Methods** field, enter the methods supported by the resource.

   Leave the asterisk default if the resource supports all HTTP methods, including custom methods.

   > **Tip:**
   > Defining methods for a resource allows more fine-grained access control policies on API resources. If you have a server optimized for writing data (POST, PUT) and a server optimized for reading data (GET), you might want to segment traffic based on the operation being performed.

10. To log information about the transaction to the audit store, select the **Audit** check box.
11. To enable the resource, select the **Enabled** check box.
12. Click **Save**.

Protecting an API with PingAccess in a sideband deployment

You can protect an API from unwanted access using PingAccess.

Prerequisites

Before configuring your PingAccess deployment to protect an API:

- PingAccess must be installed and running. See *Installing and Uninstalling PingAccess* on page 50 for the full procedure.
- You must have a configured token provider. The procedures vary depending on the token provider. For more information, see:
  - *PingFederate* on page 343.
  - *PingOne* on page 352.
  - *Common token provider* on page 352.
- You must have installed a sideband client on the API gateway that serves the API you want to protect. For more information, contact Ping professional services.

Steps

After you have completed the following steps, your API is protected.

1. Configure a virtual host – A virtual host represents the API you will protect and contains information about its location.
2. Configure a rule – Rules control who can access what content under what circumstances.
3. Configure an identity mapping – An identity mapping lets you share identity information with the protected API as headers.
4. Configure an application – An application joins the other pieces together, giving users access to the API according to the configured rules.
5. Configure a resource – A resource specifies an API endpoint and the methods that can be used to access it.

Configuring a virtual host

About this task

The virtual host is the external-facing portion of an API. In a sideband deployment, the virtual host contains the actual host name and port for the protected API.

For more information about this procedure, including optional steps that are not included here, see *Creating new virtual hosts* on page 245.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the name for the virtual host.

   This is the host name of the protected API. For example, myHost.com. You can use a wildcard (*) for part or all of the host name. For example, `*.example.com` matches all host names ending in .example.com, and `*` matches all host names.
4. In the **Port** field, enter the port number for the virtual host. For example, `443`.
5. Click **Save**.

Configuring a rule

About this task

Rules are used to control the circumstances under which users can access the protected API. Rules can grant or deny access based on criteria such as user parameters from the token provider, header values, network ranges, or web session attributes. You can configure any number of rules in your environment.

You can combine rules into rule sets, which combine multiple rules. You can configure rule sets to allow access to a resource if at least one rule's criteria is met, or to only allow access if all rules have their criteria met. Access control rules are processed before processing rules. Each type of rule is otherwise processed in the order you specify when you create the rule set.

You can further combine rule sets into rule set groups, which combine multiple rule sets. As with rule sets, rule set groups can allow access if any one rule set's criteria are met, or only if all rule sets' criteria are met. Rule sets are processed in the order you specify when you create the rule set group.

This example uses an HTTP request header rule to demonstrate how rules are created and used. Each environment has different requirements, and you can use any of the rules explained in the *Rule Management* on page 260 section according to your needs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** menu, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter the header name you want to match in order to grant or not grant the client access.
6. In the **Value** field, enter the for the header you want to match in order to grant or not grant the client access. The wildcard (*) character is supported.

   Tip:
   If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the host name (`host*` or `host:*`) to match what is in the HTTP request.
7. If you need additional header pairs, click **Add Row** to add an additional row, then repeat steps 5-6.
8. Click **Save**.

Configuring an identity mapping

About this task

A header identity mapping can expose one or more attribute values to the protected API in HTTP request headers.

For more information about this procedure, including optional steps that are not included here, see *Creating header identity mappings* on page 295.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. From the **Type** list, select **Header Identity Mapping**.
5. In the **Attribute to Header Mapping** section, in the **Attribute Name** field, enter the name of the attribute to retrieve from the user web session. For example, `sub`.
6. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

   Note:

   The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.
7. In the **Certificate to Header Mapping** section, enter the header name included in a PEM-encoded client certificate.

   The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **+ Add Row** to add another row.
8. Click **Save**.

Configuring an application

About this task

The application represents the protected API as a whole, including the virtual host, sideband client, and rules. By including the virtual host and sideband client, it allows PingAccess to route requests directed at the front-end name to the correct back-end resource. After you create the application and its root resource, you can add one or more rules to control access to the protected API.

Within the application, you can add one or more resources. Resources are specific components that require a different degree of security. You can apply different rules to a resource, letting you apply specific controls to portions of an application. This example procedure does not include resources. For information about adding resources to an application, see *Configuring resource ordering in PingAccess* on page 234.

For more information about this procedure, including optional steps that are not included here, see *Adding an application* on page 228.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. In the **Name** field, enter a unique name for the application, up to 64 characters, including special characters and spaces.

4. In the **Context Root** field, enter the context root for the API. This represents the context at which all of the API endpoints are accessed at the site.

   The context root must meet the following criteria:

   - It must start with `/`.
   - It can contain additional `/` path separators.
   - It must not end with `/`.
   - It must not contain wildcards or regular expression strings.
   - The combination of the **Virtual Host** and **Context Root** must be unique.
5. From the **Virtual Host** list, select the virtual host you created.
6. In the **Application Type** section, select **API**.
7. In the **Destination** section, select **Sideband**, then select the client that is installed on the API gateway.
8. Click **Save**.
9. Click **Applications** and then go to **Applications** # **Applications**.
10. Expand the new application in the list and click the pencil icon.
11. Click the **Web Policy** tab.
12. Drag your rule from **Available Rules** onto the policy bar.
13. Click **Save**.

Configuring a resource

About this task

An application resource is a component within an application that requires a different level of security. By configuring resources for your API endpoints, you can add different rules for different endpoints and specify which methods are allowed.

For more information about this procedure, including optional steps that are not included here, see *Adding application resources* on page 235.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand the application you created in the previous procedure.
3. Click the **Pencil** icon.
4. Click the **Resources** tab.
5. Click **+ Add Resource**.
6. In the **Name** field, enter a unique name up to 64 characters, including special characters and spaces.
7. In the **Path Patterns** field, enter a list of URL path patterns, within the context root, that identify this resource.

   > **Info:**
   >
   > The path pattern must start with a forward slash (/). It begins after the application context root and extends to the end of the URL.
   >
   > When automatic path pattern evaluation ordering is in use (default), patterns can contain one or more wildcard characters (*). No use of wildcards is assumed. For example, there is a difference between `/app/` and `/app/*`. If a request matches more than one resource, the most specific match is used.

8. In the **Resource Authentication** section, select **Standard**, using the same authentication for the resource as for the root application.

9. In the **Methods** field, enter the methods supported by the resource.

   Leave the asterisk default if the resource supports all HTTP methods, including custom methods.

   Tip:
   Defining methods for a resource allows more fine-grained access control policies on API resources. If you have a server optimized for writing data (POST, PUT) and a server optimized for reading data (GET), you might want to segment traffic based on the operation being performed.

10. To log information about the transaction to the audit store, select the **Audit** check box.
11. To enable the resource, select the **Enabled** check box.
12. Click **Save**.

Introduction to PingAccess

PingAccess is an identity-enabled access management product that protects web applications and APIs by applying security policies to client requests.

PingAccess allows you to protect sites, APIs, and other resources using rules and other authentication criteria. It works in conjunction with PingFederate or other common token providers with the OAuth 2.0 and OpenID Connect (OIDC) protocols to integrate identity-based access management policies through a federated corporate identity store using open standards access protocols.

To help you get the most from PingAccess, this document offers insights about the product, such as:

- *What can I do with PingAccess?* on page 40
- *How does PingAccess work?* on page 41
- *What can I configure with PingAccess?* on page 44
- *How do I choose a deployment model?* on page 48

As you learn about PingAccess features and functions, review the PingAccess configuration and reference documentation for steps to configure them. For a comprehensive set of instructions for using the PingAccess interface, see the *PingAccess User Interface Reference Guide*.

PingAccess for Azure AD

PingAccess for Azure AD is a free version of PingAccess for users of Microsoft's Azure AD that allows you to protect up to 20 applications.

The goal of this solution is to allow for greater control over the access to legacy on-premise applications through the use of PingAccess identity mapping functionality.

PingAccess for Azure AD requires a premium license for Microsoft Azure AD. For information about licensing, see *Microsoft PingAccess for Azure AD documentation*.

This free version includes a limited feature set that is intended to support the basic requirements for application protection using this solution. Users of PingAccess for Azure AD can upgrade to a full license that will allow the use of the full PingAccess feature set.

Important:

When your PingAccess for Azure AD license expires, access to the admin API is removed, and you are unable to configure the product. Though managed access to configured applications continues, you must upload a new license file before you can make any additional configuration changes.

Upgrade notice:

PingAccess for Azure AD provides a limited feature set that may not be compatible with existing PingAccess configurations. For this reason, upgrading from an earlier full version of PingAccess to PingAccess for Azure AD is not supported.

The following table details the available functionality on each of the PingAccess versions, both in the PingAccess user interface and the API.

Functionality	PingAccess	PingAccess for Azure AD
Create applications	Yes	Limited to 20 web session applications.
Create site authenticators	Yes	Limited to Basic and Mutual TLS.
Configure identity mappings	Yes	Limited to Header and JWT.
Create load balancing strategies	Yes	Limited to Header-Based and Round Robin.
Configure web sessions	Yes	Limited to web sessions with OpenID Connect (OIDC) sign-on type CODE.
Configure token provider	Yes	Limited to Microsoft Azure AD authentication source.
Export/Import configuration	Yes	Limited to configurations that includes only features permitted by license type.
Configure policies	Yes	No
Specify authentication requirements	Yes	No
Create and configure custom plugins using the SDK	Yes	No
Configure sites	Yes	Yes
Configure agents	Yes	Yes
Create virtual hosts	Yes	Yes
Configure unknown resource handling	Yes	Yes
Configure availability profiles	Yes	Yes
Configure HTTP request handling	Yes	Yes
Configure listeners	Yes	Yes
Configure forward proxy settings	Yes	Yes
Manage certificates	Yes	Yes
Manage key pairs	Yes	Yes
Configure administrator authentication	Yes	Yes

Functionality	PingAccess	PingAccess for Azure AD
Configure clustering	Yes	Yes
Manage licenses	Yes	Yes

What can I do with PingAccess?

PingAccess provides a highly customizable solution to identity and access management (IAM) that allows you to control access in many ways by specifying a wide range of conditions that must be satisfied.

The following sections describe the methods that PingAccess uses to control access and perform system functions. For more information about the configuration required for any of the following topics, see PingAccess configuration scenarios on *support.pingidentity.com/s/documentation*.

The main functionality of PingAccess allows you to protect an application or API. You can:

- Use PingAccess to protect the application and API resources to which client requests are forwarded.
- Partition applications for tighter access control through the use of resources.
- Customize configuration of site authenticators and authentication requirements to suit the security needs of your organization.
- Incorporate legacy authentication mechanisms through token mediation.
- Apply policies to define how and when a client can access target resources.

Customize your identity access management configuration with the following features:

Apply policies

Use policies, made up of rules, set of rules, or groups of rule sets applied to an application and its resources, to define how and when a client can access target sites. Rules are the building blocks for access control and request processing.

Backup and restore

Backup or restore a PingAccess configuration with just a few clicks.

Configure a token provider

You can configure PingAccess to use PingFederate as the token provider or to use a common token provider through the OAuth 2.0 or OpenID Connect (OIDC) protocols.

Configure administrator authentication

Allow administrators to authenticate with a simple username and password or configure them to authenticate using single sign-on (SSO) or an API in conjunction with PingFederate.

Configure advanced network settings

Create an availability profile to determine how you want to classify a target server as having failed, configure listener ports, define a load balancing strategy, or use HTTP requests to match a served resource with the originating client.

Configure logging

Capture several log types, including those for the engine, security auditing, and cookies. Store logs in Splunk, in an Oracle, PostgreSQL, or SQL Server database, or in a file.

Configure Single Logout

End PingAccess sessions easily when used in conjunction with PingFederate managed sessions or compatible third-party OIDC providers.

Create clusters

Deploy PingAccess in a clustered environment to provide higher scalability and availability for critical services. Place a load balancer in front of the cluster to distribute connections to the nodes in the cluster.

Customize PingAccess look and feel

Customize and localize the PingAccess pages that your users will see, including those for error messages and logout confirmation.

Customize with SDKs

Customize development with SDKs to extend the functionality of the PingAccess server.

Manage certificates and key pairs

Import certificates to establish trust with certificates presented during secure HTTPS sessions. Import or generate key pairs that include the private key and X.509 certificate required for HTTPS communication.

Manage sessions

Use web sessions to define the policies for web application session creation, lifetime, timeout, and scope. Use multiple web sessions to scope the session to meet the needs of a target set of applications. Web sessions improve the security model of the session by preventing unrelated applications from impersonating the end user.

Manually configure runtime parameters

Use a text editor to modify configuration file settings used by PingAccess at runtime.

Protect an application or API

Use PingAccess to protect the application and API resources to which client requests are forwarded. Partition applications for tighter access control through the use of resources. Customize configuration of site authenticators and authentication requirements to suit the security needs of your organization.

The *developers page* contains additional resources for developing applications to work with PingAccess.

Tune performance

Optimize a wide variety of PingAccess components for maximum performance.

Upgrade an existing installation

Upgrade an existing installation using the installer or selectively manage the upgrade process with the PingAccess upgrade utility.

Use APIs

Use the PingAccess APIs to provide a powerful configuration and management experience outside the PingAccess user interface.

How does PingAccess work?

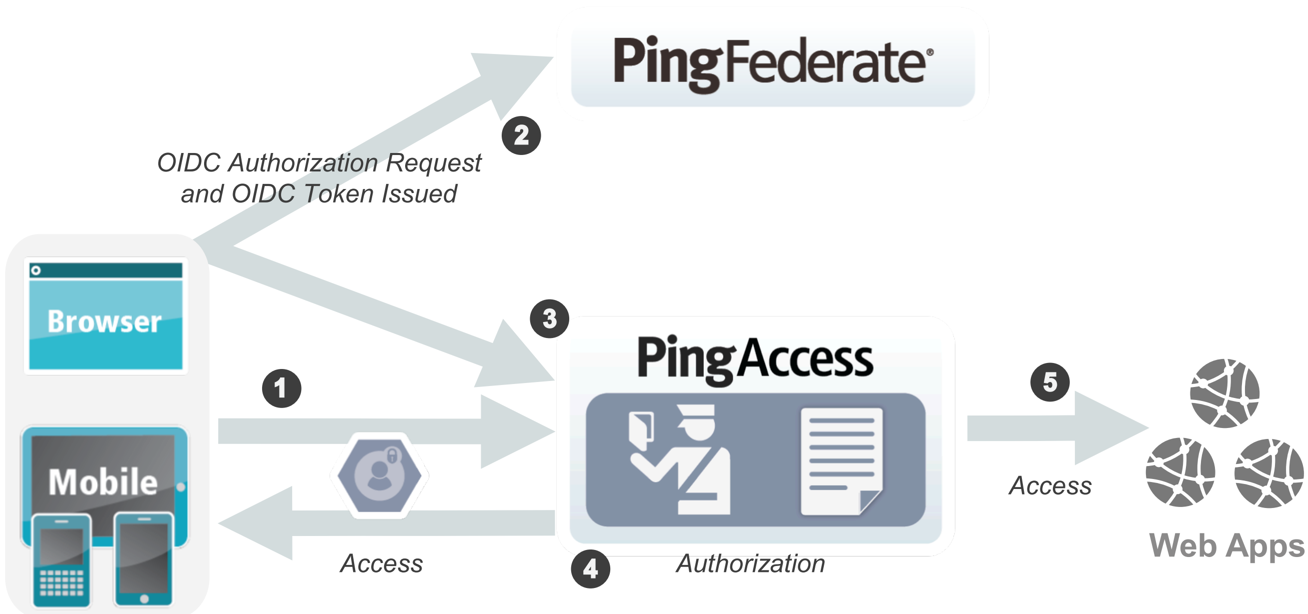
Access requests are either routed through a PingAccess gateway to the target site or intercepted at the target web application server by a PingAccess agent, which coordinates access policy decisions with a PingAccess policy server.

In either instance, policies applied to access requests for the target application are evaluated, and PingAccess makes a policy-based decision to grant or deny access to the requested resource. When access is granted, PingAccess can modify client requests and server responses to provide additional identity information required by the target application.

WAM session initiation

When a user authenticates, PingAccess applies the application and resource-level policies to the Web Access Management (WAM) request.

After policy evaluation is passed, any required token mediation between the backend site and the authenticated user is performed. The user is then granted access to the site.



Processing steps:

1. When a user requests access to a web resource from PingAccess, PingAccess inspects the request for a PingAccess token.
2. If the PingAccess token is missing, PingAccess redirects the user to an OpenID Connect Provider (OP) for authentication.

> **Note:**
>
> When using an OP, you must already have an OAuth client configured in PingAccess. For steps on configuring an OAuth client within PingFederate, see the *PingFederate Administrator's Manual*. To configure the OAuth client within PingAccess, see the PingAccess scenario to configure a token provider.

3. The OP follows the appropriate authentication process, evaluates domain-level policies, and issues an OpenID Connect (OIDC) ID token to PingAccess.
4. PingAccess validates the ID token and issues a PingAccess token and sends it to the browser in a cookie during a redirect to the original target resource. After gaining access to the resource, PingAccess evaluates application and resource-level policies and optionally audits the request.

> **Note:**
>
> PingAccess can perform *Token Mediation* by exchanging the PingAccess token for the appropriate security token from the PingFederate security token service (STS) or from a cache if token mediation occurred recently.

5. PingAccess forwards the request to the target site.

6. PingAccess processes the response from the site to the browser (step not shown).

Note:

For more information, see the session management scenario.

Token mediation

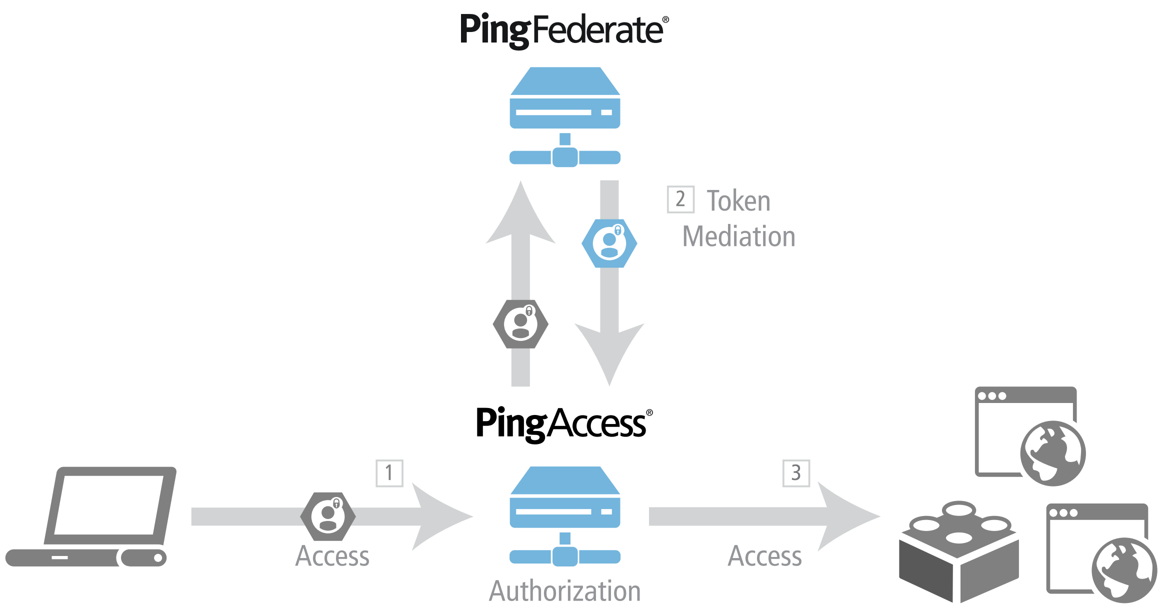
Token Mediation allows a PingAccess gateway to use a PingFederate token generator to exchange the PingAccess token or an OAuth bearer token for a security token used by the foreign authentication system.

The access request is transparent to the user, allowing PingAccess to transparently manage access to systems using those foreign tokens. The request is also transparent to the protected application, which handles the access request as if it came from the user directly. After token mediation has occurred, the token used for accessing the application is cached for continued use during the session.

Note:

When planning a PingAccess deployment, take inventory of existing applications and their authentication requirements and mechanisms. When an existing token-based authentication mechanism is in use, retrofitting that mechanism might not always be desirable or cost-effective.

The following illustration shows an example of token mediation using PingFederate to exchange a PingAccess token or OAuth bearer token for a different security token.



Processing steps:

1. A user requests a resource from PingAccess with a PingAccess token or OAuth bearer token.

   Note:

   This example assumes the user has already obtained a PingAccess token or OAuth bearer token. See the *Session Management* scenario for information on how users authenticate with PingFederate and obtain a PingAccess token or OAuth bearer token.

2. PingAccess evaluates resource-level policies and performs token mediation by acquiring the appropriate security token from the PingFederate security token service (STS) specified by the site authenticator.
3. PingAccess sends the request to the site (web application) with the appropriate token.
4. PingAccess returns the response to the client (not shown).

What can I configure with PingAccess?

PingAccess includes a wide range of features to customize your identity access management deployment.

Agents

Agents are web server plugins that are installed on the web server hosting the target application. Agents intercept client requests to protected applications and allow or deny the request to proceed by consulting the policy manager or using cached information. Agents communicate with the PingAccess policy server through the PingAccess Agent Protocol (PAAP), which defines the possible interactions between agents and policy server.

Agents have a name to identify them and a shared secret for authentication with the policy server. Agents do not need to be unique. There can be any number of agents using the same name and secret, and they are all treated equally by policy server. This is useful in complex deployments where unique agents would be difficult to manage. Agents can be assigned as the destination for one or more applications by name.

Applications

Applications represent the protected web applications and APIs to which client requests are sent. Applications are composed of one or more resources and have a common virtual host and context root corresponding to a single target site. Applications also use a common web session and identity mapping.

To protect applications and their resources, you can apply access control and request processing rules on the policy manager page using the following options:

PingAccess gateway

In a gateway deployment, the target application is specified as a site.

PingAccess agent

In an agent deployment, the application destination is an agent.

Authentication requirements

Authentication requirements are policies that dictate how a user must authenticate before access is granted to a protected web application. Authentication methods are string values and ordered in a list by preference. At runtime, the type of authentication attempted is determined by the order of the authentication methods.

For example:

1. A user attempts to access a PingAccess web application configured with an authentication requirement list containing the values, such as password and certificate.
2. PingAccess redirects the user to PingFederate requesting either password or certificate user authentication.
3. PingFederate authenticates the user based on the password and issues an OpenID Connect (OIDC) ID token to PingAccess, containing the authentication method that was used.
4. PingAccess ensures that the authentication method matches the requirements and redirects the user to the originally requested application with the PingAccess cookie set.

5. The user navigates to the application and access is granted.

   When the user attempts to access a more sensitive application, configured with an authentication requirement list containing the value (certificate), they are redirected to PingFederate to authenticate with a certificate.

If you configure applications with authentication requirement lists that have no overlap, a user navigating between those applications might be required to authenticate each time they visit an application. So, when you're configuring authentication requirement lists to protect higher value applications with step-up authentication, consider including stronger forms of authentication on lower value applications as well.

Auth token management

Auth token management settings define the issuer and signing configuration used by JSON web token (JWT) identity mappings.

Availability profiles

Availability profiles are used in a site configuration to define how PingAccess classifies a backend target server as failed. Sites require the selection of an availability profile even if only one target is provided.

If multiple targets are specified in a site configuration but a load balancing strategy is not applied, then the availability profile causes the first listed target in the site configuration to be used unless it fails. Secondary targets are only used if the first target is not available.

Certificates

Certificates are used to establish anchors used to define trust to certificates presented during secure HTTPS connections. Outbound secure HTTPS connections, such as communication with PingFederate for OAuth access token validation, identity mediation, and communication with a target site, require a certificate trusted by PingAccess. If one does not exist, communication is not allowed.

Certificates used by PingAccess can be issued by a certificate authority (CA) or self-signed. Use CA-issued certificates to simplify trust establishment and minimize routine certificate management operations. Implementations of an X.509-based PKI (PKIX) typically have a set of root CAs that are trusted, and the root certificates are used to establish chains of trust to certificates presented by a client or a server during communication.

The following formats for X.509 certificates are supported:

- Base64 encoded DER (PEM)
- Binary encoded DER

Clustering

To provide higher scalability and availability for critical services, configure PingAccess in a clustered environment.

PingAccess clusters are made up of three types of nodes:

Administrative node

: Provides the administrator with a configuration interface.

Replica administrative node

: Provides the administrator with the ability to recover a failed administrative node using a manual failover procedure.

Engine node

: Handles incoming client requests and evaluates policy decisions based on the configuration replicated from the administrative node.

> **Note:**
>
> You can configure any number of clustered engines in a cluster, but you can only configure one administrative console and one replica administrative console in a cluster.

HTTP requests

HTTP Requests are used to match a served resource with the originating client when one or more reverse proxies are between the client and the served resource. For example, when a reverse proxy sits between the client and the PingAccess server or a PingAccess agent, the additional proxy might be identified as the client. Such proxies can be configured to inject additional headers to relay the originating client address.

Identity mappings

Identity mappings make user attributes available to back-end sites that use them for authentication. There are multiple types of identity mappings, each with different behavior and a distinct set of fields to specify the identity mapping behavior.

Key pairs

Key pairs are required for secure HTTPS communication. A key pair includes a private key and an X.509 certificate. The certificate includes a public key and the metadata about the owner of the private key.

PingAccess listens for client requests on the administrative console port and on the PingAccess engine port. To enable these ports for HTTPS, the first time you start up PingAccess, it generates and assigns a key pair for each port. These generated key pairs are assigned on the **HTTPS Listeners** page.

Additionally, key pairs are used by the mutual TLS site authenticator to authenticate PingAccess to a target site. When initiating communication, PingAccess presents the client certificate from a key pair to the site during the mutual TLS transaction. The site must be able to trust this certificate for authentication to succeed.

Listeners

Listeners monitor ports for incoming requests. PingAccess can place listeners on ADMIN, ENGINE, and AGENT ports.

Load balancing strategies

Load balancing strategies are used in a site configuration to distribute the load between multiple backend target servers. Load balancing settings are optional and are only available if more than one target is listed for a site. This functionality can replace a load balancer appliance between the PingAccess engine nodes and the target servers, allowing for a simpler network architecture.

The header-based strategy requires a header be included in the request that defines the target to select from the site configuration. This strategy has an option to fall back if the requested target is unavailable or if the header is missing from the request.

The round robin strategy has a sticky session option that permits a browser session to be pinned to a persistent backend target. This strategy works in conjunction with the availability profile to select a target based on its availability, and the load balancer does not select a target that is in a failed state.

Policies

Policies are rules, rule sets, or groups of rule sets applied to an application and its resources. Policies define how and when a client can access target sites. The policy manager is a rich drag-and-drop interface where you can manage policies by:

- Creating rules

- Building rule sets and rule set groups
- Applying them to applications and resources

When a client attempts to access an application resource identified in one of the policy's rules, rule sets, or rule set groups, PingAccess uses the information contained in the policy to decide whether the client can access the application resource and whether any additional actions need to take place prior to granting access. Rules can restrict access in a number of ways such as testing user attributes, time of day, request IP addresses, or OAuth access token scopes. Rules can also perform request processing, such as modifying headers or rewriting URLs.

Proxies

Configure settings to authenticate with a forward proxy server when PingAccess makes requests to sites or token providers.

Rules, rule sets, and rule set groups

Rules are the building blocks for access control and request processing. There are many types of rules, each with different behavior and a distinct set of fields to specify the rule behavior. Rule sets allow you to group multiple rules into re-usable sets which can be applied to applications and resources. Rule set groups can contain rule sets or other rule set groups, allowing the creation of hierarchies of rules to any level of depth. Rule sets and rule set groups can be applied to applications and resources as required.

Sites

Sites are the target applications or APIs that PingAccess gateway is protecting and to which authorized client requests are ultimately forwarded to.

Site authenticators

When a client attempts to access a target web site, that site can limit access to only authenticated clients. PingAccess integrates with those security models using site authenticators. PingAccess supports a variety of site authenticators that range from basic username and password authentication to certificate and token-based authentication. Create a site authenticator for the type of authentication the site requires.

Token provider

Token providers are used as a method of providing credentials for secure access to a given target.

Unknown resources

Unknown resources are resources for which there is no PingAccess definition. You can specify the default and per-agent handling behavior for unknown resource requests and configure custom error responses.

Virtual hosts

Virtual hosts enable PingAccess to protect multiple application domains and hosts. A virtual host is defined by the host name and host port.

Web sessions

Web sessions define the policy for web application session creation, lifetime, timeouts, and their scope. You can configure multiple web sessions to scope the session to meet the needs of a target set of applications. This improves the security model of the session by preventing unrelated applications from impersonating the end user.

How do I choose a deployment model?

supports three deployment models.

The gateway, agent, and sideband deployment models each have advantages and disadvantages. Review them before selecting a model for your environment.

Gateway model

What is the gateway model?

In the gateway model, traffic is initially directed to a PingAccess node, and PingAccess grants or denies access directly. The application in PingAccess is configured with the site as the destination.

Pros

- Less cross-team coordination required — You can implement and maintain a gateway deployment with less coordination with application teams because the PingAccess infrastructure is installed on separate systems from the web servers.
- Simpler setup — Because the PingAccess nodes are the only required components, this deployment model can be set up more quickly.
- Simpler upgrade — The only components that must be upgraded in a gateway deployment are the PingAccess nodes.

> **Note:**
>
> PingAccess can be upgraded with zero downtime in a clustered environment.

- Simpler troubleshooting — Issues are easier to isolate because there are fewer components sharing a system with the PingAccess infrastructure.
- Simpler logging — All transactions processed by PingAccess are audited by the engine node, making it easier to view logs for a specific event.

Cons

- Network impact — It requires that you restructure your existing network to route traffic through PingAccess.
- Additional network overhead — The overhead of an additional network hop can theoretically exceed a latency budget. This rarely happens in practice, and the agent model often makes a similar addition to latency, but in some environments it might occur.

Agent model

What is the agent model?

In the agent model, traffic is directed to the application, which has an agent plugin installed on the web server. The agent grants or denies access, and queries the PingAccess node when it requires additional information. The application in PingAccess is configured with the agent as the destination.

Pros

- No network changes — Because the PingAccess agents are installed on the web servers, no network changes are required.
- Minor performance improvements — In some cases, the agent can determine whether to grant access using cached data, which can reduce latency. In most cases, though, the agent must communicate with a PingAccess node, which results in latency similar to the gateway model.

Cons

- Greater maintenance effort — The agents must be maintained and upgraded independently on each web server.
- Unavailable features — Some features can't be used in an agent deployment. You can't:
  - Rewrite the request URL
  - Rewrite response headers
  - Rewrite request or response body content
- Cross-team coordination — Because the agents are installed directly on the web server, you might have to coordinate with other teams to install and maintain them.
- Complex tracking — Keeping track of all agents can be difficult.
- Difficult troubleshooting — Because the agent model involves more systems which can be varied in their OS and web server versions, troubleshooting issues can be more difficult.
- Version dependencies — Because the agent must be installed as a plugin on the web server, there are dependencies on the web server and operating system versions that are not present in the gateway model.
- Less centralized logging — To view the logging for a specific transaction, you must review the agent audit log and the web server access logs.

Sideband model

What is the sideband model?

In the sideband model, traffic is directed to an API gateway, such as Apigee or Mulesoft. The API gateway makes a back channel call to PingAccess to determine if it should grant or deny the request and determine what modifications should be made to the request or response. The application in PingAccess is configured with the sideband client as the destination.

Pros

- No network changes — Because the PingAccess sideband clients are installed on the API gateways, no network changes are required.
- Simpler troubleshooting — Issues are easier to isolate because there are fewer components sharing a system with the PingAccess infrastructure.
- Simpler logging — All transactions processed by PingAccess are audited by the engine node, making it easier to view logs for a specific event.

Cons

- Greater maintenance effort — The API gateway integration kits must be maintained and upgraded independently on each API gateway.
- Unavailable features — Some features may not be available depending on your API gateway.
- Additional network overhead — The overhead of an additional network hop can theoretically exceed a latency budget. This rarely happens in practice, and the gateway and agent models often make a similar addition to latency, but in some environments it might occur.
- Cross-team coordination — Because the sideband clients are installed directly on the API gateways, you might have to coordinate with other teams to install and maintain them.
- Complex tracking — A sideband deployment has more components to maintain than a gateway deployment.
- Version dependencies — Because the sideband client must be installed on the API gateway, there are dependencies on the API gateway and operating system versions that are not present in the gateway model.

Installing and Uninstalling PingAccess

Use this section for instructions on installing, configuring, and starting PingAccess.

- Install PingAccess:
  - On *Linux*
  - On *Windows*
- After installing PingAccess, you can:
  - *Start PingAccess*.
  - *Access the admin console for the first time*.
  - *Change configuration database passwords*.
- To stop, run, or uninstall PingAccess, see the following:
  - *Stopping PingAccess* on page 62
  - *Running PingAccess as a service* on page 62
  - *Uninstalling PingAccess* on page 66

Note:

The features documented here are affected by the settings in the configuration file. For more information, see the *Configuration file reference* on page 163.

Installation requirements

See the detailed system, hardware, and port requirements for installing PingAccess.

Before you install PingAccess, see the following requirements:

System requirements

PingAccess deployment and configuration is compatible with systems meeting certain requirements.

Ping Identity qualifies the following configurations and certifies that they are compatible with the product. Variations of these platforms, such as differences in operating system version or service pack, are supported until the platform or other required software creates potential conflicts..

Note:

PingAccess supports IPv4 addressing. There is currently no support for IPv6 addressing.

Operating systems

Note:

PingAccess was tested with default configurations of operating system components. If your organization has customized implementations or has installed third-party plug-ins, deployment of the PingAccess server might be affected.

- Amazon Linux 2
- Canonical Ubuntu 18.04

- Canonical Ubuntu 20.04
- Microsoft Windows Server 2016 (x64)
- Microsoft Windows Server 2019 (x64)
- Oracle Enterprise Linux 7.9 (Red Hat Compatible Kernel)
- Oracle Enterprise Linux 8.4 (Red Hat Compatible Kernel)
- Red Hat Enterprise Linux ES 7.9
- Red Hat Enterprise Linux ES 8.4
- SUSE Linux Enterprise Server 12 SP5
- SUSE Linux Enterprise Server 15 SP3

Docker support

View PingAccess's Docker Image on *DockerHub*. Visit PingIdentity's DevOps *documentation* for more information.

> **Note:**
>
> Only the PingAccess software is licensed under Ping Identity's end user license agreement, and any other software components contained within the image are licensed solely under the terms of the applicable open source/third party license.

- Version: Docker 19.03.5
  - Host operating system: Canonical Ubuntu 18.04.3 LTS

Virtual systems

Although Ping Identity does not qualify or recommend any specific virtual machine (VM) products, PingAccess runs well on several, including VMWare, Xen, and Windows Hyper-V.

> **Note:**
>
> This list of products is provided for example purposes only. We view all products in this category equally. Ping Identity accepts no responsibility for the performance of any specific virtualization software and does not guarantee the performance or interoperability of any VM software with its products.

Java environment

- Amazon Corretto 8 (64-bit)
- Amazon Corretto 11 (64-bit)
- OpenJDK 11 (64-bit)
- Oracle Java SE Runtime Environment (Server JRE) 8 (64-bit)
- Oracle Java SE Development Kit (JDK) 11 (64-bit)

> **Note:**
>
> The Ping Identity Java Support Policy applies. For more information, see this *Ping Identity Java support policy*.

PingFederate

The following versions of PingFederate are fully certified with this version of PingAccess:

- PingFederate 10.3

- PingFederate 11

Other versions of PingFederate are expected to be compatible with this version of PingAccess as per Ping's *end of life policy*.

Some features rely on a specific version of PingFederate to work. This will always be noted in the feature's description.

Browsers for end users

- Google Android (Chrome)
- Google Chrome
- Microsoft Edge
- Mozilla Firefox
- Apple iOS (Safari)
- Apple Safari

Browsers for admin console

- Google Chrome
- Microsoft Edge
- Mozilla Firefox

Note:

For a modern browser experience, we encourage our customers to migrate off of Microsoft Internet Explorer 11. For more information, including the end of support of Internet Explorer 11 in December 2021, see *Upgrade considerations* on page 70.

Audit event storage (external database)

- MS SQL Server 2019
- MS SQL Server 2017
- Oracle 19c
- PostgresSQL 9.6.1
- PostgresSQL 11.5

Hardware security module

See *Hardware security module providers* on page 317 for information about configuring a hardware security module.

Note:

You must use Java 8 if you plan to use a hardware security module.

- AWS CloudHSM 3.0.0
- Gemalto Safenet Luna SA (Firmware version 7.4)

Supported HTTP versions

- HTTP 1.1

OIDC Providers

These are the most common providers, however, Ping strives to support any third-party OIDC-compliant provider.

Provider	Provider Type
PingFederate	PingFederate
PingOne for Enterprise	Common
PingOne SSO	PingOne
Azure	Common
Okta	Common

Hardware requirements

PingAccess is supported on hardware that meets these requirements.

Note:

Although it is possible to run PingAccess on less powerful hardware, the following guidelines accommodate disk space for default logging and auditing profiles and CPU resources for a moderate level of concurrent request processing.

Although the requirements for different environments will vary, run PingAccess on hardware that meets or exceeds these specifications:

- Multi-CPU/Cores (8 or more)
- 4 GB of RAM
- 2.1 GB of available hard drive space

Port requirements

PingAccess uses ports and protocols to communicate with external components. This information provides guidance for firewall administrators to ensure that the correct ports are available across network segments.

Note:

Direction refers to the direction of requests relative to PingAccess:

Inbound requests

Requests that PingAccess receives from external components.

Outbound requests

Requests that PingAccess sends to external components.

Service	**Port details**	**Source**	**Description**
PingAccess administrative console	▪ Protocol: HTTPS ▪ Transport: TCP ▪ Default port: 9000 ▪ Destination: PingAccess admin console ▪ Direction: Inbound	PingAccess administrator browser, PingAccess administrative API REST calls, PingAccess replica admin and clustered engine nodes	Used for incoming requests to the PingAccess administrative console. Configurable using the `admin.port` property in the `run.properties` file. For more information, see the *Configuration file reference guide*. This port is also used by clustered engine nodes and the replica admin node to pull configuration data using the admin REST API.
PingAccess cluster communications port	▪ Protocol: HTTPS ▪ Transport: TCP ▪ Default port: 9090 ▪ Destination: PingAccess admin console ▪ Direction: Inbound	PingAccess administrator browser, PingAccess administrative API REST calls, PingAccess replica admin and clustered engine nodes	Used for incoming requests where the clustered engines request their configuration data. Configurable using the `clusterconfig.port` property in the `run.properties` file. For more information, see the *Configuration file reference guide*. This port is also used by clustered engine nodes and the replica admin node to pull configuration data using the admin REST API.

Service	Port details	Source	Description
PingAccess engine	▪ Protocol: HTTP/ HTTPS ▪ Transport: TCP ▪ Default port: 3000* **Note:** Any additional engine listener ports defined in the configuration must be open as well. ▪ Destination: PingAccess engine ▪ Direction: Inbound	Client browser, mobile devices, PingFederate engine	Used for incoming requests to the PingAccess runtime engine. Configurable using the `Listeners` configuration page. For more information, see the *PingAccess user interface reference guide*.
PingAccess agent	▪ Protocol: HTTP/ HTTPS ▪ Transport: TCP ▪ Default port: 3030 ▪ Destination: PingAccess engine ▪ Direction: Inbound	PingAccess agent	Used for incoming Agent requests to the PingAccess runtime engine. Configurable using the `agent.http.port` property of the `run.properties` file. For more information, see the *Configuration file reference guide*.
PingFederate traffic	▪ Protocol: HTTPS ▪ Transport: TCP ▪ Default port: 9031 ▪ Destination: PingFederate ▪ Direction: Outbound	PingAccess engine	Used to validate OAuth access tokens and ID tokens, make security token service (STS) calls for identity mediation, and return authorized information about a user. Configurable using the `PingFederate Settings` page within PingAccess. For more information, see the *PingAccess user interface reference guide*.

Installing PingAccess on Linux

Install PingAccess on a Linux system.

Before you begin

- Ensure the *installation requirements* are met.

- Ensure you are signed on to your system with appropriate privileges to install and run an application.

> **Note:**
>
> On Linux, install and run PingAccess as a non-root user.

- Install a *supported Java runtime*.
- The system or user environment variable `JAVA_HOME` must exist and be set to a value that represents the location of your Java installation, such as `usr/java/jdk 1.8.0_74`.
- The Java Runtime Environment (JRE) `/bin` directory (for example, `usr/lib64/jvm/jre/bin`) path must be added to the `PATH` variable so it is available for scripts that depend on it.
- You must have a `pingaccess.lic` license file.

> **Note:**
>
> If you do not have a PingAccess license, you can request an evaluation key at *https://support.pingidentity.com/s/*. During the first run of PingAccess, you will be prompted to upload the license file.
>
> If you are using an existing configuration file to configure the system, copy the configuration file to the system and rename it `data.json`. For more information about exporting the configuration from an existing system, see *Exporting PingAccess configurations* on page 340.

- If you are using an existing configuration file to configure the system, copy the configuration file to the system and rename it `data.json`. See *Exporting PingAccess configurations* on page 340 for more information about exporting the configuration from an existing system. This option is available only for standalone or administrative nodes.

Steps

1. Download the distribution `.zip` file.
2. Extract the distribution `.zip` file into your installation directory.
3. Optional: If you are using an existing configuration file to configure the system, move the `data.json` file to the `<PA_Home>/data/start-up-deployer` directory.

> **Note:**
>
> When you start PingAccess for the first time, if this configuration is present it will be imported. After a successful import, the `data.json` file is deleted. If the configuration is present but cannot be imported, PingAccess is not started.

> **Tip:**
>
> If you are deploying PingAccess in a cluster configuration, see the *configuration documentation*.

Next steps
Access the administrative console to complete the configuration.

Installing PingAccess on Windows using the installer

Install PingAccess on a Windows system using the installer.

Before you begin

- Ensure the *installation requirements* are met.
- Ensure that you are signed on to your system with appropriate privileges to install and run an application.

> **Note:**
>
> The Windows installer will ask for administrator privileges during installation.

- Install a *supported Java runtime*.
- The system environment variable `JAVA_HOME` must exist and be set to a value that represents the location of your Java installation, such as `C:\Program Files\Java\jre 1.8.0_91`.
- Add the `javapath` directory path (for example, `C:\Program Files\Oracle\Java\javapath`) to the `PATH` variable.

Steps

1. Download the PingAccess Windows installer.
2. Double-click on the installer icon to launch the PingAccess setup wizard.
3. Click **Next** and follow the prompts to complete the installation using the following information for your selected operational mode.

Operational Mode	Requirements
Standalone	Ports: ▪ PingAccess administrative console: TCP 9000 ▪ PingAccess agent protocol: TCP 3030
Clustered admin node	Ports: ▪ PingAccess administrative console: TCP 9000 ▪ Configuration query port: TCP 9090
Clustered replica admin node	Ports: ▪ PingAccess administrative console: TCP 9000 ▪ Configuration query port: TCP 9090 Prerequisites: ▪ You must install and configure a clustered admin node. ▪ A configuration data archive file for the replica admin node must be available. Consult PingAccess clustering documentation for more information. **Note:** Install the clustered replica admin node on a separate machine in the same network.

Operational Mode	Requirements
Clustered engine node	Ports: ▪ PingAccess agent protocol: TCP 3030 Prerequisites: ▪ You must install a clustered admin node. ▪ A configuration data archive file for the clustered engine node must be available. **Note:** Consult PingAccess clustering documentation for more information.

4. Copy the URL of the PingAccess administrative console that is displayed on the final screen of the PingAccess setup wizard, then click **Finish**.
5. To customize and finalize the PingAccess setup, paste the URL you copied into your web browser and connect to the administrative console of the instance you have just installed.

Next steps
Access the administrative console to complete the configuration.

Installing PingAccess on Windows from the command line

Install PingAccess on a Windows system from the command line.

Before you begin

- Ensure the *installation requirements* are met.
- Ensure that you are signed on to your system with appropriate privileges to install and run an application.
- Install a *supported Java runtime*.
- The system environment variable `JAVA_HOME` must exist and be set to a value that represents the location of your Java installation, such as `C:\Program Files\Java\jre 1.8.0_91`.
- Add the `javapath` directory path (for example, `C:\Program Files\Oracle\Java\javapath`) to the `PATH` variable.

> **Note:**
>
> If you are using an existing configuration file to configure the system, copy the configuration file to the system and rename it `data.json`. For more information about exporting the configuration from an existing system, see *Exporting PingAccess configurations* on page 340.

for more information about exporting the configuration from an existing system. This option is available only for standalone or administrative nodes.

Steps

1. Download the distribution `.zip` file.
2. Extract the distribution `.zip` file into your installation directory.

3. Optional: If you are using an existing configuration file to configure the system, move the `data.json` file to the `<PA_Home>/data/start-up-deployer` directory.

> **Note:**
>
> When you start PingAccess for the first time, if this configuration is present it will be imported. After a successful import, the `data.json` file is deleted. If the configuration is present but cannot be imported, PingAccess is not started.

Next steps
Access the administrative console to complete the configuration.

Starting PingAccess

After installing PingAccess, start the PingAccess service.

About this task

> **Note:**
>
> If you installed PingAccess using the Windows installer, the service is installed and started automatically.

Steps

1. In a command prompt or terminal window, change to the PingAccess `bin` directory.
   Choose from:
   - On Linux: `cd <PA_HOME>/bin`
   - On Windows: `cd <PA_HOME>\bin`
2. Start the `run` script for the platform.
   Choose from:
   - On Linux: `./run.sh`
   - On Windows: `run.bat`

   Result: PingAccess starts when you see the message “PingAccess running...” in the command window.

Accessing the administrative console for the first time

After installing and starting PingAccess, access the administrative console and perform configuration and first-time sign on tasks.

Steps

1. Launch your browser and go to https://*DNS_NAME*:9000.

   DNS_NAME is the fully-qualified name of the machine running PingAccess.

   > **Note:**
   >
   > If you have not yet installed a PingAccess license, the server redirects you to the **License Upload** window outside of the main UI. For more information, see the *PingAccess User Interface Reference Guide*.

2. Sign on with the default username and password.
   - **Username**: `Administrator`
   - **Password**: `2Access`
3. Read and accept the license agreement.
4. Change the default administrator password on the **First Time Login** page, and then click **Continue**.

   > **Note:**
   >
   > The new password must conform to the rules specified by the `pa.admin.user.password.regex` property in `run.properties`. For more information about these properties, see the *Configuration file reference* on page 163.

   Result: The PingAccess administrative console opens.

Result

After successfully signing on, PingAccess creates a backup of the current configuration to allow the administrator to revert any changes made, stored in `<PA_HOME>/data/archive`. You can control the number of backup files to keep using the `pa.backup.filesToKeep` property in the `run.properties` file. For more information about this property, see the *Configuration file reference* on page 163.

> **CAUTION:**
>
> Because the backup file contains your complete PingAccess configuration, ensure the file is protected with appropriate security controls in place.

Accessing the PingAccess administrative API

Access the PingAccess administrative API.

Steps

- Send an HTTP request to this URL: `https://`*`host`*`:`*`admin-port`*`/pa-admin-api/v3/`*`api-endpoint`*.

> **Note:**
>
> You must provide appropriate administrator credentials in the request.

Example

For example, the following cURL command will return a list of all defined applications by sending a GET request to the `applications` resource:

```
curl -k -u Administrator:Password1 -H "X-Xsrf-Header: PingAccess" https://
localhost:9000/pa-admin-api/v3/applications
```

- The **`-u Administrator:Password1`** parameter sends basic authentication header with the username `Administrator` and password `Password1`.
- The **`-k`** parameter specifies to ignore HTTPS certificate issues.
- The **`-H "X-Xsrf-Header: PingAccess"`** parameter sends an `X-XSRF-Header` with value `PingAccess`.

Accessing the interactive administrative API documentation

View interactive documentation for the administrative API endpoints.

Steps

1. Launch your browser and go to `https://`*`host`*`:`*`admin-port`*`/pa-admin-api/v3/api-docs/`.

   Example: `https://localhost:9000/pa-admin-api/v3/api-docs/`

   > **Note:**
   >
   > The browser might prompt you to enter your credentials.
2. Enter the administrator username and password.
3. Use the administrative API to perform a variety of administrative tasks, such as gathering information, as seen in the following example that uses the interactive administrative API documentation to see all defined applications:
   a. Click to expand the `/applications` endpoint.
   b. Click to expand the `GET` method (`GET /applications`).
   c. Enter parameters values or leave all blank.
   d. Click **Try It Out**.

      Result: The request URL, response body, response code, and response headers display.

Changing configuration database passwords

Rotate the database passwords for the PingAccess configuration database.

About this task

The PingAccess configuration database is protected by randomly-generated passwords on startup. You can rotate these passwords for additional security.

Steps

1. Open a terminal window and change to the `<PA_HOME>/bin` directory.
2. To ensure the `JAVA_HOME` environment variable is set correctly, enter the command `echo $JAVA_HOME`.
3. To ensure the proper Java executable is in your path, enter the command `java -version`.

   Note:

   If this command returns a value indicating that the Java executable is not a supported version, correct this issue before continuing.
4. Stop PingAccess.
5. Run the appropriate rotation script for your environment.
   Choose from:
   - `db-passwd-rotate.bat` on Windows
   - `db-passwd-rotate.sh` on Linux
6. Restart PingAccess.

Stopping PingAccess

Stop PingAccess as a prerequisite for maintenance or uninstallation tasks.

Steps

- Stop PingAccess:
  Choose from:
  - Press Ctrl+C in the command-prompt or terminal window.
  - If PingAccess is running on Windows, to terminate the script, press `y` when prompted.

Running PingAccess as a service

PingAccess can run as a service on Linux and Windows 64-bit operating systems, enabling PingAccess to start automatically when the operating system is started.

The service runs as the `root` (Linux) and `System` (Windows) user by default.

The following tasks let you manage PingAccess as a service:

- *Configuring PingAccess to run as a Linux systemv service* on page 63
- *Configuring PingAccess to run as a Linux systemd service* on page 64
- *Configuring multiple instances of PingAccess as Linux services* on page 65
- *Removing the PingAccess Linux service* on page 65
- *Configuring PingAccess to run as a Windows service* on page 65

- *Removing the PingAccess Windows service* on page 66

Tip:

Before performing the following procedures, ensure that PingAccess runs normally by manually starting the server. For more information, see *Run PingAccess for the First Time*.

Configuring PingAccess to run as a Linux systemv service

Configure PingAccess to run as a Linux systemv service, causing it to start automatically when Linux starts.

About this task

Note:

The service script will only start if <*JAVA_HOME*> and <*PA_HOME*> are set and the PingAccess license file is found.

Steps

1. Copy the PingAccess script file from `<PA_HOME>/sbin/linux/pingaccess` to `/etc/init.d`.
2. Optional: Create a new user to run PingAccess.
3. Create the folder `/var/run/pingaccess`.

   Note:

   Ensure the user who will run the service has read and write permissions to the folder.
4. Edit the script file `/etc/init.d/pingaccess` and set the values of following variables at the beginning of the script:
   - `export JAVA_HOME=`: specify the Java install folder
   - `export PA_HOME=` : specify the PingAccess install folder
   - `export USER=`: (optional) specify username to run the service, or leave empty for default
5. To register the service, from the `/etc/init.d` folder, run this command:

   ```
   chkconfig --add pingaccess
   ```
6. To make the service script executable, run this command:

   ```
   chmod +x pingaccess
   ```

Result

After registering, you can use the `service` command to control the PingAccess service. The available commands are shown.

start

Start the PingAccess service.

stop

Stop the PingAccess service.

restart

Restart the PingAccess service.

status

Show the status of the PingAccess service and the service process identifier (PID).

Note:

The command `service pingaccess status` displays the current status of the running PingAccess service.

Configuring PingAccess to run as a Linux systemd service

Configure PingAccess to run as a Linux systemd service, causing it to start automatically when Linux starts.

About this task

Note:

The service script will only start if *<JAVA_HOME>* and *<PA_HOME>* are set and the PingAccess license file is found.

Steps

1. Copy the configuration file from `<PA_HOME>/sbin/linux/pingaccess.service` to `/etc/systemd/system/pingaccess.service`.
2. In the `pingaccess.service` file, replace the following variables:
   - *${PA_HOME}*
   - *${PA_USER}*
   - *${PA_JAVA_HOME}*
3. To allow read/write activity on the service, run this command:

```
chmod 644 /etc/systemd/system/pingaccess.service
```

4. To load the systemd service, run this command:

```
systemctl daemon-reload
```

5. To enable the service, run this command:

```
systemctl enable pingaccess.service
```

6. To start the service, run this command:

```
systemctl start pingaccess.service
```

Configuring multiple instances of PingAccess as Linux services

Configure multiple instances of PingAccess on a single host as Linux services.

About this task

For hosts running multiple instances of PingAccess that need to be started as a service, follow the procedure used for *Configuring PingAccess as a Linux Service*, but make the following modifications to the script for each service.

Steps

- Use a unique script name for each instance.
- Use a separate directory structure for each instance in the filesystem.
- Configure the following settings in the script file for each instance:
  Choose from:
  - *<APPNAME>*: A unique value for each instance
  - *<PA_HOME>*: The path to the PingAccess instance
  - *<JAVA_HOME>*: The path to the Java installation folder
  - *<USER>*: Optional value for the user name used to run the service

Removing the PingAccess Linux service

Remove the PingAccess service from a Linux system.

About this task

> **Note:**
>
> The following commands must be run as the `root` user.

Steps

1. To stop the service, run the command `/etc/init.d/pingaccess stop`.
2. Run the command `chkconfig --delete pingaccess`.
3. Optional: Delete the `/etc/init.d/pingaccess` script.

Configuring PingAccess to run as a Windows service

Configure PingAccess to run as a Windows service, causing it to start automatically when Windows starts.

Before you begin
You must install PingAccess before configuring it. You can use the *Install PingAccess* procedure to install PingAccess from the command line.

> **Note:**
>
> Before performing this procedure, ensure that PingAccess runs normally by manually starting the server. For more information, see *Run PingAccess for the First Time*. If you installed PingAccess using the Windows installer, the service is installed and started automatically.

Steps

1. Ensure you are signed on with full administrator privileges.
2. Start a command prompt as an administrator.

3. In the command prompt, change to the `<PA_HOME>\sbin\windows` directory and run the `install-service.bat` script.
4. In Windows, go to **Control Panel** # **Administrative Tools** # **Services**.
5. From the list of available services, right-click **PingAccess Service** select **Start**.

   You can change the default **Start type** setting in the **Properties** dialog.

   Result: The service starts immediately and restarts automatically on reboot.

Configuring PingAccess to run as a Windows service from the command line

Configure PingAccess to run as a Windows service from the command line, causing it to start automatically when Windows starts.

Before you begin

- Install Pingaccess before configuring it. To install PingAccess from the command line, see *Installing PingAccess on Windows from the command line* on page 58.
- Ensure that PingAccess runs normally by manually starting the server. For more information, see *Run PingAccess for the First Time*.

Steps

1. Ensure you are signed on with full administrator privileges.
2. Change your directory to `<PA_HOME>\sbin\windows` and run the `install-service.bat` script.
3. To set the PingAccess service to start automatically, run the command `sc config PingAccess start= auto`.

   Result: The service starts immediately and restarts automatically on reboot.

Removing the PingAccess Windows service

Remove the PingAccess service from a Windows system.

Before you begin

Make sure you have PingAccess administrator privileges.

Steps

1. Open a command prompt.
2. Stop the PingAccess service.
3. Change the current directory to `<PA_HOME>\sbin\windows`.
4. Run the command `uninstall-service.bat`.
5. When the script has finished, remove the *<PA_HOME>* environment variable from the system.

Uninstalling PingAccess

Uninstall PingAccess.

About this task

To uninstall PingAccess:

Steps

1. *Stop PingAccess* as appropriate for your system.

2. If you installed PingAccess using the Windows installer, use one of these options:
   Choose from:
   - Double-click on the installer icon, then click **Remove** and follow the prompts.
   - Open the Windows **Add or Remove Programs** tool, then locate and uninstall PingAccess.
3. If PingAccess is installed on a Linux system and has been configured to start automatically, remove the PingAccess service.
   a. Run the command: `chkconfig --delete pingaccess`
   b. Optional: Delete the `/etc/init.d/pingaccess` script.
4. Delete the PingAccess installation directory if it is present.

Backing up and restoring PingAccess

This section provides instructions for backing up and restoring PingAccess.

The tools in this section let you create backups of your PingAccess environment and restore your environment from them. You should back up your environment regularly.

- If you need disaster recovery, see *Backing up and restoring PingAccess using a .zip archive* on page 67.
- If you need to restore an environment's configuration or to test in a new environment, see *Backing up and restoring PingAccess using a JSON file* on page 68.

Backing up and restoring PingAccess using a `.zip` archive

Use a `.zip` archive to back up and restorePingAccess.

Using a `.zip` archive is appropriate for disaster recovery because it uses automatically generated backups and restores the entire PingAccess configuration.

Note:

The system on which you are restoring must have the same major and minor version of PingAccess installed before you begin the restoration.

Backing up PingAccess using a `.zip` archive

Back up your PingAccess by copying a `zip` archive and additional customized files to another system.

About this task

To back up your PingAccess using a `.zip` archive for disaster recovery purposes:

Steps

1. In the PingAccess system, go to `<PA_HOME>/data/archive` and copy the most recent `.zip` archive to another system.

   These archives are automatically created when an administrative user authenticates to the administrative console. The maximum number of backups is specified by the `pa.backup.filesToKeep` property in the `run.properties` file.
2. Optional: If you have created or customized templates, copy the contents of the `<PA_HOME>/conf/template` directory to another system.
3. Optional: If you have created custom plugins, copy the contents of the `<PA_HOME>/deploy` directory to another system.

4. Optional: If you have created or customized localization, copy the contents of the *<PA_HOME>*/conf/localization directory to another system.

Restoring PingAccess using a .zip archive

Restore your PingAccess using a .zip archive and additional customized files.

About this task

This procedure restores the PingAccess configuration using a .zip archive and additional customization files. You can use this method for disaster recovery because it uses an automatically-generated file and restores the entire PingAccess configuration.

To restore your PingAccess using a .zip archive and additional customized files:

Steps

1. If PingAccess is not installed on the system, install the same version of PingAccess used to create the backup.

   Note:

   The maintenance version does not have to be identical. For example, a backup made on PingAccess 6.3.0 could be restored on 6.3.1.
2. Stop PingAccess.
3. Extract the backup .zip archive to *<PA_HOME>*.
4. Optional: If you backed up custom templates, copy the backed up content to the *<PA_HOME>*/conf/template directory.
5. Optional: If you backed up custom plugins, copy the backed up content to the *<PA_HOME>*/deploy directory.
6. Optional: If you backed up custom localization, copy the backed up content to the *<PA_HOME>*/conf/localization directory.
7. Restart PingAccess.

   Result: Your PingAccess configuration is reverted to the configuration in the backup archive.

Backing up and restoring PingAccess using a JSON file

Use a JSON file to back up and restore PingAccess.

This method is appropriate for reverting to a prior configuration or testing a configuration in a new environment. It uses manually-generated backups and restores the PingAccess database configuration.

Backing up PingAccess using a JSON file

Back up your PingAccess by copying a JSON file to another system.

About this task

Back up your PingAccess by copying a JSON file to another system.

Note:

Large PingAccess configurations can take upwards of 30 minutes to export. During an export, you cannot modify the PingAccess configuration.

Steps

1. Click **Settings** and then go to **System # Configuration Export/Import**.
2. Click **Export Configuration**.

   The downloaded file name is `pa-data-<timestamp>.json`.

   Note:

   The <*timestamp*> value is formatted `MM-DD-YYYY.hh.mm.ss`. For example, a date and time of January 31, 2020 1:35 PM would be encoded as `01-31-2020.13.35.00` in the file name.

3. Copy the generated file to another system.
4. If you plan to restore PingAccess in a new environment, copy the `<PA_Home>/conf/pa.jwk` file, and save it with the generated JSON file.

Restoring PingAccess using a JSON file

Restore your PingAccess using a JSON file.

About this task

The **Import Configuration** option is a version-specific tool for importing a previously exported configuration. PingAccess checks the exported JSON file to ensure that the file is not from a later version of PingAccess and is compatible with API v3 (PingAccess 5.0 or later).

Important:

This operation is destructive and overwrites any existing PingAccess configuration.

Note:

Large PingAccess configurations can take several hours to import. During an import, you cannot modify or read the PingAccess configuration.

To restore your PingAccess using a JSON file:

Steps

1. If you are restoring on a new environment, copy the saved `pa.jwk` file to the `<PA_Home>/conf/` directory.
2. Click **Settings** and then go to **System # Configuration Export/Import**.
3. Click **Import Configuration** and select the local file you want to use.
4. Click **Import**.
5. Click **Confirm** and make any changes indicated by the import process.

   Troubleshooting:
   If the import fails, click **View failures from last import** to view all of the errors logged during the import.
6. If the Agent or Admin listener key pairs change as a result of the import operation, restart PingAccess.
7. If the environment is clustered, ensure that the replica administrative node and engine nodes are using the proper engine keys, and if they aren't, re-save them to generate a new public key, and reconfigure them to use the newly generated key.

   For more information, see *Clustering in PingAccess* on page 156.

Upgrading PingAccess

This section provides instruction on how to upgrade PingAccess to the latest version depending on what type of environment you have.

After reviewing the *Upgrade considerations* on page 70, review the instructions specific to your environment:

- *Upgrading a PingAccess standalone version using the upgrade utility* on page 72
- *Upgrading a PingAccess cluster using the upgrade utility* on page 75
- *Upgrading PingAccess using the Windows installer* on page 78
- *Upgrading a PingAccess standalone version using the incremental update package* on page 79
- *Upgrading a PingAccess cluster using the incremental update package* on page 80

See *PingAccess zero downtime upgrade* on page 91 if you have a clustered environment and want to upgrade PingAccess without impacting resource availability.

For more information on what to do during or after your upgrade, see:

- *Performing post-upgrade tasks* on page 81
- *Restoring a PingAccess configuration backup* on page 90
- *Upgrade Troubleshooting* on page 90
- *Upgrade utility configuration file reference* on page 90

Upgrade considerations

Specific changes in PingAccess might require additional steps during an upgrade to the latest version.

Java 8

With continued product and hardware security module (HSM) integration improvements, customers should migrate off of Java 8. Ping Identity removed Java 8 support from the qualification process in December 2023. For more information, including Java 11 support, see *Installation requirements* on page 50.

Microsoft Internet Explorer 11

Ping Identity commits to deliver the best experience for our administrators and users. As we continue to improve our products, we encourage our customers to migrate off of Microsoft Internet Explorer 11. As of December 2021, we no longer include Internet Explorer 11 in our qualification process.

Performing a zero downtime upgrade from 6.0, 6.0.1, 6.0.2, or 6.0.3 to a later version

If you are using PingAccess 6.0, 6.0.1, 6.0.2, or 6.0.3, zero downtime upgrades to later versions can fail due to PKCE changes.

To prevent this issue, edit your existing web sessions and enable PKCE support.

1. Click **Access** and then click **Web Sessions** # **Web Sessions**.
2. Expand the web session and click .
3. Click **Show Advanced**.
4. Click **Enable PKCE**.
5. Edit the web session. Click **Save**.
6. Repeat these steps for each web session.

New templates for error and logout pages

PingAccess 6.1 updated several error and logout page template files to modernize their appearance and remove Ping branding:

- `general.loggedout.page.template.html`
- `general.error.page.template.html`
- `admin.error.page.template.html`
- `policy.error.page.template.html`

For more information about the templates, see *User-facing page customization reference* on page 144. If you have previously customized the template files, you can re-customize them using the new files.

Using a proxied PingFederate deployment

PingAccess 6.2 Beta introduced the ability to configure a proxied PingFederate deployment through PingAccess. If you have manually configured a similar deployment in an earlier version of PingAccess, you can continue to use it.

If you plan to switch, review the configuration options in the proxied PingFederate deployment UI to verify that it can manage the use cases of your current configuration, and remove any PingFederate-related applications before migrating the configuration.

SPA support

The SPA support check box was removed from the Admin UI for web type applications in PingAccess 6.2. As a result, all web type applications created in the admin UI in PingAccess 6.2 and later have SPA support enabled by default. The SPA support check box is still available for API type applications and Web + API type applications. For more information, see *Application field descriptions* on page 229.

Note:

If the default settings for SPA support with web type applications aren't compatible with your environment or with a specific application, create an authentication challenge policy to replace these settings and achieve the desired behavior. For more information, see *Configuring authentication challenge policies* on page 291.

Runtime state clustering removal

Support for runtime state clustering was removed in PingAccess 7.0. However, one benefit of runtime state clustering was that it enabled *rate limiting rules* to behave more consistently in a clustered environment. This was because runtime state clustering enabled all of the engines in a cluster to know the total number of requests for a resource, not just the requests which that engine received.

If you're using runtime state clustering with rate limiting rules, before upgrading to PingAccess 7.0 or later, you should either:

- Configure a load balancer sitting in front of a PingAccess cluster to stick the session to a specific engine. This ensures that a single PingAccess engine node applies the rate limiting rule. For more information, see *Load balancing strategies* on page 328.
- Tune down the **Max Burst Requests** interval on the rate limiting rule, following the *<current max burst requests interval>/<number of engines in cluster>* ratio.

Upgrading your environment

You can upgrade your PingAccess deployment. The procedure you use for the upgrade varies depending on your environment.

Note:
Before beginning your upgrade, review the *Consolidated Upgrade Guide* for general guidance about planning an upgrade.

- If you have a standalone PingAccess deployment not installed using the RHEL or Windows installer, use the *Upgrading a PingAccess standalone version using the upgrade utility* on page 72 procedure.
- If you have a PingAccess cluster, use the *Upgrading a PingAccess cluster using the upgrade utility* on page 75 procedure. If you have a PingAccess cluster and want to upgrade without any downtime, use the *Zero Downtime Upgrade* guide instead.
- If you installed PingAccess on Windows using the installer, use the *Upgrading PingAccess using the Windows installer* on page 78 procedure.
- If you are applying a maintenance update, such as upgrading from version 6.1 to version 6.1.1, you can apply the upgrade without using the upgrade utility. For more information, see *Upgrading a PingAccess standalone version using the incremental update package* on page 79 and*Upgrading a PingAccess cluster using the incremental update package* on page 80.

Upgrading a PingAccess standalone version using the upgrade utility

Upgrade a standalone PingAccess deployment to a newer version.

Before you begin

- If you are using PingAccess 3.2 or earlier, upgrade to PingAccess 4.3 or 5.3 before upgrading to the current version of PingAccess.
- Create a backup of your existing PingAccess configuration. If the upgrade fails, restore your environment from this backup.
- Review the release notes for every version between your current version and the target version.

  Important:

  In release 5.0, there are potentially breaking changes to the SDK for Java, Groovy scripts, and the administrative API. For information on these changes and the actions administrators might need to take, review the *Upgrade considerations* on page 70 and the *PingAccess Release Notes for release 5.0*.
- Verify that you have the following:
  - The PingAccess distribution `.zip` file
  - Your new PingAccess license file, if you plan to switch to a new license file
  - Sign on access to the PingAccess host, as the utility is run on the host
  - Administrator credentials for the running PingAccess instance
- Verify that basic authentication is configured and enabled for the running PingAccess instance.
- Verify that the PingAccess host is running.
- Verify that you are using the same account normally used to run PingAccess.

Important:

If you have set `security.overridePropertiesFile=false` in `$JAVA_HOME/jre/lib/java.security`, the upgrade utility might fail because the PingAccess upgrade utility uses an override to enable deprecated ciphers and protocols during the upgrade process.

About this task

Use the PingAccess upgrade utility to upgrade from PingAccess 4.0 or later, the source version, to the most recent version, the target version.

The upgrade utility starts an instance of PingAccess with an administrative listener on port 9001. This port number can be changed using the `upgrade.bat` or `upgrade.sh` **`-p`** parameter. This port configuration is only used for the upgrade. The configured port is used by the upgraded server when the upgrade is complete.

Any warnings or errors encountered are recorded in `log/upgrade.log`, as well as on-screen while the utility is being run. The upgrade uses an exit code of 0 to indicate a successful upgrade and an exit code of 1 to indicate failure.

Important:

If you are upgrading from version 4.3 or earlier, and your installation uses custom plugins, they must be rebuilt using the SDK version included in PingAccess 5.0 or later. Run the upgrade utility manually with the new `-i` command-line option to specify a directory containing the custom plugin JAR files and only the custom plugin JAR files. To migrate your custom plugins, see the *PingAccess Addon SDK for Java Migration Guide*.

Note:

During the upgrade, do not make any changes to the running PingAccess environment.

Steps

1. Copy the `.zip` file for the new PingAccess version to the PingAccess host and extract it.
2. Change to the new version's `/upgrade/bin` directory.
3. Run the PingAccess upgrade utility:
   Choose from:
   - On Windows: `upgrade.bat [-p <admin_port>] [-i <directory>] [-j <jvm_memory_options_file>] [-l <newPingAccessLicense>] [-s | --silent] <sourcePingAccessRootDir>`
   - On Linux: `./upgrade.sh [-p <admin_port>] [-i <directory>] [-j <jvm_memory_options_file>] [-l <newPingAccessLicense>] [-s | --silent] <sourcePingAccessRootDir>`

   Example: For example: `./upgrade.sh -p 9002 -i MyJARDir pingaccess-5.3`

Next steps

After you complete the upgrade, see *Performing post-upgrade tasks* on page 81.

PingAccess standalone upgrade parameters

The command-line parameters are the same regardless of the platform and are defined in the following table.

Parameter	**Value description**
-p <*admin_port*>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
-i <*directory*>	An optional directory containing additional library JAR files, such as plugins and Java Database Connectivity (JDBC) drivers, to be copied into the target installation. Beginning in version 6.0, JAR files are stored in the *<PA HOME>*/deploy folder. During an upgrade from versions earlier than 6.0, third-party JAR files are migrated from the `lib` folder to the `deploy` folder if no directory is specified. During an upgrade from version 6.0 or later, the contents of the `deploy` folder are migrated to the new *<PA HOME>*/deploy folder if no directory is specified.
<*sourcePingAccessRootDir*>	The PA_HOME for the source PingAccess version.
-l <*newPingAccessLicense*>	An optional path to the PingAccess license file to use for the target version. If not specified, the existing license is reused.
-j <*jvm_memory_options_file*>	An optional path to a file with Java virtual machine (JVM) options to use for the new PingAccess instance during the upgrade.
-s | --silent	Run the upgrade with no user input required. To use this option, specify the source version's credentials using environment variables.

Environment variables

You can specify the username and password for the source version using these environment variables:

- `PA_SOURCE_API_USERNAME` – The username for the source version's Admin API. This should be set to Administrator.
- `PA_SOURCE_API_PASSWORD` – The basic authorization password for the Administrator in the source version's Admin API.

Java virtual machine (JVM) memory options

You can include these options in the JVM memory options file. Memory amounts use `m` or `g` to specify the unit.

-
    - -Xms*<amount>* – Minimum heap size
    - -Xmx*<amount>* – Maximum heap size
    - -XX:NewSize=*<amount>* – Minimum size for the Young Gen space
    - -XX:MaxNewSize=*<amount>* – Maximum size for the Young Gen space
    - `-XX:+UseParallelGC` – Specifies that the parallel garbage collector should be used

You can copy the existing `<PA_HOME>/conf/jvm-memory.options` file to create a JVM memory options file for the upgrade.

Example

```
#Sample JVM Memory options file
-Xms512m
-Xmx1g
-XX:NewSize=256m
-XX:MaxNewSize=512m
-XX:+UseParallelGC
```

Upgrading a PingAccess cluster using the upgrade utility

Upgrade a PingAccess cluster to a newer version.

Before you begin

- If you are using PingAccess 3.2 or earlier, upgrade to PingAccess 4.3 or 5.3 before upgrading to the latest version.
- Create a backup of your existing PingAccess configuration. If the upgrade fails, you can restore your environment from this backup.
- Review the release notes for every version between your current version and the target version.

> **Important:**
>
> In PingAccess 5.0 or later, there are potentially breaking changes to the SDK for Java, Groovy scripts, and the administrative API. For information on these changes and the actions administrators might need to take, see the *Upgrade considerations* on page 70 and the *PingAccess Release Notes for release 5.0*.

- Verify the following:
  - Each node is using the same PingAccess version. You can check the version by viewing the `<PA_HOME>/lib/pingaccess-admin-ui-<version number>.jar` file.
  - The PingAccess administrative node is running.
  - Basic authentication is configured and enabled for the running PingAccess administrative node.
  - You have the `.zip` bundle for the target version of PingAccess.
- Verify that you are using the same account normally used to run PingAccess.

About this task

Use the PingAccess upgrade utility to upgrade a cluster from PingAccess 4.0 or later, the source version, to the most recent version, the target version.

> **Note:**
>
> The upgrade procedure causes some downtime. To upgrade a cluster with no downtime, see the *Zero Downtime Upgrade* guide.

The upgrade utility starts an instance of PingAccess with an administrative listener on port 9001. You can change this port number using the `upgrade.bat` or `upgrade.sh` **`-p`** parameter. This port configuration is only used for the upgrade. The configured port is used by the upgraded server when the upgrade is complete.

Any warnings or errors encountered are recorded in log/upgrade.log, as well as on-screen while the utility is being run. The upgrade uses an exit code of 0 to indicate a successful upgrade and an exit code of 1 to indicate failure.

> **Important:**
>
> If you are upgrading from version 4.3 or earlier, and your installation uses custom plugins, they must be rebuilt using the SDK version included in PingAccess 5.0 or later. Run the upgrade utility manually with the new -i command-line option to specify a directory containing the custom plugin JAR files and only the custom plugin JAR files. To migrate your custom plugins, see the *PingAccess Addon SDK for Java Migration Guide*.

> **Note:**
>
> During the upgrade, do not make any changes to the running PingAccess environment.

Steps

1. On the administrative node, extract the .zip file for the target version of PingAccess.
2. Go to the new version's /upgrade/bin directory.
3. Run the PingAccess upgrade utility:
   Choose from:
   - On Windows: upgrade.bat [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*
   - On Linux: ./upgrade.sh [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*
4. Review the upgrade log. If it records any manual post-upgrade tasks:
   a. Stop the source administrative console.
   b. Start the target administrative console using the *<PA_HOME>*/bin/run.sh command on Linux systems or the *<PA_HOME>*\bin\run.bat command on Windows systems.
   c. Perform any manual post-upgrade tasks recorded in the upgrade log.
   d. Shut down the upgraded administrative console.
5. Run the upgrade utility on the replica administrative node.
   Choose from:
   - On Windows: upgrade.bat [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*
   - On Linux: ./upgrade.sh [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*
6. Run the upgrade utility on each engine node.
   Choose from:
   - On Windows: upgrade.bat [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*
   - On Linux: ./upgrade.sh [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*

7. Shut down the entire cluster.
8. Start the upgraded administrative node.
9. Start the upgraded replica administrative node.
10. Start each upgraded engine node.

Next steps

After you complete the upgrade, see *Performing post-upgrade tasks* on page 81.

PingAccess cluster upgrade parameters

The command-line parameters are the same regardless of the platform, and are defined as follows.

Parameter definitions

Parameter	Value description
-r | --disable-config-replication	Disables configuration replication on the admin node. For more information about using this parameter in an upgrade, see the *Zero Downtime Upgrade*.
-p <*admin_port*>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
-i <*directory*>	An optional directory containing additional library JAR files, such as plugins, Java Database Connectivity (JDBC) drivers to be copied into the target installation. Beginning in version 6.0, `JAR` files are stored in the `<PA HOME>/deploy` folder. During an upgrade from versions earlier than 6.0, third-party `JAR` files are migrated from the `lib` folder to the `deploy` folder if no directory is specified. During an upgrade from version 6.0 or later, the contents of the `deploy` folder are migrated to the new `<PA HOME>/deploy` folder if no directory is specified.
<*sourcePingAccessRootDir*>	The PA_HOME for the source PingAccess version.
-l <*newPingAccessLicense*>	An optional path to the PingAccess license file to use for the target version. If not specified, the existing license is reused.
-j <*jvm_memory_options_file*>	An optional path to a file with Java virtual machine (JVM) memory options to use for the new PingAccess instance during the upgrade.

Parameter	Value description
-s | --silent	Run the upgrade with no user input required. To use this option, specify the source version's credentials using environment variables.

Environment variables

You can specify the username and password for the source version using these environment variables:

- PA_SOURCE_API_USERNAME – The username for the source version's Admin API. This should be set to Administrator.
- PA_SOURCE_API_PASSWORD – The basic authorization password for the Administrator in the source version's Admin API.

Java virtual machine (JVM) memory options

These options can be included in the JVM memory options file. Memory amounts use m or g to specify the unit.

-
  - -Xms*<amount>* – Minimum heap size.
  - -Xmx*<amount>* – Maximum heap size.
  - -XX:NewSize=*<amount>* – Minimum size for the Young Gen space.
  - -XX:MaxNewSize=*<amount>* – Maximum size for the Young Gen space.
  - -XX:+UseParallelGC – Specifies that the parallel garbage collector should be used.

You can copy the existing *<PA_HOME>*/conf/jvm-memory.options file to create a JVM memory options file for the upgrade.

Example

```
#Sample JVM Memory options file
-Xms512m
-Xmx1g
-XX:NewSize=256m
-XX:MaxNewSize=512m
-XX:+UseParallelGC
```

Upgrading PingAccess using the Windows installer

Upgrade PingAccess if you installed PingAccess using the Windows installer.

Before you begin

- If you are using PingAccess 3.2 or earlier, you must upgrade to PingAccess 4.3 or 5.3 before upgrading to PingAccess 6.0.
- Review the *Upgrade considerations* on page 70.

About this task

Important:

If additional JAR files, such as custom plugins and Java database connectivity (JDBC) drivers, have been added to the existing PingAccess /lib directory, the 5.0-Beta installer cannot be used to perform the upgrade. Instead, run the upgrade utility manually, using the `-i` command-line option to specify the JAR files to be included.

Steps

1. Download the installer.
2. Start the installer.

   Result: The existing installation is detected.
3. To upgrade the installation, click **Yes**.
4. If you are switching to a new license, select a license file and specify a temporary admin port.

   Note:

   The temporary admin port is not required when upgrading a cluster node.
5. Click **Next**.
6. Specify the administrator credentials. Click **Next**.

   Note:

   Administrator credentials are not required when upgrading a cluster node.
7. Click **Finish**.

Next steps

After completing the upgrade, *Performing post-upgrade tasks* on page 81.

Upgrading a PingAccess standalone version using the incremental update package

Upgrade a standalone PingAccess deployment to a newer version using the incremental update package.

Before you begin

- Make a backup copy of the PingAccess home directory. If the upgrade fails, use the backup copy to restore PingAccess.
- Review the release notes for every version between your current version and the target version.
- Verify that you have the following:
  - The PingAccess incremental update `.zip` file for the target version.
  - Administrator credentials for the running PingAccess instance.
- Verify that basic authentication is configured and enabled for the running PingAccess instance.
- Verify that the PingAccess host is running.

About this task

Use the PingAccess incremental update bundle to upgrade from PingAccess 6.3 or later, the source version, to the most recent maintenance release for that version of PingAccess, the target version. For example, upgrade PingAccess 6.3 to the most recent maintenance release for 6.3.

Steps

1. Stop PingAccess.
2. Open the `readme` file included in the extracted `.zip` bundle.
3. Make the file changes specified in the `readme` file.
4. Restart PingAccess.

Next steps

After you complete the upgrade, see *Performing post-upgrade tasks* on page 81.

Upgrading a PingAccess cluster using the incremental update package

Upgrade a PingAccess cluster to a newer version using the incremental update package.

Before you begin

- Make a backup copy of the PingAccess home directory. If the upgrade fails, use the backup copy to restore PingAccess.
- Review the release notes for every version between your current version and the target version.
- Verify that each node is using the same PingAccess version. You can check the version by viewing the <PA_HOME>/lib/pingaccess-admin-ui-<version number>.jar file.
- Verify that the PingAccess administrative node is running.
- Verify that basic authentication is configured and enabled for the running PingAccess administrative node.
- Download the PingAccess incremental update `.zip` file for the target version.

About this task

Use the PingAccess incremental update bundle to upgrade a cluster from PingAccess 6.3 or later, the source version, to the most recent maintenance release for that version of PingAccess, the target version. For example, upgrade PingAccess 6.3 to the most recent maintenance release for 6.3.

Note:

This upgrade procedure causes some downtime. To upgrade a cluster with no downtime, see the *Zero Downtime Upgrade* guide.

Steps

1. Upgrade the administrative node.
   a. Extract the `.zip` file for the target version of PingAccess.
   b. Open the `readme` file included in the extracted `.zip` bundle.
   c. Make the file changes specified in the `readme` file.
2. Upgrade the replica administrative node.
   a. Extract the `.zip` file for the target version of PingAccess.
   b. Open the `readme` file included in the extracted `.zip` bundle.
   c. Make the file changes specified in the `readme` file.
3. Upgrade each engine node.
   a. Extract the `.zip` file for the target version of PingAccess.
   b. Open the `readme` file included in the extracted `.zip` bundle.
   c. Make the file changes specified in the `readme` file.

4. Shut down the entire cluster.
5. Start the upgraded administrative node.
6. Start the upgraded replica administrative node.
7. Start each upgraded engine node.

Next steps

After you complete the upgrade, see *Performing post-upgrade tasks* on page 81.

Performing post-upgrade tasks

After upgrading your PingAccess deployment using the upgrade utility or the installer, you must perform several post-upgrade tasks to ensure that the target version works correctly.

About this task

To see details about the upgrade, examine `log/upgrade.log`. To see details about the migrated configuration data, examine `log/audit.log`.

Steps

1. Review any warnings returned by the upgrade utility and take the actions indicated in the table below.

   At the end of an upgrade, the PingAccess upgrade utility or installer records any manual steps that require user intervention both in the command-line output and in `log/upgrade.log` at the WARN level. Information that does not require user intervention is added to the `log/upgrade.log` at the INFO level.
2. Review the HTTP requests configuration to ensure the use of the IP source settings is appropriate for the environment.
3. Stop the source version of PingAccess.
4. Start the target version of PingAccess.

Troubleshooting

Warning text	Steps to take
`Resource <ResourceName> contains an invalid path prefix and cannot be migrated to the target version. Manual intervention is required.`	This occurs when the 2.1 path prefix contains functionality supported through a Java regex, but not by the wild card support in 3.1. The user must manually migrate the regex to 1 or more path prefixes in 3.1. For example, consider the 2.1 prefix, `/(app1|app2)`. This can be translated to a single resource in 3.1.1 with path prefixes of `/app1` and `/app2`.
`Resource <ResourceName> requires a case-sensitive path. This conflicts with its containing application, which requires a case-insensitive path. Manual intervention may be required.`	The upgrade utility identifies path prefixes in 2.1 that start with `/(?i)` as path prefixes that are case-insensitive, and sets the case-sensitivity flag on the application appropriately. However, if multiple resources in a new application use inconsistent case sensitivity settings, the utility cannot determine what the case sensitivity should be. 2.1 resources are case-sensitive by default.

Warning text	Steps to take
`Resource` *`<ResourceName>`* `requires a case-insensitive path. This conflicts with its containing application, which requires a case-sensitive path. Manual intervention may be required.`	This is the same as the previous setting, but with the requirement being for a case-insensitive path rather than a case-sensitive one.
`Resource` *`<ResourceName>`* `is disabled in the source version. Resources can no longer be individually disabled. Application` *`<ApplicationName>`* `has been disabled due to this constraint.`	In 2.1, individual resources can be disabled. In 3.1, only applications can be enabled or disabled. The upgrade utility takes the approach of disabling the application if any related resources are disabled. Check the final configuration and make sure this is the desired outcome. If it is not, the disabled resources need to be deleted, and the application needs to be enabled.
`Path prefix for resource` *`<ResourceName>`* `contains a '.' character. This will be treated as a literal '.' in the target version.`	In a 2.1 setup, there might be resource names that accidentally contain a '.', assuming it is a literal '.' rather than part of a regex. For example, any file extension type resources will probably not be escaping the '.'. This message is intended to bring this change in semantics to the user's attention. This action item will not show up if the user has correctly escaped the '.' character with the '\.' sequence .

<table>
<tr><th>Warning text</th><th>Steps to take</th></tr>
<tr><td><code>Resource <ResourceName> could not be migrated to the target version due to application context root conflicts. Manual intervention is required.</code></td><td>This message indicates that multiple resources that use the same virtual host, but a different web session or site must be mapped under the same context root in the same application to preserve semantics. For example, consider the following configuration:

Resource A:
Path Prefix: /hr
Virtual Host: internal.example.com
Web Session: W
Site: Z
Resource B:
Path Prefix: /sales
Virtual host: internal.example.com
Web Session: W
Site: Z
Resource C:
Path Prefix: /payroll
Virtual Host: internal.example.com
Web Session: V
Site: Z

This configuration triggers this error because these resources cannot be grouped in the same application, but they would need to be to preserve the semantics in the internal.example.com address space. This issue could be fixed by using rewrite rules to place Resource C or Resources A and B under a different namespace. For example, use <code>/intranet/sales</code> and <code>/intranet/hr</code> on the front-end and rewrite out the <code>/intranet</code> on the backend.</td></tr>
<tr><td><code>Application <ApplicationName> contains OAuth rules, but authenticates users with a web session. Unexpected results may occur.</code></td><td>2.1 allows OAuth rules to be attached resources that use a web session. While this configuration is likely invalid in the first place, it would be possible to include both a PingAccess cookie and OAuth token in requests and PingAccess would apply policy to the requests as configured. In 3.1, however, an API application and web application are mutually exclusive so the semantics of this particular configuration cannot be preserved.</td></tr>
</table>

Warning text	Steps to take
`The resource order for virtual host <VirtualHostName> has changed in the target version.`	The upgrade utility checks that the resource order is consistent before and after the upgrade. This message indicates that the resource order from 2.1 does not match 3.1. This is likely due to how context roots in applications are ordered in 3.1. For 3.1, applications are ordered based on their context root, where the longest context root is checked first during resource matching. One way to address this is to review and potentially change the application context root values associated with the virtual host to avoid URL overlaps between applications.
`Application <ApplicationName> is no longer associated with an identity mapping. A web session or an authorization server is required to use identity mappings.`	Indicates a misconfiguration in the source version. Check whether you intended to use an identity mapping for the application and associate an appropriate web session or authorization server if necessary.
`OAuth rule with id <RuleId> is no longer associated with application <ApplicationName> because application <ApplicationName> is not an OAuth application. Manual intervention might be required.`	Indicates a misconfiguration in the source version. Check whether the OAuth rule is necessary to implement the desired access control policy.
`OAuth RuleSet with id <RuleSetId> is no longer associated with application <ApplicationName> because application <ApplicationName> is not an OAuth application. Manual intervention might be required.`	Indicates a misconfiguration in the source version. Check whether the OAuth RuleSet is necessary to implement the desired access control policy.
`Resource <ResourceName> from application with id <ApplicationId> was not migrated because the application is a web application while the resource has OAuth rules. Manual intervention might be required.`	Indicates a resource associated with the application is associated with OAuth rules. This is likely a misconfiguration, and it is necessary to evaluate whether this was intended or not.
`Upgrade created availability profile for site <SiteName>. A more descriptive name might be required.`	Indicates that an availability profile was created for the site during the upgrade. You might want to give the availability profile a more descriptive name.
`Application <ApplicationName> and associated resources were not migrated. The context root of /pa is reserved. Manual intervention might be required.`	The /pa context root was allowed as a valid context root in PingAccess 3.0 and is no longer allowed.

Warning text	Steps to take
Resource *<ResourceName>* from application with id *<ApplicationId>* was not migrated because the /pa prefix is reserved when the application context root is /. Manual intervention may be required.	The */pa* path prefix was allowed as a valid path prefix in PingAccess 3.0 and is no longer allowed.
The OAuth Groovy script rule no longer controls the realm in the response sent for an unauthorized OAuth request.	With PingAccess 3.2, realms moved to the application. The *Realm* can still be set using the PingAccess admin API interface. With the change in context for how realms are applied, it is necessary to check existing OAuth Groovy rules to ensure that they behave as expected. This message is shown if any OAuth Groovy rules exist in the migrated configuration.
The property *<PropertyName>* was set to a blank value to maintain compatibility. Set this to *<PropertyName>*=*<PropertyValue>*.	New security headers properties values are not set during an upgrade to preserve the behavior from the source release in the upgrade. If there is no reason not to in your environment, update the `run.properties` file with the recommended setting.
As a security enhancement, the default value of *<CipherList>* has changed with this version of PingAccess. Your existing ciphers remain unchanged. Use the default value: *<PropertyName>*=*<CipherList>*.	This message applies to the `admin.ssl.ciphers`, `engine.ssl.ciphers`, and `agent.ssl.ciphers` lists. This message is displayed if the upgrade source version cipher lists are changed from the defaults. Update the configuration with the new default value if possible.
The property *<PropertyName>* was set to a blank value to maintain compatibility. Set this to *<PropertyName>*=*<CipherList>*.	This message applies to the `site.ssl.protocols`, `site.ssl.ciphers`, `pf.ssl.protocols`, and `pf.ssl.ciphers` settings. The upgrade utility sets these values as empty values to maintain backwards compatibility, but the recommended value should be used if possible.
The host for virtual host *<VirtualHost>*:*<Port>* already has a keypair associated with it. The keypair previously associated with this virtual host was removed. Only one keypair can be associated with a given host.	If a virtual host has more than one key pair associated with it, only one key pair will be associated with it after the upgrade completes. This message displays to indicate which key pair was used.
Application with name *<ApplicationName>* not migrated as the context root *<Path>* was a reserved path.	If an application's context root is a reserved PingAccess path, the application will not be migrated. The indicated application will need to be created with a context root that does not conflict with the reserved path.

Warning text	Steps to take
`Resource with name` *`<ResourceName>`* `not migrated as the path` *`<Path>`* `was a reserved path.`	If a resource path is a reserved PingAccess path, the application will not be migrated. The indicated application will need to be created with a context root that does not conflict with the reserved path.
`The CIDR rule with name` *`<RuleName>`* `is associated with an agent application named` *`<ApplicationName>`* `and overrides the IP source configuration. A new Agent rule should be created that does not override the IP source.`	With changes in IP source header handling, additional options are available to override the headers used to identify the source address. When an agent is involved, the changes in IP source handling might cause the specified rule to not behave as expected.
`Require HTTPS option on application` *`<ApplicationName>`* `was set to` *`<Setting>`* `as virtual host had port` *`<Port>`*`. Please verify this setting is correct.`	The upgrade utility attempts to set the `Require HTTPS` option based on the virtual host associated with an application during an upgrade. This message is an advisory to just verify that the setting was properly detected.
`Virtual host` *`<VirtualHost>`* `was not migrated. An existing virtual host existed with the same logical name` *`<VirtualHost>`*`.`	Virtual host names are now case-insensitive. During the upgrade, after making the names case-insensitive, a duplicate virtual host was identified. It will be necessary to either recreate the virtual host with a new name, or to modify the configuration so the proper virtual host is migrated to the upgraded system.
`Renamed virtual host's hostname from` *`<VirtualHost`* `to` *`<NewVirtualHost>`* `due to virtual host spec compliance issue`	If a virtual host name contains an underscore (_) character, that does not conform to host naming requirements. In this instance, the underscore will be renamed to the string *a-z*. For example, if a virtual host named *<my_virtual_host>* is migrated, the new name will be *<mya-zvirtuala-zhost>*.
`Removed HTTP request rule with name` *`<RuleName>`*`, this rule must be converted to a Groovy script rule. Manual intervention might be required.`	When an HTTP request rule is migrated from an earlier release of PingAccess, rules that specify a source of *<Body>* are not migrated. A Groovy script rule can be used to perform a similar match, but the details of such a Groovy script require administrator intervention. A simple Groovy script rule that would perform a similar function might be: `requestBodyContains('value')` A script should be constructed that performs additional validation to ensure the rule passes only when desired. A generic match like this could lead to unexpected results depending on what content might be in the request body.

Warning text	Steps to take
The property *<PropertyName>* uses a customized value. "Your original value has not been modified. You may encounter startup or connection problems if this value is not supported by the JVM."	When migrating SSL settings between versions of PingAccess that use different Java virtual machine (JVM) or JDK versions, custom settings might not be compatible. If the protocols or ciphers used are not compatible with the target JVM or JDK, this message indicates which settings need to be manually updated. The *PropertyName* value can be any of the following values: • `site.ssl.protocols` • `site.ssl.ciphers` • `pf.ssl.protocols` • `pf.ssl.ciphers` • `admin.ssl.protocols` • `admin.ssl.ciphers` • `engine.ssl.protocols` • `engine.ssl.ciphers` • `agent.ssl.protocols` • `agent.ssl.ciphers`
Rule with ID *<RuleId>* and name *<RuleName>* was not migrated as matcher was invalid for the Groovy rule type. Invalid rules were removed from RuleSet *<RuleSetName>* which resulted in an empty set. *The RuleSet was removed. Please check your policy configuration.* Invalid rules were removed from RuleSet *<RuleSetName>*. Please check your policy configuration. Invalid rules were removed from application *<ApplicationName>*. Please check your policy configuration. Invalid RuleSets were removed from application *<ApplicationName>*. Please check your policy configuration. Invalid rules were removed from resource *<resource name>* on application *<ApplicationName>*. Please check your policy configuration. Invalid RuleSets were removed from Resource '*resource name*' on Application '*ApplicationName*'. Please check your policy configuration.	These messages might appear if the source PingAccess installation has misconfigured Groovy Rules. This indicates that you are not permitted to add an OAuth rule to an Application of type Web by editing an existing rule set. Groovy or OAuth Groovy rules will not be migrated for the following reasons: • The OAuth Groovy rule was applied to a Web application. • The Groovy or OAuth Groovy uses a matcher that is not appropriate for the application type. Check the policy configuration.

Warning text	Steps to take

`Rule with name` *<RuleName>* `has been removed from RuleSet with name` *<RuleSetName>*`. Multiple rate limiting rules with the same policy granularity cannot be included in a RuleSet."`
 	
`Rule with name` *<RuleName>* `has been removed from RuleSet with name` *<RuleSetName>*`. Multiple cross-Origin request rules cannot be included in a RuleSet."`
	The upgrade utility supports migrating a rule set containing multiple cross-origin resource sharing (CORS) or rate limiting rules with the same policy granularity. The upgrade utility will generate new action items, indicating that rules were removed from a rule set.
	
	These messages indicate that if both rules exist, there is a restriction to a single rate limiting or CORS rule. Please check to confirm that you have applied the correct rule to the policy.

`One or more notifications were issued while migrating from version` *<SOURCE>* `to version` *<TARGET>*`.`
 	
`Setting clusterconfig.enabled to false`
 	
`The new configuration query port feature has been disabled for backward compatibility. Please refer to the PingAccess clustering documentation before enabling this feature.`
	The new cluster config query port is enabled by default for new PingAccess 4.0 installations when running in CLUSTERED_CONSOLE or CLUSTERED_CONSOLE_REPLICA mode.
	
	During the upgrade process to version 4.0, the new cluster config query port is disabled. Messages are written to `upgrade.log` and `audit.log` to indicate this cluster configuration change was made.
	
	See the PingAccess clustering documentation before enabling this feature.

`One or more notifications were issued while migrating from version` *<SOURCE>* `to version` *<TARGET>* `For backward compatibility, when connecting to a protected, TLS SNI-enabled site, PingAccess will set the SNI server_name to the configured target host and not the HTTP request Host header value. Please refer to PingAccess' upgrade documentation for more information.`	During upgrades to release 4.0 and higher, the upgrade utility sets the value of `pa.site.tls.sni.legacyMode` to `true` to maintain compatibility with existing configurations. This property is controlled in the `run.properties` file and is not enabled on new installs.
`Localization property` *<{property name}>* `was added to pa-messages.properties. Any customized localization files should be updated.`	This message appears if new language properties are added between the source and target PingAccess versions and you have added additional language files or modified the en or en_US files. Update any customized files as required.
`Localization property` *<{property name}>* `in pa-messages.properties was modified. Any customized localization files should be updated.`	This message appears if the language properties have changed between the source and target PingAccess versions and you have added additional language files or modified the en or en_US files. Update any customized files as required.

Warning text	Steps to take
`Localization property <{property name}> was removed from pa-messages.properties. This property can be removed from any customized localization files.`	This message appears if the language properties have been removed between the source and target PingAccess versions and you have added additional language files or modified the en or en_US files. Update any customized files as required.
`WebSessionManagement contained an invalid cookie name. Replaced <{old cookie name}> with <{new cookie name}>. Please validate your configuration.`	This message appears if the WebSessionManagement has an invalid cookie name. Invalid characters are replaced with an underscore. Update any references as required.
`Legacy authentication requirements policy evaluation has been enabled to maintain backward compatibility with earlier versions of PingAccess. To disable this setting, remove the pa.policy.eval.acr.v42 property from run.properties.`	This message appears on upgrade to release 4.3 or later if you have one or more authentication requirements rules. You can make adjustments to configured rules so you can remove this property or you can maintain the property to leave existing rules unaffected.
`Property pa.audit.log.applicationResourceIdsAsIntegers was set to true in run.properties to maintain existing behavior. In order to log the ID of Global Unprotected Resources, this property should be removed or should be set to false (default). However, a value of false (default) will result in resourceId and applicationId audit logging fields being logged as strings, not integers, which may require audit logging database schema changes if these values are currently being used.`	This message appears on upgrade to release 5.1 or later to support the existing logging behavior of application resource IDs as integers. The default behavior of release 5.1 and later is to log these IDs as strings. You can choose to log application resource IDs as strings after the upgrade by removing, or setting to false, the applicable property in the `run.properties` file. This change might require a modification to the audit logging database schema.
`Invalid resource method <Method> was removed from resource <ResourceName> on application <ApplicationName>.`	This message appears on upgrade to release 5.3 or later if the source version has an application resource that contains a method with whitespace. The resource is preserved by the upgrade, but the method is removed.
`Invalid resource <{name}> on application <{name}> was removed because it did not have any valid methods.`	This message appears on upgrade to release 5.3 or later if all of the methods associated with a resource were removed with an `Invalid resource method` error. The resource is not migrated by the upgrade.

Warning text	Steps to take
`As of PingAccess 6.0, runtime state clustering using JGroups has been deprecated. Deployments relying on runtime state clustering will continue to function but the functionality will be replaced in a future version.`	This message appears on an upgrade to release 6.0 or later. The runtime state clustering feature is deprecated.

Restoring a PingAccess configuration backup

If an upgrade fails, restore your PingAccess configuration using an automatically generated backup.

About this task

PingAccess automatically creates a backup `.zip` file each time an administrative user authenticates to the administrative console. These backups are stored in `<PA_HOME>/data/archive`, with a maximum number of backups configurable using the `pa.backup.filesToKeep` configuration parameter in `run.properties`.

CAUTION:

This operation will replace your current configuration settings.

Steps

1. Stop PingAccess.
2. Extract the backup file to `<PA_HOME>`.
3. Restart PingAccess.

   Result: Your PingAccess configuration is reverted to the state in the backup archive that was restored.

Upgrade Troubleshooting

This table lists some potential problems and resolutions you might encounter while upgrading PingAccess.

Issue	Resolution
Upgrade from version 4.3 or earlier fails due to Groovy rule changes.	To verify your Groovy scripts are prepared for the upgrade, review the *Groovy development reference guide* and the *Upgrade considerations* on page 70.
Custom plugins are missing after upgrade.	Manually add the custom plugins to the `<PA Home>/deploy` directory.

Upgrade utility configuration file reference

This configuration file reference provides an overview of configurable parameters used by the upgrade utility. These parameters are configured in the `<UU_HOME>/conf/run.properties` file.

pa.upgrade.source.ssl.ciphers

Defines the type of cryptographic ciphers available for use with the source PingAccess pa.upgrade.source.ssl.protocols

Defines the protocols available for use with the source PingAccess

pa.upgrade.target.ssl.ciphers

Defines the type of cryptographic ciphers available for use with the target PingAccess. If not specified, the Java virtual machine (JVM) default values are used.

pa.upgrade.target.ssl.protocols

Defines the protocols available for use with the target PingAccess. If not specified, the JVM default values are used.

pa.upgrade.http.client.connection.timeout.ms

Defines, in milliseconds, the amount of time to wait before timing out the connection to the HTTP client. The default value is 3600000.

pa.upgrade.http.client.socket.timeout.ms

Defines, in milliseconds, the HTTP client socket timeout. The default value is 3600000.

PingAccess zero downtime upgrade

A zero downtime upgrade allows you to upgrade your clustered PingAccess environment to the latest version with no impact to resource availability or existing user sessions.

Though this procedure is applicable to any PingAccess cluster upgrade to version 5.0 or later, there are minor variations depending on your PingAccess source version. Those variations are clearly described where applicable.

Note:

Some steps, particularly those related to working with a load balancer, are dependent on your environment. It is expected that you are familiar with the tasks required by these steps. This document does not offer detailed instruction on performing these tasks.

You can upgrade from any version using the upgrade utility, or you can upgrade from version 6.1 to the latest maintenance release using the incremental update bundle. This procedure includes the steps for both methods.

Important:

To achieve a successful upgrade, perform the tasks in this document in the order that they are presented. Deviation from these tasks might result in a failed upgrade, system downtime, or both.

Note:

If you are using PingAccess 3.2 or earlier, you must upgrade to PingAccess 4.3 or 5.3 before upgrading to PingAccess 6.1.

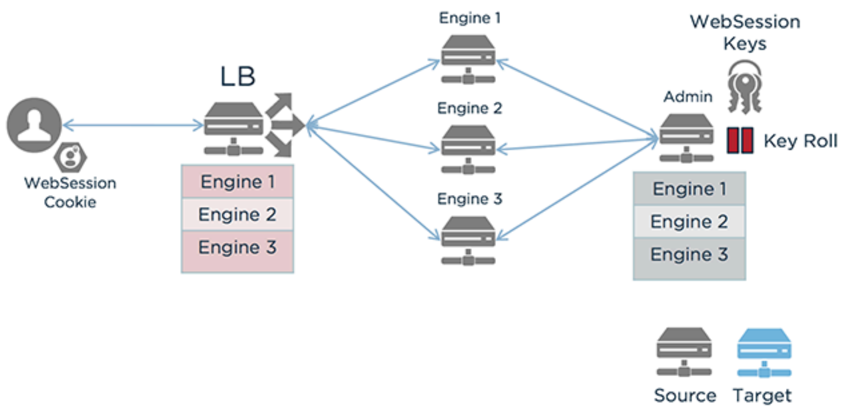
Before you begin, review the *Upgrade considerations* on page 70.

To begin the upgrade process, disable key rolling to prevent active sessions from being invalidated. For more information, see *Disabling key rolling*.

Disabling key rolling

Disable key rolling to prevent active sessions from being invalidated during the upgrade process. This is a temporary modification, and you will reenable key rolling at the end of the upgrade process.

There are different procedures at this stage depending on the source version of PingAccess.



- *Disabling key rolling in PingAccess 6.0 or later* on page 92
- *Disabling key rolling in PingAccess 5.2 or 5.3* on page 92
- *Disabling key rolling in PingAccess 5.0 or 5.1* on page 93
- *Disabling key rolling in PingAccess 4.3 or earlier* on page 93

Next, you will *upgrade the Admin node*.

Disabling key rolling in PingAccess 6.0 or later

If the source is PingAccess 6.0 or later, you can disable key rolling.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Auth Token Management**.
2. In the **Auth Token Management** section, deselect **Key Roll Enabled**.
3. Click **Save**.
4. Click **Access** and then go to **Web Sessions** # **Web Session Management**.
5. In the **Web Session Management** section, deselect **Key Roll Enabled**.
6. Click **Save**.
7. Click **Access** and then go to **Token Validation** # **OAuth Key Management**.
8. In the **OAuth Key Management** section, deselect **Key Roll Enabled**.
9. Click **Save**.

Next steps

Next, you will *upgrade the Admin node*.

Disabling key rolling in PingAccess 5.2 or 5.3

If the source is PingAccess 5.2 or 5.3, you can disable key rolling.

Steps

1. Go to **Settings** # **Access** # **Identity Mappings**.

2. In the **Auth Token Management** section, deselect **Key Roll Enabled**.
3. Click **Save**.
4. Go to **Settings** # **Access** # **Web Sessions**.
5. In the **Web Session Management** section, deselect **Key Roll Enabled**.
6. Click **Save**.
7. Go to **Settings** # **System** # **Token Validation**.
8. In the **OAuth Key Management** section, deselect **Key Roll Enabled**.
9. Click **Save**.

Next steps

Next, you will *upgrade the Admin node*.

Disabling key rolling in PingAccess 5.0 or 5.1

If the source is PingAccess 5.0 or 5.1, you can disable key rolling.

Steps

1. Go to **Settings** # **Access** # **Identity Mappings**.
2. In the **Auth Token Management** section, deselect **Key Roll Enabled**.
3. Click **Save**.
4. Go to **Settings** # **Access** # **Web Sessions**.
5. In the **Web Session Management** section, deselect **Key Roll Enabled**.
6. Click **Save**.

Next steps

Next, you will *upgrade the Admin node*.

Disabling key rolling in PingAccess 4.3 or earlier

If the source is a version of PingAccess earlier than 5.0, you can set the key rolling interval to a value that allows enough time for the upgrade to be completed successfully.

Steps

1. Go to **Settings** # **Access** # **Identity Mappings**.
2. In the **Auth Token Management** section, specify a **Key Roll Interval** of 240 (10 days).
3. Click **Save**.
4. Go to **Settings** # **Access** # **Web Sessions**.
5. In the **Web Session Management** section, specify a **Key Roll interval** of 240 (10 days).
6. Click **Save**.

Next steps

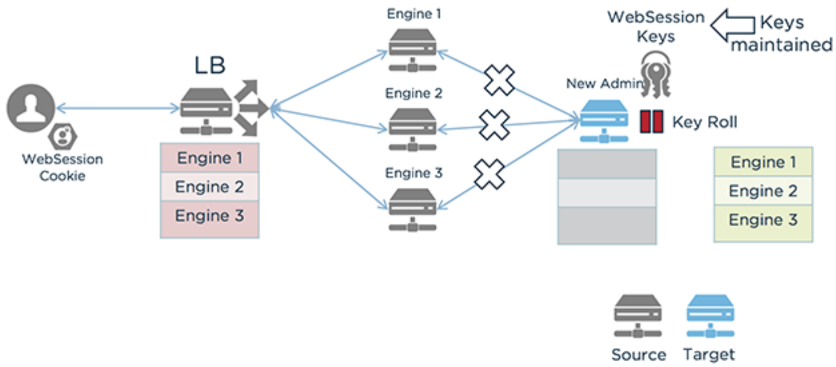
Next, you will *upgrade the Admin node*.

Upgrading the administrative node

Upgrade the PingAccess administrative node using the PingAccess Upgrade Utility. You will use the `-r` switch to disable configuration replication on the target version.

Before you begin

For more information on upgrading PingAccess, see *Upgrade PingAccess*.



Before beginning the upgrade process, make sure you have:

- Ensured PingAccess is running
- Downloaded the PingAccess *distribution `.zip` file* or the incremental update bundle and extracted it.
- The PingAccess license, if you are switching to a new license file
- Administrator credentials
- Basic Authentication enabled

About this task

Any warnings or errors encountered are recorded in `log/upgrade.log`, as well as on the screen while the utility is running. The upgrade uses an exit code of `0` to indicate a successful upgrade and an exit code of `1` to indicate failure.

Important:

If you are upgrading from version 4.3 or earlier, and your installation uses custom plugins, they will need to be rebuilt against the new (5.0) SDK. You will then run the upgrade utility manually with the new **`-i`** command-line option to specify a directory containing the custom plugin jars and only the custom plugin jars. To migrate your custom plugins, see the *PingAccess Addon SDK for Java Migration Guide*.

During the upgrade, it is important to not make any changes to the running PingAccess environment.

Steps

1. If you are using the upgrade utility, change to the new version's `/upgrade/bin` directory on the command line. For example:

```
cd /pingaccess-6.1.0/upgrade/bin
```

2. If you are using the incremental update bundle, disable configuration replication for the replica administrative node.
    a. In a browser, go to `https://<PingAccessHost>:9000/pa-admin-api/v3/api-docs/`.
    b. Expand the **/adminConfig/replicaAdmins** endpoint.
    c. Click the **GET /adminConfig/replicaAdmins** operation.
    d. Click **Try it out!** and note the `id` for the replica admin.
    e. Click the **GET /adminConfig/replicaAdmins/{id}** operation.
    f. Enter the `id` of the replica admin you want to update and click **Try it out!**
    g. Copy the **Response Body**.
    h. Click the **PUT /adminConfig/replicaAdmins/{id}** operation and enter the `id` of the replica admin you want to update.
    i. Paste the **Response Body** you copied and change `"configReplicationEnabled"` to `false`.
    j. Click **Try it out!**

        Result:

        If the operation is successful, you will receive a **Response Code** of **200**.
3. If you are using the incremental update bundle, disable configuration replication for each engine node.
    a. In a browser, go to `https://<PingAccessHost>:9000/pa-admin-api/v3/api-docs/`.
    b. Expand the **/engines** endpoint.
    c. Click the **GET /engines** operation.
    d. Click **Try it out!** and note the engine `id` for each engine.
    e. Click the **GET /engines/{id}** operation.
    f. Enter the `id` of the engine you want to update and click **Try it out!**
    g. Copy the **Response Body**.
    h. Click the **PUT /engines/{id}** operation and enter the `id` of the engine you want to update.
    i. Paste the **Response Body** you copied and change `"configReplicationEnabled"` to `false`.
    j. Click **Try it out!**

        Result:

        If the operation is successful, you will receive a **Response Code** of **200**.

4. Upgrade the system:
   Choose from:

   - If you are using the upgrade utility on a Windows system, use this command: upgrade.bat -r [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*.

     For example:

     ```
     upgrade.bat -r ../pingaccess-5.3.0
     ```
   - If you are using the upgrade utility on a Linux system, use this command: ./upgrade.sh -r [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*.

     For example:

     ```
     ./upgrade.sh -r ../pingaccess-5.3.0
     ```
   - If you are using the incremental update package, open the readme file and make the file changes specified in the readme.

Important:

The -r switch will disable configuration replication on the administrative node. You will re-enable configuration replication for each node as part of the upgrade process.

Parameter definitions

The command-line parameters are the same regardless of the platform, and are defined as follows:

Parameter	Value description
-r | --disable-config-replication	Disables configuration replication on the administrative node.
-p *<admin_port>*	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
-i *<directory>*	An optional directory containing additional library JAR files (for example, plugins, JDBC drivers) to be copied into the target installation. Beginning in version 6.0, JAR files are stored in the *<PA HOME>*/deploy folder. During an upgrade from versions earlier than 6.0, third-party JAR files are migrated from the lib folder to the deploy folder if no directory is specified. During an upgrade from version 6.0 or later, the contents of the deploy folder are migrated to the new *<PA HOME>*/deploy folder if no directory is specified.
<sourcePingAccessRootDir>	The PA_HOME for the source PingAccess version.

Parameter	**Value description**
-l <*newPingAccessLicense*>	An optional path to the PingAccess license file to use for the target version. If not specified, the existing license is reused.
-j <*jvm_memory_options_file*>	An optional path to a file with JVM memory options to use for the new PingAccess instance during the upgrade.
-s | --silent	Run the upgrade with no user input required. To use this option, specify the source version's credentials using environment variables.

Environment Variables

You can specify the username and password for the source version using these environment variables:

Environment variable	**Description**
`PA_SOURCE_API_USERNAME`	The username for the source version's Admin API. This should be set to Administrator.
`PA_SOURCE_API_PASSWORD`	The basic authorization password for the Administrator in the source version's Admin API.

JVM Memory options

These options can be included in the JVM memory options file. Memory amounts use `m` or `g` to specify the unit.

Memory option	**Description**
`-Xms`*`<amount>`*	Minimum heap size.
`-Xmx`*`<amount>`*	Maximum heap size.
`-XX:NewSize=`*`<amount>`*	Minimum size for the Young Gen space.
`-XX:MaxNewSize=`*`<amount>`*	Maximum size for the Young Gen space.
`-XX:+UseParallelGC`	Specifies that the parallel garbage collector should be used.

For example:

```
#Sample JVM Memory options file
-Xms512m
-Xmx1g
-XX:NewSize=256m
-XX:MaxNewSize=512m
-XX:+UseParallelGC
```

You can copy the existing *`PA_HOME`*`/conf/jvm-memory.options` file to create a JVM memory options file for the upgrade.

5. Stop the existing PingAccess admin instance.
6. Start the new PingAccess admin instance.

Next steps

Important:

If PingAccess is running as a service, and you upgraded using the upgrade utility:

- In Linux, update **PA_HOME** in `/etc/systemd/system/pingaccess.service` to point to the new installation.
- In Windows, remove the existing PingAccess service (`<OLD_PA_HOME>\sbin\Windows\uninstall-service.bat`) and add the new service (`<NEW_PA_HOME>\sbin\Windows\install-service.bat`).

After you have upgraded the administrative node, you can *upgrade the replica admin node*.

Upgrading the replica administrative node

Upgrade the PingAccess replica administrative node using the PingAccess Upgrade Utility, then resume configuration replication.

About this task

Any warnings or errors encountered are recorded in `log/upgrade.log`, as well as on the screen while the utility is running. The upgrade uses an exit code of `0` to indicate a successful upgrade and an exit code of `1` to indicate failure.

Info:

During the upgrade, it is important to not make any changes to the running PingAccess environment.

Steps

1. If you are using the upgrade utility, change to the new version's `/upgrade/bin` directory on the command line.

   Example:

   ```
   cd /pingaccess-6.1.0/upgrade/bin
   ```

2. Upgrade the system:
   Choose from:

   - If you are using the upgrade utility on a Windows system, use this command: upgrade.bat [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*

     For example.

     ```
     upgrade.bat ../pingaccess-5.3.0
     ```
   - If you are using the upgrade utility on a Linux system, use this command: ./upgrade.sh [-p *<admin_port>*] [-i *<directory>*] [-j *<jvm_memory_options_file>*] [-l *<newPingAccessLicense>*] [-s | --silent] *<sourcePingAccessRootDir>*

     For example.

     ```
     ./upgrade.sh ../pingaccess-5.3.0
     ```
   - If you are using the incremental update package, open the ReadMeFirst.txt file and make the file changes specified in the readme.

   The command-line parameters are the same regardless of the platform, and are defined as follows.

   Parameter definitions

Parameter	**Value description**
-p <*admin_port*>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
-i <*directory*>	An optional directory containing additional library JAR files (for example, plugins, JDBC drivers) to be copied into the target installation. Beginning in version 6.0, JAR files are stored in the *<PA_HOME>*/deploy folder. During an upgrade from versions earlier than 6.0, third-party JAR files are migrated from the lib folder to the deploy folder if no directory is specified. During an upgrade from version 6.0 or later, the contents of the deploy folder are migrated to the new *<PA_HOME>*/deploy folder if no directory is specified.
<*sourcePingAccessRootDir*>	The *PA_HOME* for the source PingAccess version.
-l <*newPingAccessLicense*>	An optional path to the PingAccess license file to use for the target version. If not specified, the existing license is reused.
-j <*jvm_memory_options_file*>	An optional path to a file with JVM memory options to use for the new PingAccess instance during the upgrade.

Parameter	Value description
-s | --silent	Run the upgrade with no user input required. To use this option, specify the source version's credentials using environment variables.

Environment Variables

You can specify the username and password for the source version using these environment variables:

Environment variable	Description
`PA_SOURCE_API_USERNAME`	The username for the source version's Admin API. This should be set to Administrator.
`PA_SOURCE_API_PASSWORD`	The basic authorization password for the Administrator in the source version's Admin API.

JVM Memory options

These options can be included in the JVM memory options file. Memory amounts use `m` or `g` to specify the unit.

Memory option	Description
`-Xms`*<amount>*	Minimum heap size.
`-Xmx`*<amount>*	Maximum heap size.
`-XX:NewSize=`*<amount>*	Minimum size for the Young Gen space.
`-XX:MaxNewSize=`*<amount>*	Maximum size for the Young Gen space.
`-XX:+UseParallelGC`	Specifies that the parallel garbage collector should be used.

For example.

```
#Sample JVM Memory options file
-Xms512m
-Xmx1g
-XX:NewSize=256m
-XX:MaxNewSize=512m
-XX:+UseParallelGC
```

You can copy the existing *<PA_HOME>*`/conf/jvm-memory.options` file to create a JVM memory options file for the upgrade.

3. Stop the existing PingAccess replica admin instance.
4. Start the new PingAccess replica admin instance.

Resume configuration replication for the replica administrative node:

5. In a browser, go to https://*<PingAccessHost>*:9000/pa-admin-api/v3/api-docs/.
6. Expand the **/adminConfig/replicaAdmins** endpoint.
7. Click the **GET /adminConfig/replicaAdmins** operation.
8. Click **Try it out!** and note the `id` for the replica admin.
9. Click the **GET /adminConfig/replicaAdmins/{id}** operation.
10. Enter the id of the replica admin you want to update and click **Try it out!**
11. Copy the Response Body.

12. Click the **PUT /adminConfig/replicaAdmins/{id}** operation and enter the id of the replica admin you want to update.
13. Paste the Response Body you copied and change `"configReplicationEnabled"` to `true`.
14. Click **Try it out!**

    Result:

    If the operation is successful, you will receive a response code of `200`.
15. Click **Settings** and then go to **Clustering** # **Administrative Nodes**.
16. Ensure the Replica Administrative Node displayed and reporting on the **Administrative Nodes** tab. A healthy node shows a green status indicator.

Next steps
After you have upgraded the administrative and replica administrative nodes, you can begin *upgrading the engines*.

Upgrading engines

This phase of the zero downtime upgrade focuses on upgrading each engine in the cluster. To maintain resource availability, perform this set of steps on one engine at a time until all engines are successfully upgraded.

Important:

Engines are identified by the engine name. Ensure that the engine that you remove from the load balancer aligns with the engine definition you import.

This phase requires that the following steps take place for each engine in the cluster, one at a time:

- *Remove the engine from the load balancer*
- *Upgrade the engine*
- *Resuming configuration replication* on page 105
- *Add the engine to the load balancer*

Important:

Do not begin the upgrade of an additional engine until the active engine upgrade is completed and the engine is reporting to the PingAccess administrative node.

Removing the engine from the load balancer configuration

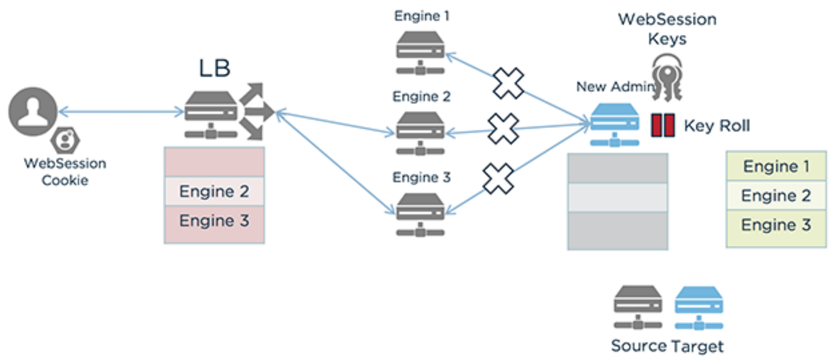
Remove the engine from the load balancer configuration. Because this step is dependent on your environment, no specific instruction will be provided.

Before you begin

You must be familiar with the steps required to temporarily remove the engine from your load balancer configuration.

Important:

To maintain resource availability, you should remove only the engine you are upgrading. After the upgrade is complete, add the engine back to the load balancer configuration. Only after you confirm that the engine has been successfully added to the load balancer and is reporting properly to PingAccess should you begin the upgrade process on additional engines.



Steps

1. Identify and note the engine you want to upgrade. Ensure you have the engine definition for this engine available.
2. Remove the engine from the load balancer.

> **Note:**
>
> Keep a record of the changes you make so that you can reverse this operation later in *Adding the engine to the load balancer configuration* on page 106.

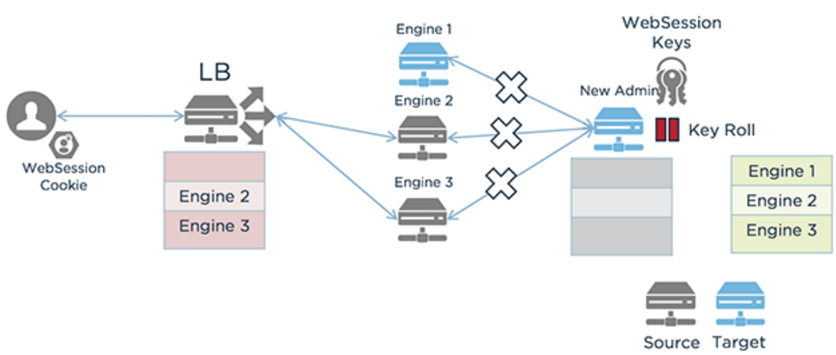
3. Restart the load balancer.

Upgrading the engine

Use the PingAccess Upgrade Utility to upgrade the engine.

Before you begin

For more information on upgrading PingAccess, see *Upgrade PingAccess*.



Before beginning the upgrade process, make sure you have:

- Ensured the PingAccess engine is running
- Downloaded the PingAccess *distribution `.zip` file* or the incremental update bundle and extracted it.
- The PingAccess license

About this task

Any warnings or errors encountered are recorded in `log/upgrade.log`, as well as on the screen while the utility is running. The upgrade uses an exit code of `0` to indicate a successful upgrade and an exit code of `1` to indicate failure.

Steps

1. If you are using the upgrade utility, change to the new version's `/upgrade/bin` directory on the command line.

   Example:

   ```
   cd /pingaccess-6.1.0/upgrade/bin
   ```

2. Upgrade the system:
   Choose from:

   - If you are using the upgrade utility on a Windows system, use this command: `upgrade.bat [-p <admin_port>] [-i <directory>] [-j <jvm_memory_options_file>] [-l <newPingAccessLicense>] [-s | --silent] <sourcePingAccessRootDir>`.

     For example.

     ```
     upgrade.bat ../pingaccess-5.3.0
     ```

   - If you are using the upgrade utility on a Linux system, use this command: `./upgrade.sh [-p <admin_port>] [-i <directory>] [-j <jvm_memory_options_file>] [-l <newPingAccessLicense>] [-s | --silent] <sourcePingAccessRootDir>`.

     For example.

     ```
     ./upgrade.sh ../pingaccess-5.3.0
     ```

   - If you are using the incremental update package, open the `ReadMeFirst.txt` file and make the file changes specified in the readme.

   The command-line parameters are the same regardless of the platform, and are defined as follows.

   Parameter definitions

Parameter	**Value description**
-p <*admin_port*>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is `9001`.

Parameter	Value description
-i <*directory*>	An optional directory containing additional library JAR files (for example, plugins, JDBC drivers) to be copied into the target installation. Beginning in version 6.0, JAR files are stored in the `<PA_HOME>/deploy` folder. During an upgrade from versions earlier than 6.0, third-party JAR files are migrated from the `lib` folder to the `deploy` folder if no directory is specified. During an upgrade from version 6.0 or later, the contents of the `deploy` folder are migrated to the new `<PA_HOME>/deploy` folder if no directory is specified.
<*sourcePingAccessRootDir*>	The PA_HOME for the source PingAccess version.
-l <*newPingAccessLicense*>	An optional path to the PingAccess license file to use for the target version. If not specified, the existing license is reused.
-j <*jvm_memory_options_file*>	An optional path to a file with JVM memory options to use for the new PingAccess instance during the upgrade.
-s | --silent	Run the upgrade with no user input required. To use this option, specify the source version's credentials using environment variables.

Environment Variables

You can specify the username and password for the source version using these environment variables:

Environment variable	Description
`PA_SOURCE_API_USERNAME`	The username for the source version's Admin API. This should be set to Administrator.
`PA_SOURCE_API_PASSWORD`	The basic authorization password for the Administrator in the source version's Admin API.

JVM Memory options

These options can be included in the JVM memory options file. Memory amounts use `m` or `g` to specify the unit.

Memory option	Description
`-Xms<amount>`	Minimum heap size.
`-Xmx<amount>`	Maximum heap size.
`-XX:NewSize=<amount>`	Minimum size for the Young Gen space.
`-XX:MaxNewSize=<amount>`	Maximum size for the Young Gen space.

Memory option	Description
`-XX:+UseParallelGC`	Specifies that the parallel garbage collector should be used.

Example:

```
#Sample JVM Memory options file
-Xms512m
-Xmx1g
-XX:NewSize=256m
-XX:MaxNewSize=512m
-XX:+UseParallelGC
```

You can copy the existing *PA_HOME*/conf/jvm-memory.options file to create a JVM memory options file for the upgrade.

3. Stop the existing PingAccess instance. Do not start the new instance.

Next steps

Important:

If PingAccess is running as a service and you upgraded using the upgrade utility:

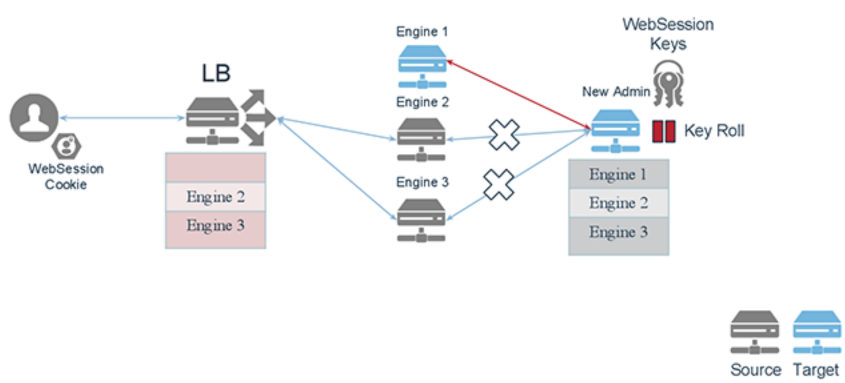
- In Linux, update `PA_HOME` in `/etc/systemd/system/pingaccess.service` to point to the new installation.
- In Windows, remove the existing PingAccess service (*<OLD_PA_HOME>*\sbin\Windows\uninstall-service.bat) and add the new service (*<NEW_PA_HOME>*\sbin\Windows\install-service.bat).

Resuming configuration replication

Resume the configuration replication that was disabled by the Upgrade Utility. Perform this step for all engine nodes in the cluster.

About this task

You will use the PingAccess Admin API to GET and PUT the relevant configuration data for each of these items.



Note:

Perform the following steps for each engine in the cluster.

To resume configuration replication:

Steps

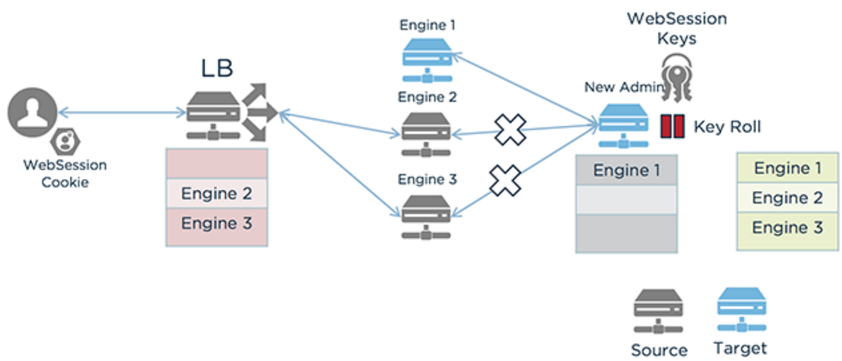
1. In a browser, go to https://<*PingAccessHost*>:9000/pa-admin-api/v3/api-docs/.
2. For engines, expand the **/engines** endpoint.
3. Click the **GET /engines** operation.
4. Click **Try it out!** and note the engine id for each engine.
5. Click the **GET /engines/{id}** operation.
6. In the **ID** field, enter the id of the engine you want to update and click **Try it out!**
7. Copy the entire Response Body.
8. Click the **PUT /engines/{id}** operation and enter the id of the engine you want to update.
9. In the **Engine** field, paste the response body you copied and change `"configReplicationEnabled"` to `true`.
10. Click **Try it out!**

    Result:

    If the operation is successful, you will receive a response code of `200`.
11. Start the node.
12. Repeat the previous steps for each node.
13. Click **Settings** and then go to **Clustering** # **Engines**.
14. Ensure the engines are displayed and reporting. A healthy engine shows a green status indicator.

Note:

There might be a delay in bringing the engine to a running status. If the engine does not immediately show as reporting, refresh the page until the engine status indicator is green (running).



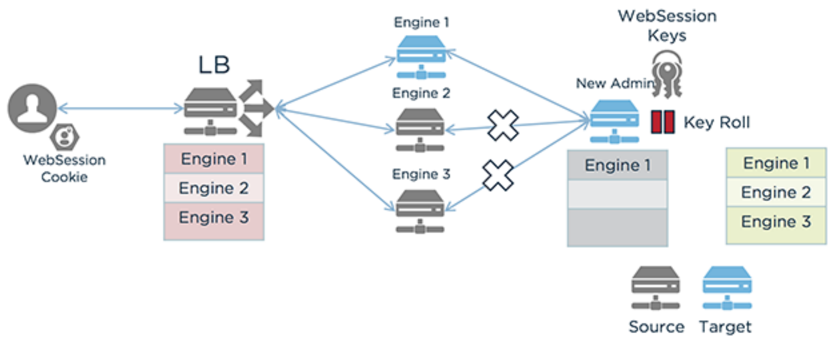
Adding the engine to the load balancer configuration

Add the engine back to the load balancer configuration. Since this step is dependent on your environment, no specific instruction will be provided.

Before you begin

You must be familiar with the steps required to add the engine back to the load balancer configuration.

After you confirm that the engine has been successfully added to the load balancer and is reporting properly to PingAccess, you can begin the upgrade process on additional engines.



Steps

1. To add the engine to the load balancer configuration, reverse the steps you took in *Removing the engine from the load balancer configuration* on page 101 to remove the engine.
2. Restart the load balancer.

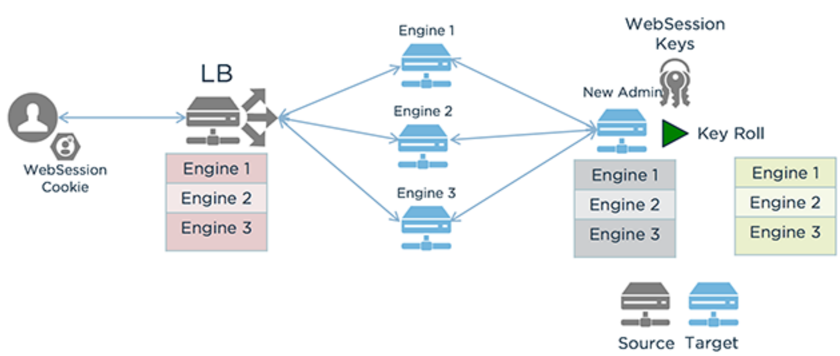
Next steps

Repeat the *Upgrading engines* on page 101 process until each engine has been upgraded. When all engines have been upgraded, added to the load balancer configuration, and are reporting to PingAccess, you can move on to the final step, *Enable key rolling*, to complete the zero downtime upgrade process.

Enabling key rolling

Resume key rolling.

About this task



Steps

1. Click **Access** and then go to **Identity Mappings** # **Auth Token Management**.
2. In the **Auth Token Management** section, select the **Key Roll Enabled** check box.
3. Verify that the **Key Roll Interval (H)** is correct, then click **Save**.
4. Click **Access** and then go to **Web Sessions** # **Web Session Management**.
5. In the **Web Session Management** section, select the **Key Roll Enabled** check box.
6. Verify that the **Key Roll Interval (H)** is correct, then click **Save**.
7. Click **Access** and then go to **Token Validation** # **OAuth Key Management**.
8. In the **OAuth Key Management** section, select the **Key Roll Enabled** check box.

9. Verify that the **Key Roll Interval (H)** is correct, then click **Save**.

Recovering from a failed upgrade

You can recover your PingAccess cluster by switching back to the source version if the upgrade fails.

About this task

The zero downtime upgrade process creates a set of new folders for the upgraded installation. The pre-upgrade source installation is not affected.

To recover your PingAccess cluster in the event of a failure, you would resume the former installation using these steps.

Steps

1. Stop any upgraded PingAccess instances.
2. Start the original PingAccess instance on the admin node.
3. Import the engine definitions back into the original PingAccess instance.
4. Start the original PingAccess instances on the engine nodes.
5. Ensure all engines are added to the load balancer configuration.

Configuring and Customizing PingAccess

This section contains information on how to configure and customize your PingAccess environment.

- For information on configuring PingAccess for server-side session management, see *Session management configuration* on page 108.
- For information on the types PingAccess logs and how to configure them, see *Logging configuration* on page 111.
- For information on customizing PingAccess templates or localizing user-facing documentation and system status messages, see *Customize and Localize PingAccess*.
- For information on enabling or disabling Federal Information Processing Standards (FIPS) mode, see *Managing Federal Information Processing Standards (FIPS) mode* on page 149.
- If you're using Amazon Web Services (AWS), see *Configuring PingAccess to use Amazon Key Management Services*.

Session management configuration

You can configure PingAccess for server-side session management using PingFederate through web session settings.

Web Sessions

Web sessions define the policy for web application session creation, lifetime, timeouts, and their scope. You can configure multiple web sessions to scope the session to meet the needs of a target set of applications. This improves the security model of the session by preventing unrelated applications from impersonating the end user. Use the following tasks to configure secure web sessions for use with specific applications and to configure global web session settings.

Application scoped Web Sessions

Several controls exist to scope the PingAccess (PA) token to an application:

Audience Attribute

The audience attribute defines who the token is applicable to and is represented as a short, unique identifier. Requests are rejected that contain a PA token with an audience that differs from what is configured in the web session associated with the target resource.

Audience Suffix

The audience attribute is also used as a suffix of the cookie name to ensure uniqueness. For example, PA.businessAppAudience.

Cookie Domain

The cookie domain can also optionally be set to limit where the PingAccess token is sent.

> **Info:**
>
> In addition to these controls, you can adjust parameters, such as session timeout, to match the policy requirements of each application.

You must define corresponding OAuth clients in PingFederate for each web session. Redirect URL whitelists defined in PingFederate dictate from which servers and domains the session can originate. Controlling this within PingFederate enables flexibility of the attribute contract, and its fulfillment, for that particular application. This ensures that each application and its associated policies only deal with attributes related to it.

Server-side session management configuration

You can implement server-side session management in one of two ways.

- PingAccess can reject a PingAccess cookie associated with a PingFederate session that has been invalidated as a result of an end-user driven sign-off.
- The end user can initiate a sign-off from all PingAccess issued web sessions using a centralized sign-off.

The first of these scenarios provides increased scalability and security, ensuring the PingFederate session is terminated and that subsequent session validation requests are rejected. This scenario implies a user sign-off from PingAccess protected resources through the invalidation of the related PingFederate session.

The second scenario provides improved performance and end user experience. When the user explicitly signs off of the PingAccess issued session, all related PingAccess cookies are deleted, ensuring the client is no longer authenticated to resources protected by PingAccess. In this scenario, the user has explicitly signed off from all of those protected services.

You must configure PingAccess only for the first scenario. These options are not mutually exclusive and can be combined to provide comprehensive session management at the server.

For more information, see the following topics:

- *Configuring PingFederate for session management* on page 109
- *Configuring PingFederate for user-initiated single logout* on page 110
- *Configuring PingAccess for server-side session management* on page 111

Configuring PingFederate for session management

Configure PingFederate to revoke PingAccess session cookies.

Steps

1. Sign on to the PingFederate Administrative Console

2. If you are using PingFederate 10.0 or earlier, go to **Server Configuration** # **Server** # **Protocol settings** # **Roles & Protocols** and ensure that **Enable OAuth 2.0 Authorization Server (AS) role** and **OpenID Connect** are enabled.
3. Go to **System** # **OAuth Settings** # **Authorization Server Settings** and configure the authorization server settings.
4. Go to the client management section.
   Choose from:
   - If you are using PingFederate 10.0 or earlier, go to **System** # **OAuth Settings** # **Client Management**.
   - If you are using PingFederate 10.1 or later, go to **Applications** # **OAuth** # **Clients**.
5. Create or modify an existing client.
6. Ensure that **Client Secret** is enabled, and then enter a client secret to be used by PingAccess for authentication.
7. Grant access to the Session Revocation API.
   Choose from:
   - If you are using PingFederate 10.0 or earlier, in the **OpenID Connect** section of the client's configuration page, enable **Grant Access to Session Revocation API**.
   - If you are using PingFederate 10.1 or later, beside **Session API Endpoints**, select **Allow Access to Session Revocation API**.

   > **Note:**
   >
   > This setting is the main setting that enables the server-side session management feature in PingFederate.
8. Click **Save** to save your changes.

Configuring PingFederate for user-initiated single logout

Configure PingFederate to provide PingAccess with access to the PingFederate-managed session.

Steps

1. Sign on to the PingFederate administrative console.
2. Go to **System** # **OAuth Settings** # **Authorization Server Settings**.
3. Select **Track User Sessions for Logout**.
4. Click **Save**.
5. Select an OpenID Connect policy.
   Choose from:
   - If you are using PingFederate 10.0 or earlier, go to **System** # **OAuth Settings** # **OpenID Connect Policy Management** and click an existing policy.
   - If you are using PingFederate 10.1 or later, go to **Applications** # **OAuth** # **OpenID Connect Policy Management** and click an existing policy.
6. On the **Manage Policy** tab, select **Include Session Identifier in ID Token**.

   For more information about configuring an OpenID Connect Policy, see *Configuring OpenID Connect Policies* in the PingFederate Administrator's Manual.
7. Click **Save**.

8. Select the client to be used by PingAccess.
   Choose from:
   - If you are using PingFederate 10.0 or earlier, go to **System** # **OAuth Settings** # **Client Management** and select the client to be used by PingAccess.
   - If you are using PingFederate 10.1 or later, go to **Applications** # **OAuth** # **Clients** and select the client to be used by PingAccess.
9. In the **OpenID Connect** section of the client's configuration page, select **PingAccess Logout Capable**.

   > **Tip:**
   >
   > If this option is not available, ensure that the **Track User Sessions for Logout** setting change made in step 3 was saved.
10. Click **Save**.

Configuring PingAccess for server-side session management

Configure PingAccess to enable server-side session management.

Steps

1. Sign on to the PingAccess administrative console.
2. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
3. Click either *Create a new web session* or *Edit an existing web session*.
4. Enter a unique **Name** for the web session, up to 64 characters, including special characters and spaces.
5. Specify the **Audience** that the PingAccess token is applicable to, represented as a short, unique identifier between 1 and 32 characters.

   Requests are rejected that contain a PingAccess token with an audience that differs from what is configured in the web session associated with the target application. Changing this setting might affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
6. In the **Client ID** field, enter the Client ID defined in PingFederate.
7. In the **Client Credentials Type** section, select **Secret**, and then enter the **Client Secret** associated with the specified Client ID.
8. Click **Show Advanced**.
9. To enable the server-side session management feature, select **Validate Session**.
10. Click **Save**.

Logging configuration

This document describes the types of logging performed by PingAccess and provides instructions for configuring PingAccess logging.

Security audit logging

The PingAccess audit logs record a selected subset of transaction log information at runtime plus additional details, intended to facilitate security auditing and regulatory compliance.

The logs are located in *`PA_HOME`*`/log/`. Elements recorded in these logs are described in the table below, and are configured in `conf/log4j2.xml`.

> **Important:**

Because log files can be viewed or modified using a variety of common applications, it is possible for log files to be manipulated to include untrusted or malicious data. Administrators should take appropriate steps to secure these files. Do not open these files in applications that could allow for data execution, such as internet browsers or Microsoft Office products. Instead, open these files in a common, lightweight text editor.

PingAccess generates these audit logs:

`pingaccess_engine_audit.log`

Records transactions of configured resources. Additionally, the log records transaction details when PingAccess sends requests to PingFederate, for example, security token service (STS), OAuth2, and JSON web signature (JWS).

`pingaccess_api_audit.log`

Records PingAccess administrative API transactions. These transactions represent activity in the PingAccess administrative console. This log also records transaction activity if you are using scripts to configure PingAccess.

`pingaccess_agent_audit.log`

Records transactions between PingAccess Agents and the PingAccess Engine.

`pingaccess_sideband_client_audit.log`

Records transactions sent to and from the sideband client integration.

pingaccess_sideband_audit.log

Records the end-user transaction captured by the sideband client request.

Audit log configuration

Item	Description
`%d`	Transaction time.
`exchangeId`	Identifies the ID for a specific request/response pair.
`AUDIT.applicationID`	Specifies the ID of the requested application.
`AUDIT.applicationName`	Specifies the name of the requested application.
`AUDIT.resourceID`	Specifies the ID of the requested resource.
`AUDIT.resourceName`	Specifies the name of the requested resource.
`AUDIT.pathPrefix`	Specifies the path prefix of the requested application or resource.
`AUDIT.pathPrefixType`	Indicates the pattern type of the path prefix, `Wildcard` or `Regex`.
`AUDIT.authMech`	Mechanism used for authentication. Engine Auditing - Cookie (WAM session), OAuth, unknown (for example, pass-through or static assets). Pass-through assets are resources with no policies or web session configured. Admin Auditing - Basic, OAuth, Cookie, unknown (unknown displays only in an authentication failure).
`AUDIT.client`	IP address of the requesting client.

Item	Description
`AUDIT.failedRuleName`	Name of the rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the pingaccess_engine_audit.log.
`AUDIT.failedRuleType`	Type of rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the pingaccess_engine_audit.log.
`AUDIT.failedRuleClass`	The Java class of rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the pingaccess_engine_audit.log.
`AUDIT.failedRuleSetName`	Name of the containing rule set that failed. If no rule failure occurred, this field is blank. This element is applicable only to the pingaccess_engine_audit.log.
`AUDIT.host`	PingAccess host name or IP address.
`AUDIT.targetHost`	Backend target that processed the request and generated a response to the PingAccess engine. This variable is unset when the response is generated by PingAccess directly.
`AUDIT.method`	HTTP method of the request. For example, GET.
`AUDIT.resource`	Name of the resource used to fulfill the request. This element is applicable only to the pingaccess_engine_audit.log.
`AUDIT.responseCode`	HTTP status code of the response. For example, 200.
`AUDIT.requestUri`	Request URI portion of the request (for example, /foo/bar).
`AUDIT.subject`	Subject of the transaction.
`AUDIT.trackingId`	The PingFederate tracking ID. This element can be used to help correlate audit information in the PingAccess audit log with information recorded in the PingFederate audit log. The value of this depends on whether the application type is `Web` or `API`. If the application type is `Web`, the value is presented as `tid:`*`<Session_Identifier>`*. The <*Session_Identifier*> can be used by the *PingFederate Session Revocation API* to revoke the session without disabling the user in the identity store. If the application type is `API`, the value is presented as `atid:`*`<Hash>`*. The <*Hash*> value is derived from the OAuth Access token for the session, and only serves as an identifier; it cannot be used for session revocation.
`AUDIT.reqReceivedMillisec`	Time in milliseconds since 1970 that a client request was first received

Item	Description
`AUDIT.reqSentMillisec`	Time in milliseconds since 1970 that the agent or engine sent a backchannel or proxy request
`AUDIT.respReceivedMillisec`	Time in milliseconds since 1970 that the agent or engine received a response from a backchannel call or proxy request
`AUDIT.respSentMillisec`	Time in milliseconds since 1970 that a response was sent back to the client
`AUDIT.roundTripMS`	The respSentMillisec time minus the reqReceivedMillisec time. This represents the total number of milliseconds it took PingAccess to respond to a client's request including the proxyRoundTripMS.
`AUDIT.proxyRoundTripMS`	The respReceivedMillisec time minus the reqSentMillisec time. This represents the total number of milliseconds PingAccess was waiting for another entity to respond to a backchannel call or proxy request.
`AUDIT.siteUnavailableInfo`	If a site is unavailable, this is reason why the last attempted site target is unavailable.
`AUDIT.agentName`	The name of the agent.
`AUDIT.responder`	The component that generated the response. Valid values are `PingAccess`, `Site`, `Third Party Service`, `OpenID Provider`, and `Authorization Server`.
`AUDIT.clientCertSerialNum`	The serial number of the client certificate.
`AUDIT.clientCertSubjectDn`	The subject of the client certificate as an X.500 domain name.
`AUDIT.clientCertIssuerDn`	The issuer of the client certificate as an X.500 domain name.
`AUDIT.sidebandName`	The name of the requesting sideband client.
`AUDIT.sidebandDecision`	The policy decision returned in response to the sideband client request. Valid values are 'accept' and 'reject'.
`agent{a-header-value-key}`	The `vnd-pi-agent` header value for a given key. Represents the header value that an agent sends to PingAccess. Well-known keys are: ▪ h – The hostname of the server where the agent resides. ▪ t – The type of agent and/or the type of platform where the agent resides. ▪ v – The version of the agent making the request. This information is not sent by default. See *Agent inventory logging* on page 117 for more information about logging this information.

Item	Description
`appRequestHeader{a-header-name}`	HTTP request header value for the given HTTP request header name. Represents the header value that PingAccess sends to the back end site.
`appResponseHeader{a-header-name}`	HTTP response header value for the given HTTP request header name. Represents the header value received from the application.
`clientRequestHeader{a-header-name}`	HTTP request header value for the given HTTP request header name. Represents the header value received from the client.
`clientResponseHeader{a-header-name}`	HTTP response header value for the given HTTP request header name. Represents the header value returned to the client.

Note:
To get information about the timing for back channel calls, such as the OIDC UserInfo endpoint call, use the `exchangeID` property to match related log entries and the `AUDIT.roundTripMS` and `AUDIT.proxyroundTripMS` properties to view the timing.

Logging

PingAccess logging is handled by the log4j2 asynchronous logging library, configured using `conf/log4j2.xml`.

Info:

Audit logs are also configurable in `conf/log4j2.xml`. These logs record a selected subset of transaction log information at runtime plus additional details. For more information, see *Security Audit Logging*.

By default, logging information is output to *`PA_HOME`*`/logs/pingaccess.log`, and file logging uses the rolling file appender. PingAccess keeps a maximum of 10 log files, each with a maximum size of 100 MB. Once 10 files accumulate, PingAccess deletes the oldest. You can change these defaults by locating and modifying the following properties in the `<Appenders>` section of `log4j2.xml`:

- Changing the log file name

```
<RollingFile name="File"
            fileName="${sys:pa.home}/log/pingaccess.log"
            filePattern="${sys:pa.home}/log/pingaccess.log.%i"
            ignoreExceptions="false">
```

- Setting the maximum log size

```
<SizeBasedTriggeringPolicy size="100000 KB"/>
```

- Setting the maximum number of log files

```
<DefaultRolloverStrategy max="10"/>
```

In addition to the standard log4j2 items, PingAccess adds the following custom item that can be used in the `log4j2.xml<PatternLayout>` configuration.

Item	Description
`exchangeId`	Identifies the ID for a specific request/response pair.

For example, the following line from `log4j2.xml` incorporates the `exchangeId` in the output.

```
<pattern>%d{ISO8601} %5p [%X{exchangeId}] %c:%L - %m%n</pattern>
```

Note:

The `%X` conversion character is required for the `exchangeId` to be displayed properly.

Configuring log levels

Define log levels for specific package or class names to get more or less logging from a class or group of classes.

About this task

If the log level is not specified for a particular package or class, the settings for the root logger are inherited.

Steps

1. Open `conf/log4j2.xml` in an editor.
2. Locate the `<AsyncLogger>` element for the package or class you want to adjust the logging level for.

   Example:

   ```
   <AsyncLogger name="com.pingidentity" level="DEBUG" additivity="false"
    includeLocation="false">
   ```
3. Modify the `level` attribute to set the desired log level.

   Valid values are `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`.
4. Save the modified file.

Result

PingAccess will automatically make the changes effective within 30 seconds.

Configuring class or package log levels

Use the `log4j2.xml` file to configure the log level for a class or package.

Steps

1. Open `conf/log4j2.xml` in an editor.

   Class or package loggers are defined in the `<AsyncLogger>name` attribute. For example, cookie logging is enabled using the following line.

   Example:

   ```
   <AsyncLogger
    name="com.pingidentity.pa.core.interceptor.CookieLoggingInterceptor"
    level="TRACE" additivity="false" includeLocation="false">
       <AppenderRef ref="File"/>
   </AsyncLogger>
   ```

2. Set the `level` value in the `<AsyncLogger>` element to one of the following values:

   `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`.

   Example: To apply `TRACE` level logging for the `com.pingidentity` package, locate the following line,

   ```
   <AsyncLogger name="com.pingidentity" level="DEBUG" additivity="false"
    includeLocation="false">
   ```

   and change it to

   ```
   <AsyncLogger name="com.pingidentity" level="TRACE" additivity="false"
    includeLocation="false">
   ```
3. Save the file.

Enabling cookie logging

Enable cookie logging, which is an optional feature in the TRACE log level.

Steps

1. Edit `conf/log4j2.xml` and uncomment the following section.

   ```
   <AsyncLogger
    name="com.pingidentity.pa.core.interceptor.CookieLoggingInterceptor"
    level="TRACE" additivity="false" includeLocation="false">
        <AppenderRef ref="File"/>
   </AsyncLogger>
   ```
2. Save the file.

Garbage collection logging

PingAccess logs Java garbage collection data by default.

The garbage collection log includes details related to each occurrence of garbage collection, such as a timestamp and the change in heap memory.

Edit the following properties in the `PA_HOME/bin/run.sh` file on Linux systems or the `PA_HOME\bin\run.bat` file on Windows systems to configure garbage collection properties.

Property	Description
`GC_FILE=`*"<filename>"*	Specifies the location of the garbage collection log. Comment out this line to disable garbage collection logging.
`GC_FILE_COUNT=`*"<count>"*	Specifies the number of garbage collection files to retain before rotating.
`GC_FILE_SIZE=`*"<size>"*	Specifies the maximum size for garbage collection files.

Agent inventory logging

To log details about your PingAccess agents, you can add custom configuration to the agents and the PingAccess system.

Agent information other than the agent name is not included in agent responses by default. You can customize agents to include the agent header, providing additional information that can be included in logs.

You must edit the `/conf/log4j2.xml` file to log the information included in the agent header. See *Security audit logging* on page 111 for more information.

For more information about agent headers, see *https://docs.pingidentity.com/csh?Product=pa-latest&topicname=sjw1565633628332.html* .

Agent Header

The optional `vnd-pi-agent` header allows the agent to communicate information about itself and its deployment environment to PingAccess. The value of this header is a map of comma-separated key-value pairs.

The agent can specify the custom keys specific to the deployment of the agent or utilize one or more of the well-known keys:

v

The version of the agent making the request.

t

The type of agent and/or the type of platform where the agent resides.

h

The hostname of the server where the agent resides.

The syntax for the `vnd-pi-agent` value conforms to a dictionary in this specification, *https://httpwg.org/http-extensions/draft-ietf-httpbis-header-structure.html#dictionary*, where member-values are constrained to be an `sh-string` item.

These header examples are considered semantically equivalent.

Example

```
vnd-pi-agent: v="1.0.0", h="apache.example.com", t="Apache 2.4.41"
```

Example

```
vnd-pi-agent: v="1.0.0", h="apache.example.com"
vnd-pi-agent: t="Apache 2.4.41"
```

Example

```
vnd-pi-agent: v="1.0.0"
vnd-pi-agent: h="apache.example.com"
vnd-pi-agent: t="Apache 2.4.41"
```

Appending log messages to syslog and the console

Enable additional output destinations, called appenders.

About this task

Console and syslog appenders are pre-configured in `log4j2.xml`, but are disabled by default.

Perform the following steps to enable additional appenders.

Steps

1. Open `conf/log4j2.xml` in an editor.

2. Locate the following lines in the `<Loggers>` element.

```
<AsyncLogger name="com.pingidentity" level="DEBUG" additivity="false"
 includeLocation="false">
    <AppenderRef ref="File"/>
    <!--<AppenderRef ref="CONSOLE" />-->
    <!--<AppenderRef ref="SYSLOG" />-->
</AsyncLogger>
```

> **Note:**
>
> If you have customized logging to enable logging for additional classes, locate the `<AsyncLogger>` element that is relevant to the class in question. This class is defined in the `<AsyncLogger>name` attribute.

3. Uncomment the `<AppenderRef>` element that applies to the appender you want to enable.

> **Note:**
>
> PingAccess will rescan the logging configuration within 30 seconds and make the change active automatically.

4. Save the file.

Log traffic for troubleshooting

Enable HTTP Archive (HAR) file audit logging to troubleshoot specific issues or gather more detailed information on the requests and responses sent to and from applications.

When you enable HAR file audit logging for a specific logging category, such as engine audit logging, PingAccess generates a HAR-formatted audit log file for that log category. For example, the HAR-formatted file for the engine audit logging category is the engine audit log HAR file or *<PA_HOME>*`/log/pingaccess_engine_audit_har.log`.

The HAR-formatted audit logs contain detailed records of specific transactions and sub-transactions between PingAccess and other systems, such as the configured OAuth authorization server (AS) or a system acting on behalf of the end user.

Audit log records are called transactions. One audit log record equals one transaction. A transaction captures an end-user transaction or an event originating from PingAccess. Sub-transactions are the request-response pairs sent through the HTTP or HTTPS connections.

You can use regex filters to include or exclude certain results from the logs.

> **Important:**
>
> HAR format within PingAccess is not exactly equivalent to a browser's HAR format. A standard HAR file is a complete Javascript Object Notation (JSON) object, whereas the PingAccess HAR-formatted audit log file is a JSON sequence of HAR objects. Each log entry is a complete HAR file, and entries are continually added to the audit log as new information comes in.
>
> Standard HAR tooling doesn’t understand what to do with a JSON sequence, so you must reformat a HAR-formatted audit log before you can view it or parse it with a HAR reader. For more information on how to convert a snapshot instance of the HAR-formatted audit log file into a complete JSON object for parsing, see *Parsing HAR-formatted audit log files* on page 135.

Structure of a HAR file log entry

Each HAR file entry within the HAR-formatted audit log file contains an array of audited PingAccess transactions, and each transaction contains one or more request-response pairs. Transactions can provide insight into PingAccess's internal processing.

If an incoming request to PingAccess requires it to make an outgoing request to a site application before it can provide a response, or if you're using an API gateway integration with PingAccess, then a transaction can contain up to two sub-transactions (request-response pairs):

- The first sub-transaction is the original request from the client or API gateway and the response from PingAccess, which it might have modified from the original backend response.
- The second sub-transaction is the request data sent from PingAccess, which PingAccess might have modified from the original request, to the backend site or API gateway and the original response from the site or API gateway.

For most other transactions, such as those seen in agent auditing or requests made from PingAccess to other external systems, there is usually only one sub-transaction.

Tip:

PingAccess records entries as request-response pairs in the audit log file, so sub-transactions aren't ordered sequentially. For example:

Sequentially, a user's inbound request prompts PingAccess to make an outbound request. The response to PingAccess's outbound request then determines PingAccess's final response to the user.

PingAccess's response to the user derives from the response to its own request, but the audit log records PingAccess's response before the response it receives. This is because the user's request and PingAccess's response to the user pair together as a sub-transaction. Likewise, PingAccess's request pairs with the response it receives.

You can use sub-transactions to:

- Check PingAccess’s connections to PingFederate.
- Follow OAuth and OpenID Connect (OIDC) flows to identify where issues might lie.
- Investigate PingAccess policy issues.
- Review the HTTP headers that PingAccess receives to investigate virtual host parsing and base configuration mapping.
- Review the HTTP headers that PingAccess sends.

Components of a sub-transaction

Metadata

The overview details of the transaction. For a list of specific metadata elements, see *Traffic logging reference* on page 127.

URL

The URL of the transaction. Path varies based on whether the transaction is an input to or output from PingAccess.

HTTP Method

The HTTP method used in the transaction.

HTTP Headers

The HTTP headers included in the transaction. For input to PingAccess, the `Host` and `X-Forwarded-For` headers are common inclusions.

Cookies

The cookies included in the transaction. These can be cookies from PingAccess and PingFederate. If the cookies aren't encrypted, you can view their contents.

> **Note:**
>
> The PingAccess cookie contains all of the request's identity mappings and the access token from PingFederate, if PingFederate is the AS. This is all of the information that PingAccess requires to make authorization and mapping decisions.

Responder

The component that generated the response. Valid values are `PingAccess`, `Site`, `Third Party Service`, `OpenID Provider`, and `Authorization Server`.

Timings

The time that it took to process specific components of the transaction. These components include `%d`, `AUDIT.reqReceivedMillisec`, `AUDIT.reqSentMillisec`, `AUDIT.respReceivedMillisec`, `AUDIT.respSentMillisec`, `AUDIT.roundTripMS`, and `AUDIT.proxyRoundTripMS`. For more information on these components, see *Security audit logging* on page 111.

Failed Policy Components

The rule or rule set name, type, and class of the policy component that failed in the transaction. Failed policy components are only available in the engine log.

PingAccess HAR file responses can contain binary and text from CSS, fonts, JavaScript, and the HTML from PingAccess errors. HAR file responses don't contain HTML responses from the back-end systems, but do indicate the length of the message’s HTML body.

> **Tip:**
>
> If you require specific HTML from a request or response, enable the `TRACE` log level and check the PingAccess server log to find this information.

Which log categories support HAR file audit logging?

API audit log

Requests made to the PingAccess APIs.

Engine audit log

Requests made to or by the PingAccess engine.

Agent audit log

Requests made to or by a PingAccess agent.

Sideband audit log

Requests made to the sideband client.

Sideband client audit log

Requests made to or by the sideband client integration.

For more information on these categories, see *Security audit logging* on page 111.

How do I enable HAR file audit logging?

When you enable HAR file audit logging for a log category, you can filter the types of entries to include in the log and specify what information you want to log. For more information, see the following topics:

- *Enabling API audit traffic logging* on page 122
- *Enabling engine traffic logging* on page 123
- *Enabling agent traffic logging* on page 124
- *Enabling sideband traffic logging* on page 125
- *Enabling sideband client traffic logging* on page 126

> **Important:**
>
> HAR-formatted audit log files are significantly larger than other log files and can include credentials. You should either carefully configure regex filters to exclude credential information or enable these logs only for troubleshooting purposes. Delete the files when they are no longer necessary.

Enabling API audit traffic logging

Enable API audit logging including request and responses.

Steps

1. Edit the `<PA_HOME>/conf/log4j2.xml` file.
2. In the Logger section, uncomment the `AppenderRef` element for the API audit log HAR file.

   Example:

```
        <!-- Audit Log Configuration-->
         <Logger name="apiaudit" level="INFO" additivity="false">
             <AppenderRef ref="APIAuditLog-File"/>
             <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
             <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-
Failover"/>-->
             <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
             <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
             <AppenderRef ref="ApiAuditLog-HarFile"/>
         </Logger>
```

3. In the Appenders section, uncomment the `RollingFile`

   Example:

```
    <Appenders>
        ...
        <RollingFile name="ApiAuditLog-HarFile"
                            fileName="${sys:pa.home}/log/
pingaccess_api_audit_har.log"
                            filePattern="${sys:pa.home}/log/
pingaccess_api_audit_har.%d{yyyy-MM-dd}.log"
                            ignoreExceptions="false">
            <StatusCodeRegExFilter regex=".*"/>
            <HarLogLayout>
                <KeyValuePair key="AUDIT.metadata" value="true"/>
                <KeyValuePair key="AUDIT.http-client" value="true"/>
            </HarLogLayout>
            <Policies>
                <TimeBasedTriggeringPolicy />
            </Policies>
        </RollingFile>
```

4. Optional: To filter the entries to add to the log file, edit the value in the `StatusCodeRegExFilter` element.
5. Optional: To specify what information to log, edd or edit the values in the `HarLogLayout` section of the `RollingFile` element.

   You can add or edit metadata and client response values. See *Traffic logging reference* on page 127 for more information.

Result

Logging begins when the configuration is reloaded. The configuration is reloaded at regular intervals according to the `monitorInterval` value.

Enabling engine traffic logging

Enable engine audit logging, including requests and responses.

Steps

1. Edit the *<PA_HOME>*`/conf/log4j2.xml` file.
2. In the Logger section, uncomment the `AppenderRef` element for the engine audit log HAR file.

   Example:

```
        <!-- Audit Log Configuration-->
         ...
         <Logger name="engineaudit" level="INFO" additivity="false">
             <AppenderRef ref="EngineAuditLog-File"/>
             <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
             <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-
Failover"/>-->
             <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
             <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
             <AppenderRef ref="EngineAuditLog-HarFile"/>
         </Logger>
```

3. In the Appenders section, uncomment the `RollingFile` element for the engine audit log HAR file.

   Example:

```
     <Appenders>
         ...
         <RollingFile name="EngineAuditLog-HarFile"
                             fileName="${sys:pa.home}/log/
pingaccess_engine_audit_har.log"
                             filePattern="${sys:pa.home}/log/
pingaccess_engine_audit_har.%d{yyyy-MM-dd}.log"
                             ignoreExceptions="false">
             <StatusCodeRegExFilter regex=".*"/>
             <HarLogLayout>
                 <KeyValuePair key="AUDIT.metadata" value="true"/>
                 <KeyValuePair key="AUDIT.http-client" value="true"/>
                 <KeyValuePair key="AUDIT.http-app" value="true"/>
             </HarLogLayout>
             <Policies>
                 <TimeBasedTriggeringPolicy />
             </Policies>
         </RollingFile>
```

4. Optional: To filter the entries to add to the log file, edit the value in the `StatusCodeRegExFilter` element.

5. Optional: To specify what information to log, add or edit the values in the `HarLogLayout` section of the `RollingFile` element.

   You can add or edit metadata, client response, and app response values. See *Traffic logging reference* on page 127 for more information.

Result
Logging begins when the configuration is reloaded. The configuration is reloaded at regular intervals according to the `monitorInterval` value.

Enabling agent traffic logging
Enable agent audit logging, including requests and responses.

Steps

1. Edit the `<PA_HOME>/conf/log4j2.xml` file.
2. In the Logger section, uncomment the `AppenderRef` element for the agent audit log HAR file.

   Example:

```
        <!-- Audit Log Configuration-->
         ...
         <Logger name="agentaudit" level="INFO" additivity="false">
             <AppenderRef ref="AgentAuditLog-File"/>
             <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
             <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-
Failover"/>-->
             <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
             <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
             <AppenderRef ref="AgentAuditLog-HarFile"/>
         </Logger>
```

3. In the Appenders section, uncomment the `RollingFile` element for the engine audit log HAR file.

   Example:

```
     <Appenders>
         ...
         <RollingFile name="AgentAuditLog-HarFile"
                             fileName="${sys:pa.home}/log/
pingaccess_agent_audit_har.log"
                             filePattern="${sys:pa.home}/log/
pingaccess_agent_audit_har.%d{yyyy-MM-dd}.log"
                             ignoreExceptions="false">
             <StatusCodeRegExFilter regex=".*"/>
             <HarLogLayout>
                 <KeyValuePair key="AUDIT.metadata" value="true"/>
                 <KeyValuePair key="AUDIT.http-client" value="true"/>
                 <KeyValuePair key="AUDIT.http-app" value="true"/>
             </HarLogLayout>
             <Policies>
                 <TimeBasedTriggeringPolicy />
             </Policies>
         </RollingFile>
     </Appenders>
```

4. Optional: To filter the entries to add to the log file, edit the value in the `StatusCodeRegExFilter` element.

5. Optional: To specify what information to log, add or edit the values in the `HarLogLayout` section of the `RollingFile` element.

   You can add or edit metadata, client response, and app response values. See *Traffic logging reference* on page 127 for more information.

Result

Logging begins when the configuration is reloaded. The configuration is reloaded at regular intervals according to the `monitorInterval` value.

Enabling sideband traffic logging

Enable sideband audit logging, including end-user transactions captured by the sideband client request.

Steps

1. Edit the `<PA_HOME>/conf/log4j2.xml` file.
2. In the Logger section, uncomment the `AppenderRef` element for the sideband audit log HAR file.

   Example:

```
        <!-- Audit Log Configuration-->
         ...
         <Logger name="sidebandaudit" level="INFO" additivity="false">
             <AppenderRef ref="SidebandAuditLog-File"/>
             <!--<AppenderRef ref="SidebandAuditLog-Database-Failover"/>--
>
             <!--<AppenderRef ref="SidebandAuditLog-SQLServer-Database-
Failover"/>-->
             <!--<AppenderRef ref="SidebandAuditLog-PostgreSQL"/>-->
             <!--<AppenderRef ref="SidebandAudit2Splunk"/>-->
             <AppenderRef ref="SidebandAuditLog-HarFile"/>
         </Logger>
```

3. In the Appenders section, uncomment the `RollingFile` element for the engine audit log HAR file.

   Example:

```
    <Appenders>
        ...
        <RollingFile name="SidebandAuditLog-HarFile"
                     fileName="${sys:pa.home}/log/
pingaccess_sideband_audit_har.log"
                     filePattern="${sys:pa.home}/log/
pingaccess_sideband_audit_har.%d{yyyy-MM-dd}.log"
                     ignoreExceptions="false">
            <StatusCodeRegExFilter regex="5.."/>
            <HarLogLayout clientBodySizeLimit="16384"
 appBodySizeLimit="16384">
                <KeyValuePair key="AUDIT.metadata" value="true"/>
                <KeyValuePair key="AUDIT.http-client" value="true"/>
                <KeyValuePair key="AUDIT.http-app" value="true"/>
            </HarLogLayout>
            <Policies>
                <TimeBasedTriggeringPolicy />
            </Policies>
        </RollingFile>
        ...
    </Appenders>
```

4. Optional: To filter the entries to add to the log file, edit the value in the `StatusCodeRegExFilter` element.

5. Optional: To specify what information to log, add or edit the values in the `HarLogLayout` section of the `RollingFile` element.

   You can add or edit metadata, client response, and app response values. See *Traffic logging reference* on page 127 for more information.

Result

Logging begins when the configuration is reloaded. The configuration is reloaded at regular intervals according to the `monitorInterval` value.

Enabling sideband client traffic logging

Enable sideband client audit logging, including transactions sent to or from the sideband client integration.

Steps

1. Edit the `<PA_HOME>/conf/log4j2.xml` file.
2. In the Logger section, uncomment the `AppenderRef` element for the sideband client audit log HAR file.

   Example:

```
        <!-- Audit Log Configuration-->
         ...
         <Logger name="sidebandclientaudit" level="INFO"
 additivity="false">
             <AppenderRef ref="SidebandClientAuditLog-File"/>
             <!--<AppenderRef ref="SidebandClientAuditLog-Database-
Failover"/>-->
             <!--<AppenderRef ref="SidebandClientAuditLog-SQLServer-
Database-Failover"/>-->
             <!--<AppenderRef ref="SidebandClientAuditLog-PostgreSQL"/>-->
             <!--<AppenderRef ref="SidebandClientAudit2Splunk"/>-->
             <AppenderRef ref="SidebandClientAuditLog-HarFile"/>
         </Logger>
```

3. In the Appenders section, uncomment the `RollingFile` element for the engine audit log HAR file.

   Example:

```
     <Appenders>
         ...
         <RollingFile name="SidebandClientAuditLog-HarFile"
                      fileName="${sys:pa.home}/log/
pingaccess_sideband_client_audit_har.log"
                      filePattern="${sys:pa.home}/log/
pingaccess_sideband_client_audit_har.%d{yyyy-MM-dd}.log"
                      ignoreExceptions="false">
             <StatusCodeRegExFilter regex="5.."/>
             <HarLogLayout clientBodySizeLimit="16384"
 appBodySizeLimit="16384">
                 <KeyValuePair key="AUDIT.metadata" value="true"/>
                 <KeyValuePair key="AUDIT.http-client" value="true"/>
                 <KeyValuePair key="AUDIT.http-app" value="true"/>
             </HarLogLayout>
             <Policies>
                 <TimeBasedTriggeringPolicy />
             </Policies>
         </RollingFile>
         ...
     </Appenders>
```

4. Optional: To filter the entries to add to the log file, edit the value in the `StatusCodeRegExFilter` element.
5. Optional: To specify what information to log, add or edit the values in the `HarLogLayout` section of the `RollingFile` element.

   You can add or edit metadata, client response, and app response values. See *Traffic logging reference* on page 127 for more information.

Result

Logging begins when the configuration is reloaded. The configuration is reloaded at regular intervals according to the `monitorInterval` value.

Traffic logging reference

You can include these metadata, client, and app elements in traffic logs.

Element hierarchy

Each section described here has child elements. If there is a disagreement in settings, the most specific setting is used.

For example, if the metadata element is set to false but the exchange ID is set to true, then only the exchange ID is logged. If the metadata element is set to true but the exchange ID is set to false, then all metadata elements except the exchange ID are logged.

Limitations

The traffic logs have the following limitations:

- If a request or response body is chunked, only the first chunk is logged by traffic logging.
- Request and response bodies are not decoded.

Metadata elements

You can include metadata elements in the API, engine, and audit traffic logs. These elements provide general information about the logged event.

Item	Description
`AUDIT.metadata`	Section setting for all metadata elements.
`AUDIT.exchangeId`	Identifies the ID for a specific request/response pair.
`AUDIT.applicationId`	Specifies the ID of the requested application.
`AUDIT.applicationName`	Specifies the name of the requested application.
`AUDIT.resourceId`	Specifies the ID of the requested resource.
`AUDIT.resourceName`	Specifies the name of the requested resource.
`AUDIT.pathPrefix`	Specifies the path prefix of the requested application or resource.
`AUDIT.pathPrefixType`	Indicates the pattern type of the path prefix, `Wildcard` or `Regex`.

Item	Description
`AUDIT.authMech`	Mechanism used for authentication. Engine Auditing - Cookie (WAM session), OAuth, unknown (for example, pass-through or static assets). Pass-through assets are Resources with no policies or Web session configured. Admin Auditing - Basic, OAuth, Cookie, unknown (unknown displays only in an authentication failure).
`AUDIT.client`	IP address of the requesting client.
`AUDIT.failedRuleName`	Name of the rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the engine log.
`AUDIT.failedRuleType`	Type of rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the engine log.
`AUDIT.failedRuleClass`	The Java class of rule that failed. If no rule failure occurred, this field is blank. This element is applicable only to the engine log.
`AUDIT.failedRuleSetName`	Name of the containing rule set that failed. If no rule failure occurred, this field is blank. This element is applicable only to the engine log.
`AUDIT.host`	PingAccess host name or IP address.
`AUDIT.targetHost`	Backend target that processed the request and generated a response to the PingAccess engine. This variable is unset when the response is generated by a target host protected by PingAccess.
`AUDIT.resource`	Name of the resource used to fulfill the request. This element is applicable only to the engine log.
`AUDIT.subject`	Subject of the transaction.
`AUDIT.trackingId`	The PingFederate tracking ID. This element can be used to help correlate audit information in the PingAccess audit log with information recorded in the PingFederate audit log. The value of this depends on whether the application type is `Web` or `API`. If the application type is `Web`, the value is presented as `tid:`*`<Session_Identifier>`*. The <*Session_Identifier*> can be used by the *PingFederate Session Revocation API* to revoke the session without disabling the user in the identity store. If the application type is `API`, the value is presented as `atid:`*`<Hash>`*. The <*Hash*> value is derived from the OAuth Access token for the session, and only serves as an identifier; it cannot be used for session revocation.

The following example shows the metadata section with all elements set to true.

Example

```
                <!-- AUDIT.metadata is the section setting for the following
fields: -->
                <!-- AUDIT.exchangeId to AUDIT.trackingId -->
                <KeyValuePair key="AUDIT.metadata" value="true"/>
                <KeyValuePair key="AUDIT.exchangeId" value="true"/>
                <KeyValuePair key="AUDIT.applicationId" value="true"/>
                <KeyValuePair key="AUDIT.applicationName" value="true"/>
                <KeyValuePair key="AUDIT.resourceId" value="true"/>
                <KeyValuePair key="AUDIT.resourceName" value="true"/>
                <KeyValuePair key="AUDIT.pathPrefix" value="true"/>
                <KeyValuePair key="AUDIT.pathPrefixType" value="true"/>
                <KeyValuePair key="AUDIT.authMech" value="true"/>
                <KeyValuePair key="AUDIT.client" value="true"/>
                <KeyValuePair key="AUDIT.failedRuleName" value="true"/>
                <KeyValuePair key="AUDIT.failedRuleType" value="true"/>
                <KeyValuePair key="AUDIT.failedRuleClass" value="true"/>
                <KeyValuePair key="AUDIT.failedRuleSetName" value="true"/>
                <KeyValuePair key="AUDIT.host" value="true"/>
                <KeyValuePair key="AUDIT.targetHost" value="true"/>
                <KeyValuePair key="AUDIT.resource" value="true"/>
                <KeyValuePair key="AUDIT.subject" value="true"/>
                <KeyValuePair key="AUDIT.trackingId" value="true"/>
```

HTTP client elements

Client elements provide information about requests made to PingAccess by clients, and the response sent back to the client. For example, a user making a call to the PingAccess administrative API is considered client traffic. You can include client elements in the API, engine, and audit traffic logs.

Item	Description
`AUDIT.http-client`	Section setting for all client elements.
`AUDIT.http-client-started-date-time`	Date and time of the beginning of the request.
`AUDIT.http-client-time`	Total elapsed time of the request and response.
`AUDIT.http-client-request-method`	Method used in the request.
`AUDIT.http-client-request-target`	The portion of the URL after the host and port.
`AUDIT.http-client-request-http-version`	HTTP version used by the request.
`AUDIT.http-client-request-cookies`	List of all cookies in the request. Parent element for `AUDIT.http-client-request-cookie-`*`{cookie}`*.
`AUDIT.http-client-request-cookie-`*`{cookie}`*	Information about the request cookie with the specified name. You can include this element multiple times for different cookie names.
`AUDIT.http-client-request-headers`	List of all headers in the request. Parent element for `AUDIT.http-client-request-header-`*`{header}`*.
`AUDIT.http-client-request-header-`*`{header}`*	Information about the request header with the specified name. You can include this element multiple times for different header names.

Item	Description
`AUDIT.http-client-request-query-strings`	List of all parameters and values parsed from the request query string. Parent element for `AUDIT.http-client-request-query-string-{query}`.
`AUDIT.http-client-request-query-string-{query}`	Information about the request query string with the specified name. You can include this element multiple times for different query string names.
`AUDIT.http-client-request-post-data-mime-type`	Mime type of posted request data.
`AUDIT.http-client-request-post-data-text`	Posted request data, in plain text.
`AUDIT.http-client-request-headers-size`	Size, in bytes, of the header from the start of the request to the body.
`AUDIT.http-client-request-body-size`	Size, in bytes, of the request body.
`AUDIT.http-client-response-status-code`	Response status code.
`AUDIT.http-client-response-status-text`	Response status description.
`AUDIT.http-client-response-http-version`	HTTP version used by the response.
`AUDIT.http-client-response-cookies`	List of all cookies in the response. Parent element for `AUDIT.http-client-response-cookie-{cookie}`.
`AUDIT.http-client-response-cookie-{cookie}`	Information about the response cookie with the specified name. You can include this element multiple times for different cookie names.
`AUDIT.http-client-response-headers`	List of all headers in the response. Parent element for `AUDIT.http-client-response-header-{header}`.
`AUDIT.http-client-response-header-{header}`	Information about the response header with the specified name. You can include this element multiple times for different header names.
`AUDIT.http-client-response-content-size`	Size, in bytes, of the response content.
`AUDIT.http-client-response-content-mime-type`	Mime type of the response content.
`AUDIT.http-client-response-content-text`	Response body.
`AUDIT.http-client-response-redirect-url`	Redirect target URL from the location response header.
`AUDIT.http-client-response-headers-size`	Size, in bytes, of the header from the start of the response to the body.
`AUDIT.http-client-response-body-size`	Size, in bytes, of the response body.

The following example shows the client section with all elements set to true.

Example

```
                <!-- AUDIT.http-client is the section setting for the
 following fields: -->
                <!-- AUDIT.http-client-started-date-time to AUDIT.http-
client-response-body-size -->
                <KeyValuePair key=”AUDIT.http-client” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-started-date-time”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-time” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-method”
 value=”true”/>
                <!-- Note: “AUDIT.http-client-request-target” is the target
 part of the url -->
                <KeyValuePair key=”AUDIT.http-client-request-target”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-http-version”
 value=”true”/>
                <!-- Sets the default value for all client request cookies.
 -->
                <!-- This overrides AUDIT.http-client and is overridden by
 individual cookie values. -->
                <KeyValuePair key=”AUDIT.http-client-request-cookies”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-cookie-
{cookie}” value=”true”/>
                <!-- Sets the default value for all client request headers.
 -->
                <!-- This overrides AUDIT.http-client and is overridden by
 individual header values. -->
                <KeyValuePair key=”AUDIT.http-client-request-headers”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-header-
{header}” value=”true”/>
                <!-- Sets the default value for all client request query
 strings. -->
                <!-- This overrides AUDIT.http-client and is overridden by
 individual query strings. -->
                <KeyValuePair key=”AUDIT.http-client-request-query-strings”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-query-string-
{query}” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-post-data-mime-
type” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-post-data-text”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-headers-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-request-body-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-status-code”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-status-text”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-http-version”
 value=”true”/>
                <!-- Sets the default value for all client response cookies.
 -->
                <!-- This overrides AUDIT.http-client and is overridden by
 individual cookie values. -->
                <KeyValuePair key=”AUDIT.http-client-response-cookies”
 value=”true”/>
```

```
                <KeyValuePair key=”AUDIT.http-client-response-cookie-
{cookie}” value=”true”/>
                <!-- Sets the default value for all client response headers.
 -->
                <!-- This overrides AUDIT.http-client and is overridden by
 individual header values. -->
                <KeyValuePair key=”AUDIT.http-client-response-headers”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-header-
{header}” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-content-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-content-mime-
type” value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-content-text”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-redirect-url”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-headers-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-client-response-body-size”
 value=”true”/>
```

HTTP app elements

App elements provide information about requests made by PingAccess to other tools or services such as PingFederate, and the response sent back to PingAccess. For example, PingAccess making a call to a protected resource is considered app traffic. You can include app elements in the engine and audit traffic logs.

Item	**Description**
`AUDIT.http-app`	Section setting for all app elements.
`AUDIT.http-app-started-date-time`	Date and time of the beginning of the request.
`AUDIT.http-app-time`	Total elapsed time of the request and response.
`AUDIT.http-app-request-method`	Method used in the request.
`AUDIT.http-app-request-target`	The portion of the URL after the host and port.
`AUDIT.http-app-request-http-version`	HTTP version used by the request.
`AUDIT.http-app-request-cookies`	List of all cookies in the request. Parent element for `AUDIT.http-app-request-cookie-`*`{cookie}`*.
`AUDIT.http-app-request-cookie-`*`{cookie}`*	Information about the request cookie with the specified name. You can include this element multiple times for different cookie names.
`AUDIT.http-app-request-headers`	List of all headers in the request. Parent element for `AUDIT.http-app-request-header-`*`{header}`*.
`AUDIT.http-app-request-header-`*`{header}`*	Information about the request header with the specified name. You can include this element multiple times for different header names.

Item	Description
`AUDIT.http-app-request-query-strings`	List of all parameters and values parsed from the request query string. Parent element for `AUDIT.http-app-request-query-string-{query}`.
`AUDIT.http-app-request-query-string-{query}`	Information about the request query string with the specified name. You can include this element multiple times for different query string names.
`AUDIT.http-app-request-post-data-mime-type`	Mime type of posted data.
`AUDIT.http-app-request-post-data-text`	Posted data, in plain text.
`AUDIT.http-app-request-headers-size`	Size, in bytes, of the header from the start of the request to the body.
`AUDIT.http-app-request-body-size`	Size, in bytes, of the request body.
`AUDIT.http-app-response-status-code`	Response status code.
`AUDIT.http-app-response-status-text`	Response status description.
`AUDIT.http-app-response-http-version`	HTTP version used by the response.
`AUDIT.http-app-response-cookies`	List of all cookies in the response. Parent element for `AUDIT.http-app-response-cookie-{cookie}`.
`AUDIT.http-app-response-cookie-{cookie}`	Information about the response cookie with the specified name. You can include this element multiple times for different cookie names.
`AUDIT.http-app-response-headers`	List of all headers in the response. Parent element for `AUDIT.http-app-response-header-{header}`.
`AUDIT.http-app-response-header-{header}`	Information about the response header with the specified name. You can include this element multiple times for different header names.
`AUDIT.http-app-response-content-size`	Size, in bytes, of the response content.
`AUDIT.http-app-response-content-mime-type`	Mime type of the response content.
`AUDIT.http-app-response-content-text`	Response body.
`AUDIT.http-app-response-redirect-uri`	Redirect target URL from the location response header.
`AUDIT.http-app-response-headers-size`	Size, in bytes, of the header from the start of the response to the body.
`AUDIT.http-app-response-body-size`	Size, in bytes, of the response body.

The following example shows the app section with all elements set to true.

Example

```
               <!-- AUDIT.http-app is the section setting for the following
 fields: -->
```

```
                <!-- AUDIT.http-app-started-date-time to AUDIT.http-app-
response-body-size -->
                <KeyValuePair key=”AUDIT.http-app” value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-started-date-time”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-time” value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-method”
 value=”true”/>
                <!-- Note: “AUDIT.http-app-request-target” is the target
 part of the url -->
                <KeyValuePair key=”AUDIT.http-app-request-target”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-http-version”
 value=”true”/>
                <!-- Sets the default value for all app request cookies. -->
                <!-- This overrides AUDIT.http-app and is overridden by
 individual cookie values. -->
                <KeyValuePair key=”AUDIT.http-app-request-cookies”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-cookie-{cookie}”
 value=”true”/>
                <!-- Sets the default value for all app request headers. -->
                <!-- This overrides AUDIT.http-app and is overridden by
 individual header values. -->
                <KeyValuePair key=”AUDIT.http-app-request-headers”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-header-{header}”
 value=”true”/>
                <!-- Sets the default value for all app request query
 strings. -->
                <!-- This overrides AUDIT.http-app and is overridden by
 individual query strings. -->
                <KeyValuePair key=”AUDIT.http-app-request-query-strings”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-query-string-
{query}” value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-post-data-mime-
type” value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-post-data-text”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-headers-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-request-body-size”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-response-status-code”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-response-status-text”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-response-http-version”
 value=”true”/>
                <!-- Sets the default value for all app response cookies. --
>
                <!-- This overrides AUDIT.http-app and is overridden by
 individual cookie values. -->
                <KeyValuePair key=”AUDIT.http-app-response-cookies”
 value=”true”/>
                <KeyValuePair key=”AUDIT.http-app-response-cookie-{cookie}”
 value=”true”/>
                <!-- Sets the default value for all app response headers. --
>
                <!-- This overrides AUDIT.http-app and is overridden by
 individual header values. -->
                <KeyValuePair key=”AUDIT.http-app-response-headers”
 value=”true”/>
```

```
                <KeyValuePair key="AUDIT.http-app-response-header-{header}"
 value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-content-size"
 value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-content-mime-
type" value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-content-text"
 value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-redirect-uri"
 value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-headers-size"
 value="true"/>
                <KeyValuePair key="AUDIT.http-app-response-body-size"
 value="true"/>
```

Parsing HAR-formatted audit log files

Reformat a snapshot instance of a HAR-formatted audit log file so that you can view it or parse it with a HTTP Archive (HAR) reader.

About this task

> **Remember:**
>
> Before sharing any HAR data with a third-party application, carefully review the third-party application's permissions and sanitize any potentially sensitive information out of the log files.
>
> - For more information on configuring regex filters for your log files, see *Log traffic for troubleshooting* on page 119.
> - For information on other precautions, see *Security audit logging* on page 111.

Steps

1. Download the jq command-line tool from *https://stedolan.github.io/jq/download/*.

   Select a jq version for the operating system that you deployed your PingAccess environment on.

   For more information on PingAccess operating system requirements, see *System requirements* on page 50.
2. Create a file called `pa-har-merge.jq`.

   Example:

```
{
log: {
  version: .[0].log.version,
  creator: .[0].log.creator,
  entries: (reduce .[] as $entry ([]; . + ($entry.log.entries | map(. +
 { _metadata: $entry.log._metadata })))
  }
```

```
}
```

For examples of how to parse the PingAccess HAR-formatted log files with `pa-merge-har.jq`, see the following commands. These examples assume that:

- You've set **`PA_HOME`** and **`PA_HAR_MERGE_HOME`** as environment variables that define the base paths to the PingAccess instance and the `pa-merge-har.jq` file respectively.
- You're attempting to parse the HAR-formatted API audit log file.

To filter requests based on request URL, run the command:

```
cat $PA_HOME/log/pingaccess_api_audit_har.log | jq -s -f
 $PA_HAR_MERGE_HOME/pa-har-merge.jq | jq '.log.entries = [ .log.entries[]
 | select(.request.url != "/pa-admin-api/v3/adminSessionInfo/
checkOnly") ]
```

To output the HAR-formatted log file into a file format that's usable with a standard HAR viewer, run the command:

```
cat $PA_HOME/log/pingaccess_api_audit_har.log | jq -s -f
 $PA_HAR_MERGE_HOME/pa-har-merge.jq > log.har
```

Note:

View the output `log.har` file with a standard HAR viewer, such as browser dev tools or the HTTP Archive Viewer.

Other logging formats

You can write logs in additional formats.

You can write the audit logs to an *Oracle or SQL Server database*.

You can also configure PingAccess to write the audit logs to a differently formatted log file that can easily be digested by *Splunk*.

For more information, see the following topics:

- *Writing logs to databases* on page 136
- *Writing audit logs for Splunk* on page 141

Writing logs to databases

Enable database logging for the API, engine, and agent audit logs in `conf/log4j2.db.properties`.

About this task

Scripts are provided in `conf/log4j/sql-scripts` to create the necessary tables. PingAccess supports logging to Oracle, SQL Server, and PostgreSQL databases.

Steps

1. Ensure that your database driver JAR file is installed in the `<PA_HOME>/deploy` directory.
2. Restart PingAccess after installing the driver.
3. In `conf/log4j2.xml`, uncomment one or more of the preset appender configurations listed in the following table.

Database	Configuration
Oracle	▪ For administrative API audit logging, uncomment the `<JDBC>` element

Database	Configuration
	with the `name="ApiAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For engine audit logging, uncomment the `<JDBC>` element with the `name="EngineAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For agent audit logging, uncomment the `<JDBC>` element with the `name="AgentAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For sideband client audit logging, uncomment the `<JDBC>` element with the `name="SidebandClientAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For sideband end-user audit logging, uncomment the `<JDBC>` element with the `name="SidebandAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements.
SQL Server	▪ For administrative API audit logging, uncomment the `<JDBC>` element with the `name="ApiAuditLog-SQLServer-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For engine audit logging, uncomment the `<JDBC>` element with the `name="EngineAuditLog-SQLServer-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For agent audit logging, uncomment the `<JDBC>` element with the `name="AgentAuditLog-SQLServer-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For sideband client audit logging, uncomment the `<JDBC>` element with the `name="SidebandClientAuditLog-SQLServer-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements.

Database	Configuration
	▪ For sideband end-user audit logging, uncomment the `<JDBC>` element with the `name="SidebandAuditLog-SQLServer-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements.
PostgreSQL	▪ For administrative API audit logging, uncomment the `<JDBC>` element with the `name="ApiAuditLog-PostgreSQL-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For engine audit logging, uncomment the `<JDBC>` element with the `name="EngineAuditLog-PostgreSQL-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For agent audit logging, uncomment the `<JDBC>` element with the `name="AgentAuditLog-PostgreSQL-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For sideband client audit logging, uncomment the `<JDBC>` element with the `name="SidebandClientAuditLog-PostgreSQL-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements. ▪ For sideband end-user audit logging, uncomment the `<JDBC>` element with the `name="SidebandAuditLog-PostgreSQL-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements.

Note:

The `<PingAccessFailover>` element is used to define how PingAccess logging fails over if a connection to the primary database is not accessible. Use the `retryIntervalSeconds` attribute to specify the number of seconds that must pass before retrying the primary JDBC appender.

4. In `conf/log4j2.db.properties`, replace the placeholder parameter values for each enabled appender with valid values to provide access to the database.

Info:

You can obfuscate the password used to access the database by running either **`obfuscate.sh`** or **`obfuscate.bat`**, located in *`<PA_HOME>`*`/bin`. Use the database password as an argument, then copy the output into the password configuration property for the appender in *<PA_HOME>*/conf/log4j2.db.properties.

5. In conf/log4j2.xml, uncomment the <AppenderRef> elements in each respective <Logger> section as shown in the following examples.

   Example: Oracle

```
<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
    <AppenderRef ref="APIAuditLog-File"/>
    <AppenderRef ref="ApiAuditLog-Database-Failover"/>
    <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>-->
    <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
    <!--<AppenderRef ref="ApiAuditLog-HarFile"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
    <AppenderRef ref="EngineAuditLog-File"/>
    <AppenderRef ref="EngineAuditLog-Database-Failover"/>
    <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>--
>
    <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
    <!--<AppenderRef ref="EngineAuditLog-HarFile"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
    <AppenderRef ref="AgentAuditLog-File"/>
    <AppenderRef ref="AgentAuditLog-Database-Failover"/>
    <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>-->
    <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
    <!--<AppenderRef ref="AgentAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandclientaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandClientAuditLog-File"/>
    <AppenderRef ref="SidebandClientAuditLog-Database-Failover"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-SQLServer-Database-
Failover"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="SidebandClientAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandAuditLog-File"/>
    <AppenderRef ref="SidebandAuditLog-Database-Failover"/>
    <!--<AppenderRef ref="SidebandAuditLog-SQLServer-Database-Failover"/
>-->
    <!--<AppenderRef ref="SidebandAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="SidebandAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-HarFile"/>-->
</Logger>
```

   Example: SQL Server

```
<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
    <AppenderRef ref="APIAuditLog-File"/>
    <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
    <AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>
    <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
    <!--<AppenderRef ref="ApiAuditLog-HarFile"/>-->
```

```
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
    <AppenderRef ref="EngineAuditLog-File"/>
    <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
    <AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>
    <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
    <!--<AppenderRef ref="EngineAuditLog-HarFile"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
    <AppenderRef ref="AgentAuditLog-File"/>
    <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
    <AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>
    <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
    <!--<AppenderRef ref="AgentAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandclientaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandClientAuditLog-File"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-Database-Failover"/>-->
    <AppenderRef ref="SidebandClientAuditLog-SQLServer-Database-
Failover"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="SidebandClientAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandAuditLog-File"/>
    <!--<AppenderRef ref="SidebandAuditLog-Database-Failover"/>-->
    <AppenderRef ref="SidebandAuditLog-SQLServer-Database-Failover"/>
    <!--<AppenderRef ref="SidebandAuditLog-PostgreSQL"/>-->
    <!--<AppenderRef ref="SidebandAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-HarFile"/>-->
</Logger>
```

Example: PostgreSQL

```
<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
    <AppenderRef ref="APIAuditLog-File"/>
    <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>-->
    <AppenderRef ref="ApiAuditLog-PostgreSQL"/>
    <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
    <!--<AppenderRef ref="ApiAuditLog-HarFile"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
    <AppenderRef ref="EngineAuditLog-File"/>
    <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>--
>
    <AppenderRef ref="EngineAuditLog-PostgreSQL"/>
    <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
    <!--<AppenderRef ref="EngineAuditLog-HarFile"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
    <AppenderRef ref="AgentAuditLog-File"/>
    <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>-->
    <AppenderRef ref="AgentAuditLog-PostgreSQL"/>
    <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
    <!--<AppenderRef ref="AgentAuditLog-HarFile"/>-->
</Logger>
```

```
<Logger name="sidebandclientaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandClientAuditLog-File"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-SQLServer-Database-
Failover"/>-->
    <AppenderRef ref="SidebandClientAuditLog-PostgreSQL"/>
    <!--<AppenderRef ref="SidebandClientAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandAuditLog-File"/>
    <!--<AppenderRef ref="SidebandAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-SQLServer-Database-Failover"/
>-->
    <AppenderRef ref="SidebandAuditLog-PostgreSQL"/>
    <!--<AppenderRef ref="SidebandAudit2Splunk"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-HarFile"/>-->
</Logger>
```

6. Create the database tables. Scripts to create database tables are located in `conf/log4j/sql-scripts`.

> **Note:**
>
> The scripts are written to handle the default list of elements for the relevant database log appender. Any changes to the list require corresponding changes to the SQL table creation script, or to the table itself if it already exists. For more information on working with these scripts, see the Oracle, PostgreSQL, or MS SQL Server documentation.

> **Important:**
>
> For PostgreSQL database scripts, use of the default **`public`** schema is not recommended. To run the scripts against a different schema, choose one of the following options:
>
> - Prepend the schema before the table name. For example, `api_audit_log` would become `my_schema.api_audit_log`
> - Run the script using psql and specify an options parameter to define the schema. For example,
>
> ```
> psql postgresql://<user>@<db_hostname>:5432/<db_name>?options=--
> search_path=<schema> -f api-audit-log-postgresql.sql
> ```

Writing audit logs for Splunk

Ping Identity provides a custom Splunk app for PingAccess to process audit logs generated by a PingAccess deployment.

Before you begin

- Go to the *Splunk website* and download Splunk.
- Install Splunk.

About this task

Splunk is enterprise software that allows for monitoring, reporting, and analyzing consolidated log files. Splunk captures and indexes real-time data into a single searchable repository from which reports, graphs, and other data visualization can be generated.

The PingAccess app for Splunk provides rich system monitoring and reporting, including:

- Current transaction and system reports
- Service reports, such as a daily usage report and IdP and SP reports per connection
- Trend reports, such as weekly and monthly usage reports, and trend analysis

The application uses a specially formatted version of the audit logs, which are written to the PingAccess log directory when you perform the following steps.

Note:

The PingAccess app for Splunk is available separately. It requires enterprise-licensed (or trial) installation of the Splunk software and the Splunk Universal Forwarder, which collects data from the PingAccess Splunk audit logs. The application includes additional documentation on installation and available features.

Tip:

To download the free application from *Splunkbase.splunk.com*, search for PingAccess.

Note:

The PingAccess app for Splunk was designed to use the default Splunk log pattern configuration. If you have changed the output format of the Splunk rolling files, those changes could impact the functionality of the PingAccess App for Splunk.

Steps

1. Set up your Splunk server.
   a. Enable a receiver to listen for data from the servers hosting PingAccess.

      For more information, see the *Splunk documentation*.
   b. Install the PingAccess app for Splunk.

      For installation instructions, see the *Splunk Add-on documentation*.
2. Configure PingAccess to output the following available Splunk audit logs:
   - `pingaccess_engine_audit_splunk.log`
   - `pingaccess_api_audit_splunk.log`
   - `pingaccess_agent_audit_splunk.log`

   Note:

   These logs output to the `<PA_HOME>/log/` by default.

   a. Edit `<PA_HOME>/conf/log4j2.xml`.
   b. In the `Audit Log Configuration` section, edit the `apiaudit`, `engineaudit`, and `agentaudit` logger configurations to uncomment the Splunk `AppenderRef`:

      Example:

```
<!-- ======================= -->
<!-- Audit log configuration -->
<!-- ======================= -->
<Logger name="apiaudit" level="INFO" additivity="false">
    <AppenderRef ref="APIAuditLog-File"/>
    <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
```

```
    <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>--
>
    <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
    <AppenderRef ref="ApiAudit2Splunk"/>
    <!--<AppenderRef ref="ApiAuditLog-HarFile"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
    <AppenderRef ref="EngineAuditLog-File"/>
    <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/
>-->
    <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
    <AppenderRef ref="EngineAudit2Splunk"/>
    <!--<AppenderRef ref="EngineAuditLog-HarFile"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
    <AppenderRef ref="AgentAuditLog-File"/>
    <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/
>-->
    <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
    <AppenderRef ref="AgentAudit2Splunk"/>
    <!--<AppenderRef ref="AgentAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandclientaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandClientAuditLog-File"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-Database-Failover"/
>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-SQLServer-Database-
Failover"/>-->
    <!--<AppenderRef ref="SidebandClientAuditLog-PostgreSQL"/>-->
    <AppenderRef ref="SidebandClientAudit2Splunk"/>
    <!--<AppenderRef ref="SidebandClientAuditLog-HarFile"/>-->
</Logger>
<Logger name="sidebandaudit" level="INFO" additivity="false">
    <AppenderRef ref="SidebandAuditLog-File"/>
    <!--<AppenderRef ref="SidebandAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-SQLServer-Database-
Failover"/>-->
    <!--<AppenderRef ref="SidebandAuditLog-PostgreSQL"/>-->
    <AppenderRef ref="SidebandAudit2Splunk"/>
    <!--<AppenderRef ref="SidebandAuditLog-HarFile"/>-->
</Logger>
```

c. Uncomment the `RollingFile` appender references for the `ApiAudit2Splunk`, `EngineAudit2Splunk`, and `AgentAudit2Splunk` `RollingFile` elements.

Example:

This is the default configuration for the `ApiAudit2Splunk` file.

```
<!--
<RollingFile name="ApiAudit2Splunk"
             fileName="${sys:pa.home}/log/
pingaccess_api_audit_splunk.log"
             filePattern="${sys:pa.home}/log/
pingaccess_api_audit_splunk.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
    <PatternLayout>
        <pattern>%d{ISO8601} exchangeId="%X{exchangeId}"
 trackingId="%X{AUDIT.trackingId}" subject="%X{AUDIT.subject}"
 authMech="%X{AUDIT.authMech}" client="%X{AUDIT.client}"
 method="%X{AUDIT.method}" requestUri="%X{AUDIT.requestUri}"
```

```
 responseCode="%X{AUDIT.responseCode}" responder="%X{AUDIT.responder}"
 engineHostname="%X{AUDIT.host}" %n</pattern>
    </PatternLayout>
    <Policies>
        <TimeBasedTriggeringPolicy />
    </Policies>
</RollingFile>
-->
```

This is the updated configuration for the `ApiAudit2Splunk` file, with the `RollingFile` uncommented and no other changes.

```
<RollingFile name="ApiAudit2Splunk"
             fileName="${sys:pa.home}/log/
pingaccess_api_audit_splunk.log"
             filePattern="${sys:pa.home}/log/
pingaccess_api_audit_splunk.%d{yyyy-MM-dd}.log"
             ignoreExceptions="false">
    <PatternLayout>
        <pattern>%d{ISO8601} exchangeId="%X{exchangeId}"
 trackingId="%X{AUDIT.trackingId}" subject="%X{AUDIT.subject}"
 authMech="%X{AUDIT.authMech}" client="%X{AUDIT.client}"
 method="%X{AUDIT.method}" requestUri="%X{AUDIT.requestUri}"
 responseCode="%X{AUDIT.responseCode}" responder="%X{AUDIT.responder}"
 engineHostname="%X{AUDIT.host}" %n</pattern>
    </PatternLayout>
    <Policies>
        <TimeBasedTriggeringPolicy />
    </Policies>
</RollingFile>
```

3. Set up Splunk Universal Forwarder.
   a. Download the Splunk Universal Forwarder from *Splunk* and install it on the PingAccess server.
   b. Configure the Splunk Universal Forwarder to monitor the three Splunk log files (`pingaccess_engine_audit_splunk.log`, `pingaccess_api_audit_splunk.log`, and `pingaccess_agent_audit_splunk.log`) and forward the data to the receiver you configured.

   For detailed installation and configuration instructions, see *Splunk documentation*.

Customize and Localize PingAccess

This section contains information on how to customize PingAccess templates and localize user-facing messages.

- For information on customizing PingAccess page templates and understanding the difference between customizable templates and system templates, see *User-facing page customization reference* on page 144.
- For information on localizing user-facing system status messages, see *User-facing page localization reference* on page 148 .

User-facing page customization reference

PingAccess supplies templates to provide information to the end user. These template pages use the Velocity template engine, an open-source Apache project, and are located in the *<PA_HOME>*`/conf/template` directory.

You can modify most of these pages in a text editor to suit the particular branding and informational needs of your PingAccess installation. Cascading style sheets and images for these pages are included in the *<PA_HOME>*`/conf/static/pa/assets` subdirectory. Each page contains both Velocity constructs and standard HTML. The Velocity engine interprets the commands embedded in the template page before the

HTML is rendered in the user's browser. At runtime, the PingAccess server supplies values for the Velocity variables used in the template.

> **Important:**
>
> If you have modified the reserved application context root using the PingAccess Admin API, file system requests to the configured reserved application context root will be translated to `/pa`. This allows the file system behavior for PingAccess resources to remain unchanged. Thus, if the reserved context root is set to `/ping`, templates and other resources would still be stored on the file system in the `/pa` directory, as indicated by this document.

For information about Velocity, see *Velocity project documentation* on the Apache Web site. Changing Velocity or JavaScript code is not recommended. The following variables are the only variables that can be used for rendering the associated web browser page.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Variable	Description
`title`	The browser tab title for the message. For example, `Not Found`.
`header`	The header for the message. For example, `Not Found`.
`info`	The information for the message. For example, `No Resource configured for request`.
`exchangeId`	A value that identifies the request/response pair. This can be used to locate messages in the PingAccess logs.
`trackingId`	A value that identifies either the tracking ID, identified with a `tid:` prefix, or an access token ID, identified with a `atid:` prefix. This can be used to identify the session in the PingAccess and PingFederate logs.

Customizable page templates

At runtime, the user's browser is directed to the appropriate page, depending on the operation being performed and where the related condition occurs. For example, if rule evaluation fails, the user's browser is directed to the policy error-handling page. The following table describes each template.

Template File Name	Purpose	Type	Action
`admin.error.page.template.html`	Indicates an error occurred while the admin console was processing a request	Error	Consult *<PA_HOME>*`/log/pingaccess.log` to determine the underlying cause of the issue.
`general.error.page.template.html`	Indicates that an unknown error has occurred and provides an error message.	Error	Consult *<PA_HOME>*`/log/pingaccess.log` to determine the underlying cause of the issue.

Template File Name	Purpose	Type	Action
`general.loggedout.page.template.html`	Displayed when a user logs out of PingAccess.	Normal	User should close the browser.
`oauth.error.json`	Indicates that rule evaluation has failed and provides an optional error message. To customize this information, see Error-Handling Fields for OAuth rules documentation.	Normal	If necessary, consult the audit logs in *<PA_HOME>*`/log` for details about why the policy denied the request.
`policy.error.page.template.html`	Indicates that rule evaluation has failed and provides an optional error message. To customize this information, see Error-Handling Fields for rules documentation.	Normal	If necessary, consult the audit logs in *<PA_HOME>*`/log` for details about why the policy denied the request.

System Templates

The templates stored in *<PA_HOME>*`/conf/template/system` are system templates. Do not modify these templates directly unless directed by Ping. This table shows the purpose and associated action, if any, for each of these files.

File Name	Purpose	Type	Action
`admin.loggedout.page.template.html`	Displayed when a user completes a single logout (SLO) initiated from the PingAccess admin console.	Normal	The user's session at the identity provider (IdP) and the PingAccess administrative console has been terminated.
`agent.bootstrap.template.properties`	Used to generate the `agent.properties` file for an agent.	Normal	None
`engine.bootstrap.template.properties`	Used to generate the `bootstrap.properties` file for an engine.	Normal	None
`fragment.preservation.template.html`	Used to preserve the fragment from the requested URL in client-side storage during a PingAccess OpenID Connect (OIDC) sign-on flow.	Normal	None

File Name	Purpose	Type	Action
`fragment.preservation.response.html`	Used to restore the fragment from client-side storage for the originally requested URL when a PingAccess OIDC sign-on flow has completed.	Normal	None
`invalid.token.json`	Used to challenge a user agent for authentication when the user-agent specifies an Accept header field containing `application/json`.	Normal	The user agent interacts with the end user to obtain an OAuth token.
`post.preservation.request.html`	Used to preserve the HTML form data from a POST request in client-side storage during a PingAccess OIDC sign-on flow.	Normal	None
`post.preservation.response.encoded.html`	Used to submit encrypted HTML form data to PingAccess from a previously preserved POST request when a PingAccess OIDC sign-on flow completes.	Normal	None
`post.preservation.response.html`	Used to reconstruct an HTML form to resubmit restored POST data when a PingAccess OIDC sign-on flow completes.	Normal	None
`redirect.response.html`	Used to redirect a browser to the token provider for authentication.	Normal	None
`replica.bootstrap.template.properties`	Used to generate the bootstrap.properties file for a replica admin.	Normal	None
`site.authenticator.rst.xml`	Used to produce a request to send to the PingFederate security token service (STS) endpoint to exchange a PingAccess cookie or OAuth token for a Web Access Management (WAM) token.	Normal	None

File Name	Purpose	Type	Action
`unauthorized.response.html`	Used to produce a challenge for authentication to an OAuth client running in a browser-based application.	Normal	None

User-facing page localization reference

In addition to the use of Velocity templates to change the look and feel of user-facing pages, administrators can provide localized versions of user-facing status messages generated by PingAccess.

In `<PA_HOME>/conf/localization/`, properties files contain the messages to be returned to the client in various languages; by default, only English language messages are provided, using the default `pa-messages.properties` file. This file serves as a fallback for any message not found in other files in the directory.

The selection of a messages file is determined based on several different factors:

- The browser's `Accept-Langauge` header, based on a best-match first check against the `pa-messages` files
- The value of a cookie named `ping-accept-language`, which can be defined by the protected application
- A custom-developed PingAccess add-on that can customize the order of localization resolution

The default behavior allows the `ping-accept-language` cookie to override the browser preferences, and if that cookie is not set, then to use the `Accept-Language` header preference order, starting with the highest priority preference and trying to match the locale exactly. If none of the specified locales cannot be matched exactly, a more generic locale will be used, starting with the highest priority value.

If no matches are found, then the value in the `pa-messages.properties` file is used.

For example, suppose your browser had the following `Accept-Language` header,

```
Accept-Language: fr-CA;q=0.9, en-US;q=0.8
```

and PingAccess attempted to display a localized version of the message for

```
pa.response.status.service.unavailable
```

The order in which PingAccess searches for the string to display is:

1. `pa-messages_fr_CA.properties`
2. `pa-messages_en_US.properties`
3. `pa-messages_fr.properties`
4. `pa-messages_en.properties`
5. `pa-messages.properties`

If the `ping-accept-language` cookie is set by the protected application to the value `en-US`, then the above list would be ignored, and PingAccess would search for the string in:

1. `pa-messages_en_US.properties`
2. `pa-messages_en.properties`
3. `pa-messages.properties`

Important:

Most browsers support the use of an ordered list of languages. Safari is an exception to this. Even though the system supports an ordered list of languages, only the preferred language is sent with its requests.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Managing Federal Information Processing Standards (FIPS) mode

Federal Information Processing Standards (FIPS) mode ensures that PingAccess uses encryption algorithms that meet FIPS requirements.

When FIPS mode is enabled, PingAccess updates the `security.providers` list to use the Bouncy Castle FIPS (BCFIPS) provider and removes all unneeded security providers that are not FIPS-compliant.

Viewing your environment's FIPS mode status

If FIPS mode is enabled, you can view your environment's FIPS mode status in the administrative console or the audit log:

- To view FIPS mode in the administrative console, go to **Account** # **About** and in the **System Information** section, find FIPS mode status.
- To view FIPS mode in the audit log, review the audit log after starting PingAccess. If FIPS mode is enabled, an info-level entry indicates this status. For example:

```
INFO [] Fipsconfig - PingAccess is currently running in FIPS Mode.
```

Feature changes in FIPS mode

Some features of PingAccess operate differently or are unavailable in FIPS mode.

Certificate and private key format requirements:

- In non-FIPS mode, PingAccess supports PKCS#12 and PEM-formatted certificates and private keys. It automatically detects which format was used.
- In FIPS mode, PingAccess only supports PEM-formatted certificates and private keys. That is, key pairs can only be imported or exported using the PEM-encoded format. Only PBES2 and AES or Triple DES encryption are accepted and 128-bit salt is required. In practice, this could mean that you can only import PEM files generated by PingFederate.
- For PEM files in FIPS mode, the private key must precede the certificates.

Password format requirements:

- In FIPS mode, the password must contain at least 14 characters.

To manage FIPS mode, select a tab.

Enabling FIPS mode

About this task

Enable FIPS mode to ensure that PingAccess exclusively uses encryption algorithms permitted by the FIPS standard. If your environment is clustered, make sure to perform this procedure on all nodes.

Note:

In this procedure, you can manually specify security providers, TLS protocols, and TLS cipher suites that can be used. If your manual inclusions are not FIPS-compliant, your environment might not be FIPS-compliant even in FIPS mode.

Steps

1. Open the `<PA Home>/conf/fips-mode.properties` file, or create it if it has been removed.
2. Set the `pa.fips.mode` property to `true`.

```
pa.fips.mode=true
```

3. Optional: Exempt one or more security providers from being excluded by FIPS mode by adding a comma-separated list of class names to the `pa.fips.additionalAllowedProviders` property.

   Example:

```
pa.fips.additionalallowedproviders=X,Y
```

4. Optional: Add or remove TLS protocols by editing the `pa.fips.tls.protocols` property to include a comma-separated list of valid TLS protocols.

   The default is:

```
pa.fips.tls.protocols = TLSv1.2
```

5. Optional: Add or remove TLS cipher suites by editing the `pa.fips.tls.ciphers` property to include a comma-separated list of valid TLS cipher suites.

   The default is:

```
pa.fips.tls.ciphers = TLS_AES_256_GCM_SHA384, \
                      TLS_AES_128_GCM_SHA256, \
                      TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, \
                      TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, \
                      TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, \
                      TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, \
                      TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, \
                      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, \
                      TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, \
                      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, \
                      TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384, \
                      TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384, \
                      TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256, \
                      TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256, \
                      TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384, \
                      TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384, \
                      TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256, \
                      TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256, \
                      TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

6. Save and close the file.
7. Restart PingAccess.

Disabling FIPS Mode

About this task

Disable FIPS Mode to allow the use of non-FIPS compliant encryption. If your environment is clustered, perform this procedure on all nodes.

Steps

1. Open the *<PA Home>*`/conf/fips-mode.properties` file.
2. Set the `pa.fips.mode` property to `false`.

   ```
   pa.fips.mode=false
   ```
3. Save and close the file.
4. Restart PingAccess.

Configuring PingAccess to use Amazon Key Management Services

During initial startup, PingAccess automatically generates a randomized master key, which by default is not encrypted. If you are running in Amazon Web Services (AWS), you can configure PingAccess to use Amazon Key Management Services (KMS) to encrypt the master key.

Before you begin

- Make sure that you have an active connection to AWS.
- Use AWS KMS to generate a key to use for the PingAccess master key encryption.

> **Note:**
>
> For more information about managing access rights to your keys using key policies or AWS Identity and Access Management (IAM), see *AWS Key Management Service*.

About this task

To configure the encryption of the PingAccess master key, modify the `pa.jwk.properties` file found in *<PA_HOME>*`/conf`.

Steps

1. Stop PingAccess.
2. In a text editor, open *<PA_HOME>*`/conf/pa.jwk.properties`.
3. Locate the `pa.hostkey.masterKeyEncryptor` property .
4. Enable master key encryption.
   a. Change `com.pingidentity.pa.crypto.NoOpMasterKeyEncryptor` to the AWS KMS master key encryptor class name `com.pingidentity.pingcommons.aws.key.AwsKmsMasterKeyEncryptor`.
   b. Locate the ID for the key that you generated using AWS KMS.
   c. If this is not the first time starting PingAccess, prefix the key ID with "`ENCRYPT:`".

   Example: After making changes, the properties file should look similar to the following:

   ```
   pa.hostkey.masterKeyEncryptor=com.pingidentity.pingcommons.aws.key.AwsKmsMasterKeyEn

   pa.hostkey.keyId=ENCRYPT:d4e6adab-e20c-4339-ba76-e4cb1348713f
   ```
5. Save and close the updated `pa.jwk.properties` file.
6. Restart PingAccess.

   The PingAccess master file `pa.jwk` is encrypted using Amazon KMS.

Reference Guides

This section contains reference guides for common tasks related to making the most of your PingAccess environment and extending its capabilities.

- For information on enabling external applications to communicate with PingAccess, see *PingAccess API endpoints* on page 152.
- For information on setting up a clustered PingAccess environment, see *Clustering in PingAccess* on page 156.
- For a list of configurable parameters that PingAccess uses at runtime, see the *Configuration file reference* on page 163.
- For help deciding what type of PingAccess deployment best suits your environment, see the *PingAccess deployment guide* .
- For information on extending the functionality of PingAccess rules in your environment, see *Groovy in PingAccess*.
- For information on optimizing your PingAccess environment, see *Performance tuning* on page 218.

PingAccess API endpoints

The following endpoints enable external applications to communicate with the PingAccess server and provide complete administrative capabilities of the product.

Heartbeat endpoint

Enables administrators to verify that the server is running.

OpenID Connect endpoints

Enable PingFederate or other token providers to interface with PingAccess using the OpenID Connect (OIDC) protocol.

Authentication Token Management endpoint

Enables protected applications to validate authentication tokens issued by a PingAccess identity mapping.

OAuth endpoint

Enables an OAuth Authorization Server to interface with PingAccess as an OAuth Resource Server.

Administrative API endpoints

Enable users to use PingAccess administrative functions. These are REST APIs that include documentation and testing tools.

Important:

Some endpoint examples in this document include the `/pa` reserved path. This document assumes the default application reserved path has not been modified. You can modify the reserved path using the *PingAccess Admin API*. If the reserved path has been modified, update endpoint and other applicable application URLs appropriately.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Heartbeat endpoint

The heartbeat endpoint verifies that the PingAccess server is running and, depending on security settings, displays details about the configuration.

You can make this call to any active PingAccess listener and on any node in a PingAccess cluster. For example, with default port configurations, a clustered console replica will respond to this endpoint on port 9000, and a clustered engine will respond to it on port 3000.

/pa/heartbeat.ping

This endpoint returns a short or detailed status for the target PingAccess server, based on the value of the `enable.detailed.heartbeat.response` parameter in `run.properties`. Load balancers can use this endpoint to determine the status of PingAccess.

> **Info:**
>
> Begin the URL with the server name and the PingAccess runtime port number. For example: `https://hostname:3000/pa/heartbeat.ping`.

If an error is returned, this indicates that the PingAccess instance associated with the endpoint is down.

If `enable.detailed.heartbeat.response` is set to `false`, the default value, and the PingAccess instance is running, the endpoint returns an HTTP `200` status and the text `OK`.

If `enable.detailed.heartbeat.response` is set to `true` and the PingAccess instance is running, a configurable status with additional details is returned. The response output format is an Apache Velocity template defined in `PA_HOME/conf/template/heartbeat.page.json`. You can modify this template to suit your needs. The following values are available.

Value	Description
$monitor.getTotalJvmMemory('bytes'|'KB'|'MB'|'GB')	Returns the total memory in the Java virtual machine (JVM). Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getUsedJvmMemory('bytes'|'KB'|'MB'|'GB')	Returns the used memory in the JVM. Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getFreeJvmMemory('bytes'|'KB'|'MB'|'GB')	Returns the free memory in the JVM. Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getTotalPhysicalSystemMemory('bytes'|'KB'|'MB'|'GB')	Returns the total system memory. Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getTotalUsedPhysicalSystemMemory('bytes'|'KB'|'MB'|'GB')	Returns the used system memory. Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getTotalFreePhysicalSystemMemory('bytes'|'KB'|'MB'|'GB')	Returns the free system memory. Specify 'bytes', 'KB', "MB", or 'GB' to specify the units. 'bytes' is the default if not specified.
$monitor.getHostname()	Returns the hostname for the system running PingAccess.
$monitor.getNumberOfCpus()	Returns the number of CPU cores in the system.

Value	Description
$monitor.getCpuLoad('###.##')	Returns the current CPU utilization. The parameter contains an optional format value. If the format is specified, the value returned is returned as a percentage value from 0%-100%, formatted using the *Java DecimalFormat* specification. If no format value is specified, then the value returned is a real number from 0 to 1 which represents the CPU utilization percentage. For example, a format value of "###.##" will return a value similar to "56.12", but no specified format would result in the value being returned as "0.5612".
$monitor.getOpenClientConnections()	Returns the current number of clients connected to PingAccess.
$monitor.getNumberOfVirtualHosts()	Returns the current number of configured virtual hosts in PingAccess.
$monitor.getNumberOfApplications()	Returns the current number of configured applications in PingAccess.
$monitor.getNumberOfSites()	Returns the current number of configured sites in the PingAccess configuration database. In a clustered environment, on the engine nodes, this number will reflect the number of sites associated with applications rather than the number of configured sites that show on the admin node. For more information, see Server Clustering documentation. This value is not included in the default template, but can be added by the system administrator if desired.
$monitor.getLastRefreshTime('yyyy/MM/dd HH:mm:ss')	Returns the time the PingAccess configuration was last refreshed. The parameter specifies the date format to use; if no value is specified, the ISO 8601 date format is used. If the parameter is specified, the format used comes from the Joda DateTimeFormat specification.

The default content type for the output is **application/json**. However, you can specify a content type header using the `$monitor.setContentType()` line in the template.

If you update the `enable.detailed.heartbeat.response` value, you must restart PingAccess to make the new value take effect.

Calls to this endpoint can be logged in the audit log. You can enable the logging of heartbeat calls using the `/httpConfig/monitoring` administrative endpoint. For more information, see *Administrative API endpoints* on page 156.

OpenID Connect endpoints

Specific endpoints are needed for PingFederate or another token provider to interface with PingAccess using the OpenID Connect (OIDC) protocol.

These endpoints are available on the `engine.http.port` and `agent.http.port` ports defined in `<PA_HOME>/conf/run.properties`.

/pa/oidc/logout

This endpoint clears the browser cookie containing the PingAccess Token. This endpoint enables end users to trigger the removal of their own PingAccess Cookie from the browser they are using. The user is redirected to the logged out page. You can modify the template for this page, located at `<PA_INSTALL>/conf/template/general.loggedout.page.template.html`.

> **Info:**
>
> This endpoint does not retain any server-side state to denote log off. Additionally, unless single-logout (SLO) is selected for the token provider, this endpoint clears the cookie only from the requested host/domain, and the cookie might still exist in requests bound for other hosts/domains.

> **Note:**
>
> If you selected the **Use Single-Logout** option when configuring the token provider, this endpoint also sends a logout request to the token provider, which completes a full SLO flow.

/pa/oidc/cb

This endpoint, along with the application virtual host, becomes the redirect URI for the token provider configuration on the client.

/pa/oidc/JWKS

This endpoint is used by the token provider's JSON web token (JWT) Token Processor for signature verification. This endpoint must be used in conjunction with the configuration of a JWT token processor instance in the token provider. For more information on configuring a JWT in PingFederate, see the *PingFederate documentation*.

/pa/oidc/logout.png

This endpoint is used by the token provider to initiate a logout from PingAccess in conjunction with SLO functionality, terminating the PingAccess tokens across domains.

Authentication Token Management endpoint

/pa/authtoken/JWKS

The Authentication Token Management endpoint is used by backend sites to validate the signature of a JSON web token (JWT). For more information on JWT, see the *OAuth 2.0 Developer Guide Guide*.

OAuth endpoint

This page describes the endpoint used by an OAuth authorization server to interface with PingAccess as an OAuth resource server.

/pa/oauth/JWKS

An OAuth authorization server uses this endpoint to acquire PingAccess public keys for encryption of access tokens. The output uses the Internet Engineering Task Force (IETF) JSON web token (JWK) format for public keys.

Administrative API endpoints

PingAccess ships with interactive documentation for both developers and non-developers to explore the PingAccess API endpoints, view a reference of the metadata for each API, and experiment with API calls.

PingAccess APIs are REST APIs that provide complete administrative capabilities of the product. They can be called from custom applications or from command line tools such as cURL.

These endpoints are only available on the `admin.port` defined in `<PA_HOME>/conf/run.properties` at path `/pa-admin-api/v3/api-docs/` (`https://<PA_HOME>:<PORT>/pa-admin-api/v3/api-docs/`).

Note:

For enhanced API security, you must include `X-XSRF-Header: PingAccess` in all requests and use the `application/json` content type for PUT/POST requests.

Clustering in PingAccess

PingAccess provides clustering features that allow a group of PingAccess servers to appear as a single system.

Server clustering can facilitate high availability of critical services and can also increase performance and overall system throughput. However, availability and performance are often at opposite ends of the deployment spectrum. You might need to make some configuration tradeoffs that balance availability with performance to accommodate specific deployment goals.

Note:

Settings in the configuration file could affect the features documented here. For more information, see the *Configuration file reference* on page 163 guide.

Components of a PingAccess Cluster

PingAccess clusters are made up of three types of nodes:

The Administrative Node

Provides the administrator with a configuration interface.

The Replica Administrative Node

Provides the administrator with the ability to recover a failed administrative node using a manual failover procedure. For more information, see *Manually promoting the replica administrative node* on page 161.

The Engine Nodes

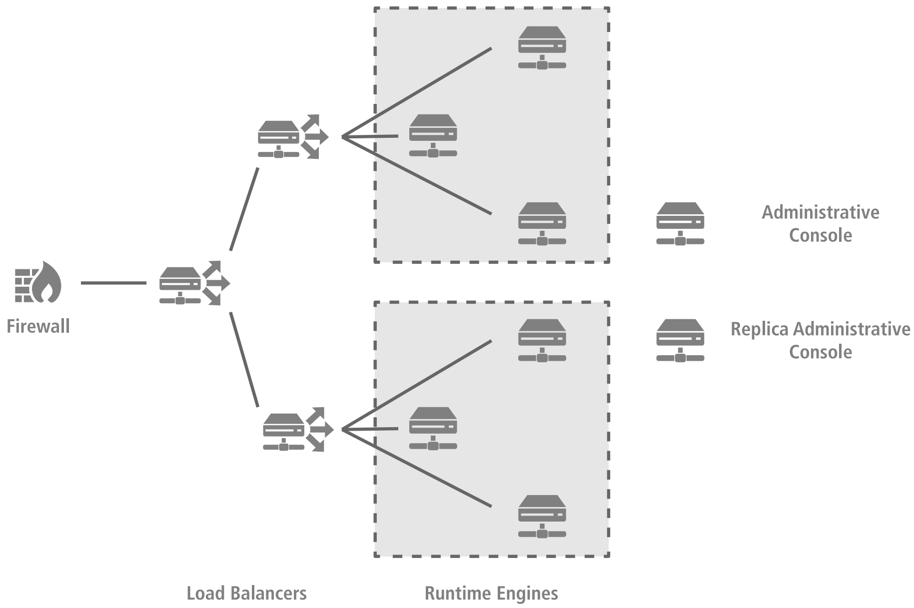
Handle incoming client requests and evaluate policy decisions based on the configuration replicated from the administrative node.

Note:

You can configure any number of engine nodes in a cluster, but you can configure only one administrative node and one replica administrative node in a cluster. State information is not shared between engine nodes.

Configuration information is replicated to all of the engine nodes and the replica administrative node from the administrative node, as is the license file on the administrative node. Engine nodes do not require a license to function, but some default templates look different depending on the information in the license.

You should manage incoming traffic to the engine nodes using load balancers or other mechanisms. PingAccess clusters do not dynamically manage or load-balance request traffic to individual engine nodes.



Node failure implications

Node failure within a PingAccess cluster can have short-term or long-term implications for your environment, depending on the state of your network and the type of node or nodes that failed. The following table describes some common node issues and recommends what kind of action to take.

Node issue	Result	Recommendation
Administrative node failure	The engine nodes can function using their stored configurations but cannot update their configurations.	Fail over to the replica administrative node until the administrative node can be restarted.
Replica administrative node failure	The engine nodes and administrative node can function normally, but you won't be able to fail over to the replica administrative node if something happens to the administrative node.	Restart the replica administrative node as soon as possible.
Administrative and replica node failure	The engine nodes can function using their stored configurations, but cannot update their configurations. No failover option is available.	Restart the administrative node as soon as possible, or restart the replica administrative node and fail over to it.

Node issue	Result	Recommendation
One or more engine nodes cannot reach the administrative node	Affected engine nodes can function using their stored configurations, if any but cannot update their configurations. If the administrative node performs key rolling, the affected engine nodes cannot recognize the new PingAccess internal cookie.	Restore access to the administrative node as soon as possible.

Cluster properties

Remember:

Use the `run.properties` and `bootstrap.properties` files to configure your environment.

In a cluster, you can configure each PingAccess node to serve as either an administrative node, a replica administrative node, or an engine node in the `run.properties` file. The `run.properties` file for the administrative node also contains server-specific configuration data.

At startup, a clustered PingAccess engine node checks its local configuration and then makes a call to the administrative node to check for changes. You can configure how often each engine node in a cluster checks the administrative node for changes in the engine `run.properties` file.

Information needed to bootstrap an engine node is stored in the `bootstrap.properties` file on each engine node.

bootstrap.properties

Property	Description
`engine.admin.configuration.host`	Defines the host where the administrative console is available. The default is `localhost`.
`engine.admin.configuration.port`	Defines the port where the administrative console is running. The default is `9000`.
`engine.admin.configuration.userid`	Defines the name of the engine.
`engine.admin.configuration.keypair`	Defines an elliptic curve key pair that is in the JSON Web Key (JWK) format.
`engine.admin.configuration.bootstrap.truststore`	Defines the trust store, in JWK format, that is used for communication with the administrative console.

Note:

You can tune the cache using the EHCache Configuration Properties, `pa.ehcache.*`, listed in the *Configuration file reference guide*.

Cluster node status

The administrative console provides two important visual elements which communicate the current status of the replica administrative node and the engine nodes:

- A status indicator, which communicates whether the node is healthy.
- A **Last Updated** field, which communicates the date and time that the node was last updated.

You can find this information on the **Administrative Nodes** page and the **Engines** page.

Status indicators use the value for <*admin.polling.delay*> as an interval to measure node health. A node's status can be green (good status), yellow (degraded status), or red (failed status):

Green (good status)

The node contacted the administrative node on the last pull request.

Yellow (degraded status)

The node contacted the administrative node between 2 and 10 intervals.

Red (failed status)

The node has either never contacted the administrative node or it has been more than 10 intervals since the nodes communicated.

Using multiple network interface cards to route traffic

PingAccess binds to all network interfaces by default to support routing traffic over multiple network interfaces. The default bind address PingAccess uses is 0.0.0.0. To prevent PingAccess from binding to all network interfaces, you can edit one or more of the following parameters in the `conf/run.properties` file:

```
admin.bindAddress=0.0.0.0
clusterconfig.bindAddress=0.0.0.0
engine.http.bindAddress=0.0.0.0
agent.http.bindAddress=0.0.0.0
```

Specify a new bind address for the parameter that you want to modify.

Configuring a PingAccess cluster

Install and configure PingAccess on each node in a cluster, including the administrative node, a replica administrative node, and one or more engine nodes.

About this task

The initial node you configure becomes the administrative node, which you will use to configure the rest of the cluster.

Important:

Setting the `pa.operational.mode` property on each node is part of the configuration process. Do not modify this property until directed to do so.

Steps

1. Install PingAccess on each cluster node.

2. Configure the administrative node:
   a. Open the `conf/run.properties` file in a text editor and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE`.

      This property is case-sensitive.
   b. Start PingAccess.
   c. Follow steps 1-14 of *Generating new key pairs* on page 313 to create a new key pair for the CONFIG QUERY listener. Make the following adjustments to steps 4-5:

      1. To complete step 4, enter the DNS name of the administrative node in the **Common Name** field.
      2. To complete step 5, enter both the DNS name of the replica administrative node and the DNS name of the administrative node in the **Subject Alternative Names** field. Alternately, configure the **Subject Alternative Names** field as a wildcard certificate.

      Note:

      You can use an IP address as the common name or in the **Subject Alternative Names** field, as long as those values are used in the administrative node fields on the **Administrative Nodes** configuration page.

      Remember:

      You will need this key pair in step 3a to set up the replica administrative console.
   d. Follow steps 1-4 of *Assigning key pairs to HTTPS listeners* on page 315 to assign the key pair you just created to the CONFIG QUERY listener.
   e. Follow steps 1-6 in *Configuring administrative nodes* on page 322 to configure the administrative node settings, then review the *What to do next* section. Make the following adjustment to step 2:

      1. To complete step 2, define the primary administrative node as a `host:port` pair in the **Host** field.

         Note:

         The host you specify must be a resolvable DNS name for the node or the node's IP address. The port must be the TCP port that PingAccess listens to for the administrative interface. By default, this port is 9090.
   f. Follow steps 1-14 of *Generating new key pairs* on page 313 to create a new key pair for the ADMIN listener. Make the following adjustments to steps 4-5:

      1. To complete step 4, enter the DNS name of the administrative node in the **Common Name** field.
      2. To complete step 5, enter both the DNS name of the replica administrative node and the DNS name of the administrative node in the **Subject Alternative Names** field. Alternately, configure the **Subject Alternative Names** field as a wildcard certificate.

      Note:

      You can use an IP address as the common name or in the **Subject Alternative Names** field as long as those values are used in the administrative node fields on the **Administrative Nodes** configuration page.
   g. Follow steps 1-4 of *Assigning key pairs to HTTPS listeners* on page 315 to assign the key pair you just created to the ADMIN listener.

h. Restart PingAccess.

3. Configure the replica administrative node.

> **Note:**
>
> If you add a replica administrative node after you deploy the cluster, you must update the configuration for each engine node.

   a. Complete steps 1-11 of *Configuring replica administrative nodes* on page 323. Make the following adjustments to step 2 and step 5:
      1. To complete step 2, the host you specify must be a resolvable DNS name for the node or the node's IP address. The port must be the TCP port that PingAccess listens to for the administrative interface. By default, this port is 9090.
      2. To complete step 5, select the key pair that you created for the CONFIG QUERY listener in step 2c of this topic as the **Replica Administrative Node Trusted Certificate**.
4. Configure the engine nodes in the cluster one at a time. For each engine node:
   a. Complete steps 1-10 of *Configuring engine nodes* on page 320.
   b. On the engine node, open the `conf/run.properties` file in a text editor and change the `pa.operational.mode` value to `CLUSTERED_ENGINE`.
   c. Complete step 11 of *Configuring engine nodes* on page 320.

   If you specified a proxy for the engine node, see the *What to do next* section also.

Next steps

1. Go to **Settings** # **System** # **Clustering** to check your cluster's status. If everything is configured properly, the cluster engine nodes and the replica administrative node should display a green status icon, indicating that the cluster is operational. For more information about status icons, see *Clustering in PingAccess* on page 156.
2. Optionally, you can configure each node in the cluster to run PingAccess as a service. This set-up prompts PingAccess to run automatically when you start a node. For more information, see *Running PingAccess as a service* on page 62 in *Installing and Uninstalling PingAccess*.

Manually promoting the replica administrative node

Manually promote the replica administrative node to the administrative node if the administrative node fails.

About this task

The replica administrative node is intended to be used for disaster recovery purposes. If you can recover the clustered console, then you should focus on recovery rather than failing over to the replica administrative node.

> **Warning:**
>
> Only one primary administrative node should be running for the cluster at any given time.

Steps

1. Open `<PA_HOME>/conf/run.properties` in a text editor.

2. Locate the `pa.operational.mode` line and change the value from `CLUSTERED_CONSOLE_REPLICA` to `CLUSTERED_CONSOLE`.

   These properties are case-sensitive.

   > **Important:**
   >
   > Do not restart the replica node during the promotion process. PingAccess can detect and apply this change without a restart, and restarting the node during its promotion can cause file corruption or failure to promote correctly.

Reinstating a replica administrative node after failing over

If you fail over to your replica administrative node, you must configure a new replica administrative node.

About this task

> **Note:**
>
> If you want to then switch back to the original console, you must recreate it as a replica administrative node, then fail over to it.

Steps

1. Install the new replica administrative node.
2. In the `run.properties` file, change the value for `pa.operational.mode` to `CLUSTERED_CONSOLE_REPLICA`.

   This property is case-sensitive.
3. Click **Settings** and then go to **Clustering** # **Administrative Nodes**.
4. Change the **Primary Administrative Node** host name and port to the failed-over node.
5. Remove the **Replica Administrative Node** public key, then change the **Replica Administrative Node** host name and port to point to the new replica node.

   > **Tip:**
   >
   > If your key pair does not include a wildcard, you can use the same host name as the original console to avoid having to recreate the console key pair and the `bootstrap.properties` files for each engine.
6. Click **Save & Download** to download the bootstrap file for the replica administrative node.
7. Copy the downloaded file to the new replica administrative node's `<PA_HOME>/conf` directory and rename it to `bootstrap.properties`.
8. Edit the `<PA_HOME>/conf/run.properties` file on the new replica administrative node and change the value of `pa.operational.mode` to `CLUSTERED_CONSOLE_REPLICA`.
9. Start the new replica node.
10. To verify replication has completed, monitor the `<PA_HOME>/log/pingaccess.log` file and find the message `Configuration successfully synchronized with administrative node`.

Configuration file reference

You can configure any of the following parameters used by PingAccess at runtime in the *<PA_HOME>*/conf/run.properties file.

In a clustered environment, each node has a unique `run.properties` file. Because changes to the `run.properties` file can significantly impact performance, use an identical `run.properties` configuration on all engine nodes.

Note:

Changes made to the `run.properties` file will take effect after the PingAccess service is restarted on the given node.

Tip:

When storing passwords in `run.properties`, you should obfuscate them using the `obfuscate.bat or obfuscate.sh` utility to mask the password value. This utility is located in the *<PA_HOME>*/bin folder.

If you are running PingAccess in FIPS mode, all SSL cipher and protocol settings in the `run.properties` file are ignored. See *Managing Federal Information Processing Standards (FIPS) mode* on page 149 for information about the protocols and ciphers used in this mode.

Operational mode

For more information about operational modes in PingAccess, see the following guides.

- *Clustering Reference Guide* – Discusses configuring nodes using clustered operational modes
- *Installing and Uninstalling PingAccess* on page 50 – Discusses installing PingAccess in standalone mode

pa.operational.mode

Controls the operational mode of the PingAccess server in a cluster. Valid values are:

Value	Description
`STANDALONE`	Use this value for a standalone (unclustered) PingAccess instance that runs both the administrative console and the engine. This is the default value.
`CLUSTERED_CONSOLE`	Use this value for the server instance you want to use as the administrative console server. **Info:** Only one engine in a cluster can run the administrative console.
`CLUSTERED_CONSOLE_REPLICA`	Use this value for the server instance you want to use as the backup administrative console server.

Value	Description
`CLUSTERED_ENGINE`	Use this value to indicate a server engine.

Note:

Define the following Engine and Admin properties depending on what operational mode an engine is using.

- Define all Engine and Admin properties when `pa.operational.mode` is set to `STANDALONE`.
- Define only the Admin properties when using `CLUSTERED_CONSOLE` or `CLUSTERED_CONSOLE_REPLICA` mode.
- Define only the Engine properties when using `CLUSTERED_ENGINE` mode.

Admin properties

For more information about administrative properties, see the following guides.

- *Installing and Uninstalling PingAccess* on page 50 – Discusses how some properties are configured during installation.
- *PingAccess User Interface Reference Guide* on page 227 – Discusses how some properties impact administrative use
- *Managing Federal Information Processing Standards (FIPS) mode* on page 149 – Discusses how some SSL properties are overridden in FIPS mode.

admin.port

Defines the TCP port on which the PingAccess administrative console runs. The default value is `9000`.

admin.bindAddress

Defines the IP address that `admin.port` will bind to. This is typically required on multihomed servers having multiple IP addresses. The default value of `0.0.0.0` means that the port will bind to all of the server's IP addresses.

admin.ssl.protocols

Defines the protocols for use with administrative HTTPS ports. The default value is `${tls.default.protocols}`, which uses the protocols specified by the `tls.default.protocols` parameter.

admin.ssl.ciphers

Defines the type of cryptographic ciphers available for use with administrative HTTPS ports. The default value is `${tls.default.cipherSuites}`, which uses the ciphers specified by the `tls.default.cipherSuites` parameter.

admin.acceptors

Defines the number of admin acceptor threads used to establish connections. The default value is `1`.

admin.backlog

Defines the maximum queue length for incoming admin connection indications. The default value is `512`.

admin.httptransport.coreThreadPoolSize

Defines the number of threads to keep in the admin transport pool, even if they are idle. The default value is `5`.

admin.httptransport.ioThreads

Defines the number of I/O threads for the admin host. The default value is `0`, which indicates that PingAccess should automatically calculate the appropriate number of I/O threads for the host.

admin.httptransport.maxThreadPoolSize

Defines the maximum number of threads for the admin transport pool. The default value is `-1`, which denotes no limit.

admin.httptransport.socketTimeout

Defines, in milliseconds, the admin socket timeout. The default value is `30000`.

admin.auth

Overrides the administrator authentication method. For example, if single sign-on (SSO) authentication is enabled and is somehow misconfigured, this property can be used to bypass the database configuration and force the use of Basic Authentication. The default value is `default`. A value of `native` overrides the administrator authentication method, meaning that only the local administrator credentials can be used to access the PingAccess console.

admin.reuseAddress

When enabled, allows a process to bind to a port which remains in a TIME_WAIT state for the admin transport. The default value is `true`.

admin.max.request.bodylength

Defines, in megabytes, the maximum body length for a request to the administrative API endpoint. The default value is `15`.

admin.ui.max.sessions

Defines the maximum number of sessions for the admin UI when admin single logout (SLO) is not enabled. The default value is `100`.

admin.export.encryption.mode

Specifies how sensitive data should be encrypted on export. The default value is `MASTER_KEY`, which uses the system default master key for encryption. The `PORTABLE_INSECURE` value uses a randomly generated key for each export and includes the key in the export data. This method allows the exported data to be imported anywhere, including another cluster with a different master key, but since it includes the key it can present a significant security risk.

admin.startup.config.import.failfast

Defines the behavior when attempting to import a configuration file on startup. A value of `true` stops at the first failure, while a value of `false` continues and notes all errors. The default value is `false`.

Token provider communication settings

For more information about token providers, see the token providers section of the *PingAccess User Interface Reference Guide* on page 227.

pa.default.availability.ondemand.maxRetries

Defines the maximum number of retries before marking the target system down. The default value is `2`.

pa.default.availability.ondemand.connectTimeout

Defines, in milliseconds, the amount of time to wait before trying to connect to the remote host. The default value is `10000`.

pa.default.availability.ondemand.retryDelay

Defines, in milliseconds, the amount of time to wait after a timeout before retrying the host. The default value is `250`.

pa.default.availability.ondemand.failedRetryTimeout

Defines, in seconds, the amount of time to wait before retrying a failed host. The default value is `60`.

pa.default.availability.ondemand.pooledConnectionTimeout

Defines, in milliseconds, the amount of time to wait before timing out the request for a pooled connection to the target site. The default value is `-1`, which indicates no timeout.

pa.default.availability.ondemand.readTimeout

Defines, in milliseconds, the amount of time to wait before timing out the read response for a target site. The default value is `-1`, which indicates no timeout.

Cluster configuration settings

For more information about cluster configuration, see the *Clustering Reference Guide*.

clusterconfig.enabled

When enabled, uses the cluster configuration port for cluster replication. When disabled, the admin port is used for cluster configuration replication. The default value is `true`.

> **Note:**
>
> This parameter is set to `false` by the PingAccess Upgrade Utility after a PingAccess cluster is upgraded from a version earlier than 4.0.

clusterconfig.port

Defines the optional port used for cluster configuration. The default value is `9090`.

clusterconfig.bindAddress

Defines the optional address used for cluster configuration. The default value is `0.0.0.0`.

clusterconfig.acceptors

Defines the number of cluster configuration acceptor threads used to establish connections. The default value is `1`.

clusterconfig.backlog

Defines the maximum queue length for incoming cluster configuration connection indications. The default value is `512`.

clusterconfig.reuseAddress

When enabled, allows a process to bind to a port, which remains in a TIME_WAIT state for the cluster configuration transport. The default value is `true`.

clusterconfig.httptransport.socketTimeout

Defines, in milliseconds, the cluster configuration socket timeout. The default value is `30000`.

clusterconfig.httptransport.ioThreads

Defines the number of I/O threads for the cluster configuration host. The default value is `0`, which indicates that PingAccess should automatically calculate the appropriate number of I/O threads for the host.

clusterconfig.httptransport.coreThreadPoolSize

Defines the number of threads to keep in the cluster configuration transport pool, even if they are idle. The default value is `5`.

clusterconfig.httptransport.maxThreadPoolSize

Defines the maximum number of threads for the cluster configuration transport pool. The default value is `-1`, which denotes no limit.

engine.admin.configuration.audience

Defines the audience used for cluster authentication. This property must be set to the same value on all nodes in a PingAccess cluster. The default value is `PingAccessAdminServer`.

engine.polling.initialdelay

Defines, in milliseconds, how long after the engine starts up before it begins to poll the administrative console for configuration information. The default value is `500`.

engine.polling.delay

Defines, in milliseconds, how long after the prior query to the administrative console that the engine begins a new query for configuration information. The default value is `2000`.

admin.polling.initialdelay

Defines, in milliseconds, how long after the replica administrative node starts up before it begins to poll the administrative console for configuration information. The default value is `500`.

admin.polling.delay

Defines, in milliseconds, how long after the prior query to the administrative console that the replica administrative node begin a new query for configuration information. The default value is `2000`.

pa.config.replication.readTimeout

Defines, in milliseconds, the amount of time to wait before timing out the read response for the administrative node. The default value is `30000`.

pa.config.replication.maxRetries

Defines the maximum number of retries before marking the administrative node system down. The default value is `5`.

pa.config.replication.connectTimeout

Defines, in milliseconds, the amount of time to wait before trying to connect to the administrative node. The default value is `5000`.

pa.config.replication.retryDelay

Defines, in milliseconds, the amount of time to wait after a timeout before retrying the administrative node. The default value is `2000`.

pa.config.replication.failedRetryTimeout

Defines, in seconds, the amount of time to wait before retrying a failed connection to the administrative node. The default value is `-1`, which indicates no timeout.

pa.config.replication.pooledConnectionTimeout

Defines, in milliseconds, the amount of time to wait before timing out the request for a pooled connection to the administrative node. The default value is `-1`, which indicates no timeout.

Engine properties

For more information about engine settings, see the following guides.

- *Clustering Reference Guide* – Discusses cluster configuration.
- *Managing Federal Information Processing Standards (FIPS) mode* on page 149 – Discusses how some SSL properties are overridden in FIPS mode.

.

engine.http.bindAddress

Defines the address for an engine in a clustered environment. The default value is `0.0.0.0`.

engine.http.acceptors

Defines the number of engine acceptor threads used to establish connections. The default value is `1`.

engine.http.backlog

Defines the maximum queue length for incoming engine connection indications. The default value is `512`.

engine.http.reuseAddress

When enabled, allows a process to bind to a port which remains in a TIME_WAIT state for the engine transport. The default value is `true`.

engine.http.enabled

Defines whether a STANDALONE or CLUSTERED_ENGINE node listens for requests on the ports defined by the Engine Listeners. The default value is `true`.

engine.httptransport.coreThreadPoolSize

Defines the number of threads to keep in the engine transport pool, even if they are idle. The default value is `5`.

engine.httptransport.maxThreadPoolSize

Defines the maximum number of threads for the engine transport pool. The default value is `-1`, which denotes no limit.

engine.httptransport.socketTimeout

Defines, in milliseconds, the engine socket timeout. The default value is `30000`.

engine.httptransport.ioThreads

Defines the number of I/O threads for the engine host. The default value is `0` which denotes that PingAccess should automatically calculate the appropriate number of I/O threads for the host.

engine.websocket.maxConnections

Sets the maximum number of allowed web socket connections. The default value is `-1`, which denotes no limit.

engine.ssl.protocols

Defines the protocols used with engine HTTPS ports. The default value is `TLSv1, TLSv1.1, TLSv1.2, TLSv1.3`.

engine.ssl.ciphers

Defines the type of cryptographic ciphers available for use with engine HTTPS ports. The default value is `${tls.default.cipherSuites}`, which uses the ciphers specified by the `tls.default.cipherSuites` parameter.

client.ioThreads

Defines the number of threads for client connections to backend sites. The default value is `0`, which denotes no limit.

pa.default.contentRewrite.buffer.min

Defines, in bytes, the minimum buffer size used when using a rewrite content rule. The default value is `1024`.

pa.default.contentRewrite.buffer.default

Defines, in bytes, the default buffer size when using a rewrite content rule to do a search and replace of content. The default value is `2048`.

pa.default.limitRequestLine

Defines the maximum number of bytes to read from the request line. The default value is `8192`.

pa.default.maxHeaderCount

Defines the maximum number of headers to read from a request. The default value is `100`.

pa.default.maxHttpHeaderSize

Defines the maximum number of bytes to read when reading headers. The default value is `8192`.

pa.default.maxRequestBodySize

Defines the maximum number of bytes to read from a request body. The default value is `204800`.

pa.default.maxConnectionsPerSite

Defines the maximum number of connections PingAccess will open to the PingFederate Admin or Engine. The default value is `-1`, which denotes no limit.

pa.default.session.cookie.attributes.httponly

Defines the default setting for the **HTTP-Only Cookie** setting for newly-created web sessions. The default value is `true`.

pa.default.session.cookie.attributes.secure

Defines the default setting for the **Secure Cookie** setting for newly-created web sessions. The default value is `true`.

pa.default.session.cookie.size.threshold

Defines, in bytes, the default maximum session cookie size. The default value is `4093`.

pa.websession.cookie.sameSiteExcludedUserAgentPatterns

A comma-separated list of regex that specifies whether an end-user browser should have `SameSite=None` applied to cookies issued to it. If the user-agent header from a request matches any of the values in the list, any PingAccess-issued cookie is set with no SameSite attribute if `SameSite=None` would otherwise have been applied. The default value is:

```
^.*\\(iP.+; CPU .*OS 12[_\\d]*.*\\) AppleWebKit\\/.*$,\
^.*Macintosh;.*Mac OS X 10_14.*Version.*Safari.*$,\
^.*(Chromium|Chrome)\\/(5[1-9]|6[0-6])\\.(\\d+)(?:\\.(\\d+)|)(?:\
\.(\\d+)|).*$,\
^.*UCBrowser\\/[0-9][0-1]?.(\\d+)\\.(\\d+)[\\.\\d]*.*$,\
^.*UCBrowser\\/12.[0-9][0-2]?.(\\d+)[\\.\\d]*.*$,\
^.*UCBrowser\\/12.13.[0-2][\\.\\d]*.*$
```

pa.uri.strict

When enabled, this setting requires the raw input URI be in strict compliance with the URI spec implemented by `java.net.URI` when generating URIs. The default value is `false`.

pa.uri.canonicalize

When enabled, PingAccess normalizes empty and dot path segments that contain URL-encoded forward slashes (`/`, encoded as `%2f`) or periods (encoded as `%2e`). When this setting has a value of `false`, PingAccess doesn't normalize empty and dot path segments that contain URL-encoded forward slashes or periods. The default value is `true`.

Agent properties

For more information about agents, see the following guides.

- *PingAccess User Interface Reference Guide* on page 227 – Discusses agent settings

- *PingAccess Agent for Apache (RHEL)* on page 357 – Discusses agent installation and management
- *PingAccess Agent for Apache (SLES)* on page 380 – Discusses agent installation and management
- *PingAccess Agent for Apache (Windows)* on page 393 – Discusses agent installation and management
- *PingAccess Agent for IIS* on page 403 – Discusses agent installation and management
- *PingAccess Agent for NGINX* on page 422 – Discusses agent installation and management
- *Managing Federal Information Processing Standards (FIPS) mode* on page 149 – Discusses how some SSL properties are overridden in FIPS mode.

agent.http.port

Defines the TCP port on which the engine listens for agent requests. The default value is `3030`.

agent.http.bindAddress

Defines the address from which an engine listens for agent requests. The default value is `0.0.0.0`.

agent.http.acceptors

Defines the number of admin acceptor threads used to establish agent connections. The default value is `1`.

agent.http.secure

Defines whether the engine is using HTTPS for agent requests. The default value is `true`.

agent.http.backlog

Defines the maximum queue length for incoming admin connection indications. The default value is `512`.

agent.http.enabled

Defines whether a STANDALONE or CLUSTERED_ENGINE node listens for agent requests on the port defined by the `agent.http.port` setting. The default value is `true`.

agent.http.reuseAddress

When enabled, allows a process to bind to a port which remains in a TIME_WAIT state for the agent transport. The default value is `true`.

agent.ssl.protocols

Defines the protocols used for communication with agent HTTPS ports. The default value is `${tls.default.protocols}`, which uses the protocols specified by the `tls.default.protocols` parameter.

agent.ssl.ciphers

Defines the type of cryptographic ciphers available for use with agent HTTPS ports. The default value is `${tls.default.cipherSuites}`, which uses the ciphers specified by the `tls.default.cipherSuites` parameter.

agent.httptransport.coreThreadPoolSize

Defines the number of threads to keep in the agent transport pool, even if they are idle. The default value is `5`.

agent.httptransport.maxThreadPoolSize

Defines the maximum number of threads for the agent transport pool. The default value is `-1`, which denotes no limit.

agent.httptransport.socketTimeout

Defines, in milliseconds, the agent socket timeout. The default value is `30000`.

agent.httptransport.ioThreads

Defines the number of I/O threads for the agent host. The default value is `0`, which denotes that PingAccess should automatically calculate the appropriate number of I/O threads for the host.

agent.authz.header.required

Defines whether PingAccess server should authenticate agent requests using agent name and shared secret in the vnd-pi-authz header. The default value is `true`. Setting this to `false` is useful for POCs and/or debugging.

agent.default.token.cache.ttl

Defines, in seconds, the time to live for cached agent tokens. The default value is `60`.

Sideband properties

For more information about sideband clients, see the following guides.

- *Sideband Clients* on page 257 – Discusses how to configure a sideband client in the user interface.
- *Managing Federal Information Processing Standards (FIPS) mode* on page 149 – Discusses how some SSL properties are overridden in FIPS mode.

sideband.http.port

Defines the TCP port on which the engine listens for sideband requests. The default value is `3030`.

sideband.http.bindAddress

Defines the address from which an engine listens for sideband requests. The default value is `0.0.0.0`.

sideband.http.acceptors

Defines the number of admin acceptor threads used to establish sideband connections. The default value is `1`.

sideband.http.secure

Defines whether the engine is using HTTPS for sideband requests. The default value is `true`.

sideband.http.backlog

Defines the maximum queue length for incoming admin connection indications. The default value is `512`.

sideband.http.enabled

Defines whether a STANDALONE or CLUSTERED_ENGINE node listens for sideband requests on the port defined by the `agent.http.port` setting. The default value is `true`.

sideband.http.reuseAddress

When enabled, allows a process to bind to a port which remains in a TIME_WAIT state for the agent transport. The default value is `true`.

sideband.ssl.protocols

Defines the protocols used for communication with sideband HTTPS ports. The default value is `${tls.default.protocols}`, which uses the protocols specified by the `tls.default.protocols` parameter.

sideband.ssl.ciphers

Defines the type of cryptographic ciphers available for use with sideband HTTPS ports. The default value is `${tls.default.cipherSuites}`, which uses the ciphers specified by the `tls.default.cipherSuites` parameter.

sideband.httptransport.coreThreadPoolSize

Defines the number of threads to keep in the sideband transport pool, even if they are idle. The default value is `5`.

sideband.httptransport.maxThreadPoolSize

Defines the maximum number of threads for the sideband transport pool. The default value is `-1`, which denotes no limit.

sideband.httptransport.socketTimeout

Defines, in milliseconds, the sideband socket timeout. The default value is `30000`.

sideband.httptransport.ioThreads

Defines the number of I/O threads for the sideband host. The default value is `0`, which denotes that PingAccess should automatically calculate the appropriate number of I/O threads for the host.

URL filtering settings

For more information about URL filtering, see Adding rewrite URL rules in the *PingAccess User Interface Reference Guide* on page 227.

pa.interceptors.relativepath.strict

When this property is set to `true`, the incoming URL is matched with the whitelist pattern defined in `pa.interceptors.relativepath.decode.regex`. All other request URLs are rejected. The default value is `false`.

pa.interceptors.relativepath.decode.count

Number of times the URL is decoded to check for path traversal characters. The default value is `3`.

pa.interceptors.relativepath.decode.regex

Defines the regular expression to use when checking for a valid path in an incoming request. The default value is:

```
[\\p{Po}\\p{N}\\p{Z}\\p{L}\\p{M}\\p{Zs}\\./_\\-\\\\~()\\{\\}\\[\
\]]*
```

Note:

This value is double-escaped as required by the `java.util.regex.Pattern` Java class.

Monitoring

For more information about monitoring, see the *PingAccess Monitoring Guide* on page 528.

pa.mbean.site.connection.pool.enable

When set to `true`, enables Java Management Extensions (JMX) read-only access to backend connection pools. This can be useful when troubleshooting latency issues because it provides information about requests that are waiting for a connection to targets in a site when `maxConnections` is not unlimited. The default value is `false`.

enable.detailed.heartbeat.response

When enabled, this setting enables a customizable heartbeat response to be returned. When disabled, the heartbeat endpoint returns a `200 OK` response. The default value is `false`.

pa.statistics.window.seconds

If the `enable.detailed.heartbeat.response` parameter is set to `true`, this parameter sets the number of seconds back to collect response statistics. A value less than 1 disables collection. The default value is `0`.

TLS/SSL

For more information about the use of TLS/SSL, see the following guides.

- *PingAccess Hardening Guide* – Discusses the use of these settings for security
- *Managing Federal Information Processing Standards (FIPS) mode* on page 149 – Discusses how some TLS properties are overridden in FIPS mode.

tls.default.protocols

Defines the default protocols used for HTTPS communication. The default value is `TLSv1.1, TLSv1.2, TLSv1.3`.

tls.default.cipherSuites

Defines the default set of ciphers used for HTTPS communication. The default value is:

```
TLS_CHACHA20_POLY1305_SHA256,\
TLS_AES_256_GCM_SHA384,\
TLS_AES_128_GCM_SHA256,\
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,\
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,\
TLS_RSA_WITH_AES_128_GCM_SHA256,\
TLS_RSA_WITH_AES_128_CBC_SHA256,\
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,\
TLS_EMPTY_RENEGOTIATION_INFO_SCSV
```

Note:

Legacy browsers might require the addition of SHA1-based ciphers to negotiate a cipher suite with the server. In this case, add the following ciphers to the run.properties file and restart PingAccess:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA

clusterconfig.ssl.protocols

Defines the protocols used for communication with HTTPS ports in a clustered configuration. The default value is `${tls.default.protocols}`, which uses the protocols specified by the `tls.default.protocols` parameter.

clusterconfig.ssl.ciphers

Defines the type of cryptographic ciphers available for use with HTTPS ports in a clustered configuration. The default value is `${tls.default.cipherSuites}`, which uses the ciphers specified by the `tls.default.cipherSuites` parameter.

site.ssl.protocols

Defines the protocols used for communication with Site HTTPS ports. There is no default value. When not specified, the protocols defined in the Java Development Kit (JDK) are used.

site.ssl.ciphers

Defines the type of cryptographic ciphers available for use with Site HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

pf.ssl.protocols

Defines the protocols used for communication with PingFederate HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

pf.ssl.ciphers

Defines the type of cryptographic ciphers available for use with PingFederate HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

provider.ssl.protocols

Defines the protocols used for communication with Provider HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

provider.ssl.ciphers

Defines the type of cryptographic ciphers available for use with Provider HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

as.ssl.protocols

Defines the protocols used for communication with authorization server HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

as.ssl.ciphers

Defines the type of cryptographic ciphers available for use with authorization server HTTPS ports. There is no default value. When not specified, the protocols defined in the JDK are used.

p14c.ssl.protocols

Defines the protocols used for communication with PingOne. There is no default value. When not specified, the protocols defined in the JDK are used.

p14c.ssl.ciphers

Defines the type of cryptographic ciphers available for use with PingOne. There is no default value. When not specified, the protocols defined in the JDK are used.

thirdpartyservice.ssl.protocols

Defines the protocols used for communication with third-party services. There is no default value. When not specified, the protocols defined in the JDK are used.

thirdpartyservice.ssl.ciphers

Defines the type of cryptographic ciphers available for use with third-party services. There is no default value. When not specified, the protocols defined in the JDK are used.

POST preservation properties

For more information about POST preservation, see the following documents.

- *User-facing page customization reference* on page 144 – Discusses the use of POST preservation in system templates
- *User-facing page localization reference* on page 148 – Discusses the use of POST preservation in system templates
- *PingAccess User Interface Reference Guide* on page 227 – Discusses the use of POST preservation in Web Sessions

pa.oidc.post.preservation.encrypt

When enabled, POST data preserved through a redirection to PingFederate for authentication is encrypted on the client to be used after the authentication is successful. The default value is `false`.

pa.oidc.post.preservation.maxRequestBodySize

Defines, in bytes, the maximum size of the post body for POST preservation. The default value is `8192`.

pa.oidc.post.preservation.paramsAttributeName

Used to store the encoded or encrypted POST payload in the browser session storage during POST preservation. The default value is `postParams`.

Configuration database and keystore settings

For more information about configuration database and keystore settings, see the following documents:

- *Installing and Uninstalling PingAccess* on page 50 – Discusses the initial database settings
- *Performance Tuning Reference Guide* – Discusses adjusting PingAccess settings for specific environments
- *Apache Derby documentation* – Discusses how the native Apache Derby properties are used.

derby.language.statementCacheSize

Defines the number of statements that are stored in memory. The default value is `500`.

derby.storage.pageCacheSize

Defines the number of pages cached in memory. The default value is `1000`.

pa.keystore.pw

Defines the password for the `$JAVA_HOME/lib/security/cacerts` keystore. The value is encrypted.

PingFederate administration integration properties

For more information about using PingAccess with PingFederate, see the following documents:

- *Configure PingFederate as the token provider for PingAccess* on page 507
- *PingAccess User Interface Reference Guide* on page 227

pf.api.maxRetries

Defines the maximum number of retries PingAccess attempts to make to the PingFederate server before declaring the server unavailable. The default value is `0`.

pf.api.socketTimeout

Defines, in milliseconds, the socket timeout for the PingFederate API endpoint. The default value is `5000`.

pf.api.maxConnections

Defines the maximum number of connections PingAccess will establish to the PingFederate API endpoint. The default value is `-1`, which means there is no limit.

pf.api.keepAliveTimeout

Defines, in milliseconds, the keep alive timeout for the PingFederate API. The default value is `30000`.

pf.api.readTimeout

Defines, in milliseconds, how long the API will wait for responses from PingFederate when making calls to the PingFederate Admin API. The default value is `-1`, which means there is no limit.

Administrative console settings

For more information about PingAccess administration, see the *PingAccess User Interface Reference Guide* on page 227.

pa.backup.filesToKeep

Defines the number of backup files to preserve when the Administrator authenticates to PingAccess. The default value is `25`. A value of `0` disables the creation of backup files.

Note:

Disabling the creation of backup files can speed up the sign-on process in large environments. If you disable the creation of backup files, use the administrative API backup endpoint to create regular backups.

pa.admin.user.password.regex

Defines the regex that controls password complexity for the Administration Console. The default value is

```
((?=.*\\d)(?=.*[a-z])(?=.*[A-Z]).{8,20})
```

pa.admin.user.password.error.message

Defines the message returned when password complexity is not satisfied. The default value is `Password must be at least 8 characters in length, contain one upper-case letter, one lower-case letter and one digit.`.

pa.admin.test.connections

A boolean property that allows the PingAccess admin UI to make HTTP calls to validate that it can reach PingFederate and sites when the user configures them. The default value is `true`.

account.locking.max.consecutive.failures

Defines the maximum number of failed login attempts before locking the account when using basic authentication in the administrative UI or administrative REST APIs. The default value is `3`.

account.locking.max.lockout.period

Defines, in minutes, the amount of time to lock an account out from the administrative interfaces after exceeding the `account.locking.max.consecutive.failures`. The default value is `1`.

EHCache configuration properties

For more information about EHCache configuration, see the *Clustering Reference Guide*.

pa.ehcache.PingFederateReferenceTokenCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for OAuth tokens. The default value is `10000`.

pa.ehcache.PingFederateReferenceTokenCache.timeToIdleSeconds

Defines, in seconds, the time an entry in the OAuth token cache can be idle before it is expired. The default value is `0`.

pa.ehcache.PingFederateReferenceTokenCache.timeToLiveSeconds

Defines, in seconds, the maximum time an entry can be in the OAuth token cache. The default value is `0`.

pa.ehcache.ServiceTokenCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for token mediation. The default value is `10000`.

pa.ehcache.ServiceTokenCache.timeToIdleSeconds

Defines, in seconds, the time an entry in the token mediation cache can be idle before it is expired. The default value is `1800`.

pa.ehcache.ServiceTokenCache.timeToLiveSeconds

Defines, in seconds, the maximum time an entry can be in the token mediation cache. The default value is `14400`.

pa.ehcache.PATokenValidationCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for decryption of signed or encrypted PingAccess tokens. The default value is `10000`.

pa.ehcache.PATokenValidationCache.timeToIdleSeconds

Defines, in seconds, the time an entry in the token validation cache can be idle before it is expired. The default value is `120`.

pa.ehcache.PATokenValidationCache.timeToLiveSeconds

Defines, in seconds, the maximum time an entry can be in the token validation cache. The default value is `300`.

pa.ehcache.PFSessionValidationCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for the session validation cache. The default value is `10000`.

pa.ehcache.PFSessionValidationCache.timeToIdleSeconds

Defines, in seconds, the time an entry in the session validation cache can be idle before it is expired. The default value is `120`.

pa.ehcache.PFSessionValidationCache.timeToLiveSeconds

Defines, in seconds, the maximum time an entry can be in the session validation cache. The default value is `300`.

pa.ehcache.PAWamUserAttributesCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for the PingAccess Web Access Management (WAM) user attribute cache. The default value is `10000`.

pa.ehcache.PAWamUserAttributesCache.timeToIdleSeconds

Defines, in seconds, the time an entry in the PA WAM user attribute cache can be idle before it is expired. The default value is `120` seconds.

pa.ehcache.PAWamUserAttributesCache.timeToLiveSeconds

Defines, in seconds, the maximum time an entry can be in the PA WAM user attribute cache. The default value is `300` seconds.

pa.ehcache.AuthTokenCache.maxEntriesLocalHeap

Defines the maximum size of the JSON web token (JWT) identity mapping token cache used when sending tokens to a protected site. The default value is `10000`.

pa.ehcache.SessionStateCache.maxEntriesLocalHeap

Defines the maximum size of the identity attribute entry cache when the user's attributes are stored on the server rather than as a cookie. The default value is `10000`.

pa.ehcache.AzureGroupNameCache.maxEntriesLocalHeap

Defines the maximum number of entries in the local heap for the Azure group name cache. The default value is `10000`.

Security headers properties

For more information about security headers, see the following documents.

- *PingAccess API endpoints* on page 152 – Discusses the behavior of the Admin API
- *PingAccess User Interface Reference Guide* on page 227 – Discusses Admin UI settings
- *PingAccess Hardening Guide* – Discusses measures to ensure security

admin.headers

Additional headers added to responses from the PingAccess administrative console and the administrator API interface. Header values are defined using the `admin.header` prefix. The default value is:

```
X-Frame-Options,X-XSS-Protection,X-Content-Type-Options,Strict-
Transport-Security,Content-Security-Policy
```

admin.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser when an admin is interacting with the Admin UI. The default value is `DENY`.

admin.header.X-XSS-Protection

Sets the parameters for the X-XSS-Protection HTTP response header sent to the browser when an admin is interacting with the Admin UI. The default value is `1; mode=block`.

admin.header.X-Content-Type-Options

Sets the parameters for the X-Content-Type-Options response header sent to the browser when an admin is interacting with the Admin UI. The default value is `nosniff`.

admin.header.Content-Security-Policy

Sets the parameters for the `content-security-policy` response header sent by PingAccess in response to API calls. The default value is:

```
default-src 'self'; style-src 'self' 'unsafe-inline'; script-src
 'self' 'unsafe-inline'; font-src 'self' data:;
```

admin.header.Strict-Transport-Security

Sets the parameters for the Strict-Transport-Security response header sent to the browser when an administrator is interacting with the Admin UI. This parameter is commented out by default, and should be enabled only if the admin and engine use different host names. The default value is `max-age=31536000; includeSubDomains`.

agent.assets.headers

Additional headers added to responses from PingAccess agents. Header values are defined using the `agent.assets.header` prefix. The default value is `X-Frame-Options`.

agent.assets.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the agent when responding to a request for an asset used by a PingAccess template. The default value is `DENY`.

agent.error.headers

Additional headers added to error responses from PingAccess Agents. Header values are defined using the `agent.error.header` prefix. The default value is `X-Frame-Options`.

agent.error.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the agent when responding with a PingAccess error template. The default value is `DENY`.

engine.assets.headers

Additional headers added to responses from the PingAccess Engine. Header values are defined using the `engine.assets.header` prefix. The default value is `X-Frame-Options`.

engine.assets.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the engine when responding to a request for an asset used by a PingAccess template. The default value is `DENY`.

engine.error.headers

Additional headers added to error responses from the PingAccess Engine. Header values are defined using the `engine.error.header` prefix. The default value is `X-Frame-Options`.

engine.error.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the engine when responding with a PingAccess error template. The default value is `DENY`.

sideband.assets.headers

Additional headers added to responses from PingAccess sideband clients. Header values are defined using the `sideband.assets.header` prefix. The default value is `X-Frame-Options`.

sideband.assets.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the sideband client when responding to a request for an asset used by a PingAccess template. The default value is `DENY`.

sideband.error.headers

Additional headers added to error responses from PingAccess sideband clients. Header values are defined using the `sideband.error.header` prefix. The default value is `X-Frame-Options`.

sideband.error.header.X-Frame-Options

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser using the sideband client when responding with a PingAccess error template. The default value is `DENY`.

pf.redirect.headers

Additional headers added to the redirection response that sends the client to PingFederate for authentication. Header values are defined using the `pf.redirect.header` prefix. The default value is `X-Frame-Options`.

pf.redirect.header.X-Frame-Options

Sets the parameters for the X-Frame-Options value that is sent when the user is redirected to PingFederate to authenticate. The default value is `DENY`.

rule.error.headers

Additional headers added to responses that result from policy rule results. Header values are defined using the `rule.error.header` prefix. There is no default value.

oauth.error.headers

Additional headers added to responses that result from requests made to a protected API application that lack a valid OAuth Bearer token. Define header values using the `oauth.error.header` prefix. The default value is `Content-Security-Policy`.

oauth.error.header.Content-Security-Policy (commented out by default)

Sets the parameters for the `Content-Security-Policy` HTTP response header sent to the browser when PingAccess receives a request made to a protected API application that doesn't contain a valid OAuth Bearer token. The default value is:

```
default-src 'self'; style-src 'self'; \ script-src 'self'; font-
src 'self' data:; object-src 'none';
```

Localization settings

For more information about localization, see the following documents. Customize and Localize PingAccess.

- *User-facing page customization reference* on page 144
- *User-facing page localization reference* on page 148

pa.localization.resource.bundle.cache.enable

When set to `false`, allows language files in /conf/localization to be added or modified. When `true`, enables caching of language files and properties. The default value is `true`.

pa.localization.missing.message.placeholder

Defines the message used when an error message is unresolvable. There is no default value.

PingAccess deployment guide

Use the deployment guide to decide how PingAccess fits into your existing network, such as determining the deployment architecture required for your use case and whether you require high-availability options.

This section provides information to help you make the right decisions for your environment in PingAccess.

Use cases and deployment architecture

Depending on your needs and infrastructure capabilities, there are many options for deploying PingAccess in your network environment.

You can design a deployment that supports mobile and API access management, web access management, or auditing and proxying. For each of these environments, you can choose a stand-alone deployment for proof of concept or deploy multiple PingAccess servers in a cluster configuration for high availability, server redundancy, and failover recovery.

You have a choice between using PingAccess as a gateway or using a PingAccess agent plugin on the web server. In a gateway deployment, all client requests first go through PingAccess and are checked for authorization before they are forwarded to the target site. In an agent deployment, client requests go directly to the web server serving up the target site, where they are intercepted by the agent plugin and checked for authorization before they are forwarded to the target resource. The same access control checks are performed by the PingAccess policy server in both cases and only properly authorized client request are allowed to reach the target assets. The difference is that in a gateway deployment client requests are rerouted through PingAccess gateway, while in an agent deployment, they continue to be routed directly to the target site, where PingAccess agent is deployed to intercept them.

PingAccess agent makes a separate access control request to PingAccess Policy Server using the PingAccess Agent Protocol (PAAP). The agent request contains just the relevant parts of the client request so that PingAccess Policy Server can make the access control decision and respond with instructions to the agent regarding any modifications to the original client request that the agent should perform prior to forwarding the request. For example, the agent can add headers and tokens required by the target resource. Under the PingAccess policy server's control, the agent might perform a certain amount of caching of information in order to minimize the overhead of contacting the PingAccess policy server, thus minimizing response time.

In both gateway and agent deployment, the response from the target resource is processed on the way to the original client. In an agent deployment, the amount of processing is more limited than in a gateway deployment. The agent does not make another request to the policy server, so response processing is based on the initial agent response. Consequently, the agent is not able to apply the request processing rules available to the gateway.

When designing a deployment architecture, many requirements and components must be identified for a successful implementation. Proper network configuration of routers/firewalls and DNS ensure that all traffic is routed through PingAccess for the resources it is protecting and that alternative paths, such as backdoors, are not available.

The following sections provide specific use cases and deployment architecture requirements to assist with designing and implementing your PingAccess environment.

Deploy for gateway web access management

A PingAccess web access management (WAM) deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to web-based applications while integrating with existing identity management infrastructure.

With growing numbers of internal and external users, and more and more enterprise resources available online, it is important to ensure that qualified users can access only those applications to which they have permission. A WAM environment provides authentication and policy-based access management while integrating with existing infrastructure.

Deployed at the perimeter of a protected network between browsers and protected web-based applications, PingAccess Gateway performs the following actions:

- Receives inbound calls requesting access to web applications

  Web session-protected requests contain a previously-obtained PingAccess token in a cookie derived from the user's profile during an OpenID Connect (OIDC) based sign on at PingFederate.
- Evaluates application and resource-level policies and validates the tokens in conjunction with an OIDC Policy configured within PingFederate
- Acquires the appropriate target security token (site authenticators) from the PingFederate security token service (STS) or from a cache, including attributes and authorized scopes, should a web application require identity mediation
- Makes authorized requests to the sites where the web applications reside and responses are received and processed
- Relays the responses on to the browsers

The following sections describe sample proof of concept and production architectures for a WAM use case deployment:

- *WAM Gateway POC Deployment Architecture*
- *WAM Gateway Production Deployment Architecture*

Deploy for agent web access management

A PingAccess web access management (WAM) agent deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to web-based applications while integrating with existing identity management infrastructure and minimal network configuration changes.

With growing numbers of internal and external users, and more enterprise resources available online, ensure that qualified users can access only those applications to which they have permission. A WAM environment provides authentication and policy-based access management while integrating with existing infrastructure.

The PingAccess agent plugin is installed on the web server hosting the protected web-based applications and configured to communicate with PingAccess server also deployed on the network. When the agent intercepts a client request to a protected web application resource, it performs the following actions:

- Intercepts inbound requests to web applications
- Sends agent requests to the PingAccess Policy Server sending along relevant request information needed by policy server
- Receives agent responses from policy server and follows the instructions from policy server, modifies the request as specified, and allows the request to proceed to the target resource
- Intercepts responses from the application and modifies response headers as instructed in the initial agent request to policy server

- Relays responses on to the browsers

The PingAccess policy server listens for agent requests and performs the following actions:

- Evaluates application and resource-level policies and validates the tokens in conjunction with an OpenID Connect (OIDC) Policy configured within PingFederate
- Acquires the appropriate HTTP request header configuration from the associated identity mappings
- Sends an agent response with instructions on whether to allow the request and how to modify the client request headers

The following sections describe sample proof of concept and production architectures for a WAM use case deployment:

- *WAM Agent POC Deployment Architecture*
- *WAM Agent Production Deployment Architecture*

Deploy for gateway API access management

A PingAccess API access management deployment enables an organization to quickly set up an environment that provides a secure method of controlling access to APIs while integrating with existing identity management infrastructure.

Pressure from an expanding mobile device and API economy can lead developers to hastily design and expose APIs outside the network perimeter. Standardized API access management leads to a more consistent, centrally-controlled model that ensures existing infrastructure and security policies are followed, thereby safeguarding an organization's assets.

PingAccess Gateway sits at the perimeter of a protected network between mobile, in-browser, or server-based client applications and protected APIs and performs the following actions:

- Receives inbound API calls requesting protected applications

  OAuth-protected API calls contain previously-obtained access tokens retrieved from PingFederate acting as an OAuth authorization server.
- Evaluates application and resource-level policies and validates access tokens in conjunction with PingFederate
- Acquires the appropriate target site security token (site authenticators) from the PingFederate security token service (STS) or from a cache, including attributes and authorized scopes, should an API require identity mediation
- Makes authorized requests to the APIs and responses are received and processed
- Relays the responses on to the clients

The following sections describe sample proof of concept and production architectures for an API access management use case deployment:

- *API Access Management POC Deployment Architecture*
- *API Access Management Production Deployment Architecture*

Deploy for sideband API access management

A PingAccess API access management sideband deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to API-based applications while integrating with existing identity management infrastructure and minimal network configuration changes.

With growing numbers of internal and external users, and more enterprise resources available online, ensure that qualified users can access only those applications to which they have permission. An API access environment provides authentication and policy-based access management while integrating with existing infrastructure.

The PingAccess sideband plugin is installed on the API gateway serving the protected API applications and configured to communicate with PingAccess server also deployed on the network. When the API gateway intercepts a client request to a protected API resource, it performs the following actions:

- Intercepts inbound requests to API applications

- Sends requests to the PingAccess sideband API endpoint, sending along relevant request information needed by policy server
- Receives responses from policy server and follows the instructions from policy server, modifies the request as specified, and allows the request to proceed to the target resource
- Intercepts responses from the application
- Sends requests to the PingAccess sideband API endpoint, sending along relevant response information needed by the policy server.
- Applies modifications from the policy server and relays response

The PingAccess policy server listens for agent requests and performs the following actions:

- Evaluates application and resource-level policies and validates the tokens in conjunction with an OpenID Connect (OIDC) Policy configured within PingFederate
- Acquires the appropriate HTTP request header configuration from the associated identity mappings
- Sends a response with instructions on whether to allow the request and how to modify the client request headers

Deploy for auditing and proxying

A PingAccess deployment for auditing and proxying enables an organization to quickly set up an environment that provides a secure method of controlling access to backend sites.

With growing numbers of internal and external users, you need to know which users are accessing applications, where and when they are accessing them, and ensuring that they are correctly accessing only the applications to which they have permission. A standardized auditing and proxying deployment provides a centrally-controlled model that enforces existing infrastructure and security policies, safeguarding an organization’s assets.

At the perimeter of a protected network, between mobile, in-browser, or server-based client applications and backend sites, PingAccess performs the following actions:

- PingAccess receives inbound calls requesting access to protected backend sites.
- PingAccess audits the request and then makes authorized requests to the backend sites.
- PingAccess receives and processes responses and relays them to the clients.

The following sections describe sample proof of concept and production architectures for an auditing and proxying use case deployment:

- *Audit and Proxy POC Deployment Architecture*
- *Audit and Proxy Production Deployment Architecture*

Configuration by use case

Configuration steps vary depending on what type of deployment you are implementing.

For a detailed discussion of deployment considerations and best practices in designing your architecture, see the *Deployment Guide*. The following sections describe the configuration steps for the most common use cases:

- *API Access Management Gateway Deployment*
- *Web Access Management Agent Deployment*
- *Web Access Management Gateway Deployment*
- *Auditing and Proxying Gateway Deployment*

Next steps

After you complete the above configuration settings, the following steps are similar for all use cases:

- Configure sites and agents to define the target applications you want protected. Sites might need site authenticators to define the credentials the site expects for access control.
- Configure applications and resources to define the assets you want to allow clients to access.

- Create policies for the defined applications and resources to protect them.

Web Access Management Gateway deployment table
The following table describes the important configuration options for a Web Access Management (WAM) Gateway deployment.

For specific use case information, see *Deploying for Gateway Web Access Management* in the *Deployment Guide*.

Step	Description
Configure the connection to the PingFederate.	PingAccess uses PingFederate to manage web session and authentication.
Configure the OpenID Connect Relying Party Client for PingAccess.	The client must be registered with PingFederate and the client credentials configured in PingAccess to identify PingAccess when requesting authentication for users trying to access web applications.
Configure Web session details to enable protection of Web Resources.	Configures settings for secure web sessions such as timeout values, cookie parameters, and cryptographic algorithms.
Generate or Import Key Pairs and configure HTTP Listeners.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical services, and increases performance and overall system throughput.
Add trusted CA certificates.	Defines trust to certificates presented during outbound secure HTTPS connections.
Create a trusted certificate group.	Provides a trusted set of anchor certificates for use when authenticating outbound secure HTTPS connections.
Define virtual servers for protected resources.	Allows one server to share PingAccess resources without requiring all sites on the server to use the same host name. If SNI is available (Java 8), specific key pairs can be assigned to virtual hosts.

Web Access Management Agent deployment table
This table describes the important configuration options for a Web Access Management (WAM) agent deployment.

For specific use case information, see *Deploying for Agent Web Access Management*.

Deploy PingAccess agent using the following steps:

1. Install PingAccess agent on web server. For more information, see instructions in *PingAccess Agent for Apache Installation* or *PingAccess Agent for IIS Installation*, depending on your specific web server.
2. Define the agents and download agent `bootstrap.properties` file using the download field in the **Shared Secrets** field.

3. Deploy the agent `bootstrap.properties` file to agents. For more information, see *PingAccess Agent Configuration* .

The rest of PingAccess deployment is similar to *Web Access Management Gateway Deployment*.

Step	**Description**
Configure the connection to the PingFederate.	PingAccess uses PingFederate to manage web session and authentication.
Configure the OpenID Connect Relying Party Client for PingAccess.	The client must be registered with PingFederate and the client credentials configured in PingAccess to identify PingAccess when requesting authentication for users trying to access web applications.
Configure Web session details to enable protection of Web Resources.	Configures settings for secure web sessions such as timeout values, cookie parameters, and cryptographic algorithms.
Generate or Import Key Pairs and configure HTTP Listeners.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical services, and increases performance and overall system throughput.
Add trusted CA certificates.	Defines trust to certificates presented during outbound secure HTTPS connections.
Create a trusted certificate group.	Provides a trusted set of anchor certificates for use when authenticating outbound secure HTTPS connections.
Define virtual servers for protected resources.	Allows one server to share PingAccess resources without requiring all sites on the server to use the same host name. If SNI is available (Java 8), specific key pairs can be assigned to virtual hosts.

API Access Management Gateway deployment table

This deployment table describes the important configuration options for deploying an API Gateway.

For specific use case information, see *Deploying for Gateway API Access Management* in the *Deployment Guide*.

Step	**Description**
Configure the connection to the PingFederate OAuth Authorization Server.	PingAccess uses this connection and credentials to validate incoming access tokens for securing API calls.
Configure the OpenID Connect Relying Party Client for PingAccess.	The client must be registered with PingFederate and the client credentials configured in PingAccess to authenticate PingAccess when validating incoming access tokens.

Step	Description
Generate or Import Key Pairs and configure HTTP Listeners.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical services, and increases performance and overall system throughput.
Add trusted CA certificates.	Defines trust to certificates presented during outbound secure HTTPS connections.
Create a trusted certificate group.	Provides a trusted set of anchor certificates for use when authenticating outbound secure HTTPS connections.
Define virtual servers for protected applications.	Allows one server to share PingAccess Resources without requiring all sites on the server to use the same host name. If SNI is available (Java 8), specific key pairs can be assigned to virtual hosts.

Auditing and proxying Gateway deployment table
This gateway deployment table describes the important configuration options for an auditing or proxying deployment.

For specific use case information, see *Deploying for Auditing and Proxying*.

Step	Description
Generate or Import Key Pairs and configure HTTP Listeners.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical services, and increases performance and overall system throughput.
Add trusted CA certificates.	Defines trust to certificates presented during outbound secure HTTPS connections.
Create a trusted certificate group.	Provides a trusted set of anchor certificates for use when authenticating outbound secure HTTPS connections.
Define virtual servers for protected resources.	Allows one server to share PingAccess resources without requiring all sites on the server to use the same host name.

Web Access Management

PingAccess uses Web Access Management (WAM) capabilities to allow organizations to manage access rights to web-based resources.

With growing numbers of internal and external users, and more and more enterprise resources available online, ensure that qualified users can access only those resources to which they have permission.

WAM is a form of identity management that controls access to web resources, providing authentication and policy-based access management. After a user is authenticated, PingAccess applies application and resource-level policies to the request. After policy evaluation is passed, any required identity mediation between the backend site and the authenticated user is performed. The user is then granted access to the requested resource.

PingAccess provides two deployment architectures for WAM - gateway and agent. In a gateway deployment client requests are routed to PingAccess, which then forwards authorized requests to the target application. In an agent deployment, client requests are intercepted at the web server hosting the application using the PingAccess agent plugin. The agent then communicates with PingAccess policy server to validate access before allowing the request to proceed to the target application resource.

Choose between an agent or gateway deployment

Deploy PingAccess using Agents, as a Gateway (or reverse proxy), or using a combination of both. Before choosing a deployment, understand the pros and cons of each deployment scenario and determine how they impact your strategy.

Gateway

Pros:

- Fewer number of deployed components that require maintenance
- Independent of target application platform
- No impact on web or app server processing and performance
- Works with existing security token types, such as creating third party Web Access Management (WAM) tokens

Cons:

- Requires networking changes
- Requires strategy for securing direct access to backend web or app servers (network routing or service level authentication)
- Depending on the application, might require content/request/response rewriting
- Another layer that requires HA/DR planning

Agents

Pros:

- No networking or server level authentication changes required
- Tight integration with web server handling requests
- Scales with application

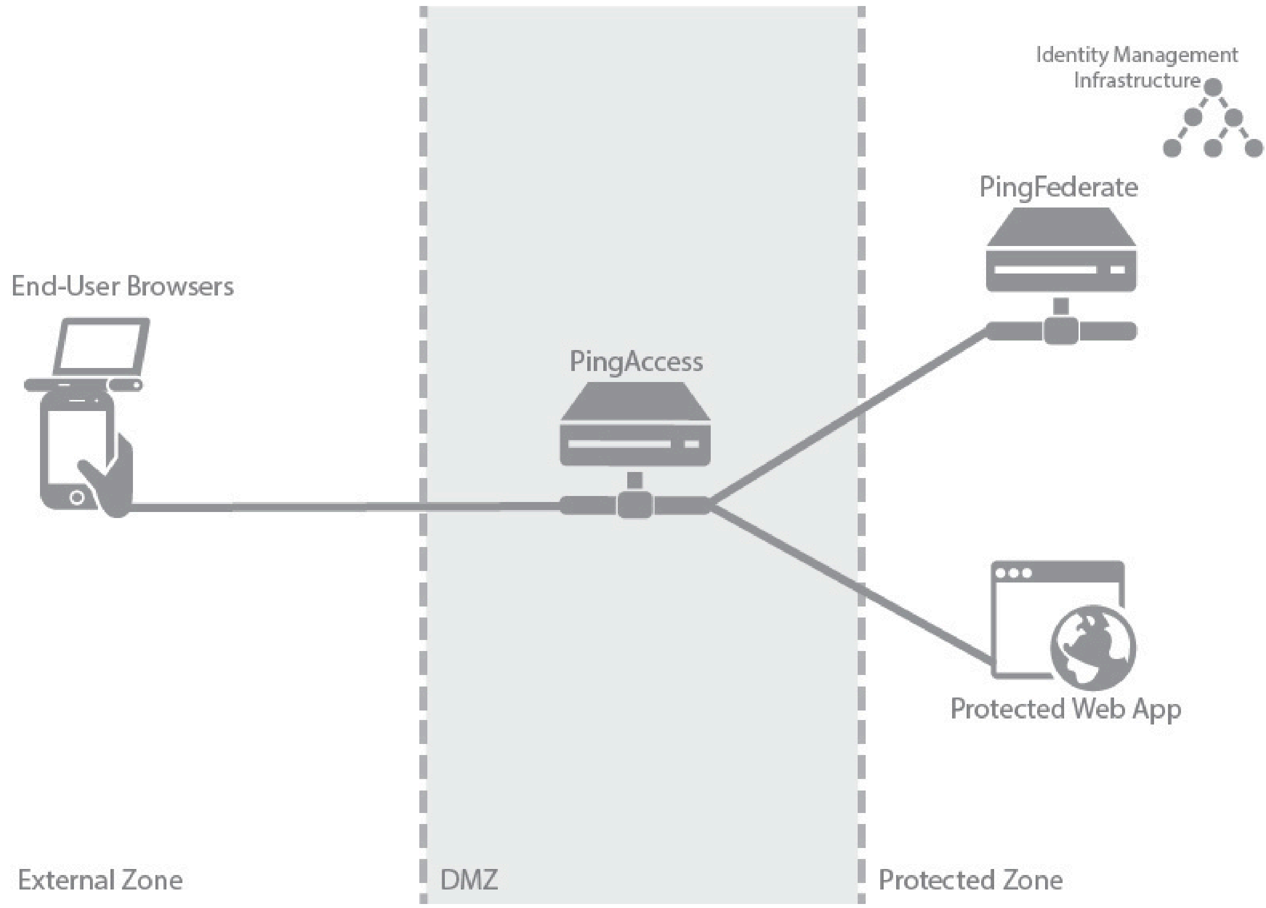
Cons:

- High cost of ownership when many agent instances are deployed, although should be upgradable or patchable independently of PingAccess policy server
- Policy evaluation is cached, and although periodically flushed or re-evaluated (for new sessions, updates to session token, etc.) , isn't as "real time" as proxy
- Tight dependency on web server version and platform

Web Access Management Gateway proof of concept deployment architecture

This proof of concept deployment environment is used to emulate a Web Access Management (WAM) gateway production environment for testing purposes.

In the test environment, PingAccess can be set up with the minimum hardware requirements. This environment example does not provide high availability and is not recommended for a production environment.



The following table describes the three zones within this proposed architecture.

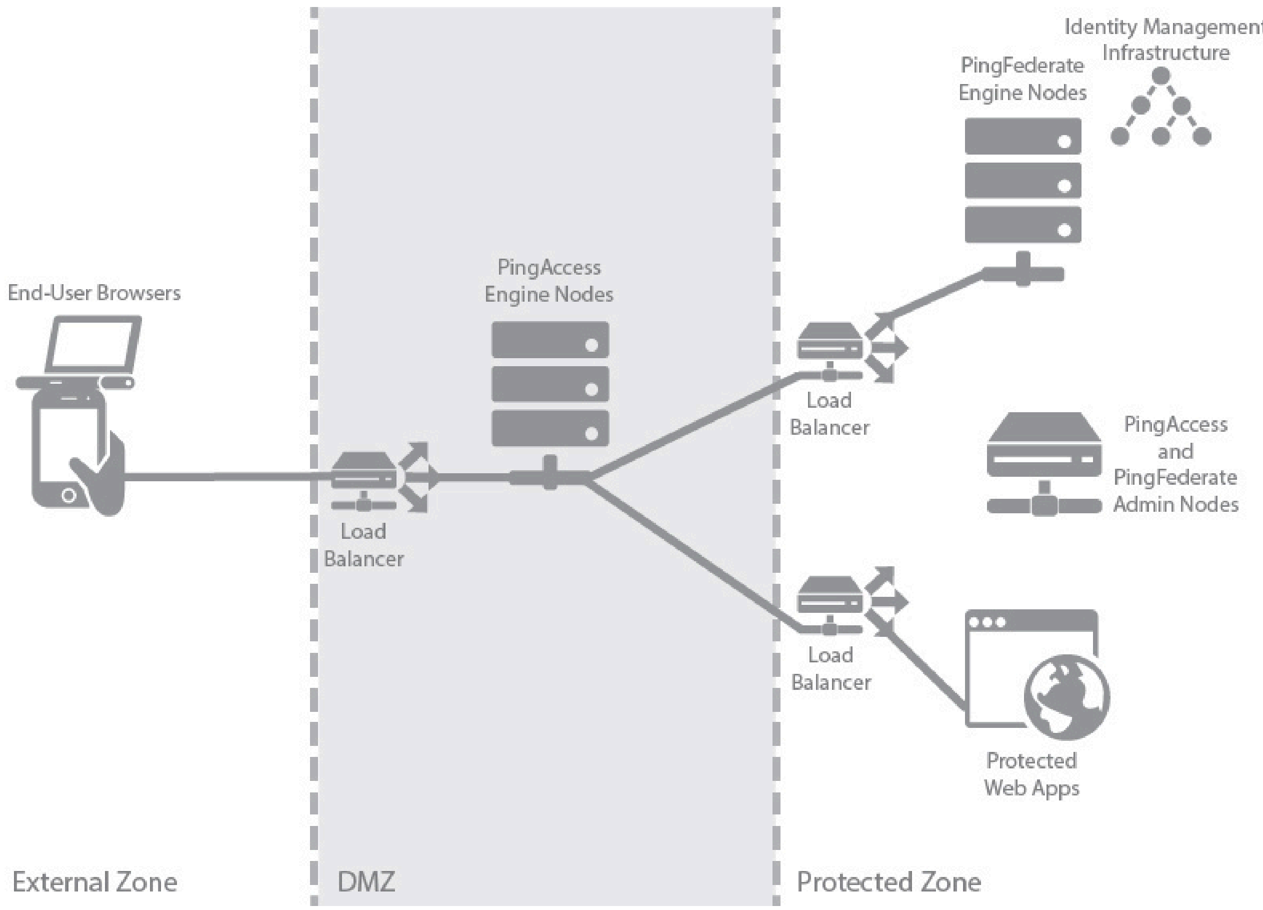
Zone	Description
External Zone	External network where incoming requests for web applications originate.
DMZ	Externally exposing segment where PingAccess is accessible to web browsers. PingAccess is a standalone instance in this environment, serving as both a runtime and an administrative port.
Protected Zone	Backend controlled zone in which sites hosting the protected web applications are located. All requests to these web applications must be designed to pass through PingAccess. PingFederate is accessible to web browsers in this zone and is a standalone instance in this environment, serving as both a runtime and an administrative port. PingFederate requires access to identity management infrastructure to authenticate users, depicted by the icon in the diagram.

Web Access Management Gateway production deployment architecture

This environment shows a Web Access Management (WAM) Gateway production architecture.

There are many considerations when deploying a production environment. For high availability and redundancy, the environment requires clustering and load-balancing. Load balancers are required as part of the networking infrastructure to achieve high availability by ensuring that requests are sent to available servers they are front-ending. Best practices in network design and security also include firewalls to ensure that only required ports and protocols are permitted across zones.

> **Note:**
>
> PingAccess provides high availability and basic load balancing for the protected web apps in the protected zone. For more information, see the availability profiles and load balancing strategies documentation.



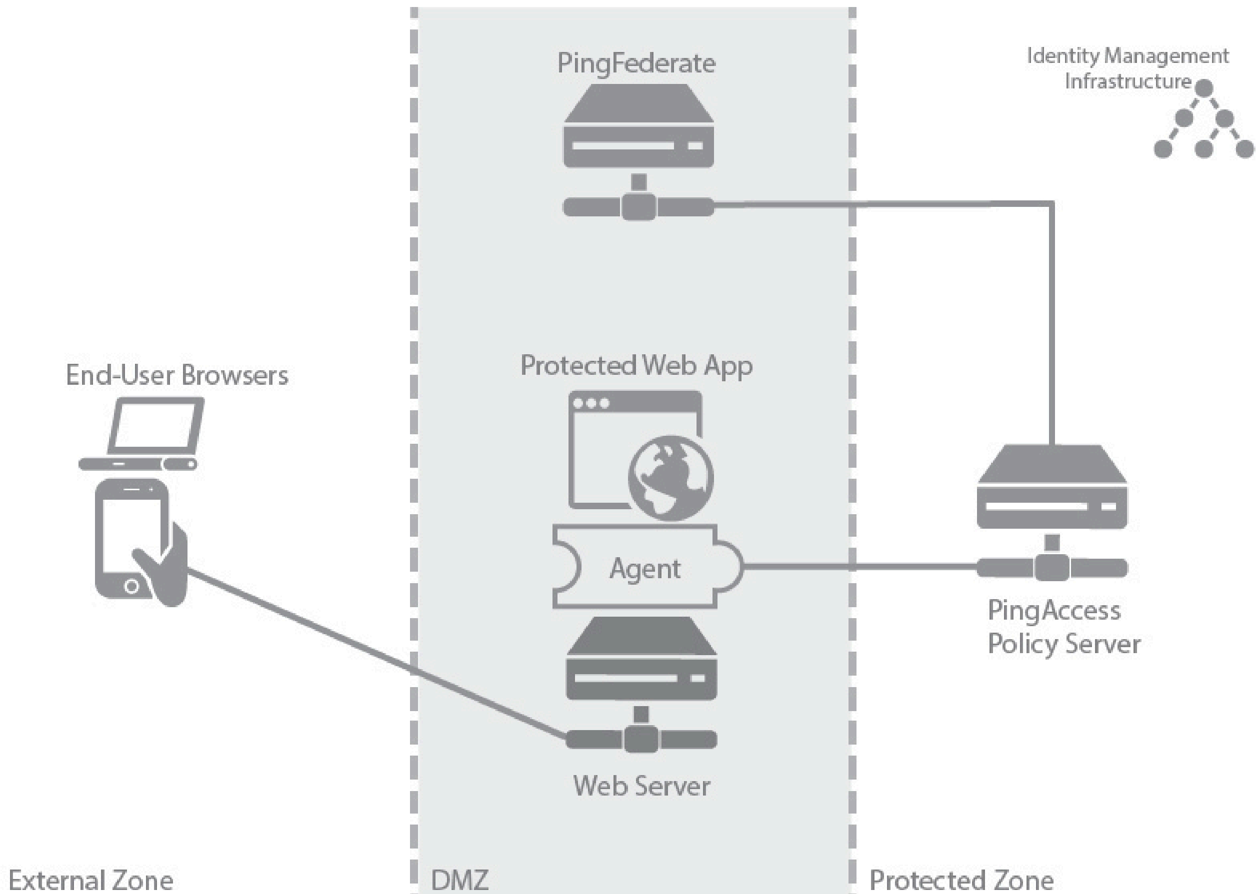
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for web applications originate.
DMZ	Externally exposing segment where PingAccess is accessible to web browsers. A minimum of two PingAccess engine nodes will be deployed in the DMZ to achieve high availability. Depending on your scalability requirements, more nodes might be required.

Zone	Description
Protected Zone	Backend controlled zone in which sites hosting the protected web applications are located. All requests to these web applications must be designed to pass through PingAccess. PingFederate is accessible to web browsers in this zone and requires access to identity management infrastructure in order to authenticate users, depicted by the icon in the diagram. A minimum of two PingFederate engine nodes will be deployed in the protected zone. Administrative nodes for both PingAccess and PingFederate can be co-located on a single machine to reduce hardware requirements.

Web Access Management Agent proof of concept deployment architecture
This proof of concept deployment environment emulates a Web Access Management (WAM) agent production environment for testing purposes.

In the test environment, PingAccess can be set up with the minimum hardware requirements. This environment example does not provide high availability and is not recommended for a Production environment.

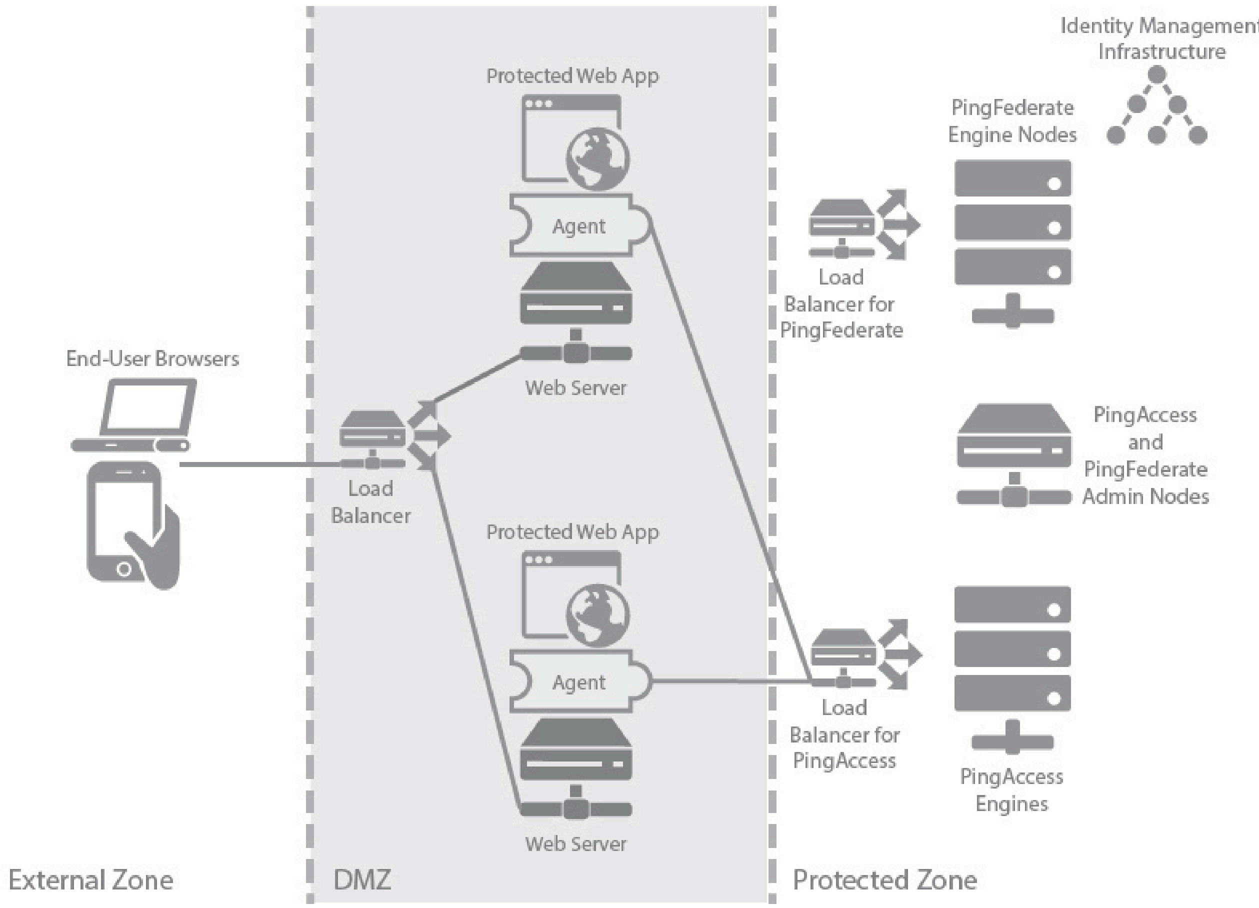


The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for web applications originate.
DMZ	Externally exposed segment where application web server is accessible to web clients. PingAccess agent is deployed as a plugin on this web server. The agent interacts with PingAccess policy server in the protected zone. PingFederate is deployed as a standalone instance in this environment because during user authentication clients interact with PingFederate. PingFederate requires access to identity management infrastructure to authenticate users.
Protected Zone	Backend controlled zone with no direct access by web clients. PingAccess policy server is deployed in this zone. PingAccess interacts with PingFederate in the DMZ zone. Identity management infrastructure is deployed in this zone.

Web Access Management Agent production deployment architecture
The deployment environment shows a Web Access Management (WAM) agent production architecture.

There are many considerations when deploying a production environment. For high availability and redundancy, the environment requires clustering and load-balancing. Load balancers are required as part of the networking infrastructure to achieve high availability by ensuring that requests are sent to available servers they are front-ending. Best practices in network design and security also include firewalls to ensure that only required ports and protocols are permitted across zones.



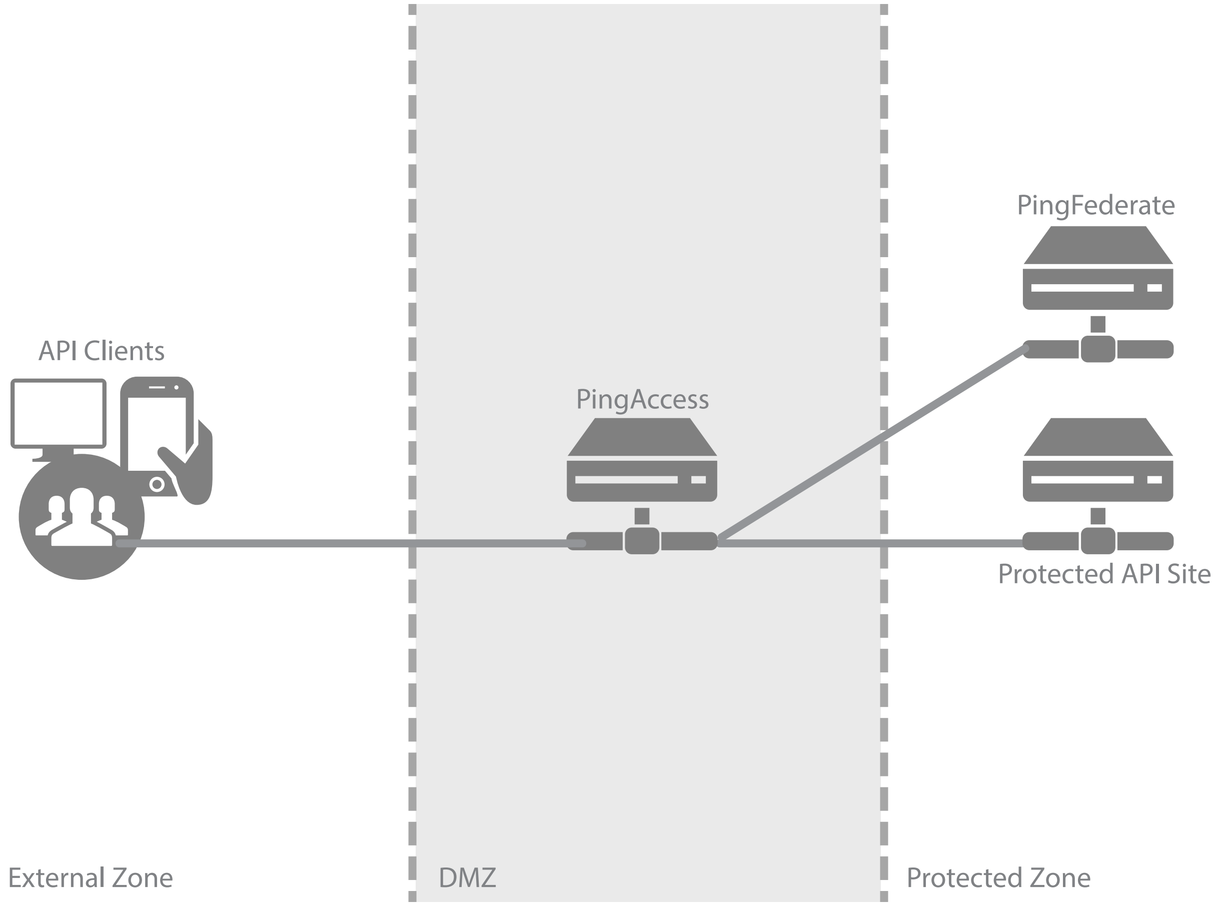
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for web applications originate.
DMZ	Externally exposed segment where possibly multiple application web servers are accessible to web clients. PingAccess agent is deployed as a plugin on these web servers. Agents interact with PingAccess policy server in the protected zone .
Protected Zone	Backend controlled zone with no direct access by web clients. PingAccess policy server is deployed in a cluster in this zone with a separate administrative engine. PingFederate is also deployed in this zone in a cluster with its own separate administrative engine. PingFederate needs access to the identity management infrastructure in order to authenticate users. Since during user authentication web clients need to interact with PingFederate directly, a reverse proxy such as PingAccess gateway is required to forward client requests through the DMZ. This aspect is not shown in the diagram.

API access management proof of concept deployment architecture

The proof of concept environment emulates an API access management environment for testing purposes.

In the test environment, PingAccess can be set up with the minimum hardware requirements. Given these conditions, do not use this proposed architecture in a production deployment because it does not provide high availability.



The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming API requests originate.
DMZ	Externally exposing segment where PingAccess is accessible to API clients. PingAccess is a standalone instance in this environment, serving as both a runtime and an administrative port.
Protected Zone	Backend controlled zone in which sites hosting the protected APIs are located. All requests to these APIs must be designed to pass through PingAccess. PingFederate is accessible to API clients in this zone and is a standalone instance, serving as both a runtime and an administrative port.

API access management production deployment architecture

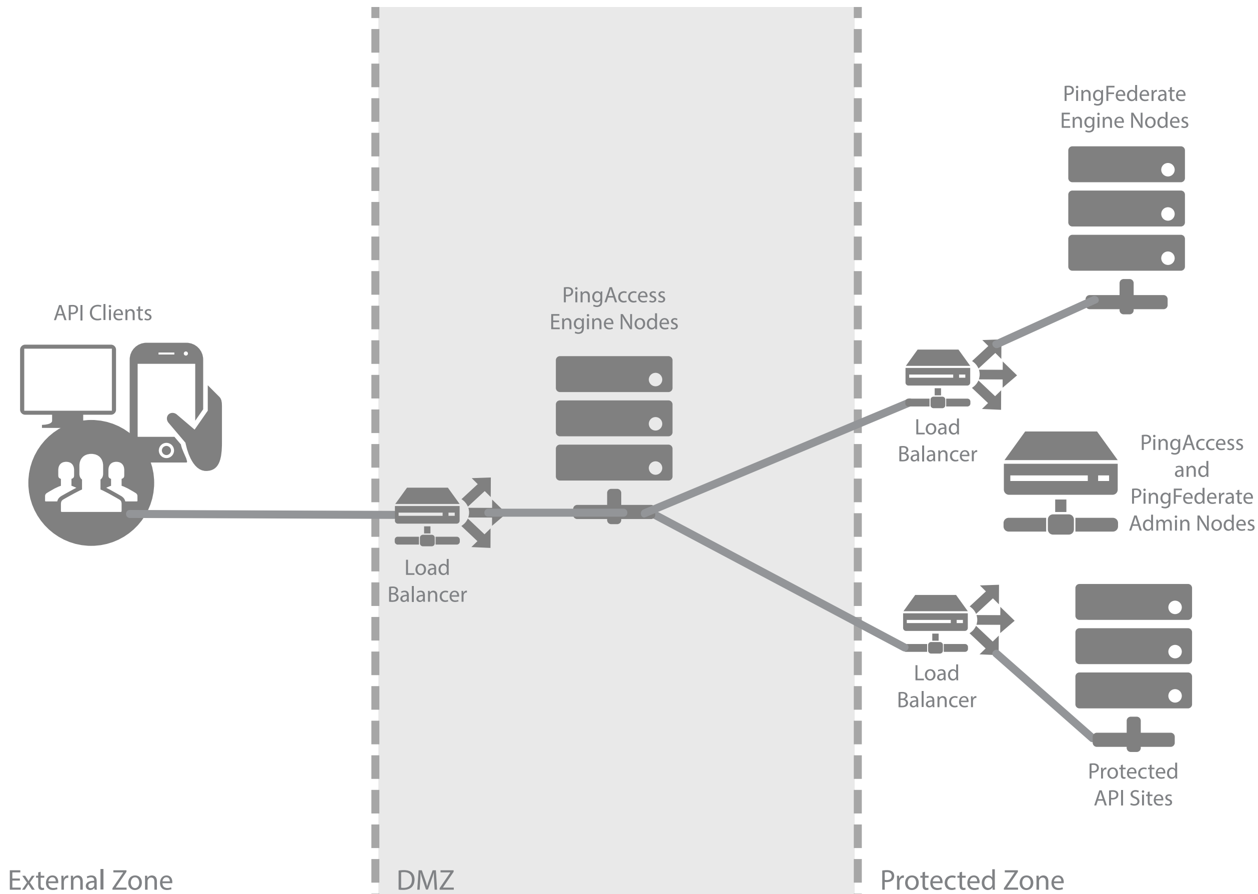
This production deployment environment shows an API access management architecture.

There are many considerations when deploying a production environment. For high availability and redundancy, the environment requires clustering and load-balancing. Load balancers are required as part of the networking infrastructure to achieve high availability by ensuring that requests are sent to available servers they are front-ending. Best practices in network design and security also include firewalls to ensure that only required ports and protocols are permitted across zones.

Note:

PingAccess provides high availability and basic load balancing for the protected web apps in the protected zone. For more information, see .

The following environment example is a recommended production quality deployment architecture for an API access management use case.



The following table describes the three zones within this proposed architecture.

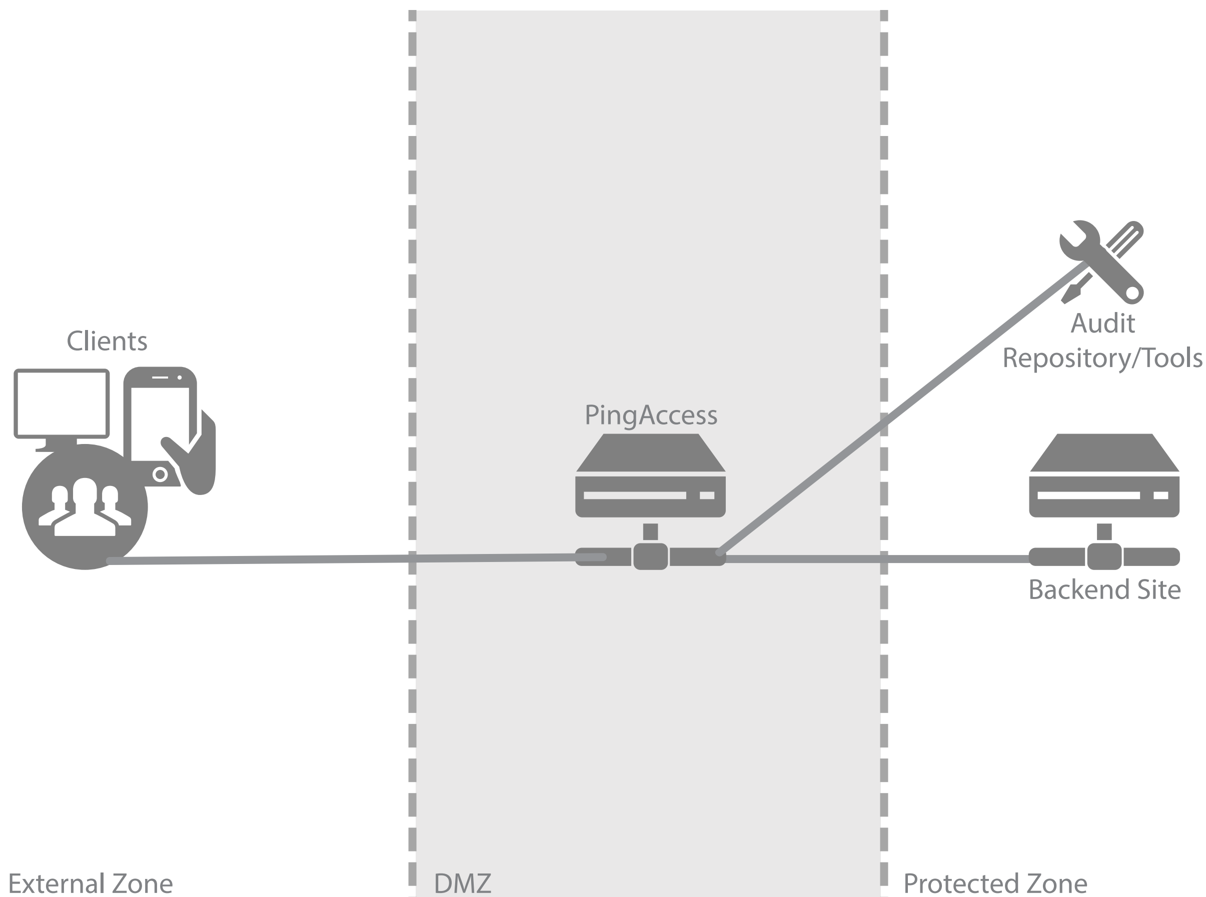
External Zone	External network where incoming API requests originate.

DMZ	Externally exposing segment where PingAccess is accessible to API clients. A minimum of two PingAccess engine nodes will be deployed in the DMZ to achieve high availability. Depending on your scalability requirements, you might require more nodes.
Protected Zone	Backend controlled zone in which Sites hosting the protected APIs are located. All requests to these APIs must be designed to pass through PingAccess. PingFederate is accessible to API clients in this zone. A minimum of two PingFederate engine nodes will be deployed in the protected zone. Administrative nodes for both PingAccess and PingFederate can be co-located on a single machine to reduce hardware requirements.

Auditing and proxying proof of concept deployment architecture

This proof of concept deployment environment is used to emulate an auditing and proxying environment for testing purposes in PingAccess.

In the test environment, you can set up PingAccess with the minimum hardware requirements. Given these conditions, do not use this proposed architecture in a production deployment because it does not provide high availability.



The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests originate.
DMZ	Externally exposing segment where PingAccess is accessible to clients. PingFederate and PingAccess are standalone instances in this environment, serving as both runtime and administrative ports.
Protected Zone	Contains back-end sites audited and proxied through PingAccess. Audit results are sent to an audit repository or digested by reporting tools. Many types of audit repository/tools are supported such as SIEM/GRC, Splunk, database, and flat files.

Auditing and proxying production deployment architecture

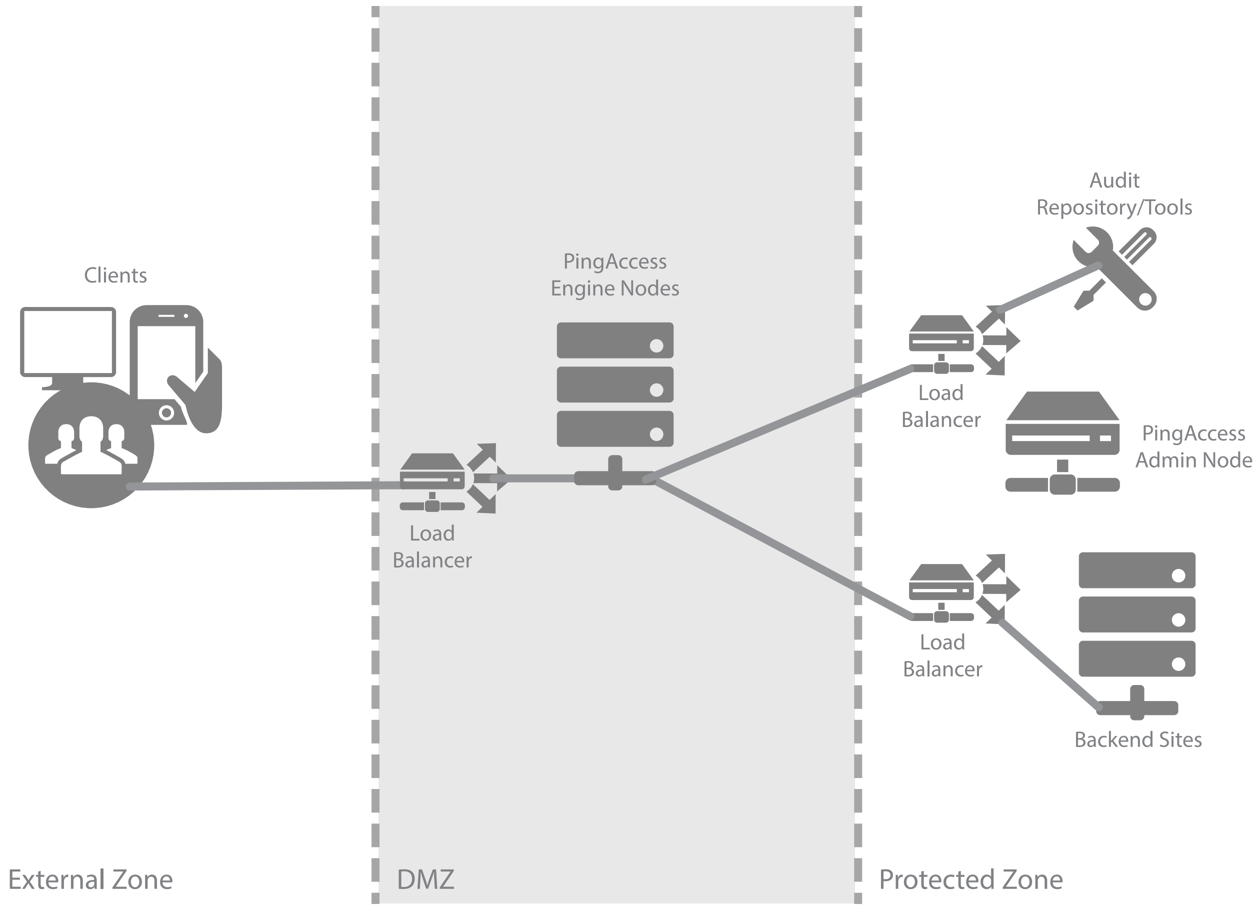
This production deployment environment shows an auditing and proxying architecture in PingAccess.

There are many considerations when deploying a production environment. For high availability and redundancy, the environment requires clustering and load-balancing. Load balancers are required as part of the networking infrastructure to achieve high availability by ensuring that requests are sent to available servers they are front-ending. Best practices in network design and security also include firewalls to ensure that only required ports and protocols are permitted across zones.

Note:

PingAccess can provide high availability and basic load balancing for the protected web apps in the protected zone. For more information, see .

The following environment example is a recommended production quality deployment architecture for an auditing and proxying use case.



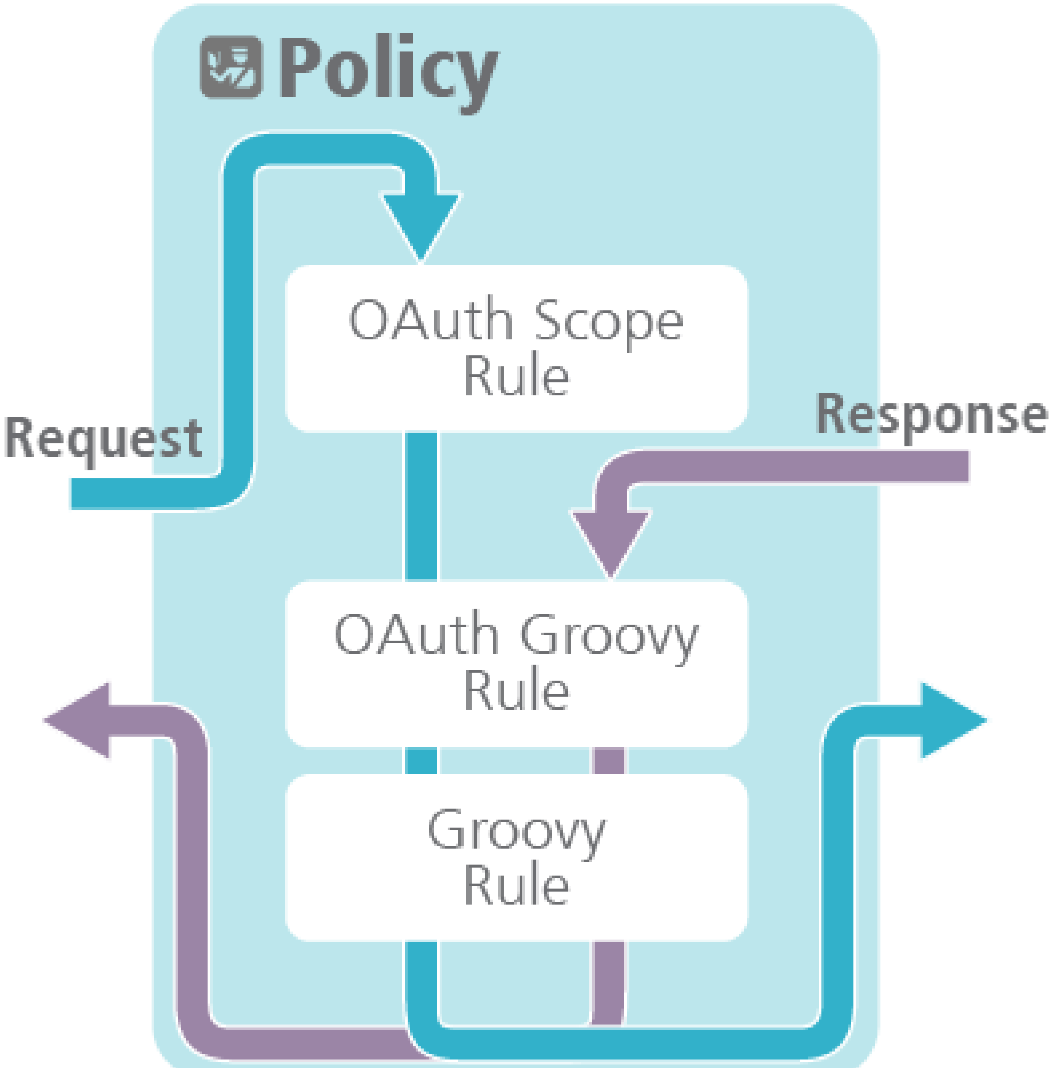
The following table describes the three zones within this proposed architecture.

External Zone	External network where incoming requests originate.
DMZ	Externally exposing segment where PingAccess is accessible to clients. A minimum of two PingAccess engine nodes will be deployed in the DMZ. Depending on your scalability requirements, you might require more nodes.
Protected Zone	Contains backend Sites audited and proxied through PingAccess. Audit results are sent to an audit repository or digested by reporting tools. Many types of audit repository tools are supported such as SIEM/GRC, Splunk, database, and flat files.

Groovy in PingAccess

PingAccess provides the Groovy script and OAuth Groovy script rule types that enable the use of Groovy, a dynamic programming language for the Java Virtual Machine (JVM).

Groovy scripts provide advanced rule logic that extends PingAccess rule development beyond the capabilities of the packaged rules. For more information, see *Groovy documentation*. Groovy scripts have access to important PingAccess runtime objects, such as the *Exchange* and *PolicyContext* objects, which the scripts can interrogate and modify. Groovy script rules are invoked during the request processing phase of an exchange, allowing the script to modify the request before it is sent to the server. Groovy script rules are also invoked during the response, allowing the script to modify the response before it is returned to the client. The diagram below highlights the flow of rule processing.



Processing steps

1. During request processing, rules associated with the application are evaluated.
2. The request passes through each of the rules before PingAccess allows it to proceed.
3. The response passes through the rules in a manner based on your deployment:
    a. In a proxy deployment, the response from the site passes through each of the rules.
    b. In an agent deployment, the response to the agent indicating the policy approval or denial passes through each of the rules.

Groovy Scripts

Groovy scripts provide advanced rule logic that extends PingAccess rule development beyond the capabilities of the packaged rules.

Groovy scripts have access to important PingAccess runtime objects, such as the *Exchange* and *PolicyContext* objects, which the scripts can interrogate and modify. Groovy script rules are invoked during the request processing phase of an exchange, allowing the script to modify the request before it is sent to the server. Groovy script rules are also invoked during the response, allowing the script to modify the response before it is returned to the client. See *Groovy* for more information about Groovy.

> **Note:**
>
> Through Groovy scripts, PingAccess administrators can perform sensitive operations that could affect system behavior and security.

Matchers

Groovy scripts must end execution with a matcher instance. Matchers provide a framework for establishing declarative rule matching objects. You can use a matcher from the list of *PingAccess Matchers* or from the *Hamcrest library*.

The following are Hamcrest method examples for constructing access control policies with the web session attribute rule using evaluations such as an `OR` group membership evaluation.

allOf

Matches if the examined object matches all of the specified matchers. In this example, the user needs to be in both the sales and managers groups for this rule to pass.

```
allOf(containsWebSessionAttribute("group","sales"),
 containsWebSessionAttribute("group","managers"))
```

anyOf

Matches any of the specified matchers. In this example, the rule passes if the user is in any of the specified groups.

```
anyOf(containsWebSessionAttribute("group","sales"),
 containsWebSessionAttribute("group","managers"),
 containsWebSessionAttribute("group","execs"))
```

not

Inverts the logic of a matcher to not match. In this example, the rule fails if the user is in both the sales and the managers groups.

```
not(allOf(containsWebSessionAttribute("group", "sales"),
 containsWebSessionAttribute("group", "managers")))
```

See *Matchers* for more information.

Objects

The following objects are available in Groovy. For more information on an object, click the link.

Exchange Object

Contains the HTTP request and the HTTP response for the transaction processed by PingAccess.

PolicyContext Object

Contains a map of objects needed to perform policy decisions. The contents of the map vary based on the context of the current user flow.

Request Object

Contains all information related to the HTTP request made to an application.

Response Object

Contains all information related to the site HTTP response.

Method Object

Contains the HTTP method name from the request made to an application.

Header Object

Contains the HTTP header information from the request made to an application or the HTTP header from a Site response.

Body Object

Contains the HTTP body from the application request or the HTTP body from the site response.

OAuthToken Object

Contains the OAuth access token and related identity attributes.

Logger Object

Configure and view the state of logging.

MediaType Object

Contains information related to the media type.

Debugging/troubleshooting

Groovy script rules are evaluated when saved to ensure that they are syntactically valid. If a Groovy script rule fails to save, hover over the information icon to view additional information about the reason for the failure.

If a rule fails when it is run, information about the failure is added to the `<PA_HOME>/log/pingaccess.log` file.

Info:

Some error messages about Groovy rule failures are only logged if `DEBUG` level output is enabled for the `com.pingidentity` logger.

Body object reference

This object accesses the Body object in Groovy `exc?.request?.body` or `exc?.response?.body`.

Purpose

The Body object contains the HTTP body from the application request or the HTTP body from the site response. The request HTTP body is sent on to the site after the rules are evaluated. The response HTTP body is sent on to the User-Agent after the response rules are evaluated.

Groovy sample

```
//Checks the actual length of the body content and set the Content-Length
 response header
def body = exc?.response?.body;
def header = exc?.response?.header;
header?.setContentLength(body?.getLength());
pass();
```

Method summary

Method	Description
byte[] getContent()	Returns the body content of the request or response.
int getLength()	Returns the length of the body content.

Exchange object reference

The Exchange object is available to both the OAuth Groovy script rule and the regular Groovy script rule. PingAccess makes the Exchange object available to Groovy Script developers to provide request and response information for custom Groovy Rules. This object accesses the Exchange object in Groovy - `exc`.

Purpose

The Exchange object contains both the HTTP request and the HTTP response for the transaction processed by PingAccess. You can use this object to manipulate the request prior to it being sent to the site. You can also use this object to manipulate the response from the site before it is sent to the client.

An instance of the Exchange object lasts for the lifetime of a single application request. You can use the Exchange object to store additional information determined by the developer.

Some fields and methods for the Response object are not available in scripts used with an Agent. See the following Method Summary table for more information.

Groovy sample

```
//Evaluate if the content length of the request is empty
if (exc?.request?.header?.contentLength > -1 )
{
    //Set a custom header in the request object
    exc?.request?.header?.add("X-PINGACCESS-SAMPLE", "SUCCESS")
    pass()
}
else
{
    println("Request content is empty") //Debugging statement
    fail()
}
```

Method summary

Method	Description
Identity getIdentity()	Obtains the PingAccess representation of the identity associated with the request. This object will be null for requests to an unprotected application or an unauthenticated request to an anonymous resource.
Request getRequest()	Obtains the PingAccess representation of the request. This request is sent to the site with any changes that might be made in a Groovy script.
Response getResponse()	Obtains the PingAccess representation of the response. If the site has not been called, the response is null. This field is not available in scripts used with an agent.

Method	Description
long getTimeReqSent()	Obtains the time, in milliseconds, when the request was sent to the site. This field is not available in scripts used with an Agent.
long getTimeResReceived()	Obtains the time, in milliseconds, when the response was received from the site. This field is not available in scripts used with an Agent.
String getRequestURI()	Returns the PingAccess URI that received the request.
String getRequestScheme()	Obtains the scheme used by the browser or other user agent that made the request.
Object getProperty(String key)	Returns the value of a custom property.
void setProperty(String key, Object value)	Sets a custom property.
SslData getSslData()	Obtains information established in the TLS handshake made with PingAccess.

Headers object reference

Access the Headers object in Groovy `exc?.request?.header` or `exc?.response?.header`.

Purpose

The Headers object contains the HTTP Header information from the request made to an application or the HTTP Header from a site response. The *Request* HTTP Header is sent on to the site after the rules are evaluated. The *Response* HTTP Header is returned to the client after the Response rules are evaluated.

Use the Headers object to add custom HTTP headers for site.

Groovy sample

```
if ( !(exc.response) )
{
     // Set a custom header for the Site request
     def header = exc?.request?.header
     header?.add("X-PINGACCESS-SAMPLE", "SUCCESS")
}
pass()
```

Method summary

Method	Description
void add(String key, String val)	Adds HTTP header fields for the request. **Info:** If Groovy Rules are used to inject HTTP headers for the backend protected application, the script must sanitize the same headers from the original client request.

Method	Description
String getAccept()	Returns the acceptable response Content-Types expected by the User-Agent.
void setAccept(String value)	Sets the acceptable response Content-Types expected by the User-Agent.
String getAuthorization()	Returns the authentication credentials for HTTP Authentication.
void setAuthorization(String username, String password)	Sets authentication credentials for HTTP Authentication.
String getConnection()	Returns the connection type preferred by the User-Agent.
void setConnection(List<String> values)	Sets the connection type preferred by the User-Agent.
int getContentLength()	Returns the request body content length.
void setContentLength(int length)	Sets the request body content length.
MediaType getContentType()	Returns media type of Header with content type
void setContentType(String)	Sets the request body MIME type.
Map <String, String[]> getCookies()	Returns all cookies sent with the request.
void setCookie(String)	Overwrites the request's cookie header with the passed string. This method cannot be used to set cookies in the response header.
String getFirstCookieValue(String)	Returns the first cookie in the cookie header.
String getFirstValue(String)	Returns the first value of the HTTP header specified by the name.
void setDate(Date date)	Sets the date of the message in the Date HTTP header.
List<GroovyHeaderField> getAllHeaderFields()	Returns a list of GroovyHeaderFields.
String getHost()	Returns the hostname specified in the request.
void setHost(String value)	Sets the hostname for the request to the Site.
String getLocation()	Gets the redirect location URL for the response.
void setLocation(String value)	Sets the redirect location URL for the response.
String getProxyAuthorization()	Returns the proxy credentials.
void setProxyAuthorization(String value)	Sets the request proxy credentials.
void setServer(String value)	Sets the server name for the response.
List<String> getValues(String name)	Returns a list of string values for the supplied header name.
String getXForwardedFor()	Returns the originating IP address of the client and the proxies, if set.
void setXForwardedFor(String value)	Sets the IP address for the client and the proxies.

Method	Description
boolean removeContentEncoding()	Removes the Content-Encoding header value. Returns true if the value has been removed.
boolean removeContentLength()	Removes the Content-Length header value. Returns true if the value has been removed.
boolean removeContentType()	Removes the Content-Type header value. Returns true if the value has been removed.
boolean removeExpect()	Removes the Expect header value. Returns true if the value has been removed.
boolean removeFields(String name)	Removes the header value specified by the name parameter. Returns true if the value has been removed.
boolean removeTransferEncoding()	Removes the Transfer-Encoding header value. Returns true if the value has been removed.

GroovyHeaderField object

Method summary

Method	Description
String getValue();	Returns the string's value.
GroovyHeaderName getHeaderName();	Returns the header's name.

Example

Groovy sample

The following example demonstrates usage of the `getAllHeaderFields()` method, which includes both request and response logging:

```
exc?.log.info "Display Headers: "
exc?.log.info "-->Request Headers"
reqHdrs = exc?.request?.header?.getAllHeaderFields()
reqLoop = reqHdrs?.iterator()
while (reqLoop?.hasNext()) {
  hdr = reqLoop?.next()
  exc.log.info "-->reqHeader  Name: "+hdr?.getHeaderName()?.toString()
  exc.log.info "-->reqHeader Value: "+ hdr?.getValue()
}
exc?.log.info "-->Response Headers"
exc?.log.debug "-->Response HTTP Status: "+ exc?.response?.statusCode
rspHdrs = exc?.response?.header?.getAllHeaderFields()
rspLoop = rspHdrs?.iterator()
while (rspLoop?.hasNext()) {
  hdr = rspLoop?.next ()
  exc.log.info "-->rspHeader  Name: "+ hdr?.getHeaderName()?.toString()
  exc.log.info "-->rspHeader Value: "+ hdr?.getValue()
}
exc?.log.info "Display Headers EOF: "
pass()
```

Identity object reference

The Identity object contains information about the authenticated identity associated with the current HTTP request.

Groovy sample

```
// Only allow access for an identity with subject "user"
def subject = exc?.identity?.subject

if ("user".equals(subject)) {
  pass()
} else {
  fail()
}
```

Method summary

Method	Description
String getSubject()	Returns the subject of the identity.
String getMappedSubject()	Returns the subject set by the identity mapping. If there is no identity mapping associated with the application, the return value will be null. If there is an identity mapping associated with the application, but the identity mapping did not determine a subject to map, the returned value might be the empty string.
String getTrackingId()	Returns the tracking identifier used in PingAccess logs. This value is not guaranteed to be globally unique and should be used for diagnostic purposes only
String getTokenId()	Returns the unique ID for the associated authentication token. This value might change when new tokens are issued for the same identity.
Date getTokenExpiration()	Returns a Date object representing the time at which the authentication token expires. This might be null if the authentication provider did not indicate an expiry.
JsonNode getAttributes()	Returns a JsonNode object representing the attributes of the identity.

JsonNode object reference

The JsonNode object represents the attributes of an identity.

Groovy sample

```
// Only allow access if the user is in the group "staff"
def groups = exc?.identity?.attributes?.get("groups")

foundGroup = falseif (groups) {
  for (group in groups) {
    if ("staff".equals(group.asText())) {
      foundGroup = truebreak
```

```
    }
  }
}

if (foundGroup) {
  pass()
} else {
  fail()
}
```

Method summary

Method	**Description**
JsonNode get(String fieldName)	Gets the JsonNode representing a field of this JsonNode. This method will return null if no field exists with the specified name.
boolean has(String fieldName)	Returns true if this JsonNode has a field with the specified name.
java.util.Iterator<*String*> fieldNames()	Returns an java.util.Iterator providing access to the names of all the fields of this JsonNode.
boolean isTextual()	Returns true if this JsonNode represents a string value.
String asText()	Returns a string representation of this JsonNode. If this JsonNode is an array or object, this will return an empty string.
int intValue()	Returns an integer representation of this JsonNode. If this JsonNode does not represent a number, 0 is returned.
boolean isArray()	Returns true if this JsonNode is an array.
boolean isObject()	Returns true if this JsonNode is an object.
int size()	For an array JsonNode, returns the number of elements in the array. For an object JsonNode, returns the number of fields in the object. 0 otherwise.
java.util.Iterator<*JsonNode*>iterator()	Returns an java.util.Iterator over all JsonNode objects contained in this JsonNode. For an array JsonNode, the returned java.util.Iterator will iterate over all the elements in the array. For an object JsonNode, the returned java.util.Iterator will iterate over all field values in the object.

Remarks

A JsonNode implements java.lang.Iterable<*JsonNode*> so a for loop can be used to iterate over all the elements in an array JsonNode or the field values in an object JsonNode.

Logger object reference

This object accesses the Logger object.

Configuration

PingAccess must be configured to accept logging from Groovy rules.

In the `conf/log4j2.xml` file, uncomment or add the following line to enable debug-level logging from Groovy rules.

```
<AsyncLogger name="GroovyRule" level="DEBUG"/>
```

Uncomment or add the following line to enable info-level logging from the *<RuleName>* Groovy rule.

```
<AsyncLogger name="GroovyRule.<RuleName>" level="INFO"/>
```

Method summary

Method	**Description**
void trace(String format, Object... arguments)	Logs a `TRACE` level message based on the specified format and arguments.
void debug(String format, Object... arguments)	Logs a `DEBUG` level message based on the specified format and arguments.
void info(String format, Object... arguments)	Logs an `INFO` level message based on the specified format and arguments.
void warn(String format, Object... arguments)	Logs a `WARN` level message based on the specified format and arguments.
void error(String format, Object... arguments)	Logs an `ERROR` level message based on the specified format and arguments.
boolean isTraceEnabled()	Checks if the logger instance is enabled for the `TRACE` level.
boolean isDebugEnabled()	Checks if the logger instance is enabled for the `DEBUG` level.
boolean isInfoEnabled()	Checks if the logger instance is enabled for the `INFO` level.
boolean isWarnEnabled()	Checks if the logger instance is enabled for the `WARN` level.
boolean isErrorEnabled()	Checks if the logger instance is enabled for the `ERROR` level.

MediaType object reference

Access the MediaType object.

Method summary

Method	**Description**
Map getParameters()	Returns a list of parameters.
String getBaseType()	Returns the media base type.

Method	Description
String getSubType()	Returns the media sub type.
String getParameter(String)	Returns a string containing the value of the request parameter.
String getPrimaryType()	Returns the primary media type.

Method object reference

Access the Method object in Groovy `exc?.request?.method`.

Purpose

The Method object contains the HTTP method name from the request made to an application. The HTTP method is sent on to the site after the rules are evaluated.

Groovy sample

```
//Retrieve the HTTP Method name and make different decisions based on the
 method name
def method = exc?.request?.method?.methodName
switch (method) {
     case "GET":
         println("GET")
         break;
     case "POST":
         println("POST")
         break;
     case "PUT":
         println("PUT")
         break;
     case "DELETE":
         println("DELETE")
         break;
default:
     println("DEFAULT")
     pass()
}
```

Method summary

Method	Description
String getMethodName()	Returns the name of the HTTP method, GET, PUT, POST, DELETE, HEAD.

OAuth Token object reference

Access the OAuth Token object in Groovy `policyCtx?.context.get("oauth_token")`.

Purpose

The OAuthToken object contains the OAuth access token and related identity attributes. The OAuthToken instance is available only for OAuth Groovy script rules.

Groovy sample

```
def scopes = policyCtx?.context.get("oauth_token")?.scopes
```

```
def attr = policyCtx?.context.get("oauth_token")?.attributes
def username =
 policyCtx?.context.get("oauth_token")?.attributes?.get("username")?.get(0)
exc?.request?.header?.add("x-scopes", "$scopes")
exc?.request?.header?.add("x-attributes", "$attr")
exc?.request?.header?.add("x-username", "$username")
pass()
```

Method summary

Method	Description
`Instant getExpiresAt()`	Contains the expiration instant of the OAuth access token.
`Instant getRetrievedAt()`	Contains the instant that the OAuth access token was retrieved from PingFederate.
`String getTokenType()`	Contains the type of OAuth access token. (Bearer, JSON Web Token (JWT)).
`String getClientId()`	Contains the client ID associated with the OAuth access token.
`Set getScopes()`	Contains the set of scopes associated with the OAuth access token.
`Map<String, List<String> >getAttributes()`	Contains a map of identity attributes specific to the user.

PolicyContext object reference

Access the PolicyContext object in Groovy `policyCtx`.

Purpose

The PolicyContext object is a map of objects needed to perform policy decisions. The contents of the map vary based on the context of the current user flow. A common example is OAuth token information stored in an OAuthToken object contained within the context map. In this example, an OAuthToken object is retrieved from the policy context by using the `oauth_token` key. The OAuthToken object is available only for the OAuth Groovy scripts rules.

Groovy sample

```
def oauthToken = policyCtx?.context.get("oauth_token")
```

Method summary

Method	Description
objectMap<String, Object> getContext()	Container for the *OAuthToken*.
Exchange getExchange()	Returns the exchange a message relates to.

Request object reference

Access the Request object in Groovy `exc?.request`.

Purpose

The Request object contains all information related to the HTTP request made to an application. The request instance is sent on to the site after the rules are evaluated.

Some fields and methods for the Response object are not available in scripts used with an agent. See the Field Summary and Method Summary tables below for more information.

Groovy sample

```
//Retrieve the request object from the exchange object
def request = exc?.request
def contentType = request?.header?.getContentType()
def containsJson = contentType?.matchesBaseType("application/json")
//Check to make sure the request body contains JSON
if (!containsJson) {
fail()
} else {
    pass()
}
```

Field summary

Field	Description
String uri	Returns the PingAccess URI that received the request.
void setUri(String)	Sets the PingAccess URI.

Method summary

Method	Description
Method getMethod	Contains the HTTP method information from the request sent to the application.
Header getHeader	Contains the HTTP header information from the request sent to the application. **Warning:** Previously executed custom rules can modify these values.
Body getBody	Contains the HTTP body information from the request sent to the application. This field is not available in scripts used with an agent. **Warning:** Previously executed custom rules can modify these values.

Method	Description
Map<String, String[]> getQueryStringParams()	Parses and returns the query string parameters from the request. If the query string parameters cannot be parsed due to formatting errors, this method will throw a URISyntaxException. Groovy scripts that use this method are not required to catch this exception. Scripts that choose not to catch this exception will fail if the query string parameters are invalid.
Map<String, String[]> getPostParams()	Parse the form parameters from the body content of the request, assuming the content is encoded using the encoding defined by the application/x-www-form-urlencoded content type.
void setBodyContent(byte[] content)	Replaces the body content of the request. This method will also adjust the Content-Length header field to align with the length of the specified content.

Response object reference

Access the Response object in Groovy `exc?.response`.

Purpose

The Response object contains all information related to the service HTTP response. The response instance is sent on to the User-Agent after the rules are evaluated.

The fields and methods for the Response object are not available in scripts used with an agent.

Groovy sample

```
if(exc?.response && exc?.identity) {
    exc.response.header.add("PA-Tracking-ID", exc.identity.trackingId)
}
pass()
```

Field summary

Field	Description
int getStatusCode()	Contains the HTTP response status code.
void setStatusCode(int)	Sets the status code from an integer.
String getStatusMessage()	Contains the HTTP response status message.
void setStatusMessage(String)	Sets the status message from a string.

Method summary

Method	Description
boolean isRedirect()	Returns true if the status code is in the 300s.

Method	**Description**
Header getHeader	Contains the HTTP header information from the response. **Warning:** Previously executed custom rules can modify these values.
Body getBody	Contains the HTTP body information from the response. **Warning:** Previously executed custom rules can modify these values.
void setBodyContent(byte[] content)	Replaces the body content of the response. This method will also adjust the Content-Length header field to align with the length of the specified content.

SslData object reference

The SslData object provides access to information established in the TLS handshake with PingAccess.

Groovy sample

```
// Force TLS client authentication
def certChain = exc?.sslData?.clientCertificateChain
if(certChain && !certChain.isEmpty())
{
  pass();
}
else
{
  fail();
}
```

Method summary

Method	**Description**
List<String> getSniServerNames()	Returns a list of server name indication (SNI) server_names sent by the user agent in the TLS handshake. Empty if the user agent did not utilize the SNI TLS extension.
List<java.security.cert.X509Certificate> getClientCertificateChain()	Returns the certificate chain presented by the user agent in the TLS handshake. Empty if the user agent did not utilize TLS client authentication.

Groovy script examples

The following examples show possible uses for Groovy scripts.

OAuth Policy context example

In some instances, it might be necessary to transmit identity information to sites to provide details of the user attempting to access a site. In such instances, Groovy scripts can be used to inject identity information into various portions of the HTTP request to the target. In this example, the site is expecting the identity of the user to be conveyed through the `User` HTTP header. This can be accomplished using the OAuth Groovy script rule and the following Groovy script.

```
user=policyCtx?.context.get("oauth_token")?.attributes?.get("user")?.get(0)
exc?.request?.header?.add("User", "$user")
pass()
```

The following is more complex Groovy script logic.

```
test = exc?.request?.header?.getFirstValue("test");
if(test != null && test.equals("foo"))
{
  //rule will fail evaluation if Test header has value 'foo'
  fail()
}
else
{
  //rule will pass evaluation is Test header has value of anything else
  //or isn't present
  pass()
}
```

Set an exchange property named `com.pingidentity.policy.error.info` so the value will be available for the `$info` variable in error templates when an error is encountered. The `$info` variable can be set by a Groovy Script rule or an OAuth Groovy script rule.

```
exc?.setProperty("com.pingidentity.policy.error.info", "this value will be
 passed to the template in $info variable")
not(anything())
```

Create a whitelisting rule for certain characters.

```
if (!exc?.request?.uri?.matches("[\\p{Po}\\p{N}\\p{Z}\\p{L}\\p{M}\\p{Zs}\\./
_\\-\\()\\{\\}\\[\\]]*"))
 {
  fail()
 }
 else
 {
  pass()
 }
```

Add a cookie to the response.

```
// Construct the cookie value
value = "cookie-value"
cookieHeaderFieldValue = "ResponseTestCookie=${value}; Path=/"

// Add the cookie on to the response
exc?.response?.header?.add("Set-Cookie", cookieHeaderFieldValue)

pass()
```

Combine an `AND` and `OR`, invoking an existing rule matcher.

```
if ((anyOf(containsWebSessionAttribute("engineering",
 "true"), containsWebSessionAttribute("marketing", "true")) &&
 (containsWebSessionAttribute("manager", "true")))
{pass()
}
else{
fail()
}
```

Matcher usage reference

Groovy script rules and OAuth Groovy script rules must end execution with a Matcher instance. This could either be a Matcher from the list of PingAccess Matchers or from the *Hamcrest library*. For more information on Hamcrest, see the *Hamcrest Tutorial*.

Examples

In the following example, the Simple Groovy rule inserts a custom HTTP header, and the script ends with a call to the Matcher `pass()`. The `pass()` Matcher signals that the rule has passed.

```
test = "let's get Groovy!"
exc?.response?.header?.add("X-Groovy", "$test")
pass()
```

In the following example, the OAuth Groovy rule checks the HTTP method and confirms the OAuth scope, and a Matcher is evaluated at the end of each line of execution. The first Matcher used is the `hasScope()` Matcher that confirms if the OAuth access token has the `WRITE` scope. If this is true, the rule passes.

```
//Get the HTTP method name
def methodName = exc?.request?.method?.methodName()
if (methodName == "POST") {
    hasScope("WRITE")
} else {
    fail()
}
```

The `fail()` Matcher combination is evaluated when the `methodName` does not equal `POST`. This Matcher combination evaluates to false.

PingAccess Matchers

The following table lists the Matchers available for the Groovy script rule and the OAuth Groovy script rule.

Matcher	Description
`pass()`	Signals that the rule has passed.
`fail()`	Signals that the rule has failed.

Matcher	Description
`inIpRange(String cidr)`	Validates the source IP address of the request against the **`cidr`**string parameter in CIDR notation. When source IP headers defined in the *HTTP Requests* page are found, the source IP address determined from those headers is used as the source address. For agents, this value is also potentially controlled by the override options on the gent settings. **Example:** `inIpRange("127.0.0.1/8")`
`inIpRange(java.net.InetAddress ipAddress, int prefixSize)`	Validates the source IP address against the **`ipAddress`** and the **`prefixSize`** parameters specified individually. When source IP headers defined in the *HTTP Requests* page are found, the source IP address determined from those headers is used as the source address. For agents, this value is also potentially controlled by the override options on the Agent settings. **Example:** `inIpRange(InetAddress.getByName("127.0.0.1"),8)` is equivalent to `inIpRange("127.0.0.1/8")`
`inIpRange(String cidr, String listValueLocation, boolean fallBackToLastHopIp, String... headerNames)`	Validates the source IP address in the first of the specified **`headerNames`** using the **`cidr`** value. Can be specified as part of a Groovy script as a means of overriding the configuration stored in PingAccess for a specific Groovy script rule. Valid values for the **`listValueLocation`** parameter are `FIRST`, `LAST`, and `ANY`. This parameter controls where, in a multivalued list of source IP addresses, the last source should be taken from. If `ANY` is used, if any of the source IP addresses in a matching header match the CIDR value, the Matcher evaluates to `true`. **Example:** `inIpRange("127.0.0.1/8", "LAST", true, "X-Forwarded-For", "Custom-Source-IP")`
`inIpRange(java.net.InetAddress address, int prefixSize, String listValueLocation, boolean fallBackToLastHopIp, String... headerName)`	Validates the source IP address in the first of the specified **`headerNames`** using the `address` and `prefixSize` values. In all other respects, this Matcher behaves the same as the version that uses a **`cidr`** value for comparison. **Example:** `inIpRange(InetAddress.getByName("127.0.0.1"), 8, "LAST", true, "X-Forwarded-For", "Custom-Source-IP")`

Matcher	Description
`requestXPathMatches(String xPathString, String xPathValue)`	Validates that the value returned by the **`xPathString`** parameter is equal to the **`xPathValue`** parameter. **Example:** `requestXPathMatches("//header[@name='Host']/text()","localhost:3000")`
`inTimeRange(String startTime, String endTime)`	Validates that the current server time is between the **`startTime`** and **`endTime`** parameters. **Example:** `inTimeRange("9:00 am","5:00 pm")`
`inTimeRange24(String startTime, String endTime)`	Validates that the current server time is between the specified 24-hour formatted time range between the **`startTime`** and **`endTime`** parameters. **Example:** `inTimeRange24("09:00","17:00")`
`requestHeaderContains(String field, String value)`	Validates that the HTTP header field value is equal to the **`value`** parameter. **Example:** `requestHeaderContains("User-Agent", "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.36")`
`requestHeaderContains(Map<String, String> fieldValuesMap, boolean caseSensitive)`	Validates that all of the HTTP header fields map to the associated value. The first **`fieldValuesMap`** string contains the HTTP header name, and the second string contains the value to compare the incoming request header value with. The **`caseSensitive`** parameter determines whether a case-sensitive comparison is performed on the value. The second string in the **`fieldValuesMap`** supports Java regular expressions. If multiple pairs of strings are present in the **`fieldValuesMap`** parameter, then all conditions must be met in order for the Matcher to pass. **Example:** `requestHeaderContains(['User-Agent':'Mozilla/5.0', 'Cookie':'JSESSIONID'], false)`

Matcher	Description
`requestPostFormContains(Map<String, String> fieldValuesMap, boolean caseSensitive)`	Validates that all of the HTTP form fields maps to the associated value. The first **`fieldValuesMap`** string contains the form header name, and the second string contains the value to compare the incoming request header value with. The **`caseSensitive`** parameter determines whether a case-sensitive comparison is performed on the value. **Note:** This Matcher determines whether to use fields passed in the URL or forms with a `content-type` header of `application/x-www-form-urlencoded`. The second string in the **`fieldValuesMap`** supports Java regular expressions. If multiple pairs of strings are present in the **`fieldValuesMap`** parameter, then all conditions must be met in order for the Matcher to pass. **Example:** `requestPostFormContains(['email':'@example.com', 'phonenumber':'720'], false)`
`requestHeaderDoesntContain(String field, String value)`	Validates that the HTTP header field value is not equal to the **`value`** parameter. **Example:** `requestHeaderDoesntContain("User-Agent", "InternetExplorer")`
`requestBodyContains(String value)`	Validates that the HTTP body contains the **`value`** parameter. **Example:** `requestBodyContains("production")`
`requestBodyDoesntContain(String value)`	Validates that the HTTP body does not contain the value parameter. **Example:** `requestBodyDoesntContain("test")`
`containsWebSessionAttribute(String attributeName, String attributeValue)`	Validates that the PingAccess token contains the attribute name and value. **Example:** `containsWebSessionAttribute("sub", "sarah")`
`containsACRValues(String value)`	Validates that the PingAccess token contains a matching ACR value.

The following table lists the Matchers available to only the OAuth Groovy rule.

Matcher	Description
`hasScope(String scope)`	Validates that the OAuth access token contains the **`scope`** parameter. **Example:** `hasScope("access")`
`hasScopes(String... scopes)`	Validates that the OAuth access token contains the list of scopes. **Example:** `hasScopes("access","portfolio")`
`hasAttribute(String attributeName, String attributeValue)`	Checks for an attribute value within the current OAuth2 policy context. **Example:** `hasAttribute("account","joe")`

Performance tuning

While PingAccess has been engineered as a high performance engine, its default configuration might not match your deployment goals nor the hardware you have available. Use the recommendations here to optimize various aspects of a PingAccess deployment for maximum performance.

Info:

An additional document related to performance, the PingAccess Capacity Planning Guide, is also available to customers as a performance data reference. This document is available from the *Customer Portal*.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Java tuning

One of the most important tuning options you can apply to the Java Virtual Machine (JVM) is to configure how much heap, memory for runtime objects, to use.

The JVM grows the heap from a specified minimum to a specified maximum. If you have sufficient memory, fix the size of the heap by setting minimum and maximum to the same value. This allows the JVM to reserve its entire heap at startup, optimizing organization and eliminating potentially expensive resizing.

By default, PingAccess fixes the Java heap at 512 megabytes (MB). This is a fairly small footprint and not optimal for supporting higher concurrent user loads over extended periods of activity. If you expect your deployment of PingAccess to serve more than 50 concurrent users per PingAccess node, if deploying a cluster, increase the heap size.

For more information, see the following topics:

- *Configuring JVM crash log in Java startup* on page 218
- *Configuring memory dumps in Java startup* on page 219
- *Modifying the Java heap size* on page 219

Configuring JVM crash log in Java startup

Enable or disable the Java Virtual Machine (JVM) crash log.

About this task

The JVM crash log location is specified in `run.bat` on Windows, or `run.sh` on Linux, and is enabled by default.

Steps

- Open `<PA_HOME>/bin/run.bat` on Windows, or `<PA_HOME>/bin/run.sh` on Linux, for editing. Choose from:
  - To disable JVM crash log reporting, comment out the line that specifies the JVM crash log location. For example, `#ERROR_FILE="-XX:ErrorFile=$PA_HOME/log/java_error%p.log"`.
  - To enable JVM crash log reporting, remove the comment tag and make the line active. For example, `ERROR_FILE="-XX:ErrorFile=$PA_HOME/log/java_error%p.log"`.

Configuring memory dumps in Java startup

You can enable or disable Java Virtual Machine (JVM) memory dump, or change the location where the dump is stored.

About this task

The JVM memory dump location is specified in `run.bat` on Windows, or `run.sh` on Linux, and is disabled by default.

Steps

- Open `<PA_HOME>/bin/run.bat` on Windows, or `<PA_HOME>/bin/run.sh` on Linux, for editing. Choose from:
  - To enable JVM memory dump, remove the comment tag on the line that specifies the JVM memory dump location. For example, `HEAP_DUMP="-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$PA_HOME/log"`.
  - To disable JVM memory dump, comment out the line. For example, `#HEAP_DUMP="-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$PA_HOME/log"`.

Modifying the Java heap size

Modify the Java heap size for both Windows and Linux installations, including their services.

Steps

1. Open the `jvm-memory.options` file located in `<PA_HOME>/conf`.
2. Specify overall heap size by modifying the `#Minimum heap size` and `#Maximum heap size` parameters.
   Choose from:
   - Modify `-Xms512m` to change the `#Minimum heap size` value.
   - Modify `-Xmx512m` to change the `#Maximum heap size` value.

   Specify units as `m`, megabytes, or `g`, gigabytes.
3. Specify young generation size by modifying the `#Minimum size for the Young Gen space` and `#Maximum size for the Young Gen space` variables.
   Choose from:
   - Modify `-XX:NewSize=256m` to change the `#Minimum size for the Young Gen space` value.
   - Modify `-XX:MaxNewSize=256m` to change the `#Maximum size for the Young Gen space` value.

   Set values to 50% of `#Minimum heap size` and `#Maximum heap size`.

   > **Info:**
   >
   > Not advisable if selecting the G1 collector. For more information, see *Garbage Collector Configuration*.

4. If you are running PingAccess as a Windows service, run the `generate-wrapper-jvm-options.bat` file located in `<PA_HOME>/sbin/windows`.

   This file applies the changes from the `jvm-memory.options` file to the `wrapper-jvm-options.conf` file, which is used by the Windows service.

Operating system tuning

This section contains tuning recommendations for your operating system.

The tuning recommendations provided here are useful in preventing deployment issues in high capacity environments. See the included topics for guidance specific to your operating system:

- *Linux tuning* on page 220
- *Windows tuning* on page 222

Linux tuning

This section describes tuning recommendations for the Linux operating system environment.

Implement these recommendations to prevent deployment issues, particularly in high capacity environments. The following settings will increase the performance and capacity of the networking, particularly TCP, stack, and file descriptor usage, respectively, enabling PingAccess to handle a high volume of concurrent requests.

For more information, see the following topics:

- *Tuning network and TCP settings* on page 220
- *Increasing file descriptor limits (systemv)* on page 221
- *Increasing file descriptor limits (systemd)* on page 222

Tuning network and TCP settings

Increase the performance and capacity of the networking, particularly TCP, stack to enable PingAccess to handle a high volume of concurrent requests.

Steps

1. Open the `/etc/sysctl.conf` file.
2. Add or modify the following properties.

```
##TCP Tuning##
# Controls the use of TCP syncookies (default is 1)
# and increase the number of outstanding syn requests allowed.
net.ipv4.tcp_syncookies=1
net.ipv4.tcp_max_syn_backlog=8192

# Increase number of incoming connections.
# somaxconn defines the number of request_sock structures allocated
# per each listen call.
# The queue is persistent through the life of the listen socket.
net.core.somaxconn=4096

# Increase number of incoming connections backlog queue.
# Sets the maximum number of packets, queued on the INPUT side,
# when the interface receives packets faster
# than kernel can process them.
net.core.netdev_max_backlog=65536

# increase system IP port limits
net.ipv4.ip_local_port_range=2048 65535

# Turn on window scaling which can enlarge the transfer window:
net.ipv4.tcp_window_scaling=1
```

```
# decrease TCP timeout
net.ipv4.tcp_fin_timeout=10

# Allow reuse of sockets in TIME_WAIT state for new connections
# (While this may increase performance, use with caution according
# to the kernel documentation.  This setting should only be enabled
# after the system administrator reviews security considerations.)
net.ipv4.tcp_tw_reuse=1

# Increase the read and write buffer space allocatable
# (minimum size, initial size, and maximum size in bytes)
net.ipv4.tcp_rmem = 4096 65536 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

# The maximum number of packets which may be queued
# for each unresolved address by other network layers
net.ipv4.neigh.default.unres_qlen=100
net.ipv4.neigh.eth0.unres_qlen=100
net.ipv4.neigh.em1.unres_qlen=100

# Default Socket Receive and Write Buffer
net.core.rmem_default=8388608
net.core.wmem_default=8388608
##############
```

Increasing file descriptor limits (systemv)

Increase file descriptor limits in a systemv environment to enable PingAccess to handle a high volume of concurrent requests.

Steps

1. Open the `/etc/security/limits.conf` file.
2. Add or modify the following lines.

   ```
   <pingAccessAccount> soft nofile <value>
   <pingAccessAccount> hard nofile <value>
   ```

   `<pingAccessAccount>` is the user account used to run the PingAccess java process, or `*` for all users, and <*value*> is the new value. A value of 65536 (64K) should be sufficient for most environments.

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 65536 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 65536, the host is likely not sized appropriately for PingAccess.

Increasing file descriptor limits (systemd)

Increase file descriptor limits in a systemd environment to enable PingAccess to handle a high volume of concurrent requests.

Steps

1. Open the `/etc/systemd/system/pingaccess.service` file.
2. Modify the following line under the `[Service]` section.

   ```
   LimitNOFILE=<value>
   ```

   <*value*> is the new value. The default value of 65536 (64K) should be sufficient for most environments.

   > **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 65536 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 can set the file descriptor limit to 100000. If the file-max value is lower than 65536, the host is likely not sized appropriately for PingAccess.

3. Run the following command as root.

   ```
   systemctl daemon-reload
   ```

4. Restart the PingAccess service.

Windows tuning

This section describes tuning recommendations for the Windows, version 7 or later, operating system environment.

Implement these recommendations to prevent deployment issues, particularly in high capacity environments. The following settings will increase the performance and capacity of network, specifically the TCP socket, connectivity, enabling PingAccess to handle a high volume of concurrent requests.

For more information, see the following topics:

- *Increasing the number of available ephemeral ports* on page 222
- *Reducing the socket TIME_WAIT delay* on page 223

Increasing the number of available ephemeral ports

Increase the number of available ephemeral ports to prevent deployment issues, particularly in high capacity environments.

About this task

This setting increases the performance and capacity of network, specifically the TCP socket, connectivity, enabling PingAccess to handle a high volume of concurrent requests.

Steps

1. View ephemeral ports: `netsh int ipv4 show dynamicportrange tcp`.
2. Increase ephemeral ports: `netsh int ipv4 set dynamicport tcp start=1025 num=64510`.
3. Reboot the machine.

4. View and confirm updated port range: `netsh int ipv4 show dynamicportrange tcp`.

Reducing the socket TIME_WAIT delay

Reduce the socket `TIME_WAIT` delay to prevent deployment issues, particularly in high capacity environments.

About this task

This setting increases the performance and capacity of network, specifically the TCP socket, connectivity, enabling PingAccess to handle a high volume of concurrent requests.

Steps

1. Click **Start** # **Run**.
2. Type `regedit` and click **OK** to open the Registry Editor.
3. Go to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters`.
4. Create a new `DWORD` value, 32 bit, and provide the name `TcpTimedWaitDelay`.
5. Set a decimal value of `30`.
6. Reboot the machine.

Garbage collector configuration reference

The following table provides guidance for configuring the garbage collector.

Selecting the appropriate garbage collector depends on the size of the heap and available CPU resources. The following is a table of available collectors and some general guidance on when and how to use them.

Specify the garbage collector using the `jvm-memory.options` file located in *<PA_HOME>*`/conf`. Modify the parameter beneath `#Use the parallel garbage collector` using the information provided below.

Garbage Collector	Description	Modifications
Parallel	▪ Best used with heaps 4GB or less ▪ Full stop-the-world copying and compacting collector ▪ Uses all available CPUs, by default, for garbage collection	Default collector for server Java virtual machine (JVM). No modification is required.
Concurrent Mark Sweep (CMS)	▪ Best for heaps larger than 4GB with at least 8 CPU cores ▪ Mostly a concurrent collector ▪ Some stop-the-world phases ▪ Non-Compacting ▪ Can experience expensive, single threaded, full collections due to heap fragmentation	Set to `#XX:+UseConcMarkSweepGC` in `jvm-memory.options`.

Garbage Collector	Description	Modifications
Garbage First (G1)	▪ Best for heaps larger than 6GB with at least 8 CPU cores ▪ Combination concurrent and parallel collector with small stop-the-world phases ▪ Long-term replacement for CMS collector, does not suffer heap fragmentation like CMS	Set to `#XX:+UseG1GC` in `jvm-memory.options`. Also disable `#Minimum size for the Young Gen space` and `#Maximum size for the Young Gen space` tuning. Explicit sizing adversely affects pause time goal. To disable, comment the lines out in the script.

Configuring acceptor threads

Configure the pool of acceptor threads based on your environment.

About this task

PingAccess uses a pool of threads to respond to HTTP/S requests made to the TCP ports in use. This applies to both administrative and runtime engine listening ports. Acceptor threads read user requests from the administrative or runtime port and pass the requests to worker threads for processing. For performance, only one acceptor thread need be used in most situations. On larger multiple CPU core machines, more acceptors can be used.

To modify, open the `run.properties` file located in the `conf` directory of your PingAccess deployment and specify the number of acceptors you want to use on the following lines:

Steps

1. Open the `run.properties` file located in the `conf` directory of your PingAccess deployment.
2. Specify the number of acceptors you want to use on the following lines:
   - `admin.acceptors=<N>`
   - `engine.http.acceptors=<N>`
   - `agent.http.acceptors=<N>`

   Where <*N*> represents the number of acceptor threads.

Configuring worker threads

Modify the minimum and maximum number of worker threads to increase performance.

About this task

PingAccess engines use a pool of worker threads to process requests received on the runtime ports, and another pool to process requests from agents. These worker threads do the initial handling of request processing within PingAccess. By default, PingAccess starts with a minimum of 5 worker threads in each pool. New threads are created as needed, with the maximum size being unlimited.

Maintenance of the pool size is such that if the number of threads in the pool exceeds the value of `engine.httptransport.coreThreadPoolSize` or `agent.httptransport.coreThreadPoolSize`, threads idle for 60 seconds are terminated and removed from the pool. The idle timeout value is not modifiable. However, if the values of `engine.httptransport.coreThreadPoolSize` and `engine.httptransport.maxThreadPoolSize` are the same, a fixed sized pool is created and idle threads are not terminated and removed. The same is true for `agent.httptransport.coreThreadPoolSize` and `agent.httptransport.maxThreadPoolSize`.

Unless you want to restrict the maximum number of concurrent requests admitted for processing by a PingAccess engine, you should leave `engine.httptransport.maxThreadPoolSize` and `agent.httptransport.maxThreadPoolSize` set to unlimited (-1).

For `engine.httptransport.coreThreadPoolSize` and `agent.httptransport.coreThreadPoolSize`, the defaults should be sufficient for all deployments. The creation of new threads is efficient, so it is unlikely that end users could detect any improvement in performance by increasing the values for these properties. Increasing these values also increases the baseline memory footprint of PingAccess due to the increase in memory for the additional threads.

These worker threads do not perform any input or output processing. Incoming and outgoing communication (I/O) is performed by two additional pools of threads. One pool handles request from, and responses to, clients; another handles requests to, and responses from, application servers proxied by PingAccess. These pools use a fixed number of threads and by default, the number of threads in these pools is equal to twice the number of logical CPU cores on the host computer.

Steps

1. Open the `run.properties` file located in the `conf` directory of your PingAccess deployment.
2. Add or edit the following properties,

   ```
   engine.httptransport.maxThreadPoolSize=<N>
   ```

   and

   ```
   agent.httptransport.maxThreadPoolSize=<N>
   ```

   where <*N*> represents the number of worker threads.

Backend server connections

PingAccess provides the Max Connections option to control and optimize connections to the proxied site.

Maximum Connections

Connections to PingAccess are not explicitly connections to the proxied site. PingAccess creates a pool of connections, unlimited in size by default, that are multiplexed to fulfill client requests. Maintenance of the pool includes creating connections to the site when needed, if none are available, and removing connections when they are closed by the backend server due to inactivity.

In certain situations, it can be advantageous to limit the number of connections in the pool for a given website. If, for example, the website is limited to the number of concurrent connections it can handle or has specific HTTP Keep Alive settings, limiting the number of connections from PingAccess can improve overall performance by not overloading the backend server. In the event that all connections in the pool are in use, a requesting thread waits for one to become available. Assuming that the response time from the backend site is sufficiently fast, the time spent waiting for a connection is likely to be less than if the system becomes overloaded.

> **Info:**
>
> It is important to understand the limits and tuning of the server application being proxied. Setting the **Maximum Connections** value too low might create a bottleneck to the proxied site, setting the value too high, or unlimited, might cause PingAccess to overload the server.

For information on setting the **Maximum Connections**, see *Sites documentation*.

Logging and Auditing

PingAccess uses a high performance, asynchronous logging framework to provide logging and auditing services with the lowest possible impact to overall application performance.

For more information, see the following topics:

- *Logging* on page 226
- *Auditing* on page 226

Logging

Modify your logging settings to increase performance.

Although logging is handled by a high performance, asynchronous logging framework, it is more efficient for the system overall to log the minimum amount of information required. Review the section of the documentation for logging and adjust the level to the lowest, most appropriate level to suit your needs.

Auditing

Modify your environment's auditing settings based on your security and performance needs.

As with logging, auditing is provided by the same high performance, asynchronous logging framework. Auditing messages can be written to a database instead of flat files, decreasing file I/O. If you do not require auditing for interactions with a resource or between PingAccess and PingFederate, it is more efficient to disable audit logging. However, if you do require auditing services and have access to a Relational Database Management System (RDBMS), audit to a database. You will see a decrease in disk I/O, which might result in increased performance, depending on database resources.

Agent tuning reference

Modify the properties of your PingAccess agents to improve performance.

You can configure several properties in the `agent.properties` file for increased performance. See the agent documentation for *Apache* or *IIS* for more information on agent configuration and setting properties.

Maximum Connections

Connections from the agent to PingAccess are limited by `agent.engine.configuration.maxConnections`. Though the default is set to `10`, the PingAccess policy server sees optimal performance at 50 concurrent requests per CPU. In certain situations it can be advantageous to increase the number of connections. In the event that all connections in the pool are in use, a requesting thread waits for one to become available. Assuming that the response time to PingAccess is sufficiently fast, the time spent waiting for a connection is likely to be less than if the system becomes overloaded.

Note:

This is the maximum number of connections per worker process, and not simply the total number of workers the agent has access to. Setting the `agent.engine.configuration.maxConnections` value too low might create a bottleneck to PingAccess, and setting the value too high might cause PingAccess to become overloaded.

Maximum Tokens

By default, the maximum number of cached tokens in an agent is unlimited. In certain situations, it can be advantageous to limit the size of the cache for the agent, as a smaller cache has a smaller memory footprint, freeing up memory available to the application for servicing requests. However, when the token cache limit is reached, the least-recently used token-policy mapping will be removed from the cache. If that token-policy mapping happens to be needed again, the agent will have a cache miss, resulting in the need to obtain a new token-policy mapping from PingAccess.

PingAccess User Interface Reference Guide

This guide provides a reference for configuration of PingAccess features and components. Use this guide in conjunction with PingAccess use case documentation for a comprehensive set of instructions to your PingAccess implementation.

For ease of use, this guide is modeled after the PingAccess user interface. To learn more about configuration options for a particular window, go to its corresponding topic.

To learn about PingAccess, including its features and functions, see the *Introduction to PingAccess* on page 38.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Applications header

The Applications header contains menu options related to directly administering sites and applications.

The applications header contains these menu options:

- *Applications* on page 227
- *Sites* on page 247
- *Agents* on page 254

Applications

This section contains controls for managing applications, resources, and redirects.

Choose from one of the following sub-sections:

- *Applications operations* on page 227
- *Global unprotected resources* on page 242
- *Redirects* on page 244
- *Virtual hosts* on page 245

Applications operations

Applications represent the protected web applications and APIs that receive client requests.

Applications consist of one or more resources, have a common virtual host and context root, and correspond to a single target site. Applications use a common web session and identity mapping. Apply access control and request processing rules and their resources on the **Policy Manager** window to protect them. Applications can be protected by a PingAccess gateway or a PingAccess agent. In a gateway deployment, the target application is specified as a site. In an agent deployment, the application destination is an agent.

There are three application types:

- Web
- API
- Web + API

Web + API applications allow administrators to configure both Web and API settings for an application. These applications are able to switch between web and API processing behaviors on the fly based on whether the inbound request contains a web session cookie (Web) or an OAuth token (API). If the inbound request contains neither, PingAccess will fall back to the method you specify as the fallback type for the application.

Use the **Policy Manager** window to define the applications which PingAccess protects and to which client requests are ultimately forwarded. Use resources to partition the application into areas requiring distinct access control. Each application contains at least a root resource. The combination of virtual server and context root must be unique for each application.

About SPA support

SPA support merges the conventional `401` unauthorized response of an API application with the traditional `302` redirect response of a web application when a client request does not contain an authentication token.

The SPA supported result is a `401` response containing a JavaScript body that can initiate a `302` redirect. API clients will ignore the JavaScript body and react appropriately to the `401` response. However, browser clients will disregard the `401` response and execute the JavaScript body, resulting in a redirect to the token provider to authenticate. Since clients self-select the portion of the response they are prepared to process, the result is a seamless authentication experience regardless of the client type.

SPA support applies to API and Web + API applications. When SPA support is enabled for Web + API applications, where a variety of client types are expected to communicate with the application, a fallback type is no longer required since both web and API clients are properly redirected to authenticate by the same response. For Web applications, authentication challenge responses fulfill the same role. See *Authentication* on page 291 for more information.

It might also benefit Web or API application types, for example, if a new version of a web application contains JavaScript framework components to call APIs. In this case, SPA support can help mitigate issues in responding to different client types for authentication, but it will not enable the full features of the other application type. You would need to migrate the application to a Web + API configuration in order to take advantage of the full functionality, such as for authentication, rules, or identity mapping.

For additional guidance in preparing a SPA to work with PingAccess, see the SPA developer's guide in the *PingAccess resources on github*. This guide contains a sample application before and after onboarding.

Adding an application

Add a new application in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click **+ Add Application**.
3. Complete the fields.

   For more information, see *Manage Applications - Field Descriptions*.

4. Click **Save**.

   Save & Go to Resources lets you configure additional application resources. For more information, see *Adding application resources* on page 235.

> **Note:**
>
> When you save the application, PingAccess verifies the redirect URI for the application's virtual host is configured in PingFederate. If PingAccess determines that the redirect URI is not defined, you will see the following warning.
>
> ```
> Save succeeded. Unable to find a matching Redirect URI in
>  the PingFederate OAuth Client configuration for [<VHost>/pa/oidc/cb]
> ```
>
> If you see this warning, ensure that there is a redirect URI that matches configured. If you have a wildcard in your virtual host configuration, ensure the redirect URI list includes the same wildcard host definition, otherwise you might have a configuration that is only valid in some circumstances.
>
> This validation is performed if the **Application Type** is `Web` or `Web + API`, a **Web Session** is selected, and the PingFederate Administration connection is configured.

Application field descriptions

This table describes the fields available for managing applications on the **Applications** tab.

<table>
<tr><th>Field</th><th>Required</th><th>Description</th></tr>
<tr><td>Name</td><td>Yes</td><td>A unique name for the application.</td></tr>
<tr><td>Description</td><td>No</td><td>An optional description for the application.</td></tr>
<tr><td>Context Root</td><td>Yes</td><td>The context at which the application is accessed at the site.

Note:

This value must meet the following criteria:
It must start with <code>/</code>.
It can contain additional <code>/</code> path separators.
It must not end with <code>/</code>.
It must not contain wildcards or regular expression strings.
The combination of the Virtual Host and Context Root must be unique. The following is allowed and incoming requests will match the most specific path first:
<code>vhost1:443/App</code>
<code>vhost1:443/App/Subpath</code>
<code>/pa</code> is, by default, reserved for PingAccess and is not allowed as a Context Root. You can change this reserved path using the PingAccess Admin API.</td></tr>
<tr><td>Case Sensitive Path</td><td>No</td><td>Indicates whether or not to make request URL path matching case sensitive.</td></tr>
<tr><td>Virtual host(s)</td><td>Yes</td><td>Specifies the virtual host for the application. Click + Create to create a virtual host. See Creating new virtual hosts on page 245 for more information.</td></tr>
</table>

Field	Required	Description
Application Type	Yes	Specifies the application type, either `Web`, `API`, or `Web + API`. ▪ If the **Application Type** is `Web`, select the **Web Session** if the application is protected and, if applicable, the **Web Identity Mapping** for the application. Select an **Authentication Challenge Policy** to produce authentication challenges for the application. Click **+ Create** to create a web session or identity mapping. You can enter an **OpenID Connect Provider Issuer URL** to replace the visible URL during authentication if the token provider supports it. ▪ If the **Application Type** is `API`, specify whether or not you want to enable *SPA Support*. Indicate the method of **Access Validation** and, if applicable, select the **API Identity Mapping** for the application. Click **+ Create** to create an access validation or identity mapping. ▪ If the **Application Type** is `Web + API`, indicate the method of **Access Validation**. Select the **Web Session** and, if applicable, the **Web Identity Mapping** and **API Identity Mapping** to use for each type. Select an **Authentication Challenge Policy** to produce authentication challenges for the application. In this configuration, the web session is required and the API is protected by default. Click **+ Create** to create an access validation, web session, web identity mapping, or API identity mapping. You can enter an **OpenID Connect Provider Issuer URL** to replace the visible URL during authentication if the token provider supports it. Specify whether or not you want to enable *SPA Support*
Destination	Yes	Specifies the application destination type, either `Site`, `Agent`, or `Sideband`. ▪ If the destination is a `Site`, select the **Site** requests are sent to when access is granted. If HTTPS is required to access this application, and at least one non-secure HTTP listening port is defined, select the **Require HTTPS** option. Click **+ Create** to create a Site. For more information, see *Adding sites* on page 247. ▪ If the destination is an `Agent`, select the agent which intercepts and validates access requests for the Application. Click **+ Create** to create an Agent. For more information, see *Adding agents* on page 255. ▪ If the destination is `Sideband`, select the sideband client that queries PingAccess for authorization and request/response modification. Click **+ Create** to create a sideband client. For more information, see *Adding sideband clients* on page 257.
Enabled	No	Select to enable the application and allow it to process requests.

Editing an application

Edit an existing application in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand the application you want to edit.
3. On the **Properties** tab, click the **Pencil** icon.
4. Make the required changes.

   For more information, see *Manage Applications - Field Descriptions*.
5. To confirm your changes, click **Save**.

Deleting an application

Delete an existing application in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand the application you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Authentication challenge responses

This table describes the authentication challenge responses generated by PingAccess, based on its configuration and properties of the request.

An authentication challenge response is an HTTP response sent to a user agent (such as a web browser) by PingAccess, telling the user agent that the corresponding request did not contain a valid authentication token. Some responses also provide instructions to the user agent to obtain a valid authentication token such as an HTTP redirect response containing an encoded OIDC authentication request.

When onboarding new applications to PingAccess, the recommended configuration is `SPA Support = Enabled`, `Request Preservation = POST and Fragment`, and `Fail on Unsupported Content Type = false`, regardless of the behavior of the application. This configuration is displayed in the first table.

Recommended configurations

PingAccess configuration			Request properties			Response characteristics	
SPA Support[1]	**Request Preservation**[2]	**Fail on Unsupported Content Type**[3]	**Method**	**Content Type**	**Accept Header Field**	**Response Code**	**Body Content**
Enabled	POST, POST and Fragment	Any	GET[4]	Any	NOT application/json	401	HTML
Enabled	POST, POST and Fragment	Any	GET[4]	Any	application/json	401	JSON

PingAccess configuration			Request properties			Response characteristics	
Enabled	POST, POST and Fragment	false	POST	Any	NOT application/json	401	HTML
Enabled	POST, POST and Fragment	false	POST	Any	application/json	401	JSON

[1]Configured on an application. In the Admin API, the field is **spaSupportEnabled**. In the UI, the field is **SPA Support**. See *Adding an application* on page 228 for more information about this field.

[2]Configured on a web session. In the Admin API, the field is **requestPreservationType**. In the UI, the field is **Request Preservation**. See *Creating web sessions* on page 300 for more information about this field.

[3]This option is only available through the Admin API.

[4]Any non-POST method receives the same response as a GET.

Additional configurations

PingAccess configuration			Request properties			Response characteristics	
SPA Support[1]	**Request Preservation**[2]	**Fail on Unsupported Content Type**[3]	**Method**	**Content Type**	**Accept Header Field**	**Response Code**	**Body Content**
Disabled	None	Any	Any	Any	Any	302	None
Disabled	POST	Any	GET[4]	Any	Any	302	None
Disabled	POST	Any	POST	application/x-www-form-urlencoded	Any	200	HTML
Disabled	POST	false	POST	NOT application/x-www-form-urlencoded	Any	302	None
Disabled	POST	true	POST	NOT application/x-www-form-urlencoded	Any	415	HTML
Disabled	POST and Fragment	Any	GET[4]	Any	Any	200	HTML

PingAccess configuration			Request properties			Response characteristics	
Disabled	POST and Fragment	Any	POST	application/x-www-form-urlencoded	Any	200	HTML
Disabled	POST and Fragment	false	POST	NOT application/x-www-form-urlencoded	Any	302	None
Disabled	POST and Fragment	true	POST	NOT application/x-www-form-urlencoded	Any	415	HTML
Enabled	None	Any	Any	Any	NOT application/json	401	HTML
Enabled	None	Any	Any	Any	application/json	401	JSON
Enabled	POST, POST and Fragment	true	POST	NOT application/x-www-form-urlencoded	NOT application/json	415	HTML
Enabled	POST, POST and Fragment	true	POST	application/x-www-form-urlencoded	NOT application/json	401	HTML
Enabled	POST, POST and Fragment	true	POST	Any	application/json	401	JSON

Application resources
Application resources are components in an application that require a different level of security. You can manage security settings for application resources.

Resource ordering

Resources have one or more path patterns. When handling requests, PingAccess determines the path pattern that matches and associates the proper resource. When one or more path patterns matches a request, PingAccess uses the first matching pattern it identifies. As such, the order in which path patterns are evaluated is important.

By default, PingAccess orders path patterns automatically so that the most specific patterns are matched first. However, if more explicit control is needed, or if regular expressions are to be used, resource ordering can be enabled to manually specify the order in which path patterns are evaluated.

For example, an application may have three resources, such as:

- `/images/logo.png` (Basic)
- `/images/*` (Basic)
- `/.+/[a-z]\.png` (Regex)

A request to resource `/images/logo.png` is matched by all 3 path patterns, yet each resource can have different policy requirements. Resource ordering allows you to specify which of these path patterns is parsed first, further allowing you to control the policy that is applied to a particular request.

Enable resource ordering

To enable resource ordering:

1. Edit an existing application.
2. On the **Resources** tab, click the **Reorder** icon .
3. Modify the resource order as needed.
4. Click **Save**.

Disable resource ordering

To disable resource ordering, you must first remove any Regex path patterns.

1. Edit an existing application.
2. On the **Resources** tab, click the **Reorder** icon .
3. Click **Disable manual ordering**.

Auto-order resources

When resource ordering is enabled, PingAccess can assist in the process by attempting to intelligently order resources based on their path patterns.

1. Edit an existing application.
2. On the **Resources** tab, click the **Reorder** icon .
3. Click **Auto Order**.
4. Modify the resource order as needed.
5. Click **Save**.

> **Important:**
>
> The **Auto Order** function will reorder all resources for an application. You cannot undo this action, though you are able to re-order resources manually as appropriate.

Configuring resource ordering in PingAccess
Enable and disable resource ordering in PingAccess to control how requests are processed.

About this task

Application resources are defined by one or more path patterns and zero or more query parameters. When handling requests, PingAccess matches the path pattern and query parameters with the proper resource. When more than one resource matches a request, PingAccess uses the first matching resource it identifies.

Resource ordering allows you to specify which resources are checked for a match in what order, letting you control the policy that is applied to a particular request. These instructions describe how to enable and disable resource ordering in PingAccess, as well as how to use the auto-ordering feature.

Note:
Resources can only include query parameters if manual resource ordering is enabled.

Steps

1. To enable resource ordering:
   a. Edit your application.
   b. On the **Resources** tab, click the **Reorder**icon button showing upward and downward arrows (.).
   c. Modify the resource order as necessary.
   d. Click **Save**.
2. To disable resource ordering:
   a. Remove any Regex path pattern.
   b. Edit your application.
   c. On the **Resources** tab, click the button**Reorder** icon showing upward and downward arrows (.
   d. Click **Disable manual ordering**.
3. To auto-order resources:
   a. While editing an application, navigate to the **Resources** tab, click the **Reorder** icon button showing upward and downward arrows (.).
   b. Click **Auto Order**.

      Modify the resource order as necessary.
   c. Click **Save** to keep the changes.

Adding application resources
Add application resources to existing applications in PingAccess.

About this task

An application resource is a component within an application that requires a different level of security. These instructions describe how to add, edit, and delete application resources, as well as how to configure resource ordering, authentication policy, and application type.

There are two resource types: standard and virtual. Standard resources exist on the target destination, and users can be directed to them. Virtual resources exist only in PingAccess. When a user attempts to access them, PingAccess generates a specified response.

Note:

Some applications allow the parameters of a request to be specified in the query string or the POST body. If you are managing such an application, and you are defining its resources using query parameters, use caution when defining the resource so that PingAccess and the application treat the resource in the same way.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand an application you want to modify.
3. Click the **Pencil** icon.

4. Click the **Resources** tab.

> **Note:**
>
> A group containing all global unprotected resources is displayed on the first **Resources** window. Review this list before adding a resource to ensure that there is no conflict between the new resource's path patterns and any unprotected resource path pattern.

5. To add a resource, click **Add Resource**.

> **Info:**
>
> To edit a resource, expand the resource and click the **Pencil** icon. To delete the resource, expand the resource and click the **Delete** icon.

6. Enter a unique **Name** up to 64 characters, including special characters and spaces.
7. Enter a list of URL path patterns, within the context root, that identify this resource.

   If *resource ordering* is enabled, select the path pattern type, **Basic** or **Regex**.

> **Info:**
>
> The path pattern must start with a forward slash (/). It begins after the application context root and extends to the end of the URL.
>
> a. If automatic path pattern evaluation ordering is in use (default), patterns can contain one or more wildcard characters (*). No use of wildcards is assumed. For example, there is a difference between `/app/` and `/app/*`. If a request matches more than one resource, the most specific match is used.
> b. If you enable manual path pattern ordering (resource ordering), the use of regular expressions is permitted. When one or more path patterns contain a regular expression, you cannot revert to automatic path pattern ordering unless that path pattern is removed.
>
>   - If you have specified a regular expression, ensure you select the **Regex** path pattern type. If you don't, the pattern will be interpreted incorrectly as a **Basic** text string.
>   - The application reserved path cannot be used as a path pattern when the context root is `/`. The default application reserved path is `/pa` (`/pa*`). You can modify the default application reserved path using the PingAccess `Admin` API.

8. If you have enabled resource ordering, select an option in the **Query Parameters** section. This option lets you define the resource by query parameters in addition to path patterns.
   Choose from:

   - Select **Match Any** to define the resource without regard to query parameters.
   - Select **Match Specific** to define the resource using one or more query parameters.

     Check **Matches No Parameters** to match the result to the resource if no query parameters are present, as well as if at least one query parameter is present and matches. If this option is deselected, at least one query parameter must be present and must match.

     Enter one or more **Name**-**Value** pairs, or enter a **Name** and check **Any** to match any value for the given name.

9. Select the type of **Resource Authentication**:
   Choose from:

   - **Standard** if the resource requires the same authentication as the root application.
   - **Anonymous** if this resource has no authentication requirements. Identity mappings are still applied if the user is already authenticated. Access Control and Processing rules are applied where applicable.
   - **Unprotected** if this resource has no authentication requirements. Processing rules are applied where applicable. No application or resource access control policy is applied.

   > **Note:**
   >
   > These options are not available for unprotected applications. Web applications types are unprotected when they do not have an associated web session. API applications are unprotected when they are not protected by an authorization server.

10. If the application is a protected Web application with a Web Session, select an **Authentication Challenge Policy** to generate authentication challenge responses for the resource. Click **+ Create** to create a new authentication challenge policy.
11. From the **Methods** list, select one or more methods supported by the resource.

    Leave the asterisk default if the resource supports all HTTP methods, including custom methods.

    Defining methods for a resource allows more fine-grained access control policies on resources. If you have a server optimized for writing data (POST, PUT) and a server optimized for reading data (GET), you might want to segment traffic based on the operation being performed.
12. To log information about the transaction to the audit store, select the **Audit** check box.
13. If the application type is **Web + API**, and **SPA Support** is disabled on the root application, indicate whether the application resource should override the fallback type specified for the main application.

    If you select **Yes** for this option, select the method to be used for the application resource when a request does not contain a web session cookie or OAuth token.

    > **Important:**
    >
    > Carefully consider your configuration when making this selection. Changing the application fallback type can have unexpected effects on resources that do not override the fallback.
    >
    > For example, if you configure a **Web + API** application with a fallback type of **Web** along with several resources that do not override the fallback type, these resources will emit a `401` response (rather than a `302` to PingFederate) if you later change the fallback type to **API** on the main application.
    >
    > The PingAccess runtime uses fallback type to determine which processing flow (Web or API) to use when the request does not contain a web session or an API OAuth Bearer token. When a request does not contain either of these authentication mechanisms, it will rely on this configuration to determine which processing flow to use.

14. To enable the resource, select the **Enabled** check box.
15. From the **Resource Type** list, select a resource type:
    Choose from:

    - **Standard** – This resource exists on the target destination.
    - **Virtual** – This resource only exists in PingAccess. PingAccess generates a response when a user attempts to access the resource.

16. If you selected the **Virtual** resource type, from the **Type** list, select a response generator type: Choose from:

    - **Redirect** – Redirect the user to a new URL with the specified response code.
    - **Template** – Create a response using a specified template.
    - **JSON Identity Mapping** – Make user attributes available to other applications as a JSON payload.
    - **Logout** – End the application web session and optionally redirect the user to a specific landing page after logout. All applications using the same web session are logged out.

    > **Note:**
    >
    > The **Logout** virtual resource type can only be used if PingFederate is the configured token provider.

17. If you selected the **Redirect** response generator, specify the redirect parameters.
    a. In the **Redirect URL** field, enter a relative or absolute URL to which users should be redirected.
    b. From the **Response Code** list, select a response code.

        - **301 – Moved permanently** – This is a permanent redirect that does not require the redirect to maintain the original HTTP method.
        - **302 – Found** – This is a temporary redirect that does not require the redirect to maintain the original HTTP method.
        - **307 – Temporary Redirect** – This is a temporary redirect that requires the redirect to maintain the original HTTP method.
        - **308 – Permanent Redirect** – . This is a permanent redirect that requires the redirect to maintain the original HTTP method.

    c. To opt out of automatic URL encoding, deselect the **Encode URL** check box.

    Learn more in *Release Notes* on page 8.

18. If you selected the **Template** response generator, specify the template parameters.
    a. From the **Media Type** list, select or enter a media type for the template.
    b. In the **Template** field, enter a template in Velocity Template Language (VTL).

       When a user accesses the virtual resource, the template is processed and returned as the response.

       The template can include information about the user, resource, and application according to this data model:

       - `identity.subject` – A string containing the subject name of the identity. This property is only available if the user is authenticated.
       - `identity.attributes` – An object containing user attributes set by the token provider. For example, `identity.attributes.role` could contain a role set by the token provider. This property is only available if the user is authenticated.
       - `identity.trackingID` – A string containing the tracking ID of the identity. This property is only available if the user is authenticated.
       - `resource.name` – A string containing the name of the requested resource.
       - `application.name` – A string containing the name of the requested application
       - `application.realm` – A string containing the OAuth realm associated with the application. If the realm is not defined by the application, it is inferred to be the requested authority and the application's context root.
       - `exchangeId` – A string containing the ID for the current transaction.
    c. From the **Response Code** list, select a response code:
       - 200 – OK
       - 201 – Created
       - 400 – Bad Request
       - 401 – Unauthorized
       - 403 – Forbidden
       - 404 – Not Found
       - 405 – Method Not Allowed
19. If you selected the **JSON Identity Mapping** response generator, select **Inclusion List** or **Exclusion List**.
    Choose from:
    - **Inclusion List** – This option maps the specified attributes to corresponding property names. If you select this option, enter a corresponding **Attribute Name** and **Property Name** on each row. Click **+ Add Row** to add additional rows.
    - **Exclusion List** – This option exposes all attributes except for those you specify. If you select this option, enter zero or more excluded attributes in the **Excluded Attributes** field.
20. If you selected the **Logout** response generator, specify the logout parameters:
    a. Optional: In the **Post-logout Redirect URI** field, enter a URI to which the user is directed after logout. The format of this URI determines the logout behavior.
       - No URI: SLO defaults to the token provider settings.
       - Absolute URL without variables: The PingAccess session is cleared and SLO is not triggered.
       - URL containing the `${SLO}` variable: The `${SLO}` variable is replaced with the PingFederate `ping_end_session_endpoint`, which triggers SLO. For example, if the PingFederate `ping_end_session_endpoint` is `https://pingfederate:9031/idp/startSLO.ping`, a value of `${SLO}?TargetResource=https://example.com` would

direct the user to the PingFederate endpoint, trigger SLO, and then redirect the user to `https://example.com`.
- Relative path: The relative path is appended to the application path to form the destination and SLO isn't triggered.
- PingFederate parameters: The parameters are passed to PingFederate and SLO is triggered. For more information, see *IdP endpoints*.

b. To opt out of automatic URL encoding, deselect the **Encode URL** check box.

Learn more in *Release Notes* on page 8.

21. Click **Save**.

Path patterns reference

PingAccess uses application resource path patterns to match resources. This reference describes the two path pattern types used by PingAccess and how they are processed.

For more flexible resource matching, PingAccess supports two types of path matching patterns:

- Basic
- Regex

To specify a path pattern as Basic or Regex, enable resource ordering. When resource ordering is not enabled, all path patterns are assumed to be Basic, and are parsed as such.

Basic patterns

Basic path patterns (or “wildcard patterns”) are the default path pattern type. Each pattern defines a path to a specific resource or a pattern that matches multiple paths. Basic patterns may contain any number of “*” wildcards, which match zero to many characters in the path.

Example

```
/path/x/*
```

matches any of the following request paths:

```
/path/x/
/path/x/index.html
/path/x/y/z/index.html
```

Regex patterns

Regex path pattern support occurs when you enable resource ordering.

Note:

When one or more Regex path patterns are defined, resource ordering cannot be disabled. You must delete any Regex path pattern entries before you can disable resource ordering.

Regex path patterns allow for more flexibility in resource matching.

Example

```
/[^/]+/[a-z]+\.html
```

matches any of these request paths:

```
/images/gallery.html
```

```
/search/index.html
```

However, it would not match any of these paths:

```
/images/gallery2.html
/search/pages/index.html
/index.html
```

The supported syntax for Regex patterns is documented by the *RE2 wiki*.

Use of Regex path patterns in agent deployments:

- Though Regex path patterns function in an agent deployment, a performance decrease might occur because the agent must consult PingAccess for policy decisions on all Regex path pattern resources.
- In a deployment with Basic path patterns and Resource Ordering disabled, when a PingAccess agent receives a request for a resource, it consults its policy cache for policy decisions.
- Agents are unable to interpret Regex path patterns, so a request to an agent for a resource with a Regex path pattern will result in the agent consulting PingAccess for each policy decision.
- In a resource ordering scenario, the agent stops consulting its policy cache if it reaches a Regex path pattern, and continues this behavior for all resources ordered after the Regex path pattern resource, regardless of their type. Thus, the ordering of resources is critical to performance.

Consider the following scenario.

```
Application A: context root /, resource ordering enabled
Resource 1, Basic, /content
Resource 2, Regex, /\w+-\w+/.*
Root Resource
```

If *Resource 2* is ordered before *Resource 1*, and a request for *Resource 1* is received by the agent, the agent will not leverage its policy cache, since a Regex path pattern disables caching for the associated resource and all resources after it. If *Resource 1* is ordered before *Resource 2*, the agent will leverage its policy cache for requests to *Resource 1*.

The agent is only able to consult the policy cache for basic path pattern resources that are ordered before any Regex path pattern resources. If a basic path pattern resource is ordered after a Regex path pattern resource, the agent will not consult the policy cache, instead contacting PingAccess directly, therefore a performance decrease might occur.

Tip:

If you are using Regex path patterns in an agent deployment, and the order in which resources are ordered is unimportant, order Regex path patterns at the end of the list. If the order is important, place the resource where appropriate to ensure the correct policy is applied at the correct time, while potentially incurring a performance impact.

If your deployment makes extensive use of agents and Regex path patterns, and you are experiencing performance problems, consider redeploying these applications in a proxy configuration where possible.

Applying rules to applications and resources
Apply rules, rule sets, and rule set groups to applications and resources in PingAccess.

About this task

You can apply application access control and request processing rules to applications and their resources. These instructions describe how to create, apply, organize, and remove application rules.

Steps

1. Click **Applications** and then go to **Applications** # **Applications**.
2. Click to expand an application in the list.
3. Click the **Pencil** icon.
4. Optional: Manage the policies for a resource.
   a. Click the **Resources** tab.
   b. Click to expand the resource you want to edit.
   c. Click the **Pencil** icon.
   d. Make the desired changes to the resource.
   e. To confirm your changes, click **Save**.
5. Select the applicable tab.
   Choose from:
   - For Web applications, select the **Web Policy** tab.
   - For API applications, select the **API Policy** tab.
   - For Web + API applications, you can configure both **Web Policy** and **API Policy** on separate tabs, as required.
6. Using the radio selection, filter by ***Rules***, ***Rule Sets***, ***Rule Set Groups***, or **Rule Type**.
7. To create a new rule, click **Create Rule**.
8. To apply a rule, rule set, or rule set group, drag a rule from **Available Rules** onto the policy bar.
9. Drag items to change the order in which they are evaluated at runtime.

   Note:

   Rule ordering can affect PingAccess performance. If an access control rule is more likely to reject access, it should appear near the top of the list to reduce the amount of processing that occurs before that rule is applied. This can be more noticeable if, for example, access control policies are applied along with processing rules. Applying access control policies first ensures that no processing happens on responses unless the user is allowed access.
10. To remove an item from an application or resource, click **-** next to the item you want to remove.

Global unprotected resources

Global unprotected resources are resources that you specify as unprotected for all applications.

To specify a resource as unprotected for a single application, see *Adding application resources* on page 235.

Adding global unprotected resources

Create a new global unprotected resource in PingAccess.

About this task

Warning:

The following steps describe how to globally make resources unprotected. Because any resource captured by the wildcard path of any entry is left unprotected for all applications, you must carefully plan these entries. To make a resource unprotected for a specific application, see *Adding application resources* on page 235.

Steps

1. Click **Applications** and then go to **Applications** # **Global Unprotected Resources**.

2. Click **+ Add Global Unprotected Resource**.
3. In the **Name** field, enter a name for the entry.
4. Optional: In the **Description** field, enter a description for the entry.
5. Optional: If you want to record access requests for this resource in the audit store, select the **Audit** check box.
6. In the **Path Pattern** field, specify the path pattern that identifies the global unprotected resource.

   This entry must start with a forward-slash (/) and can contain one or more wildcard characters (*), such as:

   - `/*.jpg`
   - `/resources/*.css`
   - `/*/resources/favicon.ico`

   > **Note:**
   >
   > Global unprotected resource paths are relative to the application context root. Reserved paths such as `/pa`, `/pa/`, or `/pa/*` are allowed at the global level, but will not be evaluated for applications that are configured with a context root of `/`.

7. To enable the global unprotected resource, select the **Enabled** check box.
8. Click **Save**.

Editing global unprotected resources
Edit an existing global unprotected resource in PingAccess.

About this task

Change global unprotected resources within your application resources in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Global Unprotected Resources**.
2. Click to expand the global unprotected resource you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the global unprotected resource.
5. To confirm your changes, click **Save**.

Deleting global unprotected resources
Delete a global unprotected resource in PingAccess.

About this task

Remove global unprotected resources from your application resources in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Global Unprotected Resources**.
2. Click to expand the global unprotected resource you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Redirects

Redirects reroute an incoming request to another target in PingAccess.

To configure a redirect, you must map the host and port of an incoming request to that of a different target. At runtime, requests made to the source are redirected to the configured target. This feature is useful in redirecting HTTP requests to an equivalent HTTPS URL.

Redirects are not associated with applications, but rather with the source:port combination you specify.

Adding a redirect

Add a new redirect in PingAccess.

About this task

Map the host and port of an incoming request to a different target.

Steps

1. Click **Applications** and then go to **Applications** # **Redirects**.
2. Click **+ Add Redirect**.
3. In the **Source** field, enter the source host and port that you want to redirect.
4. In the **Target** field, enter the target host and post that indicates the destination for the redirect.
5. Optional: To use HTTPS for the request made to the redirect target, select the **Secure Target** check box.
6. In the **Response Code** field, enter the HTTP response code you want to associate with the redirect.

   `301` is specified as the default.
7. To audit redirects, select the **Audit** check box.
8. To confirm your changes, click **Save**.

Editing a redirect

Edit an existing redirect in PingAccess.

About this task

Change the host and port details of a redirect in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Redirects**.
2. Click to expand the redirect you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the redirect.
5. To confirm your changes, click **Save**.

Deleting a redirect

Delete an existing redirect in PingAccess.

About this task

Remove a configured redirect from PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Redirects**.
2. Click to expand the redirect you want to delete.
3. Click the **Delete** icon.

4. To confirm your changes, click **Delete**.

Virtual hosts

Virtual hosts enable PingAccess to protect multiple application domains and hosts.

A virtual host is defined by the host name and host port.

A wildcard (`*`) can be used either to define either any host, such as `*:443`, or any host within a domain, such as `*.example.com`. If a request matches more than one virtual host, the most specific match is used.

> **Note:**
>
> Prior to availability of server name indication (SNI) in Java 8, an HTTPS port could only present a single certificate. To handle multiple virtual hosts, you must use a wildcard name certificate or the *Subject Alternative Name* (SAN) extension. With SNI available, virtual hosts can present different certificates on a single HTTPS port. You can assign which certificates (key pairs) are used by which virtual host from the *HTTPS Listeners* window.

The **Agent Resource Cache TTL** advanced field is used to control PingAccess agent resources for each virtual host.

If you configure a trusted certificate group for a virtual host, or configure an engine key pair to associate it with a virtual host, those settings are used instead of any applicable HTTPS listeners or engine listeners for the virtual host.

Creating new virtual hosts

Create a new virtual host in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click **+ Add Virtual Host**.
3. In the **Host** field, enter the host name for the virtual host, such as myHost.com.

   You can use a wildcard (*) to indicate that any host name is acceptable. A wildcard host can also be specified, such as `*.example.com`.
4. In the **Port** field, enter the port number for the virtual host, such as`1234`.
5. In the **Agent Resource Cache TTL (s)** field, enter the agent resource cache TTL indicating the number of seconds the agent can cache resources for this application.

   This only applies to destination of type `Agent`.
6. To confirm your changes, click **Save**.

Configuring virtual host trusted certificate groups

Configure a virtual host trusted certificate group that can implement client certificate authentication.

About this task

Assigning a trusted certificate group to a virtual host provides a mechanism to authenticate using client certificates during any request to sites using the specified virtual host.

> **Note:**
>
> Trusted certificate groups are applied at the host name level and are independent of the configured port. This means that a mapping to a virtual host of `*.example.com` will apply to requests received on virtual hosts `*.example.com:3000` and `*.example.com:443`.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click to expand the virtual host you want to modify.
3. Click the **Pencil** icon.

   Virtual hosts that have certificate authentication configured will display the message Client Certificate Authentication in the associated bar.
4. In the **Client Certificate Authentication** field, click the **Pencil** icon.
5. From the **Trusted Certificate Group** list, select the appropriate certificate group.

   You can select an existing trusted certificate group, or use one of the following options.

   Choose from:

   - **No Certificate Authentication** – Does not require certificate authentication.
   - **Java Trust Store** – Uses the Java Trust Store for certificate authentication.
   - **Trust Any** – Allows client authentication with any certificate including self-signed certificates.

   If you use the **Trust Any** method in production, you should log client certificates in the audit log.
6. To save the trusted certificate group settings, click **Save**.
7. To confirm your changes, click **Save**.
8. Add the following two *Groovy script rules* to force validation of the server name indication (SNI) and client certificate chain.

   `Validate SNI`

   ```
   if(exc?.getSslData()?.getSniServerNames()?.isEmpty())
   {
     fail();
   }
   else
   {
     pass();
   }
   ```

   `Validate client certificate chain`

   ```
   if(exc?.getSslData()?.getClientCertificateChain()?.isEmpty())
   {
     fail();
   }
   else
   {
     pass();
   }
   ```
9. *Apply these rules* to applications using this virtual host.

Editing virtual hosts
Edit the properties of an existing virtual host in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click to expand the virtual host you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the virtual host.
5. To confirm your changes, click **Save**.

Deleting virtual hosts

Delete an existing virtual host in PingAccess.

Steps

1. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
2. Click to expand the virtual host you want to edit.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Sites

Review information for sites, site authenticators, and third-party services.

Choose from one of the following sub-sections:

Sites operations

Sites are the target applications, endpoints, or APIs which PingAccess Gateway protects and to which authorized client requests are forwarded.

Choose from one of the following sections:

Adding sites

Add sites in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Sites**.
2. Click **+ Add Site**.
3. Complete the fields.

   For more information about the site fields, see *Site field descriptions* on page 248.
4. To configure advanced settings, click **Show Advanced**.
5. Click **Save**.

> **Note:**
>
> If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

Editing sites

Edit the properties of existing sites in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Sites**.
2. Click to expand the site you want to edit.
3. Click the **Pencil** icon.

4. Make the desired edits to the site.
5. To confirm your changes, click **Save**.

> **Note:**
>
> If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

Deleting sites

Delete an existing site in PingAccess.

Steps

1. Go to **Applications** # **Sites** # **Sites**.
2. Click to expand the site you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.
5. To confirm your changes, click **Delete**.

Site field descriptions

The following table describes the fields available for managing applications on the **Sites** window.

Field	Required	Description
Name	Yes	Enter a unique **Site Name**, up to 64 characters, including special characters and spaces.
Targets	Yes	Specify one or more **Targets**. The format for this is `hostname:port`, such as `www.example.com:80`.
Secure	Yes	Select **Secure** if the site is expecting HTTPS connections. If the site is configured for **Secure** connections, select a **Trusted Certificate Group** from the list, or select **Trust Any** to trust any certificate presented by the listed targets.
Site Authenticators	No	If the site requires the use of site authenticators, select one or more authenticators from the list. Click **+ Create** to create a site authenticator. Click **x** to remove a site authenticator.
Use Target Host Header	No	Select the check box to have PingAccess modify the `Host` header for the site's target host and target port rather than the virtual host configured in the application. **Note:** When cleared, PingAccess makes no changes to the `Host` header. This is often required by target web servers to ensure they service the HTTP request with the correct internal virtual server definition.
Skip Hostname Verification	No	Select this check box if the site should not perform hostname verification of the certificate.

<table>
<tr><th>Field</th><th>Required</th><th>Description</th></tr>
<tr><td>Expected Certificate Hostname</td><td>No</td><td>If you have not selected to skip host name verification, enter the name of the host expected in the certificate in the Expected Certificate Hostname field. This field is available only if the Skip Hostname Verification check box is not selected. If left blank, the certificates are verified using the target host names.</td></tr>
<tr><td>Availability Profile</td><td>No</td><td>Select an Availability profile. To create a new availability profile, click + Create Availability Profile.</td></tr>
<tr><td>Load Balancing Strategy</td><td>No</td><td>If the site contains more than one target, select a Load balancing strategy. To create a new load balancing strategy, click + Create Load Balancing Strategy.</td></tr>
<tr><td>Send Token</td><td>No</td><td>If your site uses the identity information in the PingAccess Token or OAuth access token, leave this check box selected to include the token in the request to the backend site.

If you do not need the token information, you can clear the check box to remove the PingAccess Token from the request. This excludes unnecessary information and decreases the payload size, which might improve performance.</td></tr>
<tr><td>Maximum Connections</td><td>Yes</td><td>Enter the maximum number of HTTP persistent connections you want PingAccess to have open and maintain for the site. The default of <code>-1</code> indicates unlimited connections.</td></tr>
<tr><td>Maximum Websocket Connections</td><td>Yes</td><td>If the number of WebSocket connections needs to be limited, enter a value. The default of <code>-1</code> indicates no limit.</td></tr>
<tr><td>Use Proxy</td><td>No</td><td>Select if requests to the site should use a configured proxy.

Note:

If the node is not configured with a proxy, requests are made directly to the site.

CAUTION:

If your proxy uses availability handling to retry multiple targets in the event of a network problem, you should configure PingAccess to use only one target for the site. Unexpected behavior can occur if PingAccess and the proxy are both configured to perform availability handling.</td></tr>
<tr><td>Keep Alive Timeout</td><td>No</td><td>The time, in milliseconds, that a pooled connection to the site can be idle before PingAccess closes the connection and removes it from the pool. The default of <code>0</code> indicates no timeout.</td></tr>
</table>

Site authenticators

Site authenticators define the authentication mechanism that target sites require to control access.

Choose from one of the following sections:

- *Adding site authenticators* on page 250
- *Editing site authenticators* on page 250
- *Deleting site authenticators* on page 250

Adding site authenticators

Create a new site authenticator in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Site Authenticators**.
2. Click **+ Add Site Authenticator**.
3. In the **Name** field, enter a unique name.

> **Note:**
>
> Special characters and spaces are allowed. This name appears in the **Site Authenticator** list on the **New Site** tab.

4. Select the type of authentication from the **Site Authenticator** list.
5. To continue, select one of the following authentication types:
   - *Basic authentication site authenticator*
   - *Mutual TLS site authenticator*
   - *Token mediator site authenticators* on page 251

Editing site authenticators

Edit the properties of an existing site authenticator.

Steps

1. Click **Applications** and then go to **Sites** # **Site Authenticators**.
2. Click to expand the site authenticator you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the site authenticator.
5. To confirm your changes, click **Save**.

Deleting site authenticators

Delete existing site authenticators in PingAccess.

About this task

You cannot delete site authenticators if they are associated with a site.

Steps

1. Click **Applications** and then go to **Sites** # **Site Authenticators**.
2. Click to expand the site authenticator you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Basic authentication site authenticators

Use HTTP Basic authentication (user name:password) to authenticate a client requesting access to a site that requires basic authentication.

Info:
Obtain the user name and password from your target site provider.

Field	Description
Username	The user name required for access to the protected site.
Password	The password required for access to the protected site.

Mutual TLS site authenticators

Use key pairs to authenticate PingAccess to a target site. When initiating communication, PingAccess presents the client certificate from a key pair to the site during the mutual TLS transaction.

The site must trust this certificate for authentication to succeed.

Tip:

Several steps are required for PingAccess certificate management before configuring the mutual TLS site authenticator.

Field	Description
Key Pair	The imported or generated key pair for client authentication. Select the key pair you want to use to authenticate PingAccess to the target site. To create a key pair, see *Importing existing key pairs* on page 312 or *Generating new key pairs* on page 313.

Token mediator site authenticators

The token mediator site authenticator uses the PingFederate security token service (STS) to exchange a PingAccess token for a security token, such as a Web Access Management (WAM) token or OpenToken, that is valid at the target site.

Field	Description
Token Generator ID	Defines the **Instance Name** of the token generator that you want to use. The token generator is configured in PingFederate. For more information, see the PingFederate documentation. If *PingFederate Administration* is configured, and PingFederate has one or more token generators configured, this field becomes a list of available token generator IDs.
Logged In Cookie Name	Defines the cookie name containing the token that the target site is expecting.

Field	Description
Logged Off Cookie Name	Defines the cookie name that the target site responds with in the event of an invalid or expired token. If the PingAccess token is still valid, PingAccess re-obtains a valid WAM token and makes the request to the site again. If the site responds with the cookie set as logged off again, PingAccess responds to the client with an `access denied` message.
Logged Off Cookie Value	Defines the value placed in the **Logged Off** cookie to detect an invalid/expired WAM token event.
Send Cookies to Browser	Allows the token mediator to send the back end cookie defined in the **Logged In Cookie Name** field back to the browser if the protected application has updated it. If the set-cookie header isn't in the response from the protected site, and the token mediator site authenticator has a cached token for that session, the token mediator site authenticator will create a new set-cookie response header based on the **Cookie Domain**, **Cookie Max Age**, **HTTP-Only Cookie** and **Secure Cookie** fields in the UI. The administrator now can direct the token mediator site authenticator to actively return cookies to the user's browser, even when the protected site isn't doing that. This is used to enable a seamless single sign-on (SSO) experience for users navigating from PingAccess protected applications to those protected by a third-party WAM system.
Cookie Domain	Enter the domain of the logged-in cookie.
Cookie Max Age	Define the length of time in minutes, that you want the generated logged-in cookie to be valid.
HTTP-Only Cookie	Define the logged-in cookie as HTTP-Only. An HTTP-only cookie is not accessible using non-HTTP methods, such as calls through JavaScript, such as referencing `document.cookie`.
Secure Cookie	Indicate whether the generated logged-in cookie must be sent using only HTTPS connections.

Field	Description
Token Processor ID	Defines the instance name of a token processor that you want to use. The token processor is configured in PingFederate. Specify this value if more than one instance of either the JSON web token (JWT) processor or the OAuth bearer access token processor is defined in PingFederate. If *PingFederate Administration* is configured, and PingFederate has one or more token processors configured, this field becomes a list of available token processor IDs.

Third-party services

A third-party service configuration defines the destination for HTTPS outbound calls. These definitions are used by custom plugins to indicate how the HTTP client will communicate with the destination.

The configuration of a third-party is similar to that of a site. Choose from one of the following topics:

- *Adding third-party services* on page 253
- *Editing third-party services* on page 253
- *Deleting third-party services* on page 253

Adding third-party services

Add new third-party services in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Third Party Services**.
2. Click **+ Add Third-Party Service**.
3. Complete the fields.

   For information about completing the fields, see *Third-Party Service Field Descriptions*.
4. Click **Save**.

Editing third-party services

Edit the properties of existing third-party services in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Third Party Services**.
2. Click to expand the third-party service you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the third-party service.
5. To confirm your changes, click **Save**.

Deleting third-party services

Delete existing third-party services in PingAccess.

Steps

1. Click **Applications** and then go to **Sites** # **Third Party Services**.
2. Click to expand the third-party service you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Third-party service field descriptions

The following table describes the fields available for managing applications at **Sites** # **Third-Party Services** in PingAccess.

Field	Required	Description
Name	Yes	Specify a name that identifies the third-party service.
Targets	Yes	Specify one or more host name:port pairs used to reach the third-party service.
Secure	No	Indicate whether or not the target is expecting a secure connection.
Host Value	No	An optional value used as the host header field value used in requests to a third-party service regardless of the target used.
Skip Host name Verification	No	For secure connections, select to indicate that the third-party service should not perform host name verification of the certificate.
Expected Certificate Host name	No	For secure connections, enter the name of the host expected in the certificate when host name verification is enabled.
Availability Profile	Yes	Indicate the availability profile to use. To create a new availability profile, click **+ Create Availability Profile**.
Load Balancing Strategy	No	Select the load-balancing strategy to use if more than one target is defined.
Maximum Connections	Yes	Indicates the maximum number of HTTP-persistent connections PingAccess will open and maintain for the service. The default of `-1` indicates unlimited connections.
Use Proxy	No	Indicates that requests to the site should use a configured proxy.

Agents

You can manage agents in PingAccess. Agents are web server plugins installed on the web server hosting the target application. Agents intercept client requests to protected applications and allow or deny the request to proceed by consulting the policy manager or using cached information.

Agents communicate with the PingAccess policy server through the PingAccess Agent Protocol (PAAP), which defines the possible interactions between agents and the policy server. Agents have identifying names and a shared secret to authenticate with the policy server. Agents do not need to be unique. There can be multiple agents using the same name and secret, and they are all treated equally by the policy server. This is useful in complex deployments where unique agents are difficult to manage. Agents can be assigned as the destination for one or more applications by name.

Assigning agent listener key pairs

Before you create an agent, import or create an agent listener key pair and assign it to the agent listener.

Steps

1. Import or generate a key pair.

   The key pair's subject or subject alternative names list needs to include the host or hosts the agent will use to contact the PingAccess policy server.
2. Click **Security** and then go to **Key Pairs**.
3. Click the **Pencil** icon.
4. Click **Assign HTTPS Listener** for the key pair.

5. Select the **Agent** check box.
6. Click **Save**.

> **Info:**
>
> If the environment is clustered, check the `pingaccess.log` file on each engine to ensure replication completed before restarting each engine.

Adding agents

Create new agents in PingAccess.

Steps

1. Click **Applications** and then go to **Agents**.
2. Click **+ Add Agent**.
3. Complete the fields.

   For more information about the fields, see *Agent Field Descriptions*.
4. To configure advanced settings, click **Show Advanced**.
5. To save the configuration and download the `<agent-name>_agent.properties` file for use with the PingAccess agent, click **Save & Download**..

> **Note:**
>
> The shared secret is generated by the PingAccess server and identified on this page with a timestamp. You can delete existing secrets by clicking **Remove** in the **secret** field. If an additional secret is needed, *edit* the agent and click **Save & Download** to generate and download a new shared secret.

   PingAccess can generate additional agent `agent.properties` files containing the specified information that can configure the agent plugin. Existing configurations can also be re-downloaded if necessary.

Editing agents

Edit existing agents in PingAccess.

Steps

1. Click **Applications** and then go to **Agents**.
2. Click to expand an existing agent you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits.
   a. To download the shared secret, click **Download**.
   b. To remove the shared secret, click **Remove**.
5. Click **Save & Download**.

Deleting agents

Delete existing agents in PingAccess.

Steps

1. Click **Applications** and then go to **Agents**.
2. Click to expand an existing agent you want to delete.
3. Click the **Delete** icon.

4. To confirm your changes, click **Delete**.

Agent field descriptions

The following table describes the fields available for managing applications in the **Agents** window.

Field	Required	Description
Name	Yes	Enter a unique alphanumeric name for the agent, up to 64 characters.
Description	No	Enter an optional description for the agent and its purpose.
PingAccess Host	Yes	In the **PingAccess Host** fields, enter the **Hostname** and **Port** of the PingAccess server where the agent should send requests. **Info:** The PingAccess **Hostname** and **Port** might not be the actual host and port to which that policy server is listening, depending on network routing configuration and network elements such as reverse proxies and load balancers. The PingAccess **Host** and PingAccess **Port** are where the agent sends its requests. For example, if you have a cluster of engines behind a load balancer, the PingAccess **Host** and PingAccess **Port** values might point to the load balancer, rather than directly to an engine host in order to provide fault tolerance for the agent connectivity.
Failover Host	No	In the **Failover Host** fields, enter the **Hostname** and **Port** of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess **Host**. **Tip:** Additional failover hosts can be added using API. For more information, see the *PingAccess API Management Guide.*
Agent Trusted Certificate	Yes	Specify the **Agent Trusted Certificate** to export in the agent properties file. The agent uses the selected certificate to communicate with the PingAccess engine using SSL/TLS. PingAccess gathers these certificates from imported certificates. If the appropriate certificate is not available, it needs to be *imported into the system*. **Note:** You must specify the certificate authority (CA) root certificate if the agent listener presents a CA-signed certificate chain.

Field	Required	Description
Override Request IP Source Configuration	No	If required, select **Yes** to **Override Request IP Source Configuration** and enable additional controls that configure the agent to use different IP source information. 1. Enter the header names used to identify the source IP address. 2. If more than one value is included in the **Header Names** field, use **List Value Location** to specify whether the first value or the last value in the list is used as the source address. The default value is **Last**. 3. Select **Fall Back to Last Hop IP** to use the last hop IP address as the source address when none of the listed header names are found. When this option is not selected, if none of the listed header names are found, access is denied and a `Forbidden` result is returned.
Override Unknown Resource Configuration	No	If required, select **Yes** to **Override Unknown Resource Configuration** to specify how requests for unknown resources are handled. This mode is optional. If not set, the default agent mode will be used. Select a **Mode** to specify how requests for unknown resources are handled, either **Deny** or **Pass-Through**.
Max Retries	Yes	Enter the number for **Max Retries** before considering a PingAccess server unavailable.
Failed Retry Timeout	Yes	Enter the number, in seconds, for the **Failed Retry Timeout** before retrying a failed PingAccess server.

Sideband Clients

Use sideband clients to query the PingAccess Sideband API for authorization and request/response modification for API gateways. This lets PingAccess supply access decisions to gateway tools.

This section covers the following:

- *Adding sideband clients* on page 257
- *Editing sideband clients* on page 258
- *Deleting sideband clients* on page 258

Adding sideband clients

Create new sideband clients in PingAccess to let them query PingAccess for access decisions.

Steps

1. Click **Applications** and then go to **Sideband Clients**.
2. Click **+ Add Sideband Clients**.
3. In the **Name** field, enter a name for the sideband client.
4. Optional: In the **Description** field, enter a description for the sideband client.

5. In the **Secrets** section, add one or more secrets:
   a. Click **+ Add**.

      Result: The **New Secret** pane displays.
   b. To copy the new secret to your clipboard, click **Copy** and save it in a secure location.

      Note:

      Your sideband integration must use this secret as the header field value in the specified request header.
   c. Click **Done**.
   d. Optional: Repeat steps 5.a through 5.c to create additional secrets.
6. Click **Save**.

Editing sideband clients

Edit existing sideband clients in PingAccess to update their characteristics.

Steps

1. Click **Applications** and then go to **Sideband Clients**.
2. Click the **Expand** icon to expand an existing sideband client you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits.
5. Click **Save**.

Deleting sideband clients

Delete existing sideband clients in PingAccess to remove a client's access.

Steps

1. Click **Applications** and then go to **Sideband Clients**.
2. Click to expand an existing sideband client you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Access header

The **Access** header contains options related to access control, such as rules, web sessions, and token validation.

The **Access** header contains these menu options:

- *Rules* on page 259
- *Authentication* on page 291
- *Identity mappings* on page 295
- *Web sessions* on page 298
- *Token validation* on page 306
- *Unknown resources* on page 308

Rules

The PingAccess policy manager contains controls for adding and managing rules. Use rules to specify who can access your applications and resources, how and when they can do so, and what modifications should be made to the requested content.

The policy manager is an interface in the PingAccess administrative console where you can create rules, *rule sets*, and *rule set groups*, and apply them to *applications* and *application resources*. Policies are the rules, rule sets, or groups of rule sets applied to a specific application and its resources. Policies define how and when a client can access target sites.

When a client attempts to access an application resource identified in one of the policy's rules, rule sets, or rule set groups, PingAccess uses the information within the policy to decide whether the client can access the application resource and whether any additional actions need to occur before granting that access.

For information on how to assign rules, rule sets, and rule set groups, see *applying rules to applications and resources*.

Rule types

Access control rules

Access control rules can restrict access in a number of ways. For example, an access control rule might:

- Test user attributes (for more information, see *OAuth attribute rules*)
- Check the time of day the request was made at (for more information, see *time range rules*
- Request [MISSING glossAcronymn or glossSurfaceform for gIP] addresses (for more information, see *network range rules*)
- Test *[[]]* access token scopes (for more information, see *OAuth scope rules*)

> **Tip:**
>
> Ensure that any headers used in access control rules, such as the `X-Forwarded-For` header that network range rules use, are sanitized and managed exclusively by inline infrastructure that users must be routed through before reaching PingAccess and the protected applications.

Processing rules

Processing rules can perform request processing. For example, a processing rule might:

- Modify headers (for more information, see *rewrite response header rules*)
- Rewrite URLs (for more information, see *rewrite URL rules*)

Processing order

Access control rules are applied before processing rules. For each type of rule, the rules are applied in the order configured in the policy manager. All rules are evaluated after *identity mappings* are, so that the rules have access to the **request header** field set by the identity mapping.

If rules for an application and rules for a resource both apply to a request, PingAccess applies the rules in the following order:

1. Application access control rules
2. Resource access control rules
3. Resource processing rules
4. Application processing rules

Agent deployments

The following rules aren't available for agent deployments:

- *Adding PingAuthorize access control rules* on page 271
- *Adding PingAuthorize response filtering rules* on page 279

- *Rewrite content rules*
- *Rewrite cookie domain rules*
- *Rewrite cookie path rules*
- *Rewrite response header rules*
- *Rewrite URL rules*

Rule Management

You can create and manage rules to control access to your web apps and APIs. PingAccess supports a variety of rule types, and the procedures for rule creation are different for each rule type.

Choose from one of the following topics:

-
    - *Creating access control rules* on page 260
    - *Creating processing rules* on page 278
    - *Editing rules* on page 287
    - *Deleting rules* on page 287

Creating access control rules

Create access control rules to control how and when users can access sites and APIs.

For more information about access control rules in PingAccess, see the following topics:

- *Adding an authentication requirements rule* on page 260
- *Adding Groovy script rules* on page 261
- *Adding HTTP request header rules* on page 262
- *Adding HTTP request parameter rules* on page 264
- *Adding network range rules* on page 265
- *Adding OAuth attribute rules* on page 266
- *Adding OAuth client rules* on page 267
- *Adding OAuth Groovy script rules* on page 268
- *Adding OAuth scope rules* on page 269
- *Adding one-time authorization rules* on page 270
- *Adding PingAuthorize access control rules* on page 271
- *Adding rate limiting rules* on page 272
- *Adding redirect rules* on page 273
- *Adding rejection rules* on page 274
- *Adding time range rules* on page 274
- *Adding web session attribute rules* on page 275
- *Adding web session scope rules* on page 276
- *Adding WebSocket handshake rules* on page 277

Adding an authentication requirements rule

Add an authentication requirements rule in PingAccess to limit access to resources or applications protected by PingAccess based on the access control rule (ACR) values returned by the PingFederate request AuthN context authentication selector.

Before you begin

Verify that you have:

- A PingFederate configuration that uses the `Requested AuthN Context Authentication Selector`

- A configured authentication list

About this task

An authentication requirements rule allows authentication requirements to be applied when a policy decision is being made by the PingAccess engine, allowing an entire application or individual resources to require a particular authentication type.

This rule also allows for configurations that require more secure authentication methods, such as multi-factor authentication (MFA). For example, a website might allow a user to authenticate and view personal data using only a user name and password, but editing their personal data could require an additional PingID verification step. When used in this manner, an additional step-up authentication event is automatically triggered.

> **Important:**
>
> To ensure that step-up authentication is triggered, this rule should always be positioned first in a list of rules, rule sets, or rule set groups, regardless of whether the criteria is `Any` or `All`.
>
> PingAccess uses rules to trigger different authentication paths in PingFederate. If the authentication requirements rule isn't the first item in a list, then it isn't sent to PingFederate in the initial request.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Authentication Requirements**.
5. Select an *Authentication Requirements List*.
6. Select a **Minimum Authentication Requirement**.

   > **Note:**
   >
   > The possible values for the **Minimum Authentication Requirement** are derived from the selected Authentication Requirements list.
7. Click **Save**.

Adding Groovy script rules

Add a Groovy script rule to provide advanced rule logic that extends PingAccess rule development beyond the capabilities of the packaged *Policy Manager* rules.

About this task

> **Note:**
>
> Through Groovy scripts, PingAccess administrators can perform sensitive operations that might affect system behavior and security. Since the regular Groovy rule and the OAuth Groovy rule differ in the scope of their functionality, the relevant rules are tagged for Web App or for API, respectively, in the rules list.

For more information about error handling, see *Advanced Fields*.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Groovy Script (for Web App)**.
5. Enter the Groovy script to use for rule evaluation.

   Example:

   To create an OAuth scope rule that matches more than one scope, your Groovy script might contain: `hasScopes("access","portfolio")`.
6. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
   Choose from:

   - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
   - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

     a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
     b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
     c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
     d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

   If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:
7. Click **Save**.

Adding HTTP request header rules
Add an HTTP request header rule to examine a request and determine whether to grant access to a requested resource based on a match found in one of the specified headers in the HTTP request.

About this task

If more than one **Field** and **Value** pair is listed, then all conditions must match in order for the rule to succeed.

Note:

For more information about error handling, see *Advanced Fields*.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.

3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **HTTP Request Header**.
5. In the **Field** column, in the **Header** field, enter a header name you want to match to grant or not grant the client access.
6. In the **Value** field, enter a value for the header you want to match in order to grant or not grant the client access.

   The wildcard (*) character is supported.

> **Tip:**
>
> If you want to match on the `Host` header, include both the host and port in the **Value** field, or add a wildcard after the hostname (`host*` or `host:*`) to match the HTTP request.

7. If additional header pairs are needed, click **Add Row** to add an additional row, then repeat steps 5-6.
8. If the values should be an exact match to the value case, select the **Case Sensitive** check box.
9. If access is not allowed when a match is found, select the **Negate** check box.

> **Info:**
>
> Ensure that the attribute name entered in the **Field** field is spelled correctly and exists. If you enter an attribute that does not exist and you select **Negate**, the rule will always succeed. The **Negate** control applies to the entire set of conditions specified, and passes the rule if any condition is not met.

10. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
    Choose from:

    - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
    - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

      a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
      b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
      c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
      d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

    If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:
11. Click **Save**.

Adding HTTP request parameter rules

Add an HTTP request parameter rule to examine a request and determine whether to grant access to a requested resource based on a match found in specified form parameters of the HTTP request.

About this task

Add an HTTP request parameter rule determines if the parameters are passed as part of the URL query string parameters or as part of a request body submitted using an HTTP PUT or POST method. If the request is a POST request, the `content-type` must be set to `application/x-www-form-urlencoded` to process the field names in the request.

If this rule is applied to an agent configuration, only URL query string parameters are compared, because the agent does not receive the request body for processing.

If more than one **Field** and **Value** pair is listed, then all conditions must match for the rule to succeed.

> **Note:**
>
> For more information about error handling, see *Advanced Fields*.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **HTTP Request Parameter**.
5. In the **Field** column, in the **Parameter** field, enter a parameter name you want to match to grant or not grant the client access.
6. In the **Value** field, enter a value for the parameter you want to match in order to grant or deny the client access.

   The wildcard (*) character is supported.

   > **Note:**
   >
   > Values entered here will be URL-encoded prior to the comparison. For example, if the value specified in the **Value** field is `v1 v2`, when the engine performs the comparison, this value will convert to `v1%20v2` before the search is performed.
7. If additional parameters pairs are needed, click **Add Row** to add an additional row, then repeat steps 5-6.
8. If the values should be an exact match to the value case, select the **Case Sensitive** check box.
9. If access is not allowed when a match is found, select the **Negate** check box.

   > **Info:**
   >
   > Ensure that the field name you enter is spelled correctly and exists. If you enter a field name that does not exist and you select **Negate**, the rule will always succeed. The **Negate** control applies to the entire set of conditions specified, and passes the rule if any condition is not met.

10. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
    Choose from:
    - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
    - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:
        a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
        b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
        c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
        d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

    If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:
11. Click **Save**.

Adding network range rules
Add a network range rule to examine a request and determine whether to grant access to a target site based on whether the IP address falls within a specified range, using Classless Inter-Domain Routing notation.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Network Range**.
5. In the **Network Range** field, enter a network range value, such as `127.0.0.1/8`.

   PingAccess supports IPv4 addresses.
6. Select **Negate** if when a match is found, access is not allowed.

7. If you want to override source address handling defined in the **HTTP Requests** configuration, click **Show Advanced Settings** and perform the following steps:
   a. Click **Override Request IP Source Configuration**.
   b. In the **Headers** field, enter the headers used to define the source IP address to use.
   c. Select the **Header Value Location** to use when multiple addresses are present in the specified header.

      Valid values are `Last` (the default) and `First`.
   d. Click **Fall Back to Last Hop IP** to determine if, when the specified **Headers** are not present, PingAccess should return a `Forbidden` result or if it should use the address of the previous hop as the source to make policy decisions.
   e. Optional: To configure rejection handling, select a rejection handling method:

      If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.

      If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

      1. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
      2. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
      3. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `PA_HOME/conf/template/` directory.
      4. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.
8. Click **Save**.

Adding OAuth attribute rules

Add an OAuth attribute rule to examine a request and determine whether to grant access to a target service based on a match found between the attributes associated with an OAuth access token and attribute values specified in the rule.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **OAuth Attribute**.
5. From the **Attribute Name** list, select the attribute name you want to match to an attribute associated with an OAuth access token.
6. In the **Attribute Value** field, enter the value to match.

> **Note:**
>
> The attribute values come from the contract in your OAuth access token manager in PingFederate. For more information, see *Defining the Access Token Attribute Contract*.

7. Add additional rows of attribute name and value pairs as needed.

> **Note:**
>
> If multiple rows are included here, all conditions must match for the rule to match.

8. Select **Negate** if, when a match is found, access is not allowed.

> **Info:**
>
> Verify what you enter for the attribute. If you enter an attribute that does not exist, such as if the attribute is misspelled, and you select **Negate**, the rule will always succeed.

9. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
   Choose from:

   - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
   - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

     a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
     b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
     c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
     d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

   If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:
10. Click **Save**.

Adding OAuth client rules
Add an OAuth client rule to restrict access to API applications based on one or more OAuth client IDs.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **OAuth Client**.
5. In the **Client IDs** section, enter one or more Client IDs that allow access. To add additional fields, click **+ New Value**.

6. Optional: If you want to configure rejection handling, click **Show Advanced Settings**, and then from the **Rejection Handler** list, select an existing rejection handler that defines whether to display an error template or redirect to a URL.

   > **Note:**
   >
   > You can include information about missing Client IDs in the rejection response using the `$info` variable.
   >
   > For example, if you are using the Default API rejection handler, you could edit the `<PA_HOME>/conf/template/oauth.error.json` file and change this line:
   > `{"$Encode.forJavaScriptSource($header)":""}`
   >
   > to
   >
   > `{"$Encode.forJavaScriptSource($header)":"#if($info)$Encode.forJavaScriptSource($info)`

7. Click **Save**.

Adding OAuth Groovy script rules

Add an OAuth Groovy script rule to determine whether to grant access to a target site based on the results returned from a Groovy script that evaluates request details and OAuth details.

About this task

Adding an OAuth Groovy script rule allows you to create more sophisticated OAuth scope and OAuth attribute value rules for API applications.

> **Info:**
>
> Since the regular Groovy rule and the OAuth Groovy rule differ in the scope of their functionality, the relevant rules are tagged for Web App or for API, respectively, in the rules list.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **OAuth Groovy Script (for API)**.
5. In the **Groovy Script** field, enter the Groovy script to use for rule evaluation.
6. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
   Choose from:

   - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
   - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates

you can modify with your own branding or other information. If you select **Basic**, provide this information:

a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

7. Click **Save**.

Adding OAuth scope rules

Add an OAuth scope rule to examine the contents of the PingFederate validation response and determine whether to grant access to a backend target site based on a match found between the scopes of the validation response and scope specified in the rule.

About this task

For example, a resource might require that the OAuth access token contain the scope `superuser`.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **OAuth Scope**.
5. From the **Scope** list, select the scope you want to match to values returned from the access token.

   > **Info:**
   >
   > This is one scope requirement in the set of scopes associated with the access token.
6. Select **Negate** if, when a match is found, access is not allowed.
7. Optional: To configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:
   Choose from:

   - If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
   - If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end user's browser if rule evaluation fails. This page is among the templates

you can modify with your own branding or other information. If you select **Basic**, provide this information:

a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.

If you select **Basic**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

8. Click **Save**.

Adding one-time authorization rules

Add a one-time authorization rule to let the user obtain authorization for a mobile app or single-page application using the Client-Initiated Back-channel Authentication (CIBA) specification.

Before you begin

You must have a configured token provider and an OAuth client with the CIBA grant type enabled.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **One-Time Authorization**.
5. In the **Client ID** field, enter the Client ID of the OAuth client.
6. Select a **Client Credentials Type**, then provide the information required for the selected credential type.
   Choose from:
   - **Secret** – In the **Client Secret** field, enter the secret used by the OAuth client to authenticate to the authorization server.
   - **Mutual TLS** – From the **Mutual TLS** list, select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – Select this option to use Private Key JSON web token (JWT). No additional information is required.
7. From the **Login Hint Request Attribute** list, select an attribute.

   When a user authenticates, the value of this attribute is included in the call to the token provider. This attribute value can identify the user.
8. Optional: In the **Scopes** field, enter or select a scope to request from the token provider. The `openid` scope is automatically requested.
   a. Optional: Click **+ New Value** to add additional fields.

9. Optional: Click **Show Advanced** to configure advanced options:
   a. Optional: In the **Requested Expiry (S)** field, enter the transaction lifetime in seconds.

      If not specified, the value defined in the CIBA request policy is used.
   b. Optional: From the **Timeout Rejection Handler** list, select the handler to use for an expired request.
   c. Optional: From the **Deny Rejection Handler** list, select the handler to use for a denied request.
10. Click **Save**.

Adding PingAuthorize access control rules

Add an access control rule to contact PingAuthorize for access information.

Before you begin

- Create a third-party service with PingAuthorize configured as the target. For more information, see *Adding third-party services* on page 253.
- Confirm that you are not using the *agent* model. PingAuthorize access control rules aren't available for agent deployments.

About this task

An access control rule can grant or deny access, and can modify the request, based on the response from the PingAuthorize request [MISSING glossAcronymn or glossSurfaceform for gAPI].

Important:

The PingAuthorize sideband API cannot accept gzipped data from upstream server responses. Ensure that upstream server requests add or replace the `Accept-Encoding` header with `Accept-Encoding: identity` to prevent the upstream server from sending compressed responses.

To add a PingAuthorize access control rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. In the **Type** list, select **PingAuthorize Access Control**.
5. In the **Third Party Service** list, select your PingAuthorize service.
6. In the **Shared Secret** field, enter the shared secret from PingAuthorize.
7. Optional: To include access token data in the request to PingAuthorize, select the **Include Identity Attributes** check box.

   This option is selected by default.
8. Optional: To configure advanced options, click **Show Advanced**.
   a. Optional: In the **Sideband Endpoint** field, enter the sideband API endpoint location.
   b. Optional: In the **Shared secret header name** field, enter a header in which to send the shared secret.
9. Click **Save**.

Adding rate limiting rules

Add a rate limiting rule to limit a client from overloading the server with too many requests in a specified period of time.

About this task

The implementation of this rule uses a token bucket to control the number of incoming requests.

The configuration defines a number of requests and an interval that must elapse between requests. The allowed number of requests within the time window is controlled by the **Max Burst Requests** setting, which is visible when you click **Show Advanced**. For example, if the **Max Burst Requests** value is `1`, two requests are allowed in the request interval — one normal request, and one burst request.

If a request was not received, the number of allowed requests is incremented by one at the end of each **Request Interval**. This continues until the number of allowed requests equals the value defined by the **Max Burst Requests** setting.

Note:

Using the rate limiting rule in a clustered PingAccess environment can impose stricter clock synchronization requirements for requests processed by multiple engine nodes. Alternatively, you can configure a load balancer sitting in front of a PingAccess cluster to stick the session to a specific engine, ensuring that the rate limiting rule is applied by a single PingAccess engine node.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Rate Limiting**.
5. Select a **Policy Granularity**, as defined in the following table.

Policy Granularity	Definition
Resource	Restricts the rate of requests based on the resource requested.
Identity	Restricts the rate of requests to the identity associated with the current authentication token, such as a PingAccess cookie or an OAuth token. This is the default value.
IP	Restricts the number of requests based on the source IP address. The IP address used to apply this policy comes from the HTTP Requests IP Source configuration options, or options that override that configuration, if those options are configured.
OAuth Client	Restricts the number of requests to all OAuth tokens obtained by a specific **Client ID**.

6. Enter, in milliseconds, a **Request Interval**.

7. Optional: Click **Show Advanced Settings** to configure advanced options:
   a. If more than `1` request should be allowed a request interval, enter the number of requests to allow in the **Max Burst Requests** field.

      Note:

      PingAccess increases the number of available requests only after a request interval that serves no requests to the client. As a result, in the period following a cycle where the remaining allowed burst requests is reduced to `0`, no burst requests are allowed, regardless of this setting.

   b. If PingAccess should reply to the client with a `Retry-After` header instructing the client to wait for a period of time, select the **Set Retry-After Header** option.
   c. Optional: To configure rejection handling, select a rejection handling method:

      If you select **Default**, use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.

      If you select **Basic,**, you can customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information. If you select **Basic**, provide this information:

      1. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
      2. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
      3. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the *`PA_HOME`*`/conf/template/` directory.
      4. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.
8. Click **Save**.

Adding redirect rules

Add a redirect rule to specify a URL that a user agent should request in response to being denied access to the requested resource.

About this task

Redirect rules can be applied to one or more applications.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Redirect**.
5. In the **Response Status Code** field, specify the response status code you want to associate with the redirect.

   The default is `302`.
6. In the **URL** field, specify the URL to which you want to redirect requests.

   You can use an absolute or a relative URL. If you use an absolute URL, it must include the `http/https` prefix. If you use a relative URL, it must begin with a slash. The URL can be specified with or without defining the port.

7. To opt out of automatic URL encoding, deselect the **Encode URL** check box.

   Learn more in *Release Notes* on page 8.
8. Click **Save**.

Adding rejection rules

Add a rejection rule to specify an action to take when a request to an application or resource is rejected by policy evaluation.

About this task

A rejection rule uses *Rejection handlers* on page 290 to define which action you want to take. You can either:

- Reject the request and display an error template.
- Redirect the user to another URL for error details, instructions, or additional actions.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Rejection**.
5. In the **Rejection Handler** field, specify the rejection handler you want to use for the rule.
6. Click **Save**.

Adding time range rules

Add a time range rule, which examines a request and determines whether to grant access to a backend target site based on the request falling within a defined time frame.

About this task

You can use this rule when you want to restrict access to specific endpoints for certain time periods, such as during the work day from 8:00 a.m. to 5:00 p.m.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Time Range**.
5. In the **Start Time** field, enter the beginning time for the time frame, such as 8:00 a.m.
6. In the **End Time** field, enter the ending time for the time frame, such as 5:00 p.m.

   > **Info:**
   >
   > If you are using Internet Explorer or Firefox, you must enter the time in 24-hour format. For example, 5:00 p.m. is 17:00.
7. If, when a match is found, access is not allowed, select **Negate**.

   Additional advanced fields for handling error responses can also be defined here. For more information, see *Advanced Fields*.

8. Click **Save**.

   For more information about error handling, see *Advanced Fields*.

Adding web session attribute rules

Add a web session attribute rule, which examines a request and determines whether to grant access to a target site based on an attribute value match found within the PingAccess token.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Web Session Attribute**.
5. To grant the client access, select the **Attribute Name** that you want to match, such as `Group`.
6. Enter the **Attribute Value** for the attribute name, such as `Sales`.

   If the attribute has multiple values at runtime, the attribute value you specify here must match one of those values.

   PingAccess token attributes are obtained from the PingFederate OpenID Connect (OIDC) policy attribute contract. For more information, see *Configuring OpenID Connect Policies*.
7. To add more attributes, click **Add Row**.
8. To remove a row, click the **Delete** icon.
9. To disallow access when a match is found, click **Negate**.

   > **Info:**
   >
   > Ensure the attribute name is spelled correctly and exists. If you enter an attribute that does not exist and you select **Negate**, the rule will always succeed.
10. If you want to configure rejection handling, click **Show Advanced Settings**, then select a rejection handling method:

    Choose from:

    - Select **Default** to use the **Rejection Handler** list to select an existing *rejection handler* that defines whether to display an error template or redirect to a URL.
    - Select **Basic** to customize an error message to display as part of the default error page rendered in the end-user's browser if rule evaluation fails. This page is among the templates you can modify with your own branding or other information.

    If you select **Basic**, provide this information:

    a. In the **Error Response Code** field, enter the HTTP status response code to send if rule evaluation fails. The default is 403.
    b. In the **Error Response Status Message** field, enter the HTTP status response message to send if rule evaluation fails. The default is Forbidden.
    c. In the **Error Response Template File** field, enter the HTML template page for customizing the error message that displays if rule evaluation fails. This template file is located in the `<PA_HOME>/conf/template/` directory.
    d. From the **Error Response Content Type** list, select the type of content for the error response. This lets the client properly display the response.
11. Click **Save**.

    To use this rule, select the **Request Profile** check box, indicating that you want PingAccess to request additional profile attributes from PingFederate when requesting the ID token.

Adding web session scope rules

Add web session scope rules, which examine the contents of the PingFederate validation response and determine whether to grant access to a backend target site based on a match found between the scopes of the validation response and the scope specified in the rule.

Before you begin

Support for the web session support rule might require the PingFederate access token to contain the scope `superuser`. To configure this, see *Configuring access token attributes for superuser scope in PingFederate* on page 276.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Web Session Scope**.
5. From the **Scope** list, select the scope you want to match to values returned from the access token.

   > **Info:**
   >
   > This is one scope requirement in the set of scopes associated with the access token.
6. From the **Rejection Handler** list, select the rejection handler you want to associate with this rule.
7. Click **Save**.

Configuring access token attributes for superuser scope in PingFederate

A resource might require that the access token contains the scope `superuser`. Configure the `superuser` scope in PingFederate.

Steps

1. *Enable Expressions* within PingFederate.
2. *Extend the Access Token Attribute Contract* to include the value `scope`.
3. Map the following value into the *access token attribute contract*.

Contract	Source	Value
`scope`	Expression	@com.pingidentity.sdk.oauth20.Scope@encode(#this.get("context.OAuthS

4. Manage the *OpenID Connect policy* to add the following information:
   a. *Attribute Contract*— To extend the contract to include the `scope` attribute, select **Override Default Delivery** using the **ID Token**.

      Note:

      This step is not applicable to PingFederate 9.0 and earlier. Instead, in the **Manage Policy** window, select the **Include User Info in ID Token** check box.

   b. *Attribute Scopes*— From the **Scope** list, select `openid`, and from the **Attribute** list, select `scope`.

      Note:

      This feature does not exist in PingFederate versions earlier than 9.0. To work around this issue:

      1. Ensure PingAccess is configured to include `profile` in the list of **Web Session** scopes.
      2. In PingFederate, ensure the `profile` scope is *defined* in **Scope Management**.
      3. During authentication, the user must accept usage of the `profile` scope. If the user does not accept usage of the `profile` scope, then the web session scope rule will always fail for that user.

   c. *Contract Fulfillment*— Modify the `scope`**Attribute Contract** to use `Access Token` as the **Source** with a **Value** of `scope`.

Adding WebSocket handshake rules

Add a WebSocket handshake rule, which lets you define the domains that can open a cross-origin WebSocket to the application or resource.

About this task

You can also define allowed WebSocket subprotocols and extensions, providing more fine-grained control over how the application behaves.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **WebSocket Handshake**.
5. In the **Allowed Origins**, enter one or more origins.

   If no origins are defined, all cross-origin WebSocket requests are denied.

   Important:

   Avoid using a value of `*` in this field. While this is a valid configuration, it is considered an insecure practice.
6. Modify the list of **Allowed Subprotocols**.

   Subprotocols are defined in the `Sec-WebSocket-Protocol` handshake header. The default value of `*` allows all subprotocols.

7. Modify the list of **Allowed Extensions**.

   WebSocket extensions are defined in the `Sec-WebSocket-Extensions` handshake header. The default value of `*` allows all extensions.

   Additional advanced fields for handling error responses can also be defined here. For more information, see *Advanced Fields*.
8. Click **Save**.

Creating processing rules
Create processing rules to control how requests are processed in PingAccess.

For more information, see the following topics:

- *Adding a cross-origin request rule* on page 278
- *Adding OAuth token cache time to live rules* on page 279
- *Adding PingAuthorize response filtering rules* on page 279
- *Rewrite rules overview* on page 280

Adding a cross-origin request rule
Add a cross-origin request rule, which uses cross-origin resource sharing (CORS) to let a web server grant access to restricted resources, such as fonts, JavaScript, images, etc., to an application served by another domain without granting access to those resources beyond a list of predefined origin servers.

About this task

Before a CORS request is sent, the originating web server generally sends a pre-flight `OPTIONS` request if the request from the client includes credentials. This pre-flight request is used to determine if the target server permits CORS requests to be processed from the originating web server.

PingAccess can evaluate the headers provided in a CORS request to grant or deny access to resources.

Note:

In addition to allowing PingAccess to evaluate the CORS request, you can also allow the request to be handled by the protected application, and let PingAccess be excluded from the process of evaluating the access request, if the target application type is `API`. To do this with a resource path that is protected by PingAccess and requires user authentication, configure a second resource with the same path pattern, but set the **Methods** field to `OPTIONS` and the **Anonymous** option needs to be cleared. This configuration allows the API request being made to be handled anonymously.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **Cross-Origin Request**.
5. In the **Allowed Origins** field, enter one or more origin values.
   a. Click **+ New Value** to add additional values.

   Important:

   Avoid using a value of `*` in this field. While this is a valid configuration, it is considered an insecure practice.

6. Optional: To configure additional options, click **Show Advanced**.
   a. To permit user credentials to be used in determining access, enable **Allow Credentials**.
   b. If you entered a wildcard in the **Allowed Origins** field, select the **Mask Wildcard Policy** checkbox to replace the `Access-Control-Allow-Origin` response header with the value provided in the request's `Origin` header.
   c. To modify the **Allowed Request Headers** values, use the following options:
      - To add a new header, click **+ New Value**.
      - To edit an existing header, click the field and make your changes.
      - To remove an existing header, click the **Delete** icon.

      The default headers are `Authorization`, `Content-Type`, and `Accept`.
   d. To make specific response headers available to the client that originated the cross-origin request, enter the headers in the **Exposed Response Headers** field.
   e. To add additional headers to the list, click **+ New Value** .
   f. To define the request methods allowed in cross-origin requests, enter the desired overrides in the **Overridden Request Methods** field.
   g. To modify the amount of time the pre-flight `OPTIONS` request is cached, enter the maximum age (in seconds) in the **OPTIONS Cache Max Age** field.

      The default is 600 seconds.
7. Click **Save**.

Adding OAuth token cache time to live rules
Add an OAuth token cache time to live rules, which configures the caching behavior for access tokens.

About this task

This rule allows the global OAuth token cache configuration to be selectively overridden for specific applications or resources.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. From the **Type** list, select **OAuth Token Cache Time to Live**.
5. If you want to cache the introspection of access tokens, click **Cache Tokens**.
6. In the **Time to Live (s)** field, specify the number of seconds to cache the introspection of the access token.

   This value should be less than the token provider's token lifetime. A value of -1 means no limit.
7. Click **Save**.

Adding PingAuthorize response filtering rules
Add a response filtering rule, which contacts PingAuthorize for filtering information.

Before you begin

- Create a third party service with PingAuthorize configured as the target. For more information, see *Adding third-party services* on page 253.
- Confirm that you are not using the *agent* model. PingAuthorize response filtering rules aren't available for agent deployments.

Important:

Ensure that the sideband API setting **`request-context-method`** is set to `request` in PingAuthorize. For more information on how to set this property and why it is necessary, see *Request context configuration* in the PingAuthorize documentation.

About this task

A response filtering rule can modify the response given by PingAccess, based on the response from the PingAuthorize response API.

Important:

The PingAuthorize sideband API cannot accept gzipped data from upstream server responses. To prevent the upstream server from sending compressed responses, ensure that upstream server requests add or replace the `Accept-Encoding` header with `Accept-Encoding: identity`.

To add a PingAuthorize response filtering rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name, up to 64 characters long.

   Special characters and spaces are allowed.
4. In the **Type** list, select **PingAuthorize Response Filtering**.
5. In the **Third Party Service** list, select your PingAuthorize service.
6. In the **Shared Secret** field, enter the shared secret from PingAuthorize.
7. Optional: To include the HTTP request body in the HTTP request data sent to PingAuthorize, select the **Include Request Body** check box
8. Optional: To configure advanced options, click **Show Advanced**.
   a. Optional: In the **Sideband Endpoint** field, enter the sideband API endpoint location.
   b. Optional: In the **Shared secret header name** field, enter a header in which to send the shared secret.
9. Click **Save**.

Rewrite rules overview

PingAccess allows for the manipulation of the Request URI, the cookie domain, the cookie path, three of the response headers (`Location`, `Content-Location`, and `URI`), and the response content.

For example, a site is hosted on `https://server1.internalsite.com` under `/content/`. Users access the site through the following URL in their browser: https://server1.internalsite.com/content/

For demonstration purposes, assume this results in a *302 Redirect* to an `importantContent.html` page as well as setting a domain cookie for `.internalsite.com`. If you protect this site with PingAccess using the virtual host `publicsite.com` under the application `/importantstuff/`, you must rewrite the content. The information below discusses an example scenario.

Note:

This conceptual overview assumes that a virtual host, a site, and an application are already configured.

Create a Rewrite Content Rule

A rewrite content rule alters content in the HTTP response body.

- In the **Response Content-Types** field, you define a response type of text/html.
- In the Find and Replace criteria, you specify <a href="https://server1.internalsite.com/content/"> and <a href="https://publicsite.com/importantstuff/">.
- Add the rule to the application. A query to a page with links in it that points to https://server1.internalsite.com/content/ now points to https://publicsite.com/importantstuff/.

Create a Rewrite Cookie Domain Rule

A rewrite cookie domain rule allows the rewriting of the Domain field on cookies when they are set by the backend site.

- In the server-facing cookie domain, you enter internalsite.com.
- In the public-facing cookie domain, you enter publicsite.com.
- Add the rule to the application.

  Cookies associated with the domain publicsite.com, or .publicsite.com, are rewritten to pertain to internalsite.com, or .internalsite.com.

Create a Rewrite Cookie Path Rule

A rewrite cookie path rule converts the cookie path returned by the site into a public-facing path.

- In the **Server-Facing Cookie Path** field, you enter /content/.
- In the **Public-Facing Cookie Path** field, you enter /importantstuff/.
- Add the rule to the application.

  Cookies associated with a cookie path of /content/ are rewritten to pertain to /importantstuff/. After configuring the rewrite rules as discussed above, a user could access the https://publicsite.com/importantstuff/ and PingAccess would route that request to https://server1.internalsite.com/content/.

  If the site sends a redirect to https://server1.internalsite.com/content/index.html, PingAccess would return a redirect to https://publicsite.com/importantstuff/index.html. If the site then returned a cookie with a domain of .internalsite.com and a path of /content/, PingAccess would rewrite that cookie to be relevant to .publicsite.com and /importantstuff/.

Create a Rewrite Response Header Rule

A rewrite response header rule alters the response header used in the 302 Redirect.

- In the **Server-Facing URI** field, you enter https://server1.internalsite.com/content/.
- In the **Public Path** field, you enter /importantstuff/.
- Add the rule to the application. A query resulting in a response containing a 302 Redirect to https://server1.internalsite.com/content/ is rewritten to https://publicsite.com/importantstuff/.

> **Info:**
>
> This also works for relative redirects: /content/ is rewritten to /importantstuff/. It also works for the path beneath the one defined in the URI: /content/news/index.html is rewritten to importantstuff/news/index.htm.

Create a Rewrite URL Rule

A rewrite URL rule alters the request URI.

- In the **Map From** field, you enter `^/importantstuff/(.*)` as the regex of the URL's path and query you want to match.
- In the **Map To** field, you enter `/content/$1`.
- Add the rule to the application. A query to `https://publicsite.com/importantstuff/` results in PingAccess routing that query to `https://server1.internalsite.com/content/`.

Adding rewrite content rules

Add rewrite content rules, which modify text in HTTP response bodies as it is served to the client in PingAccess.

Before you begin

Confirm that you are not using the *agent* model. Rewrite content rules aren't available for agent deployments.

About this task

A rewrite content rule uses a subset of the Java regular expression syntax that excludes look-behind constructs, such as `\b`, and the boundary matcher, `\G`. If no Java regular expression syntax is used, the rule performs a case-sensitive search and replace.

The most common use case for this rule is to rewrite host names within URLs contained in HTML, JavaScript, or CSS content.

> **Info:**
>
> Extensive use of rewrite content rules might have significant performance implications.

This rule supports content that is either chunked or streamed from the target server. When sent to the client, the content is always chunked.

To add a rewrite content rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters long.

   Special characters and spaces are allowed.
4. In the **Type** list, select **Rewrite Content**.
5. To define what type of response data to which the rewrite rule applies, enter one or more **Response Content-Types**.

   The default values are `text/html`, `text/plain`, and `application/json`. The list is an ordered list.

   > **Info:**
   >
   > Only text-based content types are supported. Text-based content types compressed with gzip, deflate, or compress will decompress prior to rewrite rule processing, however, the content is not re-compressed before sending to the client.

6. Define one or more sets of **Find and Replace Criteria**.

   If multiple criteria are specified, each operation is performed against the original content, effectively applying the rule concurrently.

> **Important:**
>
> Changes can affect CSS, Javascript, and other text-based elements served to the client. Be sure to properly craft the regular expression to avoid unintentionally modifying content.

7. If the protected application does not return a **Content-Type** header, select **Missing Content-Type Allowed**.
8. If you enable **Missing Content-Type Allowed**, specify the encoding the application returns in the **Missing Content-Type Charset** field.

   This field could contain `UTF-8`. A list of valid values is available in the *Oracle Java 8 SE Technical Note*.
9. If necessary, increase the size of the buffer used to perform the replace operation by clicking **Show Advanced** and entering a value in **Maximum Buffer Size**.

> **Note:**
>
> Replacement values cannot be larger than the buffer size. The minimum buffer size you can specify is 1024 bytes. There is no maximum value.

10. Click **Save**.

Rewrite content rule examples

This table provides examples of rewrite content rule use cases and their results.

Example description	Original content	Content type	Find criteria	Replacement value	Modified text
Rewrite URL portion of a web link	<a href="https://serverx.inside.corp/app/">	text/html	serverx.inside.corp	www.acme.com	<a href="https://www.acme.com/app/">
Case-sensitive text replacement	ACMEcorp	text/html	Ecorp	E Corporation	ACME Corporation
JSON Value masking	{ "origin": "127.0.0.1, 192.168.1.1" }	application/json	(127.0.0.1,)*.*	************"	{ "origin": "127.0.0.1, ***********" }
Replacing text inside a specified element using Java regex groups	This text is **bold**.	text/html	(bold)	not $1	This text is not bold.

Example description	Original content	Content type	Find criteria	Replaceme value	Modified text
Case-insensitive text replacement using a Java regex match flag	HTTP	`text/html`	`(?i)http`	`FTP`	FTP

Adding rewrite cookie domain rules

Add a rewrite cookie domain rule, which converts the cookie domain returned by the site into a public-facing domain, in PingAccess.

Before you begin

Confirm that you are not using the *agent* model. Rewrite cookie domain rules aren't available for agent deployments.

About this task

When a site places a cookie on a cookie domain such as `internalsite.com`, or `.internalsite.com`, PingAccess rewrites the `Domain` portion of the `Set-Cookie` response header with a public-facing domain such as `publicsite.com`, or `.publicsite.com`, using the information configured in the rewrite cookie domain rule.

Note:

You should only set the cookie in the **Public-Facing Cookie Domain** field to the virtual host name associated with that application, or to a domain that is above. For example, `myserver.acme.com` can be set to `acme.com`.

To add a rewrite cookie domain rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters.

   Special characters and spaces are allowed.
4. In the **Type** list, select **Rewrite Cookie Domain**.
5. If you need to explicitly define the target host, clear the **Any Site Target Host** check box.

   When you select the **Any Site Target Host** check box, PingAccess will rewrite the cookie domain if it is set to the domain defined in a site's target host list.
6. If you clear the **Any Site Target Host** check box, enter the domain name used by the backend site in the **Server-Facing Cookie Domain** field.
7. In the **Public-Facing Cookie Domain** field, enter the domain name that you want to display in the response from PingAccess.
8. Click **Save**.

Adding rewrite cookie path rules

Add a rewrite cookie path rule, which converts the cookie path returned by the site into a public-facing path, in PingAccess.

Before you begin

Confirm that you are not using the *agent* model. Rewrite cookie path rules aren't available for agent deployments.

About this task

This task enables the details of exposed applications to be managed by PingAccess for security and request routing purposes.

For example, a site places a cookie in a server-facing cookie path such as `/content/`. Using the information configured in the rewrite cookie path rule, PingAccess rewrites the `Path` portion of the `Set-Cookie` response header with a public-facing cookie path such as `/importantstuff/`.

To add a rewrite cookie path rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters.

   Special characters and spaces are allowed.
4. In the **Type** list, select **Rewrite Cookie Path**.
5. In the **Server-Facing Cookie Path** field, enter the path name where the cookie is valid for the backend site.
6. In the **Public-Facing Cookie Path** field, enter the path name that you want to display in the response from PingAccess.
7. Click **Save**.

Adding rewrite response header rules

Add a rewrite response header rule, which converts the response header value returned by the site into a public-facing value.

Before you begin

Confirm that you are not using the *agent* model. Rewrite response header rules aren't available for agent deployments.

About this task

This rule rewrites one of three response headers:

1. `Location`
2. `Content-Location`
3. `URI`

For example, the server-facing `Location` response header includes a path that begins with `/test-war/`. Using the information configured in the rewrite response header rule, PingAccess rewrites `http://private/test-war/` with a public-facing path such as:

```
http://public/path/
```

To add a rewrite response header rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name up to 64 characters.

   Special characters and spaces are allowed.
4. In the **Type** list, select **Rewrite Response Header**.
5. If the target host needs to be explicitly defined, clear the **Any Site Target Host** check box.

   When you select the **Any Site Target Host** check box, PingAccess will rewrite the response header URI if it contains a domain defined in a site's target host list.
6. If you clear the **Any Site Target Host** check box, enter the domain name used by the backend site in the **Server-Facing URI** field.
7. In the **Public Path** field, enter a valid URI path that you want to write into the URI.

   This must be a valid [MISSING glossAcronymn or glossSurfaceform for gURI] path and begin and end with a forward slash (`/`).

   Example:

   `/importantstuff/` or `/`
8. Click **Save**.

Adding rewrite URL rules

Add a rewrite URL rule, which examines the URL of every request and determines if a pattern matches, in PingAccess.

Before you begin

Confirm that you are not using the *agent* model. Rewrite URL rules aren't available for agent deployments.

About this task

When you define a regular expression in a rule (such as `regex`), and if a pattern matches, PingAccess uses the information configured in the rewrite URL rule and rewrites that portion of the [MISSING glossAcronymn or glossSurfaceform for gURL] into a path that the site can understand.

To add a rewrite URL rule:

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name for the rule up to 64 characters.

   Special characters and spaces are allowed.
4. In the **Type** list, select **Rewrite URL**.
5. In the **Map From** field, enter the `regex` of the URL path and the query you want to match.

   Example:

   `^/bank/(.*)`

   This example illustrates matching the `Request-Line` in the request. The `Request-Line` begins with `/bank/` (the `^` indicates "begins with") and places the rest of the URL into the first capture group.

   For more information on `regex` patterns, see the *Oracle Java Docs*.

6. In the **Map To** field, enter the URL path and the query that you want to generate.

   Example:

   `/application/$1`

   This example defines the replacement string, which generates `/` followed by the content of the first capture group.

   To better understand the use of special characters, such as `\` and `$`, in the replacement string, see the *Oracle Java Docs*.
7. Click **Save**.

Rewrite URL rule configuration examples

This table displays four examples of rewrite URL rule configurations in PingAccess.

Map from value	Map to value	Example request	Rewrite by PingAccess
`/bank/`	`/application/`	`/bank/content.html`	`/application/content.html`
`^/bank/(.*)`	`/application/$1`	`/bank/content.html`	`/application/content.html`
`/bank/index.html`	`/application/index.jsp`	`/bank/index.html`	`/application/index.jsp`
`/bank/index.html`	`/application/index.jsp`	`/bank/index.html?query=stuff`	`/application/index.jsp?query=stuff`

Editing rules

Edit the properties of an existing rule in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click to expand the rule you want to edit.
3. Click the **Pencil** icon.
4. Make the desired changes to the rule you want to edit.
5. To confirm your changes, click **Save**.

Deleting rules

Delete an existing rule set in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click to expand the rule you want to delete.
3. Click the **Delete** icon.
4. To confirm your change and delete the rule, click **Delete**.

Rule sets

Rule sets let you combine rules into reusable groupings in PingAccess.

Adding rule sets

Add a new rule set in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rule Sets**.

2. Click **+ Add Rule Set**.
3. In the **Name** field, enter a name for the rule set you want to add.

   Special characters and spaces are allowed.
4. In the **Success Criteria** field, select the rules in the set you want to succeed.
   Choose from:
   - To require all rules in the set to succeed, click to select **All**.
   - To require just one of the rules in the set to succeed, click to select **Any**.
5. To select a rule, click to drag the desired rules from the **Available Rules** column into the **Selected Rules** column.
6. To save the rule set, click **Save**.

Editing rule sets

Edit an existing rule set in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rule Sets**.
2. Click to expand the rule set you want to edit.
3. Click the **Pencil** icon.
4. To make the desired edits:
   Choose from:
   - Edit the rule set **Name** or **Success Criteria**.
   - To remove a rule from a rule set, click **-** next to the rule.
   - To re-order the rules, drag a rule within a rule set up or down.
5. To confirm your edits, click **Save**.

Deleting rule sets

Delete an existing rule set in PingAccess. You must remove all associations between the rule set and any applications or resources before you can delete it.

Steps

1. Click **Access** and then go to **Rules** # **Rule Sets**.
2. Click to expand the rule set you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Rule set groups

Rule set groups let you combine one or more rule sets into reusable groups in PingAccess.

Adding rule set groups

Add a new rule set group in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rule Set Groups**.
2. Click **+ Add Rule Set Group**.
3. In the **Name** field, enter a name for the rule set group.

   Special characters and spaces are allowed.

4. To require all rules in the set to succeed, select **All** as the **Success Criteria** field.
    a. To require only one of the rules in the set to succeed, select **Any**.

   When **Success Criteria** is set to **Any**, the first rule set establishes the error handling and is flagged with a tool tip that displays a message indicating that.

   When **Success Criteria** is set to **All**, the first rule in the set that fails establishes the error handling.

   When **Success Criteria** is set to **Any**, PingAccess flags processing rules in a rule set group with a tool tip that warns if the first rule in the list succeeds, additional rules will not be processed. This is considered a misconfiguration because in an **Any** rule set group, the first processing rule should succeed, causing all other rules in the set to not be evaluated.

   If you want to use processing rules on protected applications and handle access control decisions using **Any** criteria, assign processing rules directly to the application, or create a separate rule set group for the processing rules using the **All** criteria.
5. To add one or more rule sets or rule set groups, select **Rule Sets** or **Rule Set Groups** and drag them from the available list to the selected list.
6. To configure rule set hierarchy, drag rule sets into the rule set group.

   Processing occurs from top to bottom. The configuration is automatically saved.
7. To save the rule set group, click **Save**.

Editing rule set groups
Edit the properties of an existing rule set group in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rule Set Groups**.
2. Click to expand the rule set group you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the rule set groups:
    a. Edit the rule set group **Name** or **Success Criteria**.
    b. To add a new rule set or rule set group to the existing rule set group, drag them into the selected column.
    c. To create a new rule set, click **+ Create Rule Set**

       The new rule set is not automatically added to the rule set group.
    d. To reorder the rules, drag a rule set within a rule set group up or down.
    e. To remove a rule set from a rule set group, click **-** to the right of any rule set.
5. To confirm your changes, click **Save**.

Deleting rule set groups
Delete an existing rule set group in PingAccess. You must remove any associations between the rule set group and any applications or resources before you can delete it.

Steps

1. Click **Access** and then go to **Rules** # **Rule Set Groups**.
2. Click to expand the rule set group you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Rejection handlers

A rejection handler defines the action to take when a request to an application or resource is rejected by policy evaluation. This lets you decide if you want to display an error template or redirect the user to another URL for error details, instructions, or additional actions.

You can specify the response status code to send if policy evaluation fails.

You include a rejection handler in a rule by clicking to expand **Advanced** on the **Create Rule** window.

PingAccess contains three predefined rejection handlers:

Default API Rejection Handler

Returns a `403` status code in a JSON template.

Default Rate Limiting Rejection Handler

Returns a `429` status code in a JSON template.

Default Web Rejection Handler

Returns a `403` status code in an HTML template.

Creating rejection handlers

Create a new rejection handler in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rejection Handlers**.
2. Click **+ Add Rejection Handler**.
3. In the **Name** field, enter a name for the object.
4. Choose the **Type**:
   Choose from:
   - **Error Template**
   - **Redirect**

   a. If you selected **Error Template**, specify the **Response Status Code**, the **Template File**, and the **Content Type**.
   b. If you selected **Redirect**, specify the **Response Status Code**, and **URL** to which you want to redirect if policy evaluation fails.

      You can use an absolute or a relative URL. If you use an absolute URL, it must include the `http/https` prefix. If you use a relative URL, it must begin with a slash. The URL can be specified with or without defining the port.

> **Tip:**
>
> To opt out of automatic URL encoding, deselect the **Encode URL** check box. Learn more in *Release Notes* on page 8.

Editing rejection handlers

Edit the properties of an existing rejection handler in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rejection Handlers**.
2. Click to expand the rejection handler you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the rejection handler.

5. To confirm your changes, click **Save**.

Deleting rejection handlers
Delete an existing rejection handler in PingAccess.

Steps

1. Click **Access** and then go to **Rules** # **Rejection Handlers**.
2. Click to expand the rejection handler you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Authentication

Authentication challenge policies and authentication requirements let you control how users are authenticated.

Authentication challenge policies

Authentication challenge policies set the response sent by PingAccess when it receives unauthenticated requests for protected resources from web applications.

Each authentication challenge policy consists of one default mapping and zero or more challenge response mappings. When a user attempts to access a protected resource and a PingAccess web session has not yet been established, and when the request characteristics match those of a challenge response mapping, the response specified in the challenge response mapping is used. If the request does not match any challenge response mappings, the default mapping is used.

Authentication requirements

Authentication requirements are methods of authentication that are ordered by preference.

When a user attempts to access a PingAccess web application configured with an authentication requirement list containing the values password and certificate, PingAccess redirects the user to PingFederate requesting either password or certificate user authentication. PingFederate authenticates the user based on the password and issues an OpenID Connect (OIDC) ID token to PingAccess, containing the authentication method that was used. PingAccess ensures that the authentication method matches the requirements and redirects the user to the originally requested application with the PingAccess cookie set. When the user attempts to access a more sensitive application configured with an authentication requirement list containing the value (certificate), they are redirected to PingFederate to authenticate with a certificate.

You can configure applications with authentication requirement lists that have no overlap. For example, if one list has a password and another list has a certificate, a user navigating between applications might be required to authenticate each time they visit an application. When configuring authentication requirement lists to protect higher value applications with step-up authentication, include stronger forms of authentication when configuring lower value applications.

Configuring authentication challenge policies
Configure an authentication challenge policy in PingAccess to set the response PingAccess sends when it receives unauthenticated requests for protected resouces.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Challenge Policies**.
2. Click **+ Add Authentication Challenge Policy**.
3. In the **Name** field, enter a unique name for the authentication challenge policy.
4. Optional: In the **Description** field, enter a description for the authentication challenge policy.

5. Optional: From the **Challenge Response Mapping** list, select a mapping type.
   a. If you selected **Content Negotiation**, in the **Media Types** list, select one or more media types.

      If the `Accept` header field in the request matches any of the specified media types, the mapping is applied.
   b. If you selected **Header Fields**, click **+ Add Row** to add one or more rows, and then in the **Name** and **Value Pattern** fields, enter a name and value pattern and for each row.

      If all of the specified header fields in the request match the specified value patterns, the mapping is applied.
6. Configure a challenge response generator for the challenge response mapping.
   a. From the **Challenge Response Generator** list, select a challenge response generator.

Challenge Response Generator	Description
HTML OIDC Authentication Request	Generates a response with a 401 response code. The response body is an HTML document that automatically issues the OIDC authentication request using JavaScript. The HTML always attempts to preserve the fragment of the current browser URL and preserves a POST body if the `Content-Type` is `application/x-www-url-formencoded`.
OIDC Authentication Request Redirect	Generates a response with a 302 response code. The response body directs the browser to send an OIDC authentication request to the OpenID provider.
PingFederate Authentication API Challenge	Generates a response with a 401 response code. The body is a JSON object that directs the application to connect to the PingFederate redirectless authorization API. The JSON object contains three strings: ▪ `authorizationUrl` represents the OIDC authentication request. ▪ `method` indicates the HTTP method for the request to the PingAccess OIDC callback endpoint. ▪ `oidcAuthnResponseEndpoint` is the location of the PingAccess OIDC callback endpoint. For more information, about the required PingFederate configuration, see *Authentication API* in the PingFederate documentation. For more information about configuring the JavaScript widget to enable this challenge response, see the *Redirectless support* page on github.
Redirect Challenge	Generates a response with the specified response code that redirects the user to a specified URL.

Challenge Response Generator	Description
Templated Challenge	Generates a response with the specified response code based on a specified template. The allowed template variables include the following: ▪ `resource.name` represents a string containing the name of the requested resource. ▪ `application.name` represents a string containing the name of the requested application. ▪ `application.realm` represents a string containing the OAuth realm associated with the application. If the realm is not defined by the application, it is inferred to be the requested authority and the application's context root. ▪ `exchangeId` represents a string containing the ID for the current transaction.

b. If you selected **Redirect Challenge**, enter a **Redirect URL** and select a **Response Code** for the redirect.
c. If you selected **Templated Challenge**, select a **Response Code** and **Media Type** for the template, then enter the template in the **Template** field.
d. From the **Challenge Response Filter** list, select a challenge response filter:

   1. If you selected **Append Header Fields**, click **+ Add Row**.
   2. Enter a **Name** and **Value** in each row.

   Result:

   The specified HTTP response header fields are appended to the authentication challenge response.

7. Optional: To add additional challenge response mappings, click **+ Add Challenge Response Mapping**, then repeat steps 5 and 6.
8. In the **Default Challenge Response** section, select a default challenge response.

   PingAccess uses this challenge response if no other challenge responses apply.

   a. From the **Challenge Response Generator** list, select a challenge response generator.
   b. If you selected **Redirect Challenge**, enter a **Redirect URL** and select a **Response Code** for the redirect.
   c. If you selected **Templated Challenge**, select a **Response Code** and **Media Type** for the template, then enter the template in the **Template** field.
   d. From the **Challenge Response Filter** list, select a challenge response filter.

      - If you selected **Append Header Fields**, click **+ Add Row**, then enter a **Name** and **Value** in each row.

      Result:

      The specified HTTP response header fields are appended to the authentication challenge response.

9. Click **Save**.

Editing authentication challenge policies

Edit the properties of an existing authentication challenge policy in PingAccess.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Challenge Policies**.
2. Click to expand the authentication challenge policy you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the authentication challenge policy.
5. To confirm your changes, click **Save**.

Deleting authentication challenge policies

Delete an existing authentication requirements list in PingAccess.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Challenge Policies**.
2. Click to expand the authentication challenge policy you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Configuring authentication requirements lists

Configure a list of authentication requirements in PingAccess.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Requirements**.
2. Click **+ Add Authentication Requirement**.
3. In the **Name** field, enter a unique name for the authentication requirements list.
4. In the **Authentication Requirements** field, enter an authentication method, such as `cert` or `password`.

   The values you enter here must match the result values defined for the Requested AuthN Context Selector configured within PingFederate. For more information, see *Configuring the Requested AuthN Context Selector*.
5. To add one or more additional authentication requirements, click **+ Add Authentication Requirement**.
6. Click **Save**.

Editing authentication requirements lists

Edit the properties of an existing authentication requirements list in PingAccess.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Requirements**.
2. Click to expand the list you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the authentication requirements list.
5. To confirm your changes, click **Save**.

Deleting authentication requirements lists

Delete an existing authentication requirements list in PingAccess.

Steps

1. Click **Access** and then go to **Authentication** # **Authentication Requirements**.
2. Click to expand the authentication requirement you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Identity mappings

Identity mappings make user attributes available to backend sites that use them for authentication.

There are multiple types of identity mappings, each with a different behavior and a distinct set of fields to specify the identity mapping behavior.

Creating header identity mappings

Create a header identity mapping to make user attributes or client certificates available as HTTP request headers to applications, both site- and agent-based, that use them for authentication.

About this task

A single header identity mapping can expose a number of attribute values or a certificate chain up to three levels deep. Header identity mappings are assigned to applications.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. From the **Type** list, select **Header Identity Mapping**.
5. In the `Attributes` section, select a list type.
   Choose from:
   - **Inclusion List** – Includes the specified attributes as headers.
   - **Exclusion List** – Includes all attributes as headers, with the exception of those specified.
6. If you selected Inclusion List, specify the Inclusion List parameters.
   a. In the **Header Name Prefix** field, enter a prefix.

      This prefix is prepended onto all header names.
   b. In the **Attribute Name** field, enter or select the name of the attribute to retrieve from the user web session in the Attribute Name field.

      For example, sub.
   c. In the **Header Name** field, enter the name of the HTTP requests header to contain the attribute value.

      The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the User-Agent browser type identifying header as User-Agent, not HTTP_USER_AGENT.
   d. Optional: Click **+ Add Row** to add additional sets of attributes and headers.
   e. Optional: Click **Subject** to select which attribute is used as the subject.

7. If you selected Exclusion List, specify the Exclusion List parameters.
    a. In the **Header Name Prefix** field, enter a prefix.

        This prefix is prepended onto all header names.
    b. Optional: In the **Excluded Attributes** field, enter one or more attributes to exclude.

        All attributes not specified are included as headers.
    c. In the **Subject Attribute Name** list, select an attribute to use as the subject.
8. In the **Certificate to Header Mapping** section, enter the header name to contain a PEM-encoded client certificate.

    The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate.

    a. If you are using a certificate chain, click **+ Add Row** to add another row.
9. Click **Save**.

Creating JWT identity mappings

To make user attributes available in a signed JSON web token (JWT) sent to the application in a header, create a JWT identity mapping .

About this task

The JWT issuer and signing configuration is defined in *Configuring auth token management* on page 298.

When configuring identity mappings, the dot notation is supported so that session token structure can be maintained. For example, if the session token contains the following entry:

```
{
  "address": {
     "line1": "123 Any St",
     "line2": "Apt 123",
     "city": "Anytown",
     "state": "CO",
     "zip": "12345"
   }
}
```

you can define an identity mapping using the entries in the following table to maintain the structure of the target JWT.

User attribute Name	JWT claim name
address.line1	address.line1
address.line2	address.line2
address.city	address.city
address.state	address.state
address.zip	address.zip

Tip:

PingAccess engines provide a JWKS (JSON Web Key Set) endpoint at `/pa/authtoken/JWKS` that can be used by backend sites to validate the signature of the JWT.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the mapping.
4. From the **Type** list, select **JWT Identity Mapping**.
5. Select **Map as Bearer Token** to map the identity JWT as a bearer token in the Authorization request header field.

   This token replaces any existing Authorization request header field. The **Header Name** field is disabled if this option is selected.
6. In the **Header Name** field, enter the name of the header to use when sending the signed JWT to the target application.

   The HTTP header you specify here is the actual header name over the HTTP protocol, not an environment variable interpreted format. For example, enter the `User-Agent` browser type identifying header as `User-Agent`, not `HTTP_USER_AGENT`.
7. In the **Audience** field, enter the audience to be set as the `aud` claim in the signed JWT in the **Audience** field.
8. In the **Attributes** section, select a list type.

   An inclusion list includes only the specified attributes, and an exclusion list includes all attributes not specified.
9. If you selected an inclusion list, configure the inclusion list:
   a. In the **User Attribute Name** field, enter or select the name of the attribute to retrieve from the user web session, such as `sub`.
   b. In the **JWT Claim Name** field, enter the name of the JWT claim to contain the attribute value.
   c. Select which included attribute is used as the **Subject**.
10. If you selected an exclusion list, configure the exclusion list:
    a. Enter the names of the attributes to exclude.
    b. Select which included attribute is used as the **Subject**.
11. Optional: In the **Client Certificate Chain JWT Claim Name** field, enter the name of the JWT claim to contain the client certificate chain array.
12. If you are performing Certificate to JWT Claim Mapping, in the **Client Certificate Max Depth** field, specify the maximum number of certificates from the client certificate chain included in the JWT claim array.
13. Optional: To use a cached signed JWT for repeated requests for a given user, click **Show Advanced** and select **Cache JWT**.

    If user attributes change or the key used to sign the JWT changes, a new JWT will be created even if JWT caching is enabled.
14. Click **Save**.

Creating web session access token identity mappings
Add a web session access token identity mapping to PingAccess.

About this task

PingAccess can map the access token from a web session to the bearer authentication token of request headers. This lets PingAccess integrate with protected APIs that expect OAuth access tokens issued by the same OAuth authorization server used in PingAccess.

Steps

1. In the administrative console, click **Access** then go to **Identity Mappings** # **Identity Mappings**.

2. Click **+ Add Identity Mapping**.
3. In the **Name** field, enter a name for the new identity mapping.
4. From the **Type** list, select **Web Session Access Token Identity Mapping**.
5. Click **Save**.

Editing identity mappings

Edit the properties of an existing identity mapping in PingAccess.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click to expand the identity mapping you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the identity mapping.
5. To confirm your changes, click **Save**.

Deleting identity mappings

Delete an existing identity mapping in PingAccess.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
2. Click to expand the identity mapping you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Configuring auth token management

To define the issuer and signing configuration used by JSON web token (JWT) identity mappings, configure auth token management.

Steps

1. Click **Access** and then go to **Identity Mappings** # **Auth Token Management**.
2. To enable key rolling using the specified key roll interval, click **Key Roll Enabled**.
3. To indicate how often, in hours, you want to roll the keys, specify the **Key Roll Interval (h)**.

   Key rollover updates keys at regular intervals to ensure the security of the signed auth tokens.
4. In the **Issuer** field, specify a published, unique issuer identifier to use with auth tokens.

   Set the issuer to a value that more closely represents your company. PingAccess inserts this value as the iss claim within the auth token.
5. In the **Signing Algorithm** field, select the signing algorithm used to protect the integrity of the auth tokens.

   The default is ECDSA using P-256 Curve.
6. Click **Save**.

Web sessions

Web sessions define the policy for web application session creation, lifetime, timeouts, and their scope.

You can configure any number of web sessions to scope the session to meet the needs of a target set of applications. This improves the security model of the session by preventing unrelated applications from impersonating the end user. Use the tasks within this section to configure secure web sessions for use with specific applications and to configure global web session settings.

Application scoped web sessions

PingAccess tokens can be configured to have their web sessions scoped to a specific application. This improves the security model of the session by preventing unrelated applications from impersonating the end user.

Several controls exist to scope the PingAccess token to an application:

Audience Attribute

The audience attribute defines who the token is applicable to and is represented as a short, unique identifier. Requests are rejected that contain a PingAccess token with an audience that differs from what is configured in the web session associated with the target resource.

Audience Suffix

The audience attribute is also used as a suffix of the cookie name to ensure uniqueness. For example, PA.businessAppAudience.

Cookie Domain

The cookie domain can also optionally be set to limit where the PingAccess token is sent.

Note:

In addition to these controls, parameters such as session timeout can be adjusted to match the policy requirements of each application.

Corresponding OAuth clients must be defined in PingFederate for each web session. Redirect URL whitelists defined in PingFederate dictate from which servers and domains the session can originate. Controlling this within PingFederate enables flexibility of the attribute contract, and its fulfillment, for that particular application. This ensures that each application and its associated policies only deal with attributes related to it.

Configuring web session management settings

Configure web session management settings in PingAccess.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Session Management**.
2. In the **Web Session Management** section, select **Key Roll Enabled** to enable key rolling using the interval specified below.
3. Enter the **Key Roll Interval**, in hours, to specify how often you want to roll the keys (the default is `24` hours).

   Key rollover updates keys at regular intervals to ensure the security of signed and encrypted PingAccess tokens.
4. In the **Issuer** field, enter the published, unique identifier to be used with the web session (the default is PingAccess).

   Example: Set the issuer to a value that more closely represents your company. PingAccess inserts this value as the `iss` claim within the PingAccess token
5. Select the **Signing Algorithm** used to protect the integrity of the PingAccess token (the default is `ECDSA using P-256 Curve`).

   PingAccess uses the algorithm when creating signed PingAccess tokens and when verifying signed tokens in a request from a user's browser. The algorithm is also used for signing tokens in token mediation use cases when PingAccess tokens are encrypted

6. Select the **Encryption Algorithm** used to encrypt and protect the integrity of the PingAccess Token (the default is `AES 128 with CBC and HMAC SHA 256`).

   PingAccess uses the algorithm when creating encrypted PingAccess tokens and when verifying them from a user's browser.

   Higher encryption levels are available if the administrative console supports it. To enable higher encryption levels, update the administrative console Java Runtime Environment (JRE) to support unlimited strength security policy.

   In a clustered environment, add the security policy changes to the engines as well as the administrative console for the cluster.
7. Enter the browser **Cookie Name** that contains the PingAccess token (the default is `PA`).
8. In the **Session State Cookie Name** field, enter a name for the browser cookie to contain session state attributes.
9. In the **Update Token Window (s)** field, enter the number of seconds before the idle timeout is updated in the PingAccess token.

   When this time window expires, PingAccess will reissue a new PingAccess cookie.
10. In the **Nonce Cookie Time to Live (m)** field, enter the number of minutes for which the nonce cookie is valid.

    The default value is `5`. PingAccess deletes cookies that are older than this threshold.
11. Click **Save**.

Creating web sessions
Create a new web session in PingAccess.

Before you begin

If you are using PingFederate as the OIDC token provider and plan to use **Mutual TLS**, you must make two changes to the PingFederate configuration.

- Enable the use of the secondary HTTPS port in PingFederate by editing the `<PF_HOME>/pingfederate/bin/run.properties` file and setting the `pf.secondary.https.port` value to a port value. For more information, see *the PingFederate documentation*.
- Modify the `openid-configuration.template.json` to add the `mtls_endpoint_aliases` object, with content defined by *RFC-8705*. For more information about this file, see the *PingFederate documentation*.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
2. Click **+ Add Web Session**.
3. In the **Name** field, enter a unique name for the web session, up to 64 characters, including special characters and spaces.
4. Select a **Cookie Type**.

   An encrypted JSON web token (JWT) uses authenticated encryption to simultaneously provide confidentiality, integrity, and authenticity of the PingAccess token. **Encrypted JWT** is the default setting.

   A **Signed JWT** uses asymmetric cryptography with a private/public key pairing to verify the signed message and to confirm that the message was not modified during transit.

   Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.

5. In the **Audience** field, specify the audience that the PingAccess token is applicable to, represented as a short, unique identifier between 1 - 32 characters.

   Requests are rejected that contain a PingAccess token with an audience that differs from what is configured in the web session associated with the target application. Changing this setting can affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
6. In the **OpenID Connect Login Type** field, specify the OpenID Connect (OIDC) login type.

   For more information on the available profiles, see *OpenID Connect login types*.
7. In the **Client ID** field, select the client ID that was assigned when you created the OAuth relying party client within PingFederate.

   For more information, see *Configuring a Client*. Enter the unique identifier (Client ID).
8. Select a **Client Credentials Type**, then provide the information required for the selected credential type.

   This is required when configuring the **Code** login type.

   Choose from:

   - **Secret** – Enter the **Client Secret** assigned when you created the OAuth relying party client in the token provider.
   - **Mutual TLS** – Select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – Select this option to use Private Key JWT. No additional information is required.

   The OAuth client you use with PingAccess web sessions must have an OIDC policy specified. For more information, see *Configuring OpenID Connect Policies*.
9. In the **Idle Timeout** field, specify an idle timeout that defines the amount of time, in minutes, that the PingAccess token remains active when no activity is detected by the user.

   The default is `60` minutes.

   If there is an existing valid PingFederate session for the user, an idle timeout of the PingAccess session might result in its re-establishment without forcing the user to sign on again
10. In the **Max Timeout** field, specify a max timeout that defines the maximum amount of time, in minutes, that the PingAccess token remains active.

    The default is `240` minutes. Once the PingAccess Token expires, an authenticated user must re-authenticate. This protects against unauthorized use of a resource, ensuring that a session ends after the specified time and requiring the user to re-authenticate to continue.

    Note:

    This value needs to be smaller than the PingFederate access token lifetime defined in the PingFederate access token management instance. For more information, see *Configuring Reference-Token Management*.
11. Optional: To configure advanced settings, click **Show Advanced**.

Advanced setting	Description
Cookie Domain	Specify the valid **Cookie Domain** where the cookie is stored, such as `corp.yourcompany.com`. **Note:** If you set the **Cookie Domain**, all of your web resources must reside within that domain. If you

Advanced setting	Description
	do not set the **Cookie Domain**, the PingAccess token is recreated for each host domain where you access applications.
Secure Cookie	Select **Secure Cookie** to indicate that the PingAccess cookie must be sent using only HTTPS connections. This is selected by default. **Note:** Setting an invalid **Cookie Domain** or selecting **Secure Cookie** in a non-HTTPS environment causes authentication to fail. This results in PingAccess re-directing the user to re-authenticate with PingFederate indefinitely.
HTTP-Only Cookie	Select **HTTP-Only Cookie** to enable the `HttpOnly` flag on cookies that contain the PingAccess token. An `HttpOnly` flagged cookie is not accessible using non-HTTP methods such as calls through JavaScript, like referencing `document.cookie`, and cannot be easily stolen using cross-site scripting.
Enable PKCE	If you want PingAccess to send a SHA256 code challenge and corresponding code verifier as a proof key for code exchange during the code authentication flow, click **Enable PKCE**.
SameSite Cookie	From the **SameSite Cookie** list, select the level of restriction for when cookies can be sent in a cross-site request. The options are: ▪ `Lax` – The cookie should be sent on the initial navigation to a site, and is sent in same-site requests but not cross-site requests. ▪ `None` – The cookie is intended to be used across different sites without restriction. ▪ `Disabled` – The SameSite attribute is not set. This option is the default. **Note:** Safari 12 will not function correctly if the SameSite attribute is set to **None**. Regardless of the selected setting, the SameSite attribute is disabled if Safari 12 is detected. **Note:**

Advanced setting	Description
	See the *Known Issues and Limitations* for information about a browser issue that can prevent sign on if the **SameSite Cookie** attribute is set.
Scopes	Configure the **Scopes** you want to request from the token provider when requesting the ID token. If you have a token provider configured, published scopes are available to select from the list based on the selected Client ID. You can specify unverified scopes by typing the scope and clicking **Use unverified scope "[scopename]"**. Your token provider must be properly configured to handle all of the requested scopes you specify, including any custom scope values. **Note:** The user can access all attributes by examining browser traces. While they are integrity protected to prevent changes, any sensitive or confidential attributes can be viewed should the user decode the ID Token's value.
Validate Session	Select **Validate Session** so that PingAccess will validate sessions with the configured PingFederate instance during request processing. Use of this feature requires additional configuration in PingFederate. This option is not selected by default. Session timeouts are synchronized between PingAccess and PingFederate when the following conditions are met: ▪ A minimum release of PingFederate 8.2 is deployed with **Authentication Session Settings** configured. ▪ You have selected the **Validate Session** check box. Changing this setting might affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
Refresh User Attributes	When you enable **Refresh User Attributes**, PingAccess will periodically contact PingFederate to update user data used in evaluating policy claims. This option works in conjunction with the PingAccess web session management features to automatically require user re-authentication if user attribute data used as issuance criteria for

Advanced setting	Description
	a token in PingFederate causes the token to be revoked. PingFederate provides data according to its OIDC policy, which can pull data from the access token or from an attribute source lookup. If it pulls data from the access token, the data does not change until the token expires. If it pulls data from an attribute source lookup, the new data is available whenever the query is made. If the PingFederate OIDC policy uses an attribute source lookup and has issuance criteria configured to only issue a token if the account is enabled, enabling this web session option allows PingAccess to terminate the session the next time the user accesses a protected resource if the user's account was disabled in the user datastore. The **Refresh User Attributes Interval** determines the length of time the user data is cached, so the effect of a change that results in a session being terminated may take up to 60 seconds (by default) to take effect. Changing this setting can affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources. This option is selected by default.
Cache User Attributes	When **Cache User Attributes** is enabled, PingAccess caches user attributes internally for use in policy decisions. By doing this, an attribute list that is longer than the maximum cookie size can contain information used to evaluate access requests. In practice, this is 4096 bytes, although the maximum cookie size can vary depending on the browser. When this option is disabled, user attribute data is encoded, signed or encrypted, depending on the web session cookie type, and stored in the browser's cookie store. The information is sent from the browser back to PingAccess with each request. Changing this setting can affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources. This option is not selected by default.
Request Preservation	Select **Request Preservation** to specify the type of request data to be preserved if the user is redirected to an authentication page when submitting information to a protected resource.

Advanced setting	Description
	Available options are **None**, **POST**, or **POST and Fragment**.
Web Storage	Specify the type of **Web Storage** for request preservation data. Use **Session Storage**, and use **Local Storage** if it is common for users to use Internet Explorer with security zones enabled and PingFederate is in a different zone than PingAccess.

12. Click **Save**.

OpenID Connect login types

OpenID Connect (OIDC) supports three login types that define how the user's identity is verified based on authentication performed by an OpenID provider, and how additional profile claims are obtained.

Three OIDC sign-on profiles are supported: `Code`, `POST`, and `x_post`.

Login type	Description
Code	A standard OIDC login flow that provides confidentiality for sensitive user claims. In this profile the relying party, PingAccess, makes multiple back-channel requests to exchange an authorization code for an ID token. Then PingAccess exchanges an access token for additional profile claims from the **UserInfo** endpoint at the provider, PingFederate. This login type is for maximum security and standards interoperability.
POST	A login flow that uses the `form_post` response mode. This flow follows the *OAuth 2.0 Form Post Response Mode* draft specification. This option requires PingFederate 7.3 or later. A form auto-POST response containing the ID token, including profile claims, is sent to PingAccess from PingFederate through the browser after authentication. Back-channel communication between PingAccess and PingFederate is required for key management in order to validate ID tokens. This login type is for maximum performance in cases where the exchanged claims do not contain information that should be hidden from the end user. Select the **Implicit** grant type when configuring the OAuth Client within PingFederate. For more information, see *Configuring a Client*. The ID token-signing algorithm in PingFederate must be set to either one of the ECDSA algorithms or one of the RSA algorithms.
x_post	A login flow based on OIDC that passes claims from the provider through the browser. Select the **Implicit** grant type and use either one of the ECDSA algorithms or one of the RSA algorithms as the ID token-signing algorithm. **Note:** If you are using PingFederate 7.3 or later in the environment, use `POST` rather than `x_post`, which was defined by Ping Identity prior to the development of the OAuth 2.0 Form Post Response Mode draft specification.

Editing web sessions

Edit the properties of an existing web session in PingAccess.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
2. Click to expand the web session you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits to the web session. Click **Save**.

Deleting web sessions

Delete an existing web session in PingAccess.

About this task

If the web session is currently associated with an application, you cannot delete it.

Steps

1. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
2. Click to expand the web session you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Token validation

You can configure API and Web + API applications to use access token validators to locally verify signed and encrypted access tokens. This feature works in conjunction with token providers that support JSON web signature (JWS) and JSON web encryption (JWE) validation.

> **Tip:**
>
> When using PingFederate as the token provider for this feature, export the `Generated: ENGINE` keypair from PingAccess, located under **Security** # **Key Pairs**, and import to PingFederate trusted certificate authorities (CAs).

Adding access token validators

Add an access token validator to verify signed or encrypted access tokens in PingAccess.

Steps

1. Click **Access** and then go to **Token Validation** # **Access Token Validators**.
2. Click **+ Add Access Token Validator**.
3. In the **Name** field, enter a name for the token validator.
4. From the **Type** list, select the type of key you want to validate.

   The type of key is specified in the token provider configuration.

   > **Note:**
   >
   > For more information about configuring PingFederate, see *Configure JSON token management*.

5. Optional: In the **Description** field, enter a description for the token validator.

6. In the **Path** field, specify the endpoint path used to verify the signature.

   This entry must start with a forward slash (/), and must not end with a forward slash (/). Host and port are derived from PingFederate token provider configuration. A query string is permitted in the path.
7. Optional: In the **Subject Attribute Name** field, enter the attribute expected as the subject.

   If the specified subject attribute name is not present in the token, validation will fail.
8. Optional: In the **Issuer** field, enter the expected value of the issuer to include in the access token.

   If configured, and the value is not present in the token, validation will fail.
9. Optional: In the **Audience** field, specify the audience value to include in the access token.

   If configured, and the value is not present in the token, validation will fail.
10. Click **Save**.

Editing access token validators

Edit an existing access token validator in PingAccess.

Steps

1. Click **Access** and then go to **Token Validation** # **Access Token Validators**.
2. Click to expand the access token validator you want to edit.
3. Click the **Pencil** icon.
4. Make the desired edits. Click **Save**.

Deleting access token validators

Delete an existing access token validator in PingAccess.

Steps

1. Click **Access** and then go to **Token Validation** # **Access Token Validators**.
2. Click to expand the access token validator you want to delete.
3. Click the **Delete** icon.
4. To confirm your selection, click **Delete**.

Configuring OAuth key management settings

Configure settings for OAuth key management in PingAccess.

Steps

1. Click **Access** and then go to **Token Validation** # **OAuth Key Management**.
2. Choose to enable or disable key rolling:
   Choose from:
   - To enable key rolling, select the **Key Roll Enabled** check box.
   - To disable key rolling, clear the **Key Roll Enabled** check box.
3. To specify the interval at which you want to roll keys, enter a value (in hours) in the **Key Roll Enabled (H)** field.
4. From the **Signing Algorithm** list, select a signing algorithm to protect the integrity of the token when you use private key JSON web token (JWT) OAuth client authentication.

   If you select **Automatic**, you will use the algorithm specified in the OpenID provider metadata.
5. Click **Save**.

Unknown resources

Unknown resources in PingAccess are resources that are not associated with an application.

These settings define the error responses to be generated for requests that don't match the virtual host and context root of an application. Additionally, you can configure agents to allow unprotected access instead of returning an error response.

Configuring unknown resource management

Define the action to take when an unknown resource is requested.

Steps

1. Click **Access** and then go to **Unknown Resources** # **Error Responses**.
2. In the **Error Status Code** field, enter the error status code for the HTTP response.

   This must be a client or server error code in the 400 - 599 range.
3. In the **Error Template File** field, enter the name of the velocity error template file to use for generating the response body.

   This template file is located in the `<PA_HOME>/conf/template/` directory.
4. In the **Error Content Type** field, enter the file type of the response.
   Choose from:

   - HTML
   - JSON
   - TEXT
   - XML
5. To audit unknown resource activity, select the **Audit** check box.
6. Click **Save**.

Configuring agent defaults

Configure agents to allow unprotected access to unknown resources instead of returning an error response.

Steps

1. Click **Access** and then go to **Unknown Resources** # **Agent Defaults**.
2. Under the **Agent Default** header, specify the **Mode** that determines whether an agent should deny requests for unknown resources and generate an error response or allow requests to pass-through unfiltered.

   Individual agents might override this default setting.
3. Specify the default agent resource **Cache TTL (s)** (in seconds) to be used for unknown resources if you enable pass-through mode.
4. Click **Save**.

Security header

The **Security** header contains controls for certificates and key pairs.

The **Security** header contains these menu options:

- *Certificates* on page 309
- *Key pairs* on page 312

Certificates

Import certificates into PingAccess to establish anchors used to define trust to certificates presented during secure HTTPS connections.

Outbound secure HTTPS connections, such as communication with PingFederate for OAuth access token validation, identity mediation, and communication with a target site, require a certificate trusted by PingAccess. If one does not exist, communication is not allowed.

Certificates used by PingAccess can be issued by a certificate authority (CA) or self-signed. CA-issued certificates are recommended to simplify trust establishment and minimize routine certificate management operations. Implementations of an X.509-based PKI (PKIX) typically have a set of root CAs that are trusted, and the root certificates are used to establish chains of trust to certificates presented by a client or a server during communication.

The following formats for X.509 certificates are supported:

- Base64 encoded DER (PEM)
- Binary encoded DER

A Certificate Group is a trusted set of anchor certificates used when authenticating outbound secure HTTPS connections. The Java trust store group contains all the certificates included in the keystore located in the Java installation at `$JAVA_HOME/lib/security/cacerts`. This group of certificates contains well-known, trusted CAs. If you are connecting to sites that make use of certificates signed by a CA in the Java trust store, you do not need to create an additional trusted certificate group for that CA. You cannot manage the Java trust store group from the PingAccess administrative console. Expand a section for steps to import and manage certificates and create and manage trusted certificate groups.

Importing certificates
Import a new certificate.

Steps

1. Click **Security** and then go to **Certificates** # **Certificates**.
2. Click **+ Add Certificate**.
3. In the **Name** field, enter a name for the certificate.
4. To select the certificate, click **Choose File**.
5. To import the certificate, click **Add**.

> **Note:**
>
> If the certificate is either expired or not yet valid, PingAccess displays a warning, but the import will proceed.

Result: A new certificate row appears on the **Certificates** window.

Deleting certificates
Delete an existing certificate.

Steps

1. Click **Security** and then go to **Certificates** # **Certificates**.
2. Expand the certificate you want to delete.
3. Click the **Delete** icon.

4. When prompted, click **Delete** to confirm the deletion request.

> **Info:**
>
> If the certificate is associated with a trusted certificate group, you cannot delete it.

Creating trusted certificate groups
Create a new trusted certificate group.

Steps

1. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
2. Click **+ Add Trusted Certificate Group**.
3. Drag a certificate into the box that appears.
4. In the **Name** field, enter a name for the group.
5. To set the new group to include the Java Trust Store group, select the **Use Java Trust Store** check box.

   Select this option if you create your own intermediate certificate authority (CA) certificate that is signed by a well-known CA in the Java Trust Store.
6. To allow PingAccess to ignore date-related errors for certificates that are not yet valid or have expired, select the **Skip certificate date check** check box.
7. To check the client certificate revocation status using certificate revocation list (CRL), select the **CRL checking** check box.
8. To check the client certificate revocation status using Online Certificate Status Protocol (OCSP), select the **OCSP** check box.

   > **Note:**
   >
   > If both CRL checking and OCSP are enabled, OCSP checking is used preferentially, and CRL checking is used if OCSP fails.
9. To deny access when any certificate in the certificate chain cannot be verified using its CRL endpoint, select the **Deny when unable to determine revocation status** check box.
10. To validate client certificate chains that are not in the standard order, such as a reversed certificate chain of `[root, intermediate, leaf]`, select the **Validate disordered certificate chains** check box.
11. To skip validation of any CA certificates configured in the trusted certificate group and their subsequent chain of issuers when trusted CA certificates are found in the client certificate chain, select the **Bypass trust anchor validation** check box.
12. Click **Add**.

13. Optional: Add additional certificates to the new trusted certificate group by dragging them into the group.

> **Note:**
>
> PingAccess has increased `WARN` logging during the certificate revocation check. You can adjust the log level using the AsyncLogger in `log4j2.xml` (search for "Certificate Revocation").
>
> A commented out `JAVA_SECURITY_OPTS` line is shipped as part of the `run.sh` and `run.bat` scripts.
>
> Uncommenting the `JAVA_SECURITY_OPTS` line enables extra java security logging/debugging for the PKIX CertPathValidator and CertPathBuilder implementations. You can use the `ocsp` option with the `certpath` option for OCSP protocol tracing.

Adding certificates to trusted certificate groups

Add a certificate to an existing trusted certificate group.

Steps

1. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
2. Drag a certificate into an existing trusted certificate group.

Editing trusted certificate groups

Edit the properties of an existing trusted certificate group.

Steps

1. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
2. Expand the trusted certificate group you want to edit.

Choice	Action
Add a certificate to the group	Drag it into the group from the certificate list.
Delete a certificate from the group	Click **–** to the right of the certificate.
Edit the trusted certificate group parameters	Click the **Edit** icon and then make your changes. If you edit these options, click **Save** to save them.

Removing certificates from trusted certificate groups

Remove a certificate from a trusted certificate group.

Steps

1. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
2. Click to expand the trusted certificate group containing the certificate you want to remove.
3. Click the **–**icon next to the certificate you want to remove.

Deleting trusted certificate groups

Delete an existing trusted certificate group.

Steps

1. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
2. Click to expand the trusted certificate group you want to delete.
3. Click the **Delete** icon.
4. To confirm the deletion request, click **Delete**.

Key pairs

PingAccess provides an interface for creating and managing key pairs, which are required for secure HTTPS communication.

A key pair includes a private key and an X.509 certificate. The certificate includes a public key and the metadata about the owner of the private key.

The user interface displays a list of existing key pairs. You can search for key pairs using the **Search** bar or filter the list using the **Filters** list.

PingAccess listens for client requests on the administrative console port and on the PingAccess engine port. To enable these ports for HTTPS, the first time you start up PingAccess, it generates and assigns a key pair for each port.

Additionally, key pairs are used by the mutual TLS site authenticator to authenticate PingAccess to a target site. When initiating communication, PingAccess presents the client certificate from a key pair to the site during the mutual TLS transaction. The site must be able to trust this certificate in order for authentication to succeed.

Info:

Ensure that the administrative console node and engines in a cluster have the same cryptographic configuration. For example, if you generate an elliptic curve key pair on the administrative console and the engines in the cluster are not configured to support elliptic curve key pairs, then the engines are not able to use that key pair for the engine HTTPS listeners or as the key pair in a mutual TLS site authenticator. Cryptographic configuration differences are often caused by having a Java cryptographic extension with limited strength providers installed. For more information, see *Oracle Java documentation*.

Importing existing key pairs

Import a key pair from a PKCS#12 or PEM-encoded file.

About this task

Note:

If PingAccess is running in Federal Information Processing Standards (FIPS) mode, only PEM-encoded key pairs can be imported or exported.

For more information, see *Managing Federal Information Processing Standards (FIPS) mode* on page 149.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click **Import**.
3. In the **Alias** field, enter a name that identifies the key pair.

   Special characters and spaces are allowed. This name identifies the key pair when assigning the key pair to various configurations such as *HTTPS Listeners*.
4. In the **Password** field, enter a password used to protect the key pair file.

   PingAccess uses the password to read the file.
5. Click **Choose File** to locate the key pair file.

6. Click **Save** to import the file.

> **Note:**
>
> If the key pair is either expired or not yet valid, PingAccess displays a warning, but the import will proceed. If the key pair cannot be read using the specified password, the import fails.

PEM-encoded format

PEM-encoded key pair files use this format for the key and certificates.

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
Base64-encoded private key
(Private Key: domain_name.key)
-----END ENCRYPTED PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
(Primary SSL certificate: domain_name.crt)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
(Intermediate certificate: Intermediate.crt)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded certificate
(Root certificate: Root.crt)
-----END CERTIFICATE-----
```

Generating new key pairs

Generate a key pair and self-signed certificate.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click **+ Add Key Pair**.
3. In the **Alias** field, enter an internal alias for the key pair.
4. In the **Common Name** field, enter the common name identifying the certificate.
5. Optional: If the key pair is going to be used for incoming requests on multiple hosts or multiple IP addresses, enter additional **Subject Alternative Names** to meet those requirements.
6. In the **Organization** field, enter the organization or company name creating the certificate.
7. Optional: In the **Organization Unit** field, enter the unit within the organization.
8. Optional: In the **City** field, enter the city or primary location where the organization operates.
9. Optional: In the **State** field, enter the state or political unit where the organization operates.
10. In the **Country** field, enter the country where the organization operates.
11. In the **Valid Days** field, enter the number of days that the certificate is valid.
12. Optional: From the **Selected HSM** list, select a hardware security module in which to store the key pair.
13. In the **Key Algorithm** section, select an algorithm.
    a. From the **Key Size** list, select the number of bits in the key.
    b. From the **Signature Algorithm** list, select the signature algorithm to use for the key.
14. Click **Save**.

Managing certificates for key pairs with ACME

Manage key pairs using the automatic certificate management environment (ACME) protocol, which automatically obtains and renews certificates indirectly signed by a well-known trust anchor.

About this task

The ACME protocol is an Internet Engineering Task Force (IETF) proposed standard protocol that automates the signing of TLS certificates by a certificate authority (CA).

The ACME certificate management option in PingAccess uses the staging *Let's Encrypt* ACME CA by default.

> **Note:**
>
> The Let's Encrypt staging server that PingAccess uses by default has more lenient rate limits, but it doesn't generate functional certificates, to support its use for testing purposes. For more information about rate limits, see the *Let's Encrypt documentation*.
>
> After testing your environment, you must switch to a production server using the PingAccess Administrative API.
>
> 1. Use a `GET` call to `/pa-admin-api/v3/acme/servers` to retrieve the ID of a production server.
> 2. Use a `PUT` call to `/pa-admin-api/v3/acme/servers/default` to set the production Let's Encrypt server as the default.
>
> To add more ACME servers, use a `POST` call to `/pa-admin-api/v3/acme/servers`. See the *Administrative API endpoints* on page 156 documentation for more information about the administrative API endpoints.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click the **Pencil** icon, and then click **Manage with ACME** for the key pair.

   Result: The ACME status changes to **Pending**. When the protocol has completed, the status changes to **Valid** if the protocol completed successfully.

Downloading certificates

Download a certificate when you need to configure a peer to trust a certificate used by PingAccess.

About this task

Download the certificate for the key pair used by a mutual TLS site authenticator and configure the target site to trust the certificate.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Locate the row corresponding to the key pair, then click the **Pencil** icon. Click **Download Certificate** .

   Result: Your browser downloads the certificate and saves it in your local file system.

Generating certificate signing requests

Generate a certificate signing request (CSR) to establish more security and trust than using a self-signed certificate.

Steps

1. Click **Security** and then go to **Key Pairs**.

2. Click the **Pencil** icon, and then click **Generate CSR** for the certificate you want to generate a CSR for.

   Result: PingAccess generates a CSR file, and your browser downloads it.
3. Provide this file to a certificate authority (CA).

   The CA signs the file and provides a CSR response that you can upload and use to replace the self-signed certificate. If the CA is well known, its certificates are installed by default in most browsers, and the user is not prompted to trust an unknown certificate.
4. When you receive the CSR response, follow the instructions in *Importing certificate signing request responses* on page 315.

Importing certificate signing request responses
Import a CSR response to replace the self-signed certificate in a key pair.

Before you begin
Before you import the CSR response, import the signing certificate authority (CA) certificate into PingAccess and add it to a *Trusted Certificate Group*.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click the **Pencil** icon, and then click **CSR Response** for the key pair the CSR applies to.
3. To select the CSR response file, under the **CSR Response File** heading, click **Choose File**.
4. Optional: To choose one or more chain certificate files associated with this key pair, under the **Chain Certificates** heading, click **Choose Files**.
5. Click **Save**.

Assigning key pairs to virtual hosts
Assign a key pair to a virtual host.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click the **Pencil** icon, and then click **Assign Virtual Host** for the key pair.
3. Use the **Virtual Hosts** list select the virtual hosts for which the key pair should be used.

   > **Note:**
   >
   > When you assign a key pair to a virtual host, the key pair is also assigned to all other virtual hosts with the same hostname.
4. Click **Save**.

Assigning key pairs to HTTPS listeners
Assign a new key pair for any of the active HTTPS listeners.

About this task

For details about the available listeners, see *HTTPS listeners* on page 316.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click the **Pencil** icon, and then click **Assign HTTPS Listener** for the key pair.

3. Use the **Listeners** list to select the HTTPS listeners for which the key pair should be used.

> **Note:**
>
> Changes to an HTTPS listener's active key pair are used for all new connections, but existing connections continue to use the old configuration.

4. Click **Save**.

HTTPS listeners

PingAccess listens for HTTPS requests on the Admin, Engine, and Agent ports in all deployments, and on the Config query port in clustered deployments.

A key pair must be assigned to each listener. By default, the listeners are configured for HTTPS and use pregenerated key pairs associated with `localhost`.

HTTPS Listener Descriptions

HTTPS Listener	Description
Admin	Listens for requests for the administrative user interface and the PingAccess REST APIs.
Engine	Listens for HTTP or HTTPS requests that are proxied to target web servers associated with *Sites*.
Agent	Listens for requests from PingAccess agents.
Sideband	Listens for requests from sideband clients.
Config query	Listens for requests for configuration information from replica administrative nodes and engine nodes in clustered deployments.

If you configure a trusted certificate group for a virtual host, or configure an engine key pair to associate it with a virtual host, those settings are used instead of any applicable HTTPS listeners or engine listeners for the virtual host.

Cipher suite ordering for HTTPS listeners

PingAccess supports the use of a defined order for cipher suite usage to help ensure the most secure cipher suites are used first, regardless of the client request. The cipher suite order is defined in `<PA_HOME>/conf/run.properties` using the `tls.default.cipherSuites` property.

On new installs, or in the case of an upgrade to release 5.1 or later, this behavior is the default. You can disable this behavior and specify PingAccess to use the order provided by the client by setting `useServerCipherSuiteOrder` to `false` using the PingAccess API `/httpsListeners` endpoint.

Adding certificates to key pairs

Add a certificate to an existing key pair by starting with a leaf certificate, then adding the intermediate and root certificates as required.

About this task

> **Note:**
>
> To modify the certificates included in a chain, remove the certificates from the key pair and add them again or delete the certificate and recreate it by importing a new certificate file and adding certificates to the key pair.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click to expand an existing key pair.
3. From the **Key Pair Chain Certificate** list, select **Add Certificate**.
4. To browse for and select the certificate file, click **Choose File**.
5. Click **Add**.

Removing certificates from key pairs
Remove a certificate from an existing key pair.

About this task

> **Note:**
>
> This procedure removes the last certificate in the chain. Certificates can only be removed in reverse order.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click to expand an existing key pair.
3. To remove the last certificate in the chain, click the **Delete** icon.
4. To confirm, click **Delete**.

Deleting key pairs
Delete an existing key pair.

About this task

> **Note:**
>
> If a key pair is currently in use, you can't delete it.

Steps

1. Click **Security** and then go to **Key Pairs**.
2. Click to expand the key pair you want to delete.
3. Click the **Delete** icon.
4. To confirm the request, , click **Delete** when prompted.

Hardware security module providers

Configure PingAccess to use a hardware security module (HSM) to generate and store key pairs to be used in SSL/TLS cryptographic operations.

PingAccess uses the HSM agent to direct the creation of new key pairs on the HSM. When you create a key pair, you can specify that it should be created on the HSM.

For more information, see the following topics:

- *Adding an AWS CloudHSM provider* on page 318
- *Adding a Safenet Luna provider* on page 318
- *Editing an HSM provider* on page 319
- *Deleting an HSM provider* on page 319

Adding an AWS CloudHSM provider

Add an Amazon Web Services (AWS) CloudHSM provider to begin using hardware security module (HSM)-stored key pairs in PingAccess.

Before you begin

- Configure your hardware security module. For more information, see the *Amazon documentation*.
- Download the AWS CloudHSM software library for Java version 3.1.2, install it, and move the `Cloudhsm-3.1.2.jar` file from the `/opt/cloudhsm/java/` directory to the `deploy` directory on the PingAccess system. For more information, see the *Install and Use the AWS CloudHSM Software Library for Java* procedure. If 3.1.2 is not the latest version of CloudHSM, you can download it from the *Client and Software Version History*.
- Verify that you are using Oracle Java SE Runtime Environment (Server JRE) 8.
- Verify that your PingAccess deployment is running in the same AWS EC2 instance as the CloudHSM client.

Steps

1. Click **Security** and then go to **HSM Providers**.
2. Click **+ Add HSM Provider**.
3. In the **Name** field, enter a name for the HSM provider.
4. From the **Type** list, select **AWS CloudHSM Provider**.
5. In the **User** field, enter a user name for connecting to the HSM provider.
6. In the **Password** field, enter a password for connecting to the HSM provider.
7. Optional: In the **Partition** field, enter the partition to use on the HSM provider.
8. Click **Save**.
9. Restart PingAccess.

Note:

The following are known issues:

- `RSASSA-PSS` signing algorithms fail with `Java8u261` or later. HSM vendors and core Java use different naming conventions for the `RSASSA-PSS` algorithm.
- PingAccess Cloud HSM functionality works in FIPS mode but not in regular mode for `Java8u261` and later.

To bypass the known issues, a user can edit the `additional.security.jdk.tls.disabledAlgorithms` in the `run.properties` file. For more information, see the following example:

```
additional.security.jdk.tls.disabledAlgorithms=RSASSA-PSS, TLSv1.3
```

Adding a Safenet Luna provider

Add a Safenet Luna provider to begin using hardware security module (HSM)-stored key pairs in PingAccess.

Before you begin

- Configure your hardware security module.
- Configure a Luna client on the PingAccess system. The PingAccess service must have full permissions over the client.

- Move the `/usr/safenet/lunaclient/lib/libCryptoki2_64.so` library on Linux systems, or the `\Program Files\SafeNet\LunaClient\win32\cryptoki.dll` library on Windows systems, to the `deploy` directory on the PingAccess system.

Steps

1. Click **Security** and then go to **HSM Providers**.
2. Click **+ Add HSM Provider**.
3. In the **Name** field, enter a name for the HSM provider.
4. From the **Type** list, select **Safenet Luna Provider**.
5. In the **Slot ID** field, enter the slot ID of the HSM slot to use.
6. In the **Library** field, enter the name of the library you copied from the Luna client to the `deploy` directory.
7. In the **Password** field, enter a password for connecting to the HSM provider.
8. Click **Save**.
9. Restart PingAccess.

Editing an HSM provider
Edit the properties of an existing hardware security module (HSM) provider.

Steps

1. Click **Security** and then go to **HSM Providers**.
2. Click to expand the HSM provider, and then click the **Pencil** icon.
3. Edit one or more properties.
4. Click **Save**.

Deleting an HSM provider
Delete an existing hardware security module (HSM) provider.

Steps

1. Click **Security** and then go to **HSM Providers**.
2. Click to expand the HSM provider, and then click the **Delete** icon.
3. Click **Delete**.

Settings header

The **Settings** header contains menu options related to internal settings and configuration for PingAccess, such as clustering, networking, and authentication.

The **Settings** header contains these menu options:

- *Clustering* on page 320
- *HTTP requests* on page 324
- *Networking* on page 326
- *Admin authentication* on page 331
- *System* on page 339

Clustering

You can configure PingAccess in a clustered environment to provide higher scalability and availability for critical services.

> **Important:**
>
> Availability and performance are often at opposite ends of the deployment spectrum. You might need to make some configuration tradeoffs that balance availability with performance to accommodate specific deployment goals.

To learn more about how to set up cluster components in the PingAccess administrative console, choose one of the following sections:

- For more information on how to set up and manage the administrative nodes in a clustered PingAccess deployment, see *Administrative nodes* on page 322.
- For more information on how to set up and manage the engine nodes in a clustered PingAccess deployment, see *Engines* on page 320.

> **Attention:**
>
> These topics contain instructions to set up specific pieces of a cluster. For more information about clustering and setting up an efficient clustered environment, you should review the PingAccess *Clustering in PingAccess* on page 156 *Reference Guide*. This guide provides information about subjects such as cluster node statuses, node failure implications, and which files could affect cluster settings.

Engines

Configure and manage the engine nodes in a cluster in PingAccess.

- *Configuring engine nodes* on page 320
- *Editing engine nodes* on page 321
- *Revoking access from an engine node* on page 322
- *Removing engine nodes* on page 322

Configuring engine nodes

Configure an engine node as part of a cluster in PingAccess.

Before you begin

Make sure that you've configured an administrative node and a replica administrative node.

> **Note:**
>
> For a comprehensive overview of the steps necessary to set up a clustered environment, see *Configuring a PingAccess cluster* on page 159 in the *Clustering in PingAccess* on page 156 *reference guide*.

Steps

1. Click **Settings** and then go to **Clustering** # **Engines**.
2. To configure a new engine, click **+ Add Engine**.
3. In the **Name** field, enter a name for the engine.

   Special characters and spaces are allowed.
4. Optional: In the **Description** field, enter a description of the engine.

5. If applicable, specify an **HTTP Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. To create an HTTP proxy, click **+Create**.
6. If applicable, specify an **HTTPS Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. To create an HTTPS proxy, click **+Create**.
7. Specify an **Engine Trusted Certificate** if a TLS-terminating network appliance, such as a load balancer, is placed between the engines and administrative node.

   > **Note:**
   >
   > Select the certificate that the network appliance uses. The certificate helps establish a secure HTTP connection with the administrative node.

8. To generate and download a public and private key pair into the `<enginename>_data.zip` file for the engine, click **Save & Download**.

   This file is prepended with the name you give the engine. Depending on your browser configuration, you might be prompted to save the file.
9. Copy the `.zip` file to the `<PA_HOME>` directory of the corresponding engine in the cluster and extract it.

   The engine uses these files to authenticate and communicate with the administrative console.

   > **Tip:**
   >
   > You can generate a new key for the engine at any time, just repeat steps 8-9.
   >
   > a. Click **Save & Download**.
   > b. Extract the `<enginename>_data.zip` file within the engine's `<PA_HOME>` directory.
   >
   > When the engine node starts up and begins using the new configuration files, PingAccess deletes the old key.

10. On Linux systems running the PingAccess engine, run the **`chmod 400 conf/pa.jwk`** command on the `pa.jwk` file after you've extracted the `.zip` file.

    Result:

    The `pa.jwk` becomes read only, preventing it from being overwritten accidentally.
11. Start each engine.

Next steps

If you specified any proxies, enable the **Use Proxy** option for any sites, token providers, and third party services that require the use of a proxy. For more information, see *Adding sites* on page 247 and the *Token provider* on page 342 section.

Editing engine nodes

Edit the name and description of an engine node within your cluster, and download a new public key if necessary.

Steps

1. Click **Settings** and then go to **Clustering** # **Engines**.
2. Expand the node you want to edit.

3. Click the **Pencil** icon.
4. Edit the node **Name** or **Description**, as appropriate.
5. If a new public key is needed, click **Save & Download**. If not, click **Save**.

Revoking access from an engine node
Remove an engine node's access to the administrative node.

About this task

If an engine node is compromised, you can delete its public keys from the administrative node to prevent it from accessing the administrative node. You can recreate these keys after you have recovered the engine node.

Steps

1. Click **Settings** and then go to **Clustering** # **Engines**.
2. Expand the engine node that you want to remove from the cluster and click the **Pencil** icon to edit it.
3. In the **Public Keys** section, click **Delete** to revoke the engine node's access to the administrative node.

   > **Info:**
   >
   > You can use the **Save & Download** button to create a new key for the engine. For more information, see *Configuring engine nodes* on page 320.

4. Click **Save**.

Removing engine nodes
Remove an engine node from the cluster.

Steps

1. Click **Settings** and then go to **Clustering** # **Engines**.
2. To permanently remove all references to the node from the cluster, expand the engine node you want to delete and click the **Delete** icon.
3. In the confirmation window, click **Delete**.

Administrative nodes
Configure and manage the administrative nodes in a cluster in PingAccess.

- *Configuring administrative nodes* on page 322
- *Configuring replica administrative nodes* on page 323

Configuring administrative nodes
Configure one PingAccess node as the administrative node.

About this task

> **Warning:**
>
> If you are promoting a replica administrative node to an administrative node, remove the `bootstrap.properties` file from the replica administrative node.

In this procedure, you can specify an HTTP or HTTPS proxy. A proxy configuration defined in a properties file, such as `bootstrap.properties` or `run.properties`, takes precedence over a proxy configuration defined in the PingAccess administrative console or API.

If you configure a proxy on a replica administrative node and need to fail over to that node, make sure that the administrative node has the same proxy configuration as the replica administrative node before you remove the `bootstrap.properties` file from the replica administrative node.

For a comprehensive overview of the steps necessary to set up a clustered environment, see *Configuring a PingAccess cluster* on page 159 in the *Clustering in PingAccess* on page 156 *reference guide*.

Steps

1. Click **Settings** and then go to **Clustering # Administrative Nodes**.
2. In the **Host** field, in the **Primary Administrative Node** section, enter the host and port for the administrative console.

   The default is `localhost:9000`.
3. If applicable, specify an **HTTP Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. Click **+ Create** to create an HTTP proxy.
4. If applicable, specify an **HTTPS Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. Click **+ Create** to create an HTTPS proxy.
5. Click **Save**.

Next steps

If you specified any proxies, enable the **Use Proxy** option for any sites, token providers, and third party services that require the use of a proxy. For more information, see *Adding sites* on page 247 and the *Token provider* on page 342 section.

Configuring replica administrative nodes

Configure one PingAccess node as a replica administrative node to provide an alternative if the administrative node fails.

About this task

The key pair that you create for the CONFIG QUERY listener must include both the administrative node and the replica administrative node. To make sure the replica administrative node is included, you can either use a wildcard certificate or define subject alternative names in the key pair that use the replica administrative node's DNS name. For more information, see step 2c in *Configuring a PingAccess cluster* on page 159.

Important:

If you use a replica administrative node in your configuration, configure the replica administrative node before defining the engine nodes, or the `bootstrap.properties` files generated for the engine nodes will not include information about the replica administrative node.

Steps

1. Click **Settings** and then go to **Clustering # Administrative Nodes**.
2. In the **Host** field, in the **Replica Administrative Node** section, enter the host and port for the replica administrative node.

   This name and port pair must match either a subject alternative name in the key pair or be considered a match for the wildcard specified if the key pair uses a wildcard in the common name.

3. If applicable, specify an **HTTP Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. Click **+ Create** to create an HTTP proxy.
4. If applicable, specify an **HTTPS Proxy** for the engine.

   For more information about creating proxies, see *Adding proxies* on page 330.

   a. Click **+ Create** to create an HTTPS proxy.
5. Specify the **Replica Administrative Node Trusted Certificate** if a TLS-terminating network appliance, such as a load balancer, is placed between the engines and administrative node.

   > **Note:**
   >
   > Select the certificate that the network appliance uses. The certificate helps establish a secure HTTP connection with the administrative node.
6. Click **Save & Download** to download the `*<replicaname>*_data.zip` file for the replica administrative node.

   PingAccess automatically generates and downloads a public and private key pair into the `bootstrap.properties` file for the node. The public key is indicated in this window.
7. Copy the downloaded file to the replica administrative node's *`<PA_HOME>`* directory and extract it.
8. If the replica administrative node is running on a Linux host, run the command **`chmod 400 conf/pa.jwk`**.
9. Edit *`<PA_HOME>`*`/conf/run.properties` on the replica administrative node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`.

   This property is case-sensitive.
10. Start the replica administrative node.
11. Verify replication has completed by monitoring the *`<PA_HOME>`*`/log/pingaccess.log` file and looking for the message `Configuration successfully synchronized with administrative node`.

HTTP requests

The settings for HTTP requests are used to match a served resource with the originating client when one or more reverse proxies are between the client and the served resource.

When a reverse proxy sits between the client and the PingAccess server or agent, the additional proxy might be identified as the client. You can configure such proxies to inject additional headers to relay the originating client address.

Host Source and **Protocol Settings** allow PingAccess to determine the effective URL of a request using a list of alternative headers. PingAccess uses this URL to apply security policies and perform HTTP redirects.

When the PingAccess agent is behind a load balancer that is performing HTTPS offload, the load balancer must inject the Host Source and Protocol Source headers.

IP Source

: Lets you specify an ordered list of header names to identify the source IP address. By default, `X-Forwarded-For` is configured as a heading.

Host Source

: Lets you specify an ordered list of header names to identify the host source name. The Host Source options are only valid in proxy deployments. By default, `X-Forwarded-Host` and `Host` are configured as headings.

Protocol Source

Can be used to define the header that identifies the protocol used for the original request. The default value is `X-Forwarded-Proto`.

Configuring alternative IP source headers

Configure an ordered list of Header names to identify the source IP address.

Steps

1. Click **Settings** and then go to **HTTP Requests** # **IP Source**.
2. From the **Header Names** list, search for the header names you want to add..

   You can add a header by clicking **+ Header Names** or delete a header by clicking the **Delete** icon.
3. In the **List Value Location** field, select either:
   Choose from:

   - `First`
   - `Last`

   When a list of values is in the header, this step determines if the first value or the last value in the list should be used as the IP Source value. The default value is `Last`.
4. To determine if you should use the upstream IP address for rule evaluation, if none of the listed headers are present in the request, either:
   Choose from:

   - Select the **Fallback to Last Hop IP** check box.
   - Clear the **Fallback to Last Hop IP** check box.

   If this value is disabled and no headers match, the network range rule will return a `Forbidden` status.

   > **Note:**
   >
   > This option uses the specified headers in an agent deployment, and uses networking layer information in a proxy deployment.
5. Click **Save**.

Configuring alternative host source headers

Configure an ordered list of Header names to identify the host source name in PingAccess.

About this task

Host source settings are valid for proxy deployments only.

Steps

1. Click **Settings** and then go to **HTTP Requests** # **Host Source**.
2. To enter a header name to search for in the **Header Names** list, click **Header Names**.

   You can add a header by clicking **+ Header Names** or delete a header by clicking the **Delete** icon.
3. In the **List Value** field, to determine (when a list of values is in the header) if the first or last value in the list should be used as the **Host Source** value, select one of the following:
   Choose from:

   - `First`
   - `Last`

   The default value is `Last`.

4. Click **Save**.

Configuring alternative protocol source headers

Configure a Header name for a protocol source in PingAccess.

Steps

1. Click **Settings** and then go to **HTTP Requests** # **Protocol Source**.
2. In the **Header Name** field, enter a header name.
3. Click **Save**.

Networking

The **Networking** tab controls how PingAccess manages network requests and load balancing.

For information related to networking in PingAccess, choose from one of the following sections:

- *Availability profiles* on page 326
- *Engine listeners* on page 327
- *Load balancing strategies* on page 328
- *Proxies* on page 330

Availability profiles

Availability profiles are used in a site configuration to define how PingAccess classifies a backend target server as failed. Sites require the selection of an availability profile, even if only one target is provided.

You can determine a connection failure based on whether a backend target is not responding, or on specified HTTP status codes that should be treated as failures of a specific backend target. For example, if a backend target is responding to requests with a `500 Server Error` status, you might consider that server down even though the web service is responsive.

If you specify multiple targets in a site configuration but a load balancing strategy is not applied, the availability profile will cause the first listed target in the site configuration to be used unless it fails. Secondary targets are only used if the first target is not available.

When you change the availability profile for a site, the new availability profile configuration only applies to new connections. Existing connections continue to use the previous availability profile configuration. To apply the new availability profile configuration to all connections for a site, you must restart the PingAccess engines.

Currently, the only availability profile type is **On-Demand**. You might want to create different profiles for different sites based on your site needs for retry counts, retry delays, timeouts, or HTTP status codes.

Creating availability profiles

Create an availability profile in PingAccess to define when a backend server is considered failed.

Steps

1. Click **Settings** and then go to **Networking** # **Availability Profiles**.
2. Click **+ Add Availability Profile**.
3. Enter a unique descriptive name for the profile.
4. Select the **On-Demand** type.
5. In the **Connect Timeout (ms)** field, enter the number of milliseconds to wait for a connection to be established to a back-end target.
6. In the **Pooled Connection Timeout (ms)** field, enter the number of milliseconds to wait before timing out the request for a pooled connection to the target site in the **Pooled Connection Timeout (ms)** field.

   Enter `-1` for no timeout.

7. In the **Read Timeout (ms)** field, enter the number of milliseconds to wait before timing out the read of the response from a target site.

   Enter `-1` for no timeout.
8. In the **Max Retries** field, enter the number of times to retry a connection to a back-end target before considering the target failed.
9. In the **Retry Delay (ms)** field, enter the number of milliseconds to wait between retries.
10. In the **Failed Retry Timeout (s)** field, enter the number of seconds to wait before trying a failed target again.
11. Optional: In the **Failure HTTP Status Codes** field, enter a list of HTTP status codes that should be considered as a failure.

    The sequence for this list is not important.
12. Click **Save**.

Editing availability profiles
Edit the properties of existing availability profiles in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Availability Profiles**.
2. Click to expand the desired profile you want to edit.
3. Click the **Pencil** icon.
4. Make the desired changes to the profile. Click **Save**.

Deleting availability profiles
Delete existing availability profiles in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Availability Profiles**.
2. Click to expand the profile you want to delete.
3. Click the **Delete** icon.
4. To confirm your selection, click **Delete**.

Engine listeners
PingAccess listens for engine traffic using the defined engine listeners.

The default listener port is 3000. You can add new engine listeners, or edit or delete existing engine listeners.

Defining engine listeners
Define a new engine listener in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Engine Listeners**.
2. Click **+ Add Engine Listener**.
3. In the **Name** field, enter a descriptive name for the listener.
4. In the **Port** field, enter the port the listener will open.

> **Note:**
>
> If you do not open the port in the system firewall, the listener will not be able to process any incoming requests.

5. If you want the port to listen for HTTP connections, clear the **Secure** option.

> **Note:**
>
> By default, engine listeners listen for HTTPS connections to protect sensitive data.

6. Optional: From the **Client Certificate Authentication** list, select a certificate authentication method.

   Assigning a certificate authentication method to an engine listener provides a mechanism to authenticate using client certificates during any request to the engine listener.

   You can select an existing trusted certificate group, or use one of the following options.

   Choose from:

   - **No Certificate Authentication** – Does not require certificate authentication.
   - **Java Trust Store** – Uses the Java Trust Store for certificate authentication.
   - **Trust Any –** Allows client authentication with any certificate including self-signed certificates.

   If you use the **Trust Any** method in production, you should log client certificates in the audit log.
7. Click **Save**.

Editing engine listeners

Modify and edit the properties of an existing engine listener in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Engine Listeners**.
2. In the **Engine Listeners** section, click to expand an existing engine listener you want to edit.
3. Click the **Pencil** icon.
4. Enter the desired changes. Click **Save**.

Deleting engine listeners

Delete an existing engine listener in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Engine Listeners**.
2. In the **Engine Listeners** section, click to expand an existing engine listener you want to delete.
3. Click the **Delete** icon.
4. To confirm your selection, click **Delete**.

Load balancing strategies

Load balancing strategies are used in a site configuration to distribute the load between multiple backend target servers. Load balancing settings are optional and only available if more than one target is listed for a site.

This functionality can replace a load balancer appliance between the PingAccess engine nodes and the target servers, allowing for a simpler network architecture.

The load balancing strategies currently available are `Header-Based` and `Round Robin`.

The `Header-Based` strategy requires the request to include a header that defines the target to select from the **Site** configuration. This strategy lets you fall back if the requested target is unavailable or if the header is missing from the request.

The `Round Robin` strategy has a sticky session option that permits a browser session to be pinned to a persistent backend target. This strategy works in conjunction with the availability profile to select a target based on its availability. The load balancer will not select a target that is in a failed state.

Configuring load balancing strategies

Create and configure a new load balancing strategy in PingAccess.

Steps

1. Click **Settings** and then go to **Networking # Load Balancing Strategies**.
2. Click **+ Add Load Balancing Strategy**.
3. Enter a unique descriptive name for the strategy.
4. In the the **Type** field, select one of the following types:
   Choose from:
   - `Header-Based`
   - `Round Robin`
5. Configure the options for the selected load balancing strategy type.
   Choose from:
   - For a `Header-Based` Load Balancing Strategy:
     a. In the **Header Name** field, enter the name of the header that contains the selected target host.
     b. To tell PingAccess to use the first available target defined for the site, click **Show Advanced** and select the **Fall Back to First Available Host** option if the target specified in the header is not available or if the header is not present in the request.

     > **Note:**
     >
     > If this option is not enabled and the specified target is not available or the request header is not present, the client will receive a `Service Unavailable` response.
   - For a `Round Robin` load balancing strategy:
     a. If browser sessions should not be pinned to a persistent backend target, clear the **Sticky Session Enabled** option.

        This option is enabled by default.
     b. If you enable the **Sticky Session Enabled** option, enter a cookie name to use in the **Cookie Name** field.

        The PingAccess engine uses this cookie to track the persistent backend targets for a session.

     > **Note:**
     >
     > When you define a web session, the **Cookie Name** field defines a cookie prefix to use. The rest of the cookie name comes from the **Audience** field in the web session.
6. Click **Save**.

Editing load balancing strategies

Edit the properties of an existing load balancing strategy in PingAccess.

Steps

1. Click **Settings** and then go to **Networking # Load Balancing Strategies**.
2. Click to expand the load balancing strategy you want to edit.
3. Click the **Delete** icon.
4. Make any desired changes to the profile.
5. To confirm your edits, click **Save**.

Deleting load balancing strategies
Delete an existing load balancing strategy in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Load Balancing Strategies**.
2. Click to expand the load balancing strategy you want to delete.
3. Click the **Delete** icon.
4. To confirm your changes, click **Delete**.

Proxies
The Proxies page lets you configure the forward proxy configuration used when PingAccess makes requests to sites or token providers.
Adding proxies
Add a forward proxy configuration to be used when PingAccess makes requests to sites or token providers.

Steps

1. Click **Settings** and then go to **Networking** # **Proxies**.
2. Click **+ Add Proxy**.
3. In the **Name** field, enter a name for the proxy configuration.
4. Optional: In the **Description** field, enter a description.
5. In the **Host** field, enter the host name for the forward proxy.
6. In the **Port** field, enter the port number for the forward proxy.
7. If the forward proxy requires authentication, select the **Requires Authentication** check box.
8. If required, enter the **Username** for the forward proxy.
9. If required, enter the **Password** for the forward proxy.
10. Click **Save**.

Next steps

If you have a clustered environment, configure your Admin and Engine nodes to use the proxy. If you are using a standalone environment, configure your Admin node to use the proxy. See *Configuring administrative nodes* on page 322 and *Configuring engine nodes* on page 320 for more information.

Then, enable the Use Proxy option for any sites, token providers, and third party services that require the use of a proxy. See *Adding sites* on page 247 and the *Token provider* on page 342 section for more information.

Editing proxies
Edit the properties for an existing proxy configuration in PingAccess.

About this task

Note:

If you edit a proxy configuration that is associated with an engine or replica administrative node, you must download and install a new configuration on those nodes.

Steps

1. Click **Settings** and then go to **Networking** # **Proxies**.
2. Click to expand an existing proxy configuration you want to edit.

3. Click the **Pencil** icon.
4. Edit the proxy as needed.
5. To confirm your edits, click **Save**.

Deleting proxies

Delete an existing proxy in PingAccess.

Steps

1. Click **Settings** and then go to **Networking** # **Proxies**.
2. Click to expand an existing proxy configuration you want to delete.
3. Click the **Delete** icon.
4. To confirm your selection, click **Delete**.

Admin authentication

Admin authentication controls the PingAccess administrator authentication method for the user interface and the Admin APIs.

Configuring basic authentication

Configure basic authentication for the administrative user interface in PingAccess.

About this task

The authentication default for the PingAccess administrative console is HTTP Basic Authentication. Basic Authentication uses the HTTP Authorization header to transmit the user name and password credentials. The PingAccess server response contains a `PA_UI` cookie, which is a signed JSON web token (JWT). Subsequent HTTP requests send this cookie for authentication rather than the less secure HTTP Authorization header.

Basic Authentication supports one user: Administrator. The Administrator user name cannot be changed. If you want to allow more than one user to access the admin UI, you should use single sign-on (SSO) authentication.

Steps

1. Click **Settings** and then go to **Admin Authentication** # **Basic**.
2. Click **Enable**.
3. Click **Save**.
4. Click **Settings** and then go to **Admin Authentication** # **UI Authentication**.
5. In the **Authentication Method** section, select **Basic Authentication**.
6. Click **Save**.

Changing the password for basic authentication

Change the password used for basic authentication in PingAccess.

Steps

1. Click **Settings** and then go to **Admin Authentication** # **Basic**.
2. Enter the current administrator password.

3. Enter and confirm the new password.

> **Important:**
>
> The new password must meet the configured password complexity rules defined in `pa.admin.user.password.regex` in `run.properties`.

4. Click **Save**.

Configuring API authentication

Configure authentication for the administrative API in PingAccess.

About this task

For more information on the PingAccess Administrative API, see *Administrative API Endpoints*.

You can configure roles for Administrative API users. Each role grants access to specific features.

- The Administrator role has full read and write access to the Admin API, unless the Platform Administrator role is enabled. If the Platform Administrator role is enabled, the Administrator only has read access to the Admin API endpoints under the `/auth`, `/users`, and `/environment` paths, and has both read and write access to all other endpoints.
- The Platform Administrator role has full read and write access to the Admin API. This role can be used with the Administrator role to grant full access to most features without the possibility of accidental lockout, with only the Platform Administrator able to change authorization configurations.
- The Auditor role has read access to all Admin API endpoints except for the `/config/*` , `/backup/`, and `/agent/*/config/` endpoints.

Steps

1. Click **Settings** and then go to **Admin Authentication** # **API Authentication**.
2. Go to **System** # **Admin Authentication** # **Admin API OAuth**.
3. To enable API OAuth authentication, select **OAuth**.
4. Click the **Properties** tab.
5. Optional: From the **Access Token Validator Type** list, select a local access token validator to use instead of remote token validation for Admin API authentication.

   If you select a local access token validator, the OAuth configuration does not require client credentials or a subject attribute name.
6. Enter the **Client ID** assigned to you when you created the OAuth client for validating OAuth access tokens.

   For more information about configuring a client ID in PingFederate, see *Configuring a Client*.
7. Optional: Select a **Client Credentials Type**, then enter the information for the selected credential type. Choose from:

   - **Secret** – Enter the **Client Secret** you were assigned when you created the PingAccess OAuth client in the token provider.
   - **Mutual TLS** – Select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – Select this option to use Private Key JSON web token (JWT). No additional information is required.
8. Enter the **Subject Attribute Name** you want to use from the OAuth access token as the subject for auditing purposes.

   At runtime, the attribute's value is used as the `Subject` field in audit log entries for the admin API.
9. Select the **Scope** required to successfully access the API. For more information about defining scopes in PingFederate, see *Authorization Server Settings*.

10. If you want to enable role-based authorization, perform the following steps:
    a. Click the **Roles** tab.
    b. To enable role-based authentication, select **Enable Roles**.
    c. In the **Administrator** section, click **Add Required Attribute** as many times as you need.

       For a role to be granted, all configured attribute values must match.
    d. Enter an **Attribute Name** and **Attribute Value** for each required attribute.

       Note:

       If you are using PingFederate as a token provider, the attribute name is defined in PingFederate under **OAuth Settings** # **OpenID Connect Policy Management** # *Your_Policy* # **Attribute Contact** as an extension to the contract. The value you use depends on the configuration of the **Contract Fulfillment** tab for the policy.

       Example: The attribute named `group` in your attribute contract can be mapped to an LDAP server attribute source that contains a `groupMembership` attribute. A valid group membership for the administrator might be the group `cn=pingaccess-admins,o=myorg`. In this example, you should use `group` as the **Attribute Name** and `cn=pingaccess-admins,o=myorg` as the **Attribute Value**.
    e. Optional: If you want to add platform administrators, select **Enable Platform Administrator Role**, then enter an **Attribute Name** and **Attribute Value** for each required attribute. Click **Add Required Attribute** to add a new attribute.
    f. Optional: If you want to add auditors, select **Enable Auditor Role**, then enter an **Attribute Name** and **Attribute Value** for each required attribute. Click **Add Required Attribute** to add a new attribute.
11. To activate API authentication, click **Save**.

Configuring admin UI SSO authentication

Configure single sign-on (SSO) for the administrative user interface in PingAccess.

To enable SSO, you must configure sections of the OpenID Connect (OIDC) provider and PingAccess. Expand a section to view which configurations are required. You can configure the administrative SSO option to require a specific authentication mechanism, leveraging the OIDC token provider Requested AuthN Context Selector using the PingAccess *authentication requirements* options.

Preparing to configure admin UI SSO authentication

About this task

Before you can configure admin UI SSO authentication, you must:

Steps

1. Configure the OIDC provider:
   Choose from:
   - *Configure PingFederate runtime*.
   - *Configure PingOne*.
   - *Configure OpenID connect*.
2. Import the OIDC token provider server certificate into a trusted certificate group and associate that trusted certificate group with the OIDC token provider runtime.

   For more information, see *Importing certificates* on page 309.

3. If you are using PingFederate, set up a **profile** scope in PingFederate that includes the openid, profile, address, email, and phone scope values.

   For more information, see the PingFederate documentation for *configuring an OAuth client*.

   a. If you're using PingFederate as the OIDC token provider, when you configure the client in PingFederate:

      - The **Client Authentication** must be set to anything but None.
      - The **Allowed Grant Types** must be set to Authorization Code.
      - The **Redirect URIs** must include https://*<PA_Admin_Host>*:*<PA_Admin_Port>*/*<reserved application context root>*/oidc/cb. The default reserved application context root is /pa.
      - If you're not using administrative roles in PingAccess, the OIDC **Policy** should be set to a policy that uses issuance criteria to restrict access based on some additional criteria.

      > **Warning:**
      >
      > If the selected OIDC policy does not use issuance criteria to limit which users can be granted an access token, all users in the associated identity store configured in PingFederate can authenticate to the PingAccess Admin console and make changes.
      >
      > For more information, see *Identifying Issuance Criteria for Policy Mapping* in the *PingFederate Administrator's Manual*.

   b. If you're using PingFederate as the OIDC token provider and plan to use **Mutual TLS**, you must make two changes to the PingFederate configuration:

      - Enable the use of the secondary HTTPS port in PingFederate by editing the *<PF_HOME>*/pingfederate/bin/run.properties file and setting the pf.secondary.https.port value to a port value. For more information, see *the PingFederate documentation*.
      - Modify the openid-configuration.template.json to add the mtls_endpoint_aliases object, with content defined by *RFC-8705*. For more information about this file, see the *PingFederate documentation*.

Configuring admin UI SSO authentication

Before you begin

Complete the configuration for connecting to the PingFederate OAuth authorization server on the *PingFederate for PingAccess SSO configuration* on page 350 page.

About this task

You can configure roles for UI users. Each role grants access to specific features:

- The Administrator role has full access to the UI, unless the Platform Administrator role is enabled. If the Platform Administrator role is enabled, the Administrator can't update authorization, user, or environment settings, but can use all other features.
- The Platform Administrator role has full access to all features. This role can be used with the Administrator role to grant full access to most features without the possibility of accidental lockout, with only the Platform Administrator able to change authorization configurations.
- The Auditor role can view the user interface but can't change the configuration.

To configure admin UI SSO:

Steps

1. Click **Settings** and then go to **Admin Authentication** # **UI Authentication**.

2. On the **Authentication Method** page, click **Single Sign-On**.

> **Tip:**
>
> To define a fallback administrator authentication method if the OIDC token provider is unreachable, enable the `admin.auth=native` property in the `run.properties` file. This overrides any configured administrative authentication to *basic authentication*.

3. In the **OpenID Connect Login Type** list, select a sign-on type:
   Choose from:
   - **Code** (default): The standard OIDC sign-on flow.
   - **POST**: A sign-on flow using the form_post response mode, which returns response parameters as `application/x-www-form-urlencoded` HTML form values.
   - **x_post**: A sign-on flow based on OIDC that passes claims from the provider through the browser using the implicit grant type.
4. In the **Client ID** field, enter the unique identifier assigned when you created the PingAccess OAuth client within your OIDC token provider.
5. Select a **Client Credentials Type**, then provide the information required for the selected credential type.

   This is required when configuring the **Code** sign-on type or if you enabled session validation.

   Choose from:
   - Click **Secret** to use a client secret. In the **Client Secret** field, enter the client secret assigned when you created the OAuth relying party client in the token provider.
   - Click **Mutual TLS** to use Mutual TLS client authentication. In the **Key Pair** list, select a configured key pair to use for Mutual TLS client authentication.
   - Click **Private Key JWT** to use Private Key JSON web token (JWT). No additional information is needed.

> **Info:**
>
> The OAuth client you use with PingAccess web sessions must have an OIDC policy specified. For more information, see *Configuring OpenID Connect Policies*.

6. Optional: If your environment requires an authentication requirements list, in the **Authentication Requirements** list, select a defined authentication requirements list or click **Create** to create a new list.
7. Optional: In the **Username Attribute Name** field, enter the attribute from the ID token to be used as the display name in the user interface and included in the audit logs.

   If the attribute isn't specified or can't be found, the `sub` attribute is used.
8. Optional: If you want to enable advanced settings, click **Show Advanced** and use one or more of the advanced options.

Advanced Option	Description
Scopes	Configure your token provider to handle all of the requested scopes you specify, including any custom scope values.

<table>
<tr><th>Advanced Option</th><th>Description</th></tr>
<tr><td></td><td>

- To request one or more scopes from the OIDC token provider, in the **Scopes** list, select one or more scopes.

  Note:

  If you configured a token provider, published scopes are available to select in the list based on the selected Client ID.

- To specify unverified scopes, enter the scope and click **Use unverified scope "[scopename]"**.

Note:

The user can access all attributes by examining browser traces. Although they're integrity-protected to prevent changes, you can view any sensitive or confidential attributes should the user decode the ID Token's value.
</td></tr>
<tr><td>Validate Session</td><td>

To validate sessions with the configured PingFederate instance during request processing, in the **Validate Session** options, click **Yes**.

Note:

This option is not supported by PingOne or third-party OIDC token providers.
</td></tr>
<tr><td>Refresh User Attributes</td><td>

a. To periodically refresh user data from the OIDC token provider, in the **Refresh User Attributes** options, click **Yes**.
b. Specify a **Refresh User Attributes Interval** in seconds.
</td></tr>
<tr><td>Cache User Attributes</td><td>

To have PingAccess cache user attribute information for use in policy decisions, select the **Cache User Attributes** check box.

Note:

When this option is disabled, user attribute information is encoded and stored in the session cookie.
</td></tr>
<tr><td>Enable PKCE</td><td>To have PingAccess send a SHA256 code challenge and corresponding code verifier as a</td></tr>
</table>

Advanced Option	Description
	Proof Key for Code Exchange during the code authentication flow, select the **Enable PKCE** check box. **Note:** The **OpenID Connect Login Type** must be set to **Code** for PingAccess to use PKCE.
Use Single-Logout	To enable the use of single logout (SLO), select the **Use Single-Logout** check box. **Important:** You must configure this option in the OIDC provider. **Tip:** If you're using PingFederate as a token provider, enable the **Check For Valid Authentication Session** in the PingFederate access token management configuration to prevent session replay.

9. Optional: To enable role-based authorization:
    a. Click the **Roles** tab.
    b. To enable role-based authentication, select the **Enable Roles** check box.
    c. In the **Administrator** section, click **Add Required Attribute** for each attribute you want to add.

        For a role to be granted, all configured attribute values must match.
    d. Enter an **Attribute Name** and **Attribute Value** for each required attribute.

        > **Note:**
        >
        > If you're using PingFederate as a token provider, the attribute name is defined in PingFederate under **OAuth Settings** # **OpenID Connect Policy Management** # *Your_Policy* # **Attribute Contact** as an extension to the contract.
        >
        > The value you use depends on the configuration of the **Contract Fulfillment** tab for the policy.

        The attribute named `group` in your attribute contract can be mapped to an LDAP server attribute source that contains a `groupMembership` attribute.

        A valid group membership for the administrator might be the group `cn=pingaccess-admins,o=myorg`. In this example, you should use `group` as the **Attribute Name** and `cn=pingaccess-admins,o=myorg` as the **Attribute Value**.
    e. Optional: To add platform administrators:
        1. Select the **Enable Platform Administrator Role** check box.
        2. Enter an **Attribute Name** and **Attribute Value** for each required attribute.
        3. Click **Add Required Attribute** to add a new attribute.
    f. Optional: To add auditors:
        1. Select the **Enable Auditor Role** check box.
        2. Enter an **Attribute Name** and **Attribute Value** for each required attribute.
        3. Click **Add Required Attribute** to add a new attribute.
10. Click **Save**.

Troubleshooting

If you mis-configure admin UI SSO and are locked out, see *Administrative SSO lockout* on page 544 for information about regaining access.

Configuring session properties

Configure session properties for the administrative user interface in PingAccess.

About this task

> **Note:**
>
> The current authentication setting is included in the menu title. For example, if basic authentication is configured as it is by default, the menu option is **Admin UI – Basic**.

Steps

1. Click **Settings** and then go to **Admin Authentication** # **UI Session Properties**.
2. Specify the **Cookie Type**, Encrypted JSON web token (JWT) or Signed JWT.
3. Specify the unique **Audience** name the token is applicable to.

4. Specify an **Idle Timeout** in minutes.

   This sets the length of time you want the PingAccess token to remain active when no activity is detected. When the idle expiration is reached, the session automatically terminates.
5. Specify a **Max Timeout** in minutes.

   This sets the length of time you want the PingAccess token to remain active. Once the PingAccess token expires, an authenticated user must re-authenticate.
6. Specify an **Expiration Warning** in minutes.

   This specifies the point at which a user will be warned of upcoming session expiry.
7. Specify the **Session Poll Interval** in seconds.

   This sets the length of time between user info poll requests for the admin UI.
8. Click **Save**.

Configuring an admin token provider

Configure a token provider to use when accessing the PingAccess user interface if you have enabled admin UI single sign-on or admin API OAuth.

About this task

If you do not configure an admin token provider, the system token provider is used for both the PingAccess user interface and for end users.

Steps

1. Click **Settings** and then go to **Admin Authentication** # **Admin Token Provider**.
2. In the **Admin Token Provider** section, select **Admin**.
3. In the **Issuer** field, enter the issuer ID.
4. Optional: In the **Description** field, enter a description for the token provider.
5. From the **Trusted Certificate Group** list, select a trusted certificate group that PingAccess will use when authenticating to the admin token provider.
6. Optional: To configure the connection to use a configured proxy, click **Show Advanced** and select **Use Proxy**.

   For more information about creating proxies, see *Adding proxies* on page 330.
7. Click **Save**.

System

The **System** tab contains controls for the administrative UI in PingAccess.

To learn more about the actions available in the System tab, choose from one of the following sections:

- *Configuration export/import* on page 339
- *License* on page 341
- *Token provider* on page 342

Configuration export/import

The configuration export/import options create and restore a full PingAccess configuration.

You can back up and restore your configuration for disaster recovery or for testing purposes. You can restore your configuration on the same system or a new system, as long as the version used on the restore system is not older than the version used on the backup system, and as long as the backup and restore systems both use API v3 (PingAccess 5.0 or later).

The configuration backup is stored as a `JSON` file, and contains the entire PingAccess configuration, with the exceptions of the administrative user configuration and the keys used for JSON web tokens (JWTs). It uses the same format as results from the administrative APIs.

> **CAUTION:**
>
> Because the exported `JSON` file contains much of your PingAccess configuration, make sure that the file is safely stored somewhere with appropriate security controls in place.

Sensitive data, such as secrets and passwords, are encrypted in the backup file. If you have *configured a master key encryptor* using the Add-on SDK for Java, the host key file (JWK set) is encrypted and included in the exported data. When running in Amazon Web Services, the AWS Key Management Services (KMS) master key encryptor bundled with PingAccess can be configured for master key encryption. See *Configuring PingAccess to use Amazon Key Management Services* on page 151 for more information.

You can modify the backup file directly. If you plan to do so, make an unmodified copy of the backup file before you begin.

Exporting PingAccess configurations

Export your current configuration for PingAccess.

About this task

Large PingAccess configurations can take upwards of thirty minutes to export. During an export, you cannot modify the PingAccess configuration.

Steps

1. Click **Settings** and then go to **System** # **Configuration Export/Import**.
2. Click **Export Configuration**.

   The downloaded file name is pa-data-*<timestamp>*.json.

   > **Note:**
   >
   > The *<timestamp>* value is formatted `MM-DD-YYYY.hh.mm.ss`. For example, a date and time of January 31, 2020 1:35 PM would be encoded as `01-31-2020.13.35.00` in the filename.

Importing PingAccess configurations

Import a previously saved configuration into your PingAccess instance.

About this task

The **Import Configuration** option is a version-specific tool that can import a previously exported configuration. PingAccess checks the exported `JSON` file to ensure that the file is not from a later version of PingAccess and is compatible with API v3 (PingAccess 5.0 or later).

Large PingAccess configurations can take several hours to import. During an import, you cannot modify or read the PingAccess configuration.

> **Note:**
>
> You can automatically import a configuration on a new system as part of the installation and startup process. For more information, see *Installing PingAccess on Linux* on page 55 or *Installing PingAccess on Windows from the command line* on page 58.

Steps

1. Click **Settings** and then go to **System # Configuration Export/Import**.
2. Click **Import Configuration**.
3. Select the `JSON` export file that contains the configuration you want to import.
4. To start the import process, click **Import**.
5. When prompted for confirmation, click **Confirm**.

   Important:

   You might want to backup your system. This operation is destructive and overwrites your entire PingAccess configuration. Passwords in the system will revert to what they were when the backup was created. Unless you perform a backup prior to restoring a different configuration, the configuration prior to clicking **OK** will not be recoverable.

   Note:

   If the import fails, click **View failures from last import** to view all of the errors logged during the import.
6. If the Agent or Admin listener key pairs change as a result of the import operation, restart PingAccess.
7. If the environment is clustered, ensure that the engines are using the proper engine keys. If they are not, re-save the engine to generate a new public key, and reconfigure the engine to use the newly generated key.

License

View the details of your PingAccess license or upload a new license.

Go to **System # License** to display the details of the current license.

You can upload a new license file using this page. The new license is compared to the existing license, and the UI displays a warning ribbon on the page in certain cases, such as if the uploaded license will expire sooner than the current license. After reviewing any warnings, you can replace the existing license file by importing a new one.

In a clustered environment, the license file on the administrative node is replicated to all of the engine nodes and the replica administrative node. The engine nodes do not require a license to function, but some default templates appear differently depending on the information in the license.

When a license is about to expire or has expired, the UI displays a warning, and a WARNING-level message is added to the PingAccess console log.

Note:

If the installation has a running configuration, and the administrator shuts down the server, removes the license file from the file system, and restarts the server, the existing runtime configuration will continue to work. However, the administrator will have to install a new license file on the file system, or upload one through the UI, to access and apply changes through the UI.

Uploading PingAccess licenses

Upload new license files in PingAccess.

About this task

When you select a new license file to import, PingAccess compares the new license file's attributes with the current license attributes before installing it. The UI displays a warning ribbon on the page in certain cases, such as if the uploaded license will expire sooner than the current license.

If the new license is acceptable, you can commit it and successfully replace the old license.

Steps

1. Click **Settings** and then go to **System** # **License**.
2. Click **Import License**.
3. Click **Choose a File** to select a license file.

> **Warning:**
>
> The UI will display a warning ribbon for the following cases:
>
> - Expiration date:
>
>   The new license is set to expire on a date earlier than that of your current license.
> - Expired:
>
>   The new license is already expired.
> - License version:
>
>   The major version of the license doesn't match the current version of PingAccess.
> - Max Applications:
>
>   The new license is limited to support fewer applications than your current license.

4. To import the selected license, click **Import**.

> **Note:**
>
> If you select the wrong license, you can alter your selection either by clicking **Remove** to remove the selected license from the **Import License** section of the page, or by clicking **Choose File** to select a different license file.

5. To install the selected license, click **Confirm**.

Token provider

You can configure the token provider for PingAccess. The supported token providers are PingFederate, PingOne, and common providers using the OpenID Connect (OIDC) protocol.

To configure PingFederate as the token provider for PingAccess:

- *Configuring a standard PingFederate runtime* on page 343
- *Configuring PingFederate administration* on page 348
- *Configuring OAuth resource servers* on page 349
- *PingFederate for PingAccess SSO configuration* on page 350

To configure PingOne as the token provider for PingAccess:

- *Configuring PingOne* on page 352

To configure the Common Token Provider for PingAccess:

- *Configuring OpenID Connect* on page 352
- *Configuring OAuth authorization servers* on page 354

PingFederate

Configure an existing PingFederate environment as the token provider for PingAccess.

- *Configuring a standard PingFederate runtime* on page 343
- *Configuring PingFederate administration* on page 348
- *Configuring OAuth resource servers* on page 349
- *PingFederate for PingAccess SSO configuration* on page 350

PingFederate runtime

Configure an existing PingFederate environment as the token provider for PingAccess. The procedure depends on whether your PingFederate instance is proxied by the PingAccess engines.

- *Configuring a standard PingFederate runtime* on page 343
- *Configuring a proxied PingFederate runtime* on page 346

Configuring a standard PingFederate runtime

Configure a secure connection to the PingFederate runtime in PingAccess.

Before you begin

Before configuring a secure connection to the PingFederate runtime, export the PingFederate certificate and import it into a trusted certificate group in PingAccess. Perform the following steps:

1. In PingFederate, export the certificate active for the runtime server. See *SSL Server Certificates* in the *PingFederate Administrator's Manual* for more information.
2. Import the certificate into PingAccess.
3. Create a Trusted Certificate Group if one does not already exist.
4. Add the certificate to a Trusted Certificate Group.

About this task

Note:

For information on configuring PingFederate as an OAuth authorization server, see *Enabling the OAuth AS* and *Authorization Server Settings* in the PingFederate documentation.

After you save the PingFederate runtime connection, PingAccess will test the connection to PingFederate. If the connection cannot be made, an error will display in the administrative interface, and the PingFederate runtime will not save.

The steps that display depend on your environment. In a new deployment, some of the PingFederate configuration information is imported automatically from the PingFederate well-known endpoint. If you upgrade from PingAccess 5.2 or earlier and have an existing token provider configuration, this information is provided manually. If you perform an upgrade and want to see the new version of this page, configure the token provider using the `/pingfederate/runtime` API endpoint. For more information, see *Administrative API Endpoints*.

Note:

Configuring PingFederate as a token provider using the `/pingfederate/runtime` overwrites the existing PingFederate configuration.

After you successfully configure the token provider, click **View Metadata** to display the metadata provided by the token provider. To update the metadata, click **Refresh Metadata**.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **Runtime**.
2. Select **Standard Token Provider**.
3. In the **Issuer** field, enter the PingFederate issuer name.
4. Optional: In the **Descriptions** field, enter a description for the PingFederate instance.
5. From the **Trusted Certificate Group** list, select the certificate group the PingFederate certificate is in.

   This list is available only if you select **Secure**.
6. To configure advanced settings, click **Show Advanced**.
   a. If hostname verification for secure connections is not required for either the runtime or the back channel servers, select the **Skip Hostname Verification** check box.
   b. To use a configured proxy for back channel requests, select the **Use Proxy** check box.

   > **Note:**
   >
   > If the node is not configured with a proxy, requests are made directly to PingFederate.
   >
   > See *Adding proxies* on page 330 for more information about creating proxies.

   c. Select **Use Single-Logout** to enable single logout (SLO) when the `/pa/oidc/logout/` endpoint is accessed to clear the cookie containing the PingAccess token.

   If you select this option, PingAccess sends a sign off request to PingFederate, which completes a full SLO flow.

   To use this feature, SLO must be configured on the OpenID Connect (OIDC) provider.
   d. Enter the **STS Token Exchange Endpoint** to be used for token mediation if it is different from the default value of `/pf/sts.wst`.
7. Click **Save**.

> **Note:**
> Saving a new PingFederate runtime configuration overwrites any existing PingFederate runtime configuration.

Result

After you save this configuration and *Configuring OAuth resource servers* on page 349, a PingFederate access validator is available for selection when you define OAuth-type rules in Policy Manager.<issuer>

Configure a standard PingFederate runtime (original workflow)

If your PingAccess deployment is upgraded from version 5.2 or earlier with an existing token provider configuration, and you have not configured a token provider using the `/pingfederate/runtime` API endpoint, use this workflow to configure a PingFederate runtime.

Before you begin

Before configuring a secure connection to the PingFederate runtime, export the PingFederate certificate and import it into a trusted certificate group in PingAccess. Perform the following steps:

1. In PingFederate, export the certificate active for the Runtime Server. For more information, see *SSL Server Certificates* in the *PingFederate Administrator's Manual*.
2. Import the certificate into PingAccess.
3. Create a Trusted Certificate Group if one does not already exist.

4. Add the certificate to a Trusted Certificate Group.

About this task

Info:

For information on setting up PingFederate as an OAuth authorization server, see *Enabling the OAuth AS* and *Authorization Server Settings*.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **Runtime**.
2. Select **Standard Token Provider**.
3. In the **Host** field, enter the PingFederate runtime host name or IP address for the PingFederate runtime.
4. In the **Port** field, enter the PingFederate runtime port number.
5. Optional: In the **Base Path** field, enter the base path, if needed, for the PingFederate runtime. The base path must start with a slash, such as `/federation`.
6. Select the **Audit Level** check box to log information about the transaction to the audit store.

   PingAccess audit logs record a selected subset of transaction log information at runtime and are located in the `/logs` directory of your PingAccess installation.
7. Select the **Secure** check box if PingFederate is expecting HTTPS connections.
8. From the **Trusted Certificate Group** list, select the certificate group the PingFederate certificate is in.

   This field is available only if you select **Secure** in the previous step.
9. To configure advanced settings, click **Show Advanced**.
   a. Click **Add Back Channel Server**.
   b. In the **Back Channel Servers** list, enter one or more *`<hostname:port>`* pairs.
   c. If the back channel uses HTTPS, enable the **Back Channel Secure** option.

      This option becomes available when at least one back channel server is defined.
   d. If the back channel uses an alternate base path, enter the path in the **Back Channel Base Path** field.
   e. If hostname verification for secure connections is not required for either the Runtime or the Back Channel Servers, enable the **Skip Hostname Verification** option.
   f. If hostname verification is required, enter the host name PingAccess should expect in the **Expected Hostname** field.
   g. To use a configured proxy for back channel requests, select the **Use Proxy** check box.

      Note:
      If the node is not configured with a proxy, requests are made directly to PingFederate. See *Adding proxies* on page 330 for more information about creating proxies.
   h. Select **Use Single-Logout** to enable single logout (SLO).

      To use this feature, SLO must be configured on the OpenID Connect (OIDC) provider.

10. Click **Save**.

> **Note:**
>
> After you save this configuration and *Configuring OAuth resource servers* on page 349, a PingFederate access validator is available for selection when you define OAuth-type rules in Policy Manager.

Next steps

After you save the PingFederate runtime connection, PingAccess will test the connection to PingFederate. If the connection cannot be made, a warning will display in the admin interface, and the PingFederate runtime will not save.

After you configure the token provider, click **View Metadata** to display the metadata provided by the token provider. To update the metadata, click **Refresh Metadata**.

Configuring a proxied PingFederate runtime

Configure a secure connection to the proxied PingFederate runtime in PingAccess.

Before you begin

Before configuring a secure connection to the PingFederate runtime, export the PingFederate certificate and import it into a trusted certificate group in PingAccess. Perform the following steps:

1. In PingFederate, export the certificate active for the runtime server. For more information, see *SSL Server Certificates* in the PingFederate documentation.
2. Import the certificate into PingAccess.
3. Create a Trusted Certificate Group if one does not already exist.
4. Add the certificate to a Trusted Certificate Group.

About this task

> **Note:**
>
> For information on configuring PingFederate as an OAuth authorization server, see *Enabling the OAuth AS* and *Authorization Server Settings* in the PingFederate documentation.

After you save the PingFederate runtime connection, PingAccess will test the connection to PingFederate. If the connection cannot be made, an error will display in the administrative interface, and the PingFederate runtime will not save.

After you successfully configure the token provider, click **View Metadata** to display the metadata provided by the token provider. To update the metadata, click **Refresh Metadata**.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **Runtime**.
2. Click **Proxied Token Provider (PingFederate Runtime Application)**.
3. In the **Primary Virtual Host** field, enter the virtual host to use for the PingFederate application.

   This virtual host is used by default for front channel redirects to the PingFederate token provider when an application-specific OpenID Connect Issuer is not defined.

   a. If you have not created the virtual host, click **+ Create**. For more information, see *Creating new virtual hosts* on page 245.

4. Optional: In the **Additional Virtual Hosts** field, enter one or more virtual hosts that can be used for the PingFederate application.
    a. If you have not created the virtual hosts, click **+ Create**. For more information, see *Creating new virtual hosts* on page 245.
5. In the **Targets** field, enter a `hostname:port` pair used to access the PingFederate runtime servers.

    Click **+ Add Target** to add additional **Targets** fields.
6. Optional: In the **Secure** section, click **Yes** if the PingFederate runtime expects HTTPS connections.
7. Optional: To configure advanced settings, click **Show Advanced**.

Option	Description
Context Root	Enter the first part of the URL path for the PingFederate application and its resources. The context root must begin with a slash. It can contain additional slashes, but cannot end with one. It must match the path defined by the base URL in PingFederate.
Case Sensitive	Select to make the context root and resource path matching case sensitive.
Client Certificate Header Name	In this section, click **+ Add Client Certificate Header Name** and enter one or more header names to which client certificates found in the request should be mapped. The position of the header name in the list correlates to the index in the client certificate chain, with the first header mapped to the leaf certificate.
Policy	In this section, add one or more rules, rule sets, or rule set groups to be run when making requests to the PingFederate runtime. Click **Rules**, **Rule Sets**, or **Rule Set Groups**, then drag one or more rules, rule sets, or rule set groups from the **Available** column to the **Selected Policy** column. The valid rule types are Groovy script, cross-origin request, and rewrite rules. Create new rules, rule sets, or rule set groups by clicking **+ Create Rule**, **+ Create Rule Set**, or **+ Create Rule Set Group**. For more information, see *Rule Management* on page 260, *Adding rule sets* on page 287, and *Adding rule set groups* on page 288.
Load Balancing Strategy	From this list, select a load balancing strategy to use for requests to the PingFederate runtime.
Expected Certificate Hostname	Enter the host name expected in the certificate. If this field is not specified, the certificates are verified using the target host names.

Option	Description
Skip Hostname Verification	Click to stop the back channel servers from performing host name verification of the certificate.
Use Proxy	Click to make back channel requests to PingFederate use the proxy configured on the PingAccess nodes.
Use Single-Logout	Click to enable single logout if it is configured for the OIDC provider.

8. Click **Save**.

> **Note:**
>
> Saving a new PingFederate runtime configuration overwrites any existing PingFederate runtime configuration.

Result

After you save this configuration and perform the steps in *Configuring OAuth resource servers* on page 349, a PingFederate access validator is available for selection when you define OAuth-type rules in Policy Manager.

Configuring PingFederate administration
Configure your PingFederate administration settings in PingAccess.

About this task

For information on the PingFederate Administration API, see *PingFederate Administrative API*.

When you save the PingFederate administration configuration, PingAccess will test the connection to PingFederate. If the connection cannot be made, an error will display in the administration console interface, and the configuration will not be saved.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **Administration**.
2. Enter the **Host** name or IP address for access to the PingFederate administrative API.
3. Enter the **Port** number for access to the PingFederate runtime.
4. If necessary, enter the **Base Path** for the PingFederate runtime.

   The **Base Path** must start with a slash (/).

   Example: `/path`.
5. Enter the **Admin Username**.

   This username only requires Auditor (read-only) permissions in PingFederate.
6. Enter the **Admin Password**.
7. To log information about the transaction to the audit store, select **Audit**.

   PingAccess audit logs record a selected subset of transaction log information at runtime and are located in the `/logs` directory of your PingAccess installation.
8. Enable **Secure** if PingFederate is expecting HTTPS connections.

9. From the **Trusted Certificate Group** list, select the group of certificates to use when authenticating to PingFederate.

   PingAccess requires the certificate in use by PingFederate anchor to a certificate in the associated Trusted Certificate Group. This field is available only if you enable **Secure**.
10. Optional: To configure advanced settings, click **Show Advanced**.
    a. Select **Skip Hostname Verification** to not perform hostname verification of the certificate.
    b. Enter an **Expected Certificate Hostname** to verify the certificate with the specified name instead of the **Host** name.
    c. To use a configured proxy for API requests, select the **Use Proxy** check box.

    > **Note:**
    >
    > If the node is not configured with a proxy, requests are made directly to PingFederate.
11. Click **Save**.

> **Tip:**
>
> To view OpenID Connect (OIDC) metadata provided by the token provider, click **View Metadata** after saving the token provider configuration.

Configuring OAuth resource servers

When receiving OAuth-protected API calls, PingAccess acts as an OAuth resource server, checking with the PingFederate OAuth authorization server on the validity of the bearer access token it receives from a client.

Before you begin

Prior to configuring this option, you must complete the PingFederate administration configuration.

If you plan to use **Mutual TLS**, you must make two changes to the PingFederate configuration.

- Enable the use of the secondary HTTPS port in PingFederate by editing the `<PF_HOME>/pingfederate/bin/run.properties` file and setting the `pf.secondary.https.port` value to a port value. For more information, see *the PingFederate documentation*.
- Modify the `openid-configuration.template.json` to add the `mtls_endpoint_aliases` object, with content defined by *RFC-8705*. For more information about this file, see the *PingFederate documentation*.

About this task

To validate the bearer access token, a valid OAuth client must exist within the PingFederate OAuth authorization server.

> **Note:**
>
> This configuration is optional and needed only if you plan to validate PingFederate OAuth access tokens.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **OAuth Resource Server**.

2. Enter the OAuth **Client ID** you defined when creating the PingAccess OAuth client in PingFederate.

> **Info:**
>
> When you configure an OAuth client in PingFederate, select *Access Token Validation* as the allowed grant type. For more information, see *Configuring a Client* in the *PingFederate Administrator's Manual*.

3. Select a **Client Credentials Type**, then provide the information required for the selected credential type.
   Choose from:
   - **Secret** – Enter the **Client Secret** assigned when you created the PingAccess OAuth client in PingFederate.
   - **Mutual TLS** – Select a configured **Key Pair** to use for Mutual TLS client authentication.
   - **Private Key JWT** – Select this option to use Private Key JSON web token (JWT). No additional information is required.
4. Select **Cache Tokens** to retain token details for subsequent requests.

   This option reduces the communication between PingAccess and PingFederate
5. If **Cache Tokens** is enabled, specify the **Token Time To Live** by entering the number of seconds to cache the access token. The default value of -1 means no limit. This value can be -1 or above and must be less than the PingFederate Token Lifetime.
6. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes, such as `username`.

   At runtime, the attribute's value is used as the Subject field in audit log entries for API Resources with policies that validate access tokens. The attribute must align with an attribute in the *OAuth access token attribute contract* defined within PingFederate.
7. If multiple Access Token Managers are configured in PingFederate, select the **Send Audience** option to send the URI the user requested as the `aud` OAuth parameter to select an Access Token Manager.

> **Note:**
>
> Use of this option requires that the Access Token Management instances be configured with appropriate Resource URIs. Matching of the Resource URI is performed on a most-specific match basis.

8. Optional: To disable the use of OAuth 2.0 token introspection, clear the **Use Token Introspection Endpoint** option.
9. To save your changes, click **Save**.

PingFederate for PingAccess SSO configuration
Configure PingFederate to enable administrator single sign-on (SSO) for PingAccess.

To enable administrator SSO to PingAccess, configure the following settings within the PingFederate authorization server. Click the icon ( ) next to each section heading to access additional configuration information. For example, click next to **Roles and Protocols** to open a new window and view the Choosing Roles and Protocols page of the PingFederate documentation.

> **Note:**
>
> The following information is an example configuration and does not cover all required steps for each PingFederate OAuth Settings page discussed, only fields necessary for successful SSO to the PingAccess administrative console. Fields not mentioned are not necessary for this configuration. For configuration details of the PingFederate OAuth settings pages, see *Using OAuth Menu Selections*.

> **Note:**
>
> You must complete the configuration for connecting to the PingFederate OAuth authorization server instance you plan to use. For more information, see *Configuring PingFederate administration* on page 348.

Roles and Protocols

- Enable the OAuth 2.0 AS role and the OpenID Connect (OIDC) protocol.
- Enable the identity provider (IdP) Provider role and a protocol.

Password Credential Validator (PCV)

- Create a PCV for authenticating administrative users.

Adapters

- Create an HTML Form IdP Adapter and specify the PCV you configured.

Authorization Server Settings

- Select **Implicit** in the Reuse Existing Persistent Access Grants for Grant Types section.

Access Token Management

- Select **Internally Managed Reference Tokens** as the Access Token Management Type.
- Extend the contract by adding the Username attribute on the Access Token Attribute Contract page.

OpenID Connect Policy Management

> **Info:**
>
> Create an OIDC Policy to use specifically for PingAccess administrative console authentication.

- Delete all of the attributes that appear in the Extend the Contract section of the Attribute Contract page. The only required attribute is **sub**.
- Select **Access Token** as the Source and **Username** as the Value on the Contract Fulfillment page.

Client Management

> **Info:**
>
> Create a Client to use specifically for PingAccess administrative console authentication.

- Select an option other than **None** for Client Authentication.
- Add the location of the PingAccess host as a Redirect URI. For example, `https://<PA_Admin_Host>:<PA_Admin_Port>/<reserved application context root>/oidc/cb`.
- Select **Authorization Code** as an Allowed Grant Type.
- Select one of the elliptic curve (**ECDSA**) algorithms as the OIDC ID Token Signing Algorithm and select the OIDC Policy to use for PingAccess administrative console authentication.

IdP Adapter Mapping

- Map the HTML Form IdP Adapter Username value to the USER_KEY and the USER_NAME contract attributes for the persistent grant and the user's display name on the authorization page, respectively.

Access Token Mapping

- Map values into the token attribute contract by selecting **Persistent Grant** as the Source and **USER_KEY** as the value for the Username attribute. These are the attributes included or referenced in the access token.

PingOne

You can configure PingOne as the token provider for PingAccess.

- *Configuring PingOne* on page 352

Configuring PingOne

Configure PingOne as the token provider in PingAccess.

Before you begin

You must have the PingOne issuer ID, or have access to the PingOne console, to perform this procedure.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **PingOne**.
2. In the **Issuer** field, enter the PingOne issuer ID. This information is available in the PingOne console.
3. Optional: In the **Description** field, enter a description for the connection.
4. In the **Trusted Certificate Group** list, select a trusted certificate group that PingAccess will use when authenticating to PingOne.
5. To configure the connection to use a configured proxy, click **Show Advanced** and select **Use Proxy**. See *Adding proxies* on page 330 for more information about creating proxies.
6. Click **Save**.

Next steps

After you configure the token provider, click **View Metadata** to display the metadata provided by the token provider. To update the metadata, click **View Metadata** # **Refresh Metadata**.

Common token provider

You can configure a common token provider that uses the OpenID Connect (OIDC) protocol as the token provider for PingAccess.

- *Configuring OpenID Connect* on page 352
- *Configuring OAuth authorization servers* on page 354

Configuring OpenID Connect

Configure OpenID Connect (OIDC) token provider settings in PingAccess.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **Common** # **OpenID Connect**.
2. In the **Issuer** field, enter the OIDC provider's issuer identifier.
3. In the **Description** field, enter a description for the token provider.
4. Select the **Audit** check box to record requests to OIDC provider to the audit store.
5. From the **Trusted Certificate Group** list, select the group of certificates to use when authenticating to OIDC provider.

   PingAccess requires that the certificate in use by OIDC provider anchor to a certificate in the associated Trusted Certificate Group.
6. If required, click **+ Add Query Parameter** and enter custom query parameter name and value pairs used by the OIDC provider.

7. To configure advanced settings, click **Show Advanced**.
    a. To use a configured proxy, select the **Use Proxy** check box.

> **Note:**
>
> If the node is not configured with a proxy, requests are made directly to the token provider.
>
> the node is not configured with a proxy, requests are made directly to the token provider. See *Adding proxies* on page 330 for more information about creating proxies.

    b. Select **Use Single-Logout** to enable single logout (SLO) when the `/pa/oidc/logout/` endpoint is accessed to clear the cookie containing the PingAccess token. If this option is selected, PingAccess sends a logout request to the token provider, which completes a full SLO flow.

    To use this feature, SLO must be configured on the OIDC provider.
    c. Select **Request Supported Scopes Only** to limit the requested scopes to those advertised in the OIDC metadata.
8. Click **Save**.

Next steps

Once you have successfully configured the token provider, click **View Metadata** to display the metadata provided by the token provider. To update the metadata, click **View Metadata** # **Refresh Metadata**.

Creating Azure AD Graph API applications

To use the Azure AD Graph API, an application must exist to provide an application ID and key that PingAccess will use as the client ID and client secret for communication with the Graph API.

About this task

Create the application in Azure AD through the **App Registrations** blade using these criteria:

Name

Enter a unique name for the application, such as "Graph API app"

Application Type

Web app / API

Sign-on URL

This field is not relevant for this particular use case, but is required by Azure AD. Enter the PingAccess host.

Steps

1. After you create the application, navigate to the application in the list.
2. Select **Required permissions** and click **Add**.
3. Choose **Windows Azure Active Directory**, and then click **Save**.

    For **Application Permissions**, read the directory data.
4. Copy the **Application ID**.
5. Generate and copy a **Key**.

Configuring token provider-specific options

Configure plugins that perform particular functions for the selected token provider type.

Before you begin

In order to configure these options, you must first perform the steps detailed in *Creating Azure AD Graph API applications* on page 353.

About this task

In the case of the PingAccess for Azure AD solution, the plugin addresses the following problems:

- Data Transformation— The format of data returned from the OpenID Connect (OIDC) UserInfo endpoint results in some unexpected JSON formatting. This data transforms into a format that PingAccess can easily process.
- Azure AD Graph API usage— If the **groups** attribute contains more than 200 groups, the id_token contains a level of indirection that points to a URL in the Azure AD Graph API. Through the creation of a simple purpose-driven application, you can communicate with the Azure ID Graph API to retrieve the complete list of groups.
- Retrieving group display names— The **groups** attribute is a list of GUIDs. The groups for a user are only provided as GUIDs since user-friendly names for Azure AD groups are not globally unique. Configure the Graph API call to include the group names along with the GUID for creation of more robust policies.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **Common** # **OpenID Connect**.
2. Go to **Token Provider Specific Options** section.
3. From the **Type** list, select **Azure Active Directory**.
4. To extend the attributes for a web session, select the **Use Azure AD Graph API** check box.
5. In the **Client ID** field, enter the application ID you copied from the Azure AD API application you created.
6. In the **Client Secret** field, paste the key you copied. Select **Retrieve Group Display Names**.

> **Important:**
>
> To retrieve group data for a particular application in the token, the manifest for that application must be modified to include a group membership claim. In the **App Registrations** blade, select the application and click the **Manifest** button. Locate the `groupMembershipClaims` API, select the following permission, and enter and specify a group type, such as `SecurityGroup`.

7. Select **Cache Group Display Names** to instruct PingAccess to cache display names retrieved from the Azure AD Graph API.
8. In the **Display Name Cache Max Age (s)** field, enter the number of seconds to cache group display names if caching is enabled. Click **Save**.

Configuring OAuth authorization servers
Configure, modify, and edit the OAuth authorization servers in PingAccess.

Before you begin

If you plan to use **Mutual TLS**, modify the token provider to provide the `mtls_endpoint_aliases` object, with content defined by *RFC-8705*, on the OpenID Connect (OIDC) well-known configuration endpoint.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **Common** # **OAuth Authorization Server**.
2. Optional: In the **Description** field, enter a description for the authorization server.
3. In the **Targets** field, enter one or more `hostname:port` pairs for the OAuth authorization server. Click **+ Add Target** to add additional targets.
4. In the **Introspection Endpoints** field, specify the OAuth endpoint through which the token introspection operation is accomplished.
5. Select the **Audit** check box to record requests to the OAuth authorization server to the audit store.
6. Select the **Secure** option if the OAuth authorization server is expecting HTTPS connections.
7. From the **Trusted Certificate Group** list, select the group of certificates to use when authenticating to the OAuth authorization server.

   PingAccess requires that the certificate in use by OAuth authorization server anchors to a certificate in the associated trusted certificate group.
8. In the **Client ID** field, enter the unique identifier assigned when you created the PingAccess OAuth client within your OAuth authorization server.
9. Select a **Client Credentials Type**, then provide the information required for the selected credential type.
   Choose from:

   - **Secret** – Enter the **Client Secret** assigned when you created the PingAccess OAuth client in the token provider.
   - **Mutual TLS** – Select a configured **Key Pair** to use for mutual TLS client authentication.
   - **Private Key JWT** – Select this option to use Private Key JSON web token (JWT). No additional information is required.
10. Optional: Select the **Cache Tokens** option to retain token details for subsequent requests.

    This option reduces the communication between PingAccess and OAuth authorization server.
11. Optional: Select the **Token Time To Live** check box to enter the number of seconds to cache the access token.

    A value of -1 means there is no limit. This value should be less than the OAuth authorization server token lifetime.
12. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes.

    At runtime, the attribute's value is used as the Subject field in audit log entries for API Resources with policies that validate access tokens.
13. Select the **Send Audience** check box to send the URI the user requested as the `aud` OAuth parameter for PingAccess to the OAuth 2.0 authorization server.
14. To configure advanced settings, click **Show Advanced**.
    a. To use a configured proxy, select the **Use Proxy** check box.
15. Click **Save**.

> **Note:**
>
> If the node is not configured with a proxy, requests are made directly to the token provider.

Environment

View and update the details of your PingAccess environment.

You can manage information about your PingAccess environment, such as the environment name, displayed in the upper right section of the user interface.

Changing the Environment Name

Change the environment name displayed in the PingAccess user interface.

About this task

The environment name is displayed in the upper right header bar. If you manage multiple instances of PingAccess, or more than one Ping product, use this environment name to delineate these environments.

Steps

1. Click **Settings** and then go to **System** # **Environment**.
2. In the **Environment Name** field, enter an environment name.
3. Click **Save**.

   Result: The new environment name is displayed in the header bar.

Agents and Integrations

PingAccess is supported by agents that can be installed in an agent deployment, and integrations that help PingAccess work with other tools.

Agents

In an agent deployment, agents are installed on each web server. You can use an existing agent or create your own using the SDKs.

- *PingAccess Agent for Apache (RHEL)* on page 357
- *PingAccess Agent for Apache (SLES)* on page 380
- *PingAccess Agent for Apache (Windows)* on page 393
- *PingAccess Agent for IIS* on page 403
- *PingAccess Agent for NGINX* on page 422
- *PingAccess Agent Protocol* on page 434
- *PingAccess Agent SDK for C* on page 449
- *PingAccess Agent SDK for Java* on page 453

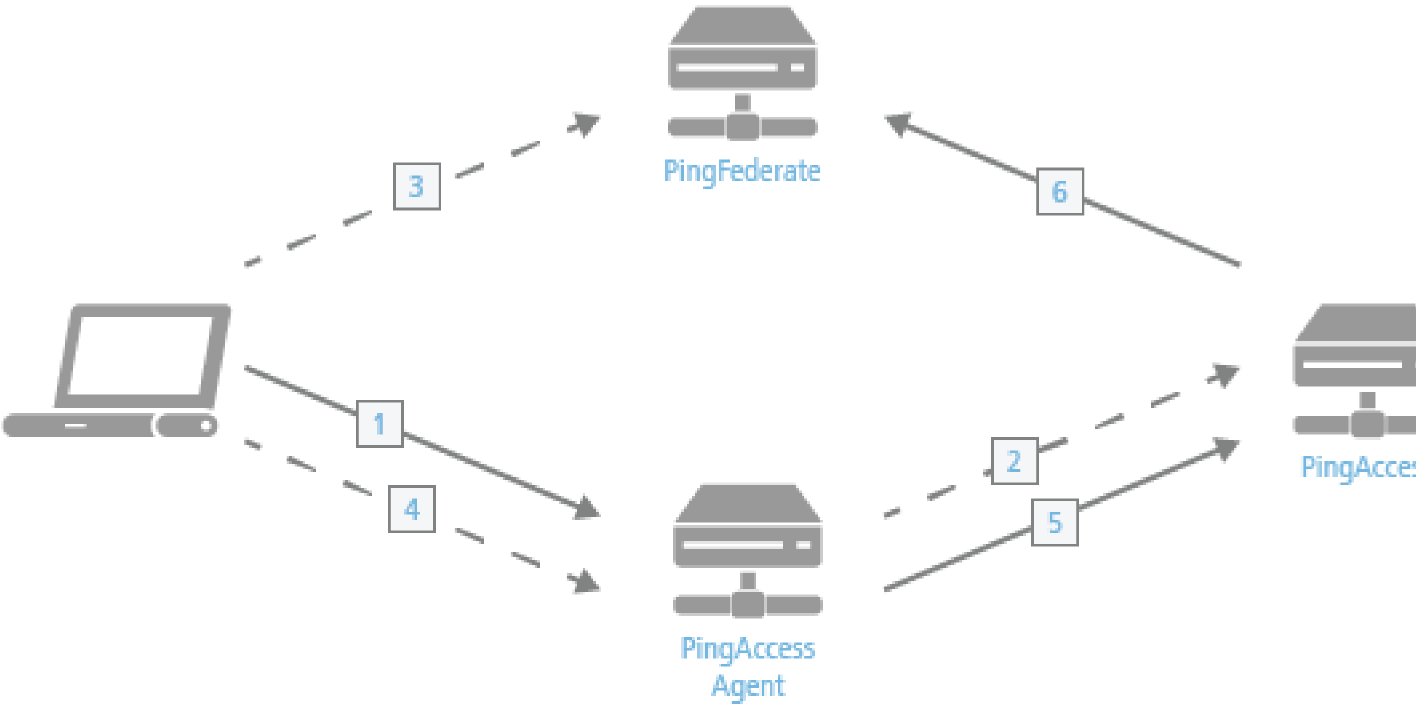
Integrations

You can create integrations for PingAccess using the Add-on SDK, or use existing integrations with products used in your environment.

- *PingAccess Add-on SDK for Java* on page 458
- *iovation FraudForce Integration* on page 501

PingAccess Agent for Apache (RHEL)

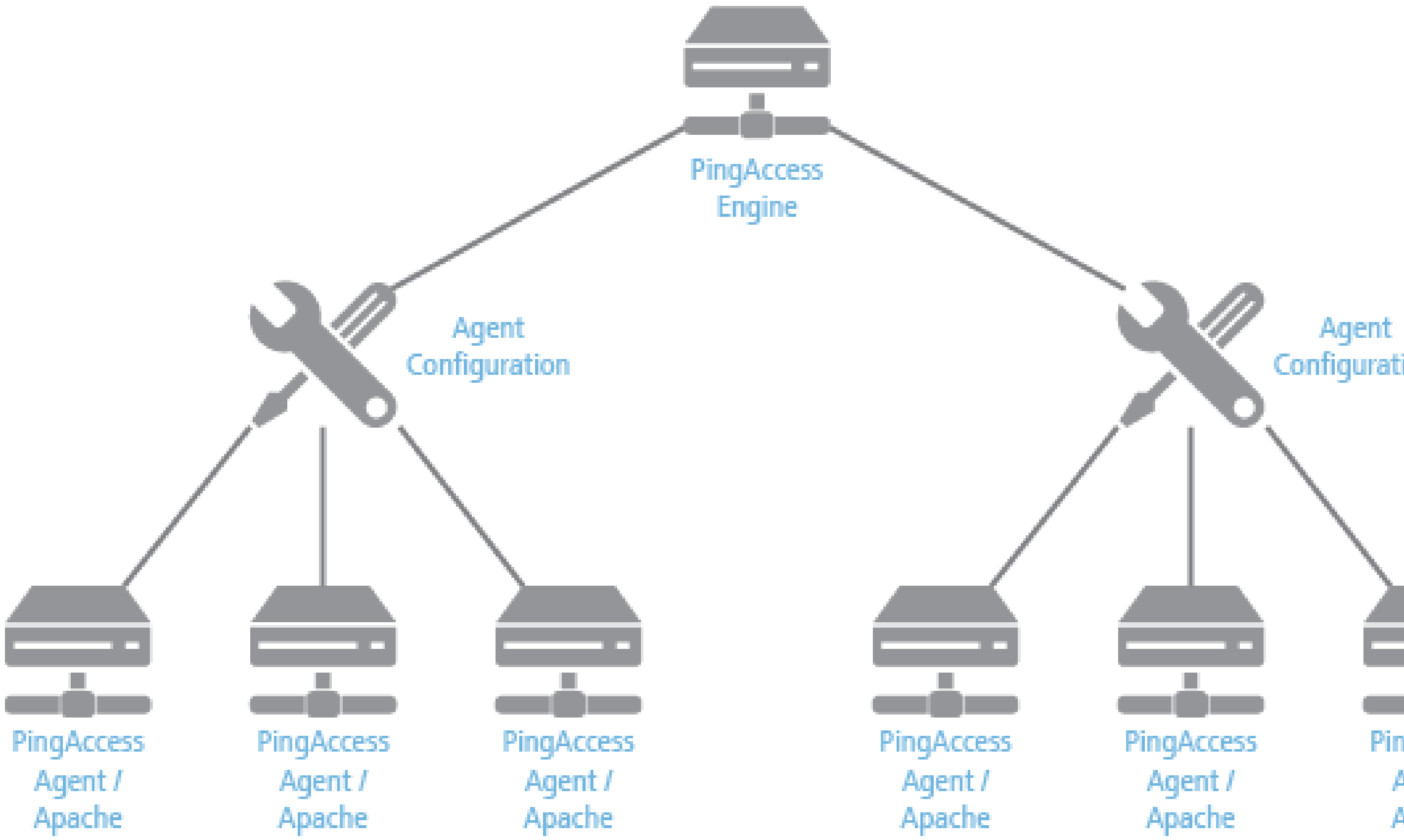
The PingAccess Agent for Apache is an Apache module that intercepts requests to the web server's protected resources and evaluates applicable access control policies. These policies are evaluated by either accessing a locally cached policy decision or by querying the PingAccess engine node.



The process used when a PingAccess agent is added to the policy decision process is as follows:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

Within the PingAccess administrative console, agent nodes are configured with information that allows a PingAccess agent to connect to the engine node to retrieve information about access control policies for resources within that agent's control. An agent configuration has a one-to-many relationship with PingAccess agents, allowing a single agent configuration bootstrap file to be used on multiple web servers within a server farm.



Tip:

An agent node is a shared configuration used by one or more agents, rather than a specific agent instance.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Installing on RHEL

Install or uninstall a PingAccess agent on a RHEL system.

Before installing the agent, review:

- *RHEL agent system requirements* on page 359.

To install the PingAccess agent for Apache:

- For information on RHEL 7, see *Installing on RHEL 7* on page 359.
- For information on RHEL 8, see *Installing on RHEL 8* on page 361.
- For information on RHEL 9, see *Installing on RHEL 9* on page 363.
- For information on IBM HTTP Server 9.0, see *Installing on an IBM HTTP Server using Apache 2.4* on page 364.
- For information on IBM HTTP Server 8.5.5, see *Installing on an IBM HTTP Server using Apache 2.2* on page 368.

To uninstall the agent:

- For information, see *Uninstalling the RHEL agent* on page 371.

RHEL agent system requirements

The PingAccess agent for Apache is supported on the following platforms:

- Apache HTTP Server 2.4 running on Red Hat Enterprise Linux Server 7 (x86_64)
- Apache HTTP Server 2.4 running on Red Hat Enterprise Linux Server 8 (x86_64)
- Apache HTTP Server 2.4 running on Red Hat Enterprise Linux Server 9 (x86_64)
- IBM HTTP Server 8.5.5 running on Red Hat Enterprise Linux Server 7 (x86_64)
- IBM HTTP Server 8.5.5 running on Red Hat Enterprise Linux Server 8 (x86_64)
- IBM HTTP Server 9.0 running on Red Hat Enterprise Linux Server 7 (x86_64)
- IBM HTTP Server 9.0 running on Red Hat Enterprise Linux Server 8 (x86_64)
- IBM HTTP Server 9.0 running on Red Hat Enterprise Linux Server 9 (x86_64)

As with any system that is reachable from the Internet, the server should be properly hardened. The PingAccess agent for Apache includes an SELinux profile. You should deploy SELinux on the server.

Installing on RHEL 7

Install a PingAccess agent on a RHEL 7 system with Apache 2.4, either conventionally or manually.

Before you begin

If you haven't downloaded an `agent.properties` file, you must first:

1. In the PingAccess console, go to **Applications** # **Agents**.
2. Click the **Pencil** icon to edit a configured agent.

   If you haven't created an agent yet, see *Agents* on page 254.
3. In the **Shared Secrets** section, click the **Download** icon to download the configuration.

> **Remember:**
>
> The configuration file will be named `<agentname>_agent.properties`.

About this task

Click the respective tab for the installation that you want to use.

Installing on RHEL 7 conventionally

Before you begin

Download and extract the `pingaccess-agent-apache24-rhel7-<version>.zip` archive.

> **Note:**
>
> The agent RPM has required dependencies that might be available through standard repositories. If these dependencies are not available in your Linux version, you can install them using the included `libpgm-5_2-0-5.2.122-32.1.x86_64.rpm`, `libsodium18-1.0.11-1.1.x86_64.rpm`, and `libzmq5-4.3.1-23.6.x86_64.rpm` packages.
>
> You can install these RPMs using the following command:
>
> ```
> yum install libsodium*rpm libpgm*rpm libzmq*rpm
> ```

About this task

To install a PingAccess agent on a RHEL 7 system with Apache 2.4:

Steps

1. In RHEL, change to the `pingaccess-agent-apache24-rhel7-<version>/x86_64` directory.
2. As root, install the PingAccess Agent for Apache using the following command:

```
yum install pingaccess-agent-apache-*.rpm
```

3. Copy the `<agentname>_agent.properties` file to `/etc/httpd/conf.d/agent.properties`.
4. As root, restart the Apache service using the following command:

```
systemctl restart httpd.service
```

Manually Installing on RHEL 7

About this task

This procedure assumes that:

- A non-root user is installing the PingAccess Agent for Apache in a custom Apache instance.
- You've installed the Apache installation at `$APACHE`. If you haven't, modify the file paths specified in this procedure based on where your Apache installation and configuration files are located.

To manually install the PingAccess Agent for Apache on a RHEL 7 system when Apache is installed in a non-standard way:

Steps

1. Install the following required dependencies from the RedHat Official Repositories:

```
libcurl.x86_64
pcre.x86_64
```

2. Copy the RPMs from the `.zip` archive into a directory called `pkgroot` and extract them using the following commands:

```
mkdir pkgroot
cp *.rpm pkgroot/
cd pkgroot
for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
```

3. Copy the extracted files to the appropriate places with the following commands:

```
cp etc/httpd/conf.modules.d/10-paa.conf $APACHE/conf
cp -av usr/lib64/*.so* $APACHE/modules
cp usr/lib64/httpd/modules/*.so $APACHE/modules
```

4. Add the following directive to the Apache configuration file, `$APACHE/conf/httpd.conf`, to include the PingAccess Agent for Apache module configuration:

```
Include conf/10-paa.conf
```

5. Edit the `10-paa.conf` file and make the following changes:
   a. Add the following lines before the `LoadModule` directive:

   ```
   LoadFile modules/libpgm-5.2.so.0
   LoadFile modules/libsodium.so.18
   LoadFile modules/libzmq.so.5
   ```

   b. Change all occurrences of `conf.d` to `conf`.

6. Copy your downloaded `<hostname>_agent.properties` to `$APACHE/conf/agent.properties`.
7. Run the **`$APACHE/bin/apachectl restart`** command to restart Apache.

Installing on RHEL 8

Install a PingAccess agent on a RHEL 8 system with Apache 2.4, either conventionally or manually.

Before you begin

If you haven't downloaded an `agent.properties` file, you must first:

1. In the PingAccess console, go to **Applications** # **Agents**.
2. Click the **Pencil** icon to edit a configured agent.

   If you haven't created an agent yet, see *Agents* on page 254.
3. In the **Shared Secrets** section, click the **Download** icon to download the configuration.

> **Remember:**
>
> The configuration file will be named `<agentname>_agent.properties`.

About this task

Click the respective tab for the installation that you want to use.

Installing on RHEL 8 conventionally

Before you begin

Download and extract the `pingaccess-agent-apache24-rhel8-<version>.zip` archive.

> **Note:**
>
> The Agent RPM has required dependencies that might be available through standard repositories. If these dependencies are not available in your Linux version, you can install them using the included `openpgm-5.2.122-21.el8.x86_64.rpm`, `libsodium-1.0.18-2.el8.x86_64.rpm`, `libunwind-1.3.1-3.el8.x86_64.rpm`, and `zeromq-4.3.2-1.el8.x86_64.rpm` packages.
>
> You can install these RPMs using the following command:
>
> ```
> yum install libsodium*rpm openpgm*rpm libunwind*rpm zeromq*rpm
> ```

About this task

To install a PingAccess agent on a RHEL 8 system with Apache 2.4:

Steps

1. In RHEL, change to the `pingaccess-agent-apache24-rhel8-<version>/x86_64` directory.
2. As root, install the PingAccess Agent for Apache using the following command:

   ```
   yum install pingaccess-agent-apache-*.rpm
   ```
3. Copy the `<agentname>_agent.properties` file to `/etc/httpd/conf.d/agent.properties`.

4. As root, restart the Apache service using the following command:

   ```
   systemctl restart httpd
   ```

Manually Installing on RHEL 8

About this task

This procedure assumes that:

- A non-root user is installing the PingAccess Agent for Apache in a custom Apache instance.
- You've installed the Apache installation at `$APACHE`. If you haven't, modify the file paths specified in this procedure based on where your Apache installation and configuration files are located.

To manually install the PingAccess Agent for Apache on a RHEL 8 system when Apache is installed in a non-standard way:

Steps

1. Install the following required dependencies from the RedHat Official Repositories:

   ```
   libcurl.x86_64
   pcre.x86_64
   ```

2. Copy the RPMs from the `.zip` archive into a directory called `pkgroot` and unpack them using the following commands:

   ```
   mkdir pkgroot
   cp *.rpm pkgroot/
   cd pkgroot
   for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
   ```

3. Copy the extracted files to the appropriate places with the following commands:

   ```
   cp etc/httpd/conf.modules.d/10-paa.conf $APACHE/conf
   cp -av usr/lib64/*.so* $APACHE/modules
   cp usr/lib64/httpd/modules/*.so $APACHE/modules
   ```

4. Add the following directive to the Apache configuration file (`$APACHE/conf/httpd.conf`) to include the PingAccess Agent for Apache module configuration:

   ```
   Include conf/10-paa.conf
   ```

5. Edit the `10-paa.conf` file and make the following changes:
   a. Add the following lines before the `LoadModule` directive:

      ```
      LoadFile modules/libpgm-5.2.so.0
      LoadFile modules/libsodium.so.18
      LoadFile modules/libzmq.so.5
      ```

   b. Change all occurrences of `conf.d` to `conf`.
6. Copy your downloaded `<hostname>_agent.properties` to `$APACHE/conf/agent.properties`.
7. Run the command **`$APACHE/bin/apachectl restart`** to restart Apache.

Installing on RHEL 9

Install a PingAccess agent on a RHEL 9 system with Apache 2.4, either conventionally or manually.

Before you begin

If you haven't downloaded an `agent.properties` file, you must first:

1. In the PingAccess console, go to **Applications** # **Agents**.
2. Click the **Pencil** icon to edit a configured agent.

   If you haven't created an agent yet, see *Agents* on page 254.
3. In the **Shared Secrets** section, click the **Download** icon to download the configuration.

> **Remember:**
>
> The configuration file will be named `<agentname>_agent.properties`.

About this task

Click the respective tab for the installation that you want to use.

Installing on RHEL 9 conventionally

Before you begin

Download and extract the `pingaccess-agent-apache24-rhel9-<version>.zip` archive.

> **Note:**
>
> The Agent RPM has required dependencies that might be available through standard repositories. If these dependencies aren't available in your Linux version, you can install them using the included `openpgm-5.2.122-28.el9.x86_64.rpm`, `libsodium-1.0.18-8.el9.x86_64.rpm`, `libunwind-1.6.2-1.el9.x86_64.rpm`, and `zeromq-4.3.4-2.el9.x86_64.rpm` packages.
>
> You can install these RPMs using **`yum install libsodium*rpm openpgm*rpm libunwind*rpm zeromq*rpm`**.

About this task

To install a PingAccess agent on a RHEL 9 system with Apache 2.4:

Steps

1. In RHEL, change to the `pingaccess-agent-apache24-rhel9-<version>/x86_64` directory.
2. As root, install the PingAccess Agent for Apache using the following command:

   ```
   yum install pingaccess-agent-apache-*.rpm
   ```
3. Copy the `<agentname>_agent.properties` file to `/etc/httpd/conf.d/agent.properties`.
4. As root, restart the Apache service using the following command:

   ```
   systemctl restart httpd
   ```

Manually Installing on RHEL 9

About this task

This procedure assumes that:

- A non-root user is installing the PingAccess Agent for Apache in a custom Apache instance.
- You've installed the Apache installation at `$APACHE`. If you haven't, modify the file paths specified in this procedure based on where your Apache installation and configuration files are located.

To manually install the PingAccess Agent for Apache on a RHEL 9 system when Apache is installed in a non-standard way:

Steps

1. Install the following required dependencies from the Red Hat official repositories:

   ```
   libcurl.x86_64
   pcre.x86_64
   ```
2. Copy the RPMs from the `.zip` distribution into a directory called `pkgroot` and unpack them using the following commands:

   ```
   mkdir pkgroot
   cp *.rpm pkgroot/
   cd pkgroot
   for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
   ```
3. Copy the extracted files to the appropriate places with the following commands:

   ```
   cp etc/httpd/conf.modules.d/10-paa.conf $APACHE/conf
   cp -av usr/lib64/*.so* $APACHE/modules
   cp usr/lib64/httpd/modules/*.so $APACHE/modules
   ```
4. Edit the `10-paa.conf` file and make the following changes:
   a. Add the following lines before the `LoadModule` directive:

      ```
      LoadFile modules/libunwind.so.8
      LoadFile modules/libpgm-5.2.so.0
      LoadFile modules/libsodium.so.23
      LoadFile modules/libzmq.so.5
      ```
   b. Change all occurrences of `conf.d` to `conf`.
5. Copy your downloaded *`<hostname>`*`_agent.properties` to `$APACHE/conf/agent.properties`.
6. Run the **`$APACHE/bin/apachectl restart`** command to restart Apache.

Installing on an IBM HTTP Server using Apache 2.4
Manually install a PingAccess agent on a RHEL system with Apache 2.4 when using an IBM HTTP Server.

Before you begin

If you haven't downloaded an `agent.properties` file:

1. In the PingAccess console, go to **Applications** # **Agents**.
2. Click the **Pencil** icon to edit a configured agent.

   If you haven't created an agent yet, see *Agents* on page 254.

3. In the **Shared Secrets** section, click the **Download** icon to download the configuration.

> **Remember:**
>
> The configuration file will be named `<agentname>_agent.properties`.

This procedure assumes that:

- You've installed and configured the IBM HTTP Server according to IBM's documentation.
- You've downloaded and extracted the version-appropriate `.zip` archive for your environment. For example:
  - `pingaccess-agent-apache24-rhel7*.zip`
  - `pingaccess-agent-apache24-rhel8*.zip`
  - `pingaccess-agent-apache24-rhel9*.zip`
- `apachectl` for the running IBM HTTP Server instance is in the file path.
- You've installed the Apache installation at `$IHS`. If you haven't, modify the file paths specified in this procedure based on where your Apache installation and configuration files are located.
- You've installed `libcurl` and PCRE or verified that they are installed. To install these packages, use the **`yum install libcurl pcre`** command.

About this task

Click the tab for the installation that you want to use.

Installing on a RHEL 7 system with an IBM HTTP Server

About this task

To manually install the PingAccess agent for Apache on RHEL 7 when using the IBM HTTP Server:

Steps

1. Go to the `pingaccess-agent-apache24-rhel7-<version>/<arch>/` directory.

   > **Note:**
   >
   > Currently, the only valid value for <*arch*> is `x86_64` for 64-bit.

   Example:

   `cd pingaccess-agent-apache24-rhel7-1.5.0/x86_64/`
2. Extract the package RPMs using the following command:

   ```
   mkdir pkgroot
   cp *.rpm pkgroot/
   cd pkgroot
   for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
   ```
3. Run the **`cp`** command to copy the libraries to the appropriate Apache directories.

   For RedHat Enterprise Linux 7 (x86_64):

   ```
   cp -av usr/lib64/*.so* $IHS/modules
   ```
4. Copy `mod_paa.so` into the Apache modules directory:

   ```
   cp -av ../mod_paa.so $IHS/modules
   ```

5. Copy the `10-paa.conf` file to the Apache configuration directory:

```
cp -av ../../10-paa.conf $IHS/conf
```

6. In the `10-paa.conf` file:
   a. Add the following lines before the `LoadModule` directive:

   ```
   LoadFile modules/libpgm-5.2.so.0
   LoadFile modules/libsodium.so.18
   LoadFile modules/libzmq.so.5
   ```

   b. Update the values for `PaaPropertyFiles` and `PaaCertificateDir` to point to your Apache `conf` directory.
7. In the Apache configuration file, `$IHS/conf/httpd.conf`, use the following directive to add the PingAccess agent for Apache's module configuration:

```
Include conf/10-paa.conf
```

8. Copy the `<agentname>_agent.properties` file to the `$IHS/conf/agent.properties` directory.

   This is the configuration file that you downloaded in step 3 of *Installing on an IBM HTTP Server using Apache 2.4* on page 364.
9. Restart the Apache service by running the **`apachectl restart`** command.

Installing on a RHEL 8 system with an IBM HTTP Server

About this task

To manually install the PingAccess agent for Apache on RHEL 8 when using the IBM HTTP Server:

Steps

1. Go to the `pingaccess-agent-apache24-rhel8-<version>/<arch>/` directory.

> **Note:**
>
> Currently, the only valid value for <*arch*> is `x86_64` for 64-bit.

   Example:

   `cd pingaccess-agent-apache24-rhel8-1.5.0/x86_64/`
2. Extract the package RPMs using the following command:

```
mkdir pkgroot
cp *.rpm pkgroot/
cd pkgroot
for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
```

3. Run the **`cp`** command to copy the libraries to the appropriate Apache directories.
   Example:

   For RedHat Enterprise Linux 8 (x86_64):

```
cp -av usr/lib64/*.so* $IHS/modules
```

4. Copy `mod_paa.so` into the Apache modules directory:

```
cp -av usr/lib64/httpd/modules/mod_paa.so $IHS/modules
```

5. Copy the `10-paa.conf` file to the Apache configuration directory:

```
cp -av etc/httpd/conf.modules.d/10-paa.conf $IHS/conf
```

6. In the `10-paa.conf` file:
   a. Add the following lines before the `LoadModule` directive:

   ```
   LoadFile modules/libpgm-5.2.so.0
   LoadFile modules/libzmq.so.5
   ```

   b. Update the values for `PaaPropertyFiles` and `PaaCertificateDir` to point to your Apache `conf` directory.
7. In the Apache configuration file, `$IHS/conf/httpd.conf`, use the following directive to add the PingAccess agent for Apache's module configuration:

```
Include conf/10-paa.conf
```

8. Copy the `<agentname>_agent.properties` file to the `$IHS/conf/agent.properties` directory.

   This is the configuration file that you downloaded in step 3 of *Installing on an IBM HTTP Server using Apache 2.4* on page 364.
9. Restart the Apache service by running the **`apachectl restart`** command.

Installing on a RHEL 9 system with an IBM HTTP Server

About this task

To manually install the PingAccess agent for Apache on RHEL 9 when using the IBM HTTP Server:

Steps

1. Go to the `pingaccess-agent-apache24-rhel9-<version>/<arch>/` directory.

   Note:

   Currently, the only valid value for <*arch*> is `x86_64` for 64-bit.

   Example:

   `cd pingaccess-agent-apache24-rhel9-1.5.0/x86_64/`
2. Extract the package RPMs using the following command:

```
mkdir pkgroot
cp *.rpm pkgroot/
cd pkgroot
for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
```

3. Run the **`cp`** command to copy the libraries to the appropriate Apache directories.
   Example:

   For RedHat Enterprise Linux 9 (x86_64):

```
cp -av usr/lib64/*.so* $IHS/modules
```

4. Copy `mod_paa.so` into the Apache modules directory:

```
cp -av usr/lib64/httpd/modules/mod_paa.so $IHS/modules
```

5. Copy the `10-paa.conf` file to the Apache configuration directory:

```
cp -av ../10-paa.conf $IHS/conf
```

6. In the `10-paa.conf` file:
   a. Add the following lines before the `LoadModule` directive:

   ```
   LoadFile modules/libpgm-5.2.so.0
   LoadFile modules/libzmq.so.5
   ```

   b. Update the values for `PaaPropertyFiles` and `PaaCertificateDir` to point to your Apache `conf` directory.
7. In the Apache configuration file, `$IHS/conf/httpd.conf`, use the following directive to add the PingAccess agent for Apache's module configuration:

```
Include conf/10-paa.conf
```

8. Copy the *<agentname>*_agent.properties file to the`$IHS/conf/agent.properties` directory.

   This is the configuration file that you downloaded in step 3 of *Installing on an IBM HTTP Server using Apache 2.4* on page 364.
9. Restart the Apache service by running the **apachectl restart** command.

Installing on an IBM HTTP Server using Apache 2.2

Manually install a PingAccess agent on a RHEL system with Apache 2.2 when using IBM HTTP Server 8.5.5.

Before you begin

If you haven't downloaded an `agent.properties` file:

1. In the PingAccess console, go to **Applications** # **Agents**.
2. Click the **Pencil** icon to edit a configured agent.

   If you haven't created an agent yet, see *Agents* on page 254.
3. In the **Shared Secrets** section, click the **Download** icon to download the configuration.

> **Remember:**
>
> The configuration file will be named *<agentname>*_agent.properties.

This procedure assumes that:

- You've installed and configured the IBM HTTP Server according to IBM's documentation.
- You've downloaded and extracted the version-appropriate `.zip` archive for your environment. For example:
  - `pingaccess-agent-apache22-rhel7*.zip`
  - `pingaccess-agent-apache22-rhel8*.zip`
- `apachectl` for the running IBM HTTP Server instance is in the file path.
- You've installed the Apache installation at `$IHS`. If you haven't, modify the file paths specified in this procedure based on where your Apache installation and configuration files are located.
- You've installed `libcurl` and PCRE or verified that they are installed. To install these packages, use the **yum install libcurl pcre** command.

About this task

Click the tab for the installation that you want to use.

Installing on a RHEL 7 system with an IBM HTTP Server

About this task

To manually install the PingAccess agent for Apache on RHEL 7 when using the IBM HTTP Server:

Steps

1. Go to the `pingaccess-agent-apache22-rhel7-<version>/<arch>/` directory.

   Note:

   Currently, the only valid value for <*arch*> is `x86_64` for 64-bit.

   Example:

   `cd pingaccess-agent-apache22-rhel7-1.5.0/x86_64/`
2. Extract the package RPMs using the following command:

   ```
   mkdir pkgroot
   cp *.rpm pkgroot/
   cd pkgroot
   for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
   ```
3. Run the **`cp`** command to copy the libraries to the appropriate Apache directories.

   For RedHat Enterprise Linux 7 (x86_64):

   ```
   cp -av usr/lib64/*.so* $IHS/modules
   ```
4. Copy `mod_paa.so` into the Apache modules directory:

   ```
   cp -av ../mod_paa.so $IHS/modules
   ```
5. Copy the `paa.conf` file to the Apache configuration directory:

   ```
   cp -av ../../paa.conf $IHS/conf
   ```
6. In the `paa.conf` file:
   a. Add the following lines before the `LoadModule` directive:

      ```
      LoadFile modules/libpgm-5.2.so.0
      LoadFile modules/libsodium.so.18
      LoadFile modules/libzmq.so.5
      ```
   b. Update the values for `PaaPropertyFiles` and `PaaCertificateDir` to point to your Apache `conf` directory.
7. In the Apache configuration file, `$IHS/conf/httpd.conf`, use the following directive to add the PingAccess agent for Apache's module configuration:

   ```
   Include conf/paa.conf
   ```
8. Copy the `<agentname>_agent.properties` file to the `$IHS/conf/agent.properties` directory.

   This is the configuration file that you downloaded in step 3 of *Installing on an IBM HTTP Server using Apache 2.2* on page 368.
9. Restart the Apache service by running the **`apachectl restart`** command.

Installing on a RHEL 8 system with an IBM HTTP Server

About this task

To manually install the PingAccess agent for Apache on RHEL 8 when using the IBM HTTP Server:

Steps

1. Go to the `pingaccess-agent-apache22-rhel8-<version>/<arch>/` directory.

   > **Note:**
   >
   > Currently, the only valid value for *<arch>* is `x86_64` for 64-bit.

   Example:

   `cd pingaccess-agent-apache22-rhel8-1.5.0/x86_64/`
2. Extract the package RPMs using the following command:

   ```
   mkdir pkgroot
   cp *.rpm pkgroot/
   cd pkgroot
   for r in *.rpm; do rpm2cpio $r | cpio -idmv; done
   ```
3. Run the **`cp`** command to copy the libraries to the appropriate Apache directories.

   Example:

   For RedHat Enterprise Linux 8 (x86_64):

   ```
   cp -av usr/lib64/*.so* $IHS/modules
   ```
4. Copy `mod_paa.so` into the Apache modules directory:

   ```
   cp -av usr/lib64/httpd/modules/mod_paa.so $IHS/modules
   ```
5. Copy the `paa.conf` file to the Apache configuration directory:

   ```
   cp -av etc/httpd/conf.d/paa.conf $IHS/conf
   ```
6. In the `paa.conf` file:
   a. Add the following lines before the `LoadModule` directive:

      ```
      LoadFile modules/libpgm-5.2.so.0
      LoadFile modules/libzmq.so.5
      ```
   b. Update the values for `PaaPropertyFiles` and `PaaCertificateDir` to point to your Apache `conf` directory.
7. In the Apache configuration file, `$IHS/conf/httpd.conf`, use the following directive to add the PingAccess agent for Apache's module configuration:

   ```
   Include conf/paa.conf
   ```
8. Copy the `<agentname>_agent.properties` file to the `$IHS/conf/agent.properties` directory.

   This is the configuration file that you downloaded in step 3 of *Installing on an IBM HTTP Server using Apache 2.2* on page 368.
9. Restart the Apache service by running the **`apachectl restart`** command.

Uninstalling the RHEL agent

Remove the PingAccess agent from a RHEL system.

About this task

- If you installed the PingAccess agent using the standard installation process, uninstall it with the following command:

  ```
  sudo yum remove pingaccess-agent*.x86_64
  ```
- If you installed the PingAccess agent manually with an IBM HTTP Server, uninstall it with the following procedure:

Steps

1. Remove the `$APACHE/conf/agent.properties` file.

   Example:

   ```
   rm $APACHE/conf/agent.properties
   ```
2. Remove the following directive from the Apache configuration file, `$APACHE/conf/httpd.conf`.

   Example:

   ```
   Include conf/10-paa.conf
   ```
3. Remove `10-paa.conf` from the `$APACHE/modules` directory.

   Example:

   ```
   rm $APACHE/modules/10-paa.conf
   ```
4. Remove all `.so` files from the `$APACHE/modules` directory.

   Example:

   ```
   rm $APACHE/modules/*.so*
   ```
5. Restart the Apache service with the **`apachectl restart`** command.

RHEL agent configuration

You manage the agent configuration through the `paa.conf` and `agent.properties` configuration files.

The `/etc/httpd/conf.d/paa.conf` file contains these configuration options.

Parameter	Definition	Default Value
`PaaCertificateDir`	String value containing the path to the certificates extracted from the `.properties` files.	`conf.d`

<table>
<tr><th>Parameter</th><th>Definition</th><th>Default Value</th></tr>
<tr><td>

`PaaEnabled`

</td><td>

Determines whether the agent is enabled or disabled for a specific server configuration. Valid values: `on/off`

This value can be set globally; set for individual virtual hosts, directories, locations, or files; or both. The most specific value is used.

Note:

If you disable the **`PaaEnabled`** parameter globally, ensure that the PaaEnabled directive is set to `on` for the PingAccess reserved application context root. This is `/pa` by default.

For example, adding this text to an included configuration file enables PingAccess for the `/pa` context root and for the `/var/www/html/one` directory.

```
<VirtualHost *:81>
    <Location /pa>
        PaaEnabled on
    </Location>
    <Directory "/var/www/
html/one">
        PaaEnabled on
    </Directory>
</VirtualHost>
```

Adding this text to an included configuration file disables PingAccess for all content in the `/var/www/html/two` directory except for files named `page2.html`.

```
<VirtualHost *:81>
    <Directory "/var/www/
html/two">
        PaaEnabled off
       <Files "page2.html">
           PaaEnabled on
       </Files>
    </Directory>
</VirtualHost>
```

</td><td>

`on`

</td></tr>
<tr><td>

`PaaPropertyFiles`

</td><td>

List of `.properties` files that store configuration data used to connect the agent to the PingAccess engine nodes the agent will communicate with.

</td><td>

`conf.d/agent.properties`

</td></tr>
</table>

Parameter	Definition	Default Value
PaaEnabledNoteName	An optional parameter which defines a note name. If a request includes a note with this name and a value of `on` or `off`, this value overrides the **PaaEnabled** setting for that request. If you want to use this feature, you must deploy a custom module to include this note with the correct value.	`paa-enabled-note`

The configured `agent.properties` files can contain the following parameters.

Parameter	Definition	Default Value
agent.engine.configuration.scheme	The URI scheme used to connect to the engine node. Valid values are `http` and `https`.	https
agent.engine.configuration.host	The PingAccess host name.	The value in the Agent Node's `PingAccess Host` field.
agent.engine.configuration.port	The port the agent connects to on the PingAccess host. This value is defined in the PingAccess `run.properties` file.	Defined in the PingAccess Admin UI
agent.engine.configuration.username	The unique agent name that identifies the agent in PingAccess.	Defined in the PingAccess Admin UI
agent.engine.configuration.shared.secret	The password used to authenticate the agent to the engine.	Defined in the PingAccess Admin UI
agent.engine.configuration.bootstrap.truststore	The base64-encoded public certificate used to establish HTTPS trust by the agent to the PingAccess engine. **Note:** If you are having difficulty connecting an agent to the PingAccess engine, verify that the Agent Trusted Certificate has been configured correctly in *Agent Management*.	Generated by PingAccess
agent.engine.configuration.maxConnections	The number of connections a single web server worker process maintains to the PingAccess engine defined in the **agent.engine.configuration.host** parameter.	10

Parameter	Definition	Default Value
`agent.engine.configuration.timeout`	The maximum time, in milliseconds, a request to PingAccess can take from the agent. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
`agent.engine.configuration.connectTimeout`	The maximum time, in milliseconds, the agent can take to connect to the PingAccess engine. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
`agent.cache.missInitialTimeout`	The maximum time, in milliseconds, a web server worker process waits for a response to a policy cache request sent to other web server worker processes.	5
`agent.cache.broker.publisherPort`	The network port web server processes use to publish policy cache requests to other web server worker processes. This port is bound to the localhost network only.	3031
`agent.cache.broker.subscriberPort`	The network port web server processes use to receive policy cache requests from other web server worker processes. This port is bound to the localhost network only.	3032
`agent.cache.maxTokens`	The maximum number of tokens stored in the policy cache for a single web server worker process. A value of `0` means there is no maximum.	0
`agent.cache.disabled`	Determines whether caching of policy decisions is enabled or disabled. A value of `1` disables caching, forcing the agent to communicate with the PingAccess host any time a policy decision needs to be made. **Warning:** Disabling caching has a significant impact on the scalability of the PingAccess Policy servers, as every rule evaluation is processed by the Policy Server. This option should only be used as a last resort because of the performance penalty.	0

Parameter	Definition	Default Value
`agent.engine.configuration.failover.hosts`	The host name and port of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess host.	Defined in the PingAccess Admin UI
`agent.engine.configuration.failover.failedRetryTimeout`	Seconds before retrying a failed PingAccess server.	60
`agent.engine.configuration.failover.maxRetries`	The maximum number of retries before considering a PingAccess server unavailable.	2
`agent.cache.type`	Controls the type of policy cache used by the agent. There are three valid values for this property: ▪ Auto - The Auto cache type determines the appropriate cache to use based on the number of worker processes. If the number of worker processes is 1, the agent uses the Standalone cache. If the number of worker processes is 2 or more, the agent uses the ZMQ cache. ▪ Standalone - The Standalone cache type does not share policy cache entries across worker processes. ▪ ZMQ - The ZMQ cache type allows the agent to share policy cache entries across all worker processes using ZeroMQ for inter-process communication.	AUTO

Parameter	Definition	Default Value

`agent.send.inventory`	Determines whether the `vnd-pi-agent` agent inventory header is sent along with each request to the PingAccess policy server.
	
	This header contains the following fields:
	
	`v`
	The PingAccess agent version.
	
	`t`
	The type of PingAccess agent retrieved using the `ap_get_server_description` function.
	
	`h`
	The hostname of the PingAccess agent retrieved using the `ServerName` directive.
	
	For more information, see *Agent inventory logging* on page 117.
		`true`

`agent.inventory`	Specifies additional values to include in the `vnd-pi-agent` agent inventory header. The following syntax is used. `agent.inventory=exampleheader=TEST;exampleheader2=TEST2;` **Note:** The specified header fields are case-sensitive.	Not present by default.
`agent.apache.host.source.headerName`	If present, specifies a header that overrides the default `X-Forwarded-Host` header. This header communicates the authority component of the effective request URL on the protected application.	Not present by default.

You can add comments to the `agent.properties` files if necessary. Lines beginning with the `#` or `!` characters are ignored by the agent.

Changes to the `agent.properties` file require a restart of the web server.

Tip:

See the *Performance Tuning Guide* for a discussion on improving agent performance.

Log configuration

The PingAccess agent for Apache writes its information to the standard Apache error log, defined in the Apache configuration with the `ErrorLog` configuration directive.

All information logged by the PingAccess agent is prefaced with the string `[paa]`. PingAccess agent monitoring and performance information is prefaced with the string `[paa-monitoring]` and contains information about how long the PingAccess agent took to fill a cache request and how long the total policy decision took.

The LogLevel used by the PingAccess agent module is taken from the top-level `httpd.conf` configuration.

Rotating a CA

Rotate the certificate authority (CA) used by an agent while minimizing the impact to agent communications.

Steps

1. On the agent web server, update the `agent.properties` file to add the new CA certificate.
   a. Concatenate the old and new CA certificates in PEM encoding format into a new file.
   b. Encode the contents of the file to Base64.
   c. Open the `agent.properties` file and set the value of the `agent.engine.configuration.bootstrap.truststore` line to the encoded content.

      Example:

      ```
      agent.engine.configuration.bootstrap.truststore=<Encoded_content>
      ```
2. Restart the agent web server.
3. Update the configuration to use a new server certificate signed by the new CA for the agent HTTPS listener.
   a. Identify a key pair to use. If necessary, create a new key pair.

      For more information, see *Generating new key pairs* on page 313.
   b. Generate a CSR for that key pair.

      For more information, see *Generating certificate signing requests* on page 314.
   c. Submit that CSR to the new CA to get a new signed certificate.
   d. Import the CSR response (the new certificate) into .

      For more information, see *Importing certificates* on page 309.
   e. Assign the key pair to the agent HTTPS listener.

      For more information, see *Assigning key pairs to HTTPS listeners* on page 315.

Troubleshooting

The following table lists some potential problems and resolutions you might encounter with the PingAccess agent for RHEL.

Issue	Resolution

Agent receives an unknown protocol error when attempting to contact the administrative node	This can indicate that the operating system is using sha1 for encryption. This protocol is no longer supported by default in PingAccess.
	
	We recommend switching to SHA-256. If you cannot switch to SHA-256 you can re-enable SHA-1:
	
	1. Open the `run.properties` file.
	2. Add TLSv1 to the protocol list. For example,
	`tls.default.protocols=TLSv1, TLSv1.1, TLSv1.2, TLSv1.3`
	3. Add the SHA entries to the cipher suites list. For example,
	`tls.default.cipherSuites = TLS_CHACHA20_POLY1305_SHA256,\`
	`TLS_AES_256_GCM_SHA384,\`
	`TLS_AES_128_GCM_SHA256,\`
	`TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\`
	`TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\`
	`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,\`
	`TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,\`
	`TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,\`
	`TLS_EMPTY_RENEGOTIATION_INFO_SCSV, \`
	`TLS_RSA_WITH_AES_128_CBC_SHA, \`
	`TLS_DHE_RSA_WITH_AES_128_CBC_SHA, \`
	`TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, \`
	`TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, \`
	`TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA`

Release Notes

The following release notes summarize the changes in current and previous PingAccess agent for Apache (RHEL) updates.

Version History

- **Version 1.5.2** – September 2021

  Agent SDK for C version 1.3

  - Added an option to override the default `X-Forwarded-Host` header with a specified header.

- **Version 1.5.1** – April 2021

  Agent SDK for C version 1.3

  - Fixed an issue that caused large bodies sent through POST preservation in an agent deployment to be corrupted.
- **Version 1.5** – June 2020

  Agent SDK for C version 1.3

  - Added support for RHEL 8
  - Added agent inventory callback API
  - Removed support for RHEL 6
- **Version 1.4.1** – February 2020

  Agent SDK for C version 1.2.1

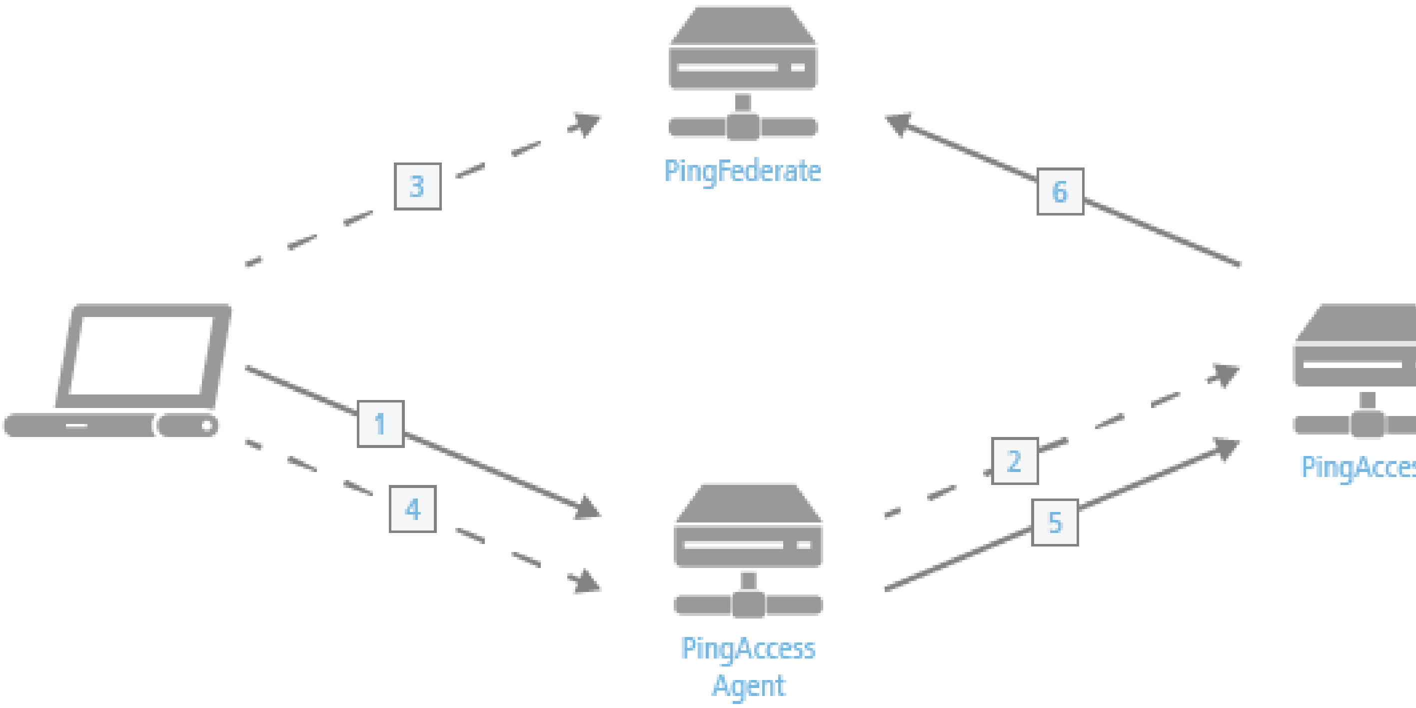
  - Fixed a potential security issue
- **Version 1.4** – June 2019

  Agent SDK for C version 1.2.0

  - The PAA Enabled directive can be used inside a directory or location container.
  - Added ability to set policy caching mechanism using a property in the `agent.properties` file.
  - Added ability to enable or disable agent processing for a request based on a note field.
  - Fixed a potential security issue.
- **Version 1.3.2** – November 2018
  - Fixed a potential security issue
- **Version 1.3** – February 2017
  - Added support for Apache 2.4 on RHEL 6.
  - The agent can be disabled for specific hosts using the new configuration option: `PaaEnabled`.
- **Version 1.2** – May 2016
  - Added support for IBM HTTP Server
- **Version 1.1** – December 2014
  - Added Support for Apache 2.4 on Red Hat Enterprise Linux 7
  - Corrected a potential security issue related to caching, SECBL007. This security bulletin is available in the *Ping Identity Support Portal*.
- **Version 1.0** – July 2014
  - Initial release

PingAccess Agent for Apache (SLES)

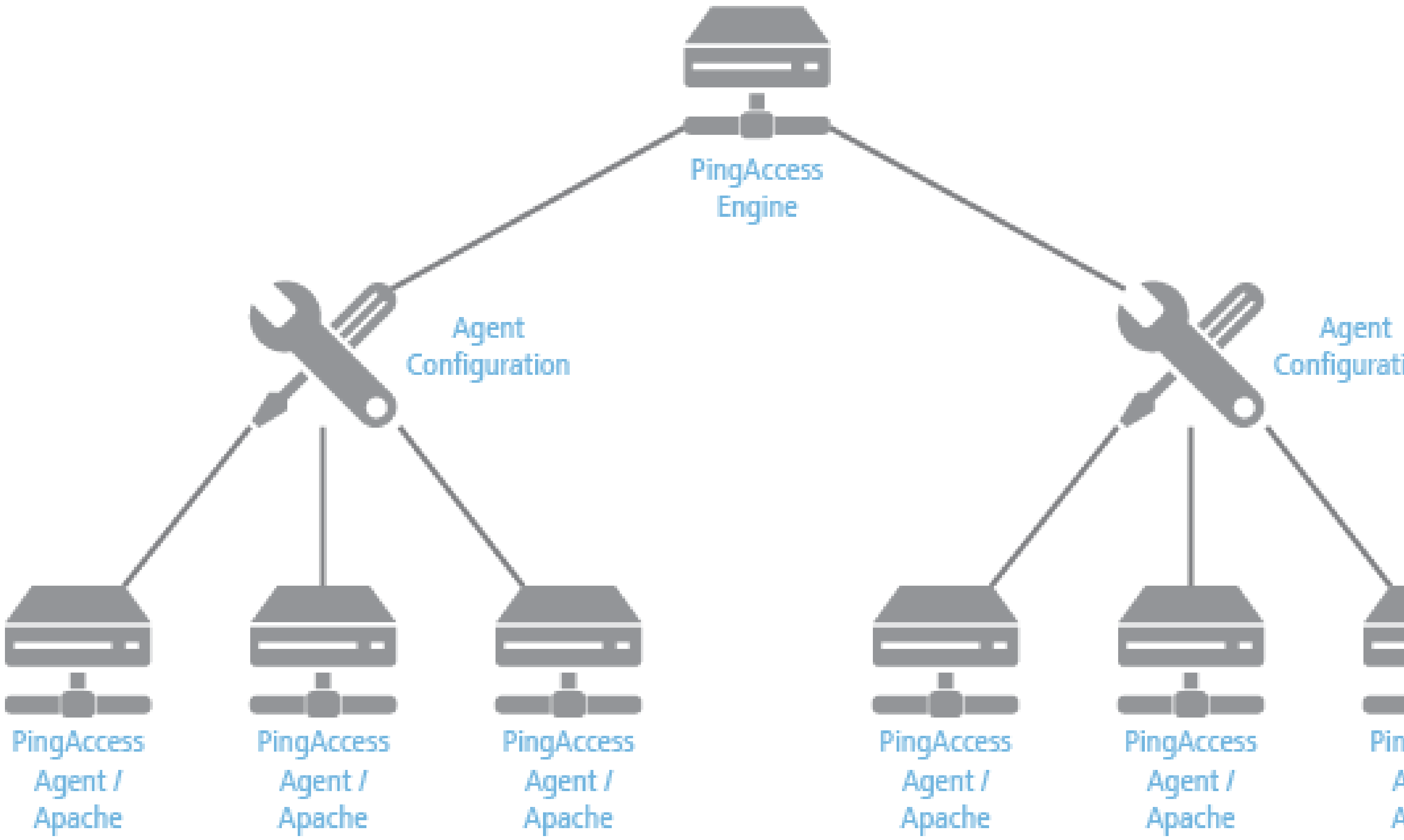
The PingAccess agent for Apache is an Apache module that intercepts requests to the web server's protected resources and evaluates applicable access control policies. These policies are evaluated by either accessing a locally cached policy decision or by querying the PingAccess engine node.



The process used when a PingAccess agent is added to the policy decision process is as follows:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

Within the PingAccess administrative console, agent nodes are configured with information that allows an agent to connect to the engine node to retrieve information about access control policies for resources within that agent's control. An agent configuration has a one-to-many relationship with PingAccess agents, allowing a single agent configuration bootstrap file to be used on multiple web servers within a server farm.



Tip:

An agent node is a shared configuration used by one or more agents, rather than a specific agent instance.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

System requirements

The PingAccess agent for Apache (SLES) is supported on these platforms.

- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2012, VC14 or later
- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2016, VC14 or later
- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2019, VC14 or later
- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2022, VC14 or later

As with any system that is reachable from the internet, the server should be properly hardened.

Installing on SLES

Install a PingAccess agent on a SUSE Linux Enterprise Server (SLES) system.

Before you begin

This procedure makes the following assumptions:

- The Apache configuration directory is `/etc/apache2/conf.d`
- The Apache modules directory is `/usr/lib64/apache2`

For custom installations:

- Modify the configuration and module paths below as needed.
- Edit the included `paa.conf` file to modify the values for Apache's configuration and module directories.

Steps

1. Download and extract `pingaccess-agent-apache<version>.zip`.
2. Go to the `pingaccess-agent-apache<version>` directory.
3. If you are installing on SLES 12, import the gpg key.

   Example:

   ```
   rpm --import https://download.opensuse.org/repositories/network:/
   messaging:/zeromq:/release-stable/SLE_12_SP4/repodata/repomd.xml.key
   ```
4. Install the dependencies.

   Example:

   ```
   zypper in ./x86_64/lib*.rpm
   ```
5. As root, copy the PingAccess Agent for Apache files to the appropriate places.

   Example:

   ```
    cp ./x86_64/mod_paa.so /usr/lib64/apache2
   cp paa.conf /etc/apache2/conf.d
   ```

> **Note:**
>
> By default, Apache on SLES will automatically include all `.conf` files contained within `conf.d` using the IncludeOptional directive. If this has been disabled, add the following to Apache's `httpd.conf`.
>
> ```
>  Include /etc/apache2/conf.d/paa.conf
> ```

6. Sign on to the PingAccess console.
7. Click **Applications** and then go to **Agents**.
8. Edit a configured agent.

   If the agent has not yet been created, see the *PingAccess User Interface Reference Guide*.
9. In the shared secret, click the **Download** icon to download the configuration.

   The configuration file will be named `<agentname>_agent.properties`.
10. Copy the `<agentname>_agent.properties` file to `/etc/apache2/conf.d/agent.properties`
11. As root, restart the Apache service using one of the following commands:
    Choose from:

    - `rcapache2 restart`
    - `$APACHE_ROOT/bin/apachectl restart`

Uninstalling on SLES

Remove the PingAccess agent from a SUSE Linux Enterprise Server (SLES) system.

Steps

- Remove the PingAccess agent for Apache files.

  Example:

  ```
  rm /usr/lib64/apache2/mod_paa.so
  rm /etc/apache2/conf.d/paa.conf
  ```

SLES agent configuration

The agent configuration is managed through the `paa.conf` and `agent.properties` configuration files.

The `/etc/httpd/conf.d/paa.conf` file contains the following configuration options.

Parameter	Definition	Default Value
`PaaCertificateDir`	String value containing the path to the certificates extracted from the `.properties` files.	`conf.d`

<table>
<tr><th>Parameter</th><th>Definition</th><th>Default Value</th></tr>
<tr>
<td>PaaEnabled</td>
<td>Determines whether the agent is enabled or disabled for a specific server configuration. Valid values: on/off

This value can be set globally; set for individual virtual hosts, directories, locations, or files; or both. The most specific value is used.

Note:

If you disable the PaaEnabled parameter globally, ensure that the PaaEnabled directive is set to on for the PingAccess reserved application context root. This is /pa by default.

For example, adding this text to an included configuration file enables PingAccess for the /pa context root and for the /var/www/html/one directory.

<VirtualHost *:81>
    <Location /pa>
        PaaEnabled on
    </Location>
    <Directory "/var/www/html/one">
        PaaEnabled on
    </Directory>
</VirtualHost>

Adding this text to an included configuration file disables PingAccess for all content in the /var/www/html/two directory except for files named page2.html.

<VirtualHost *:81>
    <Directory "/var/www/html/two">
        PaaEnabled off
        <Files "page2.html">
            PaaEnabled on
        </Files>
    </Directory>
</VirtualHost></td>
<td>on</td>
</tr>
</table>

Parameter	Definition	Default Value
`PaaPropertyFiles`	List of `.properties`files that store configuration data used to connect the agent to the PingAccess engine nodes the agent will communicate with.	`conf.d/agent.properties`
`PaaEnabledNoteName`	An optional parameter that defines a note name. If a request includes a note with this name and a value of `on` or `off`, this value overrides the **`PaaEnabled`** setting for that request. If you want to use this feature, you must deploy a custom module to include this note with the correct value.	`paa-enabled-note`

Note:

It is not necessary to make any changes to `paa.conf` if the steps in the *Installation* section were followed.

The configured `agent.properties` files can contain the following parameters.

Parameter	Definition	Default Value
`agent.engine.configuration.scheme`	The URL scheme used to connect to the engine node. Valid values are `http` and `https`.	`https`
`agent.engine.configuration.host`	The PingAccess hostname.	The value in the Agent Node's `PingAccess Host` field.
`agent.engine.configuration.port`	The port the agent connects to on the PingAccess host. This value is defined in the PingAccess `run.properties` file.	Defined in the PingAccess Admin UI
`agent.engine.configuration.username`	The unique agent name that identifies the agent in PingAccess.	Defined in the PingAccess Admin UI
`agent.engine.configuration.shared.secret`	The password used to authenticate the agent to the engine.	Defined in the PingAccess Admin UI

Parameter	Definition	Default Value
agent.engine.configuration.bootstrap.truststore	The base64-encoded public certificate used to establish HTTPS trust by the agent to the PingAccess engine. **Note:** If you are having difficulty connecting an agent to the PingAccess engine, verify that the Agent Trusted Certificate has been configured correctly in *Agent Management*.	Generated by PingAccess
agent.engine.configuration.maxConnections	The number of connections a single web server worker process maintains to the PingAccess engine defined in the **agent.engine.configuration.host parameter**.	10
agent.engine.configuration.timeout	The maximum time, in milliseconds, a request to PingAccess can take from the agent. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
agent.engine.configuration.connectTimeout	The maximum time, in milliseconds, the agent can take to connect to the PingAccess engine. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
agent.cache.missInitialTimeout	The maximum time, in milliseconds, a web server worker process waits for a response to a policy cache request sent to other web server worker processes.	5
agent.cache.broker.publisherPort	The network port web server processes use to publish policy cache requests to other web server worker processes. This port is bound to the localhost network only.	3031
agent.cache.broker.subscriberPort	The network port web server processes use to receive policy cache requests from other web server worker processes. This port is bound to the localhost network only.	3032

Parameter	Definition	Default Value
`agent.cache.maxTokens`	The maximum number of tokens stored in the policy cache for a single web server worker process. A value of 0 means there is no maximum.	0
`agent.cache.disabled`	Determines whether caching of policy decisions is enabled or disabled. A value of `1` disables caching, forcing the agent to communicate with the PingAccess host any time a policy decision needs to be made. **Warning:** Disabling caching has a significant impact on the scalability of the PingAccess Policy servers, as every rule evaluation is processed by the Policy Server. This option should only be used as a last resort because of the performance penalty.	0
`agent.engine.configuration.failover.hosts`	The hostname and port of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess host.	Defined in the PingAccess Admin UI
`agent.engine.configuration.failover.failedRetryTimeout`	Seconds before retrying a failed PingAccess server.	60
`agent.engine.configuration.failover.maxRetries`	The maximum number of retries before considering a PingAccess server unavailable.	2

Parameter	Definition	Default Value
`agent.cache.type`	Controls the type of policy cache used by the agent. There are three valid values for this property: AUTO The AUTO cache type determines the appropriate cache to use based on the number of worker processes. If the number of worker processes is 1, the agent uses the STANDALONE cache. If the number of worker processes is 2 or more, the agent uses the ZMQ cache. STANDALONE The STANDALONE cache type does not share policy cache entries across worker processes ZMQ The ZMQ cache type allows the agent to share policy cache entries across all worker processes using ZeroMQ for inter-process communication	AUTO

Parameter	Definition	Default Value

`agent.send.inventory`	Determines whether the `vnd-pi-agent` agent inventory header is sent along with each request to the PingAccess policy server.
	
	This header contains the following fields:
	
	`v`
	The PingAccess agent version.
	
	`t`
	The type of PingAccess agent retrieved using the `ap_get_server_description` function.
	
	`h`
	The hostname of the PingAccess agent retrieved using the `ServerName` directive.
	
	For more information, see *Agent inventory logging* on page 117.
		`true`

`agent.inventory`	Specifies additional values to include in the `vnd-pi-agent` agent inventory header. The following syntax is used. `agent.inventory=exampleheader=TEST;exampleheader2=TEST2;` **Note:** The specified header fields are case-sensitive.	Not present by default.
`agent.apache.host.source.headerName`	If present, specifies a header that overrides the default `X-Forwarded-Host` header. This header communicates the authority component of the effective request URL on the protected application.	Not present by default.

You can add comments to the `agent.properties` files if necessary. Lines beginning with the `#` or `!` characters are ignored by the agent.

Changes to the `agent.properties` file require a restart of the web server.

Tip:

For a discussion on improving agent performance, see the *Performance Tuning Guide*.

Log Configuration

The PingAccess agent for Apache writes its information to the standard Apache error log, defined in the Apache configuration with the `ErrorLog` configuration directive.

All information logged by the agent is prefaced with the string `[paa]`. Agent monitoring and performance information is prefaced with the string `[paa-monitoring]`, and contains information about how long the agent took to fill a cache request and how long the total policy decision took.

The LogLevel used by the agent module is taken from the top-level `httpd.conf` configuration.

Rotating a CA

Rotate the certificate authority (CA) used by an agent while minimizing the impact to agent communications.

Steps

1. On the agent web server, update the `agent.properties` file to add the new CA certificate.
    a. Concatenate the old and new CA certificates in PEM encoding format into a new file.
    b. Encode the contents of the file to Base64.
    c. Open the `agent.properties` file and set the value of the `agent.engine.configuration.bootstrap.truststore` line to the encoded content.

    Example:

    ```
    agent.engine.configuration.bootstrap.truststore=<Encoded_content>
    ```
2. Restart the agent web server.
3. Update the configuration to use a new server certificate signed by the new CA for the agent HTTPS listener.
    a. Identify a key pair to use. If necessary, create a new key pair.

    For more information, see *Generating new key pairs* on page 313.

    b. Generate a CSR for that key pair.

    For more information, see *Generating certificate signing requests* on page 314.

    c. Submit that CSR to the new CA to get a new signed certificate.
    d. Import the CSR response (the new certificate) into .

    For more information, see *Importing certificates* on page 309.

    e. Assign the key pair to the agent HTTPS listener.

    For more information, see *Assigning key pairs to HTTPS listeners* on page 315.

Troubleshooting

The following table lists some potential problems and resolutions you might encounter with the PingAccess agent for SUSE Linux Enterprise Server (SLES).

Issue	Resolution

Agent receives an unknown protocol error when attempting to contact the administrative node	This can indicate that the operating system is using sha1 for encryption. This protocol is no longer supported by default in PingAccess.
	
	We recommend switching to SHA-256. If you cannot switch to SHA-256, you can re-enable SHA-1:
	
	1. Open the `run.properties` file.
	2. Add TLSv1 to the protocol list.
	`tls.default.protocols=TLSv1, TLSv1.1, TLSv1.2, TLSv1.3`
	3. Add the SHA entries to the cipher suites list.
	`tls.default.cipherSuites = TLS_CHACHA20_POLY1305_SHA256,\ TLS_AES_256_GCM_SHA384,\ TLS_AES_128_GCM_SHA256,\ TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\ TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\ TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,\ TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,\ TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,\ TLS_EMPTY_RENEGOTIATION_INFO_SCSV, \ TLS_RSA_WITH_AES_128_CBC_SHA, \ TLS_DHE_RSA_WITH_AES_128_CBC_SHA, \ TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, \ TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, \ TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA`

Release Notes

These release notes summarize the changes in current and previous PingAccess agent for Apache (SLES) updates.

Version History

Version 1.5.2 – September 2021

Agent SDK for C version 1.3

Added an option to override the default `X-Forwarded-Host` header with a specified header.

Version 1.5.1 – April 2021

Agent SDK for C version 1.3

- Fixed an issue that caused large bodies sent through POST preservation in an agent deployment to be corrupted.

Version 1.5 – July 2020

Agent SDK for C version 1.3

- Added agent inventory callback API

Version 1.4.1 - February 2020

Agent SDK for C version 1.2.1

- Fixed a potential security issue

Version 1.4 – June 2019

Agent SDK for C version 1.2.0

- The PAA Enabled directive can now be used inside a directory or location container
- Added ability to set policy caching mechanism using a property in the `agent.properties` file
- Added ability to enable or disable agent processing for a request based on a note field
- Fixed a potential security issue

Version 1.3.2 – November 2018

Fixed a potential security issue

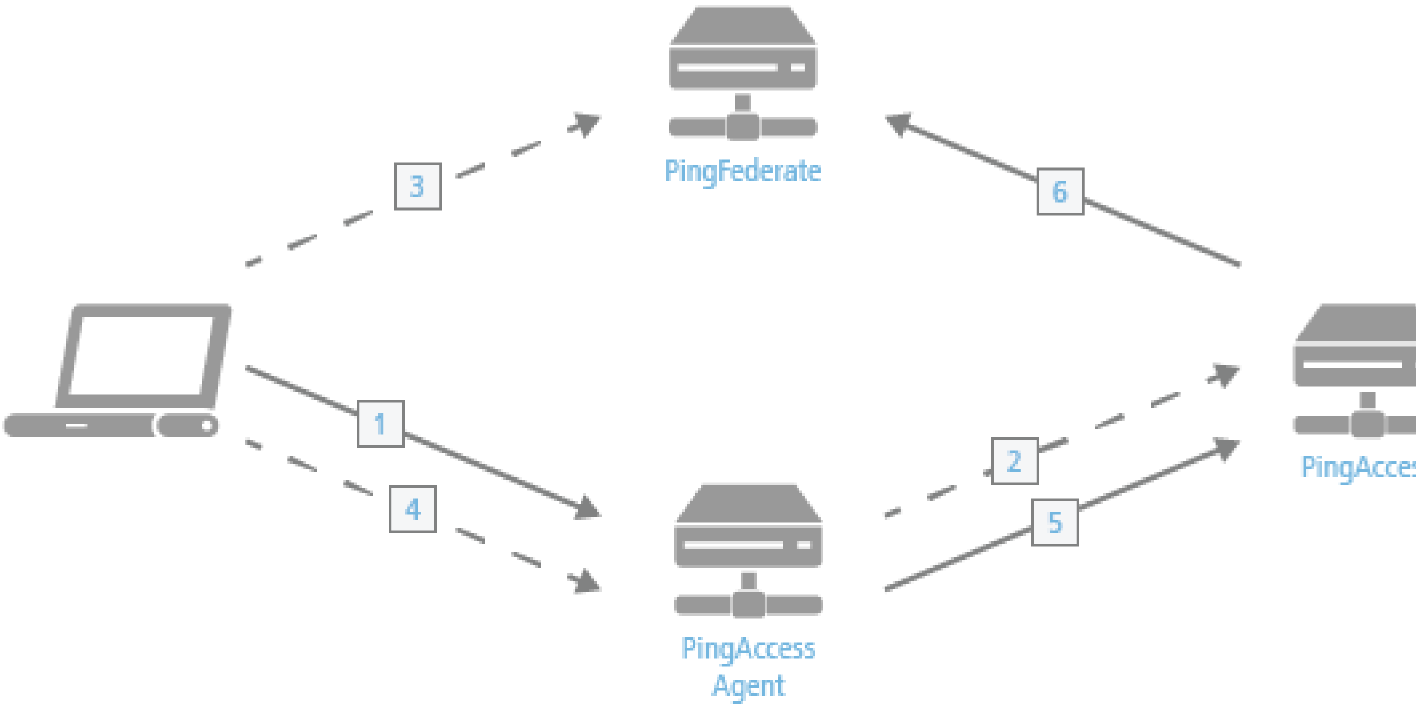
Version 1.3 – March 2017

Initial release for Apache 2.2 on SUSE Linux Enterprise Server (SLES) 11 and Apache 2.4 on SLES 12

> **Note:**
>
> Version is aligned with PingAccess agent for Apache (RHEL)

PingAccess Agent for Apache (Windows)

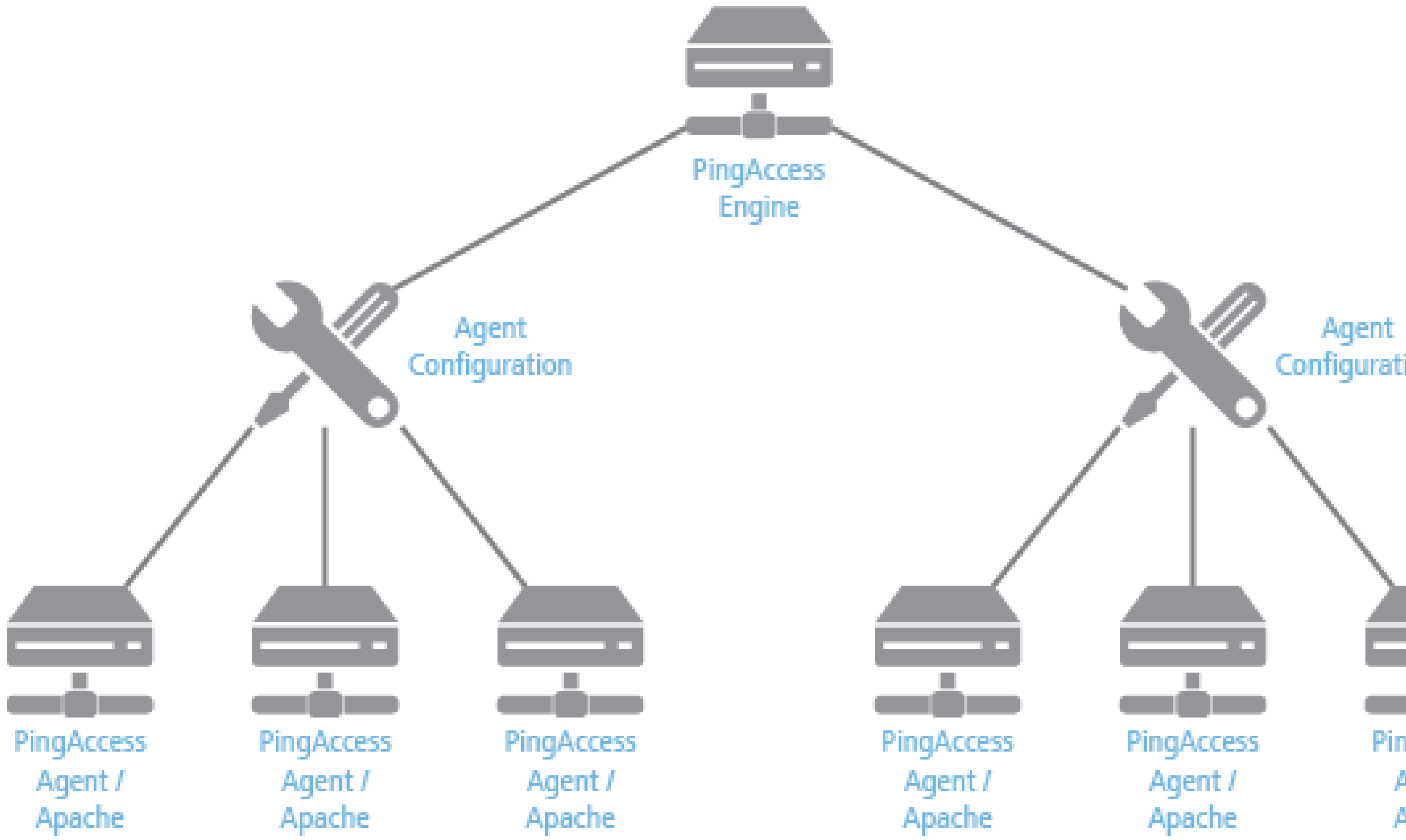
The PingAccess agent for Apache is an Apache module that intercepts requests to the web server's protected resources and evaluates applicable access control policies. These policies are evaluated by either accessing a locally cached policy decision or by querying the PingAccess engine node.



The process used when a PingAccess agent is added to the policy decision process is as follows:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

Within the PingAccess administrative console, agent nodes are configured with information that allows a PingAccess agent to connect to the engine node to retrieve information about access control policies for resources within that agent's control. An agent configuration has a one-to-many relationship with PingAccess agents, allowing a single agent configuration bootstrap file to be used on multiple web servers within a server farm.



Tip:

An agent node is a shared configuration used by one or more agents, rather than a specific agent instance.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Apache (Windows) agent system requirements

The PingAccess agent for Apache (Windows) is supported on these platforms.

- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2016, VC14 or later
- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2019, VC14 or later
- Apache HTTP Server 2.4 64-bit running on Microsoft Windows Server 2022, VC14 or later

As with any system that is reachable from the internet, the server should be properly hardened.

Installing on Windows

Install a PingAccess agent on a Windows system.

Before you begin

This procedure makes the following assumptions:

- The Apache configuration directory is `C:/apache24/conf`.
- The Apache modules directory is `C:/apache24/modules`.

For custom installations:

- Modify the configuration and module paths below as needed.

- To modify the values for Apache’s configuration and module directories, edit the included `paa.conf` file.

Steps

1. Download and extract `pingaccess-agent-apache<version>.zip`.
2. Go to the `pingaccess-agent-apache<version>` directory.
3. Copy the `paa.conf` file into the Apache configuration directory.
4. Add the following to Apache's `httpd.conf` file.

   ```
   Include conf/paa.conf
   ```
5. Copy the `paa` folder into the Apache modules directory.
6. Sign on to the PingAccess console.
7. Click **Applications** and then go to **Agents**.
8. Edit a configured agent.

   If the agent has not yet been created, see *Adding agents*.
9. In the shared secret, click **Download** to download the configuration.

   The configuration file is named `<agentname>_agent.properties`.
10. On the Agent system, create the `C:/apache24/conf.d` folder if it does not exist.
11. Copy the `<agentname>_agent.properties` file to `C:/apache24/conf.d/agent.properties`.
12. Restart Apache.

Uninstalling on Windows

Remove the PingAccess agent from a Windows system.

Steps

1. Remove the `C:/apache24/conf.d` folder and its contents.
2. Remove the `paa` folder from the Apache modules directory.
3. Remove the following from Apache's `httpd.conf` file.

   ```
   Include conf/paa.conf
   ```
4. Remove the `paa.conf` file from the Apache configuration directory.
5. Remove the `pingaccess-agent-apache<version>` directory.
6. Remove the `pingaccess-agent-apache<version>.zip` file if it is present.
7. Restart Apache.

Configuration

You manage the agent configuration through the `paa.conf` and `agent.properties` configuration files.

The `C:/Apache24/conf/paa.conf` file contains these configuration options.

Parameter	Definition	Default Value
`PaaCertificateDir`	String value containing the path to the certificates extracted from the `.properties` files.	`conf.d`

<table>
<tr><th>Parameter</th><th>Definition</th><th>Default Value</th></tr>
<tr><td><code>PaaEnabled</code></td><td>
Determines whether the agent is enabled or disabled for a specific server configuration. Valid values: <code>on/off</code>
This value can be set globally; set for individual virtual hosts, directories, locations, or files; or both. The most specific value is used.
Note:
If you disable the <code>PaaEnabled</code> parameter globally, ensure that the PaaEnabled directive is set to <code>on</code> for the PingAccess reserved application context root. This is <code>/pa</code> by default.
For example, adding this text to an included configuration file enables PingAccess for the <code>/pa/var/www/html/one</code> directory.
<pre><VirtualHost *:81>
    <Location /pa>
        PaaEnabled on
    </Location>
    <Directory "/var/
www/html/one">
        PaaEnabled on
    </Directory>
</VirtualHost> context
 root

              and for
 the</pre>
Adding this text to an included configuration file disables PingAccess for all content in the <code>/var/www/html/two</code> directory except for files named <code>page2.html</code>.
<pre><VirtualHost *:81>
    <Directory "/var/
www/html/two">
        PaaEnabled off
       <Files
 "page2.html">
            PaaEnabled
 on
       </Files>
    </Directory>
</VirtualHost></pre>
</td><td><code>on</code></td></tr>
</table>

Parameter	Definition	Default Value
`PaaPropertyFiles`	List of `.properties` files that store configuration data used to connect the agent to the PingAccess engine nodes the agent will communicate with.	`conf.d/agent.properties`
`PaaEnabledNoteName`	An optional parameter which defines a note name. If a request includes a note with this name and a value of `on` or `off`, this value overrides the **`PaaEnabled`** setting for that request. If you want to use this feature, you must deploy a custom module to include this note with the correct value.	`paa-enabled-note`

The configured `agent.properties` files can contain the following parameters.

Parameter	Definition	Default Value
`agent.engine.configuration.scheme`	The URI scheme used to connect to the engine node. Valid values are `http` and `https`.	https
`agent.engine.configuration.host`	The PingAccess hostname.	The value in the Agent Node's **`PingAccess Host`** field.
`agent.engine.configuration.port`	The port the agent connects to on the PingAccess host. This value is defined in the PingAccess `run.properties` file.	Defined in the PingAccess Admin UI
`agent.engine.configuration.username`	The unique agent name that identifies the agent in PingAccess.	Defined in the PingAccess Admin UI
`agent.engine.configuration.shared.secret`	The password used to authenticate the agent to the engine.	Defined in the PingAccess Admin UI
`agent.engine.configuration.bootstrap.truststore`	The base64-encoded public certificate used to establish HTTPS trust by the agent to the PingAccess engine. **Note:** If you are having difficulty connecting an agent to the PingAccess engine, verify that the Agent Trusted Certificate has been configured correctly in *Agent Management*.	Generated by PingAccess

Parameter	Definition	Default Value
agent.engine.configuration.maxConnections	The number of connections a single web server worker process maintains to the PingAccess engine defined in the **agent.engine.configuration.host parameter**.	10
agent.engine.configuration.timeout	The maximum time (in milliseconds) a request to PingAccess can take from the agent. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
agent.engine.configuration.connectTimeout	The maximum time (in milliseconds) the agent can take to connect to the PingAccess engine. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
agent.cache.missInitialTimeout	The maximum time (in milliseconds) a web server worker process waits for a response to a policy cache request sent to other web server worker processes.	5
agent.cache.broker.publisherPort	The network port web server processes use to publish policy cache requests to other web server worker processes. This port is bound to the localhost network only.	3031
agent.cache.broker.subscriberPort	The network port web server processes use to receive policy cache requests from other web server worker processes. This port is bound to the localhost network only.	3032
agent.cache.maxTokens	The maximum number of tokens stored in the policy cache for a single web server worker process. A value of 0 means there is no maximum.	0

Parameter	Definition	Default Value
`agent.cache.disabled`	Determines whether caching of policy decisions is enabled or disabled. A value of `1` disables caching, forcing the agent to communicate with the PingAccess host any time a policy decision needs to be made. **Warning:** Disabling caching has a significant impact on the scalability of the PingAccess Policy servers, as every rule evaluation is processed by the Policy Server. This option should only be used as a last resort because of the performance penalty.	0
`agent.engine.configuration.failover.hosts`	The hostname and port of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess host.	Defined in the PingAccess Admin UI
`agent.engine.configuration.failover.failedRetryTimeout`	Seconds before retrying a failed PingAccess server.	60
`agent.engine.configuration.failover.maxRetries`	The maximum number of retries before considering a PingAccess server unavailable.	2

Parameter	Definition	Default Value

`agent.cache.type`	Controls the type of policy cache used by the agent. There are three valid values for this property:
	`AUTO`
	The AUTO cache type determines the appropriate cache to use based on the number of worker processes. If the number of worker processes is 1, or 16 or above, the agent uses the STANDALONE cache. If the number of worker processes is between 2 and 15, the agent uses the ZMQ cache.
	`STANDALONE`
	The STANDALONE cache type does not share policy cache entries across worker processes.
	`ZMQ`
	The ZMQ cache type allows the agent to share policy cache entries across all worker processes using ZeroMQ for inter-process communication.
		AUTO

Parameter	Definition	Default Value

`agent.send.inventory`	Determines whether the `vnd-pi-agent` agent inventory header is sent along with each request to the PingAccess policy server.
	This header contains the following fields:
	`v`
	The PingAccess agent version.
	`t`
	The type of PingAccess agent retrieved using the `ap_get_server_description` function.
	`h`
	The hostname of the PingAccess agent retrieved using the `ServerName` directive.
	For more information, see *Agent inventory logging* on page 117.
		`true`

`agent.inventory`	Specifies additional values to include in the `vnd-pi-agent` agent inventory header. The following syntax is used. `agent.inventory=exampleheader=TEST;exampleheader2=TEST2;` **Note:** The specified header fields are case-sensitive.	Not present by default.
`agent.apache.host.source.headerName`	If present, specifies a header that overrides the default `X-Forwarded-Host` header. This header communicates the authority component of the effective request URL on the protected application.	Not present by default.

Add comments to the `agent.properties` files if necessary. Lines beginning with the `#` or `!` characters are ignored by the agent.

Changes to the `agent.properties` file require a restart of the web server.

Tip:

See the *Performance tuning guide* for a discussion on improving agent performance.

Log configuration

The PingAccess agent for Apache writes its information to the standard Apache error log, defined in the Apache configuration with the `ErrorLog` configuration directive.

All information logged by the agent is prefaced with the string `[paa]`. Agent monitoring and performance information is prefaced with the string `[paa-monitoring]` and contains information about how long the agent took to fill a cache request and how long the total policy decision took.

The LogLevel used by the PingAccess agent module is taken from the top-level `httpd.conf` configuration.

Rotating a CA

Rotate the certificate authority (CA) used by an agent while minimizing the impact to agent communications.

Steps

1. On the agent web server, update the `agent.properties` file to add the new CA certificate.
   a. Concatenate the old and new CA certificates in PEM encoding format into a new file.
   b. Encode the contents of the file to Base64.
   c. Open the `agent.properties` file and set the value of the `agent.engine.configuration.bootstrap.truststore` line to the encoded content.

      Example:

      ```
      agent.engine.configuration.bootstrap.truststore=<Encoded_content>
      ```
2. Restart the agent web server.
3. Update the configuration to use a new server certificate signed by the new CA for the agent HTTPS listener.
   a. Identify a key pair to use. If necessary, create a new key pair.

      For more information, see *Generating new key pairs* on page 313.
   b. Generate a CSR for that key pair.

      For more information, see *Generating certificate signing requests* on page 314.
   c. Submit that CSR to the new CA to get a new signed certificate.
   d. Import the CSR response (the new certificate) into .

      For more information, see *Importing certificates* on page 309.
   e. Assign the key pair to the agent HTTPS listener.

      For more information, see *Assigning key pairs to HTTPS listeners* on page 315.

Release notes

These release notes summarize the changes in current and previous PingAccess Agent for Apache (Windows) updates.

Version history

Version 1.5.2 – September 2021

Agent SDK for C version 1.3

Added an option to override the default `X-Forwarded-Host` header with a specified header.

Version 1.5.1 – April 2021

Agent SDK for C version 1.3

- Fixed an issue that caused large bodies sent through POST preservation in an agent deployment to be corrupted.

Version 1.5 – July 2020

Agent SDK for C version 1.3

- Added agent inventory callback API

Version 1.4.1 – February 2020

Agent SDK for C version 1.2.1

- Fixed a potential security issue

Version 1.4 – July 2019

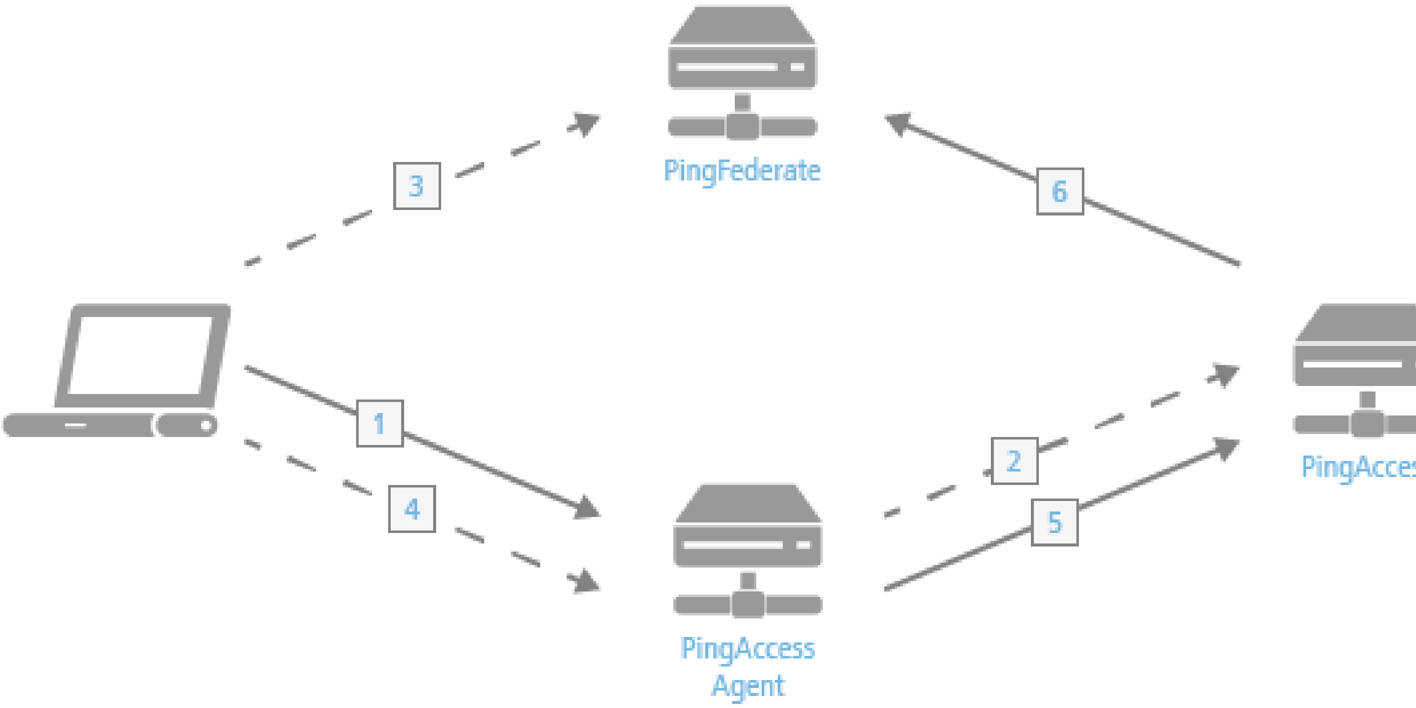
Agent SDK for C version 1.2.0

- Initial release for Apache 2.4 on Windows

> **Note:**
>
> Version is aligned with PingAccess Agent for Apache (RHEL).

PingAccess Agent for IIS

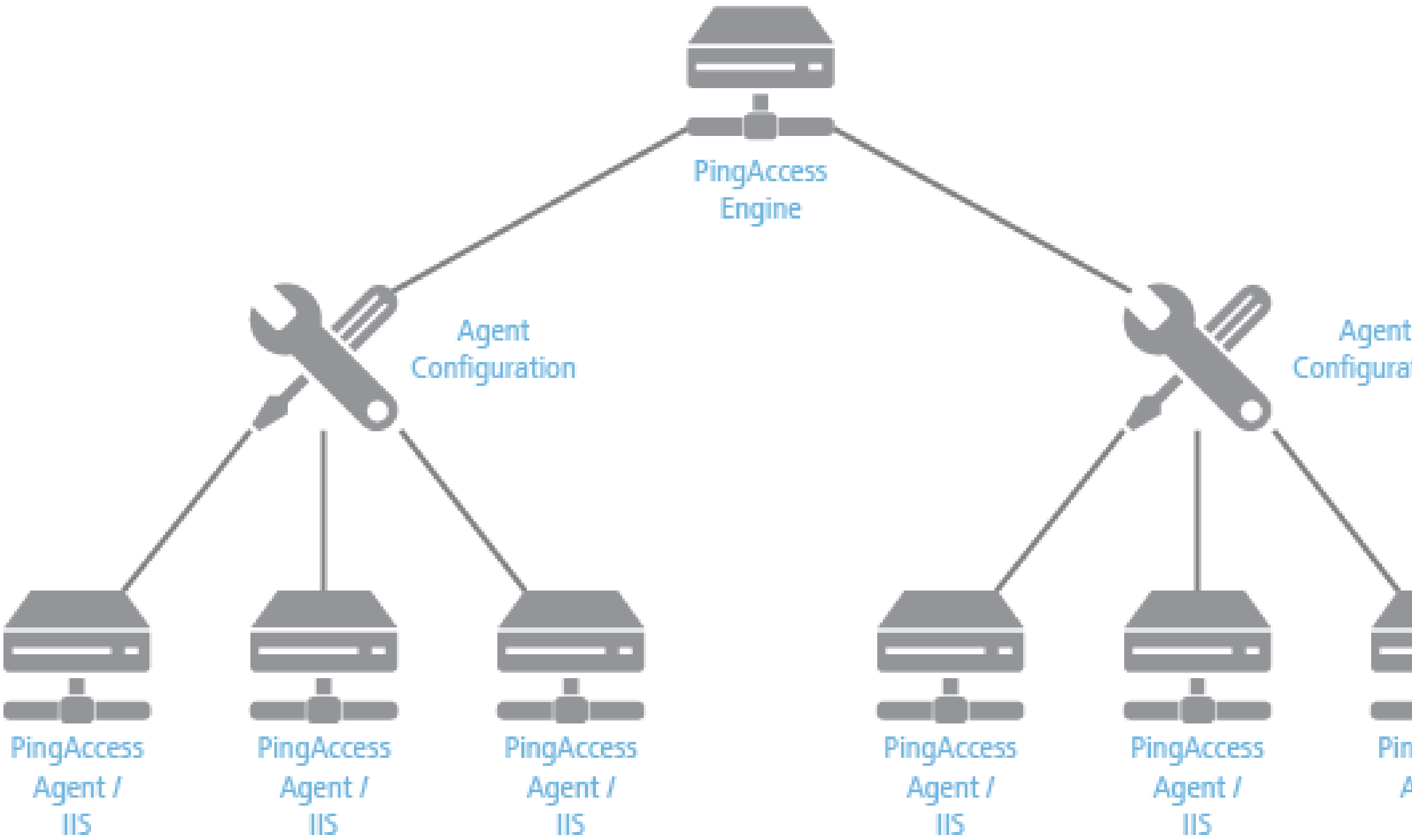
The PingAccess agent for IIS is a Microsoft Internet Information Services module that intercepts requests to the web server's protected resources and evaluates applicable access control policies. These policies are evaluated by either accessing a locally cached policy decision or by querying the PingAccess engine node.



Processing steps

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user logs in, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

Within the PingAccess administrative console, agent nodes are configured with information that allows a PingAccess agent to connect to the engine node to retrieve information about access control policies for resources within that agent's control. An agent configuration has a one-to-many relationship with PingAccess agents, allowing a single agent configuration bootstrap file to be used on multiple web servers within a server farm.



Note:

An agent node is a shared configuration used by one or more agents, rather than a specific agent instance.

Tip:

A problem with the PingAccess IIS agent configuration might cause all applications in an IIS application pool to become unreachable. To maximize availability of unprotected resources, place applications protected by the PingAccess agent in a separate pool.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

IIS agent system requirements

The PingAccess agent for IIS is supported on these platforms.

- Microsoft Internet Information Services (IIS) 10.0 running on Windows Server 2016
- Microsoft Internet Information Services (IIS) 10.0 running on Windows Server 2019
- Microsoft Internet Information Services (IIS) 10.0 running on Windows Server 2022

Note:

Only 64-bit Windows platforms and versions of IIS are supported. 32-bit compatibility mode for IIS is not supported by this agent.

All of the other dependencies required for the agent are included in the `.msi` installation package.

As with any system that is reachable from the internet, the server should be properly hardened.

Installing on IIS

Install a PingAccess agent on an Internet Information Services (IIS) system.

Before you begin

Shut down any running programs, including the Windows Event Viewer. Running applications might cause the installation to fail.

> **Important:**
>
> If the system is running application pools in 32-bit compatibility mode, review the *Troubleshooting* for information about preventing a known issue.

Steps

1. Extract the PingAccess agent for IIS `.zip` file.

   > **Note:**
   >
   > The installer cannot be run from inside the `.zip` file. You must first extract it.

2. Run the `pingaccess-agent-iis.msi` installer.
   a. Click **Next**.
   b. Optional: Specify a new destination folder, and then click **Next**.
   c. Click **Install**.
   d. Click **Finish**.
3. Sign on to the PingAccess Console.
4. Click **Applications** and then go to **Agents**.
5. Edit a configured agent.

   If the agent has not yet been created, follow the procedure in the *PingAccess User Interface Reference Guide*.
6. In the shared secret, click the **Download** icon to download the configuration.

   The configuration file is named `<agentname>_agent.properties`.
7. To create the `C:\Program Files\Ping Identity\PingAccess Agent for IIS\agent.properties` file on the agent system, copy and rename the `<agentname>_agent.properties` file. .

   > **Note:**
   >
   > If you changed the destination folder for the agent, update the file path for the `agent.properties` file to match the new location.

8. Restart Microsoft IIS Server on the system:
   a. Launch IIS Manager.
   b. Navigate to the web server node in the Connections tree.
   c. In the **Actions** pane, click **Restart**.

> **Tip**:
>
> You can restart the IIS service by running the command `iisreset /restart`.

Manually Installing on IIS

Manually install a PingAccess agent for Internet Information Services (IIS), or if the installation failed, manually complete a partial installation.

About this task

> **Important**:
>
> For information about preventing a known issue on systems running application pools in 32-bit compatibility mode, see *Troubleshooting*.

> **Tip**:
>
> If you use this procedure due to an installation problem, open a support ticket so the underlying issue can be addressed.

Steps

1. Stop Microsoft IIS:
   a. Run the command `net stop w3svc`.
   b. Run the command `net stop was`.
2. Extract the `pingaccess-agent-iis.msi` installer file from the PingAccess IIS Agent Distribution `pingaccess-agent-iis-x.x.x.zip` file.
3. Extract the MSI installer file's contents.

```
C:\Windows\System32\msiexec /a <full path to pingaccess-agent-iis.msi> /
qb TARGETDIR=<destination path>
```

> **Note:**
>
> From this step on, this procedure will refer to the target directory as <*TARGETDIR*>. The files of interest are in `<TARGETDIR>\PFiles`.

4. Copy `TARGETDIR\PFiles\Ping Identity\` and its contents to `C:\Program Files\`.
5. Download the *Microsoft Visual C++ Redistributable* and install it.
6. Add the PingAccess agent module configuration schema to IIS:
   a. `cd C:\<TARGETDIR>\PFiles\inetsrv\config\schema\`
   b. `copy paa_schema.xml C:\Windows\System32\inetsrv\config\schema\`

7. Edit `C:\Windows\System32\inetsrv\config\applicationHost.config` and make the following changes:
   a. Add `sectionGroup` to the container with `name=system.webServer` under `configSections`.

      Example:

      ```
      <section name="paa" overrideModeDefault="Deny"
       allowDefinition="AppHostOnly" allowLocation="false" />
      ```
   b. Add the following XML block to the `<system.webServer>` element.

      Example:

      ```
      <paa>
       <paaCertificateDir value="C:\Program Files\Ping Identity\PingAccess
       Agent for IIS\certs\" />
       <paaPropertyFiles>
        <file path="C:\Program Files\Ping Identity\PingAccess Agent for IIS
      \agent.properties" />
       </paaPropertyFiles>
      </paa>
      ```
8. Open IIS Manager and go to **Management # Configuration Editor**.
9. Select the `system.webServer/paa` section and validate that the paths added to `applicationHost.config` have the following values:

   paaCertificateDir

      `C:\Program Files\Ping Identity\PingAccess Agent for IIS\certs\`

   paaPropertyFiles

      (Count=1)

> **Note:**
>
> If the changes are not present, ensure that you are using a 64-bit text editor. When using a 32-bit text editor, changes to this file will be transparently saved to `%SYSTEMROOT%\SysWOW64\inetsrv\applicationHost.config`.

10. Verify that the `C:\Program Files\Ping Identity\PingAccess Agent for IIS\certs` folder has been created.
11. Change the permissions of `C:\Program Files\Ping Identity\PingAccess Agent for IIS\certs` to include read and write permissions for `IIS_IUSRS`.

    You might need to manually search for this user when modifying the permissions.
12. Register the PingAccess agent logging publisher:
    a. Run the following command.

       ```
       C:\Windows\System32\wevtutil im paa-event-logging.xml /rf:"C:\Program
        Files\Ping Identity\PingAccess Agent for IIS\paa-iis-module.dll" /
       mf:"C:\Program Files\Ping Identity\PingAccess Agent for IIS\paa-iis-
       module.dll"
       ```
    b. Run the following three commands to ensure the logging publisher installed successfully.

       ```
       C:\Windows\System32\wevtutil gl PingAccess-Agent/Admin
       C:\Windows\System32\wevtutil gl PingAccess-Agent/Analytic
       C:\Windows\System32\wevtutil gl PingAccess-Agent/Debug
       ```

13. Register the agent module with IIS:
    a. Open IIS Manager, then select the web server the agent is being added to.
    b. Click **Modules**.
    c. Click **Configure Native Modules**.
    d. Click **Register** and enter the following information.

Name	`PingAccessAgentModule`
Path	`C:\Program Files\Ping Identity\PingAccess Agent for IIS\paa-iis-module.dll`

    e. Click **OK**.
    f. Click **OK**.
    g. Execute the command `iisreset /restart`.
14. After IIS has restarted, use IIS Manager to ensure that the Default Application Pool has started.

> **Note:**
>
> If the Default Application Pool has not started, you will see 500 series server errors when navigating to a site protected by the agent.

15. Continue the installation from *Step 3* of the installation procedure.

Result

The PingAccess agent writes log information to the PingAccess-Agent logs in the Event Viewer Application and Services logs. Check these logs for any errors if the agent module does not appear to have loaded.

Uninstalling on IIS
Remove the PingAccess agent from an Internet Information Services (IIS) system.

About this task

> **Note:**
>
> Alternatively, remove the PingAccess agent from an IIS system using the **Add/Remove Programs** option in the Windows control panel.

Steps

1. Run the `pingaccess-agent-iis.msi` installer.
   Result: The installer displays the available workflows.
2. Select the **Remove** workflow.

Configuration

Manage the PingAccess agent for Internet Information Services (IIS) configuration through the IIS Manager application.

During the installation of the agent, a configuration schema extension is added to the `system.webServer` section. This schema extension adds the two configuration options defined in the following table.

Parameter	Definition	Default Value
`PaaCertificateDir`	String value containing the path to the certificates extracted from the `.properties` files.	`C:\Program Files\Ping Identity\PingAccess Agent for IIS\certs.properties`
`PaaPropertyFiles`	List of `.properties` files which store configuration data used to connect the agent to the PingAccess engine nodes the agent will communicate with.	`C:\Program Files\Ping Identity\PingAccess Agent for IIS\agent.properties`

Note:

Do not make any changes to these configuration parameters if the steps in the *Installation* section were followed.

The configured `agent.properties` files can contain the following parameters.

Parameter	Definition	Default Value
`agent.engine.configuration.scheme`	The URL scheme used to connect to the engine node. Valid values are `http` and `https`.	`https`
`agent.engine.configuration.host`	The PingAccess hostname.	The value in the Agent Node's **PingAccess Host** field.
`agent.engine.configuration.port`	The port the agent connects to on the PingAccess host. This value is defined in the PingAccess `run.properties` file.	Defined in the PingAccess Admin UI
`agent.engine.configuration.username`	The unique agent name that identifies the agent in PingAccess.	Defined in the PingAccess Admin UI
`agent.engine.configuration.checkCertRevocation`	Determines whether the agent performs certificate revocation list checking against the server certificate used by the engine nodes or by a load balancer in front of the engine nodes. A value of `1` enables CRL checking, while a value of `0` disables CRL checking.	Not present by default. Treated as 1 when not specified.
`agent.engine.configuration.shared.secret`	The password used to authenticate the agent to the engine.	Defined in the PingAccess Admin UI

Parameter	Definition	Default Value
`agent.engine.configuration.bootstrap.truststore`	The base64-encoded public certificate used to establish HTTPS trust by the agent to the PingAccess engine. **Note:** If you are having difficulty connecting an agent to the PingAccess engine, verify that the Agent Trusted Certificate has been configured correctly in *Agent Management*.	Generated by PingAccess
`agent.engine.configuration.maxConnections`	The number of connections a single web server worker process maintains to the PingAccess engine defined in the `agent.engine.configuration.host parameter`.	10
`agent.engine.configuration.timeout`	The maximum time, in milliseconds, a request to PingAccess can take from the agent. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
`agent.engine.configuration.connectTimeout`	The maximum time, in milliseconds, the agent can take to connect to the PingAccess engine. If this time is exceeded, the client will receive a generic `500 Server Error` response.	30000
`agent.cache.missInitialTimeout`	The maximum time, in milliseconds, a web server worker process waits for a response to a policy cache request sent to other web server worker processes.	5
`agent.cache.broker.publisherPort`	The network port web server processes use to publish policy cache requests to other web server worker processes. This port is bound to the localhost network only.	3031

Parameter	Definition	Default Value
`agent.cache.broker.subscriberPort`	The network port web server processes use to receive policy cache requests from other web server worker processes. This port is bound to the localhost network only.	3032
`agent.cache.maxTokens`	The maximum number of tokens stored in the policy cache for a single web server worker process. A value of 0 means there is no maximum.	0

Parameter	Definition	Default Value
`agent.cache.disabled`	Determines whether caching of policy decisions is enabled or disabled. A value of `1` disables caching, forcing the agent to communicate with the PingAccess host any time a policy decision needs to be made. This option might be desired when using PingAccess 3.1 or earlier with the following rule types: ▪ Groovy Script Rule ▪ HTTP Request Rule ▪ Network Range Rule ▪ Time Range Rule **Note:** PingAccess 3.2 does require the cache be disabled in order to process these rules correctly from an agent. This might also be desirable for custom rules created using the PingAccess SDK that involve data that changes with every request within a resource and session. **Warning:** Disabling caching has a significant impact on the scalability of the PingAccess Policy servers, as every rule evaluation is processed by the Policy Server. This option should only be used as a last resort because of the performance penalty.	0

Parameter	Definition	Default Value
`agent.cache.type`	Controls the type of policy cache used by the agent. There are three valid values for this property: AUTO The AUTO cache type determines the appropriate cache to use based on the number of worker processes. If the number of worker processes is 1, the agent uses the STANDALONE cache. If the number of worker processes is 2 or more, the agent uses the ZMQ cache. STANDALONE The STANDALONE cache type does not share policy cache entries across worker processes ZMQ The ZMQ cache type allows the agent to share policy cache entries across all worker processes using ZeroMQ for inter-process communication.	AUTO

<table>
<tr><th>Parameter</th><th>Definition</th><th>Default Value</th></tr>
<tr><td><code>agent.send.inventory</code></td><td>Determines whether the <code>vnd-pi-agent</code> agent inventory header is sent along with each request to the PingAccess policy server.
This header contains the following fields:
<code>v</code>
The PingAccess agent version.
<code>t</code>
The type of PingAccess agent retrieved from the server variable on the IIS context, which returns a string such as <code>Microsoft-IIS/7.5</code>.
<code>h</code>
The hostname of the PingAccess agent retrieved from the IIS context.
For more information, see Agent inventory logging on page 117.</td><td><code>true</code></td></tr>
<tr><td><code>agent.inventory</code></td><td>Specifies additional values to include in the <code>vnd-pi-agent</code> agent inventory header.
The following syntax is used.
<code>agent.inventory=exampleheader=TEST;exampleheader2=TEST2;</code>
Note:
The specified header fields are case-sensitive.</td><td>Not present by default.</td></tr>
</table>

You can add comments to the `agent.properties` files if necessary. Lines beginning with the `#` or `!` characters are ignored by the agent.

Changes to the `agent.properties` file require a restart of the web server.

Tip:

See the *Performance tuning guide* for discussion on improving agent performance.

Log Configuration

The PingAccess agent for Internet Information Services (IIS) installer registers the PingAccess-Agent Windows Event Log publisher when the agent is installed. This makes PingAccess agent log information available in the Windows Event Viewer in the `Applications and Services Logs\PingAccess-Agent` folder.

The PingAccess agent for IIS logs information to one of three potential logs.

Log	Description
Admin	Contains general information messages about the module and any error messages that occur during operation. This should be a low-volume log. This log is enabled and visible by default.
Analytic	Contains monitoring, performance, and timing information. Entries in this log are useful for diagnosing performance issues. Information about the source of a policy decision for each request. This log is enabled and hidden by default.
Debug	Contains debug-level information about the module's operation. This log should only be enabled at the request of Ping Identity support technicians, as it is ahigh volume log and might contain sensitive identity and token information. This log is disabled and hidden by default.

Note:

To view and enable a log:

1. To display all of the logs, go to **View** # **Show Analytic and Debug Logs** in Event Viewer.
2. In the console tree, select the log that you want to enable.
3. In the Actions pane, click **Enable Log** .

In addition to using the Windows Event Viewer, PingAccess Agent log information is accessible using the PowerShell cmdlet `get-winevent`. For example, in a PowerShell session, the content of these logs can be retrieved using this command.

```
PS> get-winevent -logname PingAccess-Agent/Admin,PingAccess-Agent/
Debug,PingAccess-Agent/Analytic -Oldest
```

Rotating a CA

Rotate the certificate authority (CA) used by an agent while minimizing the impact to agent communications.

Steps

1. On the agent web server, update the `agent.properties` file to add the new CA certificate.
   a. Concatenate the old and new CA certificates in PEM encoding format into a new file.
   b. Encode the contents of the file to Base64.
   c. Open the `agent.properties` file and set the value of the `agent.engine.configuration.bootstrap.truststore` line to the encoded content.

      Example:

      ```
      agent.engine.configuration.bootstrap.truststore=<Encoded_content>
      ```
2. Restart the agent web server.

3. Update the configuration to use a new server certificate signed by the new CA for the agent HTTPS listener.
   a. Identify a key pair to use. If necessary, create a new key pair.

      For more information, see *Generating new key pairs* on page 313.
   b. Generate a CSR for that key pair.

      For more information, see *Generating certificate signing requests* on page 314.
   c. Submit that CSR to the new CA to get a new signed certificate.
   d. Import the CSR response (the new certificate) into .

      For more information, see *Importing certificates* on page 309.
   e. Assign the key pair to the agent HTTPS listener.

      For more information, see *Assigning key pairs to HTTPS listeners* on page 315.

Troubleshooting

This table lists some potential problems and resolutions you might encounter with the PingAccess agent for Internet Information Services (IIS).

Issue	Resolution
The Installer fails to successfully install the agent.	Use the steps listed in the *Manual Installation* procedure to validate the installation and to manually complete the installation. Review the MSI installer log file for the installation to identify errors. The log file is stored in the Temp directory `C:\Users\<username>\AppData\Local\Temp` by default. The filename is not fixed, so you must locate the most recent MSI*.log file. Direct the installer to log to a specific file by launching the installer using this command. `msiexec /l*v "<location>/paAgentInstaller.log" /i "pingaccess-agent-iis.msi"`
The Uninstall program fails to successfully remove the agent.	Follow the steps in the *Manual Removal* to remove the configuration for the PingAccess agent for IIS.
The PingAccess-Agent/Admin log contains the error `SSL peer certificate or SSH remote key was not OK(0)`	It is likely that the hostname for the PingAccess engine being accessed does not match the hostname in the certificate used by the agent. Verify the certificate configuration, and if necessary, recreate the certificate for the agent HTTPS Listener and recreate the agent configuration. See *PingAccess User Interface Reference Guide* in the PingAccess documentation for more information.
500 series errors accessing protected resources	This can indicate that the PingAccess agent failed to load, or that the Default Application Pool is stopped. Correct the issue that's causing the module load failure, and then restart the Default Application Pool. One potential cause of this is that the `agent.properties` file cannot be found or loaded. Ensure that this file is copied over as described in *Step 6* of the installation procedure.

Issue	Resolution

32-bit application pools crashing	This indicates that IIS attempted to load the PingAccess 64-bit agent module in an application container that is running in 32-bit mode. Modify the `applicationHost.config` file's `PingAccessAgentModule` directive in the `globalModules` section to add the following `preCondition` directive.
	
	`preCondition="integratedMode, bitness64"`
	
	For example:
	
	`<globalModules>`
	`<add name="PingAccessAgentModule"`
	`     image="c:\Program Files\Ping Identity\PingAccess Agent for IIS\paa-iis-module.dll"`
	`     preCondition="integratedMode, bitness64" />`
	`</globalModules>`

Agent does not start. Application log contains this error: `The Module name PingAccessAgentModule path (...)\paa-iis-module.dll returned an error from registration. The data is the error.`	This can indicate a corrupted or invalid `agent.properties` file. Export the `agent.properties` file from the administrative console and replace the existing file on the IIS system with the new version, as described in *Installing on IIS* on page 406.

<table>
<tr><th>Issue</th><th>Resolution</th></tr>
<tr><td>Agent receives an unknown protocol error when attempting to contact the administrative node</td><td>
This can indicate that the operating system is using SHA-1 for encryption. This protocol is no longer supported by default in PingAccess.
We recommend switching to sha256. If you cannot switch to sha256, you can re-enable SHA-1:

Open the <code>run.properties</code> file.
Add TLSv1 to the protocol list. For example:
<pre><code>tls.default.protocols=TLSv1, TLSv1.1, TLSv1.2,
 TLSv1.3</code></pre>
Add the SHA entries to the cipher suites list. For example:
<pre><code>tls.default.cipherSuites =
 TLS_CHACHA20_POLY1305_SHA256,\
                          TLS_AES_256_GCM_SHA384,\
                          TLS_AES_128_GCM_SHA256,\

 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,\

 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,\

 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,\

 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,\

 TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,\

 TLS_EMPTY_RENEGOTIATION_INFO_SCSV, \

 TLS_RSA_WITH_AES_128_CBC_SHA, \

 TLS_DHE_RSA_WITH_AES_128_CBC_SHA, \

 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, \

 TLS_ECDH_RSA_WITH_AES_128_CBC_SHA, \

 TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA</code></pre>

</td></tr>
</table>

Validating the IIS Configuration

Verify that an Internet Information Services (IIS) agent has installed successfully.

About this task

For a minimal configuration of the PingAccess agent for IIS, the following steps outline the changes made during installation that might need to be verified if the installer fails. Use this procedure as a guide for what to check if the installation did not complete successfully.

Steps

1. Stop Microsoft IIS.
   a. Run the command `net stop w3svc`.
   b. Run the command `net stop was`.

2. Edit `C:\Windows\System32\inetsrv\config\applicationHost.config` and add the following line to the `sectionGroup` container with `name=system.webServer` under `configSections`.

   Example:

```
<section name="paa" overrideModeDefault="Deny"
 allowDefinition="AppHostOnly" allowLocation="false" />
```

3. Add the following XML block to the `<system.webServer>` element in `C:\Windows\System32\inetsrv\config\applicationHost.config`.

   Example:

```
<paa>
 <paaCertificateDir value="C:\Program Files\Ping Identity\PingAccess
 Agent for IIS\certs\" />
 <paaPropertyFiles>
  <file path="C:\Program Files\Ping Identity\PingAccess Agent for IIS
\agent.properties" />
 </paaPropertyFiles>
</paa>
```

4. Open IIS Manager and go to **Management # Configuration Editor**.
5. Select the `system.webServer/paa` section and validate that the paths added to `applicationHost.config` are correct.
6. Register the agent module with IIS.
   a. Open IIS Manager, then select the web server the agent is being added to.
   b. Click **Modules**.
   c. Click **Configure Native Modules**.
   d. Click **Register** and enter the following information.

Name	`PingAccessAgentModule`
Path	`C:\Program Files\Ping Identity\PingAccess Agent for IIS\paa-iis-module.dll`

   e. Click **OK**.
   f. Click **OK**.
   g. Run the command `iisreset /restart`.

Manually removing agents on IIS

Manually remove the agent if an attempt to remove the agent from a system fails.

Steps

1. Stop Microsoft IIS.
   a. Run the command `net stop w3svc`.
   b. Run the command `net stop was`.
2. Edit `C:\Windows\System32\inetsrv\config\applicationHost.config` and remove the following line from the `sectionGroup` container with `name=system.webServer` under `configSections`.

   Example:

```
<section name="paa" overrideModeDefault="Deny"
 allowDefinition="AppHostOnly" allowLocation="false" />
```

3. Remove the following XML block from the `<system.webServer>` element in `C:\Windows\System32\inetsrv\config\applicationHost.config`.

   Example:

```
<paa>
 <paaCertificateDir value="C:\Program Files\Ping Identity\PingAccess
 Agent for IIS\certs\" />
 <paaPropertyFiles>
  <file path="C:\Program Files\Ping Identity\PingAccess Agent for IIS
\agent.properties" />
 </paaPropertyFiles>
</paa>
```

4. Open IIS Manager and go to **Management** # **Configuration Editor**.
5. Select the `system.webServer/paa` section and validate that the paths were properly removed from `applicationHost.config`.
6. Deregister the agent module with IIS.
   a. Open IIS Manager, and then select the web server from which the agent is being removed.
   b. Click **Modules**.
   c. Click **Configure Native Modules**.
   d. Select the `PingAccessAgentModule` registered module, and then click **Remove**.
   e. Click **OK**.
   f. Run the command `iisreset /restart`.

Release Notes

These release notes summarize the changes in current and previous PingAccess agent for Internet Information Services (IIS) updates.

Version History

Version 1.4.4 – July 2021

Agent SDK for C version 1.3

- Added agent inventory response

Version 1.4.3 – December 2020

Agent SDK for C version 1.3

- Updated the agent to only disable IIS caching when the agent modifies the response. This preserves performance while mitigating an IIS session swapping vulnerability.

Version 1.4.2 – July 2020

Agent SDK for C version 1.3

- Fixed an issue that caused intermittent application pool crashes.

Version 1.4.1 – February 2020

Agent SDK for C version 1.2.1

- Fixed a potential security issue

Version 1.4 – June 2019

Agent SDK for C version 1.2.0

- Added ability to set policy caching mechanism using a property in the `agent.properties` file
- Added ability to enable or disable agent processing for a request based on a note field

- Fixed a potential security issue

Version 1.3.2 – November 2018

Fixed a potential security issue

Version 1.3 – January 2017

- Added support for IIS 10 on Windows Server 2016
- Updated to 1.1.1 of the PingAccess Agent SDK for C
- Resolved issue with IIS Preload Enabled setting

Version 1.2.1 – November 2016

- Added support for the “Preload Enabled” setting in IIS
- Security enhancements

Version 1.2 – August 2016

Updated to 1.0.1 of the PingAccess Agent SDK for C

Version 1.1.2 – February 2016

Addressed issue with custom request headers not being set when URL contains query string parameters

Version 1.1.1 – September 2015

Addressed compatibility with the IIS plugin for WebSphere

Version 1.1 – December 2014

- Added Support for Microsoft Internet Information Services (IIS) 7.0 running on Windows Server 2008
- Added Support for Microsoft Internet Information Services (IIS) 7.5 running on Windows Server 2008 R2
- Added Support for Microsoft Internet Information Services (IIS) 8.0 running on Windows Server 2012 Datacenter Edition
- Corrected a potential security issue related to caching (SECBL007). This security bulletin is available in the Ping Identity Support Portal (*https://support.pingidentity.com/s/*)

Version 1.0 – July 2014

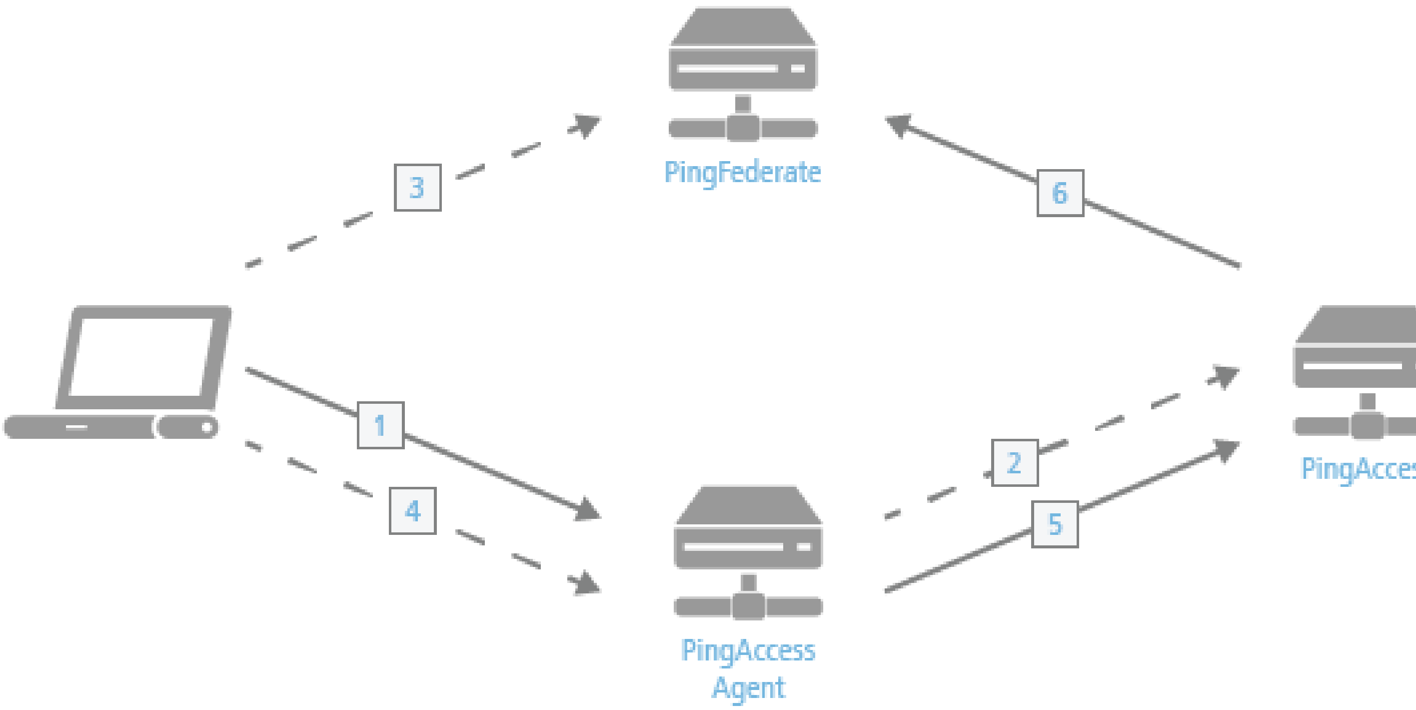
Initial Release

PingAccess Agent for NGINX

The PingAccess agent for NGINX is an NGINX module that intercepts requests to the web server's protected resources and evaluates applicable access control policies. These policies are evaluated by either accessing a locally cached policy decision or by querying the PingAccess engine node.

Note:

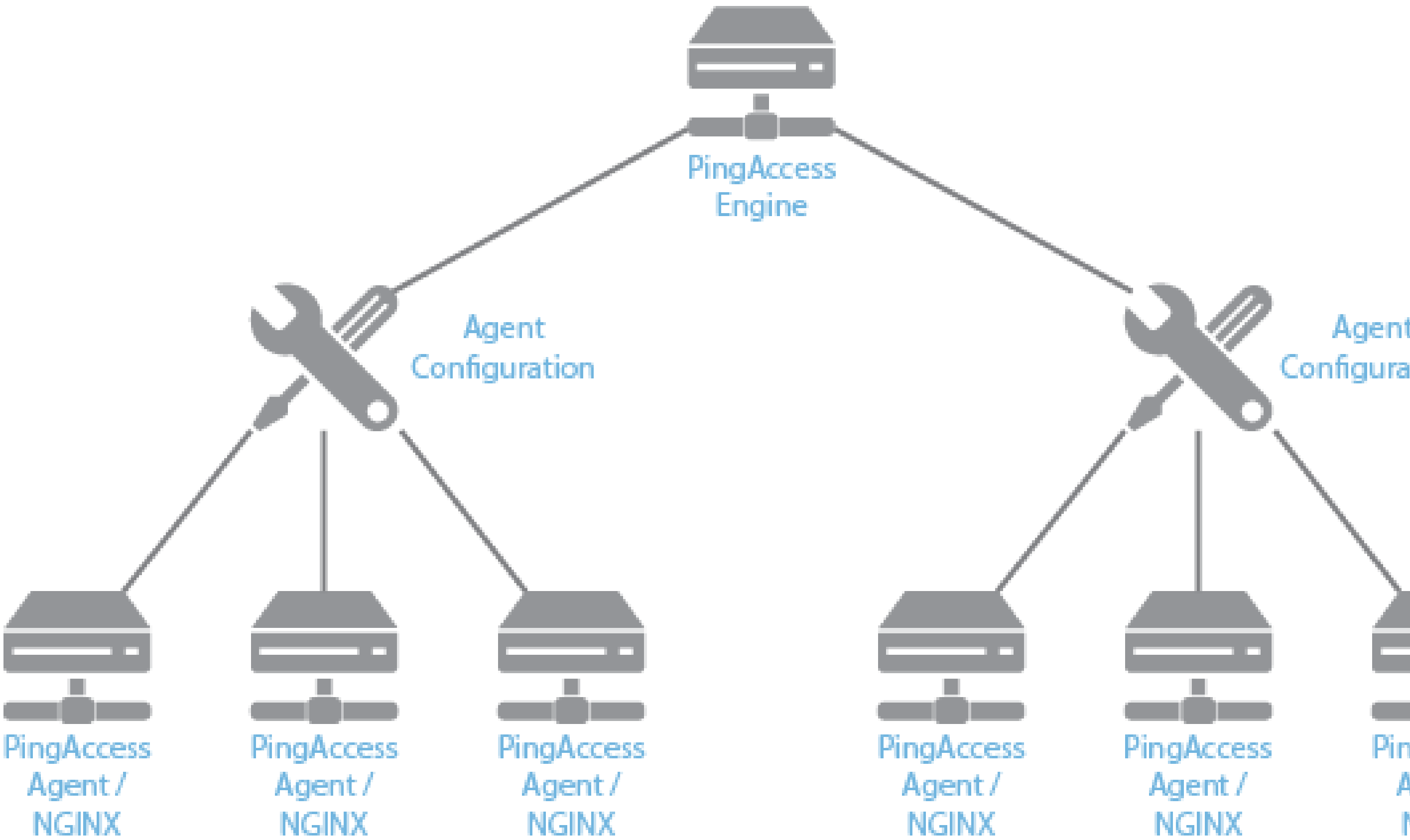
Download the PingAccess agent for NGINX on the *PingAccess Downloads* page.



When a PingAccess agent is added to the policy decision process:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

Within the PingAccess administrative console, agent nodes are configured with information that allows a PingAccess agent to connect to the engine node to retrieve information about access control policies for resources within that agent's control. An agent configuration has a one-to-many relationship with PingAccess agents, allowing a single agent configuration bootstrap file to be used on multiple web servers within a server farm.



An agent node is a shared configuration used by one or more agents, rather than a specific agent instance.

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

NGINX agent system requirements

The PingAccess agent for NGINX is supported on these platforms.

Operating System	Compatible NGINX Versions
Red Hat Enterprise Linux (RHEL) Server 7 (x86_64)	▪ NGINX Plus R26 ▪ NGINX Plus R27 or NGINX OSS 1.21.6 ▪ NGINX Plus R28 or NGINX OSS 1.23.2 ▪ NGINX Plus R29 or NGINX OSS 1.23.4 ▪ NGINX Plus R30 or NGINX OSS 1.25.1
Red Hat Enterprise Linux (RHEL) Server 8 (x86_64)	▪ NGINX Plus R26 ▪ NGINX Plus R27 or NGINX OSS 1.21.6 ▪ NGINX Plus R28 or NGINX OSS 1.23.2 ▪ NGINX Plus R29 or NGINX OSS 1.23.4 ▪ NGINX Plus R30 or NGINX OSS 1.25.1
Red Hat Enterprise Linux (RHEL) Server 9 (x86_64)	▪ NGINX Plus R27 or NGINX OSS 1.21.6 ▪ NGINX Plus R28 or NGINX OSS 1.23.2 ▪ NGINX Plus R29 or NGINX OSS 1.23.4 ▪ NGINX Plus R30 or NGINX OSS 1.25.1

Installing on NGINX

Install a PingAccess agent on an NGINX system.

Before you begin

This procedure makes the following assumptions:

- The PingAccess NGINX agent zip content is extracted to the `$PINGACCESS_AGENT_NGINX` folder.

  > **Note:**
  > Amazon Linux 2 systems use the Red Hat Enterprise Linux 7 download bundles.

- The NGINX installation is assumed to live at $NGINX. In the steps in this procedure, modify the paths specified based on where your NGINX installation and configuration files are located.
- You have downloaded the installation package from the *PingAccess Downloads* page.

About this task

To install the PingAccess agent for NGINX, perform the following steps:

> **Note:**
>
> The agent RPM has required dependencies that might be available through standard repositories. If these dependencies are not available in your Linux version, you can install them using the included `libpgm-5_2-0-5.2.122-32.1.x86_64.rpm`, `libsodium18-1.0.11-1.1.x86_64.rpm` and `libzmq5-4.3.1-23.6.x86_64.rpm` packages.

Steps

1. Install the NGINX module:

   ```
   yum install pingaccess-agent-nginx-*.rpm lib*.rpm
   ```

2. Sign on to the PingAccess console.
3. Click **Applications** and then go to **Agents**.
4. Edit a configured agent.

   If the agent has not yet been created, see the **Agents** section of the *PingAccess User Interface Reference Guide*.
5. In the shared secret, click the **Download** icon to download the agent properties file.
6. Copy the agent properties file to `$NGINX/paa/agent.properties`.
7. To load the PingAccess agent for NGINX module, add the following directive to the NGINX configuration file, `$NGINX/nginx.conf`.

   ```
   load_module modules/ngx_http_paa_module.so;
   ```

8. To configure the PingAccess Agent for NGINX module, add the following directive to the NGINX configuration file, `$NGINX/nginx.conf`, within the http {} block.

```
include $NGINX/paa/http.conf;
```

> **Important**:
>
> In PingAccess *Manage Agents*, **PingAccess Host** must match the certificate CN or Subject Alternative Name.

9. To enable the PingAccess Agent, modify the following property in the file `$NGINX/paa/http.conf`.

```
paa_enabled on;
```

10. Restart the NGINX server:
    a. To stop the NGINX server, run the following command.
       `sudo systemctl stop nginx`
    b. To start the NGINX server, run the following command.
       `sudo systemctl start nginx`

Uninstalling on NGINX
Remove the agent from an NGINX system.

Steps

- Run the following command.

```
sudo yum remove pingaccess-agent-nginx.x86_64
```

Configuration

The PingAccess agent for NGINX configuration is managed through the `$NGINX/paa/http.conf` and `agent.properties` configuration files.

The `$NGINX/paa/http.conf` file contains the configuration options defined in the following table.

Parameter	Definition	Default Value
paa_property_files	Properties file that stores configuration data used to connect the agent to the PingAccess engine nodes.	`$NGINX/paa/agent.properties`
paa_enabled on|off	Value that turns the agent on or off. This property applies to server blocks within the nginx server to control which server blocks are protected by the agent.	`off`
paa_upstream	Defines the upstream that the PingAccess Agent uses to route policy decision requests to PingAccess policy servers.	`pingaccess-policy-server`

Parameter	Definition	Default Value
paa_upstream_max_response_header_size	Defines the maximum size of the response header, in bytes, that the PingAccess agent can receive from a PingAccess policy server.	4096
paa_thread_pool	Defines the thread pool to use for blocking operations performed by the agent. Currently this only includes policy cache lookup operations when using the ZeroMQ multiprocess policy cache.	default

Note:

You do not have to make any changes to http.conf if the steps in the *Installation* section were followed.

Changes to the `paa_upstream` will impact how the agent communicates with PingAccess. Incorrect changes might lead to a non-functional agent.

The configured `agent.properties` files can contain the following parameters.

The 'upstream pingaccess-policy-server' contains the directive 'pingaccess_servers'. This directive indicates that the servers for the containing upstream are defined by the `agent.properties` file. The agent only allows this directive to be specified for a single upstream.

Parameter	Definition	Default Value
agent.engine.configuration.scheme	The URL scheme used to connect to the engine node. Valid values are http and https.	https
agent.engine.configuration.host	The PingAccess hostname.	The value in the agent node's `PingAccess Host` field.
agent.engine.configuration.port	The port the agent connects to on the PingAccess host. This value is defined in the PingAccess `run.properties` file.	Defined in the PingAccess Admin UI
agent.engine.configuration.username	The unique agent name that identifies the agent in PingAccess.	Defined in the PingAccess Admin UI
agent.engine.configuration.shared.secret	The password used to authenticate the agent to the engine.	Defined in the PingAccess Admin UI

Parameter	Definition	Default Value
`agent.engine.configuration.bootstrap.truststore`	The base64-encoded public certificate used to establish HTTPS trust by the agent to the PingAccess engine. **Note:** If you are having difficulty connecting an agent to the PingAccess engine, verify that the Agent Trusted Certificate has been configured correctly in *Agent Management*.	Generated by PingAccess
`agent.engine.configuration.maxConnections`	The number of connections a single web server worker process maintains to the PingAccess engine defined in the `agent.engine.configuration.host` parameter.	10
`agent.engine.configuration.timeout`	The maximum time (in milliseconds) a request to PingAccess can take from the agent. If this time is exceeded, the client receives a generic `500 Server Error` response.	30000
`agent.engine.configuration.connectTimeout`	The maximum time (in milliseconds) the agent can take to connect to the PingAccess engine. If this time is exceeded, the client receives a generic `500 Server Error` response.	30000
`agent.cache.missInitialTimeout`	The maximum time (in milliseconds) a web server worker process waits for a response to a policy cache request sent to other web server worker processes.	5
`agent.cache.broker.publisherPort`	The network port web server processes use to publish policy cache requests to other web server worker processes. This port is bound to the localhost network only.	3031

Parameter	Definition	Default Value
`agent.cache.broker.subscriberPort`	The network port that web server processes use to receive policy cache requests from other web server worker processes. This port is bound to the localhost network only.	3032
`agent.cache.maxTokens`	The maximum number of tokens stored in the policy cache for a single web server worker process. A value of `0` means there is no maximum.	0
`agent.cache.disabled`	Determines whether caching of policy decisions is enabled or disabled. A value of `1` disables caching, forcing the agent to communicate with the PingAccess host any time a policy decision needs to be made. This option might be desired when using PingAccess 3.1 or earlier with the following rule types: ▪ Groovy script Rule ▪ HTTP request Rule ▪ Network range Rule ▪ Time range Rule PingAccess 3.2 and later does not require the cache be disabled in order to process these rules correctly from an agent. **Warning:** Disabling caching has a significant impact on the scalability of the PingAccess policy servers, as every rule evaluation is processed by the policy server. Only use this option as a last resort because of the performance penalty.	0

Parameter	Definition	Default Value
`agent.engine.configuration.failover.hosts`	The hostname and port of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess Host. **Note:** If this parameter is set, the upstream block name in `$NGINX/paa/http.conf` needs to be modified to a name that will be found in the certificate associated with the PingAccess Agent HTTPS Listener. For example, if your PingAccess certificate contains name '`pa.nginx`', set the upstream name to `upstream pa.nginx`.	Defined in the PingAccess Admin UI
`agent.engine.configuration.failover.failedRetryTimeout`	Seconds before trying a failed PingAccess server.	60
`agent.engine.configuration.failover.maxRetries`	The maximum number of retries before considering a PingAccess server unavailable.	2

Parameter	Definition	Default Value
`agent.cache.type`	Controls the type of policy cache used by the agent. There are three valid values for this property: AUTO The AUTO cache type determines the appropriate cache to use based on the number of worker processes. If the number of worker processes is 1, the agent uses the STANDALONE cache. If the number of worker processes is 2 or more, the agent uses the ZMQ cache. STANDALONE The STANDALONE cache type does not share policy cache entries across worker processes. ZMQ The ZMQ cache type allows the agent to share policy cache entries across all worker processes using ZeroMQ for inter-process communication.	AUTO

Parameter	Definition	Default Value

`agent.send.inventory`	Determines whether the `vnd-pi-agent` agent inventory header is sent along with each request to the PingAccess policy server.
	This header contains the following fields:
	`v`
	The PingAccess agent version.
	`t`
	The type of PingAccess agent retrieved using the `NGINX_VER_BUILD` macro.
	`h`
	The hostname of the PingAccess agent retrieved using the Server Name directive.
	For more information, see *Agent inventory logging* on page 117.
		`true`

`agent.inventory`	Specifies additional values to include in the `vnd-pi-agent` agent inventory header. The following syntax is used. `agent.inventory=exampleheader=TEST;exampleheader2=TEST2;` **Note:** The specified header fields are case-sensitive.	Not present by default.

Add comments to the `agent.properties` files if necessary. Lines beginning with the `#` or `!` characters are ignored by the agent.

Changes to the `agent.properties` file require a restart of the web server.

Tip:

See **Agent Tuning** in the PingAccess *Performance tuning reference guide* for a discussion on improving agent performance.

Rotating a CA

Rotate the certificate authority (CA) used by an agent while minimizing the impact to agent communications.

Steps

1. On the agent web server, update the `agent.properties` file to add the new CA certificate.
   a. Concatenate the old and new CA certificates in PEM encoding format into a new file.
   b. Encode the contents of the file to Base64.
   c. Open the `agent.properties` file and set the value of the `agent.engine.configuration.bootstrap.truststore` line to the encoded content.

      Example:

      ```
      agent.engine.configuration.bootstrap.truststore=<Encoded_content>
      ```
2. Restart the agent web server.
3. Update the configuration to use a new server certificate signed by the new CA for the agent HTTPS listener.
   a. Identify a key pair to use. If necessary, create a new key pair.

      For more information, see *Generating new key pairs* on page 313.
   b. Generate a CSR for that key pair.

      For more information, see *Generating certificate signing requests* on page 314.
   c. Submit that CSR to the new CA to get a new signed certificate.
   d. Import the CSR response (the new certificate) into .

      For more information, see *Importing certificates* on page 309.
   e. Assign the key pair to the agent HTTPS listener.

      For more information, see *Assigning key pairs to HTTPS listeners* on page 315.

Release Notes

These release notes summarize the changes in current and previous PingAccess agent for NGINX updates.

Version History

Version 2.1.1 – November 2020

Agent SDK for C version 1.3

- Fixed an issue that caused an invalid memory access and sometimes caused crashes after a request header modification by another NGINX module.

Version 2.1 – July 2020

Agent SDK for C version 1.3

- Added agent inventory callback API

Version 2.0.2 – February 2020

Agent SDK for C version 1.2.1

- Added a configuration property to set the maximum size of the response header that can be received from a PingAccess policy server

Version 2.0.1 – June 2019

Agent SDK for C version 1.2.0

- Fixed a potential security issue

Version 2.0 – February 2019

- The PingAccess Agent for NGINX leverages the built-in, event-driven HTTP stack in NGINX to communicate with PingAccess policy servers. Previously, the agent used its own HTTP client (implemented with libcurl) to communicate with PingAccess policy servers. In certain cases, this architecture lead to poor scalability. By using NGINX's built-in, event-driven HTTP stack, the agent is able to achieve superior scalability over previous versions.
- Fixed a potential security issue.

Version 1.1.1 – July 2017

- Support for starting and stopping NGINX using the **`systemctl`** command
- Resolved issue with SSL connectivity

Version 1.1 – March 2017

Updates to meet NGINX certification requirements

Version 1.0 – January 2017

Initial release

PingAccess Agent Protocol

The PingAccess Agent Protocol (PAAP) is an HTTP-based protocol for communication and interaction between PingAccess and PingAccess agents.

An agent typically sits in front of a web application or other protected resource on the web server or load balancer, such as Apache or Microsoft IIS.

PAAP is HTTP-based and utilizes a few custom status codes and headers. One goal of basing the protocol on HTTP is to enable an agent, which runs in an HTTP environment, to use concepts and code libraries already at its disposal to do its job.

The majority of the responsibilities reside within PingAccess. The intent of this protocol is to make the agent a relatively dumb agent, largely shielded from the configuration and processing details, and to maintain policies centrally in PingAccess. This means that agents do not need to know about the signing and encryption keys used by PingAccess or PingFederate. By following this model, the protocol allows agents and PingAccess to be versioned and upgraded independently of one another.

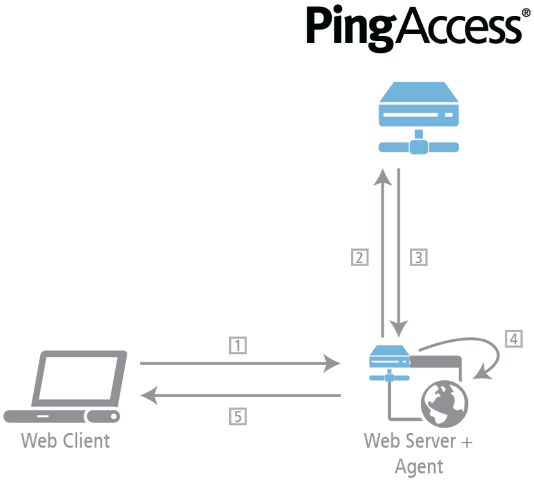
The protocol described here is supported by PingAccess 3.0 and later.

> **Note:**
>
> The prefix “vnd-pi-” was chosen for the PAAP protocol headers defined in this document. In this context, “vnd” indicates a vendor extension, and “pi” represents Ping Identity. Custom status codes were selected after consulting the *Hypertext Transfer Protocol (HTTP) Status Code Registry* with the intention of avoiding any conflicts.

PingAccess agent protocol flow

The PingAccess agent protocol has a set flow by which requests from clients are evaluated and managed.

The PingAccess agent protocol starts with the client, such as a web browser, OAuth client, or any type of HTTP client, making a request for an application resource. The agent sits in front of the resource and intercepts the request. To determine what to do with it, the agent forwards a portion of the request to PingAccess. The response from PingAccess instructs the agent whether to allow the original request, as well as any additional actions to take prior to handing it off to the application. It also includes instructions for actions to perform before sending the corresponding response.



Processing steps

1. Client request
2. Agent request
3. Agent response
4. Modified client request
5. Client response

PAAP client request

The PingAccess Agent Protocol (PAAP) flow begins when the client makes an HTTP 1.1 request to a server where an agent is set up in the filter or interception chain in front of an application or other protected resource.

Example

Unauthenticated user request

This request is coming from an unauthenticated user.

```
GET /application/headers HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: nonce=6424266c-ca9b-4e1f-9fde-d1860bfa2582
```

Example

OIDC Connect flow request

This request is part of the OpenID Connect flow for authentication. The POST body is not shown for brevity.

```
POST /pa/oidc/cb HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Content-Length: 1557
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Origin: https://rhel-test.englab.corp.pingidentity.com:9031
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8
Cookie: nonce=b000c6a2-4a03-4bde-be29-956456cd1d2a
```

Example

Authenticated resource request

This request is an authenticated request for a resource.

```
GET /application/headers HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie:
 PA.post=eyJraWQiOiJhcCIsImFsZyI6IkVTMjU2In0.eyJ6b25laW5mbyI6IkFtZXJpY2FcL05ld19Zb3JrIi
j0BDdLoGeVdqWD35n9ZxFhphEHFe7tfQ6onKAjRdXLR5rtwPBkJHkLaTLD8Yqcsf0izVw
```

PAAP agent request

When the agent intercepts a client request, it makes a correlated request to PingAccess to determine if the request is authorized and what action to take.

The agent request is made by the agent after it receives a client request to determine what actions to take. PingAccess returns the agent response to the agent communicating the action. The agent request effectively mirrors the client request, except for the differences described in the following list.

The Request-Line of the agent request is identical to the Request-Line of the client request. Unless otherwise specified, all headers in the agent request are sent as they appear in the client request.

The message-body of the client request, if any, is omitted from the initial agent request. If PingAccess needs the body, an HTTP 477 response, as defined in the agent response section, is returned. PingAccess will be able to service the vast majority of agent requests without having to see the message body. The body from the initial agent request is omitted because PingAccess will not need it to make a policy decision in most instances, and removing it provides an opportunity for significant performance efficiencies. The

HTTP 477 response mechanism provides PingAccess a way to get the body in the less common cases where it is needed.

HTTP request headers

The following HTTP request headers might require additional agent processing:

Content-Length

The Content-Length header in the agent request will have a value matching the message-body of the request being sent. For the initial agent request, which is sent without the message-body, the Content-Length is 0. A subsequent agent request resulting from a 477 sends a Content-Length that indicates the size of the entity-body of the request, which is the same as in the client request.

vnd-pi-agent

This header lets the agent communicate details of its configuration for optional logging purposes. The agent might specify the custom keys specific to the deployment of the agent, or utilize one or more of the well-known keys.

v

The version of the agent making the request.

t

The type of agent and/or the type of platform where the agent resides.

h

The hostname of the server where the agent resides.

These header examples are considered semantically equivalent.

```
vnd-pi-agent: v="1.0.0", h="apache.example.com", t="Apache
 2.4.41"
```

```
vnd-pi-agent: v="1.0.0", h="apache.example.com"
vnd-pi-agent: t="Apache 2.4.41"
```

```
vnd-pi-agent: v="1.0.0"
vnd-pi-agent: h="apache.example.com"
vnd-pi-agent: t="Apache 2.4.41"
```

The syntax for the **`vnd-pi-agent`** value conforms to a dictionary in this specification, *https://httpwg.org/http-extensions/draft-ietf-httpbis-header-structure.html#dictionary*, where member-values are constrained to be an `sh-string` item.

For more information, see *Agent inventory logging* on page 117.

vnd-pi-expect

This header allows the agent to communicate its needs to PingAccess. `!477` is the only defined value, which tells PingAccess that the agent request contains all the data the agent it is capable of sending regarding the client request. PingAccess should never respond with a 477 to an agent request that has `!477` as the value of the **`vnd-pi-expect`** request header. While *the expect header from RFC 2616* with an expectation-extension could convey the same semantics, Ping Identity elected not to use it due to language in the RFC that suggests intermediaries might, should, or must reject a request using an expectation-extension they don't understand with a 417.

A simple and effective approach for an agent implementation is to send the initial agent request with no body, a content-length of zero, and omitting the **`vnd-pi-expect`** header. The header **`vnd-pi-expect: !477`** is only ever sent when an agent receives a 477 response to its initial request. In other words, the initial agent request never has a **`vnd-pi-expect`** header, while a second agent request in response to an HTTP 477 response always has `!477` as the value for the **`vnd-pi-expect`** header.

PingAccess should never respond to a GET request with a 477, but following this standard allows an agent to handle such an occurrence in an appropriate way.

vnd-pi-v

Indicates the version of PAAP the agent is using. The value is `1.0`

vnd-pi-authz

This header is similar to the *authorization header defined in RFC 2616* but is specifically intended to enable an agent to authenticate itself to PingAccess. The syntax for the “credentials” value of the header is the same as the *section 2.1 of The OAuth 2.0 Authorization Framework: Bearer Token Usage* .

The header looks like this:

```
vnd-pi-authz: Bearer <token>
```

Where `<token>` is a secret shared between PingAccess and the PAA.

In some cases, unrestricted access to the agent protocol at PingAccess might create an information leakage vulnerability. The custom headers returned in the agent response, for example, might reveal internal details of applications or infrastructure that needs to be protected. Potentially worse, the values might reveal content from encrypted Web Access Management (WAM) tokens or reference access tokens. Authenticating PAAs to PingAccess is one means of mitigating the concern.

Authentication is optional. When it is required is at the discretion of PingAccess. Authentication of the agent to PingAccess is intended as a static deployment option, and no challenge response constructs are defined. Failed authentication – missing credentials when required or invalid credentials – is indicative of either a configuration problem or unauthorized access attempt. In such circumstances, PingAccess responds with an HTTP 403 and should include the **`vnd-pi-authz`** header in the response using a quoted string value with human readable information to help troubleshoot and allow for differentiation from an unauthorized end-user. An agent might send the 403 response or a 500 response to the client, depending on which is most appropriate.

vnd-pi-resource-cache

Indicates that for the given host the **`vnd-pi-resource-cache`** and **`vnd-pi-resource-cache-ttl`** headers, defined in the caching part of the next section, are to be returned in the agent response. Generally an agent will include this header when it needs to first establish its resource definition cache for a particular host, or when its current cache is stale or invalid. When an agent request includes the **`vnd-pi-resource-cache`** header, the agent response should include the **`vnd-pi-resource-cache`** and **`vnd-pi-resource-cache-ttl`** headers. An agent must be prepared to handle an agent response that omits those headers. The literal value `requested` can be used by the agent to request the resource cache data.

```
vnd-pi-resource-cache: requested
```

X-Forwarded-For

The **`X-Forwarded-For`** header contains the originating IP address of a client making a request. If no **`X-Forwarded-For`** header is present in the client request, it is added to the agent request with a value indicating the IP address of the client making the connection. If an **`X-Forwarded-For`** header is present in the client request, the IP address of the client is added to the end of the delimited list of IP addresses this header contains when sent in the agent request.

Host and X-Forwarded-Host

The agent sets the **`Host`** header in the agent request as it appeared in the client request, unless it is unable to easily manipulate the **`Host`** header. In the event that it could not modify the **`Host`** header, the **`X-Forwarded-Host`** header contains the original host requested by the client. If **`X-Forwarded-Host`** is already present in the client request, it is sent along unchanged in the agent request.

X-Forwarded-Proto

If **`X-Forwarded-Proto`** is present in the client request, it is sent along unchanged in the agent request. Otherwise, if the scheme used in the client request is different than the agent request (https vs. http), set the **`X-Forwarded-Proto`** header in the agent request to the scheme used in the client request. This header can be omitted if the client request and the agent request use the same scheme.

PingAccess determines the requested resource and constructs self-referential URIs using the contents of the request, including the headers listed above. The scheme is determined from the client's connection and the **`X-Forwarded-Proto`** header, with the latter taking precedence when present. The host and port are determined from the **`Host`** and **`X-Forwarded-Host`** headers, with the latter taking precedence when present.

Example

Policy decision request

This request is a policy decision request for an unauthenticated user.

```
GET /application/headers HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie: nonce=6424266c-ca9b-4e1f-9fde-d1860bfa2582
vnd-pi-v: 1.0
X-Forwarded-For: 172.30.3.248
vnd-pi-resource-cache: requested
X-Forwarded-Proto: http
vnd-pi-authz: Bearer Agent:htZ2W39EfAPLQd8w9cRT6y
```

Example

Agent request without POST body

This request is an agent request, in this case, the OpenID Connect callback in a web session POST login type, without the POST body included.

```
POST /pa/oidc/cb HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Content-Length: 0
```

```
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Origin: https://rhel-test.englab.corp.pingidentity.com:9031
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8
Cookie: nonce=b000c6a2-4a03-4bde-be29-956456cd1d2a vnd-pi-v: 1.0
X-Forwarded-For: 172.30.3.248
vnd-pi-resource-cache: requested
X-Forwarded-Proto: http
vnd-pi-authz:Bearer Agent:htZ2W39EfAPLQd8w9cRT6y
```

Example

Agent request with POST body

This request is a policy decision request with the POST body included. The **`vnd-pi-expect: !477`** header disallows the HTTP 477 agent response to request the body as it is already included.

```
POST /pa/oidc/cb HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Content-Length: 1557
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Origin: https://rhel-test.englab.corp.pingidentity.com:9031
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate
Accept-Language: en-US,en;q=0.8
Cookie: nonce=b000c6a2-4a03-4bde-be29-956456cd1d2a
vnd-pi-v: 1.0
X-Forwarded-For: 172.30.3.248
vnd-pi-resource-cache: requested
X-Forwarded-Proto: http
vnd-pi-expect: !477
vnd-pi-authz: Bearer Agent:htZ2W39EfAPLQd8w9cRT6y
```

Example

Authenticated user request

This request is for a resource by an authenticated user.

```
GET /application/headers HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

```
Cookie:
 PA.post=eyJraWQiOiJhcCIsImFsZyI6IkVTMjU2In0.eyJ6b25laW5mbyI6IkFtZXJpY2FcL05ld19Zb3JrIi
j0BDdLoGeVdqWD35n9ZxFhphEHFe7tfQ6onKAjRdXLR5rtwPBkJHkLaTLD8Yqcsf0izVw
vnd-pi-v: 1.0
X-Forwarded-For: 172.30.3.248
vnd-pi-resource-cache: requested
X-Forwarded-Proto: http
vnd-pi-authz: Bearer Agent:htZ2W39EfAPLQd8w9cRT6y
```

PAAP agent response

When PingAccess receives an agent request, it sends an agent response that includes an authorization decision and any additional actions for the agent to perform on the client request, or it requests additional information from the agent.

Any HTTP status code other than those listed below indicates that the client request was not permitted. In that case, the content of the agent response, including the status-line, the message-body, and all headers not named by the header defined later, is sent back to the client as the content of the client response. This lets PingAccess direct the client, when applicable, to PingFederate for user authentication through redirection or even auto-post form. This also lets PingAccess communicate error conditions and negative authorization decisions to the client with a consistent look and feel.

Custom HTTP status codes

The following custom status codes indicate that the client request is allowed or that additional information is needed in order to make a policy decision.

HTTP 477 Request Body Required

A 477 response indicates that request requires the request body to make a policy decision. The agent should repeat the agent request and include the request body. The subsequent response will include the policy decision and any actions to be taken.

PingAccess does not need the message body to respond to the vast majority of agent requests. This status code allows for optimization of the protocol by not sending unnecessary data by default while providing a way to ask for it when needed. The only case where PingAccess currently requires the body is to evaluate an auto-posted OpenID Connect Authentication Response. This enables the agent to not have to know about or configure any callback redirect_uri locations, but rather push that off to PingAccess. The agent only knows it received a POST request and that PingAccess asked for the message-body in order to process it.

After the agent repeats the agent request with the request body, PingAccess responds to the agent with a generic HTTP response code. Because the only time a 477 is returned by PingAccess is to get the an auto-posted OpenID Connect Authentication Response at the redirect URI, PingAccess never returns a 277 after a 477.

PingAccess should never respond to a GET request with a 477.

HTTP 277 Allowed

The 277 response indicates the client request is authorized and should be allowed to continue. Additional actions to perform on the client request and corresponding client response are indicated by one or more of the headers listed in the HTTP 277 headers section.

HTTP 277 headers

These headers can be included in the agent response if the response code HTTP 277 was used:

vnd-pi-set-req-headers

The value of this header is a comma-delimited list of header names from the agent response to be included as headers for the client request. If any included headers already exist in the client request, the values are overwritten with the values from the corresponding agent response header. If a header is named that is not present in the agent response, it is removed from the client request.

This allows PingAccess to make user attribute information available to the protected application in the headers, and to guard against header injection by the client. In order to guard against malicious header injection by end users, use this mechanism to expose user data to the application. PingAccess should include all header names used in a given context, even if they have no value, so headers supplied by the client using the same names will be ignored.

vnd-pi-append-req-headers

The value of this header is a comma-delimited list of header names from the agent response to add to the client request headers. Any existing named headers already present in the client request are not overwritten, but new headers are added with the values from the agent response. If a header is named that is not present in the agent response, no action is taken for that header name in the client request.

vnd-pi-set-req-vars

The value of this header is a comma-delimited list of header names from the agent response to be set in the client request as request scoped variables or attributes in an appropriate manner for the environment in which the agent resides. Examples might include environment variables in Apache or request attributes in a servlet container.

vnd-pi-sub

This header is used to identify the header that contains the subject or username for the transaction. This typically will be a header also named in **`vnd-pi-set-req-headers`**, as those generally expose user information to the backend application through headers in the modified client request. It might name a header not in the **`vnd-pi-set-req-headers`** list if the agent or environment in which it's deployed needs to know the username or subject in a way that differs from header injection to the client request. The **`vnd-pi-sub`** header should name a single-valued header, but if it names one with multiple values, the agent should use only one. If **`vnd-pi-sub`** names a header that is not present in the agent response, it should be ignored by the agent.

vnd-pi-set-resp-headers

The value of this header is a comma-delimited list of header names from the agent response to be set as headers of the client response. If any of the named headers already exist in the client response, the values are overwritten with the values of those headers from the agent response. If a header is named that is not present in the agent response, it is removed from the client response.

vnd-pi-append-resp-headers

The value of this header is a comma-delimited list of header names from the agent response to be added to the headers of the client response. Any existing named headers already present in the client response are not overwritten, but new headers are added with the values from the agent response. If a header is named that is not present in the agent response, no action is taken for that header name in the client response.

This allows PingAccess to set or reset the PingAccess Web Access Management (WAM) token as a cookie in the user's browser.

In general, for any particular request or response header name, PingAccess should only indicate either a set or an append directive. However, with an agent response, the set directive takes precedence. One way the agent might accomplish that is by applying all append operations first, followed by the set operations.

Common headers

These headers can be used in an agent response of any status code:

vnd-pi-omit-resp-headers

The value of this header is a comma-delimited list of header names from the agent response to omit from the headers in the client response. When present, this list implicitly includes the **`vnd-pi-omit-resp-headers`** header.

Caching

A number of caching directives are aimed at reducing calls from the agent to PingAccess and improving performance.

Resource Definition Caching

When the **`vnd-pi-resource-cache`** request header is present in the agent request, PingAccess includes the following headers in the agent response. This enables the agent to make initial decisions on handling of client requests without having to consult PingAccess directly.

vnd-pi-resource-cache

This is a multi-valued header and, in keeping with *Section 4.2 of RFC 2616*, the values might be comma-delimited, or multiple message-header fields with the same name might be included. The order of the values is significant and, in servicing future requests based on the cache, an agent should evaluate them in the order they were received.

Each value represents a group of resources and some directives about how the agent should handle requests for URIs within that group of resources. The values are made up of multiple parts delimited by semicolons.

path

The path part defines the paths against which requests are matched. The value is one or more space-delimited quoted strings. Each quoted string is a path value, which might contain wildcards using the asterisk (*) character. So, for example, `path="/app/*"` would match any requested path that starts with `/app/`. While `path="/app1/*" "/app2/*"` would match anything under `/app1/` or `/app2/`. Similarly, `path="*.jpg" "*.gif" "*.png"` would match anything ending with those common image file extensions. If no wildcard is present, the values must exactly match.

cs

Indicates if the values in the path are case-sensitive. Valid values are Y and N. If this component is omitted, the default value is Y.

method

Indicates the method or methods against which requests are matched. The value is one or more space-delimited method names such as GET, POST, PUT, etc. If this component is omitted, all methods are allowed and should match for the resource.

kind

Indicates the kind of resource and the general level of access control protection to be applied to it, such as whether access to the resource, and related resources as per the path values, requires an authorization token. Valid values are P, U, or C. The value P, meaning Protected, says that a token is required for access and the token-type and token-name indicate the token of interest so that the token value can be used in caching the response and response headers to service future requests. The value U, meaning Unprotected, indicates that no token is necessary for access and that any future request, within the cache time-to-live, will be allowed. The value C, meaning Consult with PingAccess, means that the agent must always make an agent request to PingAccess to service the client request.

token-type

The token-type part indicates what kind of token was used in making the authorization decision and what token type to use in making future cache queries. Its value is either C or A. A value of C indicates that a cookie was used to make the access control decision and that future requests with the same cookie value for the cookie named in the token-name part can use the cached content. A value of A indicates that an authorization header was used to make the access control decision and that future requests with the same credentials for the authorization scheme named by the token-name directive can use the cached content.

token-name

The token-type says what type of token for which to cache specific user details for a particular token. However, there might be more than one token for a particular type in a request. The token-name value disambiguates that situation by specifying which one to use. The token-name value is either the name of a cookie, for WAM), or the name of an authorization scheme, the Bearer value for OAuth, when token-type value is C or A, respectively.

> **Note:**
>
> When this value is a cookie, the cookie name is case-sensitive, as implied by *RFC 6265*. When this value is the name of an authorization scheme, per *section 1.2 of RFC 2617*, the value is not case-sensitive. When using the token-name header, ensure that the value follows the appropriate case-sensitivity requirements.

If the client request contains more than one token matching the name and type, the value from the first occurrence must be used as the key to lookup or establish a cache for a particular token and other occurrences must be ignored.

The resource-cache list is valid for and scoped to the host in the client request. When building the resource-cache, PingAccess includes resources associated with the virtual host that matches the host of the request as well as wildcard virtual host resources

vnd-pi-resource-cache-ttl

The value of the resource cache **`time-to -live`** header is an integer indicating the number of seconds from the time the response was sent that the values of the **`vnd-pi-resource-cache`** header can be cached and used. For example, the following header instructs the agent to cache the resource definitions for the next ten minutes.

```
vnd-pi-resource-cache-ttl: 600
```

The agent can use the **`vnd-pi-resource-cache`** header in an agent request to ask PingAccess for new **`vnd-pi-resource-cache`** and **`vnd-pi-resource-cache-ttl`** values when the time-to-live on its current resource cache has elapsed.

PingAccess provides configurability over the resource cache **`time-to-live`** value to balance performance and security goals.

Example

An example **`vnd-pi-resource-cache`** response header is shown in the following code. This example tells the client that all requests with a path starting with `/pa/oidc/` are to have the agent make an agent request to PingAccess to determine what to do. Next, it tells PingAccess that requests with a `.jpg`, `.gif`, or `.png` suffix are allowed to pass through. Requests for a path that starts with `/canada/` require a PingAccess WAM token, which will be a cookie named PA.cad. Requests for a path that starts with `/usa/` require a PingAccess WAM token which will be a cookie named PA.usd. All other requests, indicated by the slash wildcard path, are allowed. The request path is matched against the paths defined in the resource-cache in order from top to bottom. The **`vnd-pi-resource-cache-ttl`** tells the agent to use the resource cache for the next hour.

```
vnd-pi-resource-cache: path="/pa/oidc/*"; kind=C
vnd-pi-resource-cache: path="/*.jpg" "*.gif" "*.png"; method=GET; kind=U
vnd-pi-resource-cache: path="/canada/*"; cs=N; kind=P; token-type=C; token-
name=PA.cad
vnd-pi-resource-cache: path="/usa/*"; kind=P; token-type=C; token-
name=PA.usd
vnd-pi-resource-cache: path="/*" ; kind=U
vnd-pi-resource-cache-ttl: 3600
```

Note:

The above is semantically equivalent to the following headers where the multiple **`vnd-pi-resource-cache`** header fields are combined into one.

```
vnd-pi-resource-cache: path="/pa/oidc/*"; kind=C, path="/*.jpg" "*.gif"
 "*.png"; method=GET; kind=U, path="/canada/*"; cs=N; kind=P; token-type=C;
 token-name=PA.cad, path="/usa/*"; kind=P; token-type=C; token-name=PA.usd,
 path="/*" ; kind=U
vnd-pi-resource-cache-ttl: 3600
```

Individual token and agent response caching

The resource-cache defined in the previous section gives the agent meta-information about caching data for request handling. This section describes how, in some cases, data from an individual agent response can be cached relative to a particular token and used to service future client requests with the same token so that PingAccess doesn't need to be called on every client request.

The resource-cache defined in the previous section gives the agent directives about how to handle requests based on host, method and path. The agent iterates its resource-cache list in order until it finds a match based on those values. Additional caching can be done for particular kinds of resources as follows.

When the kind of the resource-cache is P, a token, as indicated by the token-type and token-name, is required but previous agent responses for a token can be cached for efficiency. If the agent does not have a cached agent response for a particular token value, it must make an agent request to PingAccess to determine how to handle the client request. The data from that agent response can then be cached using the value of the indicated token as a key. An empty, null, or missing token should also be considered a valid cache key to support the anonymous access use case, where a WAM token is not necessary for access but, if such a token is available, user attributes from it should be exposed to the application. An agent response to a request that does not have the indicated token-type or token-name will likely contain a **`vnd-pi-set-req-headers`** directive that names non-existent headers to ensure they are stripped from the modified client request. This prevents injection of those header values by the client, even in an anonymous case.

When caching individual agent responses relative to particular tokens, the protocol directives state that the token value is obtained from the client request. However, there is one important special case where, for efficiency, the token value can be obtained from the agent response. That special case is for resources with a kind of P and token-type of C that receive a 277 agent response containing a positive **`vnd-pi-token-cache-ttl`** header value and a **`vnd-pi-append-resp-headers`** that includes the **`set-cookie`** header. Under those conditions, the agent can examine the **`set-cookie`** headers for a cookie name matching the token-name of the resource and use the value of that cookie as the token value to cache the agent response. The agent should also exclude that **`set-cookie`** header from the cached agent response content. This allows the cache to be established for an individual token in only one agent request to PingAccess when the token in the cookie is updated and set on the client.

Though the token relative caching is primarily intended as an optimization to store and reuse data associated with status code 277 responses, the following cache header defined is valid on any agent response, and agents should be prepared to cache all agent responses, rather than just 277 responses.

In general, individual agent responses for resources of kind of C are not cached. The one exception is the special case of a 477 response code where an agent can cache the 477, which tells it to send the request body on the initial agent request along with a !477 value for the **`vnd-pi-expect`** header, for a specific request URI, until the **`vnd-pi-resource-cache-ttl`** passes.

A kind value of U indicates that no agent request or response is necessary for the client request, so no additional caching is necessary.

vnd-pi-token-cache-ttl

Indicates the number of seconds from the time the agent response is issued that it can be cached relative to a specific token value. The agent must make a new agent request if the TTL on the cache entry of an individual token has expired or if no cache entry exists.

The TTL should correlate to the life of the token itself. For example, the time-to-live must be shorter than the expiration, and it needs to also allow for updates to the inactivity timeout within a reasonable threshold.

There are many tradeoffs involved, so PingAccess enables tuning and configuration options for the TTL directive.

In the event that the token is empty, null, or missing, such as the anonymous use case, the value of **`vnd-pi-token-cache-ttl`** can be the same as the value of the **`vnd-pi-resource-cache-ttl`**.

Early Cache Invalidation

PingAccess might include the following header in an agent response to instruct the agent to invalidate its cache. The agent might need to do this, for example, as a result of configuration changes.

vnd-pi-cache-invalidated

The value of this header is a numeric value representing the number of seconds from 1970-01-01T0:0:0Z UTC (the epoch) until the UTC date and time of the cache invalidation event. The value is an indicator of the most recent event that would trigger an invalidation of the cache associated with the host, or X-Forwarded-Host, of the agent request correlated to the agent response in which this header appears. An agent might ignore an invalidation directive for a timestamp that it has already processed.

It is difficult for PingAccess to know the cache state of any particular agent or group of agents. It is not reasonable to expect PingAccess to identify the exact responses that should include the cache invalidation directive. Use of the timestamp as the header value allows PingAccess to send the **`vnd-pi-cache-invalidated`** more indiscriminately while allowing the agent to relatively easily determine if it has taken, or needs to take, action with respect to a specific invalidation event.

Change Propagation and Caching

An agent populates and expunges its cache over time. As a result, configuration changes in PingAccess might take some time to propagate and might yield a mixed set of old and new behavior.

The invalidation directive set using the **`vnd-pi-cache-invalidated`** header on the agent response is intended to provide some help seeing changes take effect inside the TTL window. Though caching does reduce the number of calls made from an agent to PingAccess, there are still many requests that will necessitate the call and allow the **`vnd-pi-cache-invalidated`** header to be sent to an agent.

Example

OpenID Connect authentication

This response is passed through to the client to begin the OpenID Connect authentication process. The status and headers are passed directly through to the client.

```
HTTP/1.1 302 Found
Date: Wed, 17 Sep 2014 23:10:30 GMT
Content-Length: 0
Location: https://rhel-test.englab.corp.pingidentity.com:9031/
as/authorization.oauth2?response_type=x_post
%20id_token&client_id=pa_wam&redirect_uri=http://example.com/pa/oidc/
cb&state=aHR0cDovL3JoZWw2NS9hcHBsaWNhdGlvbi9oZWFkZXJzIEFwcGxpY2F0aW9uIFJvb3QrUmVzb3VyY2U
gs0lMY&scope=openid%20profile%20address%20email%20phone
Set-Cookie: nonce=b000c6a2-4a03-4bde-be29-956456cd1d2a; Path=/; HttpOnly
vnd-pi-resource-cache: path="/pa/*";kind=C,path="/application/*" "/
application";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/protected/
*" "/protected";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/
headers*";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/*" "/
httpbin";cs=Y;kind=P;token-type=C;token-name=PA.post
vnd-pi-resource-cache-ttl: 900
vnd-pi-token-cache-ttl: 300
```

Example

Request for POST body

This response requests the POST body that was omitted from the initial agent request

```
HTTP/1.1 477 Request Body Required
Date: Wed, 17 Sep 2014 23:10:35 GMT
Content-Length: 0
vnd-pi-resource-cache: path="/pa/*";kind=C,path="/application/*" "/
application";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/protected/
*" "/protected";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/
headers*";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/*" "/
httpbin";cs=Y;kind=P;token-type=C;token-name=PA.post
vnd-pi-resource-cache-ttl: 900
```

Example

Redirect

This response issues a redirect. The status code and headers are passed directly through to the client.

```
HTTP/1.1 302 Found
Date: Wed, 17 Sep 2014 23:10:36 GMT
Content-Length: 0
Location: http://example.com/application/headers
Set-Cookie:
 PA.post=eyJraWQiOiJhcCIsImFsZyI6IkVTMjU2In0.eyJ6b25laW5mbyI6IkFtZXJpY2FcL05ld19Zb3JrIiw
j0BDdLoGeVdqWD35n9ZxFhphEHFe7tfQ6onKAjRdXLR5rtwPBkJHkLaTLD8Yqcsf0izVw;
 Path=/; HttpOnly
Set-Cookie: nonce=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT
```

```
vnd-pi-resource-cache: path="/pa/*";kind=C,path="/application/*" "/
application";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/protected/
*" "/protected";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/
headers*";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/*" "/
httpbin";cs=Y;kind=P;token-type=C;token-name=PA.post
vnd-pi-resource-cache-ttl: 900
```

Example

Grant access

This response grants access and allows the client request through to the application with the appropriate application headers set and with the caching directives.

```
HTTP/1.1 277 Allowed
Date: Wed, 17 Sep 2014 23:10:36 GMT
Content-Length: 0
vnd-pi-resource-cache: path="/pa/*";kind=C,path="/application/*" "/
application";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/protected/
*" "/protected";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/
headers*";cs=Y;kind=P;token-type=C;token-name=PA.post,path="/httpbin/*" "/
httpbin";cs=Y;kind=P;token-type=C;token-name=PA.post
vnd-pi-resource-cache-ttl: 900
vnd-pi-token-cache-ttl: 300
USER: joe
vnd-pi-sub: USER
vnd-pi-set-req-headers: USER
```

PAAP modified client request

If the agent response status is 277, the client request is modified according to the directives in the agent response and the request is passed along to the application or allowed to continue processing in the HTTP processing pipeline of the environment in which the agent is deployed.

Example

Additional HTTP headers

This example shows the additional HTTP headers added as specified by PingAccess.

```
GET /application/headers HTTP/1.1
Host: http://example.com/
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Cookie:
 PA.post=eyJraWQiOiJhcCIsImFsZyI6IkVTMjU2In0.eyJ6b25laW5mbyI6IkFtZXJpY2FcL05ld19Zb3JrIiw
j0BDdLoGeVdqWD35n9ZxFhphEHFe7tfQ6onKAjRdXLR5rtwPBkJHkLaTLD8Yqcsf0izVw
USER: joe
```

PAAP client response

The client response is the HTTP response corresponding to the client request.

If the agent response status code is anything other than 277, its content is sent back to the client as the content of the client response, minus any headers identified for exclusion. If the agent response status code is 277, the client request is passed to the protected application, and the client response is the response from the application with any header modifications indicated by the agent response.

Example

Pass-through from agent response

This client response shows the direct pass-through of the status code and headers from the agent response.

See the agent response OpenID Connect authentication example for a related example.

```
HTTP/1.1 302 Found
Date: Wed, 17 Sep 2014 23:10:30 GMT
Content-Length: 0 Location: https://rhel-
test.englab.corp.pingidentity.com:9031/as/authorization.oauth2?
response_type=x_post%20id_token&client_id=pa_wam&redirect_uri=http://
example.com/pa/oidc/
cb&state=aHR0cDovL3JoZWw2NS9hcHBsaWNhdGlvbi9oZWFkZXJzIEFwcGxpY2F0aW9uIFJvb3QrUmVzb3VyY2U
gs0lMY&scope=openid%20profile%20address%20email%20phone
Set-Cookie: nonce=b000c6a2-4a03-4bde-be29-956456cd1d2a; Path=/; HttpOnly
```

Example

Client response with redirect

This client response shows the direct pass-through of the status code and headers from the agent response.

See the agent response redirect example for a related example.

```
HTTP/1.1 302 Found
Date: Wed, 17 Sep 2014 23:10:36 GMT
Content-Length: 0
Location: http://example.com/application/headers
Set-Cookie:
 PA.post=eyJraWQiOiJhcCIsImFsZyI6IkVTMjU2In0.eyJ6b25laW5mbyI6IkFtZXJpY2FcL05ld19Zb3JrIiw
PingAccess 3.x SpecificationAgent Protocol Specification V1.0Page 25
 JmZW1hbGUiLCJwcm9maWxlIjoiaHR0cHM6XC9cL3d3dy5waW5naWRlbnRpdHkuY29tXC9wcm9kdWN0c1wvcGluZ
j0BDdLoGeVdqWD35n9ZxFhphEHFe7tfQ6onKAjRdXLR5rtwPBkJHkLaTLD8Yqcsf0izVw;
 Path=/; HttpOnly
Set-Cookie: nonce=; Path=/; Expires=Thu, 01-Jan-1970 00:00:00 GMT
```

PingAccess Agent SDK for C

This documentation provides technical guidance for using the PingAccess Agent SDK for C. Use this guide along with the API documentation for the SDK and sample source code to implement custom agents that use the PingAccess Agent Protocol to integrate with a PingAccess policy server.

Intended Audience

This guide is intended for application developers and system administrators responsible for implementing a C-based PingAccess agent. The reader should be familiar with C software development principles and practices. It describes the use of the SDK within a sample agent for Apache.

Additional documentation

The SDK documentation provides detailed reference information for developers. After extracting the `pingaccess-agent-c-sdk-<version>.zip` package, access the API documentation with a web browser by viewing the file `AGENT_SDK_C_HOME/apidocs/index.html`. Alternatively, find the current version of the API documentation online at *https://www.pingidentity.com/content/dam/developer/documentation/pingaccess/agent-c-sdk/1-1-4/apidocs/index.html*

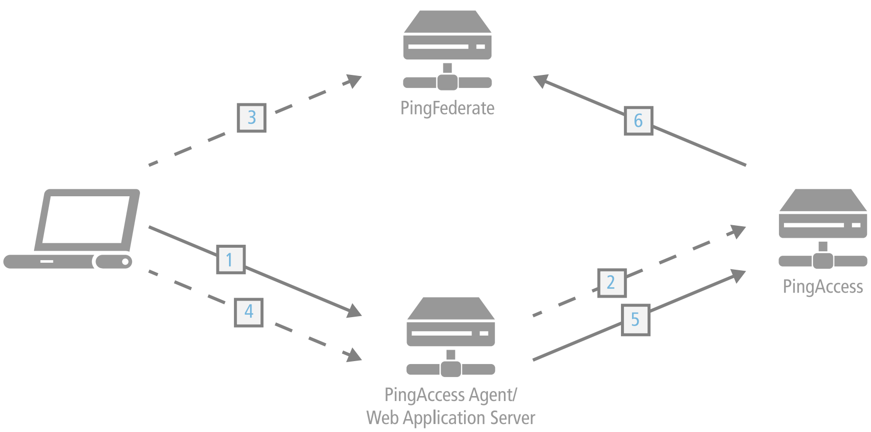
Introduction

The PingAccess Agent SDK for C provides an API and sample code to enable developers to build agents for C or C++-based application and web servers.

Supported platforms include:

- Red Hat Enterprise Linux Server 7 (32 bit)
- Red Hat Enterprise Linux Server 8 (32 bit or 64 bit)
- SUSE Linux Enterprise Server 12 SP2 (64 bit)
- Microsoft Windows Server 2012
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Agents provide access management features to their containing server by relying on central PingAccess servers over the PingAccess Agent Protocol. The *PingAccess Agent Protocol Specification* is available from the Ping Identity support portal.



Processing steps

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

The PingAccess Agent SDK for C consists of the following components:

SDK (C Agent)

The SDK is a set of C header files that represent the interface to the library that implements the PingAccess Agent Protocol.

C Agent libraries

The C libraries implement the PingAccess Agent Protocol. There are binaries for Red Hat Enterprise Linux 7/8 as well as for Windows.

PingAccess Agent SDK for C API documentation

Each of the interfaces defined in the header files is fully documented.

Apache Agent Sample

`<AGENT_SDK_FOR_C_HOME>/sample` : The Apache Agent Sample demonstrates how the SDK integrates into Apache as an Apache module that is integrated with the Apache request processing workflow. The provided source code and module configuration provide a functional example for how to integrate the SDK into an existing web application. The sample can be modified in-place and recompiled using **`make`** to test customizations to the Sample code for your environment.

Note:

This sample code demonstrates how to implement the PingAccess Agent as an Apache module and has been qualified in the following environments:

- Red Hat Enterprise Linux 7 (RHEL7), 64-bit
- Red Hat Enterprise Linux 8 (RHEL8), 64-bit

The Apache Agent itself is production-quality and can be used either as-is or as a starting point for further development. While Ping Identity provides this as a sample, the only versions that are fully supported in production are the precompiled versions available from the Ping Identity *download site*.

The sample includes instructions for how to configure the sample as a PingAccess Agent to protect websites within its scope. Further hardening of the Apache server configuration or of the sample configuration file might be required.

If you need assistance using the PingAccess Agent SDK for C, visit the Ping Identity *Support Center* for help you with your application. Engage the Ping Identity Professional Services team for assistance with developing customizations.

To download the SDK, go to the *PingAccess downloads site* and click the **Add-ons** tab.

Getting Started with the PingAccess Agent SDK for C

Agent SDK for C directory structure
The PingAccess Agent SDK for C directory contains these subdirectories.

`/`

This directory contains the Agent SDK for C `README.md`, which contains information developers need to develop agents using the SDK. It also contains `ReadMeFirst.pdf` and `Legal.pdf`, which contain general information about the kit and third-party licenses used by components of the SDK.

`/apidocs`

API documentation for the SDK. Open `index.html` to access the API documentation content.

`/include`

Agent SDK header files.

`/lib`

32-bit and 64-bit libraries for Red Hat Enterprise Linux 7 and 8, and Windows, including third-party dependencies required by the SDK.

`/sample`

Sample source code for an agent for Apache. This sample agent uses the SDK and includes a sample configuration file for Apache to use the sample agent to enforce authentication and access control policies.

Agent SDK for C sample code

The Agent SDK for C sample code is available both in the SDK distribution and on GitHub at *https://github.com/pingidentity/pa-agent-c-sdk-sample-apache*.

Before building the sample code, ensure you have the PingAccess Agent SDK for C archive, the GNU **`make`** utility and associated compiler utilities installed with your compiler, and Apache and its development libraries.

The sample uses Apache and assumes that the PingAccess Agent SDK for C can be referenced as a dependency. For more details about specific dependencies and requirements, as well as instructions on how to build the sample code, see `<AGENT_SDK_C_HOME>/sample/readme.md`.

Release notes

These release notes summarize the changes in current and previous PingAccess Agent SDK for C updates.

> **Note:**
>
> The PingAccess Agent SDK for C no longer supports FreeBSD 8.

Version 1.3 - June 2020

- Added support for RHEL 8
- Added agent inventory callback API
- Removed support for RHEL 6

Version 1.2.1 - February 2020

Fixed a potential security issue.

Version 1.2 - June 2019

Fixed a potential security issue.

Version 1.1.5 - February 2019

Added support for FreeBSD 8

Version 1.1.4 - October 2018

Fixed potential security issues.

Version 1.1.3 - August 2018

- Updated version of libcurl to fix an issue where libcurl was only checking the first SAN in the server certificate
- Fixed a potential security issue

Version 1.1.2 - March 2017

Added support for:

- SUSE Linux Enterprise Server 11 SP4 (x86_64)
- SUSE Linux Enterprise Server 12 SP2 (x86_64)

Version 1.1.1 - January 2017

- Established a workaround for a *known issue* in the Network Security Services library that results in a memory leak when the agent closes a HTTPS connection to a PingAccess policy server. For more information, see *this KB article*.
- Fixed an issue where duplicate headers were included in the backend request to the PingAccess Engine, causing the agent to block the request for content.

Version 1.1 - November 2016

Added policy server failover support. Policy server failover support is only provided by the SDK when using the libcurl HTTP client.

Version 1.0.2 - September 2016

- Fixed an issue where agents could not communicate with PingAccess servers using a certificate signed by a certificate authority because the CRL Distribution Point extension is missing. This issue is limited to agents on Windows deployments.
- Addressed a potential security vulnerability. This issue is limited to Windows deployments.

Version 1.0.1 - May 2016

Fixed an issue with ZeroMQ policy cache where a terminated process could cause a condition that resulted in unexpected CPU utilization.

Version 1.0 - April 2016

Initial Release.

PingAccess Agent SDK for Java

This document provides technical guidance for using the PingAccess Agent SDK for Java. Use this guide along with the Javadocs for the Java Agent API and sample source code to implement the PingAccess Agent Protocol in custom agents.

Intended audience

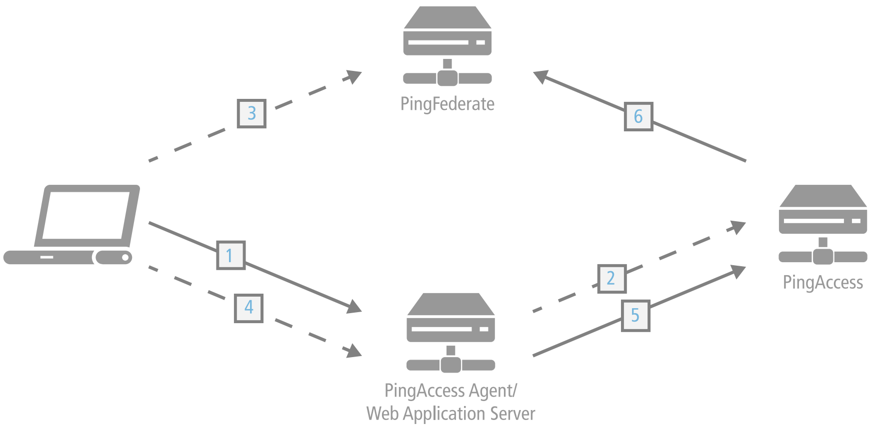
Application developers and system administrators responsible for implementing a Java PingAccess agent should be familiar with Java software-development principles and practices. It describes the use of the SDK within a sample Java Servlet Filter.

Additional documentation

The Java Agent API Javadocs provides detailed reference information for developers. After extracting the `pingaccess-agent-java-sdk-1.1.3.zip` package, the Javadocs is accessed with a web browser by viewing the file `<AGENT_SDK_JAVA_HOME>/apidocs/index.html`.

Introduction

The PingAccess Agent SDK for Java provides an API and sample code to enable developers to build agents for Java-based applications and web servers. Agents provide access management features to their containing server by relying on central PingAccess servers over the *PingAccess Agent Protocol*.



Processing steps:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user signs on, and PingFederate creates the session.
4. The client is redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

The PingAccess Agent SDK for Java consists of the following components:

Java Agent API (Java Agent)

`pingaccess-agent-java-api-1.1.3.0.jar`: The Java Agent API is a set of classes that implement the PingAccess Agent Protocol.

PingAccess Agent SDK for Java

`pingaccess-agent-java-sdk-1.1.3.zip`: The PingAccess Agent SDK for Java package.

Servlet Filter Sample

`<AGENT_SDK_FOR_JAVA_HOME>/sample` : The Servlet Filter Sample demonstrates how the Java Agent API integrates into a Java Servlet container. The provided source code, logging configuration and deployment descriptor provide a functional example for how to integrate the Java Agent API into an existing web application. The sample can be modified in place and recompiled using Maven to test customizations to the Servlet Filter Sample code for your environment.

> **Note:**
>
> This sample code demonstrates how to implement a servlet filter and has been qualified on Apache Tomcat 7. The filter itself is production quality and can be used either as-is or as a starting point for further development. Application configuration within the sample demonstrates how to associate the filter with a servlet, namely in `web.xml`. Further hardening of this file or the application server configuration might be required.

If you need assistance using the PingAccess Agent SDK for Java, visit the Ping Identity *Support Center* to see how we can help you with your application. You can engage the Ping Identity Global Client Services team for assistance with developing customizations.

Downloading the SDK

To download the SDK, go to the *PingAccess downloads site* and click the **Add-ons** tab.

Agent SDK directory structure

The PingAccess Agent SDK for Java directory contains the following directories.

`/apidocs`

The Javadocs for the Java Agent API. Open `index.html` in this directory to access the Javadocs content.

`/dist`

The directory containing `pingaccess-agent-java-api-1.1.3.0.jar`

`/sample`

A directory containing `src` and `target` directories for building a Java Servlet Filter. This filter uses the Java Agent API, an `agent.properties` configuration exported from PingAccess, and the `init-params` from the web application `web.xml` file to enforce resource policy decisions configured in PingAccess.

Agent SDK prerequisites

Verify that your system meets these prerequisites before installing the PingAccess Agent SDK for Java.

Before you start, ensure you have the Java SDK, *Apache Maven* and an application server, such as Apache Tomcat, installed. The sample uses Apache Maven and assumes that the Java Agent API can be referenced as a dependency. It references Ping Identity's public Maven repository, located at

```
http://maven.pingidentity.com/release
```

If Internet access is unavailable, there are two other ways to reference the Java Agent API. First, after Apache Maven is installed, install the Java Agent API into your local dependency repository by executing the following command.

```
mvn install:install-file -Dfile=<AGENT_SDK_JAVA_HOME>/dist/pingaccess-agent-
java-api-1.1.3.0.jar -DgroupId=com.pingidentity -DartifactId=pingaccess-
agent-java-api -Dversion=1.1.3.0 -Dpackaging=jar
```

Alternatively, update the dependency in your `pom.xml` to point to the local installation.

```
<dependency>
        <groupId>com.pingidentity</groupId>
        <artifactId>pingaccess-agent-java-api</artifactId>
        <version>1.1.3.0</version>
        <scope>system</scope>
        <systemPath><AGENT_SDK_JAVA_HOME>/dist/pingaccess-agent-java-
api-1.1.3.0.jar</systemPath>
</dependency>
```

With either of these options, replace *<AGENT_SDK_JAVA_HOME>* with the absolute path to the extracted `pingaccess-agent-java-sdk-1.1.3.0` directory.

To download the SDK, go to the *PingAccess downloads site* and click the **Add-ons** tab.

Installing the servlet filter sample

Install the servlet filter sample.

Before you begin

Ensure you have the PingAccess Agent SDK for Java, Apache Maven, and Apache Tomcat. These instructions assume that you are using Apache Tomcat.

About this task

- The servlet filter sample is installed under `<AGENT_SDK_JAVA_HOME>/sample`.
- A deployed version of the servlet filter is under `<AGENT_SDK_JAVA_HOME>/sample/target/agent-sample`.

For the initial setup of the web application, we assume you already have Tomcat or another application server set up on the same machine hosting PingAccess. Out of the box, PingAccess generates self-signed server certificates for listeners servicing runtime ports with the hostname `localhost`. By default, the servlet filter sample configures the Java Agent, Java Agent API, to use strict certificate checking for communications with PingAccess. The Java Agent will not be able to communicate with PingAccess over HTTPS if it is not also on `localhost` because of strict hostname checking. If PingAccess already has a server certificate configured with a valid hostname other than `localhost`, then you can deploy the Java Agent into a container on another system.

If you cannot setup the application server on the same system as an existing PingAccess service, and that PingAccess deployment still uses the default `localhost` server certificate for the agent port, there is another option. You can change the default `strict` certificate checking in `agent-sample/WEB-INF/web.xml` to `test`. See the comments in `agent-sample/WEB-INF/web.xml` for more detail.

Steps

1. In the Tomcat `webapps` directory, create a directory called `ROOT`.
2. Copy the `WEB-INF`, `META-INF`, and `assets` contents from `/sample/target/agent-sample/` into `webapps/ROOT`.

   This sample servlet filter must run as `/` to properly carry out the OpenID Connect workflow.
3. In the Tomcat `bin` directory, create a script called `setenv.sh` (Linux) or `setenv.bat` (Windows) with the following contents:
   Choose from:

   - For Linux:

     ```
     export CATALINA_OPTS="-Dlog4j.configurationFile=<PATH_TO_TOMCAT_ROOT>/
     webapps/ROOT/WEB-INF/logs/log4j2.xml -
     Dserver.log.file=<PATH_TO_TOMCAT_ROOT>/webapps/ROOT/WEB-INF/logs/
     server.log"
     ```
   - For Windows:

     ```
     set CATALINA_OPTS=="-Dlog4j.configurationFile=<PATH_TO_TOMCAT_ROOT>/
     webapps/ROOT/WEB-INF/logs/log4j2.xml -
     Dserver.log.file=<PATH_TO_TOMCAT_ROOT>/webapps/ROOT/WEB-INF/logs/
     server.log"
     ```

   The agent servlet filter logging is configured in `webapps/ROOT/WEB-INF/logs/log4j2.xml` and outputs to `webapps/ROOT/WEB-INF/logs/server.log`.
4. If running Tomcat on Linux, execute the command `chmod a+x setenv.sh` to make this script executable.
5. Configure a PingAccess agent.

6. Configure an application and associate the new agent with it.
7. When configuring an agent through the PingAccess administration console, it automatically exports the agent properties file. Copy the downloaded properties file to `webapps/ROOT/WEB-INF/agent-config/agent.properties`.

> **Important:**
>
> If Tomcat is running on Java version 7, some version 8 cipher suites are unavailable. This might lead to errors.
>
> To work around this issue, edit `agent.properties` to remove the following cipher suites from `agent.ssl.ciphers`:
>
> - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
> - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
> - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

8. Start Tomcat.
9. Open a browser and go to `http://<HOST>:<PORT>/sample`.

   The values for <*HOST*> and <*PORT*> here need to match the Tomcat configuration in use.

> **Note:**
>
> If your Tomcat server is not set up to use HTTPS, ensure that any related Web Sessions do not have the **Secure** option enabled.

Release Notes

These release notes summarize the changes in current and previous PingAccess Agent SDK for Java updates.

Version 1.1.4 - December 2021

Resolved issues:

- Upgraded Log4j used in the SDK sample implementation to version 2.16.

Version 1.1.3 - July 2018

Resolved issues:

- Fixed an issue where site cookies were not being set properly for HTTP redirects in the servlet container environment (Tomcat).
- Fixed an issue where an error was being generated by the SDK sample implementation of `ClientHttpServletRequest.getHeaders` when calling a header that does not exist.

Version 1.1.2 - June 2017

Resolved issues:

- Fixed an issue where the Java agent was handling the PingAccess `set-cookie` header incorrectly.
- Fixed an issue where the Java agent wasn't correctly processing multiple `set-cookie` headers sent by PingAccess.
- Fixed an issue where the SDK sample implementation was not correctly enforcing the PingAccess Agent Protocol directives when the ClientHttpServletRequest `getCookies` method was called. This resulted in a discrepancy between the cookie request headers returned from the `getHeader*` methods and the `getCookies` method.

Version 1.1.1 - April 2017

Resolved issues:

- Fixed an issue where unknown attributes should be ignored
- Fixed an issue where percent-encoded sequences in resource paths were being handled incorrectly
- Fixed an issue where an "Index out of bounds" exception occurs if a cookie value is "".

Version 1.1 - August 2015

Resolved issues:

- Fixed an issue where OAuth API response headers were getting trimmed
- Fixed an issue where the Java Agent enforced the requirement of a username and shared secret
- Fixed an issue where the Agent was not handling a 477 response correctly

Version 1.0 - June 2015

Initial Release

PingAccess Add-on SDK for Java

The PingAccess Add-on SDK provides extension points that let users customize certain behaviors of PingAccess to suit their needs. This SDK provides the means to develop, compile, and deploy custom extensions to PingAccess.

The PingAccess Add-on SDK provides the following extension points:

RuleInterceptor

An interface for developing custom rule implementations to control authorization logic in policies.

SiteAuthenticatorInterceptor

An interface for developing custom site authenticators to control how PingAccess, operating as a proxy, integrates with web servers or services it is protecting.

IdentityMappingPlugin

An interface for developing custom identity mappings to provide user identity information to an application within PingAccess.

LoadBalancingPlugin

An interface for developing custom load balancing strategies that provide the logic for load balancing requests to target hosts configured for a Site.

LocaleOverrideService

An interface for developing custom logic for resolving the locale of a request used for localization.

If you need assistance using the SDK, visit the Ping Identity *Support Center* to see how we can help you with your application. You can engage the Ping Identity Global Client Services team for assistance with developing customizations.

Get started with the SDK

This section describes the directories and build components that comprise the SDK and provides instructions for setting up a development environment.

For more information, see the following topics:

SDK directory structure

The PingAccess SDK directories, `<PA_HOME>/sdk` and `<PA_HOME>/deploy`, contain these files and directories.

Deploy directory

The `<PA_HOME>/deploy` directory is created as a location for all third-party JAR files. PingAccess does not automatically generate any contents for this directory, but any files you place in it are automatically migrated during an upgrade.

The contents of the `<PA_HOME>/deploy` directory are loaded by the `run.sh` or `run.bat` command.

SDK directory

The `<PA_HOME>/sdk` directory contains these files and directories.

File/Directory	Description
`README.md`	Contains an overview of the SDK contents.
`/samples/README.md`	Contains an overview of the steps necessary to build and use the samples.
`/samples/Rules`	Contains a Maven project with example plug-in implementations for rules showing a wide range of functionality. You can use these examples for developing your own implementations.
`/samples/Rules/README.md`	Contains the details of the `Rules` samples.
`/samples/SiteAuthenticator`	Contains a maven project with example plug-in implementations for site authenticators. Use these examples for developing your own implementations.
`/samples/SiteAuthenticator/README.md`	Contains the details of the `SiteAuthenticator` samples.
`/samples/IdentityMappings`	Contains a maven project with example plug-in implementations for identity mappings. Use these examples for developing your own implementations.
`/samples/IdentityMappings/README.md`	Contains the details of the `IdentityMappings` samples.
`/samples/LoadBalancingStrategies`	Contains a maven project with example plug-in implementations for load balancing strategies. Use these examples for developing your own implementations .

File/Directory	Description
`/samples/LoadBalancingStrategies/README.md`	Contains the details of the `LoadBalancingStrategies` samples.
`/samples/LocaleOverrideService`	Contains a maven project with example plug-in implementations for the locale override service. Use these examples for developing your own implementations.
`/samples/LocaleOverrideService/README.md`	Contains the details of the `LocaleOverrideService` samples.
`/apidocs/`	Contains the SDK Javadocs. To get started, open `index.html`.

SDK prerequisites

The following prerequisites must be met before using the Add-on SDK for Java.

Before you start, ensure you have the Java SDK and *Apache Maven* installed. The samples use Apache Maven and assume that the PingAccess SDK can be referenced as a dependency. They reference Ping Identity's public Maven repository, located at: `http://maven.pingidentity.com/release`.

Note:

The Ping Identity Maven repository cannot be accessed through a browser because it is designed solely for backend use. To use it, add it to your Maven configuration. For example, including this code in the Maven `pom.xml` file adds the Ping Identity repository to your Maven configuration:

```
    <repositories>
        <repository>
            <releases>
                <enabled>true</enabled>
                <updatePolicy>always</updatePolicy>
                <checksumPolicy>warn</checksumPolicy>
            </releases>
            <id>PingIdentityMaven</id>
            <name>PingIdentity Release</name>
            <url>http://maven.pingidentity.com/release/</url>
            <layout>default</layout>
        </repository>
    </repositories>
```

If Internet access is unavailable, update the `pingaccess-sdk` dependency in your `pom.xml` to point to the local installation.

Replace *<PA_HOME>* with the path to the PingAccess installation.

```
<dependency>
        <groupId>com.pingidentity.pingaccess</groupId>
        <artifactId>pingaccess-sdk</artifactId>
        <version>4.0.1.3</version>
        <scope>system</scope>
        <systemPath><PA_HOME>/lib/pingaccess-sdk-4.2.0.0.jar</systemPath>
</dependency>
<dependency>
        <groupId>javax.validation</groupId>
        <artifactId>validation-api</artifactId>
```

```
        <version>1.0.0.GA</version>
  <scope>system</scope>
        <systemPath><PA_HOME>/lib/validation-api-1.0.0.GA.jar</systemPath>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>1.7.4</version>
  <scope>system</scope>
        <systemPath><PA_HOME>/lib/slf4j-api-1.7.4.jar</systemPath>
</dependency>
<dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.4</version>
  <scope>system</scope>
        <systemPath><PA_HOME>/lib/slf4j-log4j12-1.7.4.jar</systemPath>
</dependency>
```

Installing the SDK samples

Install rule and site authenticator SDK samples.

Before you begin

Ensure you have the Java SDK and Apache Maven installed.

About this task

Each sample type is installed separately:

- For the rules samples, go to `<PA_HOME>/sdk/samples/Rules`.
- For the site authenticators samples, go to `<PA_HOME>/sdk/samples/SiteAuthenticator`.

Steps

- From the sample's directory, run the command `$ mvn install`.

  This builds the samples, runs their tests, and copies the resulting JAR file from the target directory to the `<PA_HOME>/lib` directory.

Example

```
jsmith-MBP-2:Rules jsmith$ mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] Using the builder
 org.apache.maven.lifecycle.internal.builder.singlethreaded.SingleThreadedBuilder
 with a thread count of 1
[INFO]
[INFO]
 ------------------------------------------------------------------------
[INFO] Building PingAccess :: Sample Rules 3.0.0-RC5
[INFO]
 ------------------------------------------------------------------------
Downloading: http://...
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @
 sample-rules ---
[INFO] Using 'ISO-8859-1' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ sample-
rules ---
```

```
[INFO] Compiling 7 source files to /Users/jsmith/Downloads/pingaccess-3.0.0-
RC5/sdk/samples/Rules/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources)
 @ sample-rules ---
[INFO] Using 'ISO-8859-1' encoding to copy filtered resources.
[INFO] Copying 4 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @
 sample-rules ---
[INFO] Compiling 4 source files to /Users/jsmith/Downloads/pingaccess-3.0.0-
RC5/sdk/samples/Rules/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ sample-rules
 ---
[INFO] Surefire report directory: /Users/jsmith/Downloads/pingaccess-3.0.0-
RC5/sdk/samples/Rules/target/surefire-reports
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.pingidentity.pa.sample.TestAllUITypesAnnotationRule
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.912 sec
Running com.pingidentity.pa.sample.TestIllustrateManyUITypesRule
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.029 sec
Running com.pingidentity.pa.sample.TestValidateRulesAreAvailable
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec
Results :
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ sample-rules ---
[INFO] Building jar: /Users/jsmith/Downloads/pingaccess-3.0.0-RC5/sdk/
samples/Rules/target/sample-rules-3.0.0-RC5.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ sample-rules
 ---
[INFO] Installing /Users/jsmith/Downloads/pingaccess-3.0.0-RC5/sdk/samples/
Rules/target/sample-rules-3.0.0-RC5.jar to /Users/jsmith/.m2/repository/com/
pingidentity/pingaccess/sample-rules/3.0.0-RC5/sample-rules-3.0.0-RC5.jar
[INFO] Installing /Users/jsmith/Downloads/pingaccess-3.0.0-RC5/sdk/samples/
Rules/pom.xml to /Users/jsmith/.m2/repository/com/pingidentity/pingaccess/
sample-rules/3.0.0-RC5/sample-rules-3.0.0-RC5.pom
[INFO]
[INFO] --- maven-antrun-plugin:1.7:run (default) @ sample-rules ---
[INFO] Executing tasks
main:
     [copy] Copying 1 file to /Users/jsmith/Downloads/pingaccess-3.0.0-RC5/
lib
[INFO] Executed tasks
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 6.418 s
[INFO] Finished at: 2014-07-08T16:38:30-07:00
[INFO] Final Memory: 16M/38M
[INFO]
 ------------------------------------------------------------------------
```

Create your own plugins

Create your own plugins from scratch using the Add-on SDK.

Generally, the following steps are taken to implement a plugin:

1. Create a new, empty Maven project. The root directory of the Maven project is referred to as `<PLUGIN_HOME>`.
2. Copy the `pom.xml` from the appropriate sample provided in `<PA_HOME>/sdk/samples`.

   Note:

   For example, to create a rule, copy the `pom.xml` from *`<PA_HOME>`*`/sdk/samples/Rules/` to `<PLUGIN_HOME>/`.
3. Modify the `groupId`, `artifactId`, `name`, and `version` in the copied `pom.xml` file as appropriate.
4. Create a Java class that implements the plugin interface from the SDK in the *`<PLUGIN_HOME>`*`/src/main/java/com/yourpackagename` directory. This interface is referred to as a Service Provider Interface (SPI).

   Note:

   For example, to implement a custom rule, the class should implement the RuleInterceptor SPI.

   For each SPI, base classes are provided that simplify the implementation of the SPI.
5. Create a provider-configuration file for the plugin SPI containing the fully-qualified class name for the class created in the previous step.

   Note:

   For example, to implement a custom rule, create a file called *`<PLUGIN_HOME>`*`/META-INF/services/com.pingidentity.pa.sdk.policy.RuleInterceptor`. Its contents are the FQCN of the class.
6. Build the Maven project to obtain a jar containing the plugin implementation.
7. Copy the jar to *`<PA_HOME>`*`/deploy`.

   Important:

   After copying a custom plugin JAR to the PingAccess lib, you must restart PingAccess to complete the deployment of the custom plugin.

The following sections provide the details required to complete these steps for each type of plugin:

- *Rule details*
- *Site authenticator details*
- *Identity mapping details*
- *Load balancing strategy details*
- *Locale override service details*

Rule details

If you do not need to integrate with a third-party service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.policy.RuleInterceptor`

Provider-configuration file

<PLUGIN_HOME>/META-INF/services/
com.pingidentity.pa.sdk.policy.RuleInterceptor

Base classes

`com.pingidentity.pa.sdk.policy.RuleInterceptorBase`

If you need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.policy.AsyncRuleInterceptor`

Provider-configuration file

<PLUGIN_HOME>/META-INF/services/
com.pingidentity.pa.sdk.policy.AsyncRuleInterceptor

Base classes

`com.pingidentity.pa.sdk.policy.AsyncRuleInterceptorBase`

Site authenticator details

If you do not need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptor`

Provider-configuration file

<PLUGIN_HOME>/META-INF/services/
com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptor

Base classes

`com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptorBase`

If you need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.siteauthenticator.AsyncSiteAuthenticatorInterceptor`

Provider-configuration file

<PLUGIN_HOME>/META-INF/services/
com.pingidentity.pa.sdk.siteauthenticator.AsyncSiteAuthenticatorInterceptor

Base classes

`com.pingidentity.pa.sdk.siteauthenticator.AsyncSiteAuthenticatorInterceptorBase`

Identity mapping details

If you do not need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.identitymapping.IdentityMappingPlugin`

Provider-configuration file

<PLUGIN_HOME>/META-INF/services/
com.pingidentity.pa.sdk.identitymapping.IdentityMappingPlugin

Base classes

`com.pingidentity.pa.sdk.identitymapping.IdentityMappingPluginBase`

`com.pingidentity.pa.sdk.identitymapping.header.HeaderIdentityMappingPlugin`

If you need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.identitymapping.AsyncIdentityMappingPlugin`

Provider-configuration file

`<PLUGIN_HOME>/META-INF/services/`
`com.pingidentity.pa.sdk.identitymapping.AsyncIdentityMappingPlugin`

Base classes

`com.pingidentity.pa.sdk.identitymapping.AsyncIdentityMappingPluginBase`

Load balancing strategy details

If you do not need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.ha.lb.LoadBalancingPlugin`

Provider-configuration file

`<PLUGIN_HOME>/META-INF/services/`
`com.pingidentity.pa.sdk.ha.lb.LoadBalancingPlugin`

Base classes

`com.pingidentity.pa.sdk.ha.lb.LoadBalancingPluginBase`

If you need to integrate with a Third-Party Service, use the following SPIs and base classes:

SPI

`com.pingidentity.pa.sdk.ha.lb.AsyncLoadBalancingPlugin`

Provider-configuration file

`<PLUGIN_HOME>/META-INF/services/`
`com.pingidentity.pa.sdk.ha.lb.AsyncLoadBalancingPlugin`

Base classes

`com.pingidentity.pa.sdk.ha.lb.AsyncLoadBalancingPluginBase`

Locale override service details

A Locale Override Service cannot integrate with a Third-Party Service, so the following SPIs and base classes are used for all implementations:

SPI

`com.pingidentity.pa.sdk.localization.LocaleOverrideService`

Provider-configuration file

`<PLUGIN_HOME>/META-INF/services/`
`com.pingidentity.pa.sdk.localization.LocaleOverrideService`

Base classes

No base classes are provided.

Integrate with third-party services

The Add-on SDK includes the ability for a custom plugin to integrate with external, third-party services using HTTP.

This section provides a high-level overview of utilizing this functionality from a custom plugin:

- *Obtaining the HTTP client instance* on page 466
- *Obtaining a handle to a third-party service* on page 466
- *Making a HTTP call to a third-party service* on page 468

- *Base classes* on page 468
- *Sample plugins* on page 468

Obtaining the HTTP client instance

PingAccess provides access to a HTTP client utility interface, HttpClient, through dependency injection. Plugins are expected to obtain an instance of this interface using an approach like the following.

```
public class DocumentationPlugin // interfaces and base classes omitted for
 brevity
{
    private HttpClient httpClient;

    // ... other code omitted ...

    @Inject
    public void setHttpClient(HttpClient httpClient)
    {
        this.httpClient = httpClient;
    }

    // ... other code omitted ...
}
```

Obtaining a handle to a third-party service

Given a HttpClient instance, a plugin will also need a handle to a third-party service to make an outbound HTTP call to the service represented by the Third-Party Service administrative configuration object. This handle is an instance of the ThirdPartyServiceModel class and is specified to the HttpClient in its send method.

There are two different ways to obtain a ThirdPartyServiceModel instance:

- *Administrator-configured third-party services*
- *Third-party services for the OAuth authorization server and OIDC provider*

Administrator-configured third-party services

The PingAccess Administrative UI and API allow administrators to define the communication configuration for an external service by defining a third-party service. These configuration objects can then be associated with custom plugins through their configuration.

To enable a plugin's configuration to reference a third-party service, it should define a field in the configuration with the type of ThirdPartyServiceModel.

```
    private static class Configuration extends SimplePluginConfiguration
    {
        // ... other code omitted ...

        @UIElement(order = 30,
                type = ConfigurationType.SELECT,
                label = "Risk Authorization Service",
                modelAccessor = ThirdPartyServiceAccessor.class,
                required = true)
        @NotNull
        private ThirdPartyServiceModel riskAuthzService;

        // ... other code omitted ...

        public ThirdPartyServiceModel getRiskAuthzService()
        {
```

```
            return riskAuthzService;
        }

        public void setRiskAuthzService(ThirdPartyServiceModel
 riskAuthzService)
        {
            this.riskAuthzService = riskAuthzService;
        }
    }
```

The important items in this example:

- The `modelAccessor` attribute of the UIElement must be set to `ThirdPartyServiceAccessor`.
- The field in the plugin configuration class must be of type `ThirdPartyServiceModel`.

Third-party services for the OAuth authorization server and OIDC provider

In addition to providing a way for an administrator to configure a plugin to use an arbitrary third-party service, PingAccess allows a plugin to use a third-party service that represents the OAuth Authorization Server or OpenID Connect (OIDC) provider. The benefit of leveraging this functionality is that a plugin can require access to either of these services without requiring the administrator to configure the plugin to use those services.

Similar to the previous section, the plugin obtains a ThirdPartyServiceModel instance that is a handle to the OAuth Authorization Server or OIDC provider by indicating this requirement in its plugin configuration class. However, the mechanism is a bit different, as shown in the following example.

```
    private static class Configuration extends SimplePluginConfiguration
    {
        // ... other code omitted ...

        private ThirdPartyServiceModel oidcProvider;

        // ... other code omitted ...


        public ThirdPartyServiceModel getOidcProvider()
        {
            return oidcProvider;
        }

        @Inject
        @OidcProvider
        public void setOidcProvider(ThirdPartyServiceModel oidcProvider)
        {
            this.oidcProvider = oidcProvider;
        }
    }
```

The setter for the oidcProvider field is annotated with the `@OidcProvider` annotation.

If the `@OidcProvider` annotation includes a parameter, that parameter specifies a required endpoint defined by the OIDC provider metadata of the current token provider. When the plugin is instantiated, the validation for the `@OidcProvider` parameter will pass only if the specified endpoint is a valid HTTP URI in the OIDC provider metadata. For example, the following annotation will require the `backchannel_authentication` URI.

```
@OidcProvider(“backchannel_authentication”)
```

Making a HTTP call to a third-party service

With an instance of HttpClient and an instance of ThirdPartyServiceModel in hand, a plugin can make a HTTP call to the external service represented by the ThirdPartyServiceModel. Here is an example method that makes a GET request to a resource on the external service with a path of `/data` and a query string of `page=1`.

```
private static CompletionStage<ClientResponse> sendRequest(HttpClient
 httpClient,

 ThirdPartyServiceModel model)
{
    Headers headers = ClientRequest.createHeaders();
    headers.setAccept(Collections.singletonList("application/json"));

    ClientRequest request = new ClientRequest(Method.GET,
                                              "/data?page=1",
                                              headers);

    return httpClient.send(request, model);
}
```

The result of the `HttpClient send` method is a CompletionStage. A CompletionStage is returned because PingAccess is performing the HTTP call asynchronously and as a result, handling of the result of the call needs to be performed by callbacks registered with the CompletionStage.

You can use the `getRequestUri()` method to stand in for the endpoint. This can be useful if the endpoint is not known during development.

```
    ClientRequest request = new ClientRequest(Method.GET,
                                              model.getRequestUri().get(),
                                              headers);
```

The `RequestUri` is set by the `@OidcProvider("RequestUri")` annotation, and is unset if the `@OidcProvider("RequestUri")` annotation is not present.

For a more complete example of using the HttpClient to make an external HTTP call, see the sample SDK plugins packaged with the PingAccess distribution.

Base classes

The SDK provides the following base classes to make it easier to implement a plugin that leverages the HttpClient interface. They all provide access to a HttpClient instance using a getHttpClient method:

- `AsyncRuleInterceptorBase`
- `AsyncSiteAuthenticatorInterceptorBase`
- `AsyncIdentityMappingPluginBase`
- `AsyncLoadBalancingPluginBase`

Sample plugins

The use of the HttpClient and ThirdPartyServiceModel classes are demonstrated in the following samples provided in the PingAccess distribution:

RiskAuthorizationRule

A rule that obtains a risk score from an external, risk service as well as leveraging the OAuth authorization server to obtain an OAuth access token used to access the risk service.

MetricBasedPlugin

A load balancing strategy that obtains host capacity metadata from an external service.

Implementation guidelines

These sections provide specific programming guidance for developing custom interfaces.

This information is not exhaustive. Consult the Javadocs to find more details about interfaces discussed here as well as additional functionality.

> **Important:**
>
> You must restart PingAccess after the deployment of any custom plugins written in Java.

Logging

Use the SLF4j API for logging activities in your module. Documentation on using SLF4j is available on the *SLF4j website*.

Lifecycle

The plugins and the implementation of a PluginConfiguration can be instantiated for a number of reasons and at many times. For example, with a RuleInterceptor here is what happens before the RuleInterceptor is available to process user requests:

1. The rule annotation on the implementation class of the RuleInterceptor is interrogated to determine which PluginConfiguration instance will be instantiated.
2. The following is performed on RuleInterceptor and PluginConfiguration. Which of these is handled first is not defined.
    - The bean will be provided to Spring for Autowiring.
    - The bean will be provided to Spring for post construction initialization. See PostConstruct.
3. `PluginConfiguration.setName(String)` is called.
4. PingAccess attempts to map the incoming JSON configuration to the PluginConfiguration instance. The JSON plugin configuration must contain a JSON member for each field, regardless of implied value. Failure to do so can lead to errors.
5. `ConfigurablePlugin.configure(PluginConfiguration)` is called.
6. `Validator.validate(Object, Class[])` method is invoked and provided to the RuleInterceptor.
7. The instance is then made available to service end user requests, such as `RequestInterceptor.handleRequest(com.pingidentity.pa.sdk.http.Exchange)` and `ResponseInterceptor.handleResponse(com.pingidentity.pa.sdk.http.Exchange)`

Injection

Before they are put into use, rules, SiteAuthenticators, and their defined PluginConfigurations are passed through Spring's Autowiring and initialization. To future-proof any code against changes in PingAccess, do not use Spring as a dependency. Use the annotation javax.inject.Inject for any injection.

Classes available for injection

Currently, injection is available for the following classes:

- `com.pingidentity.pa.sdk.util.TemplateRenderer`
- `com.pingidentity.pa.sdk.accessor.tps.ThirdPartyServiceModel`
- `com.pingidentity.pa.sdk.http.client.HttpClient`

Differences between rules for agents and sites

Rules can be applied to applications associated with agents or sites. Some features of the SDK are not available to rules that are applied to agents. Rules that use features only available to sites should be marked as only applying to sites. This is done by setting the destination element of the rule annotation to the value `{RuleInterceptorSupportedDestination.Site}`.

Rules that apply only to agents are limited in the following ways:

- The handleResponse method is not called.
- The request body is not present.
- The `Exchange.getDestinations` list is empty and modifying the destination list has no effect.

As with rules that use features only available to sites, rules that only apply to agents should be marked as only applying to agents. To do this, set the destination element of the rule annotation to the value `{RuleInterceptorSupportedDestination.Agent}`.

PingAccess Add-On SDK for Java Migration Guide

When upgrading PingAccess, review the changes made to thePingAccess add-on SDK for Java, analyze your addons, and make any necessary changes to ensure continued compatibility.

The following sections provide a detailed description of the changes, organized by package. Where relevant, code examples show you how to port existing code to account for the changes in the SDK APIs.

> **Important:**
> Because the SDK for PingAccess 6.1 uses Java 8 features, plugins built against the Java add-on SDK for PingAccess 6.1 or later must be built with JDK 8.

Prevent modification to request in response chain

Starting in PingAccess 6.1, any modifications made to a request or its header fields during response processing will now result in a warning log message and the modification operation being ignored. Previously, PingAccess would log a warning message about the modification but still allow the modification operation to complete.

Retrieving key pair and trusted certificate group configuration data

In the previous version of the SDK, a SDK plugin accessed the configuration data of a key pair or trusted certificate group configured using the administrative API by annotating a field in the plugin's PluginConfiguration class with a JsonDeserialize annotation, specifying the appropriate custom deserializer from the SDK.

```
public class Configuration extends SimplePluginConfiguration
{
   @JsonDeserialize(using = PrivateKeyDeserializer.class)
    KeyStore.PrivateKeyEntry keyPair;

   @JsonDeserialize(using = TrustedCertificateGroupDeserializer.class)
    Collection<X509Certificate> certificateGroup;
}
```

In the current version of the SDK, this mechanism has changed to be less error-prone as well as to provide access to more properties of the key pairs and trusted certificate groups. The previous configuration class should be ported to the following:

```
public class Configuration extends SimplePluginConfiguration
{
   KeyPairModel keyPair;

   TrustedCertificateGroupModel certificateGroup;
}
```

The `KeyPairModel#getPrivateKeyEntry` method provides access to the `KeyStore.PrivateKeyEntry` object for the corresponding key pair in the administrative configuration. The `TrustedCertificateGroupModel#getCertificates` method provides access to the Collection of X509Certificate objects in the corresponding trusted certificate group in the administrative configuration. Refer to the JavaDoc for each of these classes for more information.

Related to this change, the provided implementations of ConfigurationModelAccessor, PrivateKeyAccessor and TrustedCertificateGroupAccessor, have been updated to use these new classes. Both classes have also been moved to new packages. PrivateKeyAccessor has also been renamed to KeyPairAccessor.

Before PingAccess 5.0:

```
import com.pingidentity.pa.sdk.accessor.PrivateKeyAccessor;
import
 com.pingidentity.pa.sdk.accessor.TrustedCertificateGroupAccessor;

// … class definition omitted ...

private void invokePrivateKeyAccessorGet(
       PrivateKeyAccessor accessor,
       String id)
{
   KeyStore.PrivateKeyEntry keyPair = accessor.get(id);
}
private void invokeTrustedCertificateGroupAccessorGet(
       TrustedCertificateGroupAccessor accessor,
       String id)
{
   Collection<X509Certificate> certificates = accessor.get(id);
}
```

After PingAccess 5.0:

```
import
 com.pingidentity.pa.sdk.accessor.certgroup.TrustedCertificateGroupModel;
import com.pingidentity.pa.sdk.accessor.keypair.KeyPairAccessor;

// … class definition omitted ...

private void invokePrivateKeyAccessorGet(
       KeyPairAccessor accessor,
       String id)
{
   KeyStore.PrivateKeyEntry keyPair = accessor.get(id)

 .map(KeyPairModel::getPrivateKeyEntry)
                                              .orElse(null);
}

private void invokeTrustedCertificateGroupAccessorGet(
       TrustedCertificateGroupAccessor accessor,
       String id)
{
   Collection<X509Certificate> certificates = accessor.get(id)
                   .map(TrustedCertificateGroupModel::getCertificates)
                   .orElse(null);
}
```

Changes to validation of PluginConfiguration instances

In the previous version of the SDK, the ConfigurablePlugin#configure method was invoked and passed a PluginConfiguration instance. The ConfigurablePlugin was expected to assign the specified PluginConfiguration instance to a field annotated with the javax.validation.Valid annotation. After the configure method returned, PingAccess passed the ConfigurablePlugin instance to a javax.validation.Validator for further validation.

If setup correctly, this logic allows javax.validation.Constraint annotations to declare the validation to be applied to fields in a PluginConfiguration implementation, ensuring the configuration is valid as well as providing validation error message to PingAccess to provide to administrators using the Administrative API or UI.

However, if the ConfigurablePlugin#configure method needed to post-process the specified PluginConfiguration instance, the method needed to duplicate all the validation declared on the fields of the PluginConfiguration.

To remove the need for this duplication of validation logic, PingAccess now validates the PluginConfiguration instance with a javax.validation.Validator prior to passing the instance to the ConfigurablePlugin#configure method.

Further, the ConfigurablePlugin no longer needs to annotate the field used to hold the PluginConfiguration instance. The field is still necessary to implement the ConfigurablePlugin#getConfiguration method.

The following example ConfigurablePlugin implementation demonstrates this change.

```
public class ValidationExample
       implements ConfigurablePlugin<ValidationExample.Configuration>
{
   // @Valid annotation no longer required
   private Configuration configuration;

   @Override
   public void configure(Configuration configuration) throws
 ValidationException
   {
       this.configuration = configuration;

       // With the previous version of the SDK, these assertions were not
       // guaranteed to be true, despite the javax.validation.Constraint
       // annotations enforcing these conditions.
       //
       // In the current version of the SDK, these assertions are guaranteed
       // to be true because they are enforced by the
 javax.validation.Constraint
       // annotations on the fields in the PluginConfiguration class, and
 the
       // PluginConfiguration validation is performed before invoking the
       // configure method.
       //
       // The end result is that plugins can remove duplicated validation
       // logic from the configure method if further post-processing of the
       // configuration needs to be performed.
       assert(configuration.getAttributeName() != null);
       assert(configuration.getAttributeName().length() > 0);
       assert(configuration.getAttributeName().length() <= 16);
       assert(configuration.getAttributeValue() != null);

   }

   @Override
   public Configuration getConfiguration()
   {
```

```
        return configuration;
    }

    static class Configuration extends SimplePluginConfiguration
    {
        @NotNull
        @Size(min = 1,
              max = 16,
              message = "Attribute name length must be between 1 and 16
 characters")
        private String attributeName;

        @NotNull
        private String attributeValue;

        public String getAttributeName()
        {
            return attributeName;
        }

        public void setAttributeName(String attributeName)
        {
            this.attributeName = attributeName;
        }

        public String getAttributeValue()
        {
            return attributeValue;
        }

        public void setAttributeValue(String attributeValue)
        {
            this.attributeValue = attributeValue;
        }
    }
}
```

com.pingidentity.pa.sdk.http

The body interface has changed to require an explicit read of data before invoking methods to obtain that data. Previously, methods to obtain the data would result in an implicit read of the data. The following code examples illustrate this change in semantics.

com.pingidentity.pa.sdk.http.Body

As the updated Javadocs for the body interface indicates, plugins should avoid interrogating a body object unless absolutely necessary because reading a body object's data into memory can impact the scalability of PingAccess. As the plugin code is updated, evaluate whether the body object needs to be used by the plugin.

Using the Body#read method

Before PingAccess 5.0:

```
private void invokeRead(Body body) throws IOException
{
    body.read();
}
```

After PingAccess 5.0:

```
private void invokeRead(Body body) throws AccessException
{
    try
```

```
    {
        body.read();
    }
    catch (IOException e)
    {
        throw new AccessException("Failed to read body content",
                                  HttpStatus.BAD_GATEWAY,
                                  e);
    }
}
```

Using the Body#getContent method

Before PingAccess 5.0:

```
private void invokeGetContent(Body body) throws IOException
{
    byte[] content = body.getContent();
}
```

After PingAccess 5.0:

```
private void invokeGetContent(Body body) throws AccessException
{
    invokeRead(body); // see the Body#read code example for this method
    byte[] content = body.getContent();
}
```

Using the Body#getBodyAsStream method

Before PingAccess 5.0:

```
private void invokeGetBodyAsStream(Body body) throws IOException
{
    InputStream stream = body.getBodyAsStream();
}
```

After PingAccess 5.0:

```
private void invokeGetBodyAsStream(Body body) throws AccessException
{
    invokeRead(body); // see the Body#read code example for this method
    InputStream stream = body.newInputStream();
}
```

Note:

The rename of the method from `getBodyAsStream` to `newInputStream`.

Using the Body#write method

Before PingAccess 5.0:

```
private void invokeWrite(Body body, BodyTransferrer bodyTransferrer)
 throws IOException

{

    body.write(bodyTransferrer);
```

```
}
```

After PingAccess 5.0:

This functionality is no longer supported. To obtain the content of the Body, read the content into memory using the Body#read method and then invoke Body#getContent or Body#newInputStream.

Using the Body#getLength method

Before PingAccess 5.0:

```
private void invokeGetLength(Body body) throws IOException
{
    int length = body.getLength();
}
```

After PingAccess 5.0:

```
private void invokeGetLength(Body body) throws AccessException

{

    invokeRead(body); // see the Body#read code example for this method

    int length = body.getLength();

}
```

Using the Body#getRaw method

Before PingAccess 5.0:

```
private void invokeGetRaw(Body body) throws IOException
{
    byte[] rawBody = body.getRaw();
}
```

After PingAccess 5.0:

This functionality is no longer supported. This method used to provide access to the content as it appeared on the wire, which required complicated handling if the body content used a chunked Transfer-Encoding. Use Body#getContent instead.

com.pingidentity.pa.sdk.http.BodyFactory

Using the BodyFactory#continuousBody method

Before PingAccess 5.0:

```
private void invokeContinuousBody(BodyFactory bodyFactory, byte[]
 content)
{
    Body body = bodyFactory.continuousBody(content);
}
```

After PingAccess 5.0:

```
private void invokeContinuousBody(BodyFactory bodyFactory, byte[]
 content)
{
    Body body = bodyFactory.createInMemoryBody(content);
```

```
}
```

Before PingAccess 5.0:

```
private void invokeContinuousBody(BodyFactory bodyFactory, InputStream
 in)
{
    Body body = bodyFactory.continuousBody(in);
}
```

After PingAccess 5.0:

A Body instance can no longer be created from an InputStream using the BodyFactory class. Instead, a plugin should read the contents of the InputStream into a byte array and provide the byte array to BodyFactory#createInMemoryBody.

com.pingidentity.pa.sdk.http.Constants

The constants available from this class have been removed from the SDK. Plugins using these constants should maintain their own constants with the needed values.

com.pingidentity.pa.sdk.http.Exchange

A handful of methods have been removed from the Exchange.

Further, the mechanism for storing data on the exchange through properties has been enhanced to make it easier to write type-safe code when working with Exchange properties.

Using the Exchange#getCreationTime method

Before PingAccess 5.0:

```
Calendar creationTime = exchange.getCreationTime();
```

After PingAccess 5.0:

```
Calendar creationTime = Calendar.getInstance();
creationTime.setTime(Date.from(exchange.getCreationTime()));
```

Note:

If a Calendar object is not required, consider using the Instant object returned from the getCreationTime method directly instead of converting it into a Calendar object.

Using the Exchange#getDestinations method

Before PingAccess 5.0:

```
List<String> destinations = exchange.getDestinations();
```

After PingAccess 5.0:

This functionality is no longer supported. Consider using the Exchange#getTargetHosts method to obtain similar information from the Exchange.

Using the Exchange#getOriginalHostHeader method

Before PingAccess 5.0:

```
String originalHostHeader = exchange.getOriginalHostHeader();
```

After PingAccess 5.0:

This functionality is no longer supported. Consider using the Exchange#getUserAgentHost method to obtain similar information from the Exchange. The getUserAgentHost method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

Using the Exchange#getOriginalHostHeaderHost method

Before PingAccess 5.0:

```
String host = exchange.getOriginalHostHeaderHost();
```

After PingAccess 5.0:

This functionality is no longer supported. Consider using the Exchange#getUserAgentHost method to obtain similar information from the Exchange. The getUserAgentHost method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

Using the Exchange#getOriginalHostHeaderPort method

Before PingAccess 5.0:

```
String port = exchange.getOriginalHostHeaderPort();
```

After PingAccess 5.0:

This functionality is no longer supported. Consider using the Exchange#getUserAgentHost method to obtain similar information from the Exchange. The getUserAgentHost method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

Using the Exchange#getOriginalRequestBaseUri method

Before PingAccess 5.0:

```
String originalRequestBaseUri = exchange.getOriginalRequestBaseUri();
```

After PingAccess 5.0:

This functionality is no longer supported. A possible replacement is as follows:

```
String originalRequestBaseUri = exchange.getUserAgentProtocol() +
                                "://" +
                                exchange.getUserAgentHost();
```

Using the Exchange#getProperties method

Before PingAccess 5.0:

```
Map<String, String> properties = exchange.getProperties();
```

After PingAccess 5.0:

This functionality is no longer supported. Properties should be obtained individually from the Exchange.

Using the Exchange#getRequestBaseUri method

Before PingAccess 5.0:

```
String requestBaseUri = exchange.getRequestBaseUri();
```

After PingAccess 5.0:

This functionality is no longer supported. A possible replacement is as follows.

```
String requestBaseUri = exchange.getUserAgentProtocol() +
                        "://" +
                        exchange.getUserAgentHost();
```

Using the Exchange#getRequestScheme method

Before PingAccess 5.0:

```
String requestScheme = exchange.getRequestScheme();
```

After PingAccess 5.0:

This functionality is no longer supported. A possible replacement is as follows.

```
String requestScheme = exchange.getUserAgentProtocol() + "://";
```

Using the Exchange#getUser method

Before PingAccess 5.0:

```
private void invokeSetUser(Exchange exchange, User user)
{
    exchange.setUser(user);
}
```

After PingAccess 5.0:

This functionality is no longer supported. The identity associated with an Exchange cannot be replaced.

Using the Exchange#setUser method

Before PingAccess 5.0:

```
private void invokeSetUser(Exchange exchange, User user)
{
    exchange.setUser(user);
}
```

After PingAccess 5.0:

This functionality is no longer supported. The identity associated with an Exchange cannot be replaced.

Using the Exchange#setSourceIp method

Before PingAccess 5.0:

```
private void invokeSetSourceIp(Exchange exchange, String sourceIp)
{
    exchange.setSourceIp(sourceIp);
```

```
}
```

After PingAccess 5.0:

This functionality is no longer supported. This value cannot be changed.

Using the Exchange#setProperty method

Before PingAccess 5.0:

```
private void invokeSetProperty(Exchange exchange, String propertyKey,
 String value)
{
    exchange.setProperty(propertyKey, value);
}
```

After PingAccess 5.0:

```
private void invokeSetProperty(Exchange exchange,
                               ExchangeProperty<String> propertyKey,
                               String value)
{
    exchange.setProperty(propertyKey, value);
}
```

See the Javadocs for ExchangeProperty for instructions on creating an ExchangeProperty object.

Using the Exchange#getProperty method

Before PingAccess 5.0:

```
private void invokeGetProperty(Exchange exchange, String propertyKey)
{
    Object propertyValueObj = exchange.getProperty(propertyKey);
    if (propertyValueObj instanceof String)
    {
        String propertyValue = (String) propertyValueObj;
    }
}
```

After PingAccess 5.0:

```
private void invokeGetProperty(Exchange exchange,
 ExchangeProperty<String> propertyKey)
{
    String propertyValue =
 exchange.getProperty(propertyKey).orElse(null);
}
```

Note:

Exchange#getProperty now returns an Optional object instead of the Object directly.

com.pingidentity.pa.sdk.http.Header

This deprecated class has been replaced by the Headers interface. A Headers object can be created using a HeadersFactory obtained from the ServiceFactory#headersFactory method. The majority of methods on Header have counterparts on the Headers interface. See the Javadocs for the Headers interface for more information.

com.pingidentity.pa.sdk.http.HeaderField

This class is now final and cannot be extended.

Constructing a HeaderField

Before PingAccess 5.0:

```
private HeaderField createHeaderField(String line)
{
    return new HeaderField(line);
}
```

After PingAccess 5.0:

```
private HeaderField createHeaderField(String line)
{
    String name = line.substring(0, line.indexOf(':'));
    String value = (line.substring(line.indexOf(":") + 1)).trim();

    return new HeaderField(name, value);
}
```

Note:

Parsing an HTTP header field line can be error prone, consider if the plugin can avoid having to parse an HTTP header field line.

Using the HeaderField#setHeaderName method

Before PingAccess 5.0:

```
private void invokeSetHeaderName(HeaderField field)
{
    field.setHeaderName(new HeaderName("X-Custom"));
}
```

After PingAccess 5.0:

This functionality is no longer supported. A HeaderField's name is set upon construction and cannot be changed.

Using the HeaderField#getApproximateSize method

Before PingAccess 5.0:

```
int approximateSize = field.getApproximateSize();
```

After PingAccess 5.0:

This method has been removed. The value returned by the method can still be computed:

```
int approximateSize = 2 * (4 +
                           field.getHeaderName().toString().length() +
                           field.getValue().length());
```

com.pingidentity.pa.sdk.http.Headers

A few methods on the Headers interface have been updated to use the instant class, instead of date.

Using the Headers#getDate method

Before PingAccess 5.0:

```
Date date = headers.getDate();
```

After PingAccess 5.0:

```
Date date = Date.from(headers.getDate());
```

Using the Headers#setDate method

Before PingAccess 5.0:

```
private void invokeSetDate(Headers headers, Date date)
{
    headers.setDate(date);
}
```

After PingAccess 5.0:

```
private void invokeSetDate(Headers headers, Date date)
{
    headers.setDate(date.toInstant());
}
```

Using the Headers#getLastModified method

Before PingAccess 5.0:

```
SimpleDateFormat format = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss
 z",
                                               Locale.ENGLISH);

String lastModified = headers.getLastModified();
if (lastModified != null)
{
    Date lastModifiedDate = format.parse(lastModified);
}
```

After PingAccess 5.0:

```
Date lastModifiedDate = Date.from(headers.getLastModified());
```

Using the Headers#setLastModified method

Before PingAccess 5.0:

```
private void invokeSetLastModified(Headers headers, Date date)
{
    SimpleDateFormat format = new SimpleDateFormat("E, dd MMM yyyy
 HH:mm:ss z",
                                                   Locale.ENGLISH);

    headers.setLastModified(format.format(date));
}
```

After PingAccess 5.0:

```
private void invokeSetLastModified(Headers headers, Date date)
{
```

```
    headers.setLastModified(date.toInstant());
}
```

com.pingidentity.pa.sdk.http.HeadersFactory

Using the HeadersFactory#createFromRawHeaderFields method

Before PingAccess 5.0:

```
private void invokeCreateFromRawHeaderFields(HeadersFactory factory,
                                             List<String> fields)
 throws ParseException
{
    Headers headers = factory.createFromRawHeaderFields(fields);
}
```

After PingAccess 5.0:

This functionality is no longer supported. Consider if the plugin can create HeaderFields directly and utilize the HeadersFactory#create method.

com.pingidentity.pa.sdk.http.HttpStatus

The HttpStatus enum was converted to a final class. Common HttpStatus instances are defined as constants on HttpStatus.

Using the HttpStatus#getLocalizationKey method

Before PingAccess 5.0:

```
String localizationKey = status.getLocalizationKey();
```

After PingAccess 5.0:

This functionality is no longer supported. Instead, a HttpStatus contains a LocalizedMessage instance that encapsulates the localization of the status message for use in error templates.

com.pingidentity.pa.sdk.http.MimeType

The constants available in this class are now available as constant MediaType instances in the class com.pingidentity.pa.sdk.http.CommonMediaTypes.

com.pingidentity.pa.sdk.http.MediaType

This class is now final and cannot be extended.

Constructing a MediaType

Before PingAccess 5.0:

```
private void createMediaType(String mediaTypeString)
{
    MediaType mediaType = new MediaType(mediaTypeString);
}
```

After PingAccess 5.0:

```
private void createMediaType(String mediaTypeString)
{
    MediaType mediaType = MediaType.parse(mediaTypeString);
```

```
}
```

com.pingidentity.pa.sdk.http.Message

A number of methods have been removed from the Message interface.

Using the Message#getBodyAsStream method

Before PingAccess 5.0:

```
InputStream bodyStream = message.getBodyAsStream();
```

After PingAccess 5.0:

This functionality is no longer supported. However, the following code snippet can be used to maintain semantics of the old method.

```
Body body = message.getBody();
try
{
    body.read();
}
catch (IOException | AccessException e)
{
    throw new RuntimeException("Could not get body as stream", e);
}

InputStream bodyStream = body.newInputStream();
```

While this snippet maintains semantics, enable a plugin to propagate errors as an AccessException instead of as a RuntimeException.

Using the Message#getCharset method

Before PingAccess 5.0:

```
Charset charset = message.getCharset();
```

After PingAccess 5.0:

This functionality is no longer supported. However, the following code snippet can be used to maintain semantics of the old method.

```
Charset charset = message.getHeaders().getCharset();
if (charset == null)
{
    charset = StandardCharsets.UTF_8;
}
```

While this snippet maintains semantics, a plugin should consider how to handle the case where a Charset is not specified by a Message's header fields. Assuming a Charset of UTF-8 might lead to issues in some cases.

Using the Message#getHeader method

Before PingAccess 5.0:

```
Header header = message.getHeader();
```

After PingAccess 5.0:

This functionality is no longer supported. Instead, use Message#getHeaders and the Headers interface instead of Header.

Using the Message#setHeader method

Before PingAccess 5.0:

```
private void invokeSetHeader(Message message, Header header)
{
    message.setHeader(header);
}
```

After PingAccess 5.0:

This functionality is no longer supported. Instead, use Message#setHeaders and the Headers interface instead of Header.

Using the Message#isDeflate method

Before PingAccess 5.0:

```
boolean deflate = message.isDeflate();
```

After PingAccess 5.0:

This method has been removed. However, the value can still be computed with the following code snippet.

```
List<String> contentEncodingValues =
 message.getHeaders().getContentEncoding();
boolean deflate = contentEncodingValues.stream().anyMatch(v ->
 v.equalsIgnoreCase("deflate"))
                  && contentEncodingValues.size() == 1;
```

Using the Message#isGzip method

Before PingAccess 5.0:

```
boolean gzip = message.isGzip();
```

After PingAccess 5.0:

This method has been removed. However, the value can still be computed with the following code snippet.

```
List<String> contentEncodingValues =
 message.getHeaders().getContentEncoding();
boolean gzip = contentEncodingValues.stream().anyMatch(v ->
 v.equalsIgnoreCase("gzip"))
               && contentEncodingValues.size() == 1;
```

Using the Message#isHTTP10 method

Before PingAccess 5.0:

```
boolean http10 = message.isHTTP10();
```

After PingAccess 5.0:

This method has been removed. However, the value can still be computed with the following code snippet.

```
boolean http10 = message.getVersion().equals("1.0");
```

Using the Message#isHTTP11 method

Before PingAccess 5.0:

```
boolean http11 = message.isHTTP11();
```

After PingAccess 5.0:

The method has been removed. However, the value can still be computed with the following code snippet.

```
boolean http11 = message.getVersion().equals("1.1");
```

Using the Message#read method

Before PingAccess 5.0:

```
private void invokeRead(Message message,
                        InputStream inputStream,
                        boolean createBody) throws IOException
{
    message.read(inputStream, createBody);
}
```

After PingAccess 5.0:

This functionality is no longer supported. A request attached to an exchange can no longer be completely replaced, but individual components can be replaced, such as the method, URI, headers and body. A response attached to an exchange can be replaced by using Exchange#setResponse.

Using the Message#setVersion method

Before PingAccess 5.0:

```
private void invokeSetVersion(Message message, String version)
{
    message.setVersion(version);
}
```

After PingAccess 5.0:

This functionality is no longer supported. The version of a message cannot be changed.

Using the Message#write method

Before PingAccess 5.0:

```
private void invokeWrite(Message message,
                         OutputStream output) throws IOException
{
    message.write(output);
}
```

After PingAccess 5.0:

This functionality is no longer supported. However, the following code snippet can be used to perform the equivalent operation.

```
private void invokeWrite(Message message,
                         OutputStream output) throws IOException,
 AccessException
{
    Body body = message.getBody();
    body.read();


 output.write(message.getStartLine().getBytes(StandardCharsets.ISO_8859_1));

 output.write(message.getHeaders().toString().getBytes(StandardCharsets.ISO_8859_1)
    output.write("\r\n".getBytes(StandardCharsets.ISO_8859_1));
    output.write(body.getContent());
    output.flush();
}
```

com.pingidentity.pa.sdk.http.Method

The method interface has been converted to a final class. Additionally, the related methods enum has been merged into the method class. The method class provides common method instances as class-level constants.

Obtaining a common Method instance

Before PingAccess 5.0:

```
Method get = Methods.GET
```

After PingAccess 5.0:

```
Method get = Method.GET;
```

Using the Method#getMethodName method

Before PingAccess 5.0:

```
String methodName = method.getMethodName();
```

After PingAccess 5.0:

```
String methodName = method.getName();
```

com.pingidentity.pa.sdk.http.Request

A few methods have been removed from the request interface.

Using the Request#getPostParams method

Before PingAccess 5.0:

```
private void invokeGetPostParams(Request request) throws IOException
{
    Map<String, String[]> postParams = request.getPostParams();
}
```

After PingAccess 5.0:

```
private void invokeGetPostParams(Request request) throws
 AccessException
{
    Body body = request.getBody();
    try
    {
        body.read();
    }
    catch (IOException e)
    {
        throw new AccessException("Failed to read body content",
                                  HttpStatus.BAD_GATEWAY,
                                  e);
    }

    Map<String, String[]> postParams = body.parseFormParams();
}
```

Using the Request#isMultipartFormPost method

Before PingAccess 5.0:

```
boolean multipartFormPost = request.isMultipartFormPost();
```

After PingAccess 5.0:

This method has been removed from the Request interface. However, the value can still be calculated using the following code snippet.

```
Headers headers = request.getHeaders();

boolean multipartFormPost =
        request.getMethod() == Method.POST
        && headers.getContentType() != null
        && headers.getContentType().getBaseType().equals("multipart/
form-data")
        && headers.getContentType().getParameter("boundary") != null;
```

com.pingidentity.pa.sdk.http.ResponseBuilder

A handful of methods were removed from ResponseBuilder. Additionally, a handful of methods have changed their semantics, particularly those that included an HTML message payload. See the updated Javadocs for ResponseBuilder for more info.

Using the ResponseBuilder#badRequestText method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.badRequestText(message).build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.newInstance(HttpStatus.BAD_REQUEST)

 .contentType(CommonMediaTypes.TEXT_PLAIN)
                                   .body(message)
                                   .build();
```

Note:

This approach does not localize the response body. Using a TemplateRenderer is recommended instead.

Using the ResponseBuilder#contentLength method

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance().contentLength(length).build();
```

After PingAccess 5.0:

This functionality is no longer supported. Consider using one of the ResponseBuilder#body methods instead of explicitly setting the content length. This ensures that the body content of the Response aligns with the Content-Length header field.

Using the ResponseBuilder#continue100 method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.continue100().build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance(HttpStatus.CONTINUE).build();
```

Using the ResponseBuilder#forbiddenText method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.forbiddenText().build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.newInstance(HttpStatus.FORBIDDEN)

  .contentType(CommonMediaTypes.TEXT_PLAIN)

  .body(HttpStatus.FORBIDDEN.getMessage())
                                  .build();
```

Note:

This approach does not localize the response body. Use a TemplateRenderer instead.

Using the ResponseBuilder#forbiddenWithoutBody method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.forbiddenWithoutBody().build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance(HttpStatus.FORBIDDEN).build();
```

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.forbiddenWithoutBody(message).build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance(HttpStatus.FORBIDDEN).build();
```

> **Note:**
>
> In the original method, the string message parameter was not used.

Using the ResponseBuilder#htmlMessage method

Before PingAccess 5.0:

```
String message = ResponseBuilder.htmlMessage(caption, text);
```

After PingAccess 5.0:

This functionality is no longer supported. Plugins that used this method will need to construct the HTML message without this method. Consider using the TemplateRenderer utility class in place of this method.

Using the ResponseBuilder#internalServerError method

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.internalServerError(message).build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.internalServerError().body(message).build();
```

> **Note:**
>
> This approach does not localize the response body. Use a TemplateRenderer instead.

Using the ResponseBuilder#internalServerErrorWithoutBody method

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.internalServerErrorWithoutBody().build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.internalServerError().build();
```

Using the ResponseBuilder#newInstance method

The no-arg newInstance method has been removed. A HttpStatus is required to create an instance of ResponseBuilder, and the required HttpStatus object should be passed to the newInstance method that accepts a HttpStatus.

Before PingAccess 5.0:

```
Response response = ResponseBuilder.newInstance().build()
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

Using the ResponseBuilder#noContent method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.noContent().build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance(HttpStatus.NO_CONTENT).build();
```

Using the ResponseBuilder#notFoundWithoutBody method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.notFoundWithoutBody().build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.notFound().build();
```

Using the ResponseBuilder#serverUnavailable method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.serverUnavailable(message).build();
```

After PingAccess 5.0:

```
Response response =
 ResponseBuilder.serviceUnavailable().body(message).build();
```

Note:

This approach does not localize the response body. Use a TemplateRenderer instead.

Using the ResponseBuilder#serviceUnavailableWithoutBody method

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.serverUnavailableWithoutBody().build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.serviceUnavailable().build();
```

Using the ResponseBuilder#status method

The status methods have been removed. Instead the status should be specified to the newInstance method as it is now required.

Before PingAccess 5.0:

```
Response response =
 ResponseBuilder.newInstance().status(HttpStatus.OK).build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.newInstance(HttpStatus.OK).build();
```

Using the ResponseBuilder#unauthorizedWithoutBody method

Before PingAccess 5.0:

```
Response response = ResponseBuilder.unauthorizedWithoutBody().build();
```

After PingAccess 5.0:

```
Response response = ResponseBuilder.unauthorized().build();
```

com.pingidentity.pa.sdk.http.Response

A few methods were removed from the response interface.

Using the Response#isRedirect method

Before PingAccess 5.0:

```
boolean redirect = response.isRedirect();
```

After PingAccess 5.0:

```
boolean redirect = response.getStatusCode() >= 300
                   && response.getStatusCode() < 400;
```

Using the Response#setStatusCode method

Before PingAccess 5.0:

```
response.setStatusCode(HttpStatus.OK.getCode());
```

After PingAccess 5.0:

```
response.setStatus(HttpStatus.OK);
```

Using the Response#setStatusMessage method

Before PingAccess 5.0:

```
response.setStatusMessage(HttpStatus.OK.getMessage());
```

After PingAccess 5.0:

```
response.setStatus(HttpStatus.OK);
```

com.pingidentity.pa.sdk.identity

com.pingidentity.pa.sdk.identity.Identity

The getTokenExpiration method was updated to use an instant instead of date.

Using the Identity#getTokenExpiration method

Before PingAccess 5.0:

```
Date expiration = identity.getTokenExpiration();
```

After PingAccess 5.0:

```
Date expiration = Date.from(identity.getTokenExpiration());
```

com.pingidentity.pa.sdk.identity.OAuthTokenMetadata

The OAuthTokenMetadata methods now use an instant instead of a date.

Using the OAuthTokenMetadata#getExpiresAt method

Before PingAccess 5.0:

```
Date expiresAt = metadata.getExpiresAt();
```

After PingAccess 5.0:

```
Date expiresAt = Date.from(metadata.getExpiresAt());
```

Using the OAuthTokenMetadata#getRetrievedAt method

Before PingAccess 5.0:

```
Date retrievedAt = metadata.getRetrievedAt();
```

After PingAccess 5.0:

```
Date retrievedAt = Date.from(metadata.getRetrievedAt());
```

com.pingidentity.pa.sdk.identitymapping.header

ClientCertificateMapping has been removed from the SDK, as it was not required to create an IdentityMappingPlugin implementation.

Plugins utilizing this class should create their own version of this class.

com.pingidentity.pa.sdk.policy

com.pingidentity.pa.sdk.policy.AccessExceptionContext

The nested Builder class has been removed from AccessExceptionContext and instead AccessExceptionContext is a builder that can be initially created with the new AccessExceptionContext#create method.

The LocalizedMessage interface has been introduced to simplify the configuration of a localized message for use in an error template. A LocalizedMessage has three implementations provided in the SDK: FixedMessage, BasicLocalizedMessage and ParameterizedLocalizedMessage. See the following code examples for more information on using these new classes.

Constructing an AccessExceptionContext

Before PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                              Throwable
 cause)

{
    return AccessExceptionContext.builder()
                                  .cause(cause)
                                  .httpStatus(httpStatus)

 .exceptionMessage(httpStatus.getMessage())

 .errorDescription(httpStatus.getLocalizationKey())
                                  .errorDescriptionIsKey(true)
                                  .errorDescriptionSubstitutions(new
 String[0])
                                  .build();
}
```

After PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                              Throwable
 cause)
{
    return AccessExceptionContext.create(httpStatus)
                                  .cause(cause)

 .exceptionMessage(httpStatus.getMessage())

 .errorDescription(httpStatus.getLocalizedMessage());
}
```

Before PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                              String
 localizationKey,
                                                              String[]
 substitutions)
{
    return AccessExceptionContext.builder()
                                  .httpStatus(httpStatus)
```

```
                                 .errorDescription(localizationKey)
                                 .errorDescriptionIsKey(true)

 .errorDescriptionSubstitutions(substitutions)
                                 .build();
}
```

After PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                            String
 localizationKey,
                                                            String[]
 substitutions)
{
    LocalizedMessage localizedMessage =
            new ParameterizedLocalizedMessage(localizationKey,
 substitutions);

    return AccessExceptionContext.create(httpStatus)
                                 .errorDescription(localizedMessage);
}
```

Before PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                            String
 localizationKey)
{
    return AccessExceptionContext.builder()
                                 .httpStatus(httpStatus)
                                 .errorDescription(localizationKey)
                                 .errorDescriptionIsKey(true)
                                 .build();
}
```

After PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus,
                                                            String
 localizationKey)
{
    LocalizedMessage localizedMessage = new
 BasicLocalizedMessage(localizationKey);
    return AccessExceptionContext.create(httpStatus)
                                 .errorDescription(localizedMessage);
}
```

Before PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus)
{
    return AccessExceptionContext.builder()
                                 .from(httpStatus)

 .httpStatusDescription(httpStatus.getLocalizationKey())
                                 .httpStatusDescriptionIsKey(true)
                                 .templateFile("template.html")
```

```
                                 .contentType("text/html");
}
```

After PingAccess 5.0:

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
 httpStatus)
{
    return AccessExceptionContext.create(httpStatus)

 .errorDescription(httpStatus.getLocalizedMessage());
}
```

Note:

This example demonstrates that it is no longer possible to set a template file and its associated content type on an AccessExceptionContext. To generate an error response from a template file, use the TemplateRenderer class. See the Javadocs for the TemplateRenderer class for more information.

com.pingidentity.pa.sdk.policy.AccessException

The changes to AccessExceptionContext apply to the creation of AccessException because the creation of an AccessException requires an AccessExceptionContext.

In addition to these changes, obtaining information from AccessException has also changed. See the code examples below for more information.

Finally, AccessException no longer derives from IOException and derives directly from Exception instead.

Constructing an AccessException

Before PingAccess 5.0:

```
private void throwAccessException(String errorDescription,
                                  Throwable throwable) throws
 AccessException
{
    throw new AccessException(errorDescription, throwable);
}
```

After PingAccess 5.0:

```
private void throwAccessException(String errorDescription,
                                  Throwable throwable) throws
 AccessException
{
    LocalizedMessage templateMessaage = new
 FixedMessage(errorDescription);
    throw new
 AccessException(AccessExceptionContext.create(HttpStatus.INTERNAL_SERVER_ERROR)

 .exceptionMessage(errorDescription)
                                                      .cause(throwable)

 .errorDescription(templateMessaage));
}
```

Before PingAccess 5.0:

```
private void throwAccessException(String errorDescription) throws
 AccessException
{
    throw new AccessException(errorDescription);
}
```

After PingAccess 5.0:

```
private void throwAccessException(String errorDescription) throws
 AccessException
{
    LocalizedMessage templateMessage = new
 FixedMessage(errorDescription);
    throw new
 AccessException(AccessExceptionContext.create(HttpStatus.INTERNAL_SERVER_ERROR)

 .exceptionMessage(errorDescription)

 .errorDescription(templateMessage));
}
```

Before PingAccess 5.0:

```
private void createAccessException(int statusCode,
                                   String statusMessage,
                                   String errorDescription) throws
 AccessException
{
    throw new AccessException(statusCode, statusMessage,
 errorDescription);
}
```

After PingAccess 5.0:

```
private void createAccessException(int statusCode,
                                   String statusMessage,
                                   String errorDescription) throws
 AccessException
{
    HttpStatus httpStatus = new HttpStatus(statusCode, statusMessage);
    LocalizedMessage templateMessage = new
 FixedMessage(errorDescription);
    throw new AccessException(AccessExceptionContext.create(httpStatus)

 .exceptionMessage(errorDescription)

 .errorDescription(templateMessage));
}
```

Before PingAccess 5.0:

```
private void throwAccessException(int statusCode,
                                  String statusMessage,
                                  String errorDescription,
                                  Throwable throwable) throws
 AccessException
{
    throw new AccessException(statusCode, statusMessage,
 errorDescription, throwable);
```

```
}
```

After PingAccess 5.0:

```
private void throwAccessException(int statusCode,
                                  String statusMessage,
                                  String errorDescription,
                                  Throwable throwable) throws
 AccessException
{
    HttpStatus httpStatus = new HttpStatus(statusCode, statusMessage);
    LocalizedMessage templateMessage = new
 FixedMessage(errorDescription);
    throw new AccessException(AccessExceptionContext.create(httpStatus)

 .exceptionMessage(errorDescription)

 .errorDescription(templateMessage)
                                                     .cause(throwable));
}
```

Before PingAccess 5.0:

```
private void throwAccessException() throws AccessException
{
    throw new AccessException(AccessExceptionContext.builder()

 .httpStatusCode(403)

 .httpStatusMessage("Forbidden")
                                                     .build());
}
```

After PingAccess 5.0:

```
private void throwAccessException() throws AccessException
{
    throw new
 AccessException(AccessExceptionContext.create(HttpStatus.FORBIDDEN));
}
```

Using the AccessException#getExceptionContext method

Before PingAccess 5.0:

```
AccessExceptionContext context = accessException.getExceptionContext();
```

After PingAccess 5.0:

This functionality is no longer supported. The information that used to be provided by the AccessExceptionContext is now provided directly by an AccessException.

Using the AccessException#getHttpStatusCode method

Before PingAccess 5.0:

```
int statusCode = accessException.getHttpStatusCode();
```

After PingAccess 5.0:

```
int statusCode = accessException.getErrorStatus().getCode();
```

Using the AccessException#getHttpStatusMessage method

Before PingAccess 5.0:

```
String statusMessage = accessException.getHttpStatusMessage();
```

After PingAccess 5.0:

```
String statusMessage = accessException.getErrorStatus().getMessage();
```

Using the AccessException#setHttpStatusCode method

Before PingAccess 5.0:

```
accessException.setHttpStatusCode(statusCode);
```

After PingAccess 5.0:

This functionality is no longer supported. The status code associated with an AccessException is fixed once it is constructed.

Using the AccessException#setHttpStatusMessage method

Before PingAccess 5.0:

```
accessException.setHttpStatusMessage(statusMessage);
```

After PingAccess 5.0:

This functionality is no longer supported. The status message associated with an AccessException is fixed once it is constructed.

com.pingidentity.pa.sdk.policy.RuleInterceptor

The handleRequest and handleResponse methods on a RuleInterceptor no longer throw an IOException. Instead, they throw an AccessException, which no longer derives from IOException.

Accounting for the RuleInterceptor#handleRequest method signature change

Before PingAccess 5.0:

```
@Override
public Outcome handleRequest(Exchange exchange) throws IOException
{
    Outcome outcome = applyPolicy(exchange);

    return outcome;
}
```

After PingAccess 5.0:

```
@Override
public Outcome handleRequest(Exchange exchange) throws AccessException
{
    Outcome outcome = applyPolicy(exchange);

    return outcome;
}
```

Account for the RuleInterceptor#handleResponse method signature change

Before PingAccess 5.0:

```
@Override

public void handleResponse(Exchange exchange) throws IOException

{

    applyPolicyToResponse(exchange.getResponse());

}
```

After PingAccess 5.0:

```
@Override
public void handleResponse(Exchange exchange) throws AccessException
{
    applyPolicyToResponse(exchange.getResponse());
}
```

com.pingidentity.pa.sdk.policy.error.InternalServerErrorCallback

This class has been removed. Use LocalizedInternalServerErrorCallback instead.

com.pingidentity.pa.sdk.services

com.pingidentity.pa.sdk.services.ServiceFactory

This class is now final and cannot be extended.

com.pingidentity.pa.sdk.siteauthenticator

com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptor

This interface is no longer a RequestInterceptor or ResponseInterceptor, but it still defines the handleRequest and handleResponse methods.

```
public interface SiteAuthenticatorInterceptor<T extends PluginConfiguration>
        extends DescribesUIConfigurable, ConfigurablePlugin<T>
{
    void handleRequest(Exchange exchange) throws AccessException;
    void handleResponse(Exchange exchange) throws AccessException;
}
```

Additionally, these methods now only throw an AccessException instead of an IOException or InterruptedException.

Accounting for the SiteAuthenticatorInterceptor#handleRequest method signature change

Before PingAccess 5.0:

```
@Override
public Outcome handleRequest(Exchange exc)
        throws RuntimeException, IOException, InterruptedException
{
    // Site authenticator implementation //

    return Outcome.CONTINUE;
}
```

After PingAccess 5.0:

```
@Override
public void handleRequest(Exchange exc) throws AccessException
{
    // Site authenticator implementation //
}
```

Accounting for the SiteAuthenticatorInterceptor#handleResponse method signature chang

Before PingAccess 5.0:

```
@Override
public void handleResponse(Exchange exc) throws IOException
{
    // Site authenticator response implementation //
}
```

After PingAccess 5.0:

```
@Override
public void handleResponse(Exchange exc) throws AccessException
{
    // Site authenticator response implementation //
}
```

com.pingidentity.pa.sdk.ui

com.pingidentity.pa.sdk.ui.ConfigurationType

The deprecated PRIVATEKEY enum value has been removed. Use a ConfigurationType of ConfigurationType#SELECT and specify the PrivateKeyAccessor.class instance to ConfigurationBuilder#dynamicOptions or UIElement#modelAccessor.

com.pingidentity.pa.sdk.user

com.pingidentity.pa.sdk.user.User

This class has been removed from the SDK. Use the identity interface instead. An instance of identity can be retrieved from the exchange, similar to the user interface.

com.pingidentity.pa.sdk.util

com.pingidentity.pa.sdk.util.TemplateRenderer

The semantics of the renderResponse method have changed so it produces a response and does not have any side-effects on the specified parameters.

Before PingAccess 5.0:

```
private void invokeRenderResponse(TemplateRenderer templateRenderer,
                                  Map<String, String> context,
                                  String templateName,
                                  Exchange exchange,
                                  ResponseBuilder builder)
{
    templateRenderer.renderResponse(context, templateName, exchange,
 builder);
```

```
}
```

After PingAccess 5.0:

```
private void invokeRenderResponse(TemplateRenderer templateRenderer,
                                  Map<String, String> context,
                                  String templateName,
                                  Exchange exchange,8
                                  ResponseBuilder builder)
{
    Response response = templateRenderer.renderResponse(exchange,
                                                        context,
                                                        templateName,
                                                        builder);
    exchange.setResponse(response);
}
```

iovation FraudForce Integration

The iovation FraudForce integration lets you supply data to iovation and allow or deny access based on an iovation result.

The iovation FraudForce integration must be installed on each node in the deployment. Once installed, it creates two new rules, which you can use to perform FraudForce checking and grant or deny access based on FraudForce's results.

The first rule, the iovation FraudForce Device Profiling rule, lets you gather data about the end user's system. This rule must be invoked before a request that uses the iovation FraudForce Authorization rule. It cannot be used for POST requests, and only functions on requests from a top-level browsing context.

The second rule, the iovation FraudForce Authorization rule, lets you allow or deny access based on FraudForce's evaluation of the user's system. This rule must be invoked after the iovation FraudForce Device Profiling rule, within the time period defined by the **Blackbox time to live (sec.)** field.

Both rules require authentication, so they can only be used on protected applications and resources. The rules are only applicable to Web applications.

Enable logging for iovation events by updating the `<PA_HOME>/conf/log4j2.xml` file.

You can improve the accessibility of iovation, even to users that block third-party content, by configuring a reverse proxy for communicating with iovation.

The following topics are discussed in this guide:

Installing the iovation FraudForce integration

You can install the iovation FraudForce integration in your environment to enable iovation FraudForce rules.

Steps

1. Download the iovation integration bundle from the *PingAccess downloads page*.
2. Stop PingAccess.

3. From the `.zip` file, copy the integration kit files to the PingAccess directory.
    a. Copy the `dist/pa-iovation-rules-`*`version`*`.jar` file to the *`<PA_HOME>`*`/deploy` directory.
    b. Copy the contents of the `dist/conf` directory to the *`<PA_Home>`*`/conf` directory.

Note:
Preserve the relative location of each file within its subdirectory.

4. Edit the `iovation-messages.properties` file and copy its contents into the `pa-messages.properties` file.

Note:
You can edit the values of the new properties as needed.

5. Start PingAccess.
6. If you operate PingAccess in a cluster, repeat steps 2-5 for each node.

Creating iovation FraudForce device profiling rules

Create a rule to perform iovation FraudForce device profiling on the end user's system.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** list, select **iovation FraudForce device profiling**.
5. In the **Blackbox cookie name prefix** field, enter the prefix of the cookies containing the iovation blackbox captured by this rule. The default value is `iovation_bb`.

    enter the third-party service to use for fraud checks to iovation
6. In the **iovation URI hook** field, enter the location from which to load the iovation blackbox collection JavaScript. This value can be a relative or absolute URL path, but cannot be a complete URL containing scheme and authority. The default value is `/iojs`.

Note:
To unconditionally disable the first-party iovation JavaScript, set this value to *`<Reserved Application Context Root>`*`/iojs`, where *<Reserved Application Context Root>* is the context root of the reserved PingAccess application (`/pa` by default).

7. Optional: If additional options need to be configured, click **Show Advanced**.
    a. In the **Blackbox time to live (sec.)** field, enter the number of seconds to use the iovation blackbox during a session before refreshing the blackbox device profile. The default value is `300`.
    b. Optional: In the **iovation subkey** field, enter the subkey value supplied to you by iovation.

    Note:
    This value is used for debugging and troubleshooting purposes.

    c. In the **iovation script version** field, enter the version of the iovation Device Recognition JavaScript to use. The default value is `general5`.
    d. Optional: Check the **Overwrite existing blackbox** checkbox to perform blackbox device profiling with every request to resources using this rule.

        When unchecked, the rule only collects the device profile after the existing blackbox expires. This option is unchecked by default.

8. Click **Save**.

Creating Iovation FraudForce authorization rules

Create a rule to share device information with iovation FraudForce and allow or deny access based on the response.

About this task

When this rule runs, the iovation response is stored in the `com.pingidentity.pa.iovation.kit:policy.decision.outcome` property. Valid values are `allow`, `deny`, and `review`. This property can be used by Groovy rules or custom plugins.

Steps

1. Click **Access** and then go to **Rules** # **Rules**.
2. Click **+ Add Rule**.
3. In the **Name** field, enter a unique name. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. From the **Type** list, select **Iovation FraudForce authorization**.
5. From the **iovation Service** list, select the third-party service to use for outbound fraud checks to iovation.
6. In the **Blackbox Cookie Name Prefix** field, enter the prefix of the cookies containing the iovation blackbox captured previously by the iovation FraudForce Device Profiling rule. The default value is `iovation_bb`.
7. In the **Subscriber ID** field, enter the subscriber ID provided to you by iovation.
8. In the **Subscriber Account** field, enter the subscriber account name provided to you by iovation.
9. In the **Subscriber Passcode** field, enter the passcode used to authorize your ID and account with iovation.
10. In the **iovation Integration Point** field, enter the integration point associated with the rule set you want to use.
11. Optional: In the **Account Code Attribute** field, enter the name of an attribute containing a unique identifier for each end-user to send to iovation as the account code.
12. Optional: In the **Transaction Insight Parameter Mappings** section, enter one or more mappings of identity attributes in PingAccess to iovation Transaction Insight Parameters. The attributes are provided to iovation in the specified parameters.
    a. In the **Attribute Name** field, enter the attribute to use as a source.
    b. In the **Transaction Insight Parameter** field, enter the iovation Transaction Insight Parameter to use for the specified attribute.
    c. Optional: Click **Add Row** to add one or more additional mappings.
13. If additional options need to be configured, click **Show Advanced**.

Advanced Option	Description
Fraud Check Frequency (ms)	The number of milliseconds between iovation fraud checks. The default value is `20000`.
iovation Fraud Check API Endpoint	The API endpoint to which iovation fraud check requests are directed. If not specified, a value of `/fraud/v1/subs/`*`subscriberId`*`/checks` is used, where *subscriberId* is the value in the **Subscriber ID** field.
iovation Failure Mode	Specifies whether PingAccess should allow or deny access if the communication with iovation is

Advanced Option	Description
	not completed successfully. The default value is **Deny**.
Invalid Blackbox Failure Mode	Specifies whether PingAccess should allow or deny access if the blackbox device profile is not in a usable state. This situation can occur when the blackbox has not already been collected from a previous exchange processed by this rule or when the collected blackbox has reached the end of its lifetime. The default value is **Deny**, which denies access. A value of **Continue** performs a risk assessment with no blackbox profile, while a value of **Allow** allows access.
iovation Protocol Error Handling	This section specifies the error parameters to use on a failure if there is a failure to communicate with iovation for the fraud check API request. In the **Error Response Code** field, enter the HTTP response code for the error response. In the **Error Response Template File** field, you can enter the name of a customized error page template if you do not want to use the default error page. Templates are stored in the `<PA_HOME>/conf/template/` directory. In the **Error Response Content Type** field, you can specify the content type if you are using a custom error response template file.
Review Fallback Type	Specifies whether PingAccess should allow or deny access if iovation returns a review result from the risk assessment. The default value is **Deny**.
Review Deny Handling	This section specifies the error parameters to use on a failure if the **Review Fallback Type** is set to **Deny**. In the **Error Response Code** field, enter the HTTP response code for the error response. In the **Error Response Template File** field, you can enter the name of a customized error page template if you do not want to use the default error page. Templates are stored in the `<PA_HOME>/conf/template/` directory. In the **Error Response Content Type** field, you can specify the content type if you are using a custom error response template file.
Deny Handling	This section specifies the error parameters to use on a failure if iovation returns a Deny (D) result

Advanced Option	Description
	or when the blackbox is not set and the **Invalid Blackbox Failure Mode** is set to Deny. In the **Error Response Code** field, enter the HTTP response code for the error response. In the **Error Response Template File** field, you can enter the name of a customized error page template if you do not want to use the default error page. Templates are stored in the *<PA_HOME>*/conf/template/ directory. In the **Error Response Content Type** field, you can specify the content type if you are using a custom error response template file.

14. Click **Save**.

Logging iovation events

Update the PingAccess logging file to log iovation events.

About this task

This procedure modifies the existing *<PA_HOME>*/conf/log4j2.xml file to log communications with iovation to a new log file. In a clustered environment, you must perform this procedure on every node.

Steps

1. Edit the *<PA_HOME>*/conf/log4j2.xml file.
2. Locate the Appenders section and add a section to create the new log file.

   Example:

```
<RollingFile name="Iovation-File"
                     fileName="${sys:pa.home}/log/
pingaccess_iovation_audit.log"
                     filePattern="${sys:pa.home}/log/
pingaccess_iovation_audit.%d{yyyy-MM-dd}.log"
                     ignoreExceptions="false">
            <PatternLayout>
                <pattern>%d{ISO8601}| %X{exchangeId}|
 %X{IOVATION_AUDIT.trackingNumber} | %X{IOVATION_AUDIT.deviceAlias}
 | %X{IOVATION_AUDIT.accountCode} | %X{IOVATION_AUDIT.result}
 | %X{IOVATION_AUDIT.reason} | %X{IOVATION_AUDIT.ruleName} |
 %X{IOVATION_AUDIT.iovationId} | %X{IOVATION_AUDIT.statedIp} %n</pattern>
            </PatternLayout>
            <Policies>
                <TimeBasedTriggeringPolicy/>
            </Policies>
</RollingFile>
```

   This example uses a log file name of *<PA_HOME>*/log/pingaccess_iovation_audit.log.

   The following variables are used in this example.

Variable	Definition
%d	The transaction time.
exchangeId	The ID for a specific request/response pair.

Variable	Definition
`IOVATION_AUDIT.trackingNumber`	An iovation-assigned unique ID for the transaction that can be used to locate the transaction in searches and reports.
`IOVATION_AUDIT.deviceAlias`	The iovation identifier for the requesting device. If no blackbox is present at the time of the iovation authorization request, a value of 0 is used.
`IOVATION_AUDIT.accountCode`	The value of the `accountCode` attribute for the transaction.
`IOVATION_AUDIT.result`	The iovation risk check result. Valid values are: ▪ A – Accept ▪ D – Deny ▪ R – Review
`IOVATION_AUDIT.reason`	The iovation admin-specified value corresponding to the iovation rule that contributed most to the result.
`IOVATION_AUDIT.ruleName`	The name of the PingAccess rule responsible for this iovation Fraud check.
`IOVATION_AUDIT.iovationId`	A unique ID provided by iovation for the request.
`IOVATION_AUDIT.statedIp`	The IP address of the requesting client. This value is provided as the statedIp of the iovation Fraud API request.

3. Locate the Loggers section and add an entry to enable logging.

   Example:

```
<Logger name="iovationaudit" level="INFO" additivity="false">
      <AppenderRef ref="Iovation-File"/>
</Logger>
```

4. Restart PingAccess.

Improving iovation accessibility using a reverse proxy

You can improve the accessibility of iovation, even if end-users are blocking third-party content, by configuring a reverse proxy to connect to iovation.

About this task

This procedure applies only for gateway deployments. It automates much of the process of configuring a reverse proxy to connect to iovation, such that PingAccess acts as a reverse proxy as described by *Retrieving & Serving Dynamic iovation JavaScript* in the iovation Help Center. It uses Postman to create a PingAccess application with settings that allow it to act as the reverse proxy for the provided virtual hosts. You can add additional virtual hosts to this application as necessary.

If you are using an agent deployment, iovation recommends one of two models: Reverse proxy or Web device print server. See *Retrieving & Serving Dynamic iovation JavaScript* for more information about these models.

Steps

1. Go to the `<iovation_integration_home>/setup` directory.

2. Import the `iovation First Party Dynamic JavaScript Reverse Proxy.postman_collection.json` collection file into Postman.
3. Set the environment variables as described by the collection documentation.
4. Run the collection.

Token Providers

This section provides instructions on how to set up functionality between PingAccess and several common token providers.

- If you're using PingFederate, see *Configure PingFederate as the token provider for PingAccess*.
- If you're using the QuickStart Utility in conjunction with PingFederate and PingAccess, see *Use the PingAccess QuickStart utility*.
- If you're using PingOne, see *Protect applications using PingAccess and PingOne for Customers*.
- If you're using Azure AD, see *PingAccess for Azure AD*.

Configure PingFederate as the token provider for PingAccess

This section explains how to manually configure PingAccess and PingFederate to work together, with PingAccess as the access manager and PingFederate as the token provider.

For more information, see the following topics:

- *Configure PingFederate for PingAccess connectivity* on page 507
- *Connect PingAccess to PingFederate* on page 514

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Configure PingFederate for PingAccess connectivity

This section explains how to configure PingFederate for PingAccess connectivity.

This configuration procedure covers the following:

1. *Enabling PingFederate roles and protocols* on page 508
2. *Creating a password credential validator* on page 508
3. *Configuring an IdP adapter* on page 509
4. *Defining the default scope* on page 509
5. *Creating an access token manager* on page 510
6. *Configuring an IdP adapter mapping* on page 510
7. *Configuring an access token mapping* on page 511
8. *Creating an OpenID Connect policy* on page 511
9. *Creating a resource server client* on page 512
10. *Creating a web session client* on page 513
11. *Creating and exporting a certificate* on page 513

Important:

These steps assume you have installed PingFederate 10.1. If you are using an earlier version of PingFederate, the steps might differ. The example assumes that your PingFederate instance is available at https://<*mypingfedserver*>, using ports `9031` and `9999` respectively for the runtime and administration functions.

These steps assume you have installed PingAccess 6.1. If you are using an earlier version of PingAccess, the steps might differ. This example assumes that your PingAccess instance is available at https://<*mypingaccessserver*> and that `3000` is the default listening port.

Enabling PingFederate roles and protocols

If you are using PingFederate 10.0 or earlier, ensure that PingFederate is configured to respond to OAuth and OpenID Connect (OIDC) requests.

About this task

For more information on PingFederate roles and protocols, see *Choose roles and protocols*.

Steps

1. In the PingFederate administrative console, go to **System** # **Server** # **Protocol Settings**.
2. Click **Roles & Protocols** and verify that the following items are selected. Click **Next**.
    - **Enable OAuth 2.0 Authorization Server as Role** (role) and **OpenID Connect** (protocol)
    - **Enable Identity Provider (IdP) Role and Support the Following:** (role) and **SAML 2.0** (protocol)
3. On the **Federation Info** tab, enter the URL of your PingFederate environment and your SAML 2.0 entity ID, then click **Next**.

    For example:

    - **Base URL**: `https://mypingfedserver:9031`
    - **SAML 2.0 Entity ID**: `https://mypingfedserver/idp`
4. Review the summary. Click **Save**.

Next steps

Create a password credential validator.

Creating a password credential validator

Create a password credential validator (PCV) and then create a username and password to use in authentication.

About this task

For more information on PCVs, see *Manage password credential validators*.

Steps

1. Go to **System** # **Data & Credential Stores** # **Password Credential Validators**.
2. Click **Create New Instance**.
3. In the **Instance Name** field, enter an instance name of your choosing.

    For example, `My_PCV`.
4. In the **Instance ID** field, enter an instance ID of your choosing.

    For example, `mypcv`.
5. From the **Type** list, select **Simple Username Password Credential Validator**, and then click **Next**.
6. On the **Instance Configuration** tab, click **Add a new row to 'Users'**.
7. In the **Username** field, enter a username.
8. In the **Password** fields, enter and confirm a password.
9. Click **Update**, then click **Next**.
10. On the **Summary** tab, click **Save**.

Next steps
Configure an IdP adapter.

Configuring an IdP adapter
Configure an identity provider (IdP) adapter to look up session information and provide user identification to PingFederate. This example uses an instance of the HTML form adapter with an instance of the simple password credential validator (PCV).

About this task

For more information, see *Manage IdP adapters*.

Steps

1. Go to **Authentication** # **Integration** # **IdP Adapters**.
2. Click **Create New Instance**.
3. In the **Instance Name** field, enter an instance name of your choosing.

   For example, `My_IdP`.
4. In the **Instance ID** field, enter an instance ID of your choosing.

   For example, `myidp`.
5. From the **Type** list, select **HTML Form IdP Adapter**, and then click **Next**.
6. On the **IdP Adapter** tab, under **Password Credential Validator Instance**, click **Add a new row to 'Credential Validators'.**
7. From the **Password Credential Validator Instance** list, select the password credential validator you created previously, for example, **My_PCV**, and then click **Update**.
8. Click **Next** until the **Adapter Attributes** tab is displayed.
9. Locate the `username` attribute, then select the **Pseudonym** check box.
10. Click **Next** until the **Summary** tab is displayed. Click **Save**.

Next steps
Define the default scope.

Defining the default scope
Use the **Scope Management** section to define the default scope.

About this task

For more information, see *Define scopes*.

Steps

1. Go to **System** # **OAuth Settings** # **Scope Management**.
2. Click the **Common Scopes** tab, then enter the following scope values and their descriptions one at a time, clicking **Add** with each entry.

Scope Value	Scope Description
address	address
email	email
openid	openid

Scope Value	Scope Description
phone	phone
profile	profile

3. Click **Next** until you reach the **Default Scope** tab.
4. On the **Default Scope** tab, enter a description.

   For example, `default scope`.
5. Click **Save**.

Next steps
Create an access token manager.

Creating an access token manager
Create an access token to grant access and control access parameters. This sample configuration uses an instance of the Access Token Manager (ATM) using the Internally Managed Reference Tokens data model.

About this task

For more information, see *OAuth access token management*.

Steps

1. Go to **Applications** # **OAuth** # **Access Token Management**.
2. Click **Create New Instance**.
3. In the **Instance Name** field, enter an instance name of your choosing.

   For example, `General Access Token`.
4. In the **Instance ID** field, enter an instance ID of your choosing.

   For example, `GeneralAccessToken`.
5. From the **Type** list, select **Internally Managed Reference Tokens**.
6. Click **Next** until the **Access Token Attribute Contract** tab is displayed.
7. In the **Extend the Contract** field, enter `UserName`, and then click **Add**.
8. Click **Next** until the **Summary** tab is displayed. Click **Save**.

Next steps
Configure an IdP adapter mapping.

Configuring an IdP adapter mapping
Configure an identity provider (IdP) adapter mapping to map attributes.

About this task

For more information, see *Manage IdP adapter mappings for OAuth*.

Steps

1. Go to **Authentication** # **OAuth** # **IdP Adapter Grant Mapping**.
2. From the **Source Adapter Instance** list, select the adapter you created in *Configuring an IdP adapter* on page 509.
3. Click **Add Mapping**, then click **Next** until the **Contract Fulfillment** tab is displayed.

4. In the **USER_KEY** row:
    a. In the **Source** column, select **Adapter**.
    b. In the **Value** column, select **username**.
5. In the **USER_NAME** row:
    a. In the **Source** column, select **Adapter**.
    b. In the **Value** column, select **username**.
6. Click **Next** until the **Summary** tab is displayed. Click **Save**.

Next steps

Configure an access token mapping.

Configuring an access token mapping

Configure an access token mapping that maps attributes to be requested from the OAuth resource server with the corresponding access token.

About this task

For more information, see *Manage access token mappings*.

Steps

1. Go to **Applications** # **OAuth** # **Access Token Mapping**.
2. From the **Context** list, select **Default** or select your IdP adapter instance.
3. From the **Access Token Manager** list, select the access token you created in *Creating an access token manager* on page 510.

    For example, **GeneralAccessToken**.
4. Click **Add Mapping**. Click **Next**.
5. On the **Contract Fulfillment** tab, from the **Source** list, select **Persistent Grant**.
6. From the **Value** list, select **USER_KEY**.
7. Click **Next** until the **Summary** tab is displayed. Click **Save**.

Next steps

Create an OpenID Connect policy.

Creating an OpenID Connect policy

Configure an OpenID Connect (OIDC) policy to define OIDC policies for client access to attributes mapped according to OpenID specifications.

About this task

For more information, see *Configure OpenID Connect policies*.

Steps

1. Go to **Applications** # **OAuth** # **OpenID Connect Policy Management**.
2. Click **Add Policy**.
3. In the **Policy ID** field, enter an Policy ID of your choosing.

    For example, `OIDC`.
4. In the **Name** field, enter a name of your choosing.

    For example, `OIDC`.

5. From the **Access Token Manager** list, select the access token you created in *Configuring an access token mapping* on page 511.

   For example, **GeneralAccessToken**.
6. Click **Next**.
7. On the **Attribute Contract** tab, delete all items beneath the **Extend the Contract** heading.
8. Click **Next** until the **Contract Fulfillment** tab is displayed.
9. From the **Source** list, select **Access Token**.
10. From the **Value** list, select **username**.
11. Click **Next** until the **Summary** tab is displayed. Click **Save**.
12. In the **Action** column for the policy you created, if the policy is not already listed as the default, click **Set as Default** .

Next steps
Create a resource server client.

Creating a resource server client
Configure an OAuth client for use with PingFederate token provider resource server configuration in PingAccess.

About this task

For more information, see *Manage OAuth clients*.

Steps

1. Go to **Applications** # **OAuth** # **Clients**.
2. Click **Add Client**.
3. In the **Client ID** field, specify a client ID.

   ```
   pa_rs
   ```
4. In the **Name** field, specify a name.

   ```
   PingAccessResourceServer
   ```
5. In the **Client Authentication** section, select **Client Secret**.
6. In the **Client Secret** section, select **Change Secret**, and then click **Generate Secret**.

   > **Tip:**
   >
   > Copy the secret to a secure location so that you can use it in PingAccess configuration.
7. In the **Redirect URIs** field, enter the OpenID Connect (OIDC) callback redirect to the PingAccess server.

   For example, `https://`*`mypingaccessserver`*`:3000/pa/oidc/cb`.
8. Click **Add**.
9. In the **Allowed Grant Types** section, select the **Access Token Validation (Client is a Resource Server)** check box.
10. Click **Save**.

Next steps
Create a web session client.

Creating a web session client
Configure an OAuth client for use with web session configuration in PingAccess.

About this task

For more information, see *Manage OAuth clients*.

Steps

1. Go to **Applications** # **OAuth** # **Clients**.
2. Click **Add Client**.
3. In the **Client ID** field, specify a client ID.

   ```
   pa_wam
   ```
4. In the **Name** field, specify a name.

   ```
   PingAccessWebAccessManagement
   ```
5. In the **Client Authentication** section, select **Client Secret**.
6. In the **Client Secret** section, select **Change Secret**, and then click **Generate Secret**.

   Tip:

   Copy the secret to a secure location so that you can use it in PingAccess configuration.
7. In the **Redirect URIs** field, add the OpenID Connect (OIDC) callback redirect to the PingAccess server.

   For example, `https://`*`mypingaccessserver`*`:3000/pa/oidc/cb`.
8. Click **Add**.
9. Select the **Bypass Authorization Approval** check box.
10. In the **Allowed Grant Types** section, select the **Authorization Code** check box.
11. Click **Save**.

Next steps
Create and export a certificate.

Creating and exporting a certificate
Create and export a certificate for the PingFederate server that you will import to PingAccess to establish trust.

About this task

For more information, see *Manage SSL server certificates*.

Steps

1. In the PingFederate administrative console, go to **Security** # **Certificate & Key Management** # **SSL Server Certificates**.
2. Click **Create New**.
3. In the **Common Name** field, enter the PingFederate server address.

   For example, `mypingfedserver`.
4. In the **Organization** field, enter your organization's name.
5. In the **Country** field, enter the two-letter abbreviation for your country.

6. Complete the remaining fields as required.
7. Click **Next**.
8. Click **Save**.
9. In the **Action** section, click **Activate Default for Runtime Server**.
10. In the **Action** section, click **Export**.
11. Select **Certificate Only**. Click **Next**.
12. Click **Export**, and then save the exported certificate.
13. Click **Done**.

Next steps
Connect PingAccess to PingFederate and configure an application.

Connect PingAccess to PingFederate

This section explains how to configure PingAccess to communicate with PingFederate.

In this configuration procedure, you will perform the following tasks:

1. *Import certificates and create a trusted certificate group*.
2. *Configure the token provider*.

After configuring PingAccess to use PingFederate as a token provider, you can configure it to protect a web application. See *Protecting a web application* for more information.

Importing certificates and creating a trusted certificate group

Import a certificate for the PingFederate server to establish trust.

About this task

For more information, see *Certificates*.

Steps

1. Click **Security** and then go to **Certificates** # **Certificates**.
2. Click **+ Add Certificate**.
3. In the **Alias** field, enter an alias for the certificate.

   For example, `PingFed`.
4. To select the certificate, click **Choose File**.
5. To import the certificate, click **Add**.

   Result: A new certificate row appears on the **Certificates** tab.
6. Click **Security** and then go to **Certificates** # **Trusted Certificate Groups**.
7. Click **+ Add Trusted Certificate Group**.
8. Drag a certificate onto the box that appears.
9. In the **Name** field, enter a name for the group in the box that appears.

   For example, `PingFed`.
10. Click **Save**.

Next steps
Configure the token provider.

Configuring the token provider
Establish communication with the token provider, PingFederate.

About this task

For more information, see *Manage Token Provider*.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # # **Runtime**.
2. In the **Issuer** field, enter the PingFederate issuer name.
3. From the **Trusted Certificate Group** list, select the **PingFed** certificate group.
4. Click **Save**.
5. Click **Settings** and then go to **System** # **Token Provider** # # **Administration**.
6. In the **Host** field, enter the host name or IP address for the PingFederate Runtime.

   For example, `mypingfedserver`.
7. In the **Port** field, enter the port number for PingFederate Runtime.

   For example, `9031`.
8. In the **Admin Username** field, enter the username.

   This username only requires auditor (read only) permissions in PingFederate.
9. In the **Admin Password** field, enter the password.
10. From the **Secure** list, select **Secure**.
11. From the **Trusted Certificate Group** list, select the **PingFed** certificate group.
12. Click **Save**.
13. Click **Settings** and then go to **System** # **Token Provider** # # **OAuth Resource Server**.
14. In the **Client ID** field, enter the OAuth Client ID you defined when creating the PingAccess OAuth client in PingFederate.

    For example, `pa_rs`.
15. In the **Client Credentials Type** section, select **Secret**, then enter the **Client Secret** assigned when you created the PingAccess OAuth client in PingFederate.
16. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes.

    For example, `username`.
17. Click **Save**.

Next steps
You can configure PingAccess to *Protect a web application*.

Use the PingAccess QuickStart utility

This section explains how to protect a web-based application using PingAccess as the access manager, PingFederate as the token provider, and the PingAccess QuickStart utility to enable the connection and provide sample applications.

The QuickStart utility is designed to support a sample environment and aid in your understanding of how PingAccess and PingFederate work together to protect applications. You can perform the example configuration to achieve a working result and become familiar with this solution, or you can substitute your own data.

The QuickStart utility creates a basic configuration featuring:

- A PingFederate HTML Form Adapter with an instance of a simple password credential validator
- The PingFederate Access Token Manager (ATM) using the internally managed reference token data model

The QuickStart utility does not retain information about the PingAccess and PingFederate configuration. If you restart the utility, you must repeat the configuration steps before using the sample applications.

This document does not detail the usage of identity mappings or authentication requirements that are common components of a typical configuration. Because these components are simple to configure and beyond the scope of this document, you can read more about them and other features in the *PingAccess User Interface Reference Guide*.

This document does not discuss how to manually configure PingAccess and PingFederate to work together, since the QuickStart application automates this process. See *Configure PingFederate as the token provider for PingAccess* on page 507 for the manual steps.

The following topics are covered in this section:

Installing and configuring QuickStart components

Install the QuickStart utility along with PingAccess and PingFederate.

Steps

1. *Download PingFederate 10.3 or later.*
2. *Install PingFederate*.
3. Optional: If you plan to use the one-time authentication app, install the CIBA authenticator plugin.
    a. Copy the contents of the `<Quickstart Home>/plugins/deploy` directory into the `<PingFederate Home>/server/default/deploy` directory.
    b. Copy the contents of the `<Quickstart Home>/plugins/conf` directory into the `<PingFederate Home>/server/default/conf` directory.

> **Note:**
>
> If you want to use an authenticator other than the CIBA authenticator, you must manually configure it in PingFederate before configuring the one-time authentication app.

4. Perform the first-time configuration
5. *Download PingAccess 6.0 or later.*
6. *Install PingAccess* and perform the first-time configuration.
7. *Download* and extract the PingAccess QuickStart bundle.
8. Change to the QuickStart directory and run the `quickstart-server-<version>.jar` file.

```
java -jar quickstart-server-6.0.0.0.jar
```

You can use the `--server.port=<port>` argument to specify a port other than the default of 8443.

```
java -jar quickstart-server-6.0.0.0.jar --server.port=8444
```

Next steps

Connecting the QuickStart utility to PingAccess and PingFederate on page 517

Connecting the QuickStart utility to PingAccess and PingFederate

Connect the QuickStart utility to your installed PingAccess and PingFederate deployments.

Steps

1. Go to https://*hostname*:8443 and sign on to the QuickStart utility.

   Note:

   If you cannot access the utility, you might need to restart it by rerunning the `.jar` file. For more information, see the final step in *Installing and configuring QuickStart components* on page 516.

   Result: The QuickStart user interface is displayed.
2. Click **Connect**.
3. Enter the PingFederate runtime configuration.
   a. In the **Host** field, enter the host name.
   b. In the **Port** field, enter the port.
4. Enter the PingFederate admin configuration.
   a. In the **Host** field, enter the host name.
   b. In the **Port** field, enter the port.
5. In the **Username** and **Password** fields enter the admin credentials for PingFederate, and then click **Validate**.
6. Click **Next**.
7. Enter the PingAccess runtime configuration.
   a. In the **Host** field, enter the host name.
   b. In the **Port** field, enter the port.
8. Enter the PingAccess admin configuration.
   a. In the **Host** field, enter the host name.
   b. In the **Port** field, enter the port.
9. In the **Username** and **Password** fields, enter the admin credentials for PingAccess, and then click **Validate**.
10. Click **Save and Close**.

Next steps
Use sample applications.

Using sample applications

Configure and launch one or more sample applications.

About this task
Once an application is configured, users can sign on using any configured user credentials. The QuickStart utility uses five users, which are configured by default. Credentials for these users are displayed in the **User Credentials** section.

Steps

1. Go to https://<*hostname*>:8443 and sign on to the QuickStart utility.

   Note:

   If you cannot access the utility, you might need to restart it by rerunning the `.jar` file. For more information, see the final step in *Installing and configuring QuickStart components* on page 516n.
2. Click **Sample Apps**.
3. In the **Access Control** section, under one of the sample applications, click **Configure**.

   Note:

   If you have already configured another sample application, the utility skips the PingFederate and PingAccess configuration pages.

   Result: The **Configure PingFederate** window is displayed.
4. If you have not yet configured an application, configure PingFederate and PingAccess for the sample applications.

   The user interface displays the details of each PingFederate and PingAccess configuration step.

   a. Click **Configure PingFederate**.
   b. Click **Next**.
   c. Click **Configure PingAccess**.
   d. Click **Next**.
5. Click **Configure PingAccess**.
6. Click **Save and Close**.
7. Click **Launch**.

   Result: The application launches in a new tab. You can sign on to the app using any configured set of credentials.

Sample app reference

This list shows the characteristics of each sample app included in the PingAccess QuickStart utility.

Sample apps

Traditional App

A web application that renders its views on the server side in response to HTTP requests. Once accessed, it displays a simple to-do list.

Single Page App

An application that uses Javascript to render different views within the browser. Once accessed, it displays a simple to-do list.

API-Only App

An application that is intended to be accessed with APIs and not through a UI. It lets users create and manage a simple to-do list.

One-Time Auth App

An application that has a resource that requires authorization for every request. It lets users send hypothetical money to specified recipients.

Viewing apps without access control

View the sample apps without any access control to understand their behavior.

Steps

1. Go to https://*hostname*:8443 and sign on to the QuickStart utility.

> **Note:**
>
> If you cannot access the utility, you might need to restart it by rerunning the `.jar` file. For more information, see the final step in *Installing and configuring QuickStart components* on page 516.

2. Click **Sample Apps**.
3. In the **No Access Control** section, under one of the sample applications, click **Launch**.

   Result: The app is displayed.

Restoring PingFederate or PingAccess

Restore PingFederate or PingAccess from a saved configuration file.

Steps

1. Go to https://*hostname*:8443 and sign on to the QuickStart utility.

> **Note:**
>
> If you cannot access the utility, you might need to restart it by rerunning the `.jar` file. For more information, see the final step in *Installing and configuring QuickStart components* on page 516.

2. Click **Sample Apps**.
3. In the **Archive Restoration** section, click **Upload PingFederate Archive** or **Upload PingAccess Archive**.
4. Select the relevant archive.
5. Click **Restore PingFederate Instance** or **Restore PingAccess Instance**.

Result

The specified instance is restored.

Protect applications using PingAccess and PingOne for Customers

Configure PingAccess to provide secure external access to applications using PingAccess and PingOne.

In this scenario, PingAccess provides an external path to applications while PingOne acts as the token provider for associated sessions.

This solution requires you to perform the following tasks. For more information about the requirements and options available for each task, review the task.

- *Configure PingAccess to use PingOne for Customers as the token provider*
- *Configure a PingAccess application* for each application you want to protect and make available as part of this solution. Applications might require configuring:
  - A virtual host
  - A web session or access token validator
  - A site
  - An application

After you complete the configuration, you can test the application using the virtual host and context root that you assign to it in PingAccess.

Configuring PingAccess to use PingOne for Customers as the token provider

Configure PingAccess to use PingOne as the token provider in the PingAccess user interface.

Before you begin

- Install PingAccess and verify that you can access the *administrative console*. For more information on installing PingAccess, see *Installing and Uninstalling PingAccess* on page 50.

> **Note:**
>
> The default credential set should be changed upon first usage. The default credentials for your PingAccess installation are:
>
> ```
> Username: Administrator
> Password: 2Access
> ```

- *Configure an application* in PingOne.

About this task

For more information on configuring PingOne as the token provider, see *Configuring PingOne* on page 352

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **PingOne**.
2. In the **Issuer** field, enter the PingOne Issuer URL.

   To obtain the Issuer URL, in PingOne, go to the **Configuration** tab of an application and copy the **Issuer** value.
3. Optional: In the **Description** field, enter a description for the connection.
4. From the **Trusted Certificate Group** list, select a trusted certificate group that PingAccess will use when authenticating to PingOne.
5. To configure the connection to use a configured proxy, click **Show Advanced** and select **Use Proxy**.
6. Click **Save**.

Configuring a PingAccess application

Perform the following steps to configure PingAccess applications.

Before you begin

- Install PingAccess and verify that you can access the *administrative console*. For information on installing PingAccess, see *Installing and Uninstalling PingAccess* on page 50.

> **Note:**
>
> The default credential set should be changed upon first usage. The default credentials for your PingAccess installation are:
>
> ```
> Username: Administrator
> Password: 2Access
> ```

- *Configure an application* in PingOne.
- *Configure* PingAccess to use PingOne as the token provider.

About this task
For each application that you want to configure:

Steps

1. Create a virtual host.

   For more information on creating a virtual host, see *Creating new virtual hosts* on page 245.

   a. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
   b. Click **+ Add Virtual Host**.
   c. In the **Host** filed, enter a name for the virtual host.

      For example: myHost.com. You can use a wildcard (*) to indicate that any host name is acceptable. A wildcard host can also be specified, such as `*.example.com`.
   d. In the **Port** field, enter the port number for the virtual host.

      For example: `1234`.
   e. In the **Agent Resource Cache TTL (s)** field, indicate the number of seconds the agent can cache resources for this application.

      Note:

      Only applies to a destination of type `Agent`.
   f. Click **Save**.

2. Create a web session.

   For more information on creating a web session, see *Creating web sessions* on page 300.

   > **Note:**
   >
   > A web session is only used when protecting a web application. To protect APIs, configure an access token validator.

   a. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
   b. Click **+ Add Web Session**.
   c. In the **Name** field, enter a name for the web session.
   d. From the **Cookie Type** list, select your cookie type, either **Signed JWT** or **Encrypted JWT**.
   e. In the **Audience** field, enter a unique value.
   f. In the **Client ID** field, enter the PingOne client ID.

      > **Tip:**
      >
      > You can find the Client ID on the **Profile** tab of the application you created.

   g. From the **Client Credentials Type** list, select **Secret**.
   h. In the **Client Secret** field, enter the client secret found on the application's **Configuration** tab.
   i. Click **Show Advanced** .
   j. In the **Scopes** section, specify one or more scopes.

      > **Note:**
      >
      > Ensure the scopes you specify match those configured for the PingOne application. Find the scopes on the **Access** tab of your PingOne application.

   k. Click **Save**.
3. Create a site.

   For more information on creating a site, see *Adding sites* on page 247.

   > **Note:**
   >
   > In some configurations, a site might contain more than one application. A site can be used with more than one application, where appropriate.

   a. Click **Applications** and then go to **Sites** # **Sites**.
   b. Click **+ Add Site**.
   c. Specify a **Name** for the site.
   d. Enter the site **Target**.

      The target is the hostname:port pair for the server hosting the application. Do not enter the path for the application in this field. For example, an application at https://mysite:9999/AppName will have a target value of `mysite:9999`.
   e. From the **Secure** list, select whether or not the target is expecting secure connections.
   f. If the target is expecting secure connections, from the **Trusted Certificate Group** list, select **Trust Any**.
   g. Click **Save**.

4. Create an application in PingAccess for each application that you want to protect.

   For more information on creating an application, see *Adding an application* on page 228.

   a. Click **Applications** and then go to **Applications** # **Applications**.
   b. Click **+ Add Application**.
   c. In the **Name** field, enter a name for the application.
   d. In the **Description** field, optionally enter a description for the application.
   e. In the **Context Root** field, specify the context root for the application.

      For example, an application at https://mysite:9999/AppName will have a context root of `/AppName`. If the application is on the root of the server, you can set the context root as `/`. The context root must begin with a slash (/), must not end with a slash (/), and can be more than one layer deep, for example, `/Apps/MyApp`.
   f. From the **Virtual Host** list, select the virtual host you created.

      Note:

      The combination of virtual host and context root must be unique in PingAccess.
   g. From the **Application Type** list, select **Web**.
   h. From the **Web Session** list, select the web session you created.
   i. From the **Site** list, select the site you created that contains the application.
   j. Select the **Enabled** check box to enable the site when you save.
   k. Click **Save**.

PingAccess for Azure AD

Configure PingAccess to provide secure external access to legacy on-premises applications using PingAccess for Azure AD and Microsoft Azure AD.

In this scenario, PingAccess provides an external path to legacy on-premises applications using the Azure AD Application Proxy through the use of header based authentication. Additionally, Microsoft Azure AD acts as the token provider for associated sessions.

PingAccess for Azure AD is a limited, free version of PingAccess for Microsoft Azure AD customers that provides protection for up to 20 applications.

This solution requires you to perform the following tasks. For more information about the requirements and options available for each task, review the task.

- *Configure PingAccess to use Azure AD as the token provider*
- *Configure a PingAccess application* for each application you want to protect and make available to Azure AD as part of this solution. Applications require configurating:
  - A virtual host
  - A web session
  - An identity mapping
  - A site
  - An application

After you complete the configuration, you can test the application using the home page URL that you create in Azure AD.

Get started with PingAccess for Azure AD

Protect legacy on-premises applications using Microsoft Azure AD and a limited version of PingAccess called PingAccess for Azure AD.

When planning for a successful deployment:

Plan your deployment type and architecture

Use the *Deployment reference guide* to plan your deployment type and architecture. Learn about the differences between and benefits of a proxy deployment versus an agent based deployment, and decide to use one or a combination of both deployment types.

Design and plan a PingAccess cluster

Use the *Clustering reference guide* to design and plan your PingAccess cluster. For a high availability deployment, use a cluster that contains both a primary administrative node and a replica administrative node, along with additional engine nodes. For best performance, employ a *load balancing strategy*.

Install PingAccess

Ensure your systems meet the requirements so you can *Install PingAccess*.

Tune performance

Use the *Performance tuning reference guide* to configure your deployment for optimal performance.

Configure logging

Configure logging so that you can monitor your PingAccess deployment and troubleshoot application issues.

Configure the PingAccess token provider

Configure PingAccess to use Microsoft Azure AD as the *token provider*. Perform optional additional configuration that allows for communication with the *Azure AD Graph API*.

Configure applications

Configure applications to be made available by PingAccess to the Microsoft MyApps portal through Azure AD using the Azure AD Application Proxy.

Configure for dual internal and external secure access

Configure the solution so that applications are made securely available both externally through the Microsoft MyApps portal and internally through PingAccess for Azure AD.

Configuring PingAccess to use Azure AD as the token provider

Configure PingAccess to use Azure AD as the token provider.

Before you begin

- Install PingAccess and verify that you can access the *administrative console*. For information on installing PingAccess, see *Installing and Uninstalling PingAccess* on page 50.

> **Note:**
>
> The default credential set should be changed upon first usage. The default credentials for your PingAccess installation are:
>
> ```
> Username: Administrator
> Password: 2Access
> ```

- If your administrative node uses a proxy for HTTP requests to the token provider, select the HTTP Proxy in the **System** # **Clustering** section. For more information, see *Configuring administrative nodes* on page 322.

About this task

For more information on configuring the token provider, see *Token provider* on page 342.

Steps

1. Click **Settings** and then go to **System** # **Token Provider** # **Common** # **OpenID Connect**.
2. In the **Issuer** field, enter the Microsoft Azure AD **Directory ID**.

   To obtain the directory ID from Azure AD, in the Azure AD directory, go to **Manage** # **Properties** and copy the **Directory ID** value.
3. From the **Trusted Certificate Group** list,
   Choose from:
   - **Java Trust Store**
   - **Trust Any**
4. Click **Save**.

Next steps
To get the most out of the solution, see *Configuring token provider-specific options* on page 353.

Configuring PingAccess applications for Azure

Configure PingAccess applications so they are accessible to users through the Microsoft Azure *MyApps* portal.

Before you begin

- Install PingAccess and verify that you can access the *administrative console*. For information on installing PingAccess, see *Installing and Uninstalling PingAccess* on page 50.

  > **Note:**
  >
  > The default credential set should be changed upon first usage. The default credentials for your PingAccess installation are:
  >
  > ```
  > Username: Administrator
  > Password: 2Access
  > ```
- Have a *Microsoft Azure AD* Premium account for access to the Application Proxy feature.
- Configure Microsoft Azure AD. For steps to configure Microsoft Azure AD, see *https://docs.microsoft.com/azure/active-directory/application-proxy-ping-access*.
- *Configure* PingAccess to use Azure AD as the token provider.

About this task

For each application that you want to configure:

Steps

1. Create a virtual host.

   For more information on creating a virtual host, see *Creating new virtual hosts* on page 245.

   > **Important:**
   >
   > In a typical configuration for this solution, you will create a virtual host for every application.

   a. Click **Applications** and then go to **Applications** # **Virtual Hosts**.
   b. Click **+ Add Virtual Host**.
   c. In the **Host** field, enter the FQDN portion of the Azure AD **External URL**.

      Example: For example, external URLs of https://app-tenant.msappproxy.net/ and https://app-tenant.msappproxy.net/AppName will both have a **Host** entry of `app-tenant.msappproxy.net`.
   d. In the **Port** field, enter `443`.
   e. Click **Save**.
2. Create a web session.

   For more information on creating a web session, see *Creating web sessions* on page 300.

   a. Click **Access** and then go to **Web Sessions** # **Web Sessions**.
   b. Click **+ Add Web Session**.
   c. In the **Name** field, enter a name for the web session.
   d. From the **Cookie Type** list, select your cookie type, either **Signed JWT** or **Encrypted JWT**.
   e. In the **Audience** field, enter a unique value.
   f. In the **Client ID** field, enter the Azure AD application ID.
   g. From the **Client Credentials Type** list, select **Secret**.
   h. In the **Client Secret** field, enter the client secret you generated for the application in Azure AD.
   i. Optional: To create and use custom claims with the Azure AD GraphAPI, click **Advanced** and clear the **Request Profile** and **Refresh User Attributes** check-boxes.

      For more information on using custom claims, see *Optional - Use a custom claim*.
   j. Click **Save**.
3. Create an identity mapping.

   For more information on creating an identity mapping, see *Creating header identity mappings* on page 295.

   > **Note:**
   >
   > An identity mapping can be used with more than one application if more than one application is expecting the same data in the header.

   a. Click **Access** and then go to **Identity Mappings** # **Identity Mappings**.
   b. Click **+ Add Identity Mapping**.
   c. In the **Name** field, enter a name.
   d. From the **Type** list, select **Header Identity Mapping**.
   e. In the **Attribute to Header Mapping** table, specify the required mappings.

      Example: For example.

Attribute Name	Header Name
upn	x-userprincipalname

Attribute Name	Header Name
email	x-email
oid	x-oid
scp	x-scope
amr	x-amr

f. Click **Save**.

4. Create a site.

   For more information on creating a site, see *Adding sites* on page 247.

> **Note:**
>
> In some configurations, a site might contain more than one application. A site can be used with more than one application, where appropriate.

a. Click **Applications** and then go to **Sites** # **Sites**.
b. Click **+ Add Site**.
c. In the **Name** field, enter a name for the site.
d. In the **Target** field, specify the target.

   The target is the hostname:port pair for the server hosting the application. Do not enter the path for the application in this field. For example, an application at https://mysite:9999/AppName will have a target value of `mysite:9999`.
e. From the **Secure** list, select whether or not the target is expecting secure connections.
f. Click **Save**.

5. Create an application in PingAccess for each application in Azure that you want to protect.

   For more information on creating an application, see *Adding an application* on page 228.

a. Click **Applications** and then go to **Applications** # **Applications**.
b. Click **+ Add Application**.
c. In the **Name** field, enter a name for the application.
d. In the **Description** field, enter a description for the application.
e. In the **Context Root** field, specify the context root for the application.

   For example, an application at `https://mysite:9999/AppName` will have a context root of `/AppName`. If the application is on the root of the server, you can set the context root as `/`. The context root must begin with a slash (/), must not end with a slash (/), and can be more than one layer deep, for example, `/Apps/MyApp`.
f. From the **Virtual Host** list, select the virtual host you created.

> **Note:**
>
> The combination of virtual host and context root must be unique in PingAccess.

g. From the **Application Type** list, select **Web**.
h. From the **Web Session** list, select the web session you created.
i. From the **Site** list, select the site you created that contains the application.
j. From the **Web Identity Mapping** list, select the mapping you created.
k. Select the **Enabled** check box to enable the site when you save.
l. Click **Save**.

Configuring applications for dual access with PingAccess for Azure AD

Configure applications for secure access both from inside and outside the network.

Steps

1. Configure an application for secure external access using *Microsoft Azure AD* and *PingAccess for Azure AD*.
2. Ensure that the application is functioning as expected by signing on using the applications external URL.

   Example: For example, `http://app-tenant.msappproxy.net/`.
3. In PingAccess, *create a new virtual host* that maps to the PingAccess host.

   Example: For example, `<PingAccessServerName>:3000`.
4. Assign the new virtual host to the application in addition to the virtual host specified for Azure access.
5. In Azure AD, go to the **App Registrations** window and select the application.
6. Click **Reply URLs**, and add the internal PingAccess reply URL.

   Example: For example, `<PingAccessServerName>:3000/pa/oidc/cb`.
7. Save the changes and test the configuration by signing on using the application's local URL.

PingAccess Monitoring Guide

PingAccess provides a range of monitoring options, from simple heartbeat options for checking responsiveness to transaction response-time logging and resource-utilization metrics. These metrics can help you gain insight into the health and performance of your PingAccess deployment.

To help you monitor the performance of a PingAccess deployment, this guide provides the following:

- Suggestions for key performance metrics to monitor and means by which to monitor them
- Recommendations about resource-utilization thresholds and patterns
- Monitoring options, including logs that can be used to create Splunk dashboards

The features documented here are affected by the settings in the configuration file. See the *Configuration file reference* on page 163 for more information.

Liveliness and responsiveness

One of the simpler methods for monitoring the performance of a PingAccess deployment involves determining whether PingAccess Server is available and responsive. To help you identify the status of a server, PingAccess provides a heartbeat request endpoint.

Heartbeat endpoint

If PingAccess Server is running, the process of sending a request to the endpoint `/pa/heartbeat.ping` returns an `OK` browser message and an `HTTP 200` status. If the request times out or requires an extended amount of time to return, the server might be overloaded or experiencing other difficulties.

If a request requires more than two or three seconds to return, multiple factors in your PingAccess deployment might be responsible. Develop a baseline for the desired response time by testing the heartbeat endpoint of your deployment at various times. This endpoint can be useful when load balancing a cluster of PingAccess Server instances. Some load balancers can alter the number of requests that are sent to a particular server based on the response code received, or the responsiveness of requests that are made to the heartbeat endpoint.

The output of the heartbeat can be modified to provide performance-related information, such as CPU and memory usage, along with response times. The following example shows the JSON data that is returned when the template is changed to show the memory, CPU, and response time in milliseconds.

Example JSON data showing memory, CPU, and response time in milliseconds.

Example


```
{"items":[{
 "response.statistics.window.seconds": "5",
 "response.statistics.count": "1",
 "response.time.statistics.90.percentile":
 "129", "response.time.statistics.mean": "129",
 "response.time.statistics.max":"129",
 "response.time.statistics.min": "129",
 "response.concurrency.statistics.90.percentile": "1",
 "response.concurrency.statistics.mean": "1",
 "response.concurrency.statistics.max": "1",
 "response.concurrency.statistics.min": "1",
  "cpu.load": "15.53",
  "total.jvm.memory": "500.695 MB",
  "free.jvm.memory": "215.339 MB",
  "used.jvm.memory": "285.356 MB",
  "total.physical.system.memory": "17.18 GB",
  "total.free.physical.system.memory": "278.45 MB",
  "total.used.physical.system.memory": "16.901 GB",
  "number.of.cpus": "8",
  "hostname": "jdasilva-r",
  "open.client.connections": "1",
  "number.of.applications": "11",
  "number.of.virtual.hosts": "6",
  "last.refresh.time": "1969-12-31T18:00:00.000Z"
 }]}
```


For more information, see *Heartbeat endpoint*.

Response time logging

By default, the audit logs record the processing time for each transaction. With audit logging enabled, you can identify the speed with which PingAccess Server processes web and API application transactions. Depending on your logging configuration, audit logging might not log any transactions. For more information, see *Security audit logging*.

The following example shows a default audit log with the following information:

- Total roundtrip
- Proxy roundtrip
- Userinfo roundtrip

Example

Example code of processing times in bold and shown in milliseconds.

```
2019-12-15T17:23:12,192|GRmozOujPDDFct8RbtnfJw|tid:wUu9F0vDd9pZPKe4Oc5Ym_-
RFCc..9r72.v8c0Y2CUA5qSpvcxKHgd7QoCp|
81 ms| 50 ms| 0 ms| servapp.ext.wal-ping.com [] /SimpleWebApi /*:3000| joe|
 Cookie| 127.0.0.1| GET| /SimpleWebApi/web/web.jsp| 200| | | Web-API| Root
 Resource| /*
```

Resource metrics

PingAccess provides monitoring capabilities for resource-utilization metrics, such as thresholds and patterns, to strengthen the health and performance of your deployment.

PingAccess provides the following mechanisms for obtaining resource metrics:

- Java management extensions (JMX) - Ping recommends using JMX MBeans because this method provides a more comprehensive set of resource metric counters for analyzing performance. Several tools are available for collecting and analyzing data from JMX MBeans, including many security information and event management (SIEM) tools, such as Splunk.
- Heartbeat endpoint - For more information about enabling heartbeat message reporting, see *Configuring PingAccess heartbeat messages*.

Monitoring discusses the JConsole monitoring tool that is included with the Java SE platform. For more information about the Comprehensive JConsole, see *Troubleshoot with the JConsole Tool* in the Oracle JDK documentation and *The Java Monitoring and Management Console (jconsole)* in the OpenJDK documentation.

Connecting with JMX

The Java management extensions (JMX) MBeans agent included on the Java SE platform enables connections to local and remote Java clients to monitor performance.

JConsole permits connections to local and remote Java processes. If your instance of PingFederate is running as a Windows Service, you must connect through the remote option.

For information on connecting to a local process, see *Connecting to a local process*. For information on connecting to a remote process, see *Connecting to a remote process*.

Connecting to a local process

Use the local process option to establish a connection when the PingAccess Server is running on a local system.

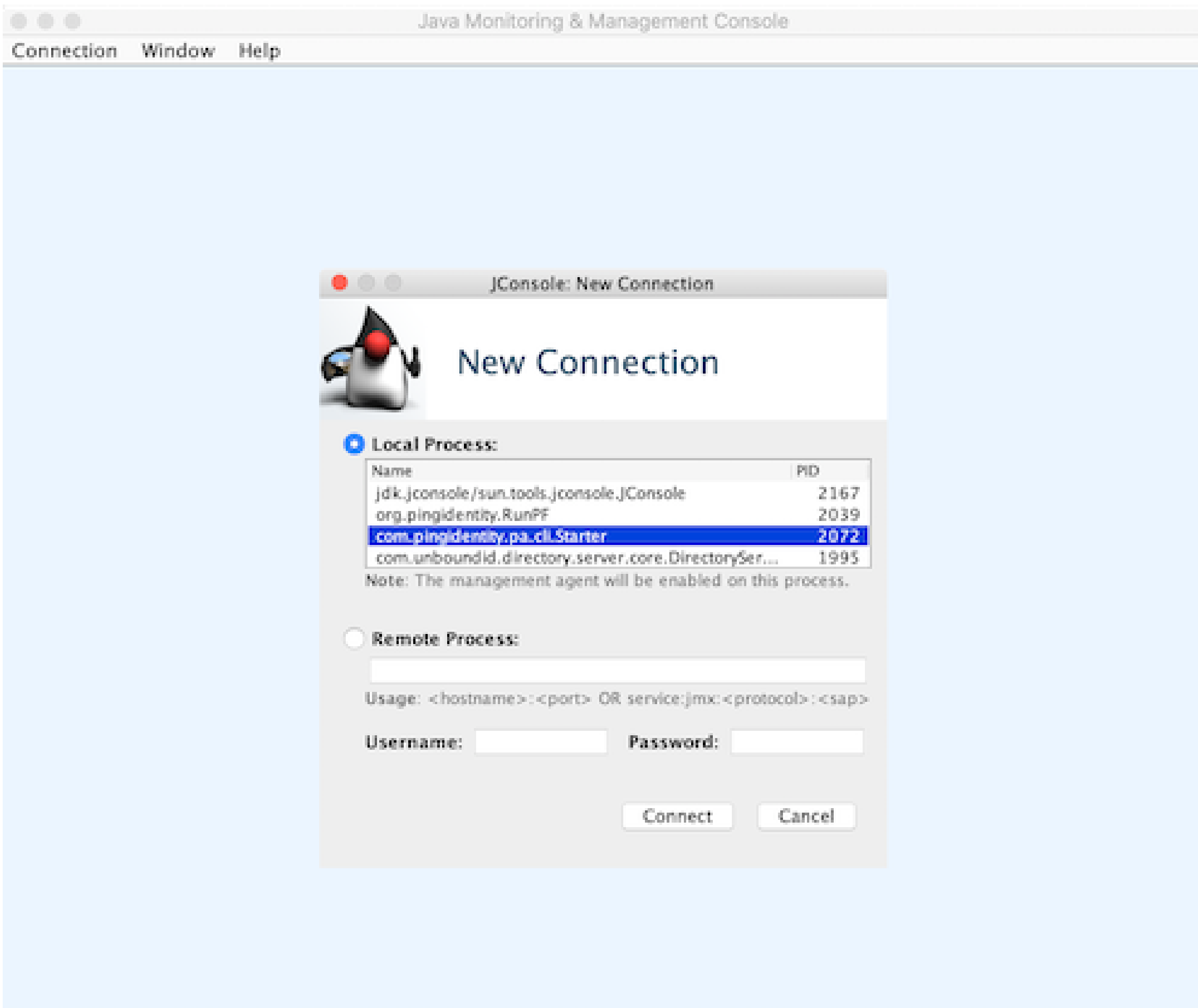
About this task

Unless you are running PingAccess Server as a Windows service, the easiest method to launch JConsole on the same machine as the server is to select **Local Process**. For information about connecting to a remote process, see *Connecting to a remote process*.

Steps

- To connect to a local instance and start the monitoring process, select `com.pingidentity.pa.cli.Starter` from the **Local Process** list and click **Connect.**

> **Note:**
>
> If you are running the process locally, the system might prompt you to accept the connection as insecure.



Connecting to a remote process

Use the remote process option to establish a connection when the PingAccess Server is running as a Windows Service, or if the `com.pingidentity.pa.cli.Starter` class is unavailable in the Local Process list.

About this task

Use these instructions to configure the remote process option to establish a connection. For demonstration purposes, the following task uses an LDAP configuration.

Note:

No direct configuration support is provided for enabling remote access Java Management Extensions (JMX) for PingAccess Server. To enable this level of access, use the built-in options that are available through the Java virtual machine (JVM). For more information, see *Monitoring and Management Using JMX Technology* in the Oracle JDK documentation.

Steps

1. In the `jvm-memory.options` file for PingAccess Server, add the following text at the end of the last memory settings.

```
#Settings to enable remote access to JMX
-Dcom.sun.management.jmxremote.port=5000"
-Dcom.sun.management.jmxremote.login.config=ExampleCompanyConfig"
#Configuration is assumed to be in the conf folder, relative path used
-Djava.security.auth.login.config=conf/ldap.config"
-Dcom.sun.management.jmxremote.ssl=false"
```

   Note:

   Each entry must reside on its own line. In this example, a relative path is used for the `ldap.config` file. Some deployments might require a full path.

   Tip:

   In a production environment, use SSL, as shown in this example for initial testing and debugging. For information about setting up SSL, see *Monitoring and Management Using JMX Technology* in the Oracle JDK documentation.

2. Create the `ldap.config` file.

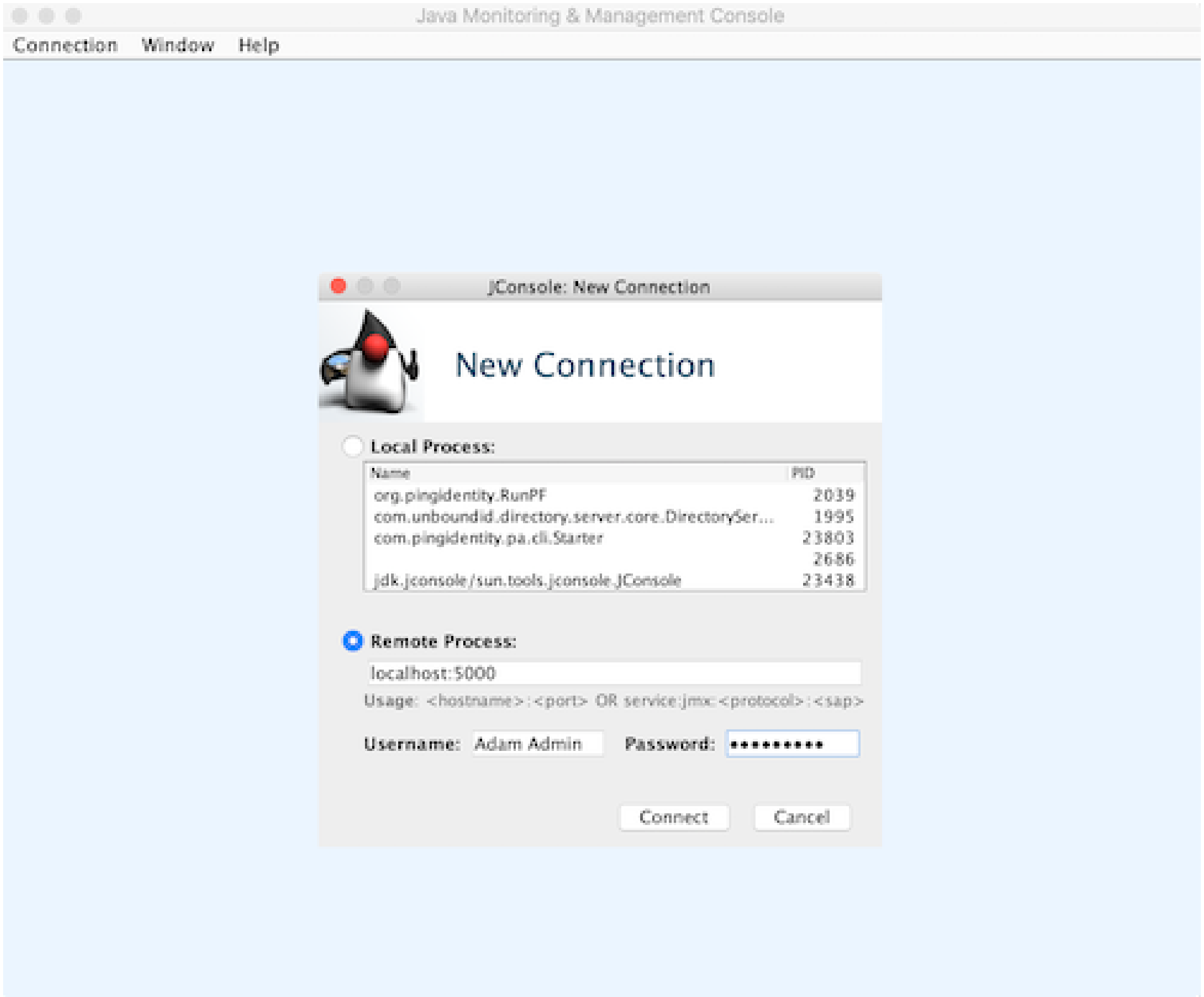
```
ExampleCompanyConfig {
    com.sun.security.auth.module.LdapLoginModule REQUIRED
    userProvider="ldaps://ldap.server:port/OU=where,OU=users,OU=located"
    userFilter="(&(uid={USERNAME})(objectClass=inetOrgPerson))"
    authIdentity="uid={USERNAME},OU=where,OU=users,OU=located"
    authzIdentity=monitorRole
    useSSL=true;
    };
```

   Note:

   Each entry must reside on its own line. In this example, `ldap.config` is placed in the PingAccess `conf` folder. If your JVM setup trusts the certificates, you can use SSL. Because of the `authIdentity` option, the configuration binds as the user that you enter. Otherwise, an anonymous bind validates the user name but not the password.

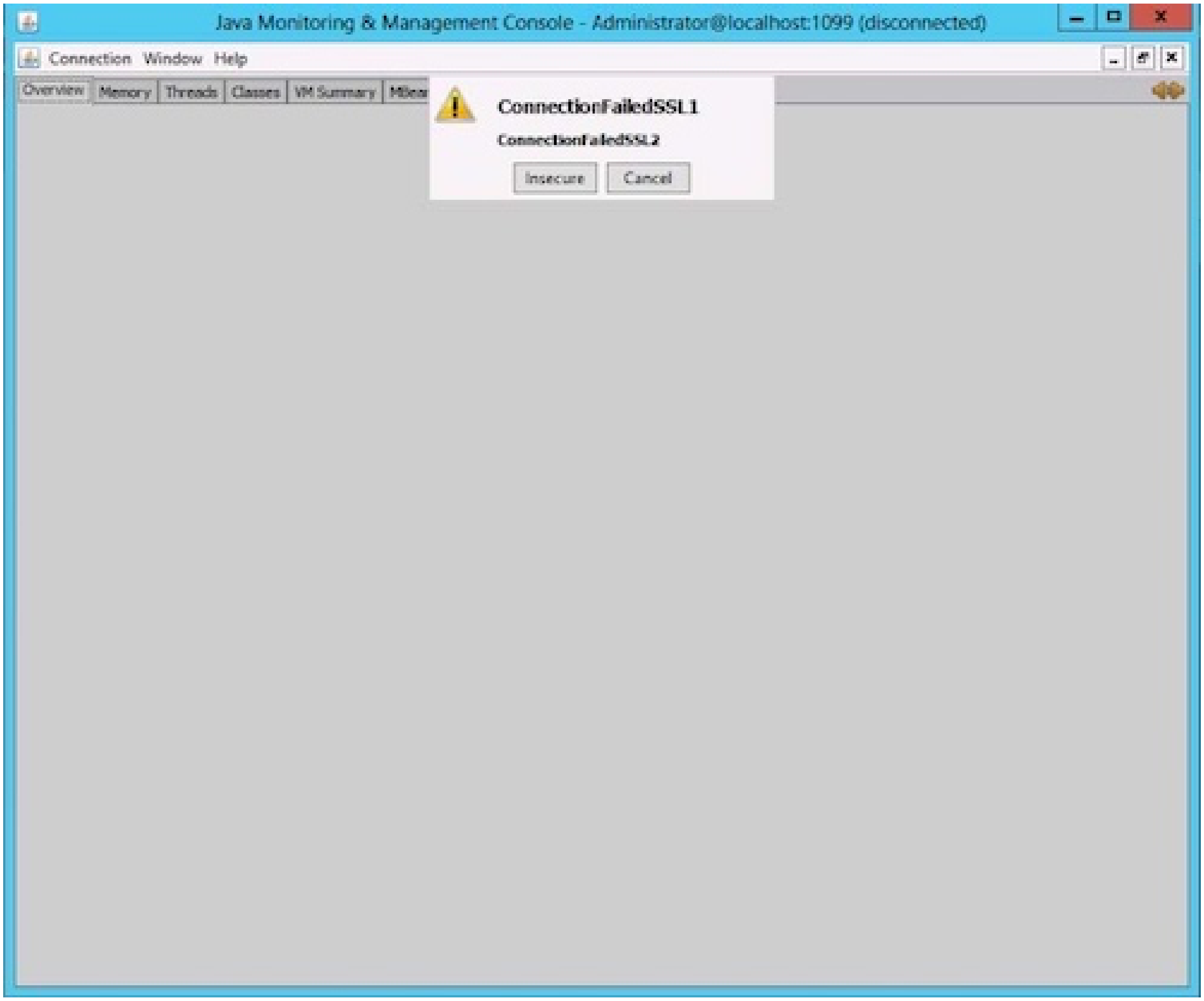
3. Place the file that you created in step 3 in a location from which the PingAccess process can read it at start up.
4. In a clustered PingAccess environment:
   a. Make the changes outlined in steps 1 - 3 to each node in the cluster.
   b. Restart each node.

5. After you enable the JMX service, connect to the remote JMX service by specifying one of the following:
   Choose from:
   - The name of the PingAccess Server instance
   - The IP address, port, and authentication credentials.



Note:

Because JMX uses SSL by default when communicating with a remote host, the client host must trust the SSL certificate that is presented during setup for JMX. If the JMX client does not trust the JMX certificate, the following message is displayed.



Troubleshooting:

a. If SSL is enabled: Import the JMX SSL certificate to the client's trusted certificates.
b. If SSL is disabled: Click **Insecure** to connect.

Monitoring

The JConsole monitoring interface is accessible after establishing a connection. This section outlines the key Java Virtual Machine (JVM) performance metrics for evaluating the activity of your PingAccess deployment.

Monitoring clustered PingAccess engines

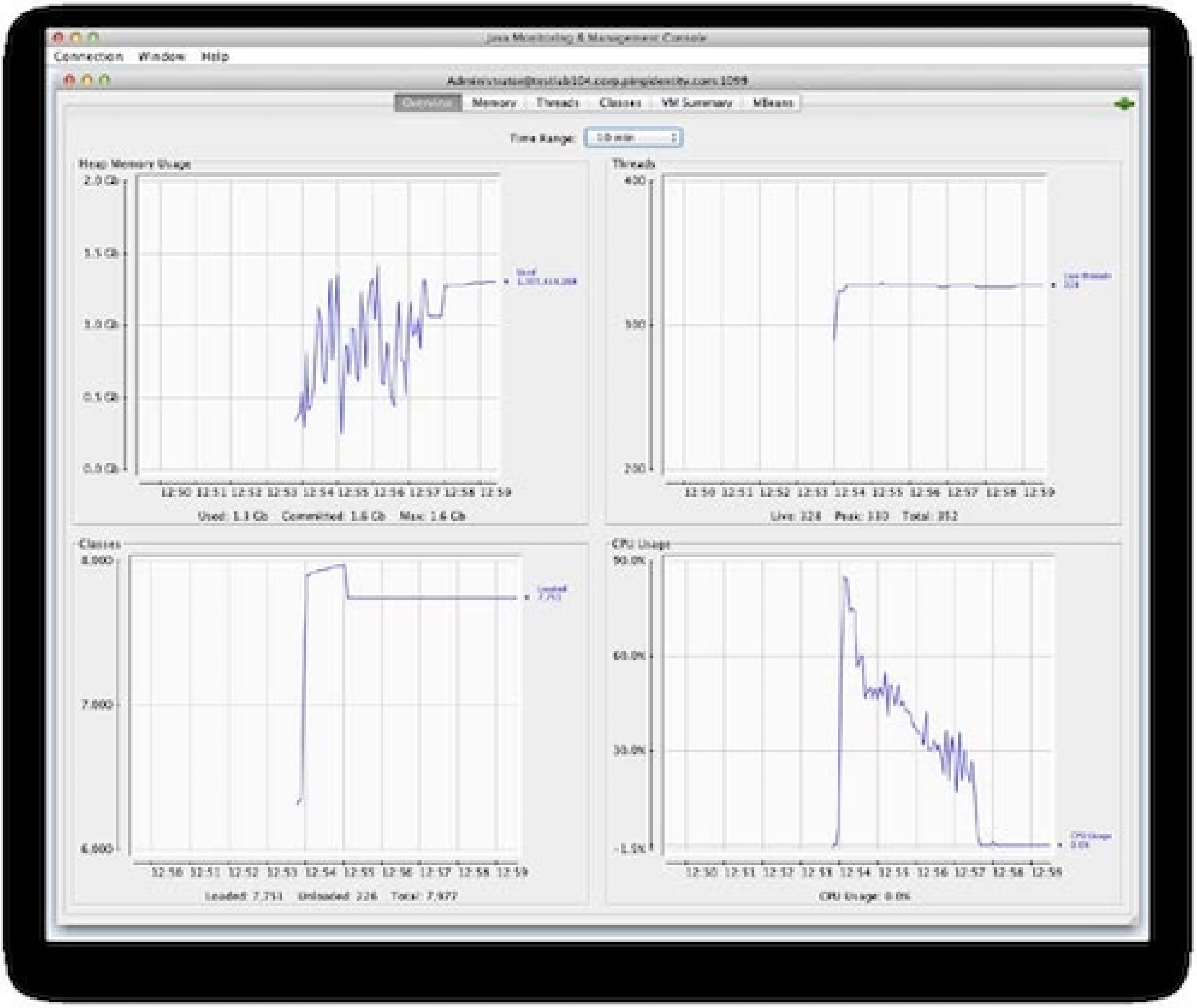
The JConsole can be connected to multiple processes. To monitor several instances of PingAccess Server after a connection is established, go to **Connection** # **New Connection** and add the additional connection.

Monitoring CPU utilization

The **Overview** tab provides a dashboard of the following performance and resource-utilization charts:

- Heap memory usage (cumulative memory that is used by all memory pools)
- Live threads
- CPU usage
- Classes (number of classes that are loaded)

This tab provides a high-level view of the JVM's performance metrics.



Use the **Overview** tab to visualize and collect CPU usage data. When your PingAccess deployment is subjected to its normal or expected load, the CPU utilization typically falls between 60% and 80%. If the system registers consistently at 80% or higher, additional CPU resources might be necessary to handle load spikes that occur during peak usage times.

Monitoring memory utilization

The **Overview** tab shows only overall heap usage. To view additional details about memory utilization, click the **Memory** tab, which lets you analyze usage patterns usage in specific memory pools within the heap. This tab also provides information about the overall heap utilization profile.

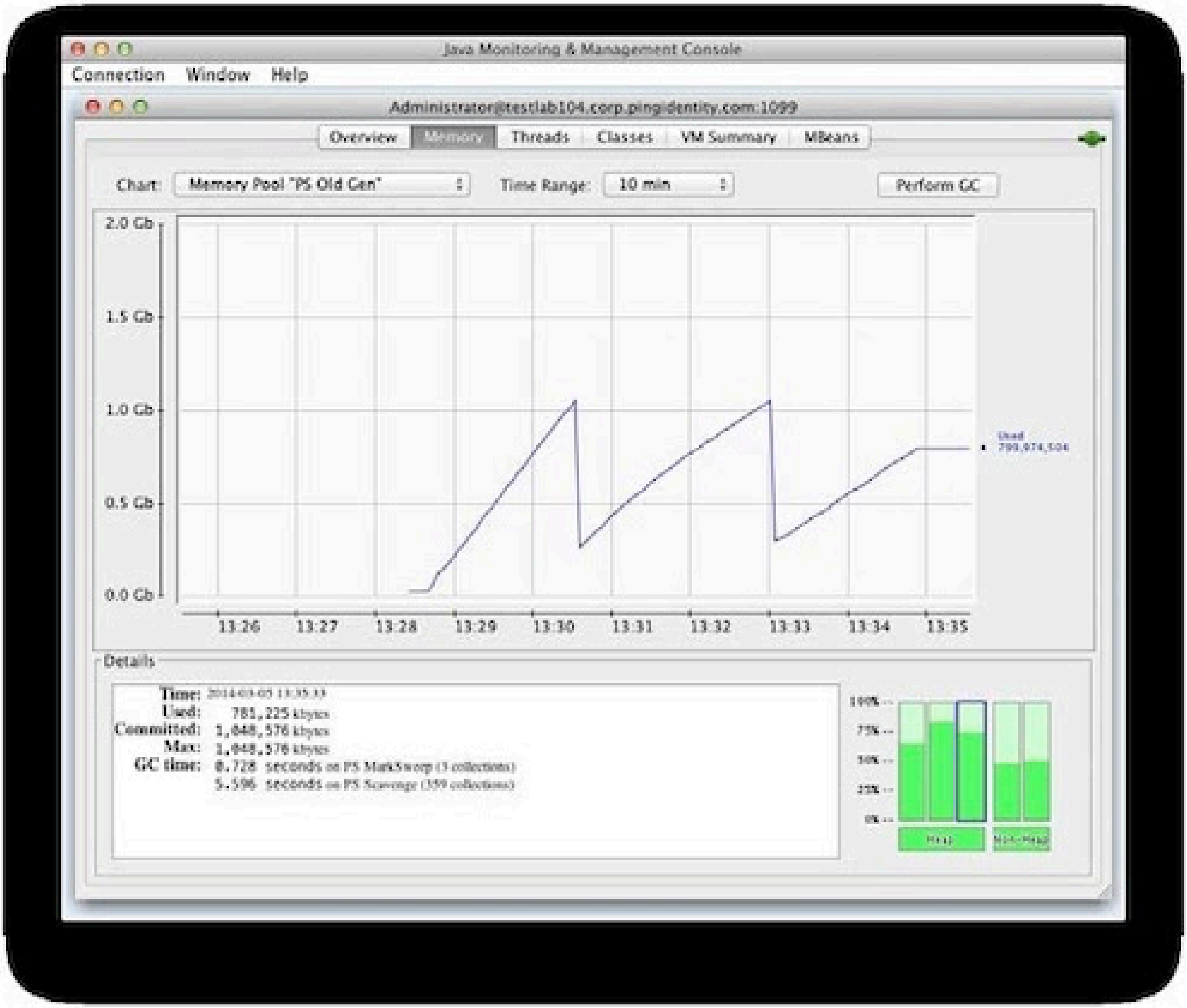
Old Generation space

Objects that survive a sufficient number of garbage-collection cycles are promoted to the Old Generation. To view the memory usage in the pool of such objects, go to **Memory Pool** # **PS Old Gen** or **Memory Pool** # **G1 Old**, depending on the relevant garbage collector. PingAccess Server services mostly short-lived transactions, such as single sign-on (SSO), security token service (STS), and OAuth requests. Most of the created memory objects are required only for a short period of time.

Although PingAccess Server makes use of some memory objects that are medium- to long-lived, such as session data for Authentication Sessions, Adapter Sessions, or single logout (SLO) functionality, most of the objects that are promoted to the Old Generation are likely to become garbage that requires cleaning up. If the younger generation, or Eden space, is not sized appropriately, objects are moved to and retained in the Old Generation before they are collected as garbage. If size limitations prevent the Old Generation from accumulating future garbage as well as longer-lived objects, then garbage-collection cycles occur more frequently.

The Old Generation space is the most important space to monitor. It is easy to identify if the heap is sized and proportioned appropriately for a specific load, based on its usage pattern. The following examples involve two Old Generation usage charts. In both examples, the same user load executes the same workflow. The size of the heap represents the only difference.

Because the heap is sized adequately in the first example, memory in the Old Generation rises at a reasonably slow rate. Garbage collection frees around 60% to 75% of the space, and room is available to accommodate the future garbage of newly created objects that are moved from the Eden space, as well as the longer-term objects that remain in use. Although the space is 1 GB in size, the average full (PS MarkSweep or G1 Old Generation) collection time is approximately only 240 milliseconds, or 0.728 seconds for three collections.



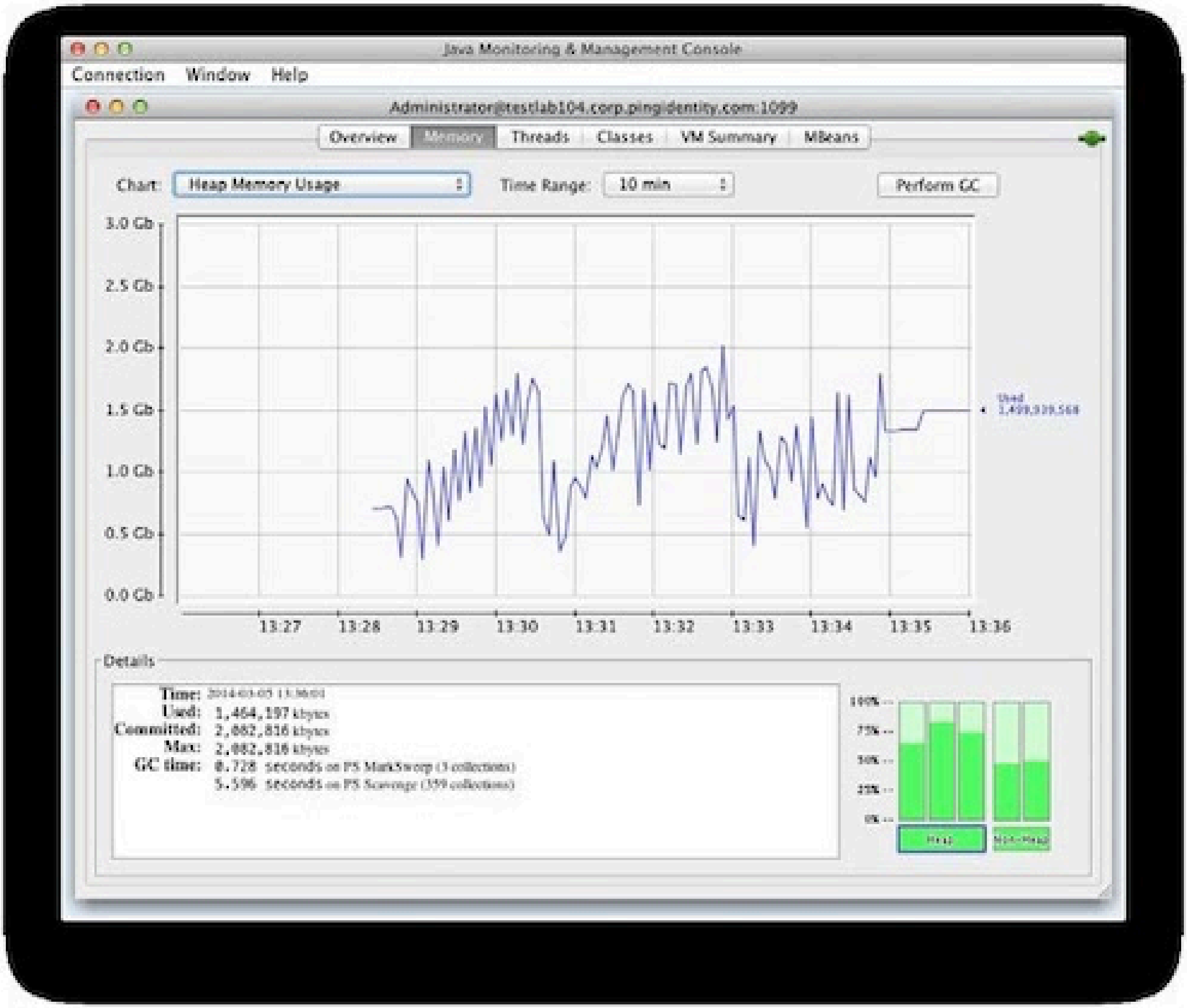
When a heap is sized inadequately, the Old Generation runs out of space.

In the following example, the amount of memory that becomes free with each garbage collection shrinks, due to the rate at which objects are promoted from the Eden space.

184 PS MarkSweep (full) collections require garbage collections more frequently, totaling 60 seconds, or an average of 326 milliseconds per collection.

Entire heap space

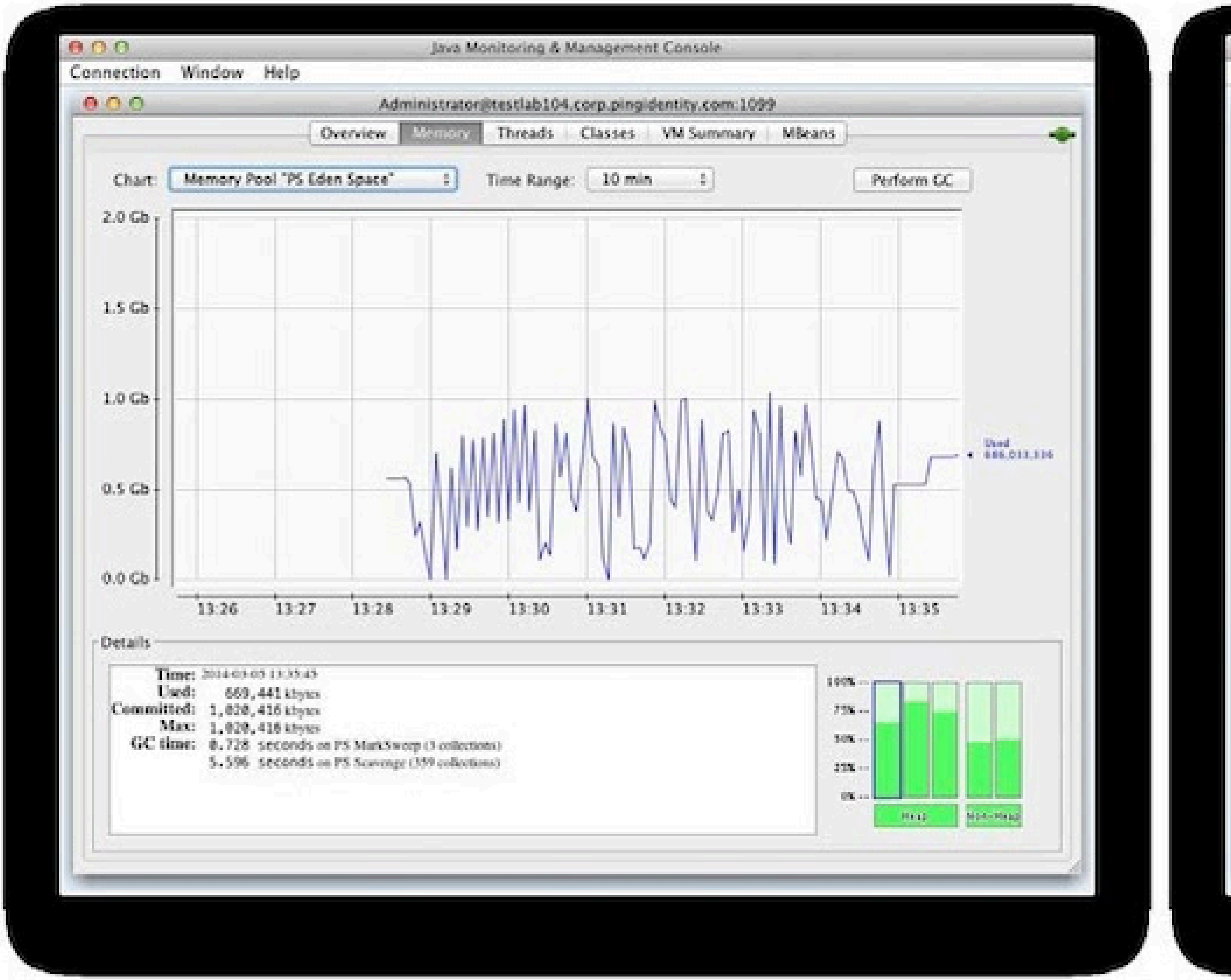
If the heap is sized appropriately for the load that the system must handle, it fills up and is followed by an appreciable drop in usage as a full garbage collection occurs, such as a PS MarkSweep collection triggered by the Old Generation filling up. In this example, the heap rises steadily, with drops from minor collections until a PS MarkSweep collection occurs and collects approximately 70% of the heap.



When the heap is undersized, full collections that are performed more frequently return less memory. In the following example, the frequency of Java Management Extensions (JMX) data that the JConsole retrieves does not keep pace with the frequency of full collections. As a result, only a fraction of them occur.

Eden space

Regardless of whether the heap is adequately sized or undersized, the usage pattern is nearly identical with the Eden space. This similarity can be due to the sampling frequency of the data-collection tool because the number of samples might be insufficient to show that, with an undersized heap, memory is consumed and subsequently freed with greater frequency. The behavior of garbage collection in the Eden space is such that when it fills, the space is completely emptied by moving live objects to the Survivor and Old Generation spaces. Under load, the pattern resembles a jagged sawtooth, as shown in the following examples of an adequately sized heap and an undersized heap.



Increasing heap size

Because garbage collectors manage memory in the Java Runtime Environment (JRE), simply increasing the size of the heap is not always the appropriate solution. The following table outlines the total heap size recommendations for the available garbage collectors, based on available CPU resources. For more information about garbage collectors, see *Garbage collector configuration reference*.

Total Heap Size Recommendations for Garbage Collectors

Garbage collector	Minimum recommended number of CPUs	Recommended heap size
Parallel	4	6 GB maximum
Concurrent Mark Sweep	12	4 - 6 GB minimum
Garbage First (G1)	12	6 GB minimum

If additional memory is unavailable, or if increasing the size of the heap is inadvisable because of these recommendations, the load that is handled by this instance is probably too high. In such instances, consider adding additional resources to your deployment. To verify whether the load for the instance is too high, check the CPU utilization.

To allow for the most efficient management of memory, set the minimum and maximum heap sizes to the maximum allowed values to avoid potentially expensive heap allocation resizing and divide it evenly between the young and old generations. If you are using the Garbage First collector, generational spaces are not specified through command line options because they are managed logically in real time. Even in such instances, we recommend setting the minimum and maximum heap sizes to the maximum allowed values. For more information about fine-tuning the JVM options in the `jvm-memory.options` file, see *Modifying the Java heap size* in the Performance Tuning Reference Guide.

Logging, reporting, and troubleshooting

This section provides a brief summary and purpose of the available logging, reporting, and troubleshooting for PingAccess.

PingAccess logs

The following table identifies the available PingAccess logs and their purposes.

PingAccess Logs and Purposes

Name	Purpose
`pingaccess_engine_audit.log`	Records transactions of configured resources. Additionally, the log records transaction details when PingAccess sends requests to PingFederate, such as security token service (STS), OAuth2, and JSON web signature (JWS).
`pingaccess_api_audit.log`	Records PingAccess administrative API transactions. These transactions represent activity in the PingAccess administrative console. If you are using scripts to configure PingAccess, this log also records transaction activity.
`pingaccess_agent_audit.log`	Records transactions between PingAccess Agents and the PingAccess Engine.
`pingaccess.log`	Primary troubleshooting log that records PingAccess runtime and administrative server activities.

Troubleshooting
The `pingaccess.log` file represents the primary troubleshooting log. However, the `pingaccess_engine_audit.log` and `pingaccess_agent_audit.log` files are also useful. Along with an HTTP trace from the browser, which can be generated from a debugging application like Fiddler, these files are helpful for identifying issues that must be resolved.

For more information about managing PingAccess logs, see *Configure logging*.

Creating an error-only server log

Modify your `log4j2.xml` file to set up a specific log to log only `ERROR` and higher notifications.

About this task
Monitor the `pingaccess.log` file for error-level messages. You can configure alerts to send notifications when events occur and to improve the monitoring of these events. Even when levels are down to a minimum, the server log generates large amounts of information in an active production environment. You

can set up a specific log to log only `ERROR` and higher alerts, which can be sent to a security information and event management (SIEM) tool, such as Splunk, when they occur.

To change your `log4j2.xml` file to enable a separate log file:

Steps

1. Create an appender.

   Tip:

   The simplest way to create an appender is to copy an existing one to use as a base.

   In the following example, the `RollingFile` is the same one that the `pingaccess.log` file uses. The bold text identifies items that have been changed.

```
<!-- Error Only Main Log : A size based file rolling appender -->
<RollingFile name="FILEERR"
          fileName="${sys:pa.home}/log/pingaccess.error.log"
          filePattern="${sys:pa.home}/log/pingaccess.error.log.%i"
          ignoreExceptions="false">
 <PatternLayout>
     <!-- Uncomment this if you want to use UTF-8 encoding instead of
 system's default encoding. -->
     <!--
     <charset>UTF-8</charset>
     -->
     <!--
         To Activate location information uncomment the following
 pattern,
         comment out the current pattern and set "includeLocation" to
 true
         in "com.pingidentity" async logger.
     -->
     <!--
     <pattern>%d{ISO8601} %5p [%X{exchangeId}] %c:%L - %m%n</pattern>
     -->
     <pattern>%d{ISO8601} %5p [%X{exchangeId}] %c - %m%n</pattern>
 </PatternLayout>
 <Policies>
     <SizeBasedTriggeringPolicy size="100000 KB"/>
 </Policies>
 <DefaultRolloverStrategy max="10"/>
</RollingFile>
```

2. Set the appender you created in step 1 for `AsyncRoot` at the end of your `log4j2.xml` file.

   The following example shows the necessary changes. In this example, the `level` attribute indicates the level of messages that are sent to the log file.

```
<!-- Root Logger-->
<AsyncRoot level="INFO" includeLocation="false" >
 <AppenderRef ref="File"/>
 <AppenderRef ref="FILEERR" level="ERROR"/>
</AsyncRoot>
```

3. Remove the attribute `additivity="false"` from all other loggers that contain a reference to the `File` appender.

   Example:

```
<AsyncLogger name="com.pingidentity" level="DEBUG" additivity="false"
```

```
            includeLocation="false">
```

Becomes:

```
<AsyncLogger name="com.pingidentity" level="DEBUG"
        includeLocation="false">
```

4. Make this change on all nodes within a cluster.

> **Tip:**
>
> To expedite this step, create a base file with the appropriate changes and copy it to all the nodes.

5. Restart the PingAccess Server.

Splunk audit log

PingAccess can enable and write audit logs for Splunk to effectively collect and analyze data from Java Management Extensions (JMX) MBeans.

You can *enable Splunk audit logs* and use them to create dashboards in Splunk. These logs record the same information as the default audit logs, but they are formatted to facilitate parsing for specific information when you create dashboards. All of the necessary information resides within the commented-out sections.

> **Note:**
>
> The link above provides instructions on how to set up the PingAccess for Splunk app, which can be found here: *https://splunkbase.splunk.com/app/5368*

Troubleshooting

This section covers troubleshooting for common issues with PingAccess.

Administrative SSO lockout

If you misconfigure Administrative single sign-on (SSO) and are locked out of the PingAccess UI, you can disable SSO and sign on using the native sign-on.

- *Editing run.properties to disable SSO* on page 544
- *Using the admin API to disable SSO* on page 545
- *Using the admin API and a new token to disable SSO* on page 545

Editing run.properties to disable SSO

If you can access the PingAccess system, or the PingAccess administrative node in a cluster, you can edit the `run.properties` file to disable single sign-on (SSO).

Steps

1. Sign on to the local PingAccess system.
2. Edit the `run.properties` file.

3. Change the `admin.auth` value from default to native.

   Example:

   ```
   admin.auth=native
   ```

4. Restart PingAccess.

Result

You can sign on normally and reconfigure the SSO from **Settings** # **Admin UI Authentication** # **Authentication Method**.

Using the admin API to disable SSO

If basic authorization was not disabled, use the admin API to disable single sign-on (SSO).

Steps

1. Sign on to the local PingAccess system and start a non-Internet Explorer (IE) browser.
2. Sign on to the API doc page at `localhost:9000/admin/api-docs/`.

   Use the normal administrator username, Administrator, and your password.
3. Click and expand the **Auth** section.
4. Select the **PUT /auth/oidc** item.
5. Enter the following code.

   ```
   {
   "enabled": false
   }
   ```

6. Click **Try it Out**.

Result

You can sign on normally and reconfigure the SSO for the admin API from **Settings** # **Admin UI Authentication** # **Authentication Method**.

Using the admin API and a new token to disable SSO

If basic authorization is disabled but admin API OAuth is enabled, you can also use the admin API to disable single sign-on (SSO).

Steps

1. Retrieve a valid token for admin API OAuth from your token provider.
2. Submit a PUT to `https://<pa-host>/pa-admin-api/<api version>/auth/oidc` with the valid access token, where *<pa-host>* is the `hostname:port` for the PingAccess admin node and *<api-version>* is the API version (`v3` on PingAccess 5.0 or later, and `v2` on 4.X).

   The request body must contain:

   ```
   {
    "enabled": false,
   }
   ```

Result

You can sign on normally and reconfigure the SSO from **Settings** # **Admin UI Authentication** # **Authentication Method**.

Collecting support data

When troubleshooting, Ping Identity Support might ask you to use the collect support data tool to compile information about your PingAccess installation.

About this task

The tool collects the following information by default:

- `<PA_Home>/bin`
- `<PA_Home>/log` (the most recent files of each type within a size limit)
- `<PA_Home>/conf` (configuration files)

The tool collects environment details, including:

- Files present and their sizes
- Certificate data
- Version data
- Java Virtual Machine (JVM) details

The tool also collects system details, depending on the operating system, including:

- Crontab
- Ifconfig
- Netstat
- Uname

If Ping Identity Support needs more information about the PingAccess installation than the default configuration provides, Support might ask you to add a data collector to the tool by modifying its `csd_configuration.yaml` file.

The tool consists of the following files in the PingAccess home directory:

- `bin/collect-support-data.bat`
- `bin/collect-support-data.sh`
- `tools/csd/csd_configuration.yaml`
- `tools/csd/csd-1.1.jar`

Steps

1. Using your PingAccess administrator account and a terminal, navigate to the `<PA_Home>/bin` directory.
2. Use one of the following commands to run the collect support data tool, depending on your operating system:
   Choose from:
   - On a Windows operating system, use `./collect-support-data.bat`.
   - On a Unix-based operating system, use `./collect-support-data.sh`.

   Note:

   If directed to do so by Ping support, you can use additional options with these commands. Run the command with the `--help` option, or see the *PingFederate and PingAccess Support Data Collector* knowledge base article for more information.

   As the tool collects data, it displays its progress and any errors. When it finishes collecting data, the tool places the data in a `.zip` file in the current directory. The file name format is `support-data-ping-$<hostname>-r-$<timestamp>.zip`.

3. Review any errors that the tool displayed during the process, and any added to the log file `support-data-ping-$<hostname>-r.log`.

   If needed, resolve the errors and run the tool again.
4. Send the support data `.zip` file to Ping Identity Support.

Minimizing the PingAccess cookie size

Reduce the size of the PingAccess cookie if its size causes problems in your environment.

The options presented here each reduce the PingAccess cookie size. The amount of reduction depends on your environment and can't be precisely quantified.

- When configuring the web session, enable the **Cache User Attributes** option. This option caches user information for use in policy decisions instead of including it in the cookie, reducing the cookie size. For more information, see *Creating web sessions* on page 300 and *Editing web sessions* on page 306 .
- When configuring the site, disable the **Send Token** option. This minimizes the amount of information forwarded to the site itself. For more information, see *Adding sites* on page 247, *Editing sites* on page 247, and *Site field descriptions* on page 248.
- When configuring web session management, select the simplest algorithms: `ECDSA using P-256 Curve` for the **Signing Algorithm** and `AES 128 with CBC and HMAC SHA 256` for the **Encryption Algorithm**. This change reduces the cookie size although the impact is not as significant as the other options. This option might not be possible depending on your environment's security needs. For more information, see *Configuring web session management settings* on page 299.