PingAccess®

Version 3.1

Ping Identity®

Copyright

PingAccess manuals

Version 3.1
February, 2015

Ping Identity Corporation
1001 17th Street, Suite 100
Denver, CO 80202
U.S.A.

Trademark

Disclaimer

Document Lifetime

Ping Identity may occasionally update online documentation between releases of the related software. Consequently, if this PDF was not downloaded recently, it may not contain the most up-to-date information. Please refer to the online documentation at *documentation.pingidentity.com* for the most current information.

From the web site, you may also download and refresh this PDF if it has been updated, as indicated by a change on this date: **February, 2015**.

Chapter 1

Getting Started

This Getting Started guide will help you install PingAccess and start using it quickly. There are two ways to use PingAccess to protect your web applications - gateway and agent. The way you choose depends on your network design and security requirements. A single PingAccess Server can protect many web applications using a combination of both techniques. See the *Deployment Guide* for a detailed discussion of deployment options and *Configuration by Use Case* in this section for configuration instructions.

To automatically configure PingAccess and PingFederate for demonstrating its base functionality, use the PingAccess Quickstart Demo App available for download from the Ping Identity *Downloads* page.

To get started with PingAccess:

1. *Install the Oracle JDK* if it isn't installed already.
2. *Install* and start PingAccess on either Windows or Linux.
3. Optional. *Configuration Properties* in the `run.properties` file.
4. Configure *settings* within PingAccess. What items you configure depends on your deployment plans.

For help in successfully configuring PingAccess to meet your use case, see *Configuration by Use Case*.

System Requirements

PingAccess is certified as compatible for deployment and configuration with the minimum system specifications defined below.

Software Requirements

Ping Identity has qualified the following configurations and certified that they are compatible with the product. Variations of these platforms (for example, differences in operating system version or service pack) are supported up until the point at which an issue is suspected as being caused by the platform or other required software.

Operating Systems

PingAccess has been 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 may be affected.

- Microsoft Windows Server 2008 R2 SP1
- Microsoft Windows Server 2012 Standard
- Microsoft Windows Server 2012 R2 Datacenter
- Red Hat Enterprise Linux ES 6.5
- Red Hat Enterprise Linux ES 7.0
- SUSE Linux Enterprise 11 SP3

Virtual Systems

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

> 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 in no way guarantees the performance and/or interoperability of any VM software with its products.

Java Runtime Environment

- Oracle Java 7 update 75 (64-bit)
- Oracle Java 8 update 31 (64-bit)

Supported PingFederate

- PingFederate 7.2
- PingFederate 7.3

Supported Browsers for End Users

- Chrome
- Firefox
- Safari
- Internet Explorer 8 and higher
- Android 5.0
- iOS 8

Supported Browsers for Admin Console

- Chrome
- Firefox
- Internet Explorer 9 and higher

Audit Event Storage (External Database)

- Oracle 11g R2

Hardware Requirements

> 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.

Minimum Hardware Requirements

- 4 CPU/Cores
- 2 GB of RAM
- 2.1 GB of available hard drive space

Minimum Hardware Recommendations

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

Install the Oracle JDK

The 64-bit version of JDK 7 (update 60 or higher) or JDK 8 (update 5 or higher) provides the supported environment for PingAccess.

> You must install the Oracle JDK before installing PingAccess.

To install the Oracle JDK for Windows and Linux:

1. Download and install Oracle JDK from *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.
2. Set the `JAVA_HOME` environment variable to the JDK installation directory path. Set the variable at either the system or user level.
3. Add the JDK `/bin` directory path to the beginning of the `PATH` variable for your platform so it is available for scripts that depend on it.

Install PingAccess

Install PingAccess by extracting the downloaded distribution ZIP file.

> On Linux we recommend that you install and run PingAccess under a local user (non-root) account.

To install PingAccess:

1. Ensure you are logged on to your system with appropriate privileges to install and run an application.
2. Verify that the JDK is installed and that environment and PATH variables are set correctly (see *Install the Oracle JDK*).
3. Extract the distribution ZIP file into an installation directory.
4. Request a license key via the *Request a License Key page* (www.pingidentity.com/content/pic/en/products/request-license-key.html).
5. Save the license key file in the directory `<pa_install>/conf` with the name `pingaccess.lic`.

   > PingAccess will not start without a valid license key file.

> If you are deploying PingAccess in a cluster configuration, see *Configure PingAccess Servers into a Cluster*.

Uninstall PingAccess

1. Make sure that PingAccess is not running (see *Start and Stop PingAccess*).
2. Delete the PingAccess installation directory.

Upgrading from an Earlier Release of PingAccess

Upgrading from PingAccess 3.1

To upgrade from PingAccess 3.1 to PingAccess 3.1.1, perform the following steps:

1. In PingAccess admin console, navigate to Settings > Backup
2. Perform a backup and download the zip file.
3. Unzip the PingAccess 3.1.1 distribution zip file to a new installation folder.
4. Follow the restore instructions as described on the Backup settings help page to restore the backup zip file to the new PingAccess 3.1.1 installation folder.

5. Copy pingaccess.lic from the PingAccess 3.1 installation to the conf folder in the new 3.1.1 installation.
6. (Optional) For each cluster engine node that is being upgraded, copy the file bootstrap.properties from the original conf folder to the new install.Shut down the older running PingAccess instance.Start the new PingAccess node.

> ⚠ If you are upgrading a clustered environment, shut down all PingAccess nodes in the cluster before restarting them.

For more information on installation of PingAccess, see the *Getting Started* on page 2 guide. For more information on clustering see *Configure PingAccess Servers into a Cluster* on page 81.

Upgrading from PingAccess 2.1 through PingAccess 3.0

Use the PingAccess Upgrade Utility to upgrade from PingAccess 2.1 through PingAccess 3.0 to version 3.1.1.

To run the upgrade utility, you will need the following:

- The PingAccess Upgrade Utility archive
- The PingAccess 3.1.1 zip file
- Your PingAccess 3 license file
- Login access to the PingAccess host, as the utility is run on the host
- Basic Authentication needs to be configured and enabled for the running PingAccess instance. Administrator Single Sign-On must be disabled for the upgrade.
- Administrator credentials for the running PingAccess instance

Copy these files to the system being upgraded, and unpack the PingAccess Upgrade Utility archive.

The upgrade utility starts an instance of PingAccess 3.1.1 with an administrative listener on port 9001. This port number can be changed using the run.bat/run.sh *-p* parameter.

Any warnings or errors encountered are recorded in log/upgrade.log, as well as on the screen while the utility is being run.

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

Running the PingAccess Upgrade Utility on Windows

From the upgrade utility's bin directory, execute the following command:

run.bat [-p <*admin_port*>] <*sourcePingAccessRootDir*> <*outputDir*> <*pingaccessZip*> <*newPingAccessLicense*>

Running the PingAccess Upgrade Utility on Linux

From the upgrade utility's bin directory, execute the following command:

./run.sh [-p <*admin_port*>] <*sourcePingAccessRootDir*> <*outputDir*> <*pingaccessZip*> <*newPingAccessLicense*>

Parameter Definitions

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

Parameter	Value Description
<*admin_port*>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
<*sourcePingAccessRootDir*>	The PA_HOME for the source PingAccess version
<*outputDir*>	The target directory which will contain the unpacked PingAccess distribution
<*pingaccessZip*>	The PingAccess distribution for the target version
<*newPingAccessLicense*>	The path to the PingAccess license file to use for the target version

In the context of an upgrade, "source" refers to the old version of PingAccess, and "target" refers to the new version.

Upgrading a cluster

To upgrade a cluster, perform the following steps:

1. Run the upgrade utility on the administrative console
2. Change the upgraded administrative console's `admin.port` value to a temporary value.
3. Start the upgraded administrative console
4. Perform any manual post-upgrade tasks recorded in the upgrade log
5. Shut down the upgraded administrative console
6. Change the upgraded administrative console's `admin.port` value back to the original value.
7. Run the upgrade utility on each engine node
8. Shut down the entire cluster
9. Start the upgraded administrative node
10. Start each upgraded engine node

It is important for any backout plan that all nodes in a clustered PingAccess be running the same software release.

Completing the Upgrade

At the end of a successful upgrade, the PingAccess Upgrade Utility will record any manual steps that require user intervention both on-screen and in log/upgrade.log. When the upgrade is complete, the source PingAccess installation is left running. Collect any information needed to complete these manual steps from the running instance, then shut down the source PingAccess server and start new PingAccess 3.1.1 server to complete the manual portion of the upgrade.

If you want to see details about the migrated configuration data, examine log/audit.log. For more information about these tasks, see *Post-Upgrade Tasks*.

Post-Upgrade Tasks

The upgrade utility may generate the following warnings indicating manual steps that need to be taken. The following table describes the steps that should be taken for common warning messages:

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 via 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.
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.

Warning Text	**Steps to Take**
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/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, it is likely that there will 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.
Resource '*ResourceName*' could not be migrated to the target version due to Application context root conflicts. Manual intervention is required.	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 action item because these resources cannot be grouped in the same application, but they would need to be in order 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 /intranet/sales and /intranet/hr on the front-end and rewrite out the /intranet on the backend.
Application '*ApplicationName*' contains OAuth rules, but authenticates users with a web session. Unexpected results may occur.	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 PA cookie and OAuth token in requests and PA would apply policy to the requests as configured. In 3.1,

Warning Text	Steps to Take
	however, an API application and web application are mutually exclusive so the semantics of this particular configuration cannot be preserved.
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 may 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 may 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 may 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 may be required.	Indicates that an Availability Profile was created for the Site during the upgrade. You may 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 may be required.	The `/pa` context root was allowed as a valid context root in PingAccess 3.0 and is no longer allowed.
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.

After the upgrade has completed, we recommend the following additional optional steps be performed:

1. Generate new obfuscated passwords for the `pa.jdbc.password`, `pa.jdbc.filepassword`, and `pa.keystore.pw` parameters in `conf/run.properties`:
   a. (Conditional) If you are on a Linux host, run `obfuscate.sh` *`password`*.
   b. (Conditional) If you are on a Windows host, run `obfuscate.bat` *`password`*.
   c. Copy the obfuscated password and paste it into the parameter in run.properties that corresponds to the password being re-obfuscated.

2. Review the *HTTP Requests* on page 75 configuration to ensure the use of the IP Source settings is appropriate for the environment. During the upgrade process, the **List Value Location** setting is changed from the default of **Last** to **First** to match the behavior from earlier releases.

Running PingAccess as a Windows Service

You can set up PingAccess to run in the background as a service on Windows running 64-bit processors.

> Before performing this procedure, ensure that PingAccess runs normally by manually starting the server (see *Run PingAccess for the First Time*).

This installation enables PingAccess to start automatically when Windows is started or rebooted.

To run PingAccess as a Windows service:

1. Install PingAccess.

   > Ensure JAVA_HOME is set as a system variable (see *Install the Oracle JDK*).
2. Ensure you are logged on with full Administrator privileges.
3. Start a Command Prompt as an Administrator.
4. In the Command Prompt, run the install-service.bat file located in <pa_install>\sbin\windows.
5. Access the Windows **Control Panel** | **Administrative Tools** | **Services**.
6. Right-click **PingAccess Service** from the list of available services and select **Start**. The service starts immediately and restarts automatically on reboot. (You can change the default **Start type** setting in the **Properties** dialog.)

Removing the PingAccess Windows Service

To remove the PingAccess Windows Service, run the `uninstall-service.bat` file located in `<pa_install>\sbin\windows`. Once the script has completed its process, manually remove the PA_HOME environment variable from the system.

Running PingAccess as a Linux Service

You can set up PingAccess to run in the background as a service on Linux. This enables PingAccess to start automatically when Linux is started or rebooted. The service will run as `root` user by default, or a specific user if specified.

> Before performing this procedure, ensure that PingAccess runs normally by manually starting the server (see *Run PingAccess for the First Time*).

To set up PingAccess as a Linux service, perform the following 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` and ensure that the user who will run the service has read and write permission 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 user name to run the service, or leave empty for default
5. Register the service by running the command "`chkconfig --add pingaccess`" from the `/etc/init.d` folder.
6. Make the service script executable by running the command "`chmod +x pingaccess`"

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

Once registered, you can use the `service` command to control the pingaccess service. The available commands are:

- start
- stop
- restart
- status - shows the status of the PingAccess service and the PID

For example, run the command "`service pingaccess restart`" to stop and restart PingAccess.

Configuring Multiple Instances of PingAccess to Run as Linux Services

For hosts running multiple instances of PingAccess that need to be started as a service, make the following adjustments to the above procedure for each instance:

- 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:
  - `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

To remove the PingAccess Linux Service, perform the following steps:

1. Stop the service by running `/etc/init.d/pingaccess stop` as root
2. Run `chkconfig --delete pingaccess` as root
3. (Optional) Delete the `/etc/init.d/pingaccess` script

Run PingAccess for the First Time

1. Start PingAccess by running the following script:

   (Windows) <pa_install>\bin\run.bat

   (Linux) <pa_install>\bin\run.sh

   The run.sh script requires bc, the GNU command line calculator. To install bc on SUSE, execute the following command: zypper install bc.

   Wait for the script to finish the start up. The server is started when you see the message " PingAccess running..." in the command window.

   If you are using the PingAccess Quick-Start Application, at this point there are initialization steps for completing your setup. See the Quick-Start Application's ReadMeFirst for more information.

   If you have not yet installed a PingAccess license, the server does not start up (see *Install PingAccess* for information on obtaining a license).
2. Launch your browser and go to: https://<DNS_NAME>:9000 where <DNS_NAME> is the fully-qualified name of the machine running PingAccess.
3. Log on with the default username and password supplied with the distribution.

Username: Administrator

Password: 2Access

4. Read and accept the license agreement.
5. Change the default administrator password on the **First Time Login** screen and click **Continue**.

> The new password must conform to the rules specified by the `pa.admin.user.password.regex` property in `run.properties`.

The PingAccess administrative console appears.

Start and Stop PingAccess

Linux

To start PingAccess:

1. From a command prompt, change directories to <pa_install>\bin.
2. Execute the run.sh shell script. Wait for the script to execute.

   The server is started when you see the message “PingAccess running...” in the command window.

To stop PingAccess:

Press Ctrl+C in the terminal window.

Windows

To start PingAccess:

1. From the **Start** > **Run** dialog or a command prompt, run the batch file: <pa_install>\bin\run.bat Or: Open the bin folder within your PingAccess installation and double-click the run.bat file.
2. Wait for the script to execute. The server is started when you see the message “PingAccess running...” in the command window.

To stop PingAccess:

1. Press Ctrl+C in the command-prompt window.
2. Press y to terminate the batch script when prompted.

All Platforms

To access the PingAccess administrative console:

1. Launch a Web browser window
2. Go to location `https://<DNS_NAME>:<PORT>` where `<DNS_NAME>` is the fully-qualified name of the machine running the PingAccess server and `<PORT>` is the port where the administrative console listens (default 9000). For example, `https://localhost:9000`.

Upon a successful login, PingAccess creates a backup of the current configuration to allow the administrator to revert any changes made. See the *Backup* section for more information.

Configuration by Use Case

Your next configuration steps depend on what type of deployment you are implementing. See the *Deployment Guide* for a detailed discussion of deployment considerations and best practices in designing your architecture. 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

Once you complete the above configuration settings, your next steps are similar for all use cases:

- Configure *Sites* and *Agents* to define the target applications to be protected. Sites may need *Site Authenticators* to define the credentials the site expects for access control.
- Configure *Applications* and *Resources* to define the assets you wish to allow clients to access.
- Create *Policies* for the defined applications and resources to protect them.

API Access Management Gateway Deployment

The following section describes the important configuration options for deploying an API Gateway. See *Deploying for Gateway API Access Management* in the *Deployment Guide* for specific use case information.

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 Resource Server OAuth Client.	The client must be registered with PingFederate and the client credentials configured in PingAccess to authenticate PingAccess when validating incoming Access Tokens.
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

Web Access Management Gateway Deployment

The following section describes the important configuration options for a Web Access Management Gateway deployment. See *Deploying for Gateway Web Access Management* in the *Deployment Guide* for specific use case information.

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.

Step	Description
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

The following section describes the important configuration options for a Web Access Management Agent deployment See *Deploying for Agent Web Access Management* for specific use case information.

First, PingAccess Agent needs to be deployed using the following steps:

1. Install PA Agent on Web Server - following instruction 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 via the download field in the Shared Secrets field.
3. Deploy the agent bootstrap.properties file to agents following instructions in *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.

Step	Description
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.

Auditing and Proxying Gateway Deployment

The following section describes the important configuration options for an auditing or proxying deployment (see *Deploying for Auditing and Proxying* for specific use case information).

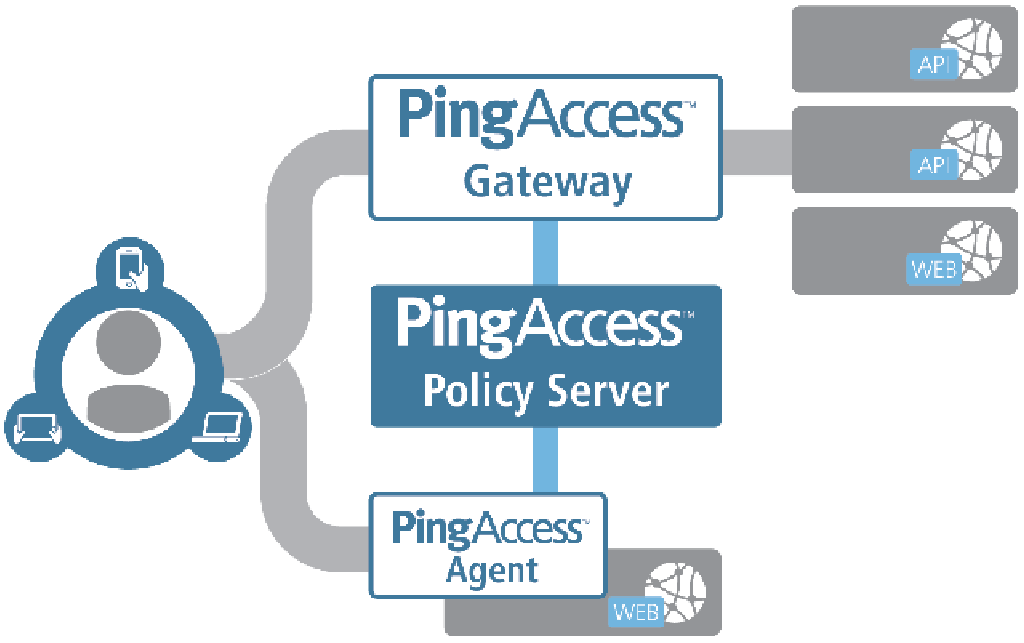
Step	Description
Generate or Import Key Pairs and *configure HTTP Listeners*.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical 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.

Chapter 2

PingAccess Overview

PingAccess protects Web Applications and APIs by applying security policies to client requests to determine if access is allowed. Requests can either be routed through PingAccess Gateway to the target Site or be intercepted at the target Web server by a PingAccess Agent which in turn communicates with PingAccess Policy Server. In either case, policies specified for the target Application are evaluated and PingAccess grants or denies access. When access is granted, the client request can be modified to provide additional identity information as needed by the target Application.

- *Sites* represent applications and APIs running on Web servers to be protected by PingAccess. They are defined by host and port settings to which PingAccess will forward authorized client requests.
- *Agents* represent Web server plugins deployed on Web servers to be protected by PingAccess. Agents contact PingAccess for authorization before allowing client requests to access the target assets.
- *Applications* represent applications and APIs which clients need to access securely and specify the information needed to protect them. Applications are composed of resources which have distinct access control requirements. Access rules can be applied to applications and their resources for greater flexibility.
- *Policies* are rules applied to applications and resources and determine if access is allowed. Rules are evaluated in the context of the client's identity and request characteristics.



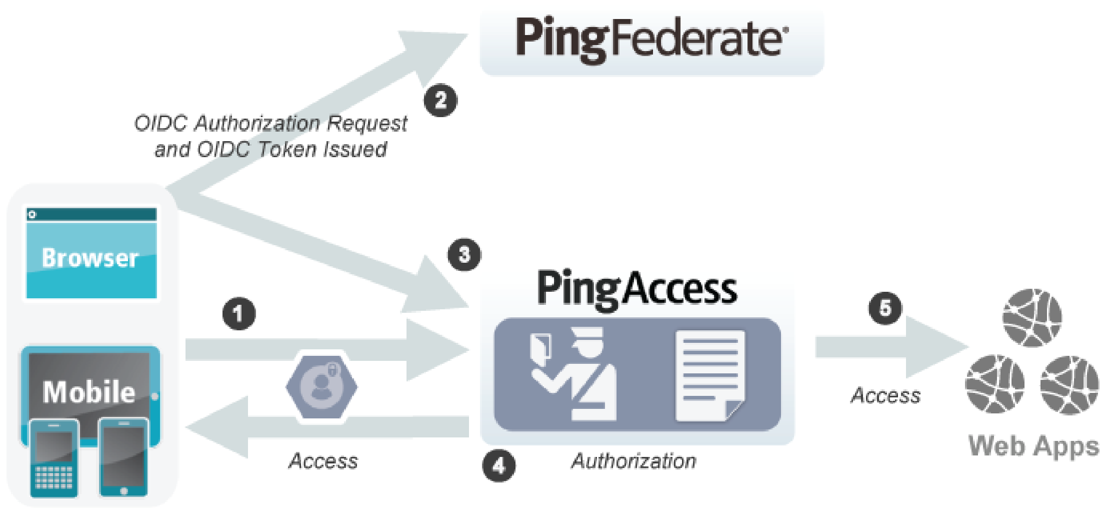
Web Access Management

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 resources to which they have permission. PingAccess uses Web Access Management (WAM) capabilities to allow organizations to manage access rights to Web-based resources. WAM is a form of identity management that controls access to Web resources, providing authentication and policy-based access management. Once a user is authenticated, PingAccess applies application and resource-level policies to the request. Once policy evaluation is passed, any required identity mediation between the back-end site and the authenticated user is performed. The user is then granted access to the requested resource.

PingAccess provides two deployment architectures for Web Access Management - 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 via 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.

WAM Session Initiation

Once a user authenticates, PingAccess applies the application and resource-level policies to the request. Once policy evaluation is passed, any required token mediation between the back-end Site and the authenticated user is performed. The user is then granted access to the Site



Processing Steps:

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

   When using an OP, an OAuth Client must already be configured in PingAccess. For steps on configuring an OAuth Client within PingFederate, see *Configuring a Client*. To then configure that OAuth Client within PingAccess, see the Web Session section on the *PingFederate* page.

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 PA Token and sends it to the browser in a cookie during a redirect to the original target resource. Upon gaining access to the resource, PingAccess evaluates application and resource-level policies and optionally audits the request.

   PingAccess can perform *Token Mediation* by exchanging the PA Token for the appropriate security token from the PingFederate 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).

See the *Web Sessions* section for more information.

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 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 PA Token is sent.

> 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.

Server-Side Session Management

The server-side session management feature allows for tighter session control, leveraging the single logout capabilities provided by PingFederate 7.2. The ability to enforce single logout enables the following scenarios:

1. PingAccess can reject a PingAccess cookie associated with a session that has been typically based on end user driven logout.
2. The end user can initiate a logout from all PingAccess issued web sessions using a centralized logout.

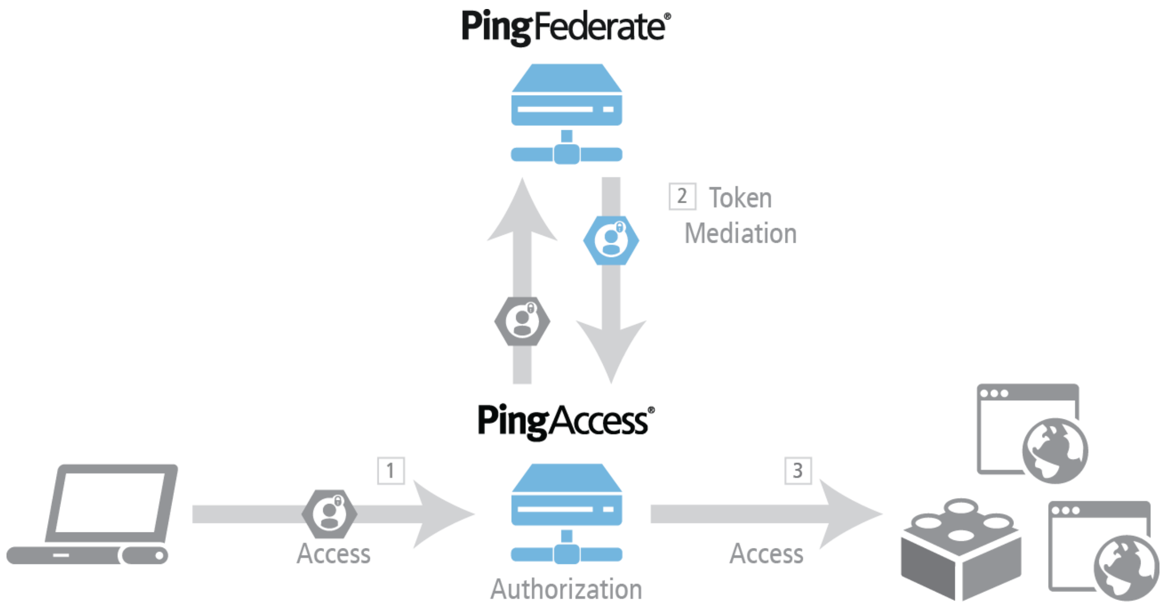
This feature performs a validation check with PingFederate when protected resources are served. The OpenID Connect option must be enabled in the OAuth 2.0 Authorization Server (AS) role, and access to the OpenID Connect session revocation API must be enabled.

Token Mediation

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

Token Mediation allows a PingAccess gateway to use a PingFederate token generator to exchange the PA 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. Once token mediation has occurred, the token used for accessing the application is cached for future use during the session.

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



Processing Steps:

1. A user requests a Resource from PingAccess with a PA Token or OAuth Bearer Token.

   > This example assumes the user has already obtained a PA Token or OAuth Bearer Token. See *Web Access Management* or *Using the OAuth Authorization Server* for details on how users authenticate with PingFederate and obtain a PA Token or OAuth Bearer Token.

2. PingAccess evaluates resource-level policies and performs token mediation by acquiring the appropriate security token from the PingFederate 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).

Using Virtual Hosts

Virtual Hosting enables you to host multiple server or domain names. This allows one server to share resources without requiring all sites on the server to use the same host name. For example, you may want to use multiple names on the same server so that each site name reflects the services offered rather than the actual server name where those sites are hosted.

PingAccess supports virtual hosting by serving requests bound for a set of defined server names and mapping them to requested applications. The target host header presented by the client can optionally be rewritten with the appropriate back-end host name. For example, say the host configured for Site One is `hr121.internal:80`. You configure Application One to use a virtual host of `hr.mycompany.com:80`. You associate Site One with Application One. PingAccess listens for incoming requests for the site at `hr.mycompany.com:80`. When a client request comes in to `hr.mycompany.com:80`, PingAccess sees the request, looks at the name of the domain configured for the back-end site, and replaces the target Host header with `hr121.internal:80`.

Supporting HTTPS requests causes additional complexity due to the need for SSL/TLS certificates. Prior to availability of SNI in Java 8, an HTTPS port could only present a single certificate. In order to handle multiple Virtual Hosts you have to 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 on the HTTPS Listeners page - see HTTPS Listeners.

PingAccess Clustering

PingAccess provides clustering features that allow a group of PingAccess servers to appear as a single system. When deployed appropriately, server clustering can facilitate high availability of critical services. Clustering can also increase performance and overall system throughput. It is important to understand, however, that availability and performance are often at opposite ends of the deployment spectrum. Thus, you may need to make some configuration tradeoffs that balance availability with performance to accommodate specific deployment goals.

In a cluster, you can configure each PingAccess engine, or node, as an administrative console, a replica administrative console, or a runtime engine in the *run.properties* file. Runtime engines service client requests, while the console server administers policy and configuration for the entire cluster (via the administrative console). The replica administrative console provides a backup copy of the information on the administrative node in the event of a non-recoverable failure of the administrative console node. A cluster may contain one or more runtime nodes, but only one console node and only one replica console node. Server-specific configuration data is stored in the PingAccess administrative console server in the run.properties file. Information needed to bootstrap an engine is stored in the *bootstrap.properties* file on each engine.

At startup, a PingAccess engine node in a cluster checks its local configuration and then makes a call to the administrative console to check for changes. How often each engine in a cluster checks the console for changes is configurable in the engine run.properties file.

Configuration information is replicated to all engine nodes. By default, engines do not share runtime state. For increased performance, you can configure engines to share runtime state by configuring cluster interprocess communication using the run.properties file (see *Cluster Configuration Settings*).

Runtime state clustering consists solely of a shared cache of security tokens acquired from the PingFederate STS for *Token Mediation* use cases using the *Token Mediator Site Authenticator*.

Using the OAuth Authorization Server

PingAccess supports the *Bearer Token Security Model* and the *Validation Grant Type* extension grant and uses an OAuth AS in the following ways:

- Works with OAuth Authorization Servers such as the PingFederate *OAuth AS* to authorize access to protected Resources.
- Protects applications by requiring an OAuth bearer access token (see *Section 2.1* of RFC 6750 for supported token transport details).
- Acts as an OAuth Resource Server, requesting validation from the OAuth AS for the bearer access token it receives from a client making a protected-resources call. The OAuth AS validates the access token and sends token attributes to PingAccess, which evaluates the returned OAuth details against policies set in the Applications section of the *Policy Manager*.
- Grants access to a Resource based on the use of Rules in combination with the OAuth AS validation.

Chapter 3

PingAccess Administrator's Manual

The PingAccess Administrator's Manual provides comprehensive reference information about configuring PingAccess.

Configure PingAccess

This section contains detailed instructions on configuring PingAccess using the Administrative Console. It is organized along the same lines as the administrative user interface.

Administrative Console Elements - covers common UI techniques and icons

Applications - covers configuration of applications and resources

Sites and Agents - covers configuration of Sites, Site Authenticators, and Agents

Policy Manager - covers the Policy Manager interface and rules

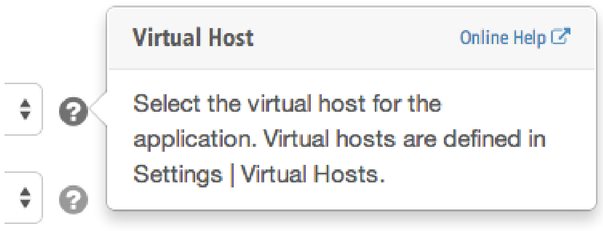
Settings - covers configuration of various global settings

Administrative Console Elements

The PingAccess Administrative Console is a rich web user interface with many icons and direct actions on objects. This page describes the techniques and icons used throughout the console.

Online Help

The Administrative Console provides extensive online help. When you hover the mouse over the help icon, a popup is presented with a brief explanation of the of the page, section, or field containing it. The popup includes a hyperlink to the relevant page of the Administrator's Manual for more detail.



Select the virtual host for the application. Virtual hosts are defined in Settings | Virtual Hosts.

Icons

- **Menu** - the menu icon indicates that one or more actions are available on the object. Click on the menu icon to see the list of actions, then click on the action to perform. Example actions are Edit, Delete and Download.
- **Edit** - the edit icon is used to modify object properties.
- **Delete** - the delete icon is used to delete the object.
- **Save** - the save icon is used to save the object values.

- **Remove** - the remove icon is used to delete objects.
- **Add** - the add icon is used to add new objects.
- **Download** - the download icon is used to download the object using the browser to the local computer.
- **Reorder** - the reorder icon is used to rearrange object order by dragging the object to the desired location.

Object Layout

Objects such as applications can be laid out in card view or row view by clicking one of the layout icons .

Object filtering

Many pages have a filter that enables narrowing down the number of available objects. As you type characters into the filter, only objects that meet the filter criteria are shown.

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 and correspond to a single target site. Applications also use a common Web Session and Identity Mapping. Access control and request processing rules can be applied to applications and their resources on the Policy Manager page to protect them. Applications can be protected by PingAccess Gateway or PingAccess Agent. In a gateway deployment, the target application is specified as a Site. In an agent deployment, the application destination is an Agent.

Use this page to define the applications which PingAccess protects and to which client requests are ultimately forwarded. You can 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.

Configure an Application

Applications represent Web applications and APIs to which a request is sent. Applications are composed of one or more Resources with a common Virtual Host and Context Root and correspond to a single Agent or Site. Applications also use a common Web Session and Identity Mapping. Rules can be applied to Applications and their Resources on the Policy Manager page to protect them.

Use this page to define the applications which PingAccess protects and to which client requests are ultimately forwarded. You can use Resources to partition the application into areas requiring different access control. Each Application contains at least the Root Resource. The combination of Virtual Server and Context Root must be unique for each Application and is used to match incoming requests. The matching process is greedy, so the Application with the longest matching Context Root will be selected. For example, if 2 applications are defined with the same Virtual Host and Context Roots `/myApp` and `/myApp/foo` respectively, a request with path `/myApp/foo/bar...` will match the second Application.

To configure an Application

1. Provide a unique **Name**. Name can be up to 64 characters long. Special characters and spaces are allowed.

2. Enter the **Context Root**, the first part of the URL path. It must begin with a slash and can contain additional slashes, but not end with one. For example, `/myApp`, or `/apps/myApp`. No wildcards or regular expression processing are allowed. The combination of Virtual Host and Context Root must be unique. In case of overlap, longest ContextRoot is matched first. Note that the value `/pa` is not allowed, as it is reserved for use by PingAccess.
3. Select the **Case Sensitive Path** checkbox to make request URL path matching case sensitive.
4. Select the **Virtual Host** for the Application from the list. Virtual Hosts are defined on the Settings | Virtual Hosts page.
5. Select the **Application Type** for the Application - Web or API. Web type applications may have a Web Session, while API type applications may present OAuth access tokens for validation by an Authorization Server.
6. Select the **Web Session** for the application from the list of defined web sessions. Only applies to applications of type Web. Web sessions are defined in Settings | Web Sessions.
7. Select the **Authorization Server** to use to validate OAuth access tokens. Only applies when Application Type is API. The PingFederate Authorization Server must be defined on the Settings | PingFederate page in order to select it here."
8. Select the application **Destination** type - Site or Agent. Destination type Site specifies a Gateway deployment wherein successfully authorized request are forwarded to the specified site. Destination type Agent specifies an agent deployment wherein the PingAccess Agent plugin is installed on the Web server hosting the application. See *Deployment Guide* for more details.
9. Select the **Site** to which requests are forwarded when access is allowed. Only applies when Destination is Site. Sites are defined on the Sites & Agents | Sites page.
10. Select the **Agent** which will be intercepting and validating access for requests to the Application. Only applies when Destination is Agent. Agents are defined on the Sites & Agents | Agents page.
11. Select the **Identity Mapping** to use for this application. Identity mappings make user attributes available to the target application as HTTP request headers. They are defined on the Settings | Identity Mappings page.
12. Select **Enabled** to indicate the application is active and can process requests.
13. Click **SAVE** when you finish, or the **/*** (Save and go to Resources) button to define resources for the application.

Configure a Resource

/*

Resources represent part of Web applications or APIs which have distinct security requirements. Resources specify what path prefixes and HTTP methods they apply to. Each application has at least the Root Resource which covers any requests not covered by other defined resources, and can have any number of additional resources. The first resource to match a request is used and no further resources are examined. Resources may specify particular authentication requirements, including no authentication if the Anonymous checkbox is selected. Use this page to view and edit existing application resources and to define new resources.

Resources are evaluated from most-specific to least-specific, based on the number of path elements and their lengths. For example, given the following list of resources:

- `/*` (The default root resource)
- `/MyApp/*`
- `/MyApp/Edit/*`
- `/MyApp/Admin/*`
- `/MyApp/Admin/ListUsers*`

The processing order would be:

- `/MyApp/Admin/ListUsers*`
- `/MyApp/Admin/*`
- `/MyApp/Edit/*`

- `/MyApp/*`
- `/*` (The default root resource)

In addition, when protecting an API, the HTTP method also is considered when identifying the order.

To configure a Resource:

1. Enter a unique **Name**. Up to 64 characters, including special characters and spaces, are allowed.
2. Enter a list of URL path prefixes (within the Context Root) that identify this resource. Prefixes must start with a slash (/) and may contain one or more wildcard characters (*). No use of wildcards is assumed, so there is a difference between `/app/` and `/app/*`.

   The path prefix starts after the application context root and extends to the end of the URL. It must begin with a forward slash. Note that `/pa` and `/pa*` are reserved prefixes and cannot be used as a path prefix when the context root is `/`.
3. If the resource does not require authentication for access, enable the **Anonymous** option.
4. Select an **Authentication Requirements** list to define how a user authenticates to the target site. This option only applies when you specify a **Web Session** for the application. See *Authentication Requirements*.
5. Enter the **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 API Resources. For example, if you have a server optimized for writing data (POST, PUT) and a server optimized for reading data (GET), you may want to segment traffic based on the operation being performed.

   Method selection only applies to applications of type API.
6. Select the **Audit** checkbox to log information about the transaction to the audit store.
7. Click **SAVE** when you finish.

The resource can be disabled by unchecking the **Enabled** control after the resource has been saved, or by sliding the slider associated with the resource in the resource card or list view.

Application and Resource Evaluation

Applications represent Web applications or APIs to which a request is sent. They are defined by a context root and virtual server which must be unique. The context root is the first part of the URL path, starting with a slash (/) and can be arbitrarily long or deep - that is, it can contain any number of slashes. No wildcards are allowed in the context root. For example, `/myApp` or `/hrApps/appOne`. When a request comes in, PingAccess needs to identify the application for the request by matching the request URL prefix against the defined applications. If two or more application context roots start with the same string, PingAccess will match the longest (and therefore most specific) context root first. For example, if App1 has context root `/hrApps` and App2 has context root `/hrApps/myApp`, then a request with URL `/hrApps/myApp/page1` will match App2.

Resources represent parts of the application URL space beyond the context root that have distinct security requirements. All applications have the default Root Resource which corresponds to all URLs not handled by other resources. Any number of additional resources can be defined. Each resource can specify an arbitrary number of URL path prefixes which may contain wildcards.

Resources can be defined in any order, as ordering is not used to determine precedence. Instead, the match that is the most specific for the requested URL path prefix is used to determine which resource was requested.

For example, suppose we have application called **App1** with a Context Root of "/" and the following resources:

- **Res1** with a path prefix of `/foo`
- **Res2** with a path prefix of `/*/bar`
- **Root Resource** with the default path prefix `/`

If a user requests the resource at `/foo/bar`, PingAccess would identify the requested resource as **Res2** and make policy decisions based on that identification.

In addition, resources that are associated with an API Application Type can be defined based on the method used with the URL. For example, suppose we have an Application called **App2** defined with the following resources:

- **Res3** with a path prefix of `/foo`
- **Res4** with a prefix path of `/*/bar` for the GET method
- **Res5** with a prefix path of `/*/bar` for the PUT method

If a user requests `/foo/bar` using a GET method, PingAccess would identify the requested resource as **Res4** and make policy decisions based on that identification. A request to `/foo/bar` using a PUT method, however, would result in PingAccess identifying the requested resource as **Res5**.

Sites & Agents

Sites & Agents is where you define application destinations - target sites or agents. Sites are the target applications or APIs which PingAccess Gateway is protecting and to which authorized client requests are ultimately forwarded to. Site Authenticators define the authentication mechanism that target sites require to control access.

Agents are instances of PingAccess agents with which Policy Server is expecting to receive agent requests from for target applications or APIs.

Sites

Sites are the locations of applications, endpoints, or APIs in your environment that you want to protect with PingAccess Gateway. Use this page to view and edit existing sites (and their resources) and to create new sites via the **NEW SITE** button.

Screen Navigation Tips

- Click **NEW SITE** on the top right to configure a new target site.
- Click the menu button (☰) on a site to edit or delete the site.

> ℹ If a Site is associated with an application, you cannot delete it.

- Use the **Filter** box on the right to search-as-you-type for sites by name.
- Click ▦ ☰ on the right to toggle the page view between **Card** and **Table** view.

To edit a site:

1. Enter a unique **Name**. Up to 64 characters, including special characters and spaces, are allowed.
2. Specify one or more **Targets** in the list of targets. The format for this is *`hostname`*`:`*`port`* . For example, you might enter `www.example.com:80`.
3. Select the **Secure** checkbox 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.
4. To override standard HTTPS certificate hostname validation with a named hostname, enter the hostname in the **Expected Certificate Hostname** field. This field is unavailable if **Skip Hostname Verification** is selected.

5. Select an *Availability Profile*.
6. Optionally select a *Load Balancing Strategy*, if the site contains more than one target.
7. If the Site requires authentication, click in the **Site Authenticators** box to select one or more authenticators from the list. Click **x** to remove a Site Authenticator.

   A combination of Site Authenticators is useful when the back-end Site requires a form of service-level authentication, but the application itself expects end-user related identity information. For example, combine the Basic Authentication and Web Session Header Attribute Site Authenticators to have PingAccess authenticate securely to the back-end site and relay user and group details via HTTP headers.

   > You must first configure Site Authenticators in order to populate this list (see *Configure Site Authenticators*).

8. Leave the **Use Target Host Header** checkbox selected to have PingAccess adjust the `Host` header to the Site's Target Host and Target Port rather than the Virtual Host configured in the application. This is often required by target Web servers to ensure they service the HTTP request with the correct internal virtual server definition. Clear this checkbox to make no changes to the `Host` header. (See *Using Virtual Hosts* for more information.)
9. In the **Keep Alive Timeout (ms)** box, enter the time (in milliseconds) an HTTP persistent connection to the Site can be idle before PingAccess closes the connection. The default is `30000` milliseconds.
10. In the **Max Connections** box, enter the maximum number of HTTP persistent connections you want PingAccess to have open and maintain for the Site. `-1` indicates unlimited connections.
11. Leave the **Send Token** checkbox selected to include the *PA Token* in the request to the back-end Site. Clear the checkbox to remove the PA Token from the request.
12. Click **SAVE** when you finish.

Site Authenticators

When a client attempts to access a target Web Site, that Site may 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/password authentication to certificate and token-based authentication. Create a Site Authenticator for the type of authentication the Site requires.

> For agent deployments, use *Identity Mappings* to expose user attributes in HTTP request headers to satisfy application security requirements. Identity Mappings are available for both agent and gateway deployments.

Screen Navigation Tips

- Click ***NEW SITE AUTHENTICATOR*** on the right to configure a new Site Authenticator.

  > Site Authenticators cannot be used with Agents.

- Use the **Filter** box on the right to search-as-you-type for Site Authenticators by entering the name or the authentication type.
- Click ☰ to access the edit and delete tasks for a Site Authenticator.

  > If a Site Authenticator is associated with a Site, you cannot delete it.

Configure Site Authenticators

When a client attempts to access a target Web Site, that Site may 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/password authentication to certificate and token-based authentication. Create a Site Authenticator for the type of authentication the Site requires. To create a Site Authenticator:

1. Enter a unique **Name**. Special characters and spaces are allowed. This name appears in the **Site Authenticator** list on the **New Site** page.
2. Select the type of authentication from the drop-down list.

   Basic Authentication Site Authenticator

   Mutual TLS Site Authenticator

   Token Mediator Site Authenticator
3. Click **SAVE** when you finish.

Basic Authentication Site Authenticator

This Site Authenticator uses HTTP Basic authentication (username:password) to authenticate a client requesting access to a Site requiring Basic authentication.

Obtain the username and password from your target Site provider.

Field	Description
Username	Defines the username required for access to the protected Site.
Password	Defines the password required for access to the protected Site.

Mutual TLS Site Authenticator

This Site Authenticator uses 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 be able to trust this certificate in order for authentication to succeed.

Several setup steps are required for PingAccess certificate management before configuring the Mutual TLS Site Authenticator (see *Key Pairs*, and *Certificates*).

Field	Description
Key Pair	The imported/generated key pair for client authentication. Select the Key Pair you want to use to authenticate PingAccess to the target Site (see *Key Pairs*).

Token Mediator Site Authenticator

This Site Authenticator uses the PingFederate STS to exchange a *PA Token* for a security token, such as a *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 within PingFederate (see *Configuring Token Generators* in the PingFederate documentation).
Logged In Cookie Name	Defines the cookie name containing the token that the target Site is expecting.
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 PA 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 page.
Logged Off Cookie Value	Defines the value placed in the Logged Off Cookie to detect an invalid/expired WAM Token event.
Source Token	Defines the token type exchanged for a security token during *identity mediation*. Leave PA Cookie for Web access or select OAuth Bearer Token for API identity mediation.
Send Cookies to Browser	Allows the Token Mediator to send the backend cookie defined in the **Logged In Cookie Name** field back to the browser if the protected application has updated it. This could be used to enable a seamless SSO experience for users navigating from PingAccess protected applications to those protected by a 3rd party Web Access Management system.

Agents

Agents are web server plugins that get 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 via the PingAccess Agent Protocol (PAAP) which defines in detail the possible interactions between agents and Policy Server. Agents have a name to identify them and a shared secret to authenticate with to 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.

Prior to defining an agent, import or create an Agent listener key pair and assign it to the AGENT HTTPS Listener by performing the following steps:

1. Perform the steps in the *Key Pairs* to **Import or Generate a Key Pair**. The key pair's subject or subject alternative names list need to include the host or hosts the agent will use to contact the PingAccess Policy Server.
2. Go to the **Listeners** configuration, and click the menu icon next to the **AGENT** listener, then click **Edit**.
3. Select the key pair created in step 1, then click **Save**.
4. Restart PingAccess.

If the environment is clustered, check the `pingaccess.log` file on each engine to ensure replication completed before restarting each engine.

The Policy Server needs to know each agent name and it's secret. The **Agents** page lists defined agents and allows the user to create, delete, and edit agents.

The **Edit Agent** page is used to change agent fields:

Field	**Description**
Name	A unique name for the agent. Can be up to 64 characters and must be alphanumeric. It may not include spaces or special characters.
Description	Description of the agent and it's purpose.
PingAccess Host	The host that the agent should send PAAP agent requests to.
PingAccess Port	The port that the agent should send PAAP agent requests to. This port is defined by the agent.http.port parameter in the PingAccess run.properties file. The default value is 3030.
Shared Secrets	The generated shared secret used to authenticate the agent. This field has buttons to add new secret, remove existing secret, and download agent `bootstrap.properties` file.

The PingAccess Host and Port may not be the actual host and port that Policy Server is listening to, 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.

The shared secret is generated by PingAccess server and identified on this page with a timestamp. Any number of secrets can be generated by clicking the + (Add new shared secret) button next to the secret field. Existing secrets can be deleted by clicking the **x** (Remove) button within the secret field. Maintaining multiple keys on the server facilitates key rotation, maintaining system availability during key updates. The server will accept any of the shared secrets from the agent.

Once an agent is saved, PingAccess can generate an agent `boostrap.properties` file containing the specified information and secret which can be used to configure the agent plugin. To download the `bootstrap.properties` file for an agent:

1. Edit an existing, saved agent by clicking the menu | Edit button.
2. Decide which specific shared secret to use. Only 1 can be included in the downloaded file.
3. Click on the **Download** button in the shared secret field of the chosen shared secret.

4. In a new browser tab or window PingAccess will bring up the file download dialog. The file name is of the form `<agent-name>_bootstrap.properties` by default.
5. Save the file, then distribute it to the intended agent installation. See either the *PingAccess Agent for Apache Installation* or the *PingAccess Agent for IIS Installation* documentation for more information.

Policy Manager

The Policy Manager is a rich drag-and-drop interface where you can manage policies by creating Rules, building Rule Sets, and applying Rules and Rule Sets to Applications and Resources. Policies are rules or set of rules applied to an 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 or Rule Sets, 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.

Ensure that any headers used in access control rules (such as `X-Forwarded-For`, which is used by Network Range rules) are sanitized and managed exclusively by inline infrastructure that users must be routed through before reaching PingAccess and the protected applications.

Rules

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.

Create a New Rule

To create a new rule, click the (New Rule) button in the Rule column which brings up the New Rule page.

1. Enter a unique **Name**. Up to 64 characters, including special characters and spaces, are allowed.
2. Select the **Type** of rule you want to implement. Fields associated with that Rule type appear below.
3. Enter the required information for the type of rule you are creating.
4. Click **SAVE** when you finish.

Rule Type Descriptions and Configuration

Access Control Rules

Groovy Script Rule

HTTP Request Rule

Network Range Rule

OAuth Attribute Value Rule

OAuth Groovy Script Rule

OAuth Scope Rule

Time Range Rule

Web Session Attribute Rule

Processing Rules

Rewrite Cookie Domain Rule

Rewrite Cookie Path Rule

Rewrite Response Header Rule

Rewrite URL Rule

Processing rules cannot be used with Agents.

Apply a Rule

To apply a rule to a Rule Set or application, drag the rule onto the drop box of an expanded Rule Set, Application, or Resource.

Edit a Rule

To edit a rule:

1. Click on the Rule you want to edit and click Edit. The Edit Rule page appears with the same fields as when the rule was created.
2. Make your edits.
3. Click **SAVE**.

Delete a Rule

1. Click on the Rule you want to delete and click Delete.
2. Click **DELETE** in the confirmation window.

If the Rule is associated with a rule set or an application or resource, you cannot delete it.

Rule Sets

Rule Sets allow you to group multiple Rules into re-usable sets which can be applied to applications and resources. The **Rule Sets** column displays available Rule Sets.

Create a Rule Set

1. Click at the top of the **Rule Sets** column.
2. Drag-and-drop a Rule from the **Rules** column onto the box that appears.
3. Enter a name for the Rule Set in the box that appears. Special characters and spaces are allowed.
4. Select All as the **Success Criteria** to require all Rules in the set to succeed. Select Any to require just one of the Rules in the set to succeed.

   When **Success Criteria** is set to **Any**, the first rule establishes the error handling and is flagged with an information icon . 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 with a warning icon . This is considered a misconfiguration because in an **Any** Rule Set, 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 as well as handle access control decisions using **Any** criteria, assign Processing Rules directly to the application or create a separate Rule Set for the Processing Rules using the **All** criteria."
5. Click to save the Rule Set.
6. Add more Rules.

Edit a Rule Set

1. Click on the Rule Set you want to edit and click Edit.

- Drag a Rule within a Rule Set up or down to re-order the Rules.
- Click on the Rule you want to remove from a Rule Set.
- Click to cancel Edit mode without saving changes.

2. Click to save your changes.

Apply a Rule Set

- Drag the Rule Set onto the drop box of an expanded application or resource to add it to create a policy.

Delete a Rule Set

1. Click on the Rule Set you want to delete and click Delete.
2. Click **DELETE** in the confirmation window.

If the Rule Set is associated with an application or resource, you cannot delete it.

Applications

Apply Rules and Rule Sets to an Application

1. Select an **Application** from the list
2. Expand the application **Resource**
3. Drag and drop Rules and Rule Sets onto the policy drop box that appears.

Edit an Application

1. Select an **Application** from the list
2. Expand the application **Resource**
3. Drag and drop new Rules and Rule Sets onto the policy drop box.
4. Drag Rules and Rule Sets around in the box to change the order in which Rules are evaluated at runtime.

   Ordering Rules may enhance performance. For example, to fail faster and improve performance as you move through PingAccess.

5. Click to remove a Rule or Rule Set from an application or resource.

Groovy Script Rule

Groovy scripts provide advanced Rule logic that extends PingAccess Rule development beyond the capabilities of the packaged *rules*. For more information on Groovy, see the *Groovy* and *Groovy Scripts* pages and the *official Groovy documentation*.

To create a Groovy Script Rule:

1. Enter the Groovy Script to use for Rule evaluation. For example, to create an OAuth Scope Rule that matches more than one scope, your Groovy script might look like the following: hasScopes("access","portfolio")

   See *Groovy Scripts* for more information.

2. Click **SAVE** when you finish.

Through Groovy scripts, PingAccess administrators can perform sensitive operations that could affect system behavior and security.

See *Advanced Fields* for information about error handling.

HTTP Request Rule

Examines a request and determines whether to grant access to a target Site based on a match found in the specified source of the HTTP request.

To configure an HTTP Request Rule:

1. Enter the **Field** name you want to match in order to grant or not grant the client access. For example, Host. This value is only relevant if the **Source** field is set to **Header**.
2. Enter the **Value** for the field (a header value or contents of the request body) you want to match in order to grant or not grant the client access. For example, localhost*. This value can be a Java regular expression.

   If you want to match on the Host header, be sure to include both the host and port as the **Value** or add a wildcard after the hostname (host* or host:*) to match what is in the HTTP request.
3. Select **Negate** if when a match is found, access is not allowed.

   Verify what you enter for the attribute. If you enter an attribute that does not exist (for example, misspell it) and you select **Negate**, the rule will always succeed.
4. Select the request **Source** you want to inspect for a match---**Header** or **Body**.

   If you want to match more values from the request, create more HTTP Request Rules or use the Groovy Script Rule.
5. Click **SAVE** when you finish.

See *Advanced Fields* for information about error handling.

Network Range Rule

Examines a request and determines 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).

To configure a Network Range Rule:

1. Enter a **Network Range** in the field. For example, 127.0.0.1/8. PingAccess supports both IPv4 and IPv6 addresses.
2. Select **Negate** if when a match is found, access is not allowed.
3. Click **SAVE** when you finish.

See *Advanced Fields* for information about error handling.

OAuth Attribute Value Rule

Examines a request and determines 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 OAuth Attribute Rule.

To configure an OAuth Attribute Value Rule:

1. Enter the **Attribute Name** you want to match to an attribute associated with an OAuth Access token. For example: Group.
2. In the **Attribute Value** box, enter a value for the attribute. For example, Sales.

   If you want to match more attributes, create more OAuth Attribute Value Rules or use the OAuth Groovy Script Rule.
3. Select **Negate** if when a match is found, access is not allowed.

   Verify what you enter for the attribute. If you enter an attribute that does not exist (for example, misspell it) and you select **Negate**, the rule will always succeed.
4. Click **SAVE** when you finish.

OAuth Groovy Script Rule

Determines whether to grant access to a target Site based on the results returned from a Groovy script that evaluates request details and OAuth details. This Rule allows you to create more sophisticated OAuth Scope and OAuth Attribute Value Rules.

See *Groovy Scripts* for more information.

To configure an OAuth Groovy Script Rule:

1. Enter the **Groovy Script** to use for Rule evaluation.
2. Enter **Realm** metadata returned to the client if a Rule fails. For example: "Company A's Partner API."
3. Click **SAVE** when you finish.

See *Advanced Fields* for information about error handling.

OAuth Scope Rule

Examines the contents of the PingFederate validation response and determines whether to grant access to a back-end target Site on a match found between the scopes of the validation response and scope specified in the OAuth Scope Rule. For example, a Resource may require that the OAuth Access Token contain the scope superuser.

To configure an OAuth Scope Rule:

1. Enter the **Scope** you want to match to values returned from the Access Validator.

   This is one scope requirement in the set of scopes associated with the access token.

2. Select **Negate** if when a match is found, access is not allowed.
3. Click **SAVE** when you finish.

Error-Handling Fields for OAuth Rules

You can customize an error message to display as part of the default oauth.error.json error page rendered in the end-user's browser if Rule evaluation fails for an OAuth-type Rule--*OAuth Attribute Value*, *OAuth Groovy Script*, and *OAuth Scope*. This page is among the templates you can modify with your own branding or other information (see *Customize User-Facing Pages*).

The response status code is always 401 with an Unauthorized status message. The WWW-Authenticate header value provides information on the OAuth credential the client needs to present.

For example:

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Bearer realm="test"

Use the following fields to configure the error handling template and content type.

Field	Description
Error Response Template File	The template page for customizing the error message that displays if Rule evaluation fails. This template file is located in the <pa_install>/conf/template/ directory.
Error Response Content Type	The type of content for the error response so the client can properly display the response.

Time Range Rule

Examines a request and determines whether to grant access to a back-end target Site based on the request falling within a defined time frame. For example, use this Rule when you want to restrict access to specific endpoints for certain time periods, such as during the work day from 8 am to 5 pm.

To configure a Time Range Rule:

1. Enter the beginning time for the time frame in the **Start Time** box. For example: 8:00 AM.
2. Enter the ending time for the time frame in the **End Time** box. For example: 5:00 PM.

   If you are using Internet Explorer or Firefox, you must enter the time in 24 hour format. For example, 5:00 PM is 17:00.

3. Select **Negate** if when a match is found, access is not allowed.

4. Click **Save** when you finish.

See *Advanced Fields* for information about error handling.

Rewrite Rules Overview

It is sometimes necessary to manipulate requests to Sites and their responses. PingAccess allows for the manipulation of the Request URI, the cookie domain, the cookie path, and three of the response headers (Location, Content-Location, and URI), as well as the response content.

For example, a Site is hosted on https://server1.internalsite.com under /content/. Users access the Site via the following URL in their browser:

https://server1.internalsite.com/content/

For example purposes, let's say 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 need to rewrite the content. The information below discusses an example scenario.

This example assumes that a Virtual Host, a Site, and an Application are already configured.

Create a Rewrite Content Rule

A *Rewrite Content Rule* on page 35 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 point to `https://server1.internalsite.com/content/` now point to `https://publicsite.com/importantstuff/`.

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/.

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/.

  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 Cookie Domain Rule

A *Rewrite Cookie Domain* Rule allows the rewriting of the Domain field on cookies when they are set by the back-end 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/.

Rewrite Content Rule

Use the Rewrite Content Rule to modify text in HTTP response bodies as it is served to the client. This rule uses a subset of the Java Regular Expression syntax that excludes look-behind constructs (for example, `\b`) and the boundary matcher (`\G`). If no Java regular expression syntax is used, the effect is to perform 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.

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 configure a Content Rewrite Rule:

1. Name the rule.
2. Select **Rewrite Content** from the list.
3. Enter one or more **Response Content-Types** to define what type of response data the rewrite rule applies to. The default values are `text/html`, `text/plain`, and `application/json`. The list is an ordered list

   Only text-based content types are supported. Text-based content types compressed with gzip, deflate, or compress will be decompressed prior to rewrite rule processing, however the content is not then re-compressed before being sent to the client. If the originating server does not specify a content-type header, this rule has no effect.
4. Define one or more set of **Find and Replace Criteria**. If multiple criteria are specified, each operation is performed against the original content - effectively applying the rule concurrently. Changes can affect CSS, Javascript, and other text-based elements served to the client, so it is very important to craft the regular expression appropriately to avoid modifying content that wasn't intended.
5. (Optional) If necessary, you can increase the size of the buffer used to perform the replace operation by opening the **Advanced** options and entering a larger buffer size. Replacement values cannot be larger than the buffer size. The minimum buffer size that can be specified is 1024 bytes; there is no maximum value.

Extensive use of Rewrite Content Rules may have significant performance implications.

Examples:

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	<b>(bold)</b>	not $1	This text is not bold.
Case-insensitive text replacement using a Java regex match flag	HTTP	text/html	(?i)http	FTP	FTP

Rewrite Cookie Domain Rule

Converts the cookie domain returned by the Site into a public-facing domain. For example, a Site places a cookie on a cookie domain such as internalsite.com (or .internalsite.com). Using the information configured in the Rewrite Cookie Domain Rule, PingAccess rewrites the Domain portion of the Set-Cookie response header with a public-facing domain such as publicsite.com (or .publicsite.com).

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 configure a Rewrite Cookie Domain Rule:

1. Name the Rule.
2. Select **Rewrite Cookie Domain** from the list.
3. In the **Server-Facing Cookie Domain** box, enter the domain name to use in the cookie returned by the back-end Site.
4. In the **Public-Facing Cookie Domain** box, enter the domain name you want to display in the response from PingAccess.
5. Click **SAVE** when you finish.

Rewrite Cookie Path Rule

Converts the cookie path returned by the Site into a public-facing path. This 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 configure a Rewrite Cookie Path Rule:

1. Name the Rule.

2. Select **Rewrite Cookie Path** from the list.
3. In the **Server-Facing Cookie Path** box, enter the path name where the cookie is valid for the back-end Site.
4. In the **Public-Facing Cookie Path** box, enter the path name you want to display in the response from PingAccess.
5. Click **SAVE** when you finish.

Rewrite Response Header Rule

Converts the response header value returned by the Site into a public-facing value. This Rule rewrites one of three response headers: Location, Content-Location, and 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 configure a Rewrite Response Header Rule:

1. Name the Rule.
2. Select **Rewrite Response Header** from the list.
3. In the **Server-Facing URI** box, enter the URI of the server sending the response. This must be a URI prefix containing at a minimum the protocol and hostname port information. For example: https://server1.internalsite.com/content/ or http://SiteHostName/path/
4. In the **Public Path** box, enter a valid URI path that you want to write into the URI. This must be a valid URI path and begin and end with a slash (/). For example: /importantstuff/ or /
5. Click **SAVE** when you finish.

Rewrite URL Rule

Examines the URL of every request and determines if a pattern matches. For example, you define a regular expression (regex) in the rule. If a pattern matches, PingAccess uses the information configured in the Rewrite URL Rule and rewrites that portion of the URL into a path that the Site can understand. The following table displays four example Rewrite URL Rule configurations:

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

To configure a Rewrite URL Rule:

1. Name the Rule.
2. Select **Rewrite URL** from the drop-down list.
3. In the **Map From** box, enter the regex of the URL's path and the query you want to match. For 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*).
4. In the **Map To** box, enter the URL's path and query you want to generate. For 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*).
5. Click **SAVE** when you finish.

Web Session Attribute Rule

Examines a request and determines whether to grant access to a target Site based on an attribute value match found within the *PA Token*.

To configure a Web Session Attribute Rule:

1. Enter the **Attribute Name** that you want to match in order to grant the client access. For example, Group.

2. Enter the **Attribute Value** for the Attribute Name. For example, Sales. If the attribute has multiple values at runtime, the attribute value you specify here must match one of those values.

   PA Token attributes are obtained from the PingFederate OpenID Connect Policy attribute contract (see *Configuring OpenID Connect Policies*).

3. Click ✚ to add more attributes. Click 🗑 to remove a row.
4. Select **Negate** if when a match is found, access is not allowed.

   Verify what you enter for the attribute. If you enter an attribute that does not exist (for example, misspell it) and you select **Negate**, the rule will always succeed.

5. Click **SAVE** when you finish.

To use this Rule, we recommend that you leave the **Request Profile** checkbox selected (see *Web Sessions*), indicating that you want PingAccess to request additional profile attributes from PingFederate when requesting the *ID Token*.

See *Advanced Fields* for information about error handling.

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 info about Groovy.

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. Click a link for more information on that object.

- *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 a *Application*.
- *Response Object*: Contains all information related to the *site* HTTP response.
- *Method Object*: Contains the HTTP method name from the request made to a *Application*.
- *Header Object*: Contains the HTTP header information from the request made to a *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.

Debugging/Troubleshooting

Groovy Script Rules are evaluated when saved to ensure that they are syntactically valid. If a Groovy Script Rule fails to save, check the log for output with the exception javax.script.ScriptException. For example, if you are trying to save a Groovy Script Rule that references the missing method foo(), the following output would be logged:

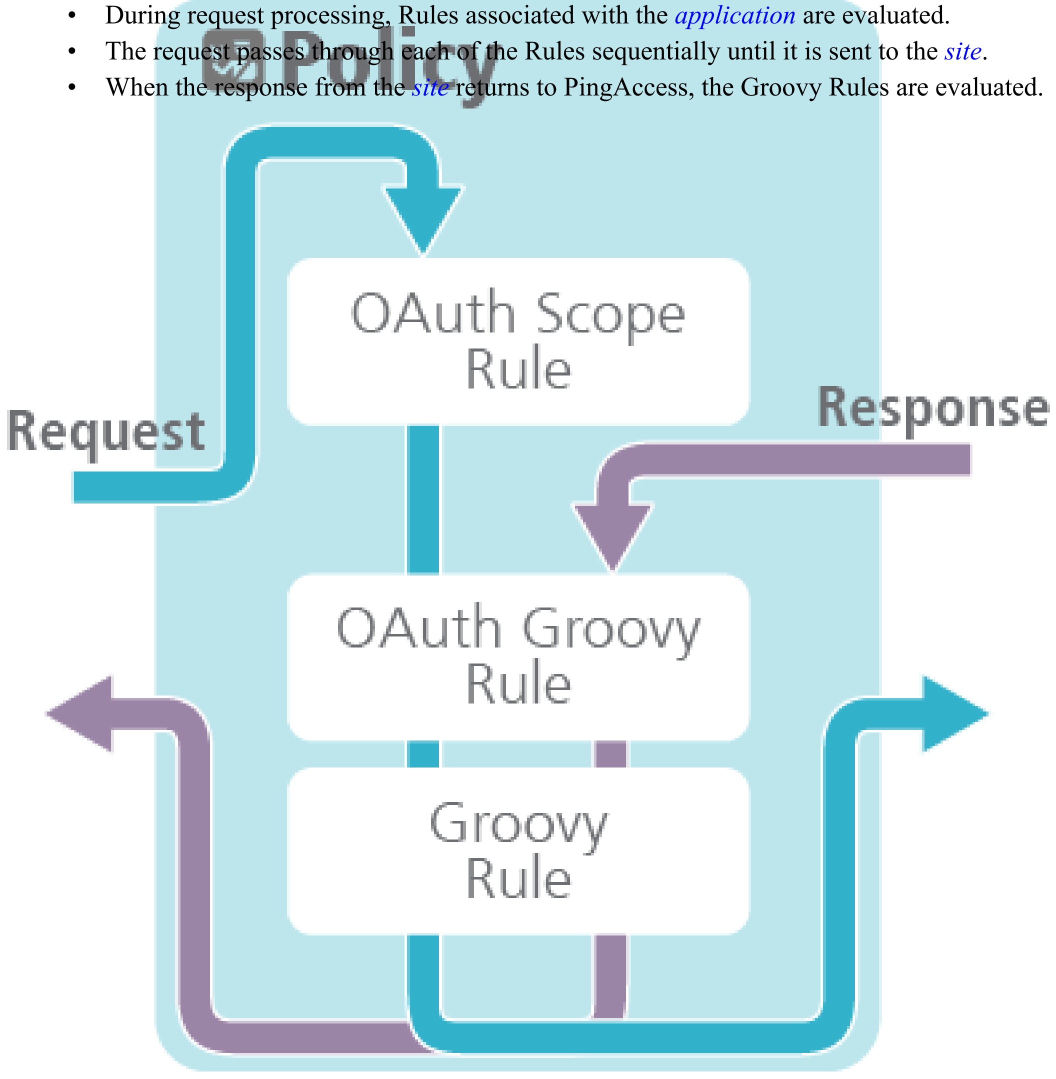
```
DEBUG com.pingidentity.synapse.adminui.AdminAPIInterceptor:1585 -
 javax.script.ScriptException:
javax.script.ScriptException: groovy.lang.MissingMethodException: No
 signature of method:
org.codehaus.groovy.jsr223.GroovyScriptEngineImpl.foo() is applicable for
 argument types: () values: []
Possible solutions: find(), any(), get(java.lang.String),
 use([Ljava.lang.Object;), is(java.lang.Object), find(groovy.lang.Closure)
DEBUG com.pingidentity.synapse.adminui.AdminAPIInterceptor:1399 - Returning
 error to UI: [[Error occurred validating policy.], {}]
```

These error messages are only logged if the DEBUG level output is enabled for the com.pingidentity logger.

Groovy

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 (see the *Groovy documentation*). 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. The diagram below highlights the flow of Rule processing.

- During request processing, Rules associated with the *application* are evaluated.
- The request passes through each of the Rules sequentially until it is sent to the *site*.
- When the response from the *site* returns to PingAccess, the Groovy Rules are evaluated.



Matchers

The *Groovy Script Rule* and the *OAuth Groovy Script Rule* 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*).

Example 1 - Simple Groovy Rule Inserts a Custom HTTP Header

```
test = "let's get Groovy!"
exc?.response?.header?.add("X-Groovy", "$test")
anything()
```

In the sample rule above, the script ends with a call to the Matcher anything(). The anything() Matcher signals that the rule has passed.

Example 2 - OAuth Groovy Rule Checks the HTTP Method and Confirms the OAuth Scope

```
//Get the HTTP method name
def methodName = exc?.request?.method?.methodName()
if (methodName == "POST") {
    hasScope("WRITE")
} else {
    not(anything())
}
```

In the sample rule above, 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.

The not(anything()) 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
inIpRange(String cidr)	Validates the source IP address of the request against the cidrString 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. **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. **Example**: inIpRange(InetAddress.getByName("127.0.0.1"),8)is equivalent to inIpRange("127.0.0.1/8")
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")
requestHeaderDoesntContain(String field, String value)	Validates that the HTTP header field value is not equal to the value parameter. **Example:** requestHeaderDoesntContain("User-Agent", "InternetExplorer")

Matcher	Description
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 *PA Token* contains the attribute name and value. **Example:** containsWebSessionAttribute("sub", "sarah")

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")

Exchange Object

Accesses the Exchange object in Groovy - `exc`

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.

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. The Exchange object can be used 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 Field Summary and Method Summary tables below 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")
    anything("Custom header added to request")
}
else
{
    println("Request content is empty") //Debugging statement
    not(anything("Request has no content"))
}
```

Field Summary

Field	Description
Request request	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 response	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.
long timeReqSent	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 timeResReceived	Obtains the time, in milliseconds, when the response was received from the *site*. This field is not available in scripts used with an Agent.

Method Summary

Method	Description
String getRequestURI()	Returns the PingAccess URI that received the request.
MediaType getRequestContentType()	Convenience method that returns the request Content-Type. This method works the same as exc?.request?.contentType.
int getRequestContentLength()	Convenience method that returns the request Content-Length. This method works the same as exc?.request?.contentLength.
MediaType getResponseContentType()	Convenience method that returns the response Content-Type. This method works the same as exc?.response?.contentType. This method is not available in scripts used with an Agent.
int getResponseContentLength()	Convenience method that returns the response Content-Length. This method works the same as exc?.response?.contentLength. This Method is not available in scripts used with an Agent.
Object getProperty(String key)	Returns the value of a custom property.
void setProperty(String key, Object value)	Sets a custom property.

PolicyContext Object

Accesses the Policy Context object in Groovy policyCtx

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 Script Rule*.

Groovy Sample

```
def oauthToken = policyCtx?.context.get("oauth_token")
```

Field Summary

Field	Description
java.util.Map context	Container for the *OAuthToken object*.

Request Object

> Accesses the Request object in Groovy exc?.request

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?.isJSON()

//Check to make sure the request body contains JSON
if (!request) {
not(anything("The request requires a JSON body"))
} else {
    anything("The request contains JSON")
}
```

Field Summary

Field	Description
String uri	Returns the PingAccess URI that received the request.
Method method	Contains the HTTP method information from the request sent to the *application*.
Header header	Contains the HTTP header information from the request sent to the *application*. **Warning**: Previously executed custom Rules can modify these values.
Body body	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 Summary

Method	Description
boolean isXML()	Returns true if the Content-Type header is set to xml.
boolean isJSON()	Returns true if the Content-Type header is set to application/json.
boolean isHTML()	Returns true if the Content-Type header is set to text/html.
boolean isBodyEmpty()	Returns true if the Content-Length header is set to zero or the HTTP body has zero length.

Response Object

Accesses the Response object in Groovy exc?.response

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

```
// Intercept a server error (status code = 500) return a failure
def response = exc?.response

if (response?.isServerError()) {
 not(anything("A server error occurred"))
}
```

Field Summary

Field	Description
int statusCode	Contains the HTTP response status code.
String statusMessage	Contains the HTTP response status message.
Header header	Contains the HTTP header information from the request sent to the *application*. **Warning**: Previously executed custom Rules can modify these values.
Body body	Contains the HTTP body information from the request sent to the *application*. **Warning**: Previously executed custom Rules can modify these values.

Method Summary

Method	Description
boolean isRedirect()	Returns true if the status code is in the 300's.
boolean isUserError()	Returns true if the status code is in the 400's.
boolean isServerError()	Returns true if the status code is in the 500's.

Method	Description
boolean isBodyEmpty()	Returns true if the Content-Length header is set to zero or the HTTP body has zero length.

Method Object

ⓘ	Accesses the Method object in Groovy exc?.request?.method

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()
}
```

Field Summary

Field	Description
String methodName	Returns the name of the HTTP Method (GET, PUT, POST, DELETE, HEAD).

Header Object

ⓘ	Accesses the Header object in Groovy exc?.request?.header or exc?.response?.header

The Header 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 Header object to add custom HTTP headers for site.

Groovy Sample

```
//Set a custom header for the Service request
def header = exc?.request?.header;
header?.add("X-PINGACCESS-SAMPLE", "SUCCESS");
anything("Custom header set into request");
```

Method Summary

Method	Description
void add(String key, String val)	Adds HTTP header fields for the request. Note that 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.
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 user, String password)	Sets authentication credentials for HTTP Authentication.
String getConnection()	Returns the connection type preferred by the User-Agent.
void setConnection(String connection)	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.
boolean hasContentLength()	Returns true of the Content-Length header is set.
void setContentType(String value)	Sets the request body MIME type.
Map getCookies()	Returns all cookies sent with the request.
void setCookie(String value)	Sets a cookie.
String getFirstCookieValue()	Returns the first cookie in the Cookie header.
String getFirstValue(String value)	Returns the first value of the HTTP header specified by the value.
void setDate(Date date)	Sets the date of the message in the Date HTTP header.
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.
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.
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.

Method	Description
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.

Body Object

Accesses the Body object in Groovy exc?.request?.body or exc?.response?.body

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());
anything("Content-Length header set");
```

Method Summary

Method	Description
byte[] getContent()	Returns the body content of the request or response.
int getLength()	Returns the length of the body content.

OAuth Token Object

Accesses the OAuth Token object in Groovy `exc?.user?.policyContext?.context?.get("oauth_token")`

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 = exc?.user?.policyContext?.context?.get("oauth_token")?.scopes
def attr = exc?.user?.policyContext?.context?.get("oauth_token")?.attributes
def username =
 exc?.user?.policyContext?.context?.get("oauth_token")?.attributes?.get("username")?.get
exc?.request?.header?.add("x-scopes", "$scopes")
exc?.request?.header?.add("x-attributes", "$attr")
exc?.request?.header?.add("x-username", "$username")
anything()
```

Field Summary

Field	Description
Date expiresAt	Contains the expiration date of the OAuth access token.
Date retrievedAt	Contains the date that the OAuth access token was retrieved from PingFederate.
String tokenType	Contains the type of OAuth access token. (Bearer, *JWT*).
String clientId	Contains the client ID associated with the OAuth access token.
Set scopes	Contains the set of scopes associated with the OAuth access token.
Map<String, List> attributes	Contains a map of identity attributes specific to the user.

Groovy Script Examples

OAuth Policy Context Example

In some instances, it may 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 via 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")
anything()
```

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'
  not(anything("Test header is foo"))
}
else
{
  //rule will pass evaluation is Test header has value of anything else
  //or isn't present
  anything("Test header is something else")
}
```

Advanced Fields for Rules

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 (see *Customize User-Facing Pages*). Use the following fields to configure the error handling.

Field	Description
Error Response Code	The HTTP status response code you want to send if Rule evaluation fails. For example, `403`.
Error Response Status Message	The HTTP status response message you want to return if Rule evaluation fails. For example, `Forbidden`.
Error Response Template File	The HTML template page for customizing the error message that displays if Rule evaluation fails. This

Field	Description
	template file is located in the `<pa_install>/conf/template/` directory.
Error Response Content Type	The type of content for the error response so the client can properly display the response.

OAuth Rule Advanced Fields

You can customize an error message to display as part of the default oauth.error.json error page rendered in the end-user's browser if Rule evaluation fails for an OAuth-type Rule--*OAuth Attribute Value*, *OAuth Groovy Script*, and *OAuth Scope*. This page is among the templates you can modify with your own branding or other information (see *Customize User-Facing Pages*).

The response status code is always 401 with an Unauthorized status message. The WWW-Authenticate header value provides information on the OAuth credential the client needs to present. For example:

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Bearer realm="test"

Use the following fields to configure the error handling template and content type.

Field	Description
Error Response Template File	The template page for customizing the error message that displays if Rule evaluation fails. This template file is located in the <pa_install>/conf/template/ directory.
Error Response Content Type	The type of content for the error response so the client can properly display the response.

Settings

The Settings pages provide access to a number of global settings that control PingAccess behavior and enable definition of artifacts used by applications and resources.

For help in successfully configuring PingAccess to meet your use case, see *Configuration by Use Case*.

Select a link below for more information:

Virtual Hosts - define virtual host names and ports for target applications.

Web Sessions - configure secure Web Sessions for use with specific applications and to configure global Web Session settings.

Authentication Requirements - identify how a user must authenticate before access is granted to protected resources.

Identity Mappings - make user attributes available to protected applications via HTTP headers. ***PingFederate*** - configure the connection to the PingFederate server runtime, OAuth Authorization Server, and Administration API.

Clustering - define clustering configuration including the administrative console and runtime engines.

Key Pairs - manage, import, and generate Key Pairs for HTTPS and mutual TLS authentication.

Certificates - import and manage certificates for HTTPS and create and manage trusted certificate groups.

HTTPS Listeners - assign Key Pairs to HTTPS Listeners and virtual hosts.

Admin Authentication - change the default PingAccess administrator authentication method.

Backup - generate backup of PingAccess configuration.

Availability Profiles - define failover behavior when a backend target server is not available.

Load Balancing Strategies - define strategies for balancing the load across a cluster of backend target servers.

HTTP Requests - define headers to inject to match a served resource with the originating client.

Virtual Hosts

Using *Using Virtual Hosts* enables PingAccess to protect multiple domain names in a single instance transparently to the clients. A Virtual Host is the combination of hostname and port name. For example, `siteOne:1000` and `siteTwo:2000`. Virtual Hosts are used in combination with Context Root to define Applications to which access control rules can be applied.

You can use a wildcard (*) for hostname which results in a Virtual Host that accepts any request to the specified port. For example, the Virtual Host `*:80` will accept requests to `myCompany:80` and `yourCompany:80` and any other host on port 80. Wildcard Virtual Hosts can be used as catch-all hosts for defining Applications - see *Configure an Application*.

Supporting HTTPS requests causes additional complexity due to the need for SSL/TLS certificates. Prior to availability of SNI in Java 8, an HTTPS port could only present a single certificate. In order to handle multiple Virtual Hosts you have to 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 on the HTTPS Listeners page - see *HTTPS Listeners*.

The Agent Resource Cache TTL advanced field is used to control PingAccess Agent resources for each virtual host.

Field	Description
Host	Enter the host name for the Virtual Host. For example: myHost.com. You can use a wildcard (*) to indicate that any host name is acceptable.
Port	Enter the integer port number for the Virtual Host. For example: 1234.
Agent Resource Cache TTL	Enter an integer indicating the number of seconds the Agent can cache resources for this application. Only applies to destination of type Agent.

Web Sessions

Web Sessions define the policy for Web application session creation, lifetime, timeouts, and their scope. Multiple Web Sessions may be configured 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 this page to configure secure Web Sessions for use with specific applications and to configure global Web Session settings.

The table below describes the fields used to configure a Web Session. Click **SAVE** when you finish. A new Web Session card appears on the Web Session page.

Field	Description
Name	The Web Session name. Enter a unique name. Up to 64 characters, including special characters and spaces, are allowed.
Cookie Domain	The valid domain where the cookie is stored. Enter a valid domain for the cookie. For example, corp.yourcompany.com. If you set the Cookie Domain, all of your web resources must reside within that domain. If you do not set the Cookie Domain, the *PA Token* is recreated for each host domain where you access applications.
Cookie Type	The type of token you want to create. An **Encrypted JWT** token uses authenticated encryption to simultaneously provide confidentiality, integrity, and authenticity of the PA Token. A **Signed JWT** token 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. **Signed JWT** is the default setting. Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
Audience	Defines who the PA Token is applicable to and is represented as a short, unique identifier. Enter a unique identifier between 1 and 32 characters. 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 application. Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
OpenID Connect Login Type	Defines how the user’s identity is verified based on authentication performed by an OpenID Provider and how additional profile claims are obtained. Three login profiles are supported: **Code** and **POST**, and **x_post**. Select a login profile. **Code** A standard OpenID Connect login flow that provides confidentiality for sensitive user claims. In this profile the relying party (PingAccess) makes multiple back-channel requests in order to exchange an authorization code for an ID Token and then exchange an access

Field	Description
	token for additional profile claims from the UserInfo endpoint at the provider (PingFederate). This login type is recommended 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. A form auto-POST response containing the ID Token (including profile claims) is sent to PingAccess from PingFederate via 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 recommended for maximum performance in cases where the exchanged claims do not contain information that should be hidden from the end user. Be sure to select the **Implicit** grant type when configuring the OAuth Client within PingFederate (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 OpenID Connect that passes claims from the provider via the browser. As with the **POST** login type, 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. If PingFederate 7.3 is used in the environment, we recommend using **POST** rather than **x_post**, as **x_post** was defined by Ping Identity prior to the development of the OAuth 2.0 Form Post Response Mode draft specification.
Client ID	Assigned when you created the OAuth Relying Party client within PingFederate (for more information, see *Configuring a Client* in the PingFederate documentation). Enter the unique identifier (Client ID).
Client Secret	Assigned when you created the OAuth Relying Party Client within PingFederate. Required when configuring the **Code** Login Type. Enter the secret (Client Secret). The OAuth Client you use with PingAccess Web sessions must have an OpenID Connect policy specified (for more information see *Configuring OpenID Connect Policies* in the PingFederate documentation).

Field	Description
Secure Cookie	Indicates that the PingAccess cookie must be sent using only HTTPS connections. Selected by default. ⚠ 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	When selected (the default), enables the HttpOnly flag on cookies that contain the PA Token. An HttpOnly flagged cookie is not accessible using non-HTTP methods such as calls via JavaScript (for example, referencing document.cookie) and therefore cannot be easily stolen via cross-site scripting.
Request Profile	When selected (the default), PingAccess requests additional profile attributes from PingFederate when requesting the *ID Token*. To use this feature, you must have a **profile** scope set up in PingFederate (see *Configuring a Client*). The **profile** scope is a standard OpenID Connect-defined scope that defines extended claims about a user. ⚠ 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	When selected, PingAccess will validate sessions with the configured PingFederate instance during request processing. Use of this feature requires additional configuration in PingFederate (see *Configure Server-Side Session Management*). This option is not selected by default. Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.
Enable Refresh User	When enabled, 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 a token in PingFederate causes the token to be revoked. For example, if the PingFederate OpenID Connect Policy 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.

Field	Description
	The refresh 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. This interval can be tuned by adding `pa.websession.refreshSessionInterval` to `conf/run.properties` and assigning it a value in seconds. Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources. This option is selected by default.
Cache User Attributes	When enabled, this option causes PingAccess to cache 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 may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources. This option is not selected by default.
Max Timeout (Minutes)	Defines the maximum amount of time, in minutes, that the PA Token remains active (the default is 240 minutes). Enter, in minutes, the length of time you want the PA Token to remain active. Once the PA 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.
Idle Timeout (Minutes)	Defines 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). Enter, in minutes, the length of time you want the PA Token to remain active when no activity is detected. Defining an idle expiration protects against unauthorized use of the resource by limiting the amount of time the session remains active when not being used. For example, idle expiration is useful when a user is no longer at the computer and does not log out of the session. When the idle expiration is reached, the session automatically terminates.

Field	Description
	ℹ If there is an existing valid PingFederate session for the user, an idle time out of the PingAccess session may result in its re-establishment without forcing the user to log in again.
Consult Server Duration (Seconds)	Defines the maximum amount of time, in seconds, that a PingAccess Agent caches policy decisions for the web session before sending a request to the Policy Server. This option only applies to agents. ℹ The value used for this setting should not be larger than the Idle Timeout value, and ideally, should be defined to be a value less than half the timeout.

Manage Web Sessions

This section provides steps for creating, editing, and deleting Web Sessions. To create a new Web Session:

1. On the Web Session page, click **NEW WEB SESSION**.
2. Enter the requested information on the form.
3. Click **SAVE**. A new card appears for the Web Session on the Web Session Page.

To edit a Web Session:

1. Click ☰ on the Web Session you want to edit and click ✎ Edit.
2. Make your edits.
3. Click **SAVE**.

To delete a Web Session:

1. Click ☰ on the Web Session you want to delete and click 🗑 Delete.
2. Click **DELETE** in the confirmation window.

ℹ If the Web Session is currently associated with an application, you cannot delete it.

Manage Global Web Session Settings

The table below describes the fields for configuring global Web Session settings.

Field	Description
Number of Keys to Cache	Indicates the number of keys you want to keep in history for validation (the default is 3). Enter how many keys you want to remain valid. For example, if you want to cache three keys and the key roll interval is every 24 hours, once a key rolls, the previous key is valid for 48 more hours.
Signing Key Roll Interval (Hours)	Indicates how often (in hours) PingAccess rolls the signing and encryption keys. Enter 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 *PA Tokens*.
Issuer	A published, unique identifier to be used with the Web session (The default is PingAccess). For example, set

Field	Description
	the issuer to a value that more closely represents your company. PingAccess inserts this value as the iss claim within the PA Token.
Signing Algorithm	The algorithm used to protect the integrity of the PA Token (the default is ECDSA using P-256 Curve). Select the signing algorithm you want to use from the list. PingAccess uses the algorithm when creating signed PA 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 PA Tokens are encrypted.
Encryption Algorithm	The algorithm used to encrypt and protect the integrity of the PA Token (the default is AES 128 with CBC and HMAC SHA 256). Select the encryption algorithm you want to use from the list. PingAccess uses the algorithm when creating encrypted PA 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 JRE to support unlimited strength security policy. In a clustered environment, be sure to add the security policy changes to the engines as well as the administrative console for the cluster.
Cookie Name	The name of the browser cookie to contain the PA Token (the default is PA). Enter a name for the browser cookie.
Update Token Window (seconds)	The number of seconds before the idle timeout is updated in the PA token. When this time window expires, PingAccess will reissue a new PA cookie.

Configure Server-Side Session Management

There are two ways Server-Side Session Management can be implemented:

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

The first of these scenarios provides increases both scalability and security, ensuring the PingFederate session is terminated and that subsequent session validation requests are rejected. This scenario implies a user logout 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 logs out 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 logged out from all of those protected services.

PingAccess needs to be configured only for the first of these two scenarios. These options are not mutually exclusive, and can be combined to provide comprehensive session management at the server.

Configure PingFederate

To configure PingFederate to be able to revoke PingAccess session cookies:

1. Log in to the PingFederate Administrative Console
2. Navigate to Server Settings -> Roles & Protocols
3. Ensure that "Enable OAuth 2.0 Authorization Server (AS) role" and "OpenID Connect" are enabled. Create or modify an existing client.
4. From the main administration page, go to OAuth Settings -> Authorization Server Settings
5. Return to the main administration page, then go to OAuth Settings -> Client Management
6. Create or modify an existing client.
7. Ensure that "Client Secret" is selected and enter a client secret to be used by PingAccess for authentication.
8. In the OpenID Connect section of the client's configuration page, enable "Grant Access to Session Revocation API."

   ⚠ This setting is the main setting that enables the server-side session management feature in PingFederate.
9. Click `Save` to save your changes.

To configure PingFederate for user-initiated single logout:

1. Select "Track User Sessions for Logout" under "OpenID Connect Settings"
2. In the OpenID Connect Policy Management page, select the policy, then enable the "Include Session Identifier in ID Token" option. (For more information about configuring an OpenID Connect Policy, see *Configuring OpenID Connect Policies* in the *PingFederate Administrator's Manual*.)
3. In the OpenID Connect section of the client's configuration page, enable "PingAccess Logout Capable." Note that this setting is only available if you enabled the "Track user Sessions for Logout."

PingFederate is now configured to provide PingAccess with access to the PingFederate-managed session.

Configure PingAccess

To configure PingAccess:

1. Log in to the PingAccess administrative console
2. Click Settings
3. Click Web Session
4. Either create a new web session or edit an existing web session
5. In the Client ID field, enter the client name from step 6 in the previous section
6. Enter the client secret defined in step 7 in the previous section
7. Select "Validate Session" to enable the server-side session management feature
8. Click Save

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, a user attempts to access a PingAccess Web Application configured with an authentication requirement list containing the values (password, cert). PingAccess redirects the user to PingFederate requesting either password or certificate user authentication. PingFederate authenticates the user based on the password and issues an OIDC ID Token to PingAccess (containing the authentication method that was used). PingAccess ensures that the authentication method matched the requirements and redirects the user to the originally requested Application with the PA cookie set. 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 (cert), they are redirected to PingFederate to authenticate with a certificate.

If you configure Applications with authentication requirement lists that have no overlap. For example, one list has (password), another list (cert), a user navigating between Applications may be required to authenticate each time they visit an Application. When configuring authentication requirement lists to protect higher value Applications with step-up authentication, consider including stronger forms of authentication when configuring lower value Applications.

To configure an Authentication Requirement List

1. Enter a unique name for the Authentication Requirements list. Up to 64 characters, including special characters and spaces, are allowed.
2. Enter an authentication method. For example, cert or password.

   The values you enter here must match the result values defined for the Requested AuthN Context Selector configured within PingFederate (see *Configuring the Requested AuthN Context Selector*).

3. Click to add the requirement.
4. Continue adding requirements.
5. Click **SAVE** when you finish.

To edit an Authentication Requirements List

1. Click for the list you want to edit and click Edit. The Edit Authentication Requirement page appears.
2. Make your changes.
   - Click to edit an authentication requirement.
   - Order the requirements by dragging and dropping them.
   - Click to save edits.
   - Click to exit Edit mode without saving changes.
   - Click to delete an authentication requirement.
3. Click **SAVE** when you finish.

To delete an Authentication Requirements List

1. Click for the list you want to delete and click Delete.
2. Click **DELETE** in the confirmation window.

Identity Mappings

Identity mappings make user attributes available as HTTP request headers to back-end sites that use them for authentication. A single Identity Mapping can expose a number of attribute values. Identity Mappings are assigned to applications.

Identity Mappings replace the Web Session Header Site Authenticator available before PingAccess 3.0.

Field	Description
Subject	Selects which attribute is used as the subject.
Attribute Name	Defines the attribute name you want to retrieve. For example, `sub`. Attributes are obtained from either the PA Token or OAuth Access Token depending on the type of application this mapping is assigned to. The contract thus is either the *OpenID Connect Policy* for PA Token or the *OAuth Access Token Contract* for access token.
Header Name	Defines the HTTP request header that contains the attribute value retrieved from the PA Token. 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`.

PingFederate

Use this page to configure the connection to the PingFederate Runtime and Administration, and to identify the Resource Server client for validating OAuth access tokens.

The PingFederate Runtime configuration includes advanced options used to configure an optional backchannel communication. Without this configuration, all communication between PingAccess and PingFederate takes place through the host specified in the PingFederate Runtime configuration section.

With a high availability configuration where multiple PingFederate Runtime Engines are in use, it may be desirable to configure PingAccess to use those Runtime Engines for behind-the-scenes communication rather than the front channel interface used for user authentication. When this configuration is used, one or more back channel servers can be configured; a basic availability profile configuration is used, which can be overridden by setting parameters in the PingAccess `run.properties` file. The values are defined in the *Availability Profile Defaults* settings.

PingFederate Runtime

For information on setting PingFederate up as an OAuth Authorization Server, see *Enabling the OAuth AS* and *Authorization Server Settings* in the PingFederate documentation.

Prerequisites

Before configuring a secure connection to the PingFederate Runtime, it is necessary to 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. (Optional) *Create a Trusted Certificate Group* if one does not already exist.
4. *Add the Certificate to a Trusted Certificate Group*.

To configure the connection to the PingFederate Runtime:

1. Enter the **Host** name or IP address for PingFederate Runtime.
2. Enter the **Port** number for PingFederate Runtime.
3. Enter the **Base Path**, if needed, for PingFederate Runtime. This field is optional. It must start with a slash - for example: `/federation`.
4. Select the **Audit** checkbox 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 (see *Security Audit Log*).
5. Select the **Secure** checkbox if PingFederate is expecting HTTPS connections.
6. From the **Trusted Certificate Group** list, select the *certificate group* the PingFederate certificate is in. This field is available only if you select the **Secure** checkbox.
7. (Conditional) If hostname verification should be disabled for the PingFederate Runtime, open the **Advanced** section and enable the **Skip Hostname Verification** option.
8. (Conditional) If hostname verification is required for the PingFederate Runtime, open the **Advanced** section and enter the hostname PingAccess should expect in the **Expected Certificate Hostname** field.

In addition to the above procedure, if your setup involves OpenID Connect flows, SSO, or other configuration options that require PingAccess and PingFederate to communicate without user involvement, PingAccess can be configured to use a separate back channel communication for those interactions.

To configure the PingFederate Back Channel servers:

1. Open the **Advanced** section of the **PingFederate Runtime** section of the page.
2. Enter one or more *`hostname:port`* pairs in the **Back Channel Servers** list.
3. (Conditional) 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.
4. (Conditional) If the back channel uses an alternate base path, enter the path in the **Back Chanel Base Path** field. This option becomes available when at least one Back Channel Server is defined.
5. (Conditional) If hostname verification for secure connections is not required for either the Runtime or the Back Channel Servers, enable the **Skip Hostname Verification** option.
6. (Conditional) If hostname verification is required, enter the hostname PingAccess should expect in the **Expected Certificate Hostname** field.
7. Click **SAVE**.

Once you save this configuration and configure the PingAccess OAuth Resource Server (see the **OAuth Resource Server** section, below), a PingFederate Access Validator is available for selection when you define OAuth-type rules in *Policy Manager*.

PingFederate Administration

For information on the PingFederate Administration API see *PingFederate Administrative API* in the PingFederate documentation.

To configure the connection to the PingFederate Administrative API:

1. Enter the **Host** name or IP address for PingFederate Administration API.
2. Enter the **Port** number for PingFederate Administration API.
3. Enter the **Base Path**, if needed, for PingFederate Administration API. This field is optional. It must start with a slash (/). For example, `/path`.
4. Enter the **Administrator Username**. This username only requires Auditor (read only) permission.

5. Enter the **Administrator Password** for the administrator username.
6. Select the **Audit** checkbox 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 (see *Security Audit Log*).
7. Select the **Secure** checkbox if the Site is expecting HTTPS connections.
8. From the **Trusted Certificate Group** list, select the *group of certificates* to use when authenticating to PingFederate. PingAccess requires that the certificate in use by PingFederate anchor to a certificate in the associated Trusted Certificate Group. This field is available only if you select the **Secure** checkbox.

OAuth Resource Server

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. In order to validate the token, a valid OAuth client must exist within the PingFederate OAuth Authorization Server. To enter PingAccess OAuth Client settings configured in PingFederate:

This configuration is optional and needed only if you plan to validate PingFederate OAuth access tokens.

You must configure a connection to the PingFederate OAuth Authorization Server instance you plan to use (see the PingFederate section, above).

1. Enter the OAuth **Client ID** assigned when you created the PingAccess OAuth client within PingFederate (for more information, see *Configuring a Client* in the PingFederate documentation). PingAccess provides this information in order to identify itself. This information is included with every request PingAccess makes.

   When you configure an OAuth client within PingFederate, be sure to select *Access Token Validation* as the allowed grant type.
2. Enter the **Client Secret** assigned when you created the PingAccess OAuth client within PingFederate.
3. Select the **Cache Tokens** checkbox to retain token details for subsequent requests. This option reduces the communication between PingAccess and PingFederate.
4. In the **Subject Attribute Name** box, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes. For example, `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.

Clustering

Use this page to define the administrative console and to set up engine and grant them access to the administrative console.

Engines

Define the runtime engines you want to have access to and communicate with the administrative console.

Set Up an Engine

For each engine:

1. Click **NEW ENGINE** on the right to configure a new engine.
2. Enter a **Name** for the engine. Special characters and spaces are allowed.
3. Enter a **Description** of the engine.

4. Click **SAVE & DOWNLOAD** to generate and download a public and private key pair into the `<enginename>_data.zip` file for the engine. This file is prepended with the name you give the engine. Depending on your browser configuration, you may be prompted to save the file.
5. Copy the zip file to the *<PA_HOME>* directory of the corresponding engine in the cluster and unzip it. The engine uses these files to authenticate and communicate with the administrative console.

> Generate a new key for an engine at any time by clicking **SAVE & DOWNLOAD** and unzipping the `<enginename>_data.zip` archive on the engine to replace the files with a new set of configuration files. When that engine starts up and begins using the new files, PingAccess deletes the old key.

6. (Conditional) On Linux systems running the PingAccess engine, change the permissions on the extracted `pa.jwk` to mode 400 by executing the command `chmod 400 conf/pa.jwk` after extracting the zip file.
7. Start each engine.

> For information on configuring engine to share information with each other in a cluster, see *Configure PingAccess Servers into a Cluster*.

Edit an Engine

1. Access the Clustering page by selecting *Settings* from the menu bar of the PingAccess administrative console.
2. Click ☰ for the engine you want to edit and click Edit. The Edit Engine page appears.
3. Make your edits.
4. Click **SAVE**.

Remove an Engine's Access to the Administrative Console

1. Access the Clustering page by selecting *Settings* from the menu bar of the PingAccess administrative console.
2. Click ☰ for the engine you want to edit and click Edit. The Edit Engine page appears.
3. Click ✖ in the Public Keys box to revoke engine access to the administrative console.

> Use the **SAVE & DOWNLOAD** button to create a new key for the engine. See the steps for setting up a PingAccess engine above.

4. Click **SAVE**.

Remove an Engine

1. Access the Clustering page by selecting *Settings* from the menu bar of the PingAccess administrative console.
2. Click ☰ for the engine you want to delete and click Delete to permanently remove all references to the engine from the cluster.
3. Click **DELETE** in the confirmation window.

Primary Administrative Node

Define the PingAccess server you want to use as the administrative node.

1. Enter the host and port for the administrative console. The default is `localhost:9000`.
2. Click **SAVE**.

Replica Administrative Node

Define the PingAccess server you want to use as the replica administrative node.

1. Enter the host and port for the replica administrative node.
2. Click **SAVE & DOWNLOAD**.
3. Copy the downloaded `replica1_data.zip` to the replica administrative node's *<PA_HOME>* directory and unzip it.
4. (Conditional) On Linux systems running the PingAccess replica administrative node, change the permissions on the extracted `pa.jwk` to mode 400 by executing the command `chmod 400 conf/pa.jwk` after extracting the zip file.

When using a replica administrative node, it is necessary to define a key pair to use for the admin HTTPS Listener that includes both the primary administrative node and the replica administrative node. This can be accomplished either by using a wildcard certificate or by defining subject alternative names in the key pair that include the replica administrative node's DNS name. If a replica administrative node is used 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.

Key Pairs

PingAccess provides built-in 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.

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 initially 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 in order for authentication to succeed..

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 (see the *Oracle Java documentation* for more information).

Use this page to manage Key Pairs and to import or generate additional Key Pairs to secure access to the PingAccess administrative console and for incoming HTTPS requests at runtime as well as for use with the Mutual TLS Site Authenticator.

Import or Generate a Key Pair

To Import an Existing Key Pair

Use this function to import a Key Pair from a *PKCS#12* file. Click **IMPORT** and enter the requested information on the form.

1. In the **Alias** box, 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*.
2. Enter the **Password** used to protect the PKCS#12 file. PingAccess uses the password to read the file.
3. Click **CHOOSE FILE** to locate the PKCS#12 file.
4. Highlight the file and click **Open**.
5. Click **SAVE** to import the file.

To Generate a New Key Pair

Use this function to generate a Key Pair and the self-signed certificate. Click **+NEW KEY PAIR**, then enter the fields required for the Key Pair. This information is used as metadata for the self-signed certificate that is generated as part of the Key Pair. Click **SAVE** when you finish.

Field	Description
Alias	A user-defined name that identifies the Key Pair. Special characters and spaces are allowed. This name identifies the Key Pair on the Key Pairs page and when assigning the Key Pair to various configurations such as when creating a *Mutual TLS Site Authenticator*.
Common Name	The common name (CN) identifying the certificate. This is typically the host name for PingAccess.
Subject Alternative Names	The optional Subject Alternative Names (SANs) for the certificate. These values are used to allow the certificate to be used for multiple DNS names or IP addresses. Select a General Name (DNS Name or IP Address) to define the type of SAN, and enter a value based on the selected General Name.
Organization	The organization (O) or company name creating the certificate.
Organization Unit	Optional. The specific unit within the organization (OU).
City	Optional. The city or other primary location (L) where the company operates.
State	Optional. The state (ST) or other political unit encompassing the location.
Country	The country (C) where the company is based. Enter the country using two capital letters. For example, `US`.
Valid Days	The number of days the certificate is valid.
Key Algorithm	The algorithm used to generate a key. PingAccess uses one of two algorithms: `EC` or `RSA`.
Key Size (bits)	The number of bits used in the key. The values available here depend on the Key Algorithm selected:

Key Algorithm	Values
RSA	1024 2048 (Default) 4096
EC	256 (Default) 384 512

Field	Description
Signature Algorithm	The signature algorithm used for the key. The values available here depend on the Key Algorithm selected:

Key Algorithm	Values
RSA	SHA1withRSA SHA256withRSA (Default) SHA384withRSA

Field	Description	
	Key Algorithm	**Values**
		SHA512withRSA
	EC	SHA256withECDSA (Default) SHA384withECDSA SHA512withECDSA

Manage Existing Key Pairs

To manage existing Key Pairs, click on a Key Pair card to view the following Key Pair management options:

Download Cert

Download a certificate when you need to configure a peer to trust a certificate used by PingAccess. For example, download the certificate for the Key Pair used by a *Mutual TLS Site Authenticator* and configure the target Site to trust the certificate.

1. Click Download Cert. PingAccess automatically downloads the certificate from the Key Pair.
2. Save the file on your system.

Generate CSR

Generate a Certificate Signing Request (CSR) to establish more security and trust than using a self-signed certificate.

1. Click Generate CSR. PingAccess automatically generates a CSR file.

   Save the file on your system. Provide this file to a Certificate Authority (CA). The CA signs the file and provides a CSR Response (see below) 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.

CSR Response

Import a CSR Response to replace the self-signed certificate in a Key Pair. Click **CSR Response** and fill out the form.

1. Select the **Trusted Certificate Group** to use for validating that the certificate in the CSR Response is correctly formed.
2. Choose the CSR Response file.
3. **UPLOAD** the file.

> Before you import the CSR Response, import the signing CA certificate into PingAccess and add it to a *Trusted Certificate Group*.

Delete

1. To delete the Key Pair, click **Delete**.
2. Click **DELETE** in the confirmation window. PingAccess removes the Key Pair from the system.

> If a Key Pair is currently in use, you cannot delete it.

Certificates

Administrators 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 may be issued by a 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.

Certificates

This section provides steps for importing and deleting certificates, and adding to and removing from Trusted Certificate Groups.

Import a Certificate

1. Click . The New Certificate page appears.
2. Click **CHOOSE FILE** to locate the certificate.
3. Highlight the file and click **Open**.
4. Click **SAVE** to import the certificate. A new certificate row appears on the Certificates page.

Delete a Certificate

1. Click on the certificate you want to delete and click Delete.
2. Click **DELETE** in the confirmation window.

> If the certificate is associated with a Trusted Certificate Group, you cannot delete it.

Add a Certificate to a Trusted Certificate Group

1. Hover the cursor over the certificate row you want to move. The move cursor appears.
2. Drag and drop the certificate onto the Trusted Certificate Group.

Remove a Certificate from a Trusted Certificate Group

1. Expand the Trusted Certificate Group containing the certificate you want to remove.
2. Click on the certificate you want to remove.

Trusted Certificate Groups

This section provides steps for creating, editing, and deleting Trusted Certificate Groups.

Create a Trusted Certificate Group

1. Click to create a new Trusted Certificate Group.
2. Drag-and-drop a certificate onto the box that appears. A new group appears at the bottom of the Trusted Certificate Groups list.
3. Enter a name for the group in the box that appears.
4. Drag-and-drop more certificates onto the Certificates box for the group.
5. Select the **Use Java Trust Store** checkbox to set the new group to include the Java Trust Store group. For example, if you create your own intermediate CA certificate that is signed by a well-known CA in the Java Trust Store.
6. Select the **Skip certificate date checks** checkbox to allow PingAccess to ignore date-related errors for certificates that are not yet valid or have expired.
7. Click to save the group.

Edit a Trusted Certificate Group

1. Click for the group you want to edit and click Edit.
   - Edit the group name and optionally set it to include the Java Trust Store group. For example, if you create your own intermediate CA certificate that is signed by a well-known CA in the Java Trust Store.
   - Expand the group and drag-and-drop certificates onto the Certificates box.
   - Click for a certificate you want to remove.
   - Click to exit Edit mode without saving changes.
2. Click to save your changes.

Delete a Trusted Certificate Group

1. Click on the group you want to remove and click Delete.
2. Click **DELETE** in the confirmation window.

Listeners

The Listeners configuration page is used to assign key pairs to the administrative, agent, and engine listeners, as well as to define additional listener ports for the PingAccess engine.

HTTPS Listeners

PingAccess listens for HTTP requests on the ADMIN, ENGINE, and AGENT ports. When HTTPS is enabled for these listeners in `run.properties`, a key pair must be assigned to the listener. See *Key Pairs* on page 64 for information on setting up a key pair. By default, the listeners are configured for HTTPS and use pre-generated key pairs associated with `localhost`.

The listeners are defined as follows:

HTTPS Listener	**Setting in `run.properties` to Enable HTTPS**	**Purpose**
ADMIN	`admin.secure`	Listens for requests for the administrative user interface and the PingAccess REST APIs.
AGENT	`agent.secure`	Listens for requests from *PingAccess Agents* running on Apache or IIS web servers.
ENGINE	`engine.secure`	Listens for HTTP or HTTPS requests that are proxied to target web servers associated with *Sites* on page 24.

Changes to the ENGINE Listener and Virtual Host Key Pairs assignments become effective immediately. Changes to the ADMIN and AGENT Listener Key Pairs require a restart of each PingAccess instance (engine nodes as well as the replica administrative node, if configured) in a clustered deployment.

To change the key pair associated with a listener:

1. Click the context menu icon to the right of the listener , then select **Edit**.
2. Select the desired Key Pair from the list.
3. Click **SAVE**.

Engine Key Pairs

If PingAccess is running under Java 8, the Engine Key Pairs section of the page can be used to assign a key pair to a specific virtual host. Virtual hosts cannot be wildcard hosts, but must include a specific hostname and port. Assigning Key Pairs to Virtual Hosts is useful in situations where the main ENGINE listener Key Pair would not be valid for client TLS handshakes for requests bound for all virtual hosts protected by PingAccess. For example, if the main ENGINE Key Pair is not a wildcard certificate, or does not contain every virtual host name in the *Subject Alternative Name* (SAN) extension.

A Virtual Host may have only one Key Pair assigned to it.

To assign a key pair to a specific virtual host:

1. Expand the **Engine Key Pairs** section of the page.
2. Click the edit icon in the row the desired key pair appears in.
3. In the **Virtual Hosts** field, select the host or hosts the key pair should be used for.
4. Click the save icon.

Engine Listeners

The Engine Listeners section lists the different ports that the PingAccess Engine listens for incoming client requests. By default, an engine listener is bound to all addresses (specified as an address of "0.0.0.0") on port 3000. The address binding can be changed in the engine's **run.properties** configuration file by changing the `engine.http.bindAddress` property.

The listener port is different from the Virtual Host configuration used by Application objects in PingAccess. For example, a listener listening on port 3000 can handle requests on port 443 if the client sends a request to a reverse proxy or other port forwarding device that redirects the traffic to port 3000 on the Engine. The Host header is used to determine which Virtual Host heard the request (and thus, which Application the client requested), regardless of the Engine listener port that receives the request. See *HTTP Requests* on page 75 for more information about this header.

To define a new Engine Listener:

1. Click **NEW ENGINE LISTENER**.
2. Enter a descriptive name for the listener.
3. Enter the port the listener will open.

   Remember to open the port in the system firewall, or the listener will not be able to process any incoming requests.
4. Click **SAVE**.

Admin Authentication

The default PingAccess administrator authentication method used to protect the administrative console is basic authentication (username and password). Change the default method to any PingAccess supported authentication method using the Settings | Admin Authentication page.

We recommend changing the default administrator authentication method to *Single Sign-On (SSO) Authentication*, leveraging the OpenID Connect Provider (OP) features of PingFederate to manage multiple administrators.

Basic Authentication

The authentication default for the PingAccess administrative console is HTTP Basic Authentication. Basic Authentication uses the HTTP Authorization header to transmit the username and password credentials. The PingAccess server response contains a `PA_UI` cookie, which is a signed *JSON Web Token*. Subsequent HTTP requests send this cookie for authentication rather than the less secure HTTP Authorization header. Basic Authentication supports one user – Administrator. To change the Administrator password, click and click Edit to access the **Basic Authentication** page. You will need the existing Administrator password, and the new password must meet the configured password complexity rules discussed in *Configuration Properties*.

Single Sign-On (SSO) Authentication

Administrators can authenticate to the PingAccess administrative console using SSO. PingAccess SSO leverages the OpenID Connect Provider (OP) features of PingFederate, allowing you to use an existing identity store such as a corporate LDAP directory to manage one or more administrators. PingAccess extracts the claims in the ID Token created by PingFederate to generate its own ID Token, caching the claims that grant access to the PingAccess Admin interface. This allows PingAccess to validate the session without further communication with PingFederate. For instructions on configuring SSO Authentication, see *Single Sign-On (SSO) Authentication*.

API Authentication

Administrators can enable API authentication using OAuth access tokens for use by scripts or other autonomous systems. This allows you to access the PingAccess Administrative API using third-party network operation control software or other administrative consoles. For example, you may have a custom program that pre-provisions Site/Resource/Rule configurations for newly deployed Web applications within your enterprise environment. To authorize the calls your program makes to the Administrative API, you must configure an OAuth client for validating access tokens. Click and click Edit to access the **API Authentication** page to configure the OAuth client for PingAccess.

You must configure a connection to the PingFederate OAuth AS instance you plan to use (see *PingFederate*).

To Configure API Authentication

1. Enter the unique identifier (**Client ID**) assigned when you created the OAuth client for validating OAuth access tokens (for more information, see *Configuring a Client* in the PingFederate documentation).
2. Enter the secret (**Client Secret**) assigned when you created the OAuth client for validating OAuth access tokens (for more information, see *Configuring a Client* in the PingFederate documentation).
3. Enter the **Scope** required to successfully access the API. For example, admin. For more information, see *Authorization Server Settings* for defining scopes.
4. Select **Enabled** to activate API Authentication.

For more information on the PingAccess Administrative API, see *Administrative API Endpoints*

Single Sign-On (SSO) Authentication

This page provides example configuration information for enabling Single Sign-On (SSO) for PingAccess administrators. There are several configuration steps required within the PingFederate Authorization Server (AS) as well as PingAccess that you must complete to enable SSO. Expand a section to view those configurations. The Administrative SSO option can be configured to require a specific authentication mechanism, leveraging the PingFederate Requested AuthN Context Selector using the PingAccess *Authentication Requirements* options.

When you enable SSO Authentication, administrative timeouts are controlled by the following settings in the run.properties file: pa.ui.idleExpirationInMinutes, pa.ui.maxExpirationInMinutes, and pa.ui.expirationWarningInMinutes (see *Configuration Properties*).

To define a fall back administrator authentication method should PingFederate be unreachable, enable the *admin.auth=native* property in the run.properties file. This overrides any configured administrative authentication to *Basic Authentication*.

Prior to configuring Administrator SSO, it is necessary to ensure that the PingFederate server certificate has been imported into PingAccess and assigned to a Trusted Certificate Group, and the Trusted Certificate Group needs to be correctly associated with the *PingFederate Runtime* configuration.

Configuring PingAccess

Use the Single Sign-On (SSO) Authentication page in PingAccess to enter the Client ID for the OAuth Client you created in the PingFederate AS.

Be sure to complete the configuration for connecting to the PingFederate OAuth AS on the *PingFederate* page as well as completing the steps below.

To configure SSO Authentication in PingAccess:

1. Enter the unique identifier (**Client ID**) assigned when you created the OAuth client for use with SSO (for more information, see *Configuring a Client* in the PingFederate documentation).
2. Select a defined **Authentication Requirements** list, if your environment requires it.
3. Select **Enabled** to activate SSO Authentication.
4. Click **SAVE** when you finish.

Configuring PingFederate

To enable administrator SSO to PingAccess, configure the following settings within the PingFederate AS. 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.

The information below 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 (see *Using OAuth Menu Selections* for configuration details of the PingFederate OAuth Settings pages).

You must complete the configuration for connecting to the PingFederate OAuth AS instance you plan to use (see *PingFederate*).

Roles and Protocols

- Enable the OAuth 2.0 AS role and the OpenID Connect protocol.
- Enable the 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

We recommend creating an OpenID Connect 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

We recommend creating a Client to use specifically for PingAccess administrative console authentication.

- Select **None** for Client Authentication.
- Add the location of the PingAccess host as a Redirect URI. For example: https://localhost:9000/*
- Select **Implicit** as an Allowed Grant Type.
- Select one of the elliptic curve (**ECDSA**) algorithms as the OpenID Connect ID Token Signing Algorithm and select the OpenID Connect 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.

Backup

The Backup option archives the configuration database and configuration files in the conf folder in a zip file and downloads it to the administrator's computer. The backup file contains the following files:

- /conf/run.properties
- /conf/blitz4j.properties
- /data/PingAccess.h2.db

The database backup file content is encrypted with the database encryption password. By default, 25 backups are stored in the PingAccess server in the `<PA_HOME>/data/archive` folder. The backup file names are of the form `pa-data-<timestamp>.zip`. The number of saved backups can be modified by adding or changing the following property to run.properties file:

```
pa.backup.filesToKeep=25
```

This property only controls backups created automatically when the administrator logs into the PingAccess administration console. Downloaded backups are not preserved in the `<PA_HOME>/data/archive` folder, as they are downloaded by the browser. If the value for this parameter is decreased and the current number of backups exceeds the new value, then the number of stored backups is pruned, starting with the oldest.

Backup is also available via the `/backup` endpoint in the PingAccess admin API.

To restore PingAccess configuration from a backup follow these steps on the administration node:

1. Stop PingAccess
2. Unzip the backup file to a temporary folder.
3. Replace the <PA_HOME>/data/PingAccess.h2.db file with the one from the backup.
4. Replace all the <PA_HOME>/conf/* files with the ones from the backup.
5. Start PingAccess.

Restoring the configuration will restore the admin password. If the password was changed after the backup was taken, the previous password will be needed in order to log in to PingAccess.

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.

A connection failure can be determined based on whether a backend target is not responding, or based 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, it may be desirable to consider that server down even though the web service is responsive.

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

Currently, the only availability profile type is **On-Demand**. You may wish to create different profiles for different sites based on differing site needs for retry counts, retry delays, timeouts, or HTTP status codes.

To configure a new availability profile:

1. Go to **Settings | Availability Profiles** and click **NEW AVAILABILITY PROFILE**.
2. Enter a unique descriptive name for the profile.
3. Select the **On-Demand** Type.
4. Enter the number of milliseconds to wait for a connection to be established to a backend target in the **Connect Timeout (ms)** field.
5. Enter the number of times to retry a connection to a backend target before considering the target failed in the **Max Retries** field.
6. Enter the number of milliseconds to wait between retries in the **Retry Delay (ms)** field.
7. Enter the number of seconds to wait before trying a failed target again in the **Failed Retry Timeout (s)** field.
8. Optionally enter a list of HTTP status codes that should be considered as a failure in the **Failure HTTP Status Codes** field. The sequence for this list is not important.

To edit an existing availability profile:

1. Go to **Settings | Availability Profiles** and click the icon in the upper right corner of the desired profile, then click **Edit**.
2. Make any desired changes to the profile.
3. Click **SAVE** to save your changes. Changes are immediately applied to the configuration.

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

To configure a new load balancing strategy:

1. Go to **Settings | Load Balancing Strategies**
2. Click **NEW LOAD BALANCING STRATEGY**
3. Enter a unique descriptive name for the strategy
4. Select **Round Robin** for the **Type**
5. If browser sessions should not be pinned to a persistent backend target, deselect the **Sticky Session Enabled** option. This option is enabled by default.
6. If the **Sticky Session Enabled** option is enabled, enter a cookie name to use in the **Cookie Name** field. This cookie is used by the PingAccess engine to track the persistent backend targets for a session.

   When a web session is defined, 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*.

To edit an existing load balancing strategy:

1. Go to **Settings | Load Balancing Strategies** and click the icon in the upper right corner of the desired profile, then click **Edit**.
2. Make any desired changes to the profile.
3. Click **SAVE** to save your changes. Changes are immediately applied to the configuration.

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. 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. The settings on this page allow PingAccess to be configured to identify the originating client's address using a list of alternative headers. These settings are used by the PingAccess Policy Server when evaluating network range rules as well as the **inIpRange()** Groovy script matcher.

The list of header names for the **IP Source** and **Host Source** sections is an ordered list, with the first header match being used. By default, **X-Forwarded-For** is configured for IP Source requests, and both **X-Forwarded-Host** and **Host** are configured for Host Source requests.

The IP Source address settings only affect PingAccess as a Gateway; Agents will always use the address for the last hop.

In addition, the **Protocol Source** section can be used to define the header used to identify the protocol used for the original request. The default value is **X-Forwarded-Proto**.

To configure an alternative header for an IP Source address:

1. Enter a header name to search for in the **Header Names** list.
2. Select either **First** or **Last** for the **List Value Location** to determine whether, when a list of values is in the header, the first value or the last value in the list should be used as the IP Source value. The default value is **Last**.
3. Enable or Disable the **Fallback to Last Hop IP** checkbox to determine, if none of the listed headers is present in the request, whether the upstream IP address should be used for rule evaluation. If this value is disabled and no headers match, the network range rule will return a **Forbidden** status.
4. Click **SAVE**.

To configure an alternative header for a Host Source identifier:

1. Enter a header name to search for in the **Header Names** list.
2. Select either **First** or **Last** for the **List Value Location** to determine, when a list of values is in the header, if the first value or the last value in the list should be used as the Host Source value. The default value is **Last**.
3. Click **SAVE**.

To configure an alternative header for the Protocol Source identifier:

1. If necessary, expand the **Protocol Source** section of the page.
2. Enter a header name in the **Header Name** field.
3. Click **SAVE**.

Configuration Properties

The default PingAccess administrative-console and some runtime behavior is controlled in part by configuration properties located in the `<pingaccess_install>/conf/run.properties` file. The majority of the runtime configuration data is stored in the data store. For example, *Applications*, *Rules*, *Sites*, etc. Refer to the file itself for default settings not specified here. Change these properties as needed.

⚠	You must restart PingAccess for the changes to `run.properties` to take effect.

✓	When storing passwords in the `run.properties` file, we recommend you obfuscate them using the `obfuscate.bat or obfuscate.sh` utility to mask the password value. This utility is located in the `<pa_install>/bin` folder.

Admin and Engine Cluster Settings

Property	Description
`pa.operational.mode`	Controls the operational mode of the PingAccess server in a cluster. Valid values are: • STANDALONE - Use this value for a standalone (unclustered) PingAccess instance that runs both the administrative console and the engine. This is the default. • CLUSTERED_CONSOLE - Use this value for the server instance you want to use as the administrative console server. ℹ 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. • CLUSTERED_ENGINE - Use this value to indicate a server engine.

Define the following Engine and Admin properties depending on what operational mode an engine is using.

- Define all of the following 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

Property	Description
`admin.port`	Defines the TCP port on which the PingAccess administrative console runs. Default 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.

Property	Description
`admin.secure`	Defines that administrative console requests must use HTTPS. Default is `true`. If set to `false`, you must also specify `admin.secure.cookie=false`.
`admin.ssl.ciphers`	Defines the type of cryptographic ciphers available for use with administrative HTTPS ports.
`admin.auth`	Overrides the *administrator authentication method*. For example, if *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. Commented out by default with value `native`.

Engine properties

Property	Description
`engine.http.enabled`	Defines whether a STANDALONE or CLUSTERED_ENGINE node listens for requests on the ports defined by the Engine Listeners. Default is `true`.
`engine.http.secure`	Defines whether the engine is using HTTPS. Default is `true`.
`engine.ssl.ciphers`	Defines the type of cryptographic ciphers available for use with engine HTTPS ports.

Agent Properties

Property	Description
`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. Default is `true`.
`agent.http.port`	Defines the TCP port on which the engine listens for agent requests. Default is `3030`.
`agent.http.secure`	Defines whether the engine is using HTTPS for agent requests. Default is `true`.
`agent.ssl.ciphers`	Defines the type of cryptographic ciphers available for use with agent HTTPS ports.
`agent.authz.header.required`	Defines whether PingAccess server should authenticate agent requests using agent name and shared secret in the vnd-pi-authz header. Default value is `true`. Setting this to `false` is useful for POCs and/or debugging.
`agent.resource.cache.ttl`	Defines the default value for the number of seconds agents can cache resources for virtual hosts. Default value is `900`. The field value is set in Settings | Virtual Hosts
`agent.cache.invalidated.response.duratio`	Defines the duration in seconds that application configuration changes are sent by PingAccess server to agents using the vnd-pi-cache-invalidated header in agent

Property	Description
	responses for the changed application. Default value is `900`.

Administrative Console Settings

These properties control the behavior of the Administrative Console. Some are commented out by default and need to be uncommented to apply.

Property	Description
`pa.ui.idleExpirationInMinutes`	Defines the length of time in minutes until an inactive administrative console (or the *interactive documentation* console) times out. Default is `30` minutes.
`pa.ui.maxExpirationInMinutes`	Defines the length of time in minutes until the administrative console (active or inactive) times out. Default is `240` minutes. User may also use `-1` to set the max expiration to one year.
`pa.ui.expirationWarningInMinutes`	Defines the length of time in minutes that a warning message displays prior to a time out of the administrative console. Default is `1` minute.
`pa.ui.legacyBrowserMode`	Adjusts Administrative console HTTP header requirements to be interoperable with legacy Web browsers (Internet Explorer 9, etc).
`pa.admin.user.password.regex`	Defines the regex that controls password complexity for the Administration Console. 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. Default value is `Password must be at least 8 characters in length, contain one upper-case letter, one lower-case letter and one digit..`

Configuration Database and Keystore Settings

Define the username and passwords for the PingAccess *configuration database* and the password for the `cacerts` keystore.

Property	Description
`pa.jdbc.username`	Defines the username for accessing the PingAccess configuration database. Default is `sa`.
`pa.jdbc.password`	Defines the password for the database user of the PingAccess configuration database. Default is `2Access`.
`pa.jdbc.filepassword`	Defines the password used to encrypt the PingAccess configuration database. Default is `2Access`.
`pa.keystore.pw`	Defines the password for the `$JAVA_HOME/lib/security/cacerts` keystore.

Cluster Configuration Settings

Use the following properties when *clustered* engines are sharing information:

Property	Description
`pa.cluster.interprocess.communication`	Defines how the JGroups cluster communicates. `none` (the default): Indicates that no communication is configured between servers in the cluster. `udp`: Indicates that the cluster uses Multicast communications to send and receive information to and from multiple servers at once. `tcp`: Indicates that the cluster uses Unicast communications to send and receive information to and from individual servers one at a time.
`pa.cluster.auth.pwd`	Sets the password that each engine in the cluster must use to authenticate when joining the group. This prevents unauthorized engines from joining a cluster. (Values: any string or blank)
`pa.cluster.encrypt`	Indicates whether to encrypt network traffic sent between engines in a cluster. (Values:`true` or `false` [default])
`pa.cluster.bind.address`	Defines the IP address to which you bind the TCP or UDP listener. The default is `127.0.0.1`.
`pa.cluster.bind.port`	The port associated with the bind-address property above. The default is `7600`. Whether this is a TCP or UPD port depends on the value configured for the `pa.cluster.interprocess.communication` property (see above).
`pa.cluster.failure.detection.bind.port`	Indicates the bind port of a server socket that is opened on the given engine and used by other engines as part of one of the cluster's failure-detection mechanisms. This port is bound to the address determined by `pa.cluster.bind.address`. The default is `7700`. Whether this is a TCP or UDP port depends on the value configured for the `pa.cluster.interprocess.communication` property (see above).
`pa.cluster.mcast.group.address`	Defines the IP address shared among engines in the same cluster for UDP multicast communication; required when the interprocess communication mode is set to `udp`. (Range: `224.0.0.0` to `239.255.255.255`; note that some addresses in this range are reserved for other purposes.) This property is not used for TCP. All engines in a cluster must use the same address for this property and the port property below.
`pa.cluster.mcast.group.port`	Defines the UDP port associated with the `pa.cluster.mcast.group.address` property above.
`pa.cluster.tcp.discovery.initial.hosts`	Designates the initial hosts to be contacted for group membership information when discovering and joining the group; required when the interprocess communication mode is set to `tcp`. The value is a comma-separated list of host names (or IP addresses) and ports. For example, `127.0.0.1[7602]`.

Property	Description
`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 is `500`.
`engine.polling.delay`	Defines, in milliseconds, how long after the initial query to the administrative console that the engine begins querying for configuration information. The default is every `2000` milliseconds.

Server-Side Session Management Configuration Settings

Use the following properties to configure session management:

Property	Description
`pa.websession.updateTokenWindowInSeconds`	Defines, in seconds, how long before an active Web Session token is updated. The default is every `60` seconds.
`pa.websession.cachePFSessionStateInSeconds`	Defines, in seconds, how long PingAccess may cache session state before re-validating it again with PingFederate. The default is `60` seconds.
`pa.websession.refreshSessionInterval`	Defines, in seconds, how frequently PingAccess contacts PingFederate to update user data used in making policy decisions.

Changes to `pa.websession.refreshSessionInterval` and `pa.websession.cachePFSessionStateInSeconds` apply to new web sessions as the default value. Changes for existing sessions can be made using the Administrative API.

EHCache Configuration Properties

Use the following properties to manage the EHCache configuration:

Property	Description
`pa.ehcache.PingFederateReferenceTokenCache.maxEntriesLocalHeap`	Defines the maximum number of entries in the local heap for OAuth tokens. The default is `10000`.
`pa.ehcache.ServiceTokenCache.maxEntriesLocalHeap`	Defines the maximum number of entries in the local heap for token mediation. The default 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 is `1800` seconds.
`pa.ehcache.ServiceTokenCache.timeToLiveSeconds`	Defines, in seconds, the maximum time an entry can be in the token mediation cache. The default is `14400` seconds.
`pa.ehcache.PATokenValidationCache.maxEntriesLocalHeap`	Defines the maximum number of entries in the local heap for decryption of signed or encrypted PingAccess tokens. The default 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 is `120` seconds.

Property	Description
`pa.ehcache.PATokenValidationCache.timeToLiveSeconds`	Defines, in seconds, the maximum time an entry can be in the token validation cache. The default is `300` seconds.
`pa.ehcache.PFSessionValidationCache.maxEntriesLocalHeap`	Defines the maximum number of entries in the local heap for the session validation cache. The default 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 is `120` seconds.
`pa.ehcache.PFSessionValidationCache.timeToLiveSeconds`	Defines, in seconds, the maximum time an entry can be in the session validation cache. The default is `300` seconds.

EHCache is used for the cached information shared by nodes in PingAccess *Subclusters*.

Availability Profile Defaults

Use the following properties to manage the default settings used for availability profiles. These values are also used to provide high availability for PingFederate Back Channel Server configurations.

Property	Description
`pa.default.availability.ondemand.maxRetries`	Defines the maximum number of retries before marking the target system down. The default 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 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 is `250`.
`pa.default.availability.ondemand.failedRetryTimeout`	Defines, in seconds, the amount of time to wait before retrying a failed host. The default is `60`.

Configure PingAccess Servers into a Cluster

Follow the steps below to configure and deploy clustered PingAccess servers. Several of these steps involve changing property values in the *`run.properties`* file located in the `<pa_install>/conf` directory.

Clustering Prerequisites

Before configuring a PingAccess cluster, there are several prerequisite steps that must be taken:

1. Install PingAccess on each cluster node (see *Install PingAccess*)
2. Create a key pair for the PingAccess administrative console that uses the DNS name of the administrative node as the common name. If an administrative replica console is created, this key pair needs to either be configured with the administrative replica node defined in the Subject Alternative Names for the key pair, or the key pair needs to be configured as a wildcard certificate. (See *Key Pairs*)

> Using an IP address as the common name or in the subject alternative names is also acceptable, as long as those values are used in the administrative node fields on the **Settings | Clustering** configuration page.

3. Edit `<PA_HOME>/conf/run.properties` on the clustered console and change the `pa.operational.mode` parameter from `STANDALONE` to `CLUSTERED_CONSOLE`.
4. Go to **Settings | Clustering** and change the **Primary Administrative Node** value from `localhost:9000` to `<dns_name>:9000`, where `<dns_name>` is the common name from the key pair defined in step 2.
5. Restart the administrative node

Configuring the Replica Administrative Node

If a Replica Administrative Node is being configured in the environment, that must be done prior to configuring the engines. Perform the following steps:

1. Go to **Settings | Clustering** and configure the **Replica Administrative Node** hostname and port. This name must match either a subject alternative name in the key pair created in the previous section, or be considered a match for the wildcard specified if the key pair uses a wildcard in the common name.
2. Click the download icon next to the **SAVE** button to download the bootstrap file for the replica administrative node.
3. Copy the downloaded file to the replica administrative node's `<PA_HOME>` directory and unzip it
4. (Conditional) If the Replica Administrative Node is running on a Linux host, execute the command `chmod 400 conf/pa.jwk`
5. Edit `<PA_HOME>/conf/run.properties` on the replica administrative node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`
6. Start the replica node
7. You can verify replication has completed by monitoring the `<PA_HOME>/log/pingaccess.log` file and looking for the message "Configuration successfully synchronized with administrative node"

Configuring the Engine Nodes

Once the replica administrative node is configured, the engine nodes can be configured. If a replica administrative node is added to the configuration later, it is necessary to reconfigure the clustered engine nodes so their `bootstrap.properties` files include the necessary configuration information about the replica administrative node.

Perform the following steps to configure each engine node:

1. Go to **Settings | Clustering** and click **NEW ENGINE**
2. Enter a unique engine name and a description for the engine
3. Click the download icon next to the **SAVE** button to download the engine's `<engine_name>_data.zip` file
4. Copy the `<engine_name>_data.zip` file to the engine's `<PA_HOME>` directory and unzip it
5. (Conditional) If the Engine is running on a Linux host, execute the command `chmod 400 conf/pa.jwk`
6. Edit `<PA_HOME>/conf/run.properties` on the engine and change the `pa.operational.mode` value to `CLUSTERED_ENGINE`
7. Start the engine node

> Once the engines are configured, they can be put behind a load balancer for load balancing and fault tolerance.

Failing Over to the Replica Administrative Node

The Replica Administrative Node is intended to be used for disaster recovery purposes. If the clustered console is recoverable, then that recovery should be used rather than failing over to the Replica Administrative Node.

> Only one primary administrative node should be running for the cluster at any given time.

To fail over to the Replica Administrative Node, modify the `<PA_HOME>/conf/run.properties` file on the replica node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE`. This change is detected while the node is running, and does not require a restart of the node.

Reinstating a Replica Administrative Node after Failing Over

Once the console has been failed over to the replica, you need to set up a new replica console again. To do this, perform the following steps:

1. Install the new replica node.
2. Change the `run.properties` value for `pa.operational.mode` to `CLUSTERED_CONSOLE_REPLICA`
3. Go to **Settings | Clustering** and change the **Primary Administrative Node** hostname and port to the failed over node.
4. Remove the **Replica Administrative Node** public key, then change the **Replica Administrative Node** hostname and port to point to the new replica node.

   If your key pair does not include a wildcard, you will want to use the same hostname as the original console in order to avoid having to recreate the console key pair and the bootstrap.properties files for each engine.
5. Click the download icon next to the **SAVE** button to download the bootstrap file for the replica administrative node.
6. Copy the downloaded file to the new replica administrative node's `<PA_HOME>/conf` directory, and rename it to `bootstrap.properties`
7. Edit `<PA_HOME>/conf/run.properties` on the new replica administrative node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`
8. Start the new replica node
9. You can verify replication has completed by monitoring the `<PA_HOME>/log/pingaccess.log` file and looking for the message "Configuration successfully synchronized with administrative node"

If you want to then switch back to the original console, shut down the original replica node, and fail over back to the newly created replica console. Follow the above steps a second time to re-establish the original replica node.

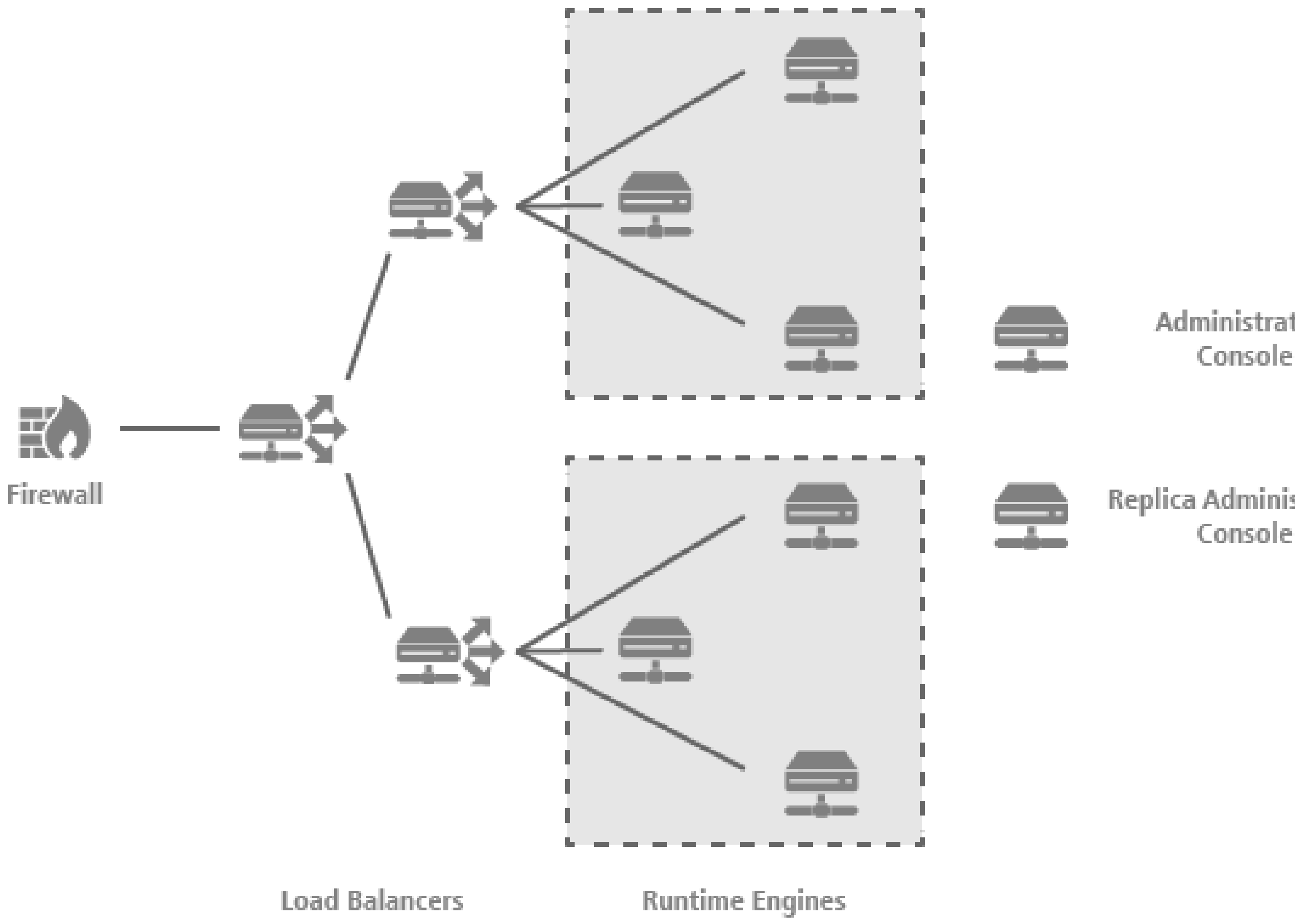
Configuring Subclusters

Subclusters are a method of scaling a very large PingAccess clustered installation in a more linear fashion, by allowing multiple engine nodes in the configuration to share certain information. A load balancer is placed in front of each subcluster in order to distribute connections to the nodes in the subcluster.

The following information is shared between nodes in a subcluster:

- Mappings from encrypted or signed PingAccess cookies to the attributes contained in the cookie
- PingFederate session identifiers and timings for when PingAccess must consult PingFederate to determine if the session is still valid
- Mediated tokens used by the Token Mediator Site Authenticator
- Status and attributes of validated PingFederate issued OAuth access tokens

These caches can be tuned using the EHCache Configuration Properties listed in the *Configuration Properties* documentation.



To configure PingAccess engine node subclusters:

1. Modify `<PA_HOME>/conf/run.properties` and change the `pa.cluster.interprocess.communication` value from `none` to either `tcp` or `udp`.

   Using UDP for the interprocess communication allows a multicast group to be used for this communication, which for a larger subcluster may be more efficient.
2. (Conditional) If TCP is used for interprocess communication, configure the `pa.cluster.tcp.discovery.initial.hosts` value to specify a list of initial hosts to contact for group discovery.
3. (Conditional) If UDP is used for interprocess communication, optionally configure the `pa.cluster.mcast.group.address` and `pa.cluster.mcast.group.port` values for each subcluster.
4. Place a load balancer in front of each subcluster to distribute the load across the subcluster nodes.
5. Restart the engine nodes.

Engine Properties File

An administrator uses PingAccess to generate and download the bootstrap.properties file when adding an engine to a cluster (see *Clustering*). This file is specific to that engine and is stored with the engine in the /conf directory. The engine uses this file to gain access to and communicate with the administrative console for configuration updates.

The following configuration properties are found in the bootstrap.properties file.

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 truststore, in JWK format, that is used for communication with the administrative console.

Customize User-Facing Pages

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_install>/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_install>/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.

For information about Velocity, refer to the *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.

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.

At runtime, the user's browser is directed to the appropriate page, depending on the operation being performed and where the related condition occurs (see the table below). 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
general.error.page.template.html	Indicates that an unknown error has occurred and provides an error message.	Error	Consult your system administrator.
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*.	Error	Verify that your OAuth credential is valid and retry the request.

Template File Name	Purpose	Type	Action
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*.	Error	Retry the request.

The engine.bootstrap.template.properties and site.authenticator.rst.xml files are system templates. We recommend that you do not modify these files.

PingAccess Endpoints

The following endpoints provide a means by which external applications can communicate with the PingAccess server and provide complete administrative capabilities of the product.

- *Heartbeat Endpoint*: A maintenance endpoint is provided for administrators to verify that the server is running.
- *OpenID Connect Endpoints*: Endpoints needed for PingFederate to interface with PingAccess using the OpenID Connect (OIDC) protocol are also included.
- *Administrative API Endpoints*: The Administrative API endpoints are used by the PingAccess administrative console. These are REST APIs that can be called from custom applications or using command line tools such as curl.

Heartbeat Endpoint

You can use an HTTPS call at any time to verify that the PingAccess server is running. This call can be made 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 is configured using the enable.detailed.heartbeat.response parameter in run.properties. If this option is set to false, then an HTTP 200 status and the text OK is returned.

If the enable.detailed.heartbeat.response parameter is set to true (the default setting), then a configurable status with more detail is returned. PingAccess must be restarted if this value is changed.

If an error is returned, then the PingAccess instance associated with the endpoint is down. Load balancers can use this endpoint to determine the status of PingAccess, independent of any other system status checks.

Begin the URL with the server name and the PingAccess runtime port number. For example: https://*hostname*:3000/pa/heartbeat.ping.

The detailed response output format is an Apache Velocity template defined in <PA_HOME>/conf/template/heartbeat.page.json. The following values are available:

Value	Description
$monitor.getTotalJvmMemory('bytes'|'KB'|'MB'|'GB')	Returns the total memory in the 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.

Value	Description
$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.
$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 the note in the *Server Clustering* section. 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 template can be modified in any way to suit your needs.

The default content type is application/json, however this can be overridden by modifying the $monitor.setContentType() line in the template to specify the desired content-type header.

OpenID Connect Endpoints

This page describes the endpoints needed for PingFederate 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

Clears the cookie containing the PA Token. This endpoint enables end users to trigger the removal of their own PA Cookie from the browser they are using. The Logged Out page is a template that can be modified (see *Customize User-Facing Pages*).

This endpoint simply clears the PA Token from the browser cookie. It does not retain any server-side state to denote log off. Additionally, this endpoint clears the cookie only from the requested host/domain and may still exist in requests bound for other hosts/domains.

Note: If logout is being performed across multiple domains, use the PingFederate `/idp/startSLO.ping` endpoint instead. See *IdP Endpoints* in the *PingFederate Administrator's Manual* for more information about this endpoint.

/pa/oidc/cb

The OIDC callback endpoint that receives the ID Token from PingFederate.

/pa/oidc/JWKS

The JSON Web Key endpoint used by the PingFederate JWT Token Processor for signature verification. This endpoint must is used in conjunction with the configuration of a JWT token processor instance in PingFederate. For more information, see *Configuring a JSON Web Token (JWT) Processor Instance* in the PingFederate Administrator's Manual.

/pa/oidc/logout.png

Used by PingFederate to initiate a logout from PingAccess in conjunction with the single logout functionality. This endpoint terminates the PA tokens across domains.

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. This endpoint is only available on the `admin.port` defined in `<PA_HOME>/conf/run.properties`.

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.

To access the interactive documentation in PingAccess:

1. *Start PingAccess*.
2. Launch your browser and go to URL `https://<host>:<admin-port>/pa-admin-api/v1/api-docs/`. For example, `https://localhost:9000/pa-admin-api/v1/api-docs/`.
3. The browser may prompt for credentials. Enter the administrator username and password.

For example, to use the interactive Administrative API documentation to see all defined applications:

1. Click on the `/applications` endpoint to expand it.
2. Click on the `GET` method (`GET /applications`) to expand it.

3. Enter parameters values or leave all blank.
4. Click **Try It Out** button.
5. The Request URL, Response Body, Response Code, and Response Headers appear.

To access the PingAccess Administrative API programmatically:

1. Send HTTP request to URL `https://<host>:<admin-port>/pa-admin-api/v1/<api-endpoint>`.
2. You must provide appropriate administrator credentials in the request.

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/v1/applications
```

- the `-u Administrator:Password1` part sends Basic Authentication header with the username `Administrator` and password `Password1`
- the `-k` part specifies to ignore HTTPS certificate issues
- the `-H "X-Xsrf-Header: PingAccess"` part sends a X-XSRF-Header with value `PingAccess`

Change Configuration Database Passwords

This page provides the steps to change the file and user passwords of the PingAccess configuration database. The server uses the file password to access the encrypted database and the user password to log in to the database after gaining access to the file. Use the dbfilepasswd and dbuserpasswd scripts located at the <pa_install> root to change these passwords to more secure, independent ones.

The default password for both the database file and database user is 2Access.

To change the file password: **Linux**

From a command prompt, execute the <pa_install>/dbfilepasswd.sh shell script.

Windows From the **Start > Run** dialog or a command prompt, run the batch file:

<pa_install>\dbfilepasswd.bat

This script uses the old password and the new password as parameters. For example, dbfilepasswd.sh old_password new_password and outputs the new password in obfuscated format (similar to running obfuscate.sh). Set this new obfuscated password as the value for the pa.jdbc.filepassword property in the run.properties file (see *Configuration Properties*).

To change the database user password:

Linux

From a command prompt, execute the <pa_install>/dbuserpasswd.sh shell script.

Windows

From the **Start > Run** dialog or a command prompt, run the batch file:

<pa_install>\dbuserpasswd.bat

This script uses the database file password, the old password, and the new password as parameters. For example, dbuserpasswd.sh file_password old_password new_password and outputs the new password in obfuscated format (similar to running obfuscate.bat). Set this new obfuscated password as the value for the pa.jdbc.password property in the run.properties file (see *Configuration Properties*).

Manage Log Files

PingAccess logging is handled by the Blitz4j asynchronous logging library, configured using the blitz4j.properties file located in the /conf directory of your PingAccess installation. Blitz4j is an extension of the Log4j framework, so the blitz4j.properties file is similar to a log4j.properties file.

Audit logs are also configurable in the blitz4j.properties file. These logs record a selected subset of transaction log information at runtime plus additional details (see *Security Audit Logging*).

By default, logging information is output to the pingaccess.log file located in the /logs directory of your PingAccess installation. Logging to a file is configured to use the *rolling* file appender, which allows a log file to be 100 MB before the system starts a new file. PingAccess keeps a maximum of 10 log files, each with a maximum size of 100 MB. Once 10 files accumulate, PingAccess deletes the oldest. Configure these options by locating and modifying the following properties in the asynchronous file logging configuration section of the blitz4j.properties file:

- log4j.appender.file.File=./logs/pingaccess.log
- log4j.appender.file.MaxFileSize=100MB
- log4j.appender.file.MaxBackupIndex=10

Log Levels

You can define log levels for specific package or class names in order to get more (or less) logging from a class or group of classes. For all other classes that do not have a specific log level defined, the root log level applies.

To set the root level of logging:

1. Locate this line:

   log4j.rootLogger=DEBUG,file
2. Modify the first value in the comma-separated list to one of the valid log levels: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE. For example, to apply TRACE level logging, change log4j.rootLogger=DEBUG,file to log4j.rootLogger=TRACE,file

To set the log level for a specific class or package:

1. Locate the package or class name in the properties file.
2. Set the first value in the comma-separated list to one of the valid log levels: OFF, FATAL, ERROR, WARN, INFO, DEBUG, TRACE. For example, to apply TRACE level logging for the com.pingidentity package, locate the following line: log4j.logger.com.pingidentity=DEBUG,file and change it to:

   log4j.logger.com.pingidentity=TRACE,file

Appending Log Messages to Syslog and to Console

Additional output destinations (called *appenders*) are available. Configuration for the console and syslog appenders is included in the blitz4j.properties file, but not enabled by default. In the file, enable the console or syslog appenders by uncommenting and modifying the configuration entries for log4j.appender.console and log4j.appender.syslog respectively.

In addition to defining and configuring the appender using the log4j.appender.*AppenderName* properties, you must do the following:

- Enable a new appender
- Add the appender to the root logger to enable logging not specifically controlled by a package/class name logger
- Add the appender to any of the package/class name specific loggers you want appended

To enable a new appender:

Add the appender name to the comma-separated list of asynchronous appenders. For example, to enable the console logger, locate the following line:

log4j.logger.asyncAppenders=DEBUG,file

and change it to:

log4j.logger.asyncAppenders=DEBUG,file,console

> The DEBUG qualifier applied to the asyncAppenders does not apply DEBUG level logging to the appender. This setting exists for compatibility in configuring Blitz4j on top of Log4j. It is recommended that you do not remove this value.

To add the appender to the root logger: Add the appender name to the rootLogger comma-separated list. For example, to add the console appender, locate the following line:

log4j.rootLogger=DEBUG,file

and change it to:

log4j.rootLogger=DEBUG,file,console

To add the appender to a package/classname specific logger:

Add the appender name to a package/classname specific logger. For example, to add the console appender to the com.pingidentity package-specific logger, locate the following line:

log4j.logger.com.pingidentity=DEBUG,file

and change it to:

log4j.logger.com.pingidentity=DEBUG,file,console

Logging Cookies

Cookie logging is an optional feature in the TRACE log level. To enable cookie logging:

1. Stop the PingAccess standalone or engine instance
2. Edit `<PA_HOME>/conf/blitz4j.properties` and uncomment the following lines:

   ```
   log4j.logger.com.pingidentity.pa.core.interceptor.CookieLoggingInterceptor=TRACE,file
   log4j.additivity.com.pingidentity.pa.core.interceptor.CookieLoggingInterceptor=false
   ```
3. Restart the PingAccess instance

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 the /logs directory of your PingAccess installation. Elements of the logs are described in the table below and configurable in the blitz4j.properties file located in <pa_install>/conf.

PingAccess generates these logs that document server events:

- pingaccess_engine_audit.log--Records transactions of configured Resources. Additionally, the log records transaction details when PingAccess sends requests to PingFederate (for example, STS, OAuth2, 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.

Audit Log Configuration

Item	Description
%d	Transaction time.
AUDIT.authMech	Mechanism used for authentication. Engine Auditing - Cookie (WAM session), OAuth, unknown (for example,

Item	Description
	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 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.
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.reqReceivedMillisec	Time in milliseconds (since 1970) that a client request was first received
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.

Writing Logs to Other Formats

PingAccess provides the option of writing the administrative API and engine audit logs to an *Oracle database*.

> To ensure availability of audit log information if database logging fails for any reason, enable both file and database audit logging. An automated failover from database to file logging is not currently available.

You may also configure PingAccess to write the audit logs to a differently formatted log file that can easily be digested by *Splunk*.

- *Writing Logs to Databases*
- *Writing Audit Logs for Splunk*

Writing Logs to Databases

You can enable database logging for the API and engine audit logs in the blitz4j.properties file located in your PingAccess install. Scripts are provided to create the necessary table(s).

> To ensure availability of audit log information if database logging fails for any reason, enable both file and database audit logging. An automated failover from database to file logging is not currently available.

> Ensure that your database-driver JAR file is installed in the <pa_install>/lib directory. You must restart PingAccess after installing the driver.

To configure database logging:

1. In the blitz4j.properties file locate in the /conf directory of your PingAccess install, uncomment one of the preset appender configurations listed below (or one from each list to configure all logs):

   For Administrative API audit logging: OracleDbApiAudit

   For Engine audit logging: OracleDbEngineAudit
2. Replace the placeholder parameter values for the appender(s) with valid values. The parameter values provide access to the database. We recommend that they be tested and validated prior to production deployment.

   > See the notes in the properties file above the appender for more details.

   > You can obfuscate the password used to access the database by running either obfuscate.sh or obfuscate.bat, located at the <pa_install> root. Use the actual password as an argument and copy the entire result into the value for the password parameter in the properties file.

   Add the appender name to the associated comma-separated list of appenders in the **Log Level Configuration** section.

   Oracle database API audit log:

   Locate the log4j.logger.apiaudit line and add OracleDbApiAudit to the list. For example:

   log4j.logger.apiaudit=INFO,apiaudit,OracleDbApiAudit

   Oracle database engine audit log:

   Locate the log4j.logger.engineaudit line and add OracleDbEngineAudit to the list. For example:

   log4j.logger.engineaudit=INFO,engineaudit,OracleDbEngineAudit
3. Create database tables. Scripts to create database tables are provided. The scripts are located in the directory:

   pingaccess/conf/blitz4j/sql-scripts

> The scripts are written to handle the default list of elements for the relevant database log-appender. Any changes to the list requires corresponding changes to the SQL table-creation script (or to the table itself if it is already created). For more information on working with this script, see the Oracle documentation.

Writing Audit Logs for Splunk

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. To configure PingAccess to write audit logs to a format for Splunk:

1. In properties file, uncomment one of the preset log-appender configurations listed below (or one from each list to configure all logs):

   API audit logging for Splunk: SplunkApiAudit

   Engine audit logging for Splunk: SplunkEngineAudit
2. Add the appender name to the associated comma-separated list of appenders in the **Log Level Configuration** section.

   API audit log for Splunk:

   Locate the log4j.logger.apiaudit line and add SplunkApiAudit to the list. For example:

   log4j.logger.apiaudit=INFO,apiaudit,SplunkApiAudit

   Engine audit log for Splunk: Locate the log4j.logger.engineaudit line and add SplunkEngineAudit to the list. For example:

   log4j.logger.engineaudit=INFO,engineaudit,SplunkEngineAudit
3. Save the file and start or restart PingAccess.
4. Download and install the Splunk Universal Forwarder on the machine running PingAccess.
5. Configure the Universal Forwarder to monitor the pingaccess_api_audit_splunk.log or the pingaccess_engine_audit_splunk.log in <pa_install>/logs.

> For detailed installation and configuration instructions, consult the Splunk documentation accompanying the Universal Forwarder.

Chapter

4

PingAccess Deployment Guide

There are many topics to consider when deciding how PingAccess fits into your existing network, from determining the deployment architecture required for your use case and whether high-availability options are required, to port requirements and clustering options. This section provides information to help you make the right decisions for your environment.

- *Use Cases and Deployment Architecture*
- *Port Requirements*
- *Performance Tuning*
- *Server Clustering*

Use Cases and Deployment Architecture

There are many options for deploying PingAccess in your network environment depending on your needs and infrastructure capabilities. For example, 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 may add headers and tokens required by the target resource. Under the PingAccess Policy Server's control, the agent may perform a certain amount of caching of information in order to minimize the overhead of contacting the PingAccess Policy Server, thus maximizing 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 (for example, backdoors) are not available.

The following sections provide specific use cases and deployment architecture requirements to assist with designing and implementing your PingAccess environment.

Deploying for Gateway Web Access Management

Deploying for Agent Web Access Management

Deploying for Gateway API Access Management

Deploying for Auditing and Proxying

Deploying 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:

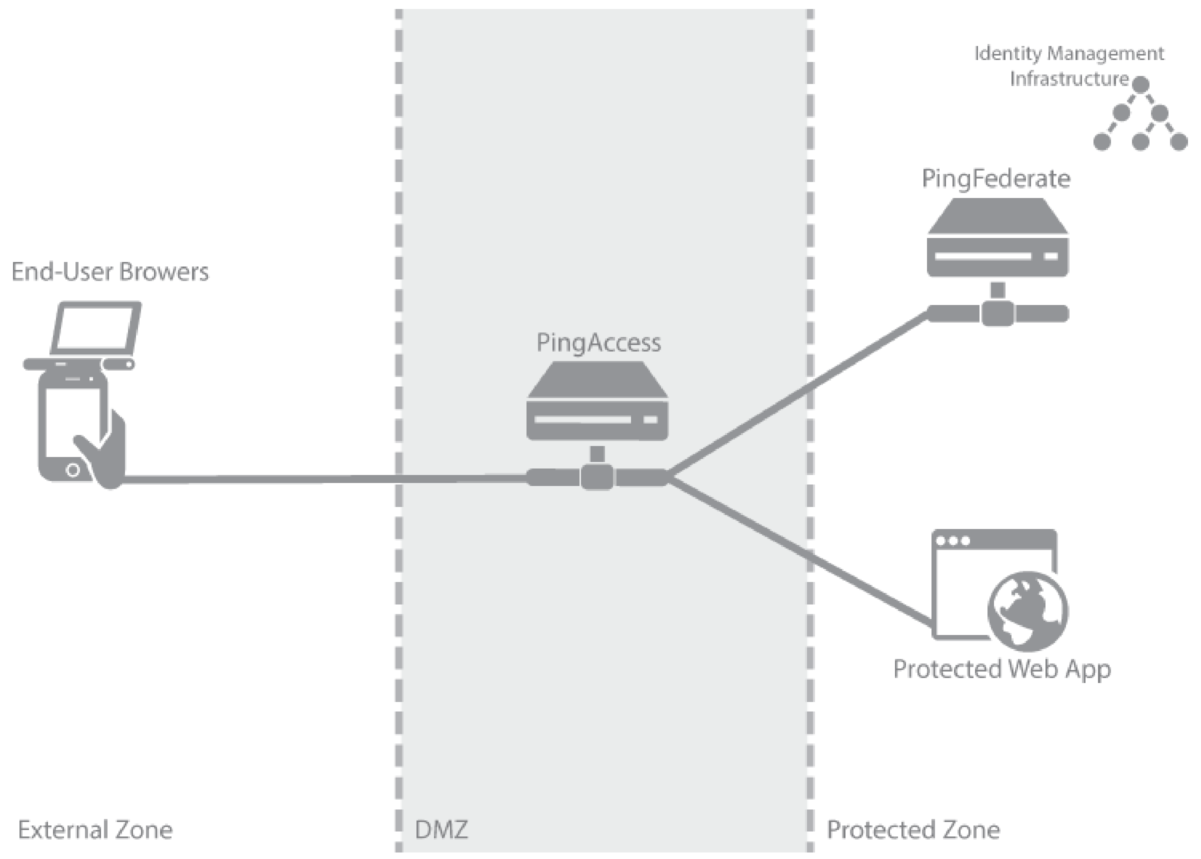
- Receives inbound calls requesting access to Web applications. Web Session protected requests contain a previously-obtained PA token in a cookie derived from the user's profile during an OpenID Connect based login at PingFederate.
- Evaluates application and resource-level policies and validates the tokens in conjunction with an OpenID Connect Policy configured within PingFederate.
- Acquires the appropriate target security token (*Site Authenticators*) from the PingFederate 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*

Web Access Management Gateway Proof Of Concept Deployment Architecture

This environment is used to emulate the 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.



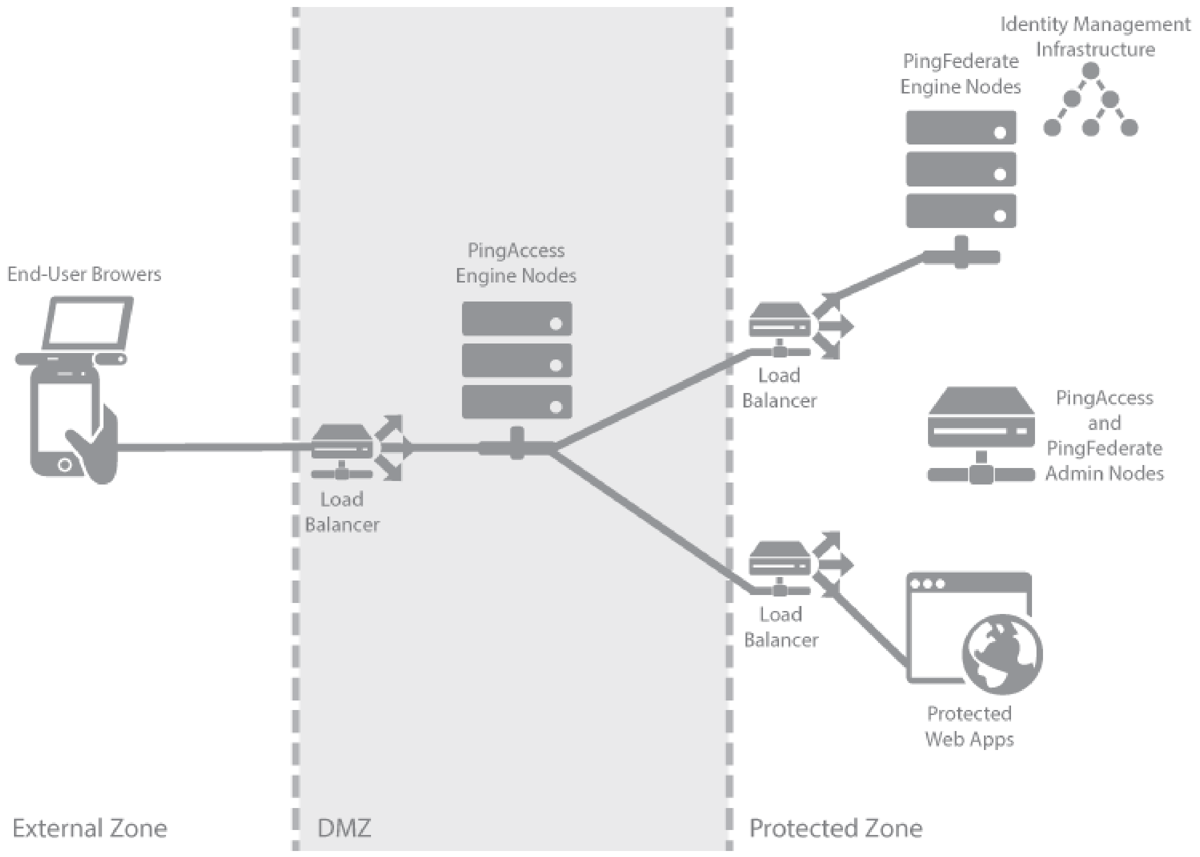
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ Zone	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	Back-end 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 in order to authenticate users (depicted by the icon in the diagram).

Web Access Management Gateway Production Deployment 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.

PingAccess can provide high availability and basic load balancing for the protected web apps in the protected zone. See the *Availability Profiles* and *Load Balancing Strategies* documentation for more information.



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 Zone	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 may be required.
Protected Zone	Back-end 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 may be co-located on a single machine to reduce hardware requirements.

Deploying 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 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.

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:

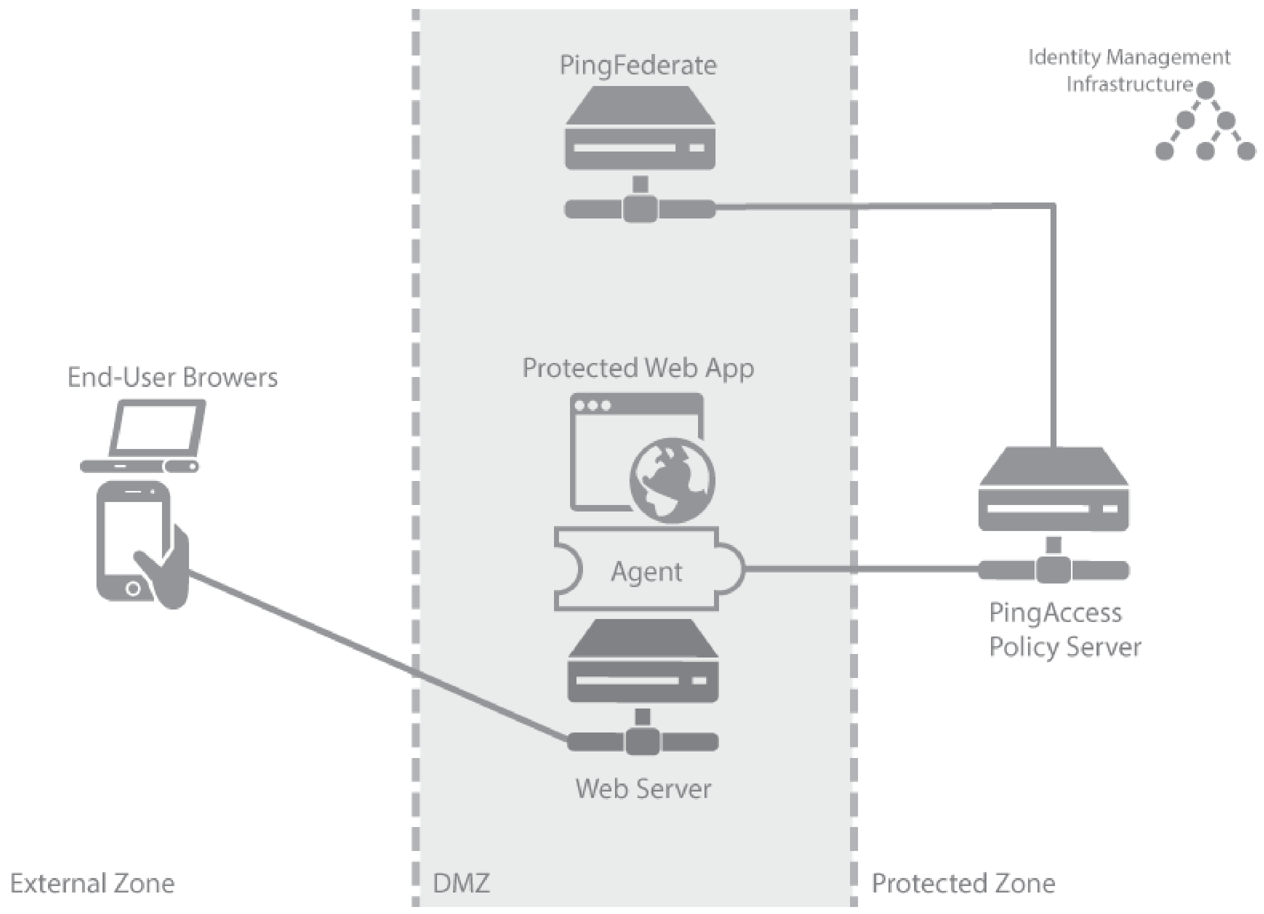
- Evaluates application and resource-level policies and validates the tokens in conjunction with an OpenID Connect Policy configured within PingFederate
- Acquires the appropriate HTTP request header configuration from the associated *Identity Mappings*.
- Sends an agent response with instructions on 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*

Web Access Management Agent Proof Of Concept Deployment Architecture

This environment is used to emulate the 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.



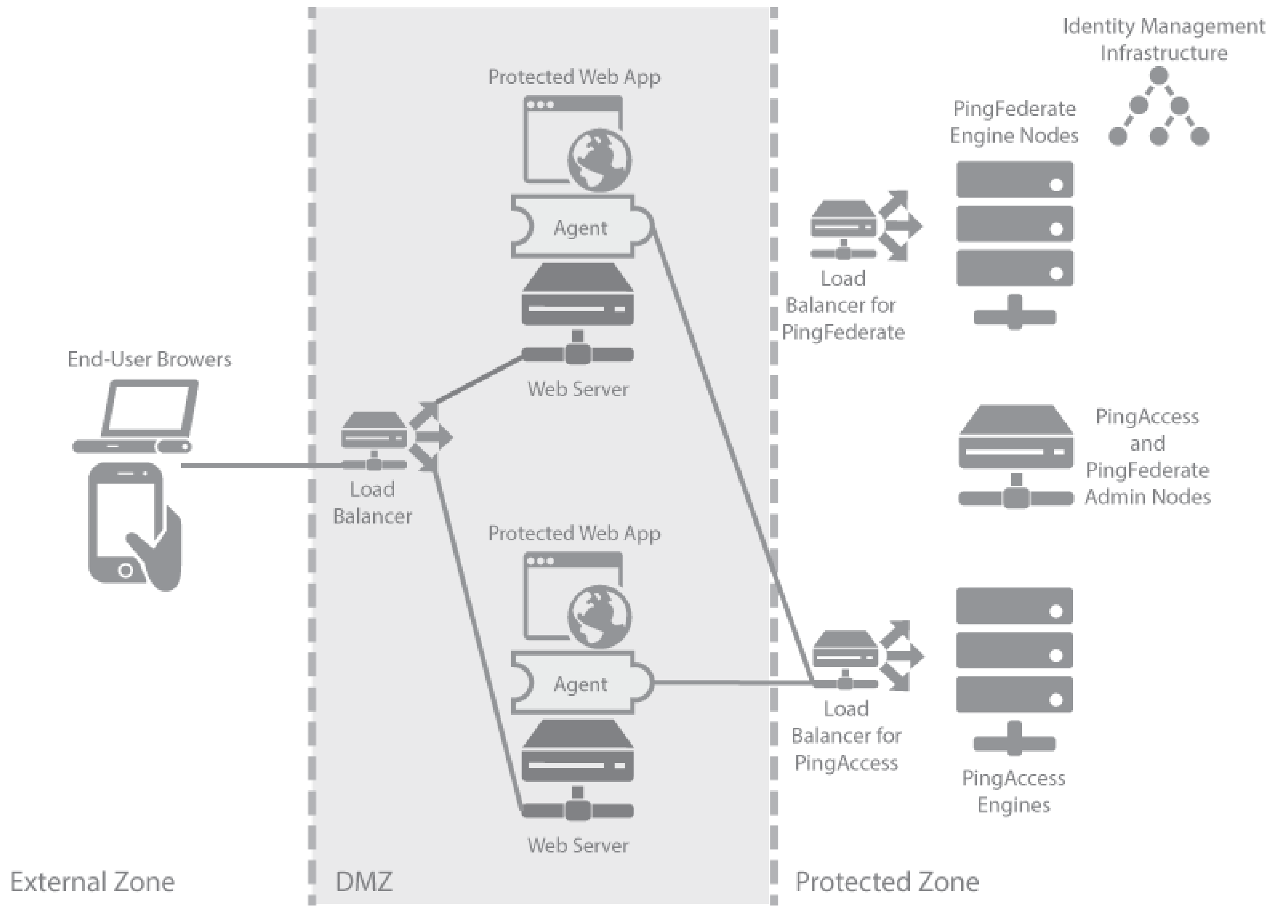
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ Zone	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 in order to authenticate users.
Protected Zone	Back-end 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

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.


Identity Management Infrastructure
Protected Web App
PingFederate Engine Nodes
Agent
Load Balancer for PingFederate
End-User Browers
Web Server
PingAccess and PingFederate Admin Nodes
Load Balancer
Protected Web App
Agent
Load Balancer for PingAccess
PingAccess Engines
Web Server
External Zone
DMZ
Protected Zone

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 Zone	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	Back-end 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.

Deploying 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 ever-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:

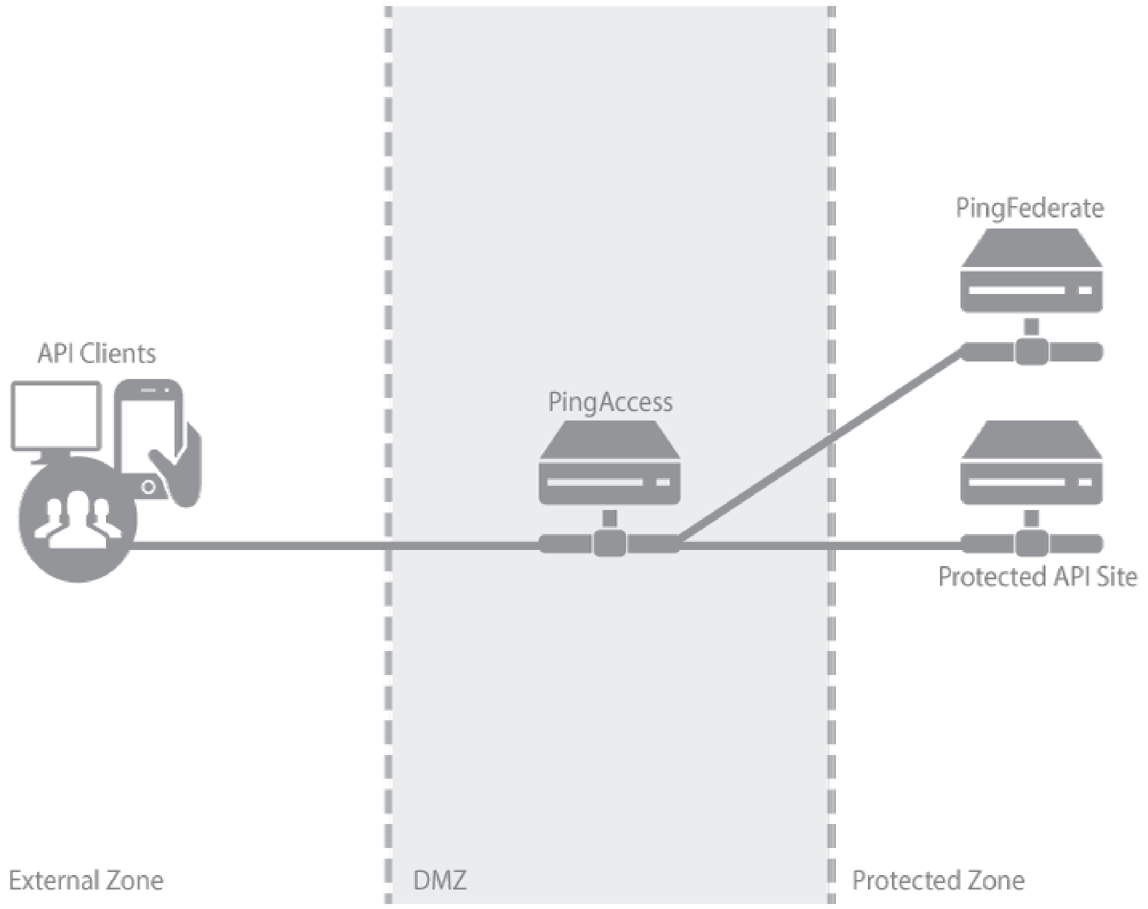
- Receives inbound API calls requesting protected applications. OAuth-protected API calls contain previously-obtained access tokens retrieved from PingFederate acting as an OAuth Authorization Server.
- Evaluates application and resource-level policies and validates access tokens in conjunction with PingFederate.
- Acquires the appropriate target site security token (*Site Authenticators*) from the PingFederate 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*

API Access Management Proof of Concept Deployment Architecture

This environment is used to emulate a production environment for development and testing purposes. In the test environment, PingAccess can be set up with the minimum hardware requirements. Given these conditions, we do not recommend using this proposed architecture in a production deployment as it does not provide high availability.



The following table describes the three zones within this proposed architecture.

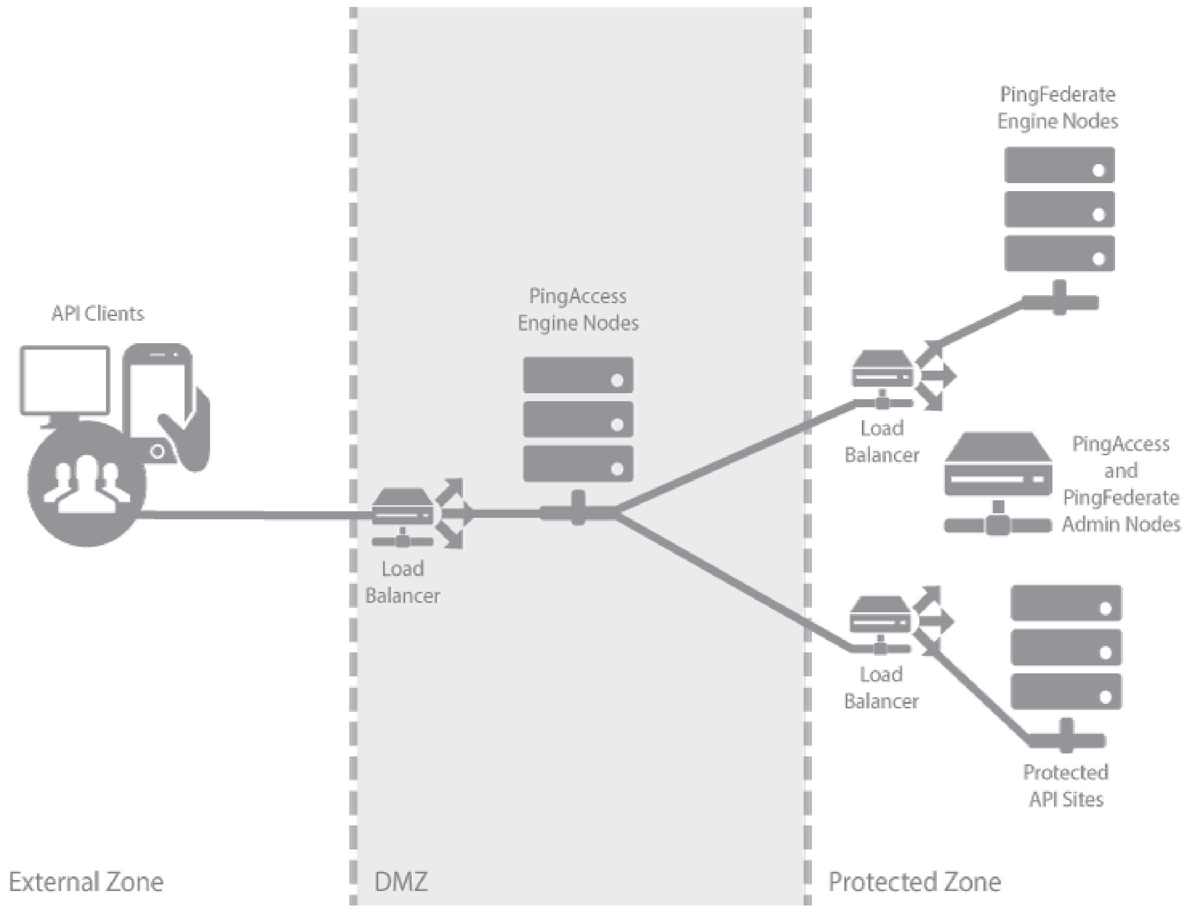
Zone	Description
External Zone	External network where incoming API requests originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to API clients. PingAccess is a standalone instance in this environment, serving as both a runtime and an administrative port.
Protected Zone	Back-end 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

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.

ℹ	PingAccess can provide high availability and basic load balancing for the protected web apps in the protected zone. See the *Load Balancing Strategies* documentation for more information.

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 Zone	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, more nodes may be required.
Protected Zone	Back-end 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 may be co-located on a single machine to reduce hardware requirements

Deploying 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 back-end Sites. With growing numbers of internal and external users, it is important to know which users are accessing applications, from where and when they are accessing them, and ensuring that they are correctly accessing only those applications to which they have permission. A standardized auditing/proxying deployment provides a centrally-controlled model that ensures existing infrastructure and security policies are followed, thereby safeguarding an organization's assets.

Sitting at the perimeter of a protected network between mobile, in-browser, or server-based client applications and back-end Sites, PingAccess performs the following actions:

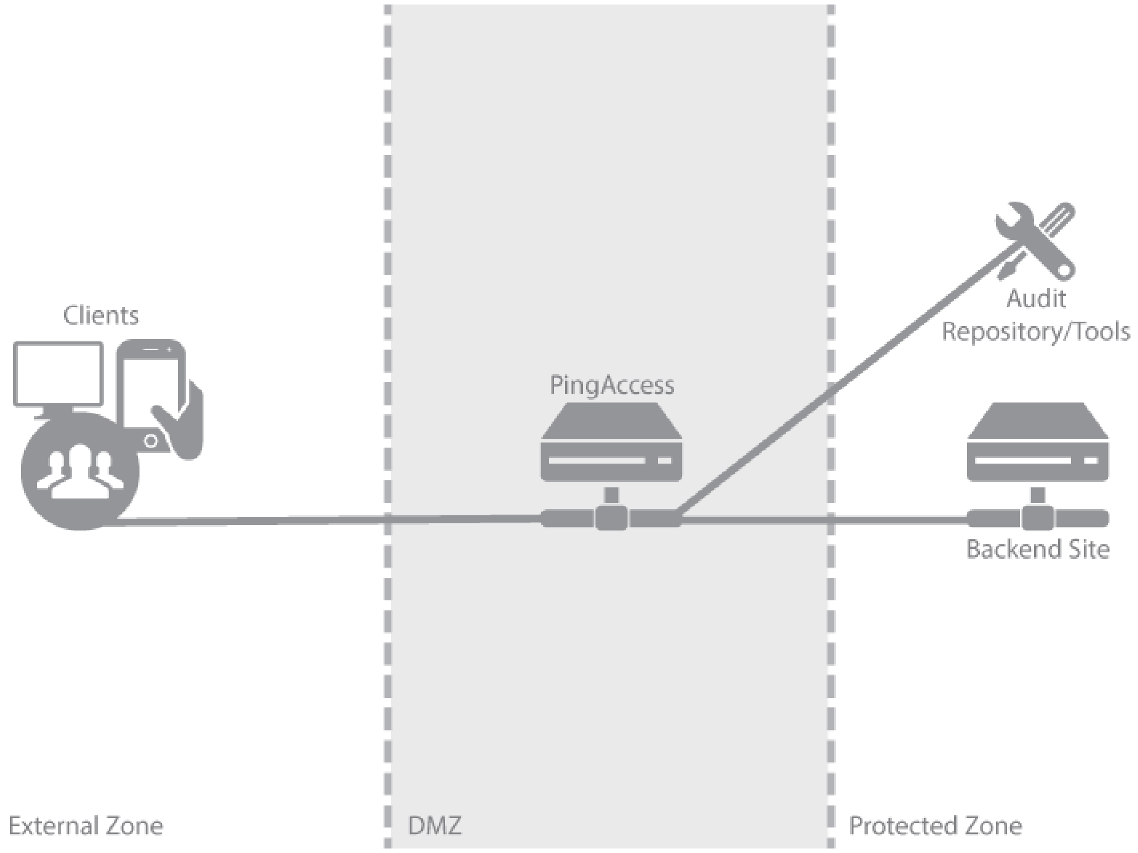
- Receives inbound calls requesting access to protected back-end Sites.
- Audits the request and then makes authorized requests to the back-end Sites.
- Receives and processes responses and relays them on to the clients.

The following sections describe sample Proof of Concept and Production architectures for an auditing/proxying use case deployment.

- *Audit and Proxy POC Deployment Architecture*
- *Audit and Proxy Production Deployment Architecture*

Auditing and Proxying Proof of Concept Deployment Architecture

This environment is used to emulate a production environment for development and testing purposes. In the test environment, PingAccess can be set up with the minimum hardware requirements. Given these conditions, we do not recommend using this proposed architecture in a production deployment as 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 Zone	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.

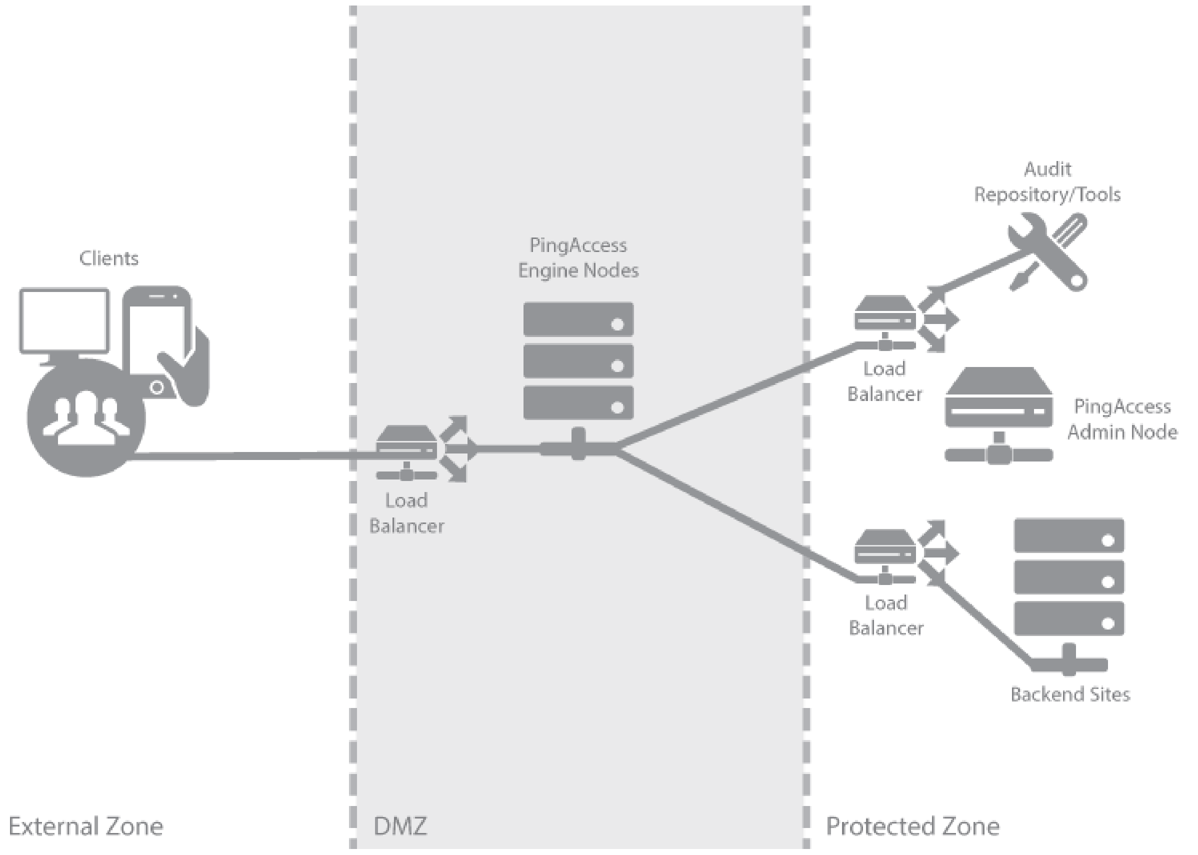
Zone	Description
Protected Zone	Contains back-end Sites audited and proxied through PingAccess. Audit results are sent to an audit repository or digested by reporting tools. Many types of audit repository/tools are supported such as SIEM/GRC, Splunk, database, and flat files.

Auditing and Proxying Production Deployment Architecture

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.

PingAccess can provide high availability and basic load balancing for the protected web apps in the protected zone. See the *Load Balancing Strategies* documentation for more information.

The following environment example is a recommended production quality deployment architecture for an auditing/ proxying use case.



The following table describes the three zones within this proposed architecture.

External Zone	External network where incoming requests originate.
DMZ Zone	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, more nodes may be required.
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.

Port Requirements

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

Direction refers to the direction of requests relative to PingAccess. *Inbound* requests are requests received by PingAccess from external components. *Outbound* requests are requests sent by PingAccess to external components.

Service (Type of Traffic)	Protocol	TCP/UDP	Default Port	Source	Destination	Direction	Description
PingAccess Administrative Console	HTTPS	TCP	9000	PingAccess Administrator browser, PingAccess administrative API REST calls, PingAccess Replica Admin and clustered Engine nodes	PingAccess Administration Engine	Inbound	Used for incoming requests to the PingAccess administrative console. Configurable using the `admin.port` property in the *run.properties* file. This port is also used by clustered engine nodes and the replica admin node to pull configuration data using the admin REST API.
PingAccess Engine	HTTPS	TCP	3000[1]	Client Browser, Mobile Devices,	PingAccess Engine	Inbound	Used for incoming requests to the PingAccess

Service (Type of Traffic)	Protocol	TCP/UDP	Default Port	Source	Destination	Direction	Description
				PingFederate Engine			runtime engine. Configurable using the *Listeners* on page 68 configuration page.
PingAccess Agent	HTTP	TCP	3030	PingAccess Agent	PingAccess Engine	Inbound	Used for incoming Agent requests to the PingAccess runtime engine. Configurable using the `agent.http.port` property of the *run.properties* file.
PingFederate Traffic	HTTPS	TCP	9031	PingAccess Engine	PingFederate	Outbound	Used to validate OAuth Access Tokens, ID Tokens, make STS calls for Identity Mediation, and return authorized information about a user. Configurable using the *PingFederate Settings* page within PingAccess.
PingAccess Cluster Traffic	JGroups	TCP	7610	PingAccess Engine	PingAccess Engine	Inbound	Used for communications between engine nodes in a cluster. Configurable using the

Service (Type of Traffic)	Protocol	TCP/UDP	Default Port	Source	Destination	Direction	Description
							run.properties file.
PingAccess Cluster Traffic	JGroups	TCP	7710	PingAccess Engine	PingAccess Engine	Inbound	Used by other nodes in the cluster as part of the cluster's failure-detection mechanism. Configurable using the *run.properties* file.
PingAccess Cluster Traffic	JGroups	UDP	7500	PingAccess Engine	PingAccess Engine	Inbound	Used by other nodes in the same cluster to share information. Configurable using the *run.properties* file.

1. In addition to port 3000, additional engine listener ports defined in the *Listeners* on page 68 configuration need to be open as well.

Performance Tuning

While PingAccess has been engineered as a high performance engine, its default configuration may not match your deployment goals nor the hardware you have available. Consult the following sections to optimize various aspects of a PingAccess deployment for maximum performance.

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* (https://www.pingidentity.com/support/customer-portal.cfm).

Java Tuning

Garbage Collector Configuration

Resource Pools

Logging and Auditing

Agent Tuning

Java Tuning

Heap (How much memory)

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, best practice is to "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), we recommend that you increase the heap size.

Modifying Heap Size

To modify heap size for run.sh/run.bat scripts, do the following:

1. Edit the run script in the bin directory of the PingAccess install: run.sh on Linux or run.bat on Windows
2. Specify overall heap size by modifying the MINIMUM_HEAP and MAXIMUM_HEAP variables: - Edit -Xms512m and -Xmx512m respectively - Specify units as m (megabytes) or g (gigabytes)
3. Specify young generation size by modifying the MINIMUM_NEW and MAXIMUM_NEW variables: - Edit -XX:NewSize=256m and -XX:MaxNewSize=256m, respectively - Set values to 50% of MINIMUM_HEAP and MAXIMUM_HEAP, respectively

ℹ Not advisable if selecting the G1 collector (see *Garbage Collector Configuration* for more information).

Windows Service

To modify heap size for Windows Service, do the following:

1. Edit the PingAccessService.conf file located in the \sbin\windows directory of the PingAccess install:
2. Specify overall heap size by modifying the wrapper.java.initmemory and wrapper.java.maxmemory settings. - Set the values (in megabytes) for initial and maximum heap sizes, respectively.
3. Specify young generation size by modifying the wrapper.java.additional.11 and wrapper.java.additional.12 settings. - Set the values (in megabytes) for initial and maximum new generation sizes, respectively.
4. Restart. The settings in the PingAccessService.conf file are only applied at service startup.

ℹ Not advisable if selecting the G1 collector (see *Garbage Collector Configuration* for more information).

Linux Service

Since the Linux Service uses the `run.sh` file, the service uses the same Java settings.

Garbage Collector Configuration

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.

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 JVM. No modification is required to the run.sh/run.bat scripts or the Windows Service configuration file or use.

Garbage Collector	Description	Modifications
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	**Run.sh/run.bat scripts**: Set GARBAGE_COLLECTOR variable to **-XX:+UseConcMarkSweepGC** in the run script. **Note**: Quote delimiters required in run.sh, not run.bat. **Windows Service**: Set wrapper.java.additional.10 to **-XX:+UseConcMarkSweepGC** in the PingAccessService.conf file.
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)	**Run.sh/run.bat scripts**: Set GARBAGE_COLLECTOR variable to **-XX:+UseG1GC** in the run script. **Note**: Quote delimiters required in run.sh, not run.bat. Also disable MINIMUM_NEW and MAXIMUM_NEW tuning. Explicit sizing adversely affects pause time goal. To disable, precede variables with **rem** in run.bat, **#** in run.sh. **Windows Service**: Set wrapper.java.additional.10 to **-XX:+UseG1GC** in the PingAccessService.conf file. Also disable wrapper.java.additional.11 and wrapper.java.additional.12. Explicit sizing adversely affects pause time goal. To disable, precede lines with **#**.

Resource Pools

Acceptor Threads

PingAccess uses a pool of threads to respond to HTTP/S requests made to the TCP port(s) 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. A best practice is to use at least two acceptors for performance. On larger multiple CPU core machines, more acceptors can be used. We recommend limiting to between two and 1/4th the number of available CPU cores.

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:

```
admin.acceptors=N
```

```
engine.http.acceptors=N
```

```
agent.http.acceptors=N
```

Where N represents the number of acceptor threads.

Worker Threads

PingAccess uses a pool of *worker* threads to process user requests and a separate pool to process agent requests. Worker threads receive user requests from Acceptor threads, process them, respond back to the client and then return to the pool for reuse. By default, PingAccess starts with a minimum of five worker threads and grows as needed (unbounded by default). You can define the minimum and maximum number of Worker threads in each pool by adding and/or modifying properties found in the `run.properties` file.

To set values, open the `run.properties` file located in the `conf` directory of your PingAccess deployment. If the properties do not exist in the file add them.

```
engine.httptransport.coreThreadPoolSize=N
```

```
engine.httptransport.maxThreadPoolSize=N
```

and

```
agent.httptransport.coreThreadPoolSize=N
```

```
agent.httptransport.maxThreadPoolSize=N
```

Where `N` represents the number of worker threads.

Maintenance of the pool is such that if the number of threads in the pool exceeds the value of `engine.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. Similarly for `agent.httptransport.coreThreadPoolSize` and `agent.httptransport.maxThreadPoolSize`.

Since the pool by default is allowed to grow and shrink based on demand, it is recommended that you tune the `engine.httptransport.coreThreadPoolSize` and `agent.httptransport.coreThreadPoolSize` (minimum) to satisfy moderate demand on the system. We recommend a minimum of 10 threads per available CPU core as a good value to support up to twice the number of concurrent users without error or significant degradation in performance.

Backend Server Connections

PingAccess provides a few options to control and optimize connections to the proxied site.

Max 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 the **Keep Alive Timeout** is reached (see Keep Alive Timeout for more details).

In certain situations it can be advantageous to limit the number of connections in the pool for a given Web site. If, for example, the Web site 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 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.

> We strongly recommended that you understand the limits and tuning of the server application being proxied. Setting the **Max Connections** value too low may create a bottleneck to the proxied site, setting the value too high (or unlimited) may cause PingAccess to overload the server.

See *Sites* for information on setting **Max Connections**.

Keep Alive Timeout

As mentioned in the previous section, the **Keep Alive Timeout** value controls how long a connection created to the proxied Site is kept in the pool for use. This value should be set lower than the HTTP Keep Alive timeout of the Site being proxied.

Configuring PingAccess to timeout the connections before the proxied server ensures that use of “stale” connections to the Site is not attempted, causing failure and retry overhead. To improve efficiency, keep the timeout value of

PingAccess connections as close as possible to the timeout value of the proxied server without matching or going over that value. This depends on the time granularity afforded by the proxied HTTP server's configuration (time set in minutes, seconds, milliseconds, etc.) and may take some testing to fully optimize. As a starting point, we suggest 500 milliseconds (half a second) to one second as PingAccess transactions typically complete in less than a half a second on a properly-sized deployment. See *Sites* for information on setting **Keep Alive Timeout**.

Logging and Auditing

PingAccess uses a high performance, asynchronous logging framework to provide logging and auditing services with as low impact to overall application performance as possible.

Logging

Although logging is handled by a high performance, asynchronous logging framework, it is more efficient to the system overall to log the minimum amount of information required. We highly recommend that you review the section of the documentation for logging and adjust the level to the lowest, most appropriate level to suit your needs (see *Manage Log Files*).

Auditing

As with logging, auditing is provided by the same high performance, asynchronous logging framework. Furthermore, 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), we recommend *auditing to the database*. You will see a decrease in disk I/O, which may result in increased performance depending on database resources.

Agent Tuning

Several properties in the `agent.properties` file can be configured for increased performance. See the agent documentation for *Apache* or *IIS* for more information on agent configuration and setting properties.

Max Connections

Connections from the agent to PingAccess are limited by `agent.engine.configuration.maxConnections`. The default is set to `10`. 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 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 that this is the maximum number of connections per worker process, and not simply the total number of workers the agent has access to. Setting `agent.engine.configuration.maxConnections` value too low may create a bottleneck to PingAccess, and setting the value too high may cause PingAccess to become overloaded.

Max 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.

Server Clustering

PingAccess provides clustering features that allow a group of PingAccess servers to appear as a single system. When deployed appropriately, server clustering can facilitate high availability of critical services. Clustering can also increase performance and overall system throughput.

Two types of information are shared when using server clustering: configuration data and runtime state. Configuration data is replicated to all engine states. At startup, a PingAccess engine node in a cluster checks its local configuration and then makes a call to the administrative console to check for changes. How often each engine in a cluster checks the console for changes is configurable in the engine run.properties file.

⚠ Site configuration data is replicated only if the site is associated with an application. Any sites not associated with an application are only stored in the admin node's configuration database.

Runtime state clustering consists solely of a shared cache of security tokens acquired from the PingFederate STS for *Token Mediation* use cases using the *Token Mediator Site Authenticator*. For increased performance, you can configure engines to share runtime state by *configuring cluster interprocess communication* using the run.properties file. By default, engines do not share runtime state.

For more information on configuring these features, see the *Clustering*, *Configure PingAccess Servers into a Cluster*, and *Configuring Subclusters* sections.

Chapter 5

PingAccess SDK Developer's Guide

- *Preface*
- *Introduction*
- *Getting Started with the SDK*
- *Creating your own Plugins*
- *Implementation Guidelines*

Preface

This document provides technical guidance for using the PingAccess Add-on SDK. Developers can use this guide, in conjunction with the installed Javadocs, to extend the functionality of the PingAccess server.

Intended Audience

This guide is intended for application developers and system administrators responsible for extending PingAccess. The reader should be familiar with Java software-development principles and practices. It describes the development of:

- SiteAuthenticators
- Rules

Additional Documentation

- The PingAccess Javadocs provide detailed reference information for developers. The Javadocs can be accessed with a web browser by viewing the file `<PA_install>/sdk/apidocs/index.html`.

Introduction

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) is able to integrate with web servers or services it is protecting.

These extension points allow users to customize certain behaviors of PingAccess to suit an organization’s needs. This SDK provides the means to develop, compile, and deploy custom extensions to PingAccess.

If you need assistance using the SDK, visit the Ping Identity *Support Center* (www.pingidentity.com/support) to see how we can help you with your application. You may also engage the Ping Identity Global Client Services team for assistance with developing customizations.

Getting 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.

Directory Structure

The PingAccess SDK directory (<PA_install>/sdk) contains the following:

- 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 may 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. You may use these examples for developing your own implementations.
- /samples/SiteAuthenticator/README.md – Contains the details of the Site Authenticator samples.
- /apidocs/ – Contains the SDK Javadocs. Open index.html to get started.

Prerequisites

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
```

If Internet access is unavailable, there are two other ways to reference the PingAccess SDK. First, once Apache Maven is installed, install the sdk into your local dependency repository by running the following command:

```
mvn install:install-file -Dfile=<PA_install>/lib/pingaccess-sdk-3.1.1.1.jar
 -DgroupId=com.pingidentity.pingaccess -DartifactId=pingaccess-sdk -
Dversion=3.1.1.1 -Dpackaging=jar
```

Alternatively, you can update the pingaccess-sdk dependency in your `pom.xml` to point to the local installation.

```
<dependency>
        <groupId>com.pingidentity.pingaccess</groupId>
        <artifactId>pingaccess-sdk</artifactId>
        <version>3.1.0.2</version>
        <scope>system</scope>
        <systemPath><PA_install>/lib/pingaccess-sdk-3.1.0.2.jar</systemPath>
</dependency>
```

With either of these options, replace <PA_install> with the path to the PingAccess installation.

How to install the samples

- Before you begin, ensure you have the Java SDK and Apache Maven installed.
- Each sample type is installed separately:
  - For the Rules samples, navigate to <PA_install>/sdk/samples/Rules
  - For the Site Authenticators samples, navigate to <PA_install>/sdk/samples/SiteAuthenticator
- 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_install>/lib directory.

```
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]
 ------------------------------------------------------------------------
```

Creating your own Plugins

This section describes using the samples as a template for creating your own plugins.

Creating a Rule

- For details on how to create a Rule, reference the javadoc at: <PA_install>/sdk/apidocs/com/pingidentity/pa/sdk/policy/RuleInterceptor.html
- Add a Java class to /sdk/samples/Rules/src that implements com.pingidentity.pa.sdk.policy.RuleInterceptor and is annotated by com.pingidentity.pa.sdk.policy.Rule.   A base class com.pingidentity.pa.sdk.policy.RuleInterceptorBase is available to simplify implementing a Rule.
- Add the class name of the new class to /sdk/samples/Rules/src/main/resources/META-INF/services/com.pingidentity.pa.sdk.policy.RuleInterceptor. Execute maven install on the Rules sample pom.

Creating a Site Authenticator

- For details on how to create a Site Authenticator, reference the javadoc at: <PA_install>/sdk/apidocs/com/pingidentity/pa/sdk/siteauthenticator/SiteAuthenticator.html
- Add a Java class to /sdk/samples/SiteAuthenticator/src that extends com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptorand is annotated by com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticator. A base class com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptorBase is available to simplify implementing a SiteAuthenticator.
- Add the class name of the new class to /sdk/samples/Rules/src/main/resources/META-INF/services/com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticator. Execute maven install on the SiteAuthenticator sample pom.

Implementation Guidelines

The following sections provide specific programming guidance for developing custom interfaces. Note that the information is not exhaustive – consult the Javadocs to find more details about interfaces discussed here as well as additional functionality.

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:

- The Rule annotation on the implementation class of the RuleInterceptor is interrogated to determine which PluginConfiguration instance will be instantiated.
- 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)
- PluginConfiguration.setName(String) is called.
- PA attempts to map the incoming JSON configuration to the PluginConfiguration instance.
- ConfigurablePlugin.configure(PluginConfiguration) is called.
- Validator.validate(Object, Class[]) method is invoked and provided to the RuleInterceptor.
- 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, we recommend that Spring not be used 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

Differences Between Rules for Agents and Sites

Rules may 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}.

Chapter 6

Glossary

This glossary defines the terms used in PingAccess descriptions.

ID Token

A token defined in the *OpenID Connect* standard that represents a set of claims about the end user.

JSON Web Token (JWT)

A URL-safe way of representing claims (attributes) transferred between two parties. This format is intended for space-constrained environments such as HTTP Authorization headers and URI query parameters. For more information, see the *JSON Web Token Specification*.

PingAccess (PA) Token

A JWT Token issued by PingAccess that contains user session attributes used to grant access to Web Resources and placed within a PingAccess cookie. PA Tokens are digitally signed using keys that are internally managed. Configure rolling intervals and cache-related settings for these keys on the *Web Session* page within Settings.

⚠ An end user can access all attributes by examining their cookie contents. While they are integrity protected to prevent changes, any sensitive or confidential attributes can be viewed should the user decode this JWT-formatted cookie.

PKCS#12

An archive file format for storing many cryptography objects as a single file. It is commonly used to bundle a private key with its X.509 certificate.

Web Access Management (WAM) Token

A data object by which a client authenticates to a Web server protected by a third party Web Access Management system. Typically, a WAM token is transported as a cookie that represents a user's logged-in session into the system.

Chapter

7

Release Notes

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 sits in front of applications to protect them, 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.

These release notes summarize the changes in current and previous product updates.

Enhancements for the 3.1 Release

PingAccess 3.1 includes enhancements and new functionality for the Session Management capabilities, the PingAccess Engine, and the Administrative interface.

Session Management

- **Session Attribute Updates and Revocation** - Administrators can now configure PingAccess to periodically query PingFederate to update attributes associated with the session, and to terminate the session based on a determination by PingFederate that the user no longer meets the criteria used to issue a token. For example, if PingFederate is configured to return the user's attributes only if the user account is enabled, disabling the user account can now trigger a session revocation. Additionally, if a user is removed from a group that grants them access to an application, access can be denied for a current session.
- **Support for Large Attribute Data** - PingAccess now has the ability to cache user attribute data that previously was limited to the browser maximum cookie size. When this feature is enabled, potentially large attribute values - such as group memberships - can be used in policy decisions.
- **OpenID Connect / OAuth 2.0 Form Post Response Mode** - Support has been added for the emerging *OAuth V2 Form Post Response Mode* standard. If you are upgrading from an earlier release of PingAccess, web sessions using the existing **POST** method will be migrated to **x_post**.

Engine

- **HTTP Request Configuration Source Handling** - To better integrate PingAccess with configurations using reverse proxies and external load balancers, support has been added to support arbitrary IP Source, Host Source, and Protocol Source headers. This allows headers such as the **X-Forwarded-For** header to be injected by those reverse proxies in order to preserve information about the originating host IP address, hostname, and protocol source, in order to be able to use that information to make policy decisions. In addition, the originating host IP address is recorded in the audit logs.
- **HTTP Response Body Content Rewriting** - The new *Rewrite Content Rule* on page 35 allows arbitrary content rewriting to be performed on outbound content. The content rewriting functionality can be used, for example, to rewrite URL text in HTTP responses so links a user might click on will use the external hostname for the Application rather than an internal name. This feature can also be constrained to particular content-types, allowing different rules to be tailored to the response `Content-Type` header.
- **Multiple Engine Ports** - The PingAccess Engine listener can now listen on multiple ports, providing greater configuration flexibility.

- **Specify Different PingFederate Runtime Engines for Backchannel Calls** - PingAccess can now use separate hostnames and ports to perform behind-the-scenes communication with PingFederate, providing greater flexibility in managing traffic between the two products. If more than one backchannel communication is set up, a built-in availability profile is used to provide failover.

Administration

- **Configurable Signature Algorithm Generated Key Pairs** - When generating a key pair, the **Signature Algorithm** can now be selected, and the options available are based on the chosen **Key Algorithm**.
- **Remove Resource Ordering** - The determination of the "most specific" match of an application resource path has been simplified, removing the need for manual ordering of resources within an application. PingAccess now evaluates this based on the length of the path requested and matching that to the Path Prefixes defined in the Application. This change also removes the `PATCH` method from the `/applications/{id}/resources` Administrative API endpoint, since that method was used to update the order of Resources in an Application.
- **Authentication Requirements for Admin SSO** - Authentication Requirements can now be specified for administrator Single Sign-On. This can be used to ensure administrative users log in with stronger forms of authentication than just a username and password.

Known Issues

- Internet Explorer and Firefox do not correctly support the HTML5 time tag. When using the Time Range rule, enter time in 24-hour format.
- PingFederate does not appear as an Authorization Server in Applications unless it is configured as an OAuth Resource server to validate OAuth access tokens. See the *PingFederate Settings*.
- 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.
- When using Java 8, when using virtual host-specific key pair assignments, only the **Host** value from the Virtual Host configuration is used to select a key pair. The **Port** value is not used.

Complete Change List by Released Version

PingAccess 3.1.1 - February, 2015

PingAccess 3.1.1 includes the following fix:

- Addressed an issue that may prevent updating of Resources, which can also affect upgrades.

Previous releases

- *PingAccess 3.1 - February, 2015* on page 123
- *PingAccess 3.0 R2 - October, 2014*
- *PingAccess 3.0.3 - November, 2014*
- *PingAccess 3.0.2 - September, 2014*
- *PingAccess 3.0.1 - August 2014*
- *PingAccess 3.0 - July 2014*
- *PingAccess 2.1.4 - June 2014*
- *PingAccess 2.1.3 - May 2014*
- *PingAccess 2.1.2 - April 2014*
- *PingAccess 2.1.1 - March 2014*
- *PingAccess 2.1 - December 2013*
- *PingAccess 2.0.1 - October 2013*
- *PingAccess 2.0 - September 2013*
- *PingAccess 1.0 - April 2013*

PingAccess 3.1 - February, 2015

PingAccess 3.1 includes enhancements and new functionality for the Session Management capabilities, the PingAccess Engine, and the Administrative interface.

Session Management

- **Session Attribute Updates and Revocation** - Administrators can now configure PingAccess to periodically query PingFederate to update attributes associated with the session, and to terminate the session based on a determination by PingFederate that the user no longer meets the criteria used to issue a token. For example, if PingFederate is configured to return the user's attributes only if the user account is enabled, disabling the user account can now trigger a session revocation. Additionally, if a user is removed from a group that grants them access to an application, access can be denied for a current session.
- **Support for Large Attribute Data** - PingAccess now has the ability to cache user attribute data that previously was limited to the browser maximum cookie size. When this feature is enabled, potentially large attribute values - such as group memberships - can be used in policy decisions.
- **OpenID Connect / OAuth 2.0 Form Post Response Mode** - Support has been added for the emerging *OAuth V2 Form Post Response Mode* standard. If you are upgrading from an earlier release of PingAccess, web sessions using the existing **POST** method will be migrated to **x_post**.

Engine

- **HTTP Request Configuration Source Handling** - To better integrate PingAccess with configurations using reverse proxies and external load balancers, support has been added to support arbitrary IP Source, Host Source, and Protocol Source headers. This allows headers such as the **X-Forwarded-For** header to be injected by those reverse proxies in order to preserve information about the originating host IP address, hostname, and protocol source, in order to be able to use that information to make policy decisions. In addition, the originating host IP address is recorded in the audit logs.
- **HTTP Response Body Content Rewriting** - The new *Rewrite Content Rule* on page 35 allows arbitrary content rewriting to be performed on outbound content. The content rewriting functionality can be used, for example, to rewrite URL text in HTTP responses so links a user might click on will use the external hostname for the Application rather than an internal name. This feature can also be constrained to particular content-types, allowing different rules to be tailored to the response `Content-Type` header.
- **Multiple Engine Ports** - The PingAccess Engine listener can now listen on multiple ports, providing greater configuration flexibility.
- **Specify Different PingFederate Runtime Engines for Backchannel Calls** - PingAccess can now use separate hostnames and ports to perform behind-the-scenes communication with PingFederate, providing greater flexibility in managing traffic between the two products. If more than one backchannel communication is set up, a built-in availability profile is used to provide failover.

Administration

- **Configurable Signature Algorithm Generated Key Pairs** - When generating a key pair, the **Signature Algorithm** can now be selected, and the options available are based on the chosen **Key Algorithm**.
- **Remove Resource Ordering** - The determination of the "most specific" match of an application resource path has been simplified, removing the need for manual ordering of resources within an application. PingAccess now evaluates this based on the length of the path requested and matching that to the Path Prefixes defined in the Application. This change also removes the `PATCH` method from the `/applications/{id}/resources` Administrative API endpoint, since that method was used to update the order of Resources in an Application.
- **Authentication Requirements for Admin SSO** - Authentication Requirements can now be specified for administrator Single Sign-On. This can be used to ensure administrative users log in with stronger forms of authentication than just a username and password.

PingAccess 3.0 R2 - October, 2014

This release introduces the following new features:

- **Backup Admin Console Nodes** - Provides the ability, in a clustered environment, to create a backup administrative node that the administrator can manually fail over to in the event of a catastrophic failure of the primary administrative node.
- **Failover and Load Balancing for Sites** - Adds new functionality to provide failover and load balancing to multiple backend target servers without requiring a load balancer.
- **Ability to Ignore HTTPS Certificate Errors** - Reduces certificate management burden for internal servers in a controlled environment by allowing the administrator to ignore certificate errors for backend connections, such as connections to Site targets.
- **Heartbeat Endpoint Enhancements** - Enhances the monitoring capabilities by adding functionality to the heartbeat to return a configurable list of performance metrics as a JSON payload for consumption by third party monitoring tools.
- **Logging Enhancements** - Adds logging options for rewritten cookies as well as cookies passed or proxied, and realign logged information with appropriate log levels.

PingAccess 3.0.3 - November, 2014

This release addresses the following issue:

- Corrected a potential security issue in Identity Mappings (SECBL006)

PingAccess 3.0.2 - September, 2014

This release addresses the following issue:

- Resolved an issue with Session Validation causing the token mediator to stop working.

PingAccess 3.0.1 - August, 2014

This release addresses the following issues:

- Resolved an issue with token validation when using OAuth Admin API Authentication
- Corrected handling of PingFederate Runtime Base Path setting
- OIDC callback endpoint improvements to support front-end load balancers listening on a different port than the PingAccess Engine

PingAccess 3.0 - July, 2014

- **PingAccess Agents** - added PingAccess Agents to provide additional architectural flexibility with an agent based deployment model.
- **Session Management enhancements** - Web session management has been enhanced to offer additional security for end user-driven logout use cases.
- **Add-on Java SDK** - new PingAccess add-on Java SDK has been introduced.
- **TLS Server Name Indication support** - HTTPS listener configuration has been extended to support the TLS Server Name Indication (SNI) extension.
- **Request/Response Time Auditing** - additional fields have been added to the engine audit logs for performance monitoring and capacity planning purposes - total request processing time and back-end proxy response time.
- **Administration enhancements** - many enhancements have been made to improve the administration and modeling of configuration in PingAccess.

PingAccess 2.1.4 – June 2014

- Resolved an issue with Resource Ordering in the Administrative Console.
- improved interoperability with backend applications introduced by URL filtering in PingAccess 2.1.3.

PingAccess 2.1.3 - May 2014

- Fix a potential security issue that affects deployments that have varying policy applied across a single virtual server.

PingAccess 2.1.2 – April 2014

- Allow for specifying the base path for PingFederate. Useful when PingFederate is behind a reverse proxy.

PingAccess 2.1.1 – March 2014

- Fix Identity Mediation back channel communication issue.
- Fix Web Session cookie attribute handling.

PingAccess 2.1 - December 2013

PingAccess 2.1 – December 2013

- Ability to encrypt the PA session token.
- Sites can have multiple Site Authenticators configured.
- Added Authentication Requirements policy to allow step-up authentication to a Resource.
- Ability to specify “Any” or “All” processing to policy rules within a rule set.
- Multiple Web Sessions can be configured to scope a PingAccess session for a specific set of Resources.
- Added OpenID Connect Basic Profile flow for obtaining claims from PingFederate.
- Enhanced Audit log options to database and Splunk.

PingAccess 2.0.1 - October 2013

PingAccess 2.0.1 – October 2013

- Fixed well-known HTTP/S port issue
- Fixed a potential security issue with the Web Session Header Site Authenticator

PingAccess 2.0 - September 2013

PingAccess 2.0 – September 2013 Initial General Availability (GA) release

PingAccess 1.0 - April 2013

PingAccess 1.0 - April 2013 Limited release