

# PingAccess<sup>®</sup>

Server 5.0



# Copyright

---

© 2018 Ping Identity® Corporation. All rights reserved.

PingAccess Server documentation  
Version 5.0  
February, 2018

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

## Trademark

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

## Disclaimer

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

## 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 [docs.pingidentity.com](https://docs.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 1, 2018**.

# Contents

<b>PingAccess overview.....</b>	<b>10</b>
PingAccess overview.....	10
What is PingAccess?.....	10
PingAccess for Azure AD.....	10
What can I do with PingAccess?.....	11
How does PingAccess work?.....	13
WAM session initiation.....	13
Token mediation.....	14
What can I configure with PingAccess?.....	15
<b>PingAccess user interface reference guide.....</b>	<b>19</b>
<b>PingAccess User Interface Reference Guide.....</b>	<b>19</b>
Applications.....	19
Manage Applications.....	19
Manage Applications - Field Descriptions.....	20
Manage Application Resources.....	21
Manage Application Policies.....	22
Sites.....	22
Sites.....	23
Manage Sites.....	23
Manage Sites - Field Descriptions.....	23
Site Authenticators.....	24
Manage Site Authenticators.....	24
Agents.....	27
Manage Agents.....	27
Manage Agents - Field Descriptions.....	28
Policies.....	29
Manage Rules.....	29
Authentication requirements rule.....	30
Cross-origin Request rule.....	31
Rewrite rules overview.....	31
Groovy script rule.....	36
HTTP Request Header rule.....	36
HTTP Request Parameter rule.....	37
Network Range rule.....	37
OAuth Attribute Value rule.....	38
OAuth Groovy Script rule.....	38
Configure an OAuth Scope rule.....	38
OAuth Token Cache Time To Live rule.....	38
Rate Limiting rule.....	39
Time Range rule.....	40
Web Session Attribute rule.....	40
WebSocket Handshake rule.....	40
Manage Rule Sets.....	41
Manage Rule Set Groups.....	41
Access.....	42
Authentication Requirements.....	43
Configure an authentication requirements list.....	43

Edit an authentication requirements list.....	43
Delete an authentication requirements list.....	43
Identity Mappings.....	43
Configure Auth Token Management.....	43
Create a new identity mapping.....	44
Edit an identity mapping.....	45
Delete an identity mapping.....	45
Virtual Hosts.....	46
Create a new virtual host.....	46
Configure virtual host trusted certificate groups.....	46
Edit a virtual host.....	47
Delete a virtual host.....	47
Web Sessions.....	47
Configure Web Session Management settings.....	48
Create a Web Session.....	48
Edit a Web Session.....	51
Delete a Web Session.....	51
Unknown Resources.....	51
Configure unknown resource management.....	51
Networking.....	52
Availability Profiles.....	52
Create an availability profile.....	52
Edit an availability profile.....	53
Delete an availability profile.....	53
HTTP Requests.....	53
Configure an alternative IP Source header.....	53
Configure an alternative Host Source header.....	53
Configure an alternative Protocol Source header.....	54
Listeners.....	54
HTTPS listeners.....	54
Engine key pairs.....	54
Engine listeners.....	54
Load Balancing Strategies.....	55
Configure a Load Balancing Strategy.....	55
Edit a Load Balancing Strategy.....	55
Delete a load balancing strategy.....	55
Proxies.....	56
Add a proxy.....	56
Edit a proxy.....	56
Delete a proxy.....	56
Security.....	56
Certificates.....	57
Import a certificate.....	57
Delete a certificate.....	57
Create a Trusted Certificate Group.....	57
Add a certificate to a Trusted Certificate Group.....	58
Edit a Trusted Certificate Group.....	58
Remove a Certificate from a Trusted Certificate Group.....	58
Delete a Trusted Certificate Group.....	58
Key Pairs.....	58
Import an existing key pair.....	58
Generate a new key pair.....	59
Generate a Certificate Signing Request.....	59
Import a Certificate Signing Request Response.....	59
Add a certificate to a key pair.....	59
Remove a certificate from a key pair.....	60

Download a Certificate.....	60
Delete a key pair.....	60
System.....	60
Admin Authentication.....	60
Configure Basic Authentication.....	60
Configure Admin UI SSO Authentication.....	61
Configure API Authentication.....	63
Configuration Export/Import.....	63
Export PingAccess configuration.....	63
Import PingAccess configuration.....	64
Clustering.....	64
Engines.....	66
Administrative nodes.....	67
License.....	68
Upload or view a PingAccess license.....	68
Token Provider.....	68
Manage Token Provider.....	68
<b>Install PingAccess.....</b>	<b>75</b>
Install PingAccess.....	75
Installation requirements.....	75
System requirements.....	75
Hardware Requirements.....	76
Port requirements.....	77
Install PingAccess on Linux.....	78
Install PingAccess on Red Hat Enterprise Linux.....	79
Install PingAccess for Windows.....	79
Start PingAccess.....	80
Access the admin console for the first time.....	81
Access the PingAccess administrative API.....	81
Access the interactive administrative API documentation.....	82
Change configuration database passwords.....	82
Stop PingAccess.....	82
Run PingAccess as a service.....	83
Configure PingAccess to run as a Linux systemd service.....	83
Configure PingAccess to run as a Linux systemv service.....	83
Configure Multiple Instances of PingAccess as Linux services.....	84
Remove the PingAccess Linux service.....	84
Configure PingAccess to run as a Windows service.....	84
Remove the PingAccess Windows service.....	85
Uninstall PingAccess.....	85
<b>Upgrade PingAccess.....</b>	<b>86</b>
Upgrade a PingAccess standalone version.....	86
Upgrade a PingAccess cluster.....	87
Upgrade Windows using the installer.....	88
Upgrade RHEL using the installer.....	88
Complete the upgrade.....	89
Restore a PingAccess configuration backup.....	95
<b>Configure session management.....</b>	<b>96</b>
Configure session management.....	96
Configure server-side session management.....	96

Configure PingFederate for session management.....	97
Configure PingFederate for user-initiated single logout.....	97
Configure PingAccess for server-side session management.....	97
<b>Configure logging.....</b>	<b>99</b>
Configure logging.....	99
Security audit logging.....	99
Logging.....	101
Configure log levels.....	102
Configure a class or package log level.....	102
Enable cookie logging.....	102
Append log messages to syslog and the console.....	103
Write logs to other formats.....	103
Write logs to databases.....	103
Write audit logs for Splunk.....	106
<b>Manage APIs.....</b>	<b>108</b>
PingAccess endpoints.....	108
Heartbeat endpoint.....	108
OpenID Connect endpoints.....	110
Auth Token Management endpoint.....	110
Administrative API endpoints.....	110
<b>Perform a zero downtime upgrade.....</b>	<b>112</b>
Introduction.....	112
Step 1: Disable key rolling.....	112
Step 2: Upgrade the Admin node.....	113
Step 3: Upgrade engines.....	114
Remove the engine from the load balancer configuration.....	114
Upgrade the engine.....	115
Resume configuration replication.....	116
Add the engine to the load balancer configuration.....	117
Step 4: Enable key rolling.....	118
Recovering from a failed upgrade.....	119
<b>Protect a web-based application using PingFederate and PingAccess in a proxy deployment.....</b>	<b>120</b>
Protect a web-based application in a proxy deployment.....	120
Configure PingFederate for PingAccess connectivity.....	120
Connect to PingFederate and configure an application in PingAccess.....	125
Test the configuration.....	128
<b>Customize and localize PingAccess.....</b>	<b>129</b>
Customization of user-facing pages.....	129
Localization of user-facing pages.....	130
<b>Groovy development guide.....</b>	<b>132</b>
Groovy.....	132
Groovy scripts.....	133

Body object.....	134
Exchange object.....	135
Headers object.....	136
Identity object.....	138
JsonNode object.....	138
Logger object.....	140
MediaType object.....	140
Method object.....	141
OAuth Token object.....	141
PolicyContext object.....	142
Request object.....	142
Response object.....	144
SslData object.....	145
Groovy script examples.....	145
Matchers.....	146
<b>Addon SDK for Java.....</b>	<b>151</b>
Preface.....	151
Introduction.....	151
Get started with the SDK.....	152
SDK directory structure.....	152
SDK prerequisites.....	152
How to install the SDK samples.....	153
Create your own Plugins.....	154
Create an identity mapping.....	154
Create a load balancing strategy.....	155
Create a locale override service.....	155
Create a rule.....	155
Create a site authenticator.....	155
Implementation guidelines.....	156
<b>Agent SDK for Java.....</b>	<b>158</b>
Preface.....	158
Introduction.....	158
Agent SDK directory structure.....	159
Agent SDK prerequisites.....	159
How to install the servlet filter sample.....	160
PingAccess Agent SDK for Java Release History.....	161
<b>Agent SDK for C.....</b>	<b>162</b>
Preface.....	162
Introduction.....	162
Getting Started with the PingAccess Agent SDK for C.....	163
Agent SDK for C directory structure.....	163
Agent SDK for C sample code.....	164
PingAccess Agent SDK for C release history.....	164
<b>Addon SDK for Java migration guide.....</b>	<b>165</b>
PingAccess Addon SDK for Java Migration Guide.....	165
<b>Performance tuning guide.....</b>	<b>194</b>

Performance tuning.....	194
Java tuning.....	194
Configure JVM crash log in Java startup.....	194
Configure memory dumps in Java startup.....	194
Modify the Java heap size.....	195
Operating system tuning.....	195
Linux tuning.....	195
Windows tuning.....	196
Garbage collector configuration.....	197
Acceptor threads.....	197
Worker threads.....	198
Backend server connections.....	198
Logging and Auditing.....	199
Logging.....	199
Auditing.....	199
Agent tuning.....	199
<b>Clustering reference guide.....</b>	<b>200</b>
Clustering.....	200
Configure cluster prerequisites.....	202
Configure a PingAccess cluster.....	202
Configure the primary administrative node.....	203
Configure PingAccess subclusters.....	204
Configure the Replica Administrative Node.....	204
Manual fail over to the replica administrative node.....	205
Reinstate a replica administrative node after failing over.....	205
Configure an engine.....	206
Edit an engine.....	206
Remove an engine's access to the Administrative Console.....	206
Remove an engine.....	206
<b>Deployment guide.....</b>	<b>208</b>
PingAccess deployment guide.....	208
Use cases and deployment architecture.....	208
Deploy for Gateway Web Access Management.....	208
Deploy for Agent Web Access Management.....	209
Deploy for Gateway API Access Management.....	209
Deploy for auditing and proxying.....	210
Configuration by use case.....	210
Web Access Management Gateway deployment.....	211
Web Access Management Agent deployment.....	211
API Access Management Gateway deployment.....	212
Auditing and proxying Gateway deployment.....	213
Web Access Management.....	213
Choose Between an Agent or Gateway deployment.....	213
Web Access Management Gateway proof of concept deployment architecture.....	214
Web Access Management Gateway production deployment architecture.....	215
Web Access Management Agent proof of concept deployment architecture.....	217
Web Access Management Agent production deployment architecture.....	218
API Access Management proof of concept deployment architecture.....	219
API Access Management production deployment architecture.....	220
Auditing and proxying proof of concept deployment architecture.....	221
Auditing and proxying production deployment architecture.....	222



<b>Configuration file reference guide.....</b>	<b>224</b>
Configuration file reference.....	224
<b>PingAccess for AWS: Solution setup guide.....</b>	<b>235</b>
Introduction.....	235
Setup and deployment.....	235
AWS prerequisites.....	235
Obtain the automation ZIP file.....	237
Create the PingAccess software repository.....	237
Populate the S3 bucket.....	237
Create the PingAccess key pair.....	238
Create the PingAccess VPC.....	238
Configure Security Groups.....	243
Access the PingAccess administration console.....	243
Access the PingFederate administration console.....	244
Configure SSH connectivity.....	244
View PingAccess log files.....	245
Maintaining the S3 bucket.....	246
Delete the PingAccess VPC.....	246
Redeploying with an existing configuration.....	246
Troubleshooting VPC creation.....	246
<b>PingAccess for AWS: PingFederate environment requirements.....</b>	<b>248</b>
Introduction.....	248
Create an IAM User in AWS.....	248
PingFederate configuration requirements.....	249
Modify api.json to include details about your environment.....	254
<b>PingAccess for AWS: Configure an application.....</b>	<b>255</b>
Introduction.....	255
Protect an application.....	255
Install and access the unprotected PingAccess QuickStart application.....	255
Configure PingFederate.....	256
Configure a PingAccess web application.....	256
Configure a PingAccess API application.....	258
Test the web application.....	259
Test the API application.....	259
<b>PingAccess release notes.....</b>	<b>261</b>
Release Notes.....	261
PingAccess 5.0.1 - December 2017.....	261
Known issues and limitations.....	261
Previous releases.....	262
PingAccess 5.0 - December 2017.....	262
PingAccess 4.3.4 - December 2017.....	267
PingAccess 4.3.3 - October 2017.....	267
PingAccess 4.3.2 - September 2017.....	267
PingAccess 4.3.1 - August 2017.....	268
PingAccess 4.3 - June 2017.....	268
PingAccess 4.2.3 - April 2017.....	270

PingAccess 4.2.2 - February 2017.....	270
PingAccess 4.2.1 - December 2016.....	271
PingAccess 4.2 - December 2016.....	271
PingAccess 4.1.3 - November 2016.....	272
PingAccess 4.1.2 - October 2016.....	273
PingAccess 4.1.1 - September 2016.....	273
PingAccess 4.1 - August 2016.....	274
PingAccess 4.0.4 - July 2016.....	275
PingAccess 4.0.3 - May 2016.....	275
PingAccess 4.0.2 - April 2016.....	276
PingAccess 4.0.1 - March 2016.....	276
PingAccess 4.0 - February 2016.....	277
PingAccess 3.2.6 - February 2016.....	282
PingAccess 3.2.5 - December 2015.....	282
PingAccess 3.2.4 - December 2015.....	282
PingAccess 3.2.3 - October 2015.....	282
PingAccess 3.2.2 - August 2015.....	283
PingAccess 3.2.1 - July 2015.....	284
PingAccess 3.2 - June 2015.....	284
PingAccess 3.1 - February 2015.....	285
PingAccess 3.0 R2 - October 2014.....	287
PingAccess 3.0.3 - November 2014.....	287
PingAccess 3.0.2 - September 2014.....	287
PingAccess 3.0.1 - August 2014.....	287
PingAccess 3.0 - July 2014.....	287
PingAccess 2.1.4 - June 2014.....	288
PingAccess 2.1.3 - May 2014.....	288
PingAccess 2.1.2 - April 2014.....	288
PingAccess 2.1.1 - March 2014.....	288
PingAccess 2.1 - December 2013.....	288
PingAccess 2.0.1 - October 2013.....	288
PingAccess 2.0 - September 2013.....	288
PingAccess 1.0 - April 2013.....	288

# PingAccess overview

---

## PingAccess overview

---

This document provides an overview of PingAccess. Use this document to gain an understanding of the product, to learn about what you can do, and to discover the many features it provides. To get the most from PingAccess, users should read about and understand the concepts included in this document.

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

This document answers the following questions:

- [What is PingAccess?](#) on page 10
- [What can I do with PingAccess?](#) on page 11
- [How does PingAccess work?](#) on page 13
- [What can I configure with PingAccess?](#) on page 15

## What is PingAccess?

---



PingAccess is an identity-enabled access management product that protects Web Applications and APIs by applying security policies to client requests. In simpler terms, PingAccess allows you to protect sites, APIs, and other resources using rules and other authentication criteria. It works in conjunction with PingFederate or other common token provider via the OAuth 2.0 and OpenID Connect protocols to integrate identity-based access management policies via a federated corporate identity store using open standards access protocols.

## PingAccess for Azure AD

---

PingAccess for Azure AD is a free version of PingAccess for users of Microsoft's Azure AD that allows you to protect up to 20 applications. The goal of this solution is to allow for greater control over the access to legacy on-premise applications through the use of PingAccess Identity Mapping functionality.

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

-  **Important:** When your PingAccess for Azure AD license expires, access to the admin API is removed and you are unable to configure the product. Though managed access to configured applications continues, you must upload a new license file before you can make any additional configuration changes.
-  **Upgrade notice:** PingAccess for Azure AD provides a limited feature set that may not be compatible with existing PingAccess configurations. For this reason, upgrading from an earlier full version of PingAccess to PingAccess for Azure AD is **not supported**.

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

Functionality	PingAccess	PingAccess for Azure AD
Create applications	Yes	Limited to 20 web session applications.
Create site authenticators	Yes	Limited to Basic and Mutual TLS.

Configure identity mappings	Yes	Limited to Header and JWT.
Create load balancing strategies	Yes	Limited to Header-Based and Round Robin.
Configure web sessions	Yes	Limited to web sessions with OIDC login type CODE.
Configure token provider	Yes	Limited to Microsoft Azure AD authentication source.
Export/Import configuration	Yes	Limited to configurations that includes only features permitted by license type.
Configure policies	Yes	No
Specify authentication requirements	Yes	No
Create and configure custom plugins using the SDK	Yes	No
Configure sites	Yes	Yes
Configure agents	Yes	Yes
Create virtual hosts	Yes	Yes
Configure unknown resource handling	Yes	Yes
Configure availability profiles	Yes	Yes
Configure HTTP request handling	Yes	Yes
Configure listeners	Yes	Yes
Configure forward proxy settings	Yes	Yes
Manage certificates	Yes	Yes
Manage key pairs	Yes	Yes
Configure administrator authentication	Yes	Yes
Configure clustering	Yes	Yes
Manage licenses	Yes	Yes

## What can I do with PingAccess?

---

PingAccess provides a highly customizable solution to identity access management that allows you to control access in a variety of ways by specifying a wide range of conditions that must be satisfied. Read the following sections to discover the methods PingAccess uses to control access and perform system functions. To learn more about the configuration required for any of the following topics, see PingAccess configuration scenarios on [docs.pingidentity.com](https://docs.pingidentity.com).

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

- Use PingAccess to protect the application and API resources to which client requests are forwarded.
- Partition applications for tighter access control through the use of resources.

- Customize configuration of site authenticators and authentication requirements to suit the security needs of your organization.
- Incorporate legacy authentication mechanisms through Token Mediation.
- Apply policies to define how and when a client can access target resources.

Customize your identity access management configuration with the following features.

### **Apply policies**

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

### **Backup and restore**

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

### **Configure a token provider**

PingAccess can be configured to use PingFederate as the token provider or may be configured to use a common token provider via the OAuth 2.0 or OpenID Connect protocols.

### **Configure administrator authentication**

Allow administrators to authenticate with a simple username and password, or configure them to authenticate via SSO or API in conjunction with PingFederate.

### **Configure advanced network settings**

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

### **Configure logging**

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

### **Configure Single Logout**

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

### **Create clusters**

Deploy PingAccess in a clustered environment to provide higher scalability and availability for critical services. Use subclusters to provide better scaling of large PingAccess deployments by allowing multiple engine nodes in the configuration to share information. Place a load balancer in front of each subcluster in order to distribute connections to the nodes in the subcluster.

### **Customize PingAccess look and feel**

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

### **Customize with SDKs**

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

### **Manage certificates and key pairs**

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

### **Manage sessions**

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

### **Manually configure runtime parameters**

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

### Protect an application or API

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

### Tune performance

Optimize a wide variety of PingAccess components for maximum performance.

### Upgrade an existing installation

Easily upgrade an existing installation using the installer, or more carefully manage the upgrade process with the PingAccess Upgrade Utility.

### Use APIs

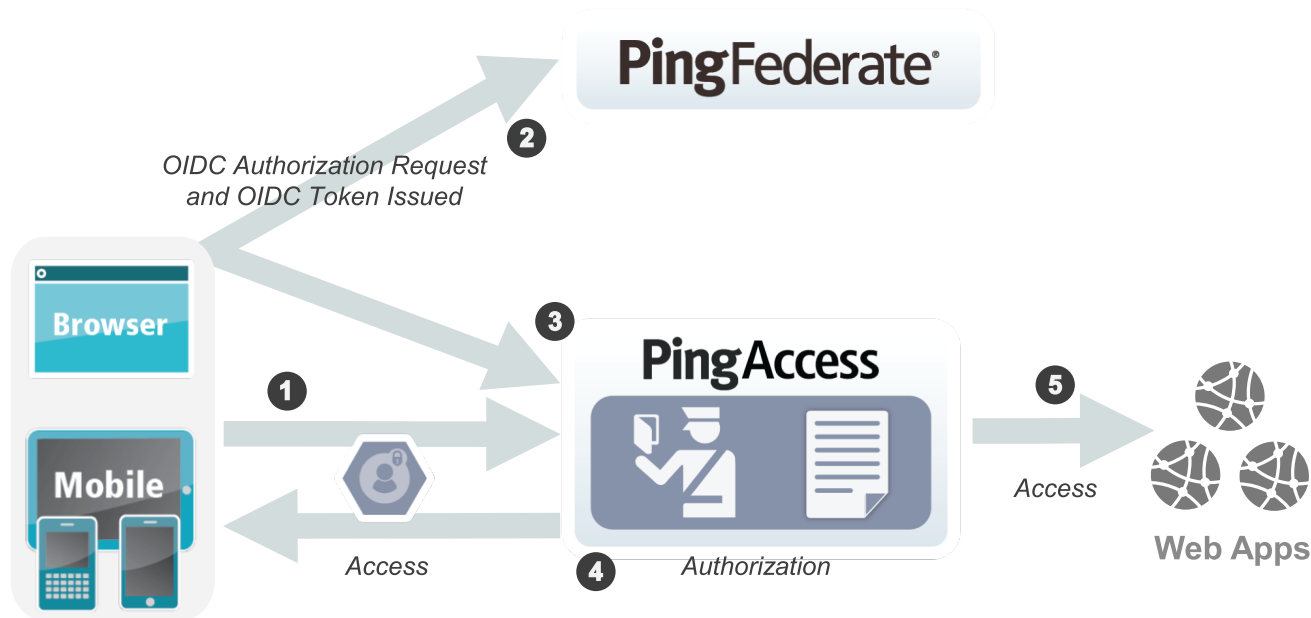
Use the PingAccess APIs to provide a powerful configuration and management experience outside the PingAccess user interface.

## How does PingAccess work?

Access requests are either routed through a PingAccess Gateway to the target Site, or they are intercepted at the target web application server by a PingAccess Agent, which in turn coordinates access policy decisions with a PingAccess Policy Server. In either instance, policies applied to access requests for the target Application are evaluated, and PingAccess makes a policy-based decision to grant or deny access to the requested resource. When access is granted, client requests and server responses can be modified to provide additional identity information required by the target Application.

### WAM session initiation

When a user authenticates, PingAccess applies the application and resource-level policies to the 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 access to a Web resource from PingAccess, PingAccess inspects the request for a PingAccess Token.

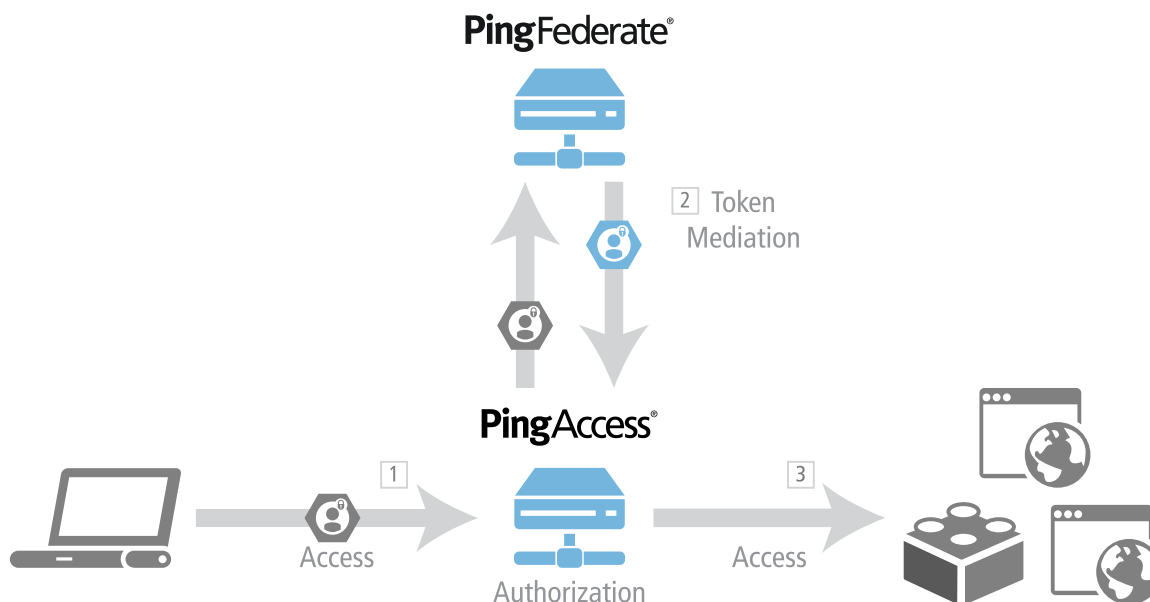
2. If the PA Token is missing, PingAccess redirects the user to an OpenID Connect Provider (OP) for authentication.
    - ➔ **Info:** When using an OP, an OAuth Client must already be configured in PingAccess. For steps on configuring an OAuth Client within PingFederate, see the *PingFederate Administrator's Manual*. To configure the OAuth Client within PingAccess, see the PingAccess scenario to *Configure a Token Provider*.
  3. The OP follows the appropriate authentication process, evaluates domain-level policies, and issues an OpenID Connect (OIDC) ID Token to PingAccess.
  4. PingAccess validates the ID Token and issues a 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.
    - ➔ **Info:** 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).
- ➔ **Info:** See the Session Management scenario for more information.

## Token mediation

When planning a PingAccess deployment, it is necessary to take inventory 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 continued 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.

- ➔ **Info:** This example assumes the user has already obtained a PA Token or OAuth Bearer Token. See the *Session Management* scenario for information 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).

## What can I configure with PingAccess?

---

PingAccess includes a wide range of features that allow you to customize your identity access management deployment. To learn more about these features, read the following descriptions.

### Agents

Agents are web server plugins that are installed on the web server hosting the target application. Agents intercept client requests to protected applications and allow or deny the request to proceed by consulting the Policy Manager or using cached information. Agents communicate with the PingAccess Policy Server 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.

### 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.

### 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.

### Auth token management

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

### Availability profiles

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



If multiple targets are specified in a site configuration but a load balancing strategy is not applied, then the Availability Profile 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.

### **Certificates**

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

Certificates used by PingAccess 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

### **Clustering**

PingAccess can be configured in a clustered environment to provide higher scalability and availability for critical services.

PingAccess clusters are made up of three types of nodes:

#### **Administrative Console**

Provides the administrator with a configuration interface

#### **Replica Administrative Console**

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

#### **Clustered Engine**

Handles incoming client requests and evaluates policy decisions based on the configuration replicated from the administrative console

Any number of clustered engines can be configured in a cluster, but only one administrative console and one replica administrative console can be configured in a cluster.

Further use of subclusters provides better scaling of very large PingAccess deployments by allowing multiple engine nodes in the configuration to share certain information.

### **HTTP requests**

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

### **Identity Mappings**

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

### **Key pairs**

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

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

Additionally, Key Pairs are used by the **Mutual TLS Site Authenticator** to authenticate PingAccess to a target Site. When initiating communication, PingAccess presents the client certificate from a Key Pair to the Site during the mutual TLS transaction. The Site must be able to trust this certificate in order for authentication to succeed.

### **Listeners**

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

### **Load balancing strategies**

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

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

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

### **Policies**

The Policy Manager is a rich drag-and-drop interface where you can manage policies by creating Rules, building Rule Sets and Rule Set Groups, and applying them to Applications and Resources. Policies are rules, set of rules, or groups of rule sets 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, Rule Sets, or Rule Set Groups, PingAccess uses the information contained in the policy to decide whether the client can access the application resource and whether any additional actions need to take place prior to granting access. Rules can restrict access in a number of ways such as testing user attributes, time of day, request IP addresses, or OAuth access token scopes. Rules can also perform request processing such as modifying headers or rewriting URLs.

### **Proxies**

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

### **Rules, Rule sets, and Rule set groups**

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

### **Sites**

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

### **Site Authenticators**

When a client attempts to access a target Web Site, that Site 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.

### **Token provider**

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

### **Unknown resources**

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

**Virtual Hosts**

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

**Web sessions**

Web Sessions define the policy for Web application session creation, lifetime, timeouts, and their scope. 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.

# PingAccess user interface reference guide

---

## PingAccess User Interface Reference Guide

---

This guide provides a reference for configuration of PingAccess features and components. Use this guide in conjunction with PingAccess use case documentation for a comprehensive set of instructions so you can get the most from your PingAccess implementation.

For ease of use, navigation of this guide is modeled after the PingAccess user interface. To learn more about configuration options for a particular screen, navigate to its topic in the same manner that you navigate the PingAccess user interface.

To learn about PingAccess, including its features and functions, see the *PingAccess Overview*.

## Applications

---

Applications represent the protected web applications and APIs to which client requests are sent. Applications are composed of one or more resources, have a common Virtual Host and Context Root, and correspond to a single target site. Applications may 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.

There are 3 application types:

- Web
- API
- Web + API

Web + API applications allow administrators to configure both Web and API settings for an application. These applications are able to switch between web and API processing behaviors on the fly based on whether the inbound request contains a web session cookie (web) or an OAuth token (API). If the inbound request contains neither, PingAccess will fallback to the method you specify as the Fallback Type for the application.

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.

## Manage Applications

This document contains the steps required to:

- [Add an application](#) on page 19
- [Edit an application](#) on page 20
- [Delete an application](#) on page 20

### Add an application

1. Navigate to **Main > Applications**.
2. Click **Add Application**.
3. Complete the fields on the screen using [Manage Applications - Field Descriptions](#) on page 20 as a guide.
4. Click either **Save** or **Save & Go to Resources** when finished. The latter option allows you to configure additional application resources.



**Note:** When you save the application, PingAccess verifies the Redirect URI for the Application's virtual host is configured in the PingFederate. If PingAccess determines that the Redirect URI is not defined, you will receive the following warning:

```
Save succeeded. Unable to find a matching Redirect URI in the
PingFederate OAuth Client configuration for [<VHost>/pa/oidc/cb]
```

If you see this warning, ensure that there is a Redirect URI that matches configured. If you have a wildcard in your Virtual Host configuration, ensure the Redirect URI list includes the same wildcard host definition, otherwise you may have a configuration that is only valid in some circumstances.

This validation is performed if the **Application Type** is Web or Web + API, a **Web Session** is selected, and the PingFederate Administration connection is configured.

### Edit an application

1. Navigate to **Main > Applications**.
2. Expand the application on the **Properties** tab and click
3. Make the required changes using [Manage Applications - Field Descriptions](#) on page 20 as a guide.
4. Click **Save** or **Save and Go to Resources**.

### Delete an application

1. Navigate to **Main > Applications**.
2. Expand the application and click
3. Click **Delete** to confirm.

## Manage Applications - Field Descriptions

The following table describes the fields available for managing applications at **Main > Applications**.


Field	Required	Description
Name	Yes	A unique name for the application.
Description	No	An optional description for the application.
Context Root	Yes	<p>The context at which the application is accessed at the site.</p> <p> <b>Note:</b> This value must meet the following criteria:</p> <ul style="list-style-type: none"> <li>• It must start with /.</li> <li>• It can contain additional / path separators.</li> <li>• It must not end with /.</li> <li>• It must not contain wildcards or regular expression strings.</li> <li>• The combination of the <b>Virtual Host</b> and <b>Context Root</b> must be unique. The following <i>is</i> allowed and incoming requests will match the most specific path first: <ul style="list-style-type: none"> <li>• vhost1:443/App</li> <li>• vhost1:443/App/Subpath</li> </ul> </li> <li>• /pa is reserved for PingAccess and is not allowed as a <b>Context Root</b></li> </ul>
Case Sensitive Path	No	Indicates whether or not to make request URL path matching case sensitive.


Virtual host(s)	Yes	Specifies the virtual host for the application. Click <b>Create</b> to create a virtual host.
Application Type	Yes	Specifies the application type, either <code>Web</code> , <code>API</code> , or <code>Web + API</code> . <ul style="list-style-type: none"> <li>If the <b>Application Type</b> is <code>Web</code>, select the <b>Web Session</b> if the application is protected and, if applicable, the <b>Identity Mapping</b> for the application. Click <b>Create</b> to create a Web Session or Identity Mapping.</li> <li>If the <b>Application Type</b> is <code>API</code>, specify whether or not the application is <b>Protected</b> by an OAuth Authorization Server defined in <b>Settings &gt; System &gt; Token Provider</b> and, if applicable, select the <b>Identity Mapping</b> for the application. Click <b>Create</b> to create an Identity Mapping.</li> <li>If the <b>Application Type</b> is <code>Web + API</code>, select the <b>Fallback Type</b> to indicate the method to use when a request for a resource does not contain a web session cookie or OAuth token. Select the <b>Web Session</b> and, if applicable, the <b>Identity Mappings</b> to use for each type. In this configuration, the Web Session is required and the API is protected by default.</li> </ul>
Destination	Yes	Specifies the application destination type, either <code>Site</code> or <code>Agent</code> . <ul style="list-style-type: none"> <li>If the destination is a <code>Site</code>, select the <b>Site</b> requests are sent to when access is granted. If HTTPS is required to access this application, and at least one non-secure HTTP listening port is defined, select the <b>Require HTTPS</b> option. Click <b>Create</b> to create a Site.</li> <li>If the destination is an <code>Agent</code>, select the agent which intercepts and validates access requests for the Application. Click <b>Create</b> to create an Agent.</li> </ul>
Enabled	No	Select to enable the application and allow it to process requests.

## Manage Application Resources

This document allows you manage application resources. Application resources are components in an application that require a different level of security.

### To configure application resources:

1. Navigate to **Main > Applications**.
2. Expand an application and click .
3. Click the **Resources** tab.
4. Click **Add Resource**.

➔ **Info:** To edit a resource, expand the resource and click .

5. Enter a unique **Name** up to 64 characters, including special characters and spaces.
6. Enter a list of URL path prefixes (within the Context Root) that identify this resource.

➔ **Info:** The path prefix starts after the application context root and extends to the end of the URL. Prefixes must start with a slash (/) and may contain one or more wildcard characters (\*). No use of wildcards is assumed, thus there is a difference between `/app/` and `/app/*`.

⚠ **Important:** `/pa` and `/pa*` are reserved prefixes and cannot be used as a path prefix when the context root is `/`.

7. If the resource does not require authentication for access, enable the **Anonymous** option.



**Note:** This option is not available for unprotected applications. **Web** applications types are unprotected when they do not have an associated web session. **API** applications are unprotected when they are not configured to be protected by an authorization server.

8. If the application type is **API** or **Web + API**, 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.
9. Select the **Audit** checkbox to log information about the transaction to the audit store.
10. If the application type is **Web + API**, indicate whether the application resource should override the fallback type specified for the main application. If you select **Yes** for this option, select the method to be used for the application resource when a request does not contain a web session cookie or OAuth token.





**Important:** It is important to carefully consider your configuration when making this selection. Changing the Application Fallback Type can have unexpected effects on Resources that do not override the Fallback. For example, if you configure a Web + API Application with a Fallback Type of Web along with several Resources that do not override the Fallback Type, these Resources will emit a 401 response (rather than a 302 to PingFederate) if you later change the Fallback Type to API on the main application. The PingAccess runtime uses Fallback Type to determine which processing flow (Web or API) to use when the request does not contain a web session or an API OAuth Bearer token. When a request does not contain either of these authentication mechanisms, it will rely on this configuration to determine which processing flow to use.

11. Select the **Enabled** check box to enable the resource.
12. Click **Save**.

## Manage Application Policies

This document allows you to apply rules, rule sets, and rule set groups to applications and resources. Note that all changes made in this task are applied instantly.

1. Navigate to **Main > Applications**.
2. Expand an application in the list and click .
3. **(Optional)** To manage the policies for a resource, select the **Resources** tab and click .
4. Select the applicable tab. For **Web** or **API** applications, select the **Policy** tab. For **Web + API** applications, you can configure both **Web Policy** and **API Policy** on separate tabs, as required.
5. Using the radio selection, filter by *Rules*, *Rule Sets*, *Rule Set Groups*, or **Rule Type**.
6. To create a new rule, click **Create Rule**.
7. To apply a rule, rule set, or rule set group, drag a rule from **Available Rules** onto the policy bar.
8. Drag items around in the box to change the order in which they are evaluated at runtime.



**Note:** Rule ordering can affect PingAccess performance; if an access control rule is more likely to reject access, it should appear near the top of the list to reduce the amount of processing that occurs before that rule is applied. This can be more noticeable if, for example, access control policies are applied along with processing rules. Applying your access control policies first ensures that no processing happens on responses unless the user is determined to be allowed access.

9. Click - next to an item to remove it from an application or resource.

## Sites

---

Choose from one of the following sections:

- [Sites](#) on page 23
- [Site Authenticators](#) on page 24

## Sites

Sites are the target applications, endpoints, or APIs which PingAccess Gateway is protecting and to which authorized client requests are ultimately forwarded to.

### Manage Sites

This document contains the steps required to:

- [Add a site](#) on page 23
- [Edit a site](#) on page 23
- [Delete a site](#) on page 23

### Add a site

1. Navigate to **Main > Sites > Sites**.
2. Click **Add Site**.
3. Complete the fields on the screen using [Manage Sites - Field Descriptions](#) on page 23 as a guide.
4. To configure advanced settings, click **Show Advanced**.
5. Click **Save**.



**Note:** If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

### Edit a site

1. Navigate to **Main > Sites > Sites**.
2. Expand the site you want to edit, then click .
3. Make the required changes.
4. Click **Save**.



**Note:** If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

### Delete a site




1. Navigate to **Main > Sites > Sites**.
2. Expand the site you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm.

### Manage Sites - Field Descriptions

The following table describes the fields available for managing applications at **Main > Sites > Sites**.

Field	Required	Description
Name	Yes	Enter a unique <b>Site Name</b> up to 64 characters, including special characters and spaces.
Targets	Yes	Specify one or more <b>Targets</b> . The format for this is <code>hostname:port</code> . For example, <code>www.example.com:80</code> .
Secure	Yes	Select <b>Secure</b> if the Site is expecting HTTPS connections. If the site is configured for <b>Secure</b> connections, select a <b>Trusted Certificate Group</b> from the list, or select <b>Trust Any</b> to trust any certificate presented by the listed targets.



Site Authenticators	No	If the Site requires the use of site authenticators, select one or more authenticators from the list. Click <b>Create</b> to create a Site Authenticator. Click <b>x</b> to remove a Site Authenticator.
Use Target Host Header	No	Select the check box to have PingAccess modify the <code>Host</code> header for the Site's Target Host and Target Port rather than the Virtual Host configured in the application.   <b>Note:</b> When cleared, PingAccess makes no changes to the <code>Host</code> header. This is often required by target web servers to ensure they service the HTTP request with the correct internal virtual server definition.
Skip Hostname Verification	No	To skip hostname verification, select the checkbox.
Expected Certificate Hostname	No	If you have not selected to skip hostname verification, enter the name of the host expected in the certificate in the <b>Expected Certificate Hostname</b> field. This field is available only if the <b>Skip Hostname Verification</b> checkbox is not selected. If left blank, the certificates will be verified using the target hostnames.
Availability Profile	No	Select an <i>Availability profile</i> . Click <b>Create</b> to create an Availability Profile.
Load Balancing Strategy	No	Select a <i>Load balancing strategy</i> if the site contains more than one target. Click <b>Create</b> to create a Load Balancing Strategy.
Send Token	No	Leave the check box selected to include the PA Token in the request to the back-end Site. Clear the check box to remove the PA Token from the request.
Maximum Connections	No	Enter the maximum number of HTTP persistent connections you want PingAccess to have open and maintain for the Site. A value of <code>-1</code> indicates unlimited connections.
Maximum Websocket Connections	No	If the number of WebSocket connections needs to be limited, enter a value. The default of <code>-1</code> indicates no limit.
Use Proxy	No	Select if requests to the site should use a configured proxy.   <b>Note:</b> If the node is not configured with a proxy, requests are made directly to the site.   <b>Caution:</b> If your proxy uses availability handling to retry multiple targets in the event of a network problem, PingAccess should be configured to use only one target for the site. Unexpected behavior could occur if PingAccess and the proxy are both configured to perform availability handling.

## Site Authenticators

Site Authenticators define the authentication mechanism that target sites require to control access.

### Manage Site Authenticators

This document contains the steps required to:

- [Add a site authenticator](#) on page 25
- [Edit a site authenticator](#) on page 25
- [Delete a site authenticator](#) on page 25

## Add a site authenticator


1. Navigate to **Main > Sites > Site Authenticators**.
2. Click **Add Site Authenticator**.
3. Enter a unique **Name**.



**Note:** Special characters and spaces are allowed. This name appears in the **Site Authenticator** list on the **New Site** page.


4. Select the type of authentication from the list. Choose one of the following authentication types to continue.
  - [Basic authentication site authenticator](#) on page 25
  - [Mutual TLS site authenticator](#) on page 25
  - [Token mediator site authenticator](#) on page 26

## Edit a site authenticator

1. Navigate to **Main > Sites > Site Authenticators**.
2. Expand the site authenticator you want to edit, then click .
3. Make the required changes.
4. Click **Save**.

## Delete a site authenticator

➔ **Info:** If a site authenticator is associated with a site, you cannot delete it.

1. Navigate to **Main > Sites > Site Authenticators**.
2. Expand the site authenticator you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm.

## Basic authentication site authenticator

Use HTTP Basic authentication (username:password) to authenticate a client requesting access to a site that requires Basic authentication.

➔ **Info:** Obtain the username and password from your target Site provider.

Field	Description
<b>Username</b>	The username required for access to the protected Site.
<b>Password</b>	The password required for access to the protected Site.

## Mutual TLS site authenticator

Use Key Pairs to authenticate PingAccess to a target Site. When initiating communication, PingAccess presents the client certificate from a Key Pair to the Site during the mutual TLS transaction. The Site must be able to trust this certificate in order for authentication to succeed.



**Tip:** Several setup steps are required for PingAccess certificate management before configuring the Mutual TLS Site Authenticator.

Field	Description
<b>Key Pair</b>	The imported/generated key pair for client authentication. Select the key pair you want to use to authenticate PingAccess to the target Site. To create a key pair, see <a href="#">Import an existing key pair</a> on page 58 or <a href="#">Generate a new key pair</a> on page 59.

## 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
<b>Token Generator ID</b>	Defines the Instance Name of the Token Generator that you want to use. The Token Generator is configured in <b>PingFederate</b> (see PingFederate documentation).
<b>Logged In Cookie Name</b>	Defines the cookie name containing the token that the target site is expecting.
<b>Logged Off Cookie Name</b>	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.
<b>Logged Off Cookie Value</b>	Defines the value placed in the Logged Off Cookie to detect an invalid/expired WAM Token event.
<b>Source Token</b>	Defines the token type exchanged for a security token during identity mediation. Select <b>PA Cookie</b> for Web access or <b>OAuth Bearer Token</b> for API identity mediation.
<b>Send Cookies to Browser</b>	<p>Allows the Token Mediator to send the backend cookie defined in the <b>Logged In Cookie Name</b> field back to the browser if the protected application has updated it.</p> <p>If the set-cookie header isn't in the response from the protected site, and the Token Mediator Site Authenticator has a cached token for that session, the Token Mediator Site Authenticator will create a new set-cookie response header based on the <b>Cookie Domain</b>, <b>Cookie Max Age</b>, <b>HTTP-Only Cookie</b> and <b>Secure Cookie</b> fields in the UI.</p> <p>The admin now can direct the Token Mediator Site Authenticator to be an active player in returning cookies to the user's browser, even when the protected site isn't doing that.</p> <p>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.</p>
<b>Cookie Domain</b>	Enter the domain of the logged in cookie.
<b>Cookie Max Age</b>	Define the length of time in minutes, that you want the generated logged in cookie to be valid.
<b>HTTP-Only Cookie</b>	Define the logged in cookie as HTTP-Only. An HTTP-Only cookie is not accessible using non-HTTP methods, such as calls via JavaScript (for example, referencing document.cookie).
<b>Secure Cookie</b>	Indicate whether the generated logged in cookie must be sent using only HTTPS connections.

Field	Description
<b>Token Processor ID</b>	Defines the Instance Name of a Token Processor that you want to use. The Token Processor is configured in <b>PingFederate</b> (see PingFederate documentation). Specify this value if more than one instance of either the JWT Token Processor or the OAuth Bearer Access Token Processor is defined in PingFederate.

## Agents

Agents are web server plugins that are installed on the web server hosting the target application. Agents intercept client requests to protected applications and allow or deny the request to proceed by consulting the Policy Manager or using cached information. Agents communicate with the PingAccess Policy Server 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 Listener by performing the following steps:

1. 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. Navigate to **Networking > Listeners > HTTPS Listeners**.
3. In the **AGENT** dropdown, select the Key Pair you want to use, and then click **Save**.
4. Restart PingAccess.

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

## Manage Agents

This document contains the steps required to:

- [Add an agent](#) on page 27
- [Edit an agent](#) on page 28
- [Delete an agent](#) on page 28

### Add an agent

1. Navigate to **Main > Agents > Agents**.
2. Click **Add Agent**.
3. Complete the fields on the screen using [Manage Agents - Field Descriptions](#) on page 28 as a guide.
4. To configure advanced settings, click **Show Advanced**.
5. Click **Save & Download** to save the configuration and download `<agent-name>_agent.properties` for use with the PingAccess Agent.




**Note:** The **Shared Secret** is generated by PingAccess server and identified on this page with a timestamp. Existing secrets can be deleted by clicking the Remove button in the secret field. If an additional secret is needed, [edit](#) the agent and click **Save & Download** to generate and download a new Shared Secret.




**Note:** PingAccess can generate additional agent `agent.properties` files containing the specified information which can be used to configure the agent plugin. Existing configurations can also be re-downloaded if necessary.

## Edit an agent



1. Navigate to **Main > Agents > Agents**.
2. Expand an existing agent, then click .
3. Make the required changes.
4. To download the Shared Secret, click the **Download** button. To remove the Shared Secret, click the **Remove** button.
5. Click **Save & Download** to download the file.

## Delete an agent

1. Navigate to **Main > Agents > Agents**.
2. Expand an existing agent, then click .
3. When prompted, click **Delete** to confirm.

## Manage Agents - Field Descriptions

The following table describes the fields available for managing applications at **Main > Agents**.

Field	Required	Description
Name	Yes	Enter a unique alphanumeric <b>Name</b> for the agent, up to 64 characters.
Description	No	Enter an optional <b>Description</b> for the agent and its purpose.
PingAccess Host	Yes	In the <b>PingAccess Host</b> fields, enter the <b>Hostname</b> and <b>Port</b> of the PingAccess server where the agent should send requests.   <b>Info:</b> The PingAccess Hostname 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.
Failover Host	No	In the <b>Failover Host</b> fields, enter the <b>Hostname</b> and <b>Port</b> of the PingAccess server where the agent should send requests in the event of a failover from the PingAccess Host.   <b>Tip:</b> Additional failover hosts may be added via API. For more information, see the <i>PingAccess API Management Guide</i> .
Agent Trusted Certificate	Yes	Specify the <b>Agent Trusted Certificate</b> to export in the agent properties file. The agent uses the selected certificate to communicate with the PingAccess engine via SSL/TLS. PingAccess gathers these certificates from imported certificates. If the appropriate certificate is not available, it needs to be <i>imported into the system</i> .
Override Request IP Source Configuration	No	If required, select <b>Yes</b> to <b>Override Request IP Source Configuration</b> and enable additional controls that configure the agent to use different IP Source information.

		<ul style="list-style-type: none"> <li>• Enter the <b>Header Names</b> used to identify the source IP address.</li> <li>• If more than one value is included in the <b>Header Names</b> field, use <b>List Value Location</b> to specify whether the first value or the last value in the list is used as the source address. The default value is <b>Last</b>.</li> <li>• Select <b>Fall Back to Last Hop IP</b> to use the last hop IP address as the source address when none of the listed <b>Header Names</b> are found. When this option is not selected, if none of the listed <b>Header Names</b> are found, access is denied and a <b>Forbidden</b> result is returned.</li> </ul>
Override Unknown Resource Configuration	No	If required, select <b>Yes</b> to <b>Override Unknown Resource Configuration</b> to specify how requests for unknown resources should be handled. This mode is optional. If not set, the default agent mode will be used. Select a <b>Mode</b> to specify how requests for unknown resources should be handled, either <b>Deny</b> or <b>Pass-Through</b> .
Max Retries	<b>Yes</b>	Enter the <b>Max Retries</b> before considering a PingAccess server unavailable.
Failed Retry Timeout	<b>Yes</b>	Enter, in seconds, the <b>Failed Retry Timeout</b> before retrying a failed PingAccess server.

## Policies

The Policy Manager is a rich drag-and-drop interface where you can manage policies by creating Rules, building Rule Sets and Rule Set Groups, and applying them to Applications and Resources. Policies are rules, set of rules, or groups of rule sets 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, Rule Sets, or Rule Set Groups, PingAccess uses the information contained in the policy to decide whether the client can access the application resource and whether any additional actions need to take place prior to granting access. Access control 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. Processing rules can perform request processing such as modifying headers or rewriting URLs.



**Note:** Rule requests are processed in the order in which they appear in the Administration interface. Rule responses are processed in reverse order.



**Tip:** 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.



**Info:** Processing rules cannot be used with Agents.

## Manage Rules

This document contains the steps required to:


- [Create a rule](#) on page 29
- [Edit a rule](#) on page 30
- [Delete a rule](#) on page 30
- [Configure advanced fields for rules](#) on page 30

### Create a rule


1. Navigate to **Main > Policies**.

2. Click + to the right of the **Rules** heading.
3. Enter a unique **Name**. The name can be up to 64 characters long. Special characters and spaces are allowed.
4. Select the rule **Type** to use. Fields associated with that Rule type are shown.
5. Enter the required information for the type of rule you are creating. For additional information, see the appropriate subtopic.
6. Click **Save**.

### Edit a rule

1. Navigate to **Main > Policies**.
2. Expand the rule you want to edit and click .
3. Make the required changes.
4. Click **Save**.

### Delete a rule

1. Navigate to **Main > Policies**.
2. Expand the rule you want to delete and click .
3. When prompted, click **Remove** to delete the rule.

### Configure 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. Use the following fields, available in the **Advanced** section of a rule, to configure the error handling.

Field	Description
<b>Error Response Code</b>	The HTTP status response code you want to send if Rule evaluation fails. For example, 403.
<b>Error Response Status Message</b>	The HTTP status response message you want to return if Rule evaluation fails. For example, Forbidden.
<b>Error Response Template File</b>	The HTML template page for customizing the error message that displays if Rule evaluation fails. This template file is located in the <code>PA_HOME/conf/template/</code> directory.
<b>Error Response Content Type</b>	The type of content for the error response so the client can properly display the response.


### Authentication requirements rule

#### Prerequisites:

- A PingFederate configuration that uses the Requested AuthN Context Authentication Selector.
- A configured authentication list.

The Authentication Requirements Rule is a PingAccess access control rule used to limit access to resources or applications protected by PingAccess based on the ACR values returned by the PingFederate Requested AuthN Context Authentication Selector. This allows authentication requirements to be applied when a policy decision is being made by the PingAccess Engine, allowing an entire application or individual resources to require a particular authentication type.

Use of this rule also allows for configurations that require more secure authentication methods. For example, a web site might allow a user to authenticate and view personal data using only a username and password, but editing their personal data could require an additional PingID verification step. When used in this manner, an additional step-up authentication event is automatically triggered.

-  **Tip:** When used in a rule set with `Any` criteria, this rule should be positioned first in the list to ensure step-up authentication is triggered upon rule set criteria failure.

#### To configure an Authentication Requirements rule:

1. Select an [Authentication Requirements List](#).
2. Select a **Minimum Authentication Requirement**.


 **Note:** The possible values for the **Minimum Authentication Requirement** are derived from the selected Authentication Requirements list.

#### Cross-origin Request rule



Cross-Origin Resource Sharing (CORS) provides a means for a web server to grant access to restricted resources (fonts, JavaScript, images, etc.) to an application served by another domain without granting access to those resources beyond a list of predefined origin servers.

Before a CORS request is sent, the originating web server generally sends a "pre-flight" `OPTIONS` request if the request from the client includes credentials. This pre-flight request is used to determine if the target server permits CORS requests to be processed from the originating web server.

PingAccess can be used to evaluate the headers provided in a CORS request to grant or deny access to resources.

 **Note:** In addition to allowing PingAccess to evaluate the CORS request, you can also allow the request to be handled by the protected application, and let PingAccess be excluded from the process of evaluating the access request, if the target application type is `API`. In order to do this with a resource path that is protected by PingAccess and requires user authentication, configure a second resource with the same path prefix, but set the **Methods** field to `OPTIONS` and the **Anonymous** option needs to be cleared. This configuration allows the API request being made to be handled anonymously.

#### To Configure a Cross-Origin Request Rule

1. Enter one or more **Allowed Origins** values, clicking **+** to add additional values.
  -  **Important:** While it is allowed, we recommend against using a value of `*` in this field. While this is a valid configuration, it is considered to be an insecure practice.
2. If additional options need to be configured, click **Show Advanced**.
3. To permit user credentials to be used in determining access, enable **Allow Credentials**.
4. To modify the **Allowed Request Headers** values, use the following options:
  - To add a new header, click **New Value**.
  - To edit an existing header, click the field and make your changes.
  - To remove an existing header, click .

The default headers are `Authorization`, `Content-Type`, and `Accept`.

5. To make specific response headers available to the client that originated the cross-origin request, enter the headers in the **Exposed Response Headers** field. Click **New Value** to add additional headers to the list.
6. To define the request methods allowed in cross-origin requests, select the desired overrides in the **Overridden Request Methods** field.
7. To modify the amount of time the pre-flight `OPTIONS` request is cached, enter the maximum age (in seconds) desired in the **OPTIONS Cache Max Age** field. The default is 600 seconds.
8. Click **Save**.

#### 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.

➔ **Info:** This example assumes that a virtual host, a site, and an application are already configured.

### Create a Rewrite Content rule

A Rewrite Content Rule alters content in the HTTP Response body.

- In the Response Content-Types field, you define a response type of `text/html`.
- In the Find and Replace criteria, you specify `<a href="https://server1.internalsite.com/content/">` and `<a href="https://publicsite.com/importantstuff/">`.
- Add the Rule to the application. A query to a page with links in it that point to `https://server1.internalsite.com/content/` now point to `https://publicsite.com/importantstuff/`.

### Create a Rewrite Cookie Domain rule

A Rewrite Cookie Domain Rule allows the rewriting of the Domain field on cookies when they are set by the 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/`.

### Create a Rewrite Response Header rule

A Rewrite Response Header Rule alters the response header used in the 302 Redirect.

- In the Server-Facing URI field, you enter `https://server1.internalsite.com/content/`.
- In the Public Path field, you enter `/importantstuff/`.
- Add the Rule to the application. A query resulting in a response containing a 302 Redirect to `https://server1.internalsite.com/content/` is rewritten to `https://publicsite.com/importantstuff/`.

- ➔ **Info:** This also works for relative redirects: `/content/` is rewritten to `/importantstuff/`. It also works for the path beneath the one defined in the URI: `/content/news/index.html` is rewritten to `importantstuff/news/index.htm`.

### Create a Rewrite URL rule

A Rewrite URL Rule alters the Request URI.

- In the Map From field, you enter `^/importantstuff/(.*)` as the regex of the URL's path and query you want to match.
- In the Map To field, you enter `/content/$1`.
- Add the Rule to the application. A query to `https://publicsite.com/importantstuff/` results in PingAccess routing that query to `https://server1.internalsite.com/content/`.

### 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.

- ⚠ **Important:** While adding a Content Rewrite Rule in PingAccess 4.2, you cannot add a new row via the UI. Clicking the link to add a new row may cause page controls to become unresponsive. To workaround this issue, use the API to create the rule or create multiple Content Rewrite Rules in the UI and combine them in a rule set.

- ➔ **Info:** Extensive use of Rewrite Content Rules may have significant performance implications.

This rule supports content that is either chunked or streamed from the target server. When sent to the client, the content is always chunked.

#### To configure a Content Rewrite rule:

1. Enter a name for 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.
  - ➔ **Info:** 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.
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.
  - ⚠ **Important:** Changes can affect CSS, Javascript, and other text-based elements served to the client. Be sure to properly craft the regular expression to avoid modifying content that wasn't intended.
5. If necessary, increase the size of the buffer used to perform the replace operation by clicking **Show Advanced** and entering a value in **Maximum Buffer Size**.
  - 📄 **Note:** Replacement values cannot be larger than the buffer size. The minimum buffer size that can be specified is 1024 bytes; there is no maximum value.
6. If the protected application does not return a `Content-Type` header, select **Missing Content-Type Allowed**.
7. If **Missing Content-Type Allowed** is enabled, you must specify the encoding the application returns in the **Missing Content-Type Charset** field. For example, this field could contain `UTF-8`. A list of valid values is available in this [Oracle Java 8 SE Technical Note](#).

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 <b>bold</b> .	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`).

➔ **Info:** 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. If the target host needs to be explicitly defined, clear the **Any Site Target Host** checkbox.

When the **Any Site Target Host** checkbox is enabled, PingAccess will rewrite the cookie domain if it is set to the domain defined in a site's target host list.

4. **Conditional:** If **Any Site Target Host** is cleared, enter the domain name to used by the back-end site in the **Server-Facing Cookie Domain** field.

5. In the **Public-Facing Cookie Domain** field, enter the domain name you want to display in the response from PingAccess.
6. Click **Save**.

### 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** field, enter the path name where the cookie is valid for the back-end Site.
4. In the **Public-Facing Cookie Path** field, enter the path name you want to display in the response from PingAccess.
5. Click **Save**.

### 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. If the target host needs to be explicitly defined, clear the **Any Site Target Host** checkbox.  
When the **Any Site Target Host** checkbox is enabled, PingAccess will rewrite the response header URI if it contains a domain defined in a site's target host list.
4. If **Any Site Target Host** is cleared, enter the domain name to be used by the back-end site in the **Server-Facing URI** field.
5. 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 `/`
6. Click **Save**.

### 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
<code>/bank/</code>	<code>/application/</code>	<code>/bank/content.html</code>	<code>/application/content.html</code>
<code>^/bank/(.*)</code>	<code>/application/\$1</code>	<code>/bank/content.html</code>	<code>/application/content.html</code>
<code>/bank/index.html</code>	<code>/application/index.jsp</code>	<code>/bank/index.html</code>	<code>/application/index.jsp</code>

Map from value	Map to value	Example request	Rewrite by PingAccess
/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**.

### Groovy script rule

Groovy scripts provide advanced Rule logic that extends PingAccess Rule development beyond the capabilities of the packaged [Policy Manager](#) rules.



**Note:** Through Groovy scripts, PingAccess administrators can perform sensitive operations that could affect system behavior and security. Please note that since the regular Groovy Rule and the OAuth Groovy Rule differ in the scope of their functionality, the relevant rules are tagged for Web App or for API, respectively, in the rules dropdown menu.

See [Advanced Fields](#) for information about error handling.

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 contain: `hasScopes("access", "portfolio")`
2. If you need to configure error handling parameters, click **Show Advanced** to provide those configuration options.
3. Click **Save** when you finish.

### HTTP Request Header rule

Examines a request and determines whether to grant access to a requested resource based on a match found in one of the specified headers in the HTTP request.



**Note:** See [Advanced Fields](#) for information about error handling.

If more than one **Field** and **Value** pair is listed, then all conditions must match in order for the rule to succeed.

### To configure an HTTP Request Header rule:

1. In the **Field** column, enter a **Header** name you want to match in order to grant or not grant the client access.
2. Enter the **Value** for the Header you want to match in order to grant or not grant the client access. The wildcard (\*) character is supported.
  - Tip:** If you want to match on the `Host` header, include both the host and port as the **Value**, or add a wildcard after the hostname (`host*` or `host:*`) to match what is in the HTTP request.
3. Select **Case Sensitive** if the values should be matched only if the value case is an exact match.
4. Select **Negate** if access should be denied when a match is found.
  - Info:** Ensure that the attribute name entered in the **Field** field is spelled correctly and exists. If you enter an attribute that does not exist and you select **Negate**, the rule will always succeed. The **Negate** control applies to the entire set of conditions specified, and passes the rule if any condition is not met.

5. If additional **Header** pairs are needed, click **Add Row** to add an additional row, then repeat steps 1-4.
6. If you need to configure error handling parameters, click **Show Advanced** to provide those configuration options.
7. Click **Save**.

### HTTP Request Parameter rule

Examines a request and determines whether to grant access to a requested resource based on a match found in specified form parameters of the HTTP request.

This rule determines if the parameters are passed as part of the URL query string parameters or as part of a request body submitted using an HTTP PUT or POST method. If the request is a POST request, the `content-type` must be set to `application/x-www-form-urlencoded` to process the field names in the request.

If this rule is applied to an Agent configuration, only URL query string parameters are compared, because the Agent does not receive the request body for processing.

If more than one **Field** and **Value** pair is listed, then all conditions must match in order for the rule to succeed.



**Note:** See [Advanced Fields](#) for information about error handling.

#### To configure an HTTP Request Parameter rule:

1. In the **Field** column, enter a **Parameter** name you want to match in order to grant or not grant the client access.
2. Enter the **Value** for the field you want to match in order to grant or deny the client access. The wildcard (\*) character is supported.



**Note:** Values entered here will be URL-encoded prior to the comparison. For example, if the value specified in the **Value** field is `v1 v2`, when the engine performs the comparison, this value will be converted to `v1%20v2` before the search is performed.

3. Select **Case Sensitive** if the values should be matched if the value case is an exact match.
4. Select **Negate** if when a match is found, access is not allowed.
  - ➔ **Info:** Ensure that the field name you enter is spelled correctly and exists. If you enter a field name that does not exist and you select **Negate**, the rule will always succeed. The **Negate** control applies to the entire set of conditions specified, and passes the rule if any condition is not met.
5. If additional **Parameters** pairs are needed, click **Add Row** to add an additional row, then repeat steps 1-4.
6. If you need to configure error handling parameters, click **Show Advanced** to provide those configuration options.
7. Click **Save**.

### 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. **Conditional:** If you wish to override source address handling defined in the **HTTP Requests** configuration, click **Show Advanced** and perform the following steps:
  - a) Select the **Override Request IP Source Configuration** option.
  - b) Enter the **Headers** used to define the Source IP address to use.
  - c) Select the **Header Value Location** to use when multiple addresses are present in the specified header. Valid values are `Last` (the default) and `First`.
  - d) Select the **Fall Back to Last Hop IP** option to determine if, when the specified **Headers** are not present, PingAccess should return a `Forbidden` result or if it should use the address of the previous hop as the source to make policy decisions.

4. Additional advanced fields for handling error responses may also be defined here. See [Advanced Fields](#) for more information about these fields.
5. Click **Save**.

### 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. Select an **Attribute Name** you want to match to an attribute associated with an OAuth access token.
2. In the **Attribute Value** box, enter the value to match.
  -  **Note:** The attribute values come from the contract in your OAuth access token manager in PingFederate. See [Defining the Access Token Attribute Contract](#) for more information.
3. Add additional rows of attribute name/value pairs as needed.
  -  **Note:** If multiple rows are included here, all conditions must match in order for the rule to match.
4. Select **Negate** if when a match is found, access is not allowed.
  -  **Info:** Verify what you enter for the attribute. If you enter an attribute that does not exist (for example, misspell it) and you select **Negate**, the rule will always succeed.
5. Click **Save**.

### 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 for API applications.

-  **Info:** Please note that since the regular Groovy Rule and the OAuth Groovy Rule differ in the scope of their functionality, the relevant rules are tagged for Web App or for API, respectively, in the UI's rules dropdown menu.


#### To configure an OAuth Groovy Script rule:

1. Enter the **Groovy Script** to use for Rule evaluation.
2. **Conditional:** If you need to specify a custom Error Response Template File, click **Show Advanced** and fill in the **Error Response Template File** and **Error Response Content Type** fields.
3. Click **Save**.

See [Advanced Fields](#) for information about error handling.

### Configure an 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`.

1. Select the **Scope** you want to match to values returned from the access token.
  -  **Info:** This is one scope requirement in the set of scopes associated with the access token.
2. Select **Negate** if when a match is found, access is not allowed.
3. Optional: If you want to use a customized error response template, click **Show Advanced** and provide the **Error Response Template File** and **Error Response Content Type** values.
4. Click **Save**.

### OAuth Token Cache Time To Live rule

Use this rule to configure the caching behavior for PingFederate-issued tokens.

**To configure an OAuth Token Cache Time To Live rule:**

1. Enter a **Name** for the rule.
2. Specify whether you want to cache PingFederate-issued access tokens.
3. Specify the number of seconds to cache the access token. This value should be less than the PingFederate Token Lifetime. A value of -1 means no limit.

See [Advanced Fields](#) for information about error handling.

**Rate Limiting rule**

The Rate Limiting Rule allows the administrator to define access to limit a client from overloading the server with too many requests in a specified period of time. The implementation of this rule uses a *Token Bucket* in order to control the number of incoming requests.

The way this works is that the configuration defines a number of requests and an interval that must elapse between requests. The allowed number of requests within the time window is controlled by the **Max Burst Requests** setting visible when you click **Show Advanced**. For example, if the **Max Burst Requests** value is 1, two requests are allowed in the request interval — one normal request, and one burst request.

The number of allowed requests is incremented by one at the end of each **Request Interval** if a request was not received. This continues until the number of allowed requests equals the value defined by the **Max Burst Requests** setting.



**Note:** Using the Rate Limiting Rule in a clustered PingAccess environment may impose stricter clock synchronization requirements for requests processed by multiple engine nodes. Alternatively, a load balancer sitting in front of a PingAccess cluster can be configured to stick the session to a specific engine, thus ensuring that the rate limiting rule is applied by a single PingAccess engine node.

1. Select a **Policy Granularity**, as defined in the following table:

<b>Policy Granularity</b>	<b>Definition</b>
Resource	Restricts the rate of requests based on the resource requested.
Identity	Restricts the rate of requests to the identity associated with the current authentication token (a PA Cookie or an OAuth token). This is the default value.
IP	Restricts the number of requests based on the source IP address. The IP address used to apply this policy comes from the HTTP Requests IP Source configuration options or options that override that configuration, if those options are configured.
OAuth Client	Restricts the number of requests to all OAuth tokens obtained by a specific <b>Client ID</b> .

2. Enter, in milliseconds, a **Request Interval**.
3. If more than 1 request should be allowed a request interval, expand the **Advanced** section of the page, and enter the number of requests to allow in the **Max Burst Requests** field.



**Note:** PingAccess increases the number of available requests only after a request interval that serves no requests to the client. As a result, in the period following a cycle where the remaining allowed burst requests is reduced to 0, no burst requests would be allowed, regardless of this setting.

4. If PingAccess should reply to the client with a **Retry-After** header instructing the client to wait for a period of time, select the **Set Retry-After Header** option.
5. To customize the error response sent to the client, click **Show Advanced** and modify the **Error Response Code**, **Error Response Status Message**, **Error Response Template File**, and **Error Response Content Type** fields. See [Advanced Fields](#) for more information about these fields.



## 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** field. For example: 8:00 AM.
2. Enter the ending time for the time frame in the **End Time** field. For example: 5:00 PM.
  - ➔ **Info:** 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.

## 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. Select 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.
  - ➔ **Info:** PA Token attributes are obtained from the PingFederate OpenID Connect Policy attribute contract (see [Configuring OpenID Connect Policies](#)).
3. Click **Add Row** to add more attributes, or click  to remove a row.
4. Select **Negate** to disallow access when a match is found.
  - ➔ **Info:** Ensure the attribute name is spelled correctly and exists. If you enter an attribute that does not exist and you select **Negate**, the rule will always succeed.
5. Click **Save**.
  - ➔ **Info:** To use this Rule, we recommend that you leave the **Request Profile** checkbox selected, 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.

## WebSocket Handshake rule

Allows the administrator to define the domains that are allowed to open a cross-origin WebSocket to the application or resource. Additionally, allowed WebSocket subprotocols and extensions may be defined, providing more fine-grained control over how the application behaves.

### To configure a WebSocket Handshake rule:

1. Enter one or more **Allowed Origins**. If no origins are defined, all cross-origin WebSocket requests are denied.
  - ⚠ **Important:** While it is allowed, we recommend against using a value of `*` in this field. While this is a valid configuration, it is considered to be an insecure practice.
2. Modify the list of **Allowed Subprotocols**. Subprotocols are defined in the `Sec-WebSocket-Protocol` handshake header. The default value of `*` allows all subprotocols.
3. Modify the list of **Allowed Extensions**. WebSocket extensions are defined in the `Sec-WebSocket-Extensions` handshake header. The default value of `*` allows all extensions.

See [Advanced Fields](#) for information about error handling.

## Manage Rule Sets

This document contains the steps required to:

- [Add a rule set](#) on page 41
- [Edit a rule set](#) on page 41
- [Delete a rule set](#) on page 41

### Add a rule set

1. Navigate to **Main > Policies**.
2. Click + to the right of the **Rule Sets and Groups** heading.
3. Drag a Rule from the **Rules** column onto the box that appears.
4. Enter a name for the Rule Set in the box that appears. Special characters and spaces are allowed.
5. 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.

➔ **Info:** When **Success Criteria** is set to **Any**, the first rule establishes the error handling and is flagged with a tooltip that displays a message indicating that. When **Success Criteria** is set to **All**, the first rule in the set that fails establishes the error handling.

📄 **Note:** When **Success Criteria** is set to **Any**, PingAccess flags Processing Rules in a Rule Set with tooltip that warns that if the first rule in the list succeeds, additional rules will not be processed. This is considered a misconfiguration because in an **Any** Rule Set, 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."

6. Click **Save** to save the Rule Set.
7. Add more Rules.

### Edit a rule set

1. Navigate to **Main > Policies > Rule Sets**.
2. Expand the rule set you want to edit. Choose from:
  - Drag a rule within a rule set up or down to re-order the rules.
  - Click - to the right of any rule you want to remove from a rule set.
  - Click 🗑️ to delete the rule set.
  - Click ✎️ to edit the rule set **Name** or **Success Criteria**.
3. Click **Save** after changing these values.

### Delete a rule set

📄 **Note:** A rule set that is associated with an application or resource cannot be deleted

1. Navigate to **Main > Policies**.
2. Expand the rule set you want to delete.
3. Click 🗑️ .
4. When prompted, click **Remove** to confirm the delete request.

## Manage Rule Set Groups



This document contains the steps required to:

- [Add a rule set group](#) on page 42
- [Edit a rule set group](#) on page 42
- [Delete a rule set group](#) on page 42

## Add a rule set group


1. Navigate to **Main > Policies**.
2. Click + to the right of the **Rule Sets and Groups** heading.
3. Drag a Rule Set from the **Rule Sets and Groups** column onto the box that appears.
4. Enter a name for the Rule Set Group in the box that appears. Special characters and spaces are allowed.
5. 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.
  - ➔ **Info:** When **Success Criteria** is set to **Any**, the first rule set establishes the error handling and is flagged with a tooltip that displays a message indicating that. When **Success Criteria** is set to **All**, the first rule in the set that fails establishes the error handling.
  - 📄 **Note:** When **Success Criteria** is set to **Any**, PingAccess flags Processing Rules in a Rule Set Group with tooltip that warns that if the first rule in the list succeeds, additional rules will not be processed. This is considered a misconfiguration because in an **Any** Rule Set Group, the first Processing Rule should succeed, causing all other rules in the set to not be evaluated. If you want to use Processing Rules on protected applications as well as handle access control decisions using **Any** criteria, assign Processing Rules directly to the application or create a separate Rule Set Group for the Processing Rules using the **All** criteria."
6. Click **Save** to save the Rule Set Group.
7. Add more rule sets or rule set groups.
8. Configure rule set hierarchy by dragging rule sets in the rule set group. Processing occurs from top to bottom. The configuration is automatically saved.
- 9.

## Edit a rule set group

1. Navigate to **Main > Policies > Rule Sets and Groups**.
2. Expand the rule set group you want to edit. Chose from:
  - Drag a rule set within a rule set group up or down to re-order the rules.
  - Click - to the right of any rule set you want to remove from a rule set group.
  - Click  to delete the rule set group.
  - Click  to edit the rule set group **Name** or **Success Criteria**.
3. Click **Save** after changing these values.

## Delete a rule set group

This task allows you to remove a rule set group. You cannot remove a rule set group if it is associated with an application or resource. To remove a rule set group, you must first remove this association.

1. Navigate to **Main > Policies**.
2. Expand the rule set group you want to delete.
3. Click .
4. When prompted, click **Remove** to confirm.

## Access

---

Choose from one of the following sections:

- [Authentication Requirements](#) on page 43
- [Identity Mappings](#) on page 43
- [Virtual Hosts](#) on page 46
- [Web Sessions](#) on page 47

- [Unknown Resources](#) on page 51

## 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.


### Configure an authentication requirements list

1. Navigate to **Settings > Access > Authentication Requirements**, and click **Add Authentication Requirement**.
2. Enter a unique name for the Authentication Requirements list.
3. Enter an authentication method. For example, `cert` or `password`.
  - ➔ **Info:** 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](#)).
4. Click **Add Authentication Requirement** to add additional authentication requirements.
5. Click **Save**.

### Edit an authentication requirements list

1. Navigate to **Settings > Access > Authentication Requirements**.
2. Expand list you want to edit, then click .
3. Make your changes.
4. Click **Save**.

### Delete an authentication requirements list

1. Navigate to **Settings > Access > Authentication Requirements**.
2. Expand the authentication requirement you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm.

## Identity Mappings

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


### Configure Auth Token Management

Auth Token Management settings define the issuer and signing configuration used by JWT Identity Mappings.

1. Navigate to **Settings > Access > Identity Mappings**.
2. Specify the **Key Roll Interval (h)** to indicate how often (in hours) you want to roll the keys. Key rollover updates keys at regular intervals to ensure the security of the signed auth tokens.
3. Specify a published, unique **Issuer** identifier to be used with auth tokens. For example, set the issuer to a value that more closely represents your company. PingAccess inserts this value as the iss claim within the auth tokens.
4. Specify the **Signing Algorithm** used to protect the integrity of the auth tokens (the default is ECDSA using P-256 Curve).
5. Click **Save**.

### Create a new identity mapping

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

-  **Tip:** When configuring identity mappings, the dot notation is supported so that session token structure can be maintained. For example, if the session token contains the following entry:

```
{
  "address": {
    "line1": "123 Any St",
    "line2": "Apt 123",
    "city": "Anytown",
    "state": "CO",
    "zip": "12345"
  }
}
```

You can define an identity mapping using the following entries to maintain the structure of the target JWT:

User Attribute Name	JWT Claim Name
address.line1	address.line1
address.line2	address.line2
address.city	address.city
address.state	address.state
address.zip	address.zip

1. Navigate to **Settings > Access > Identity Mappings**.
2. Click **Add Identity Mapping**.
3. Enter a **Name** for the mapping.
4. Select the identity mapping **Type** to use. Fields associated with the type are shown.
5. Enter the required information for the type of identity mapping you are creating.
6. Click **Save**.

### Header identity mapping

Header identity mappings make user attributes or client certificates available as HTTP request headers to back-end sites that use them for authentication. A single Header identity mapping can expose a number of attribute values or a certificate chain up to 3 levels deep. Header identity mappings are assigned to applications.

Use the following steps to configure Header identity mapping.



1. **(Optional)** In the **Attribute to Header Mapping** section, enter the name of the attribute to retrieve from the user web session in the **Attribute Name** field. For example, sub.
2. Enter the name of the HTTP requests header to contain the attribute value in the **Header Name** field. 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`.

3. Select which attribute is used as the **Subject**.
4. **(Optional)** In the `Certificate to Header Mapping` section, enter the header name included in a PEM-encoded client certificate. The row position correlates to the index in the client certificate chain. For example, the first row always maps to the leaf certificate. If you are using a certificate chain, click **Add Row** to add another row.
5. Click **Save**.

### JWT identity mapping

A JWT Identity Mapping makes user attributes available in a signed JWT (JSON Web Token) sent to the backend site in a header.


-  **Tip:** The JWT issuer and signing configuration is defined in [Configure Auth Token Management](#) on page 43.
-  **Tip:** PingAccess engines provide a JWKS (JSON Web Key Set) endpoint at `/pa/authtoken/JWKS` that can be used by backend sites to validate the signature of the JWT.

Use the following steps to configure JWT identity mapping.

1. Enter the name of the header to use when sending the signed JWT to the target application in the **Header Name** field. 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`.
2. Enter the audience to be set as the `aud` claim in the signed JWT in the **Audience** field.
3. Enter the name of the attribute to retrieve from the user web session in the **User Attribute Name** field. For example, `sub`.
4. Enter the name of the JWT claim to contain the attribute value in the **JWT Claim Name** field.
5. Select which attribute is used as the **Subject**.
6. **(Optional)** In the **Certificate to JWT Claim Mapping** field, enter the name of the JWT claim to contain the client certificate chain array.
7. If you are performing Certificate to JWT Claim Mapping, in the **Client Certificate Chain Max Depth** field, specify the maximum number of certificates from the client certificate chain included in the JWT claim array.
8. Optional: Click **Show Advanced** and select **Cache JWT**. When enabled, a cached signed JWT will be used for repeated requests for a given user. If user attributes change or the key used to sign the JWT changes, a new JWT will be created even if JWT caching is enabled.
9. Click **Save**.

### Edit an identity mapping

Follow the steps below to edit an existing identity mapping.

1. Navigate to **Settings > Access > Identity Mappings**.
2. Expand the identity mapping you want to edit.
3. Click .
4. Make the required changes.
5. Click **Save**.

### Delete an identity mapping

Follow the steps below to delete an existing identity mapping.

1. Navigate to **Settings > Access > Identity Mappings**.
2. Expand the identity mapping you want to edit.
3. Click the **Delete** icon.

4. In the confirmation window that appears, click **Delete**.

## Virtual Hosts

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

A wildcard (\*) can be used either to define either any host (\* : 443, for example) or any host within a domain (\*.example.com, for example).



**Note:** 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.

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

### Create a new virtual host

Use this task to create a new virtual host.

1. Navigate to **Settings > Access > Virtual Hosts**.
2. Click **Add Virtual Host**.
3. 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. A wildcard host may also be specified (e.g. \*.example.com).
4. Enter the **Port** number for the Virtual Host. For example: 1234.
5. Enter the **Agent Resource Cache TTL** indicating the number of seconds the Agent can cache resources for this application. Only applies to destination of type Agent.
6. Click **Save**.

### Configure virtual host trusted certificate groups

This procedure provides the steps to configure virtual host trusted certificate groups that are used to implement client certificate authentication. Assigning a trusted certificate group to a virtual host provides a mechanism to authenticate using client certificates during any request to sites using the specified virtual host.



**Note:** Trusted certificate groups are applied at the host name level and are independent of the configured port. This means that a mapping to a virtual host of \*.example.com will apply to requests received on virtual hosts \*.example.com:3000 and \*.example.com:443.

1. Navigate to **Settings > Access > Virtual Hosts**. Virtual Hosts that have certificate authentication configured will display the message *Client Certificate Authentication* in the associated bar.
2. Select and expand the Virtual Host you want to modify.
3. In the **Properties** list, click the to select a **Trusted Certificate Group**.
4. Select the appropriate **Trusted Certificate Group**.
5. Click **Save**.
6. Add the following two *Groovy script rules* to force validation of the SNI and client certificate chain. *Apply these rules* to applications using this virtual host.

Validate SNI


```
if(exc?.getSslData()?.getSniServerNames()?.isEmpty())
{
    fail();
}
else
{
    pass();
}
```

Validate client certificate chain

```
if(exc?.getSslData()?.getClientCertificateChain()?.isEmpty())
{
    fail();
}
else
{
    pass();
}
```


### Edit a virtual host

Use this task to edit a virtual host.

1. Navigate to **Settings > Access > Virtual Hosts**.
2. Expand the virtual host you want to edit and click .
3. Make the required changes.
4. Click **Save**.

### Delete a virtual host

Use this task to delete a virtual host.

1. Navigate to **Settings > Access > Virtual Hosts**.
2. Expand the virtual host you want to edit and click .
3. Click **Delete** to confirm.

## 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 the following tasks to configure secure Web Sessions for use with specific applications and to configure global Web Session settings.

### Application scoped Web Sessions

PingAccess Tokens can be configured to have their Web Sessions scoped to a specific application. This improves the security model of the session by preventing unrelated applications from impersonating the end user.

Several controls exist to scope the 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.

- ➔ **Info:** 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.

## Configure Web Session Management settings

Use this task to configure Web Session Management settings.

1. Navigate to **Settings > Access > Web Sessions**.
2. In the **Web Session Management** section, enter the **Key Roll Interval**, in hours, to specify how often you want to roll the keys (the default is 24 hours). Key rollover updates keys at regular intervals to ensure the security of signed and encrypted PA Tokens.
3. In the **Issuer** field, enter the published, unique identifier to be used with the Web session (The default is PingAccess). For example, set the issuer to a value that more closely represents your company. PingAccess inserts this value as the `iss` claim within the PA Token.
4. Select the **Signing Algorithm** used to protect the integrity of the PA Token (the default is `ECDSA using P-256 Curve`). 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.
5. Select the **Encryption Algorithm** used to encrypt and protect the integrity of the PA Token (the default is `AES 128 with CBC and HMAC SHA 256`). PingAccess uses the algorithm when creating encrypted PA Tokens and when verifying them from a user's browser.
  - ➔ **Info:** 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.
  - ➔ **Info:** In a clustered environment, be sure to add the security policy changes to the engines as well as the administrative console for the cluster.
6. Enter the browser **Cookie Name** that contains the PA Token (the default is `PA`).
7. In the **Session State Cookie Name** field, enter a name for the browser cookie to contain session state attributes.
8. In the **Update Token Window(s)** field, enter 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.
9. Click **Save**.

## Create a Web Session

This task allows you to create a new web session.

1. Navigate to **Settings > Access > Web Sessions**
2. On the Web Session page, click **Add Web Session**.
3. Enter a unique **Name** for the web session, up to 64 characters, including special characters and spaces.
4. Select a **Cookie Type**.

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.

5. Specify the **Audience** that the PA Token is applicable to, represented as a short, 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.

- Specify the **OpenID Connect Login Type** that 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 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.

➔ **Info:** 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.

- Specify the **Client ID** that was 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).
- Specify the **Client Secret** that was assigned when you created the OAuth Relying Party Client within PingFederate. Required when configuring the **Code** Login Type. Enter the secret (Client Secret).
 

➔ **Info:** The OAuth Client you use with PingAccess Web sessions must have an OpenID Connect policy specified (for more information see [Configuring OpenID Connect Policies](#) in the PingFederate documentation).
- Specify an **Idle Timeout** that 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.
 

➔ **Info:** 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.
- Specify a **Max Timeout** that 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.



**Note:** This value needs to be set to a smaller value than the PingFederate Access Token Lifetime defined in the PingFederate Access Token Management instance. See [Configuring Reference-Token Management](#) in the *PingFederate Administrator's Manual* for more information.

11. To configure advanced settings, click **Show Advanced**.

12. Specify the valid **Cookie Domain** where the cookie is stored. For example, `corp.yourcompany.com`.

- ➔ **Info:** 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.

13. Select **Secure Cookie** to indicate that the PingAccess cookie must be sent using only HTTPS connections. Selected by default.

- 📄 **Note:** Setting an invalid Cookie Domain or selecting **Secure Cookie** in a non-HTTPS environment causes authentication to fail. This results in PingAccess re-directing the user to re-authenticate with PingFederate indefinitely.

14. Select **Http-Only Cookie** to enable 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.

15. Select **Request Profile** so that 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 that includes the *openid*, *profile*, *address*, *email*, and *phone* scope values (see [Configuring a Client](#)). The **profile** scope is a standard OpenID Connect-defined scope that defines extended claims about a user.

- 📄 **Note:** The user can access all attributes by examining browser traces. While they are integrity protected to prevent changes, any sensitive or confidential attributes can be viewed should the user decode the ID Token's value.

16. Select **Validate Session** so that PingAccess will validate sessions with the configured PingFederate instance during request processing. Use of this feature requires additional configuration in PingFederate. This option is not selected by default.

Session timeouts are synchronized between PingAccess and PingFederate when the following conditions are met:

- A minimum release of PingFederate 8.2 is deployed with **Authentication Session Settings** configured.
- You have selected the **Validate Session** checkbox.

Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.

17. When **Refresh User Attributes** is 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.

The **Refresh User Attributes Interval** determines the length of time the user data is cached, so the effect of a change that results in a session being terminated may take up to 60 seconds (by default) to take effect. 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.

18. When **Cache User Attributes** is enabled, PingAccess caches user attributes internally for use in policy decisions. By doing this, an attribute list that is longer than the maximum cookie size can contain information used to evaluate access requests. In practice, this is 4096 bytes, although the maximum cookie size can vary depending on the browser.

When this option is disabled, user attribute data is encoded, signed or encrypted (depending on the web session cookie type), and stored in the browser's cookie store. The information is sent from the browser back to PingAccess with each request.


Changing this setting may affect existing ongoing sessions, forcing the user to re-authenticate to access protected resources.

This option is not selected by default.

19. Specify a **Consult Server Duration** to define 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.
  - ➔ **Info:** 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.
20. Select **Request Preservation** to specify the type of request data to be preserved if the user is redirected to an authentication page when submitting information to a protected resource. Available options are **None**, **POST**, or **POST and Fragment**.
21. Specify the type of **Web Storage** to be used for request preservation data.
  - Session Storage** is recommended. Use **Local Storage** if it is common for users to use Internet Explorer with security zones enabled and PingFederate is in a different zone than PingAccess.
22. Click **Save**.


### Edit a Web Session

This task allows you to edit a web session.

1. Navigate to **Settings > Access > Web Sessions**.
2. Expand the Web Session you want to edit and click .
3. Edit the Web Session.
4. Click **Save**.

### Delete a Web Session

This task allows you to delete a web session.

- ➔ **Info:** If the Web Session is currently associated with an application, you cannot delete it.
1. Navigate to **Settings > Access > Web Sessions**.
  2. Expand the web session you want to delete.
  3. Click .
  4. When prompted, click **Delete** to confirm the request.

## Unknown Resources

Unknown resources are those for which there is no associated application. These settings define the error responses to be generated for requests that don't match the virtual host and context root of an application. Additionally, agents may be configured to allow unprotected access instead of returning an error response.

### Configure unknown resource management

These steps allow you to define the default action to take when unknown resources are requested. Unknown resources are those for which there is no matching Application.

1. Navigate to **Settings > Access > Unknown Resources**.
2. Specify the **Error Status Code** for the HTTP response. This must be a client/server error code in the range 400 - 599.
3. Specify the name of the velocity **Error Template File** to use for generating the response body. This template file is located in the `PA_HOME/conf/template/` directory.
4. Specify the **Error Content Type** of the response; HTML, JSON, TEXT, or XML.

5. Specify the **Agent Default Mode** that determines whether an agent should **Deny** requests for unknown resources and generate an error response or allow requests to **Pass-Through** unfiltered. This default setting may be overridden by individual agents.
6. Specify the default agent resource **Cache TTL** (in seconds) to be used for unknown resources if **Pass-Through** mode is enabled.
7. Click **Save**.

## Networking

---

Choose from one of the following sections:

- [Availability Profiles](#) on page 52
- [HTTP Requests](#) on page 53
- [Listeners](#) on page 54
- [Load Balancing Strategies](#) on page 55
- [Proxies](#) on page 56

### 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, 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.


#### Create an availability profile

1. Go to **Settings > Networking > Availability Profiles**, then click **Add 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 in milliseconds the amount of time to wait before timing out the request for a pooled connection to the target site in the **Pooled Connection Timeout (ms)** field. Enter -1 for no timeout.
6. Enter the number in milliseconds the amount of time to wait before timing out the read of the response from a target site in the **Read Timeout (ms)** field. Enter -1 for no timeout.
7. Enter the number of times to retry a connection to a backend target before considering the target failed in the **Max Retries** field.
8. Enter the number of milliseconds to wait between retries in the **Retry Delay (ms)** field.
9. Enter the number of seconds to wait before trying a failed target again in the **Failed Retry Timeout (s)** field.
10. 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.
11. Click **Save**.

## Edit an availability profile

1. Navigate to **Settings > Networking > Availability Profiles**, expand the desired profile, then click .
2. Make the required changes to the profile.
3. Click **Save**.

## Delete an availability profile

1. Navigate to **Settings > Networking > Availability Profiles**.
2. Expand the desired profile, then click .
3. Click **Delete** to confirm.

## 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.

➔ **Info:** The IP Source address settings only affect PingAccess as a Gateway; Agents will use the address for the first or last hop, as configured.


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

When the PingAccess Agent is behind a load balancer that is performing HTTPS offload, the load balancer **must** inject `X-Forwarded-Proto` for the **redirect\_uri** to be properly set.

### Configure an alternative IP Source header

1. Navigate to **Settings > Networking > HTTP Requests**
2. In the **IP Source** section of the page, enter a header name to search for in the **Header Names** list.
3. 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`.
4. 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.
5. Click **Save**.

### Configure an alternative Host Source header

1. Navigate to **Settings > Networking > HTTP Requests**
2. In the **Host Source** section of the page, click **Header Names** to enter a header name to search for in the **Header Names** list.
3. If desired, click  to the right of an existing header to delete it from the list.
4. 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`.
5. Click **Save**.

## Configure an alternative Protocol Source header

1. Navigate to **Settings > Networking > HTTP Requests**.
2. In the **Protocol Source** section of the page, enter a header name in the **Header Name** field.
3. Click **Save**.

## 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

#### Assign a Key Pair to a Listener

1. Navigate to **Settings > Networking > Listeners**
2. In the **HTTPS Listeners** section of the page, click the drop-down menu to the right of the listener and select a key pair.
3. Click **Save**.

### Engine key pairs

#### Assign a Key Pair to a Virtual Host

1. Navigate to **Settings > Networking > Listeners**
2. In the **Engine Key Pairs** section of the page, click **Edit** in the row the desired key pair appears in
3. Use the drop-down list to select the virtual hosts the key pair should be used for
4. Click **Save**


### Engine listeners

#### Define an engine listener

1. Navigate to **Settings > Networking > Listeners**
2. Click **Add Engine Listener**.
3. Enter a descriptive name for the listener.
4. Enter the port the listener will open.
  - ➔ **Info:** Remember to open the port in the system firewall, or the listener will not be able to process any incoming requests.
5. If the port should listen for HTTP connections, clear the **Secure** option.
  - 📄 **Note:** By default, engine listeners listen for HTTPS connections to protect sensitive data.
6. Click **Save**.


#### Edit an engine listener

Use this task to modify an existing engine listener.

1. Navigate to **Settings > Networking > Listeners**
2. In the **Engine Listeners** section, expand an existing engine listener.
3. Click .
4. Make the required changes.
5. Click **Save**.

#### Delete an engine listener

Use this task to delete an existing engine listener.

1. Navigate to **Settings > Networking > Listeners**
2. In the **Engine Listeners** section, expand an existing engine listener.
3. Click .
4. Click **Delete** to confirm.

## Load Balancing Strategies

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


The load balancing strategies currently available are `Header-Based` and `Round Robin`.


The `Header-Based` strategy requires a header be included in the request that defines the target to select from the **Site** configuration. This strategy has an option to fall back if the requested target is unavailable, or if the header is missing from the request.

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


### Configure a Load Balancing Strategy

1. Go to **Settings > Networking > Load Balancing Strategies**
2. Click **Add Load Balancing Strategy**.
3. Enter a unique descriptive name for the strategy.
4. Select the **Type**, either `Header-Based` or `Round Robin`.
5. Configure the options for the selected Load Balancing Strategy type:
  - For a `Header-Based` Load Balancing Strategy:
    1. In the **Header Name** field, enter the name of the header that contains the selected target host.
    2. If desired, click **Show Advanced** and select the **Fall Back to First Available Host** option to tell PingAccess to use the first available target defined for the site if the target specified in the header is not available or if the header is not present in the request.
 

 **Note:** If this option is not enabled and the specified target is not available or the request header is not present, the client will receive a `Service Unavailable` response.
  - For a `Round Robin` Load Balancing Strategy:
    1. If browser sessions should not be pinned to a persistent backend target, clear the **Sticky Session Enabled** option. This option is enabled by default.
    2. 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.
 


 **Note:** 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.

### Edit a Load Balancing Strategy

1. Go to **Settings > Networking > Load Balancing Strategies**.
2. Expand the load balancing strategy you want to edit, then click .
3. Make any desired changes to the profile.
4. Click **Save**.

### Delete a load balancing strategy



1. Go to **Settings > Networking > Load Balancing Strategies**.
2. Expand the load balancing strategy you want to edit, then click .
3. Click **Delete** to confirm.

## Proxies

Where applicable, this page contains the forward proxy configuration used when PingAccess makes requests to Sites or Token Providers.

### Add a proxy

This task allows you to add a forward proxy configuration to be used when PingAccess makes requests to sites or token providers.


1. Navigate to **Settings > Networking > Proxies**.
2. Click **Add Proxy**.
3. Specify a **Name** for the proxy configuration.
4. Enter an optional **Description**.
5. Enter the **Host** name for the forward proxy.
6. Enter the **Port** number for the forward proxy.
7. If the forward proxy requires authentication, select the **Requires Authentication** checkbox.
8. If required, enter the **Username** for the forward proxy.
9. If required, enter the **Password** for the forward proxy.
10. Click **Save**.

### Edit a proxy


This task allows you to edit a proxy configuration.



**Note:** If you edit a proxy configuration that is associated with an engine or replica administrative node, you must download and install a new configuration on those nodes.

1. Navigate to **Settings > Networking > Proxies**.
2. Expand an existing proxy configuration.
3. Click .
4. Make the required changes.
5. Click **Save**.

### Delete a proxy

1. Navigate to **Settings > Networking > Proxies**.
2. Expand an existing proxy configuration.
3. Click .
4. Click **Delete** to confirm.

## Security

---

Choose from one of the following sections:

- [Certificates](#) on page 57
- [Key Pairs](#) on page 58

## 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.


### Import a certificate

1. Navigate to **Settings > Security > Certificates**.
2. Click + to the right of the **Certificates** subheading.
3. Enter an **Alias** for the certificate.
4. Click **Choose File** to select the certificate.
5. Click **Add** to import the certificate. A new certificate row appears on the Certificates page.



**Note:** If the Certificate is either expired or not yet valid, PingAccess displays a warning, but the import will proceed.

### Delete a certificate

1. Navigate to **Settings > Security > Certificates**.
2. Expand the certificate you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm the deletion request.



**Info:** If the certificate is associated with a trusted certificate group, you cannot delete it.

### Create a Trusted Certificate Group


1. Navigate to **Settings > Security > Certificates**.
2. Click + to the right of the **Trusted Certificate Groups** heading.
3. Drag a certificate onto the box that appears.
4. Enter a **Name** for the group in the box that appears.
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 **Add**.
8. Additional certificates can be added to the new trusted certificate group by dragging them into the group.


### Add a certificate to a Trusted Certificate Group

1. Navigate to **Settings > Security > Certificates**.
2. Drag a certificate into an existing trusted certificate group.


### Edit a Trusted Certificate Group

1. Navigate to **Settings > Security > Key Pairs**.
2. Expand the trusted certificate group you want to edit.
  - Add a certificate to the group by dragging it into the group from the certificate list.
  - Delete a certificate from the group by clicking - to the right of the certificate.
  - Edit the trusted certificate group parameters by clicking  and then making your changes. If you edit these options, click **Save** to save them.

### 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.

### Delete a Trusted Certificate Group

1. Navigate to **Settings > Security > Certificates**.
2. Expand the trusted certificate group you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm the deletion request.

## 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..


- ➔ **Info:** Ensure that the administrative console node and engines in a cluster have the same cryptographic configuration. For example, if you generate an elliptic curve Key Pair on the administrative console and the engines in the cluster are not configured to support elliptic curve Key Pairs, then the engines are not able to use that Key Pair for the engine **HTTPS Listeners** or as the Key Pair in a **Mutual TLS Site Authenticator**. Cryptographic configuration differences are often caused by having a Java Cryptographic Extension with limited strength providers installed (see the [Oracle Java documentation](#) for more information).

### Import an existing key pair

Use this function to import a key pair from a PKCS#12 file.

1. Navigate to **Settings > Security > Key Pairs**.
2. Click **Import**.

3. In the **Alias** field, enter a name that identifies the key pair. Special characters and spaces are allowed. This name identifies the key pair when assigning the key pair to various configurations such as [HTTPS Listeners](#).
4. Enter the **Password** used to protect the PKCS#12 file. PingAccess uses the password to read the file.
5. Click **Choose File** to locate the PKCS#12 file.
6. Click **Save** to import the file.

 **Note:** If the key pair is either expired or not yet valid, PingAccess displays a warning, but the import will proceed.

### Generate a new key pair

Use this function to generate a key pair and the self-signed certificate.

1. Navigate to **Settings > Security > Key Pairs**.
2. Click **Add Key Pair**.
3. Enter the fields required for the new key pair.

If the key pair is going to be used for incoming requests on multiple hosts or multiple IP addresses, enter additional **Subject Alternative Names** to meet those requirements.


4. Click **Save**.


### Generate a Certificate Signing Request

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

1. Go to **Settings > Security > Key Pairs**.
2. Click **Generate CSR** for the certificate you want to generate a CSR for. PingAccess generates a CSR file and your browser will download it.
3. Provide this file to a Certificate Authority (CA). The CA signs the file and provides a CSR Response that you can upload and use to replace the self-signed certificate. If the CA is well known, its certificates are installed by default in most browsers, and the user is not prompted to trust an unknown certificate.
4. When you receive the CSR response, follow the instructions at [Import a Certificate Signing Request Response](#) on page 59.

### Import a Certificate Signing Request Response


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

 **Note:** Before you import the CSR Response, import the signing CA certificate into PingAccess and add it to a [Trusted Certificate Group](#).

1. Navigate to **Settings > Security > Key Pairs**.
2. Click **CSR Response** for the key pair the CSR applies to.
3. Select the **Trusted Certificate Group** to use for validating that the certificate in the CSR Response is correctly formed.
4. Choose the CSR Response file.
5. Click **Save**.

### Add a certificate to a key pair

This topic describes the steps required to add a certificate to an existing key pair. You start with a leaf certificate, then add the intermediate and root certificates as required.

 **Note:** To modify the certificates included in a chain, remove the certificates from the key pair and add them again or delete the certificate and recreate it by importing a new certificate file and adding certificates to the key pair.

1. Navigate to **Settings > Security > Key Pairs**.

2. Expand an existing key pair.
3. At the bottom of the key pair chain certificate list, click **Add Certificate**.
4. Click **Choose File** to browse for and select the certificate file.
5. Click **Add**.

### Remove a certificate from a key pair

This topic describes the steps required to remove a certificate from an existing key pair.



**Note:** This procedure removes the last certificate in the chain. Certificates can only be removed in reverse order.

1. Navigate to **Settings > Security > Key Pairs**.
2. Expand an existing key pair.
3. To remove the last certificate in the chain, click .
4. Click **Delete** to confirm.

### Download a Certificate

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. Navigate to **Settings > Security > Key Pairs**.
2. Click **Download Certificate** in the row for the certificate you want to download.
3. Your browser will download the certificate and save it in your local filesystem.

### Delete a key pair

➔ **Info:** If a key pair is currently in use, you cannot delete it.

1. Navigate to **Settings > Security > Key Pairs**.
2. Expand the key pair you want to delete.
3. Click .
4. When prompted, click **Delete** to confirm the deletion request.

## System

---

Choose from one of the following sections:

- [Admin Authentication](#) on page 60
- [Configuration Export/Import](#) on page 63
- [Clustering](#) on page 64
- [License](#) on page 68
- [Token Provider](#) on page 68

### Admin Authentication

This page controls the PingAccess administrator authentication method. The default PingAccess administrator authentication method used to protect the administrative console is basic authentication (username and password).


#### Configure 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:

1. Navigate to **Settings > System > Admin Authentication > Basic Authentication**.
2. Enter the current administrator password.
3. Enter and confirm the new password.

 **Important:** The new password must meet the configured password complexity rules defined in `pa.admin.user.password.regex` in `run.properties`.

4. Click **Save**

## Configure Admin UI SSO Authentication

### Prerequisites:


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.


- The [PingFederate Runtime configuration](#) must be completed.
- The PingFederate server certificate must be [imported](#) into a trusted certificate group, and that trusted certificate group must be associated with the PingFederate runtime.
- You must have a **profile** scope set up in PingFederate that includes the *openid*, *profile*, *address*, *email*, and *phone* scope values (see PingFederate documentation for [Configuring a Client](#)).

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

➔ **Info:** Be sure to complete the configuration for connecting to the PingFederate OAuth AS on the [Configure PingFederate for PingAccess SSO](#) on page 71 page as well as completing the steps below.

1. Navigate to **Settings > System > Admin Authentication > Admin UI**.
2. Select the **Authentication Method**, either **Basic Authentication** or **Single Sign-On**.


 **Note:** To select Basic Authentication, it must be enabled on the **Basic Authentication** tab. To select Single Sign-On, the token provider must be configured.




 **Tip:** To define a fall back administrator authentication method if PingFederate is unreachable, enable the `admin.auth=native` property in `run.properties`. This overrides any configured administrative authentication to [Basic Authentication](#).

3. If you selected **Basic Authentication**, proceed to [configuring session properties](#), else select the **OpenID Connect Login Type**, either **Code** or **Post**.
4. Enter the **Client ID** assigned when you created the OAuth client for use with SSO (for more information, see [Configuring a Client](#) in the PingFederate documentation).

When configuring the client in PingFederate, make sure you have the following options selected:

- The **Client Authentication** must be set to `None`
- The **Allowed Grant Types** must be set to `Implicit`
- The **Redirect URIs** must include `https://<PA_Admin_Host>:<PA_Admin_Port>/*`
- If you are not using administrative roles in PingAccess, the OpenID Connect **Policy** should be set to a policy that uses issuance criteria to restrict access based on some additional criteria.

 **Warning:** If the selected OpenID Connect Policy does not use issuance criteria to limit which users can be granted an access token, *ALL* users in the associated identity store configured in PingFederate will be able to authenticate to the PingAccess Admin console and make changes. See [Identifying Issuance Criteria for Policy Mapping](#) in the *PingFederate Administrator's Manual*.

- If you are configuring administrative roles to enable the PingAccess auditor role, the issuance criteria defined in PingFederate should be defined to allow either an administrator or an auditor to be issued an access token. The attribute contract defined in the OpenID Connect Policy must include the additional attribute data that will be used to define the user's role in PingAccess.
5. If you selected the **Code** login type, enter the **Client Secret** assigned when you created the OAuth client.
  6. Select a defined **Authentication Requirements** list, if your environment requires it. Click **Create** to create an authentication requirements list.
  7. If you are enabling Admin SLO and/or using administrator/auditor roles, perform the following steps:
    - a) Click **Show Advanced**.
    - b) In the **Validate Session** field, select **Yes** to validate sessions with the configured PingFederate instance during request processing.
    - c) In the **Refresh User Attributes** field, select **Yes** to periodically refresh user data from the OpenID Connect Token Provider.
    - d) Specify the **Refresh User Attributes Interval** in seconds.
    - e) Select the **Cache User Attributes** checkbox to have PingAccess cache user attribute information for use in policy decisions. When disabled, this data is encoded and stored in the session cookie.
    - f) Click to select the **Use Single Logout** checkbox to enable the use of Single Logout if it is configured for the OIDC Provider.
      -  **Note:** To prevent session replay with SLO enabled, **Check For Valid Authentication Session** must be enabled in PingFederate Access Token Management configuration.
    - g) Select **Enable Roles** to enable roles
    - h) Click **Add Required Attribute** to define a new attribute that is required for Administrator access
      -  **Note:** More than one attribute may be defined here; if more than one is defined, all attribute values must match in order to grant access for the role.
    - i) Enter the attribute name returned in the access token and the attribute value that defines the user as an administrator
      -  **Note:** The attribute name used here is defined in PingFederate under **OAuth Settings > OpenID Connect Policy Management > Your\_Policy > Attribute Contract** as an extension to the contract. The value to use depends on the configuration of the **Contract Fulfillment** tab for the policy.

The attribute named `group` in your attribute contract may be mapped to an LDAP server attribute source that contains a `groupMembership` attribute. A valid group membership for the administrator might be the group `cn=pingaccess-admins,o=myorg`. In this example, you would use `group` as the **Attribute Name** and `cn=pingaccess-admins,o=myorg` as the **Attribute Value**.
    - j) If you want to define criteria for an auditor, select **Enable Auditor Role**
    - k) Define criteria for the auditor in the same way you did for the administrator role
    - l) Click **Add Required Attribute** to add an additional attribute.
  8. Click **Save** when you finish.

### Configure Session Properties

9. Specify the **Cookie Type**, Encrypted JWT or Signed JWT.
10. Specify the unique **Audience** name the token is applicable to.
11. Specify an **Idle Timeout** in minutes. This sets the length of time you want the PA Token to remain active when no activity is detected. When the idle expiration is reached, the session automatically terminates.
12. Specify a **Max Timeout** in minutes. This sets the length of time you want the PA Token to remain active. Once the PA Token expires, an authenticated user must re-authenticate.
13. Specify an **Expiration Warning** in minutes. This specifies the point at which a user will be warned of upcoming session expiry.
14. Specify the **Session Poll Interval** in seconds. This sets the length of time between user info poll requests for the admin UI.

## Configure API Authentication

- ➔ **Info:** For more information on the PingAccess Administrative API, see **Administrative API Endpoints**. If API authentication is not enabled, access is provided using the built-in Administrator account and the password defined in the **Basic Authentication** tab.
1. Navigate to **Settings > System > Admin Authentication > Admin API OAuth**.
  2. Select **Enable** to enable API OAuth authentication.
  3. Enter the **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).
  4. Enter the **Client Secret** assigned when you created the OAuth client in PingFederate.
  5. Select the **Scope** required to successfully access the API. For more information, see [Authorization Server Settings](#) for defining scopes.
  6. Enter the **Subject Attribute Name** you want to use from the OAuth access token as the subject for auditing purposes. At runtime, the attribute's value is used as the Subject field in audit log entries for the Admin API.
  7. If you are using administrator/auditor roles, perform the following steps:
    - a) Select **Enable Roles** to enable role based authentication.
    - b) Click **Add Required Attribute** to define a new attribute that is required for Administrator access.
      - 📄 **Note:** More than one attribute may be defined here; if more than one is defined, all attribute values must match in order to grant access for the role.
    - c) Enter the **Attribute Name** returned in the access token and the **Attribute Value** that defines the user as an administrator.
      - 📄 **Note:** The attribute name used here is defined in PingFederate under **OAuth Settings > OpenID Connect Policy Management > Your\_Policy > Attribute Contact** as an extension to the contract. The value to use depends on the configuration of the **Contract Fulfillment** tab for the policy.

The attribute named `group` in your attribute contract may be mapped to an LDAP server attribute source that contains a `groupMembership` attribute. A valid group membership for the administrator might be the group `cn=pingaccess-admins,o=myorg`. In this example, you would use `group` as the **Attribute Name** and `cn=pingaccess-admins,o=myorg` as the **Attribute Value**.
    - d) If you want to define criteria for an auditor, select **Enable Auditor Role**.
    - e) Define criteria for the auditor in the same way you did for the administrator role.
    - f) Click **Add Required Attribute** to add an additional attribute.
  8. Click **Save** to activate API Authentication.

## Configuration Export/Import

The Configuration Export/Import options create and restore a full PingAccess configuration, allowing it to be backed up and restored into a test environment for testing, or to be used for disaster recovery. The configuration backup is stored as a `json` file, and contains the entire PingAccess configuration.

- ⚠ **Caution:** As the exported `json` file contains your complete PingAccess configuration, ensure the file is stored somewhere with appropriate security controls in place.

### Export PingAccess configuration

#### To Export the PingAccess Configuration

1. Navigate to **Settings > System > Configuration Export/Import**.
2. Click the **Download** button under **Export Configuration**. The downloaded file name is `pa-data-<timestamp>.json`.
  - 📄 **Note:** The `<timestamp>` value is formatted `MM-DD-YYYY.hh.mm.ss` - so a date and time of January 31, 2015 1:35 PM would be encoded as `01-31-2015.13.35.00` in the filename.



## Import PingAccess configuration

The **Import Configuration** option is a version-specific tool used to import a previously exported configuration. PingAccess checks the exported `json` file to ensure that the file came from the same version of PingAccess that it is being imported into.

### To import a PingAccess configuration:

1. Navigate to **Settings > System > Configuration Export/Import**.
2. Under the **Import Configuration** heading, click **Choose File**.
3. Select the `json` export file containing the configuration to import.
4. Click **Import** to start the import process.
5. When prompted for confirmation, click **Confirm**.



**Important:** This operation is destructive, and overwrites your *entire* PingAccess configuration. Passwords in the system will revert to what they were when the backup was created. Unless you perform a backup prior to restoring a different configuration, the configuration prior to clicking **OK** will not be recoverable.

6. **Conditional:** If the Agent or Admin listener key pairs change as a result of the import operation, restart PingAccess.
7. **Conditional:** If the environment is clustered, ensure that the engines are using the proper engine keys. If they are not, re-save the engine to generate a new public key, and reconfigure the engine to use the newly generated key.

## Clustering

PingAccess can be configured in a clustered environment to provide higher scalability and availability for critical services. While it is important to understand that there may be tradeoffs between availability and performance, PingAccess is designed to operate efficiently in a clustered environment.

PingAccess clusters are made up of three types of nodes:

### Administrative Console

Provides the administrator with a configuration interface

### Replica Administrative Console

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

### Clustered Engine

Handles incoming client requests and evaluates policy decisions based on the configuration replicated from the administrative console

Any number of clustered engines can be configured in a cluster, but only one administrative console and one replica administrative console can be configured in a cluster.

Configuration information is replicated to all of the clustered engine nodes and the replica administrative node from the administrative console. State information replication is not part of a default cluster configuration, but some state information can be replicated using PingAccess subclusters.

## PingAccess Subclusters

Subclusters are a method to provide better scaling of very large PingAccess deployments 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.

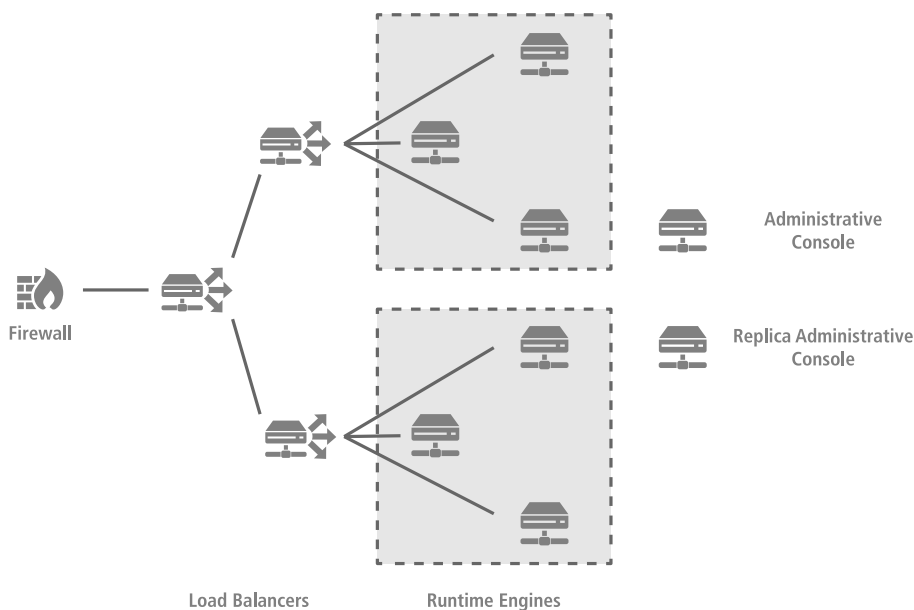
Subclusters serve three purposes:

- Providing fault-tolerance for mediated tokens if a cluster node is taken offline.
- Reducing the number of STS transactions with PingFederate when the front-end load balancer does not provide a sticky session.

- Ensure rate limits are enforced properly if the front-end load balancer does not provide a sticky session.

If token mediation and rate limiting are not used in your environment, subclustering is not necessary.

- ➔ **Info:** This cache can be tuned using the EHCACHE Configuration Properties listed in the **Configuration Properties** documentation.



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.

- ➔ **Info:** 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**.

Engine nodes include a status indicator that indicates the health of the node and a **Last Updated** field that indicates the date and time of the last update. The status indicator can be green (good status), yellow (degraded status), or red (failed status).

The status is determined by using the value for `admin.polling.delay` as an interval to measure health:

**Green (good status):**

The replica administrative node contacted the primary administrative node on the last pull request.

**Yellow (degraded status):**

The replica administrative node contacted the primary administrative node between 2 and 10 intervals.

**Red (failed status):**


The replica administrative node has either never contacted the primary administrative node, or it has been more than 10 intervals since the nodes communicated.

**Engines****Configure an engine**


For each engine:

1. Click **Settings > System > Clustering**
2. Click **Add Engine** to configure a new engine.
3. Enter a **Name** for the engine. Special characters and spaces are allowed.
4. Enter a **Description** of the engine.
5. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.
6. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.
7. Specify the **Engine Trusted Certificate** to use for cases where a TLS-terminating network appliance, such as a load balancer, is placed between the engines and the admin node.
8. 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.
9. 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.
  - ➔ **Info:** 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.
10. **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.
11. Start each engine.
  - ➔ **Info:** For information on configuring engine to share information with each other in a cluster, see [Configure a PingAccess Cluster](#).

**Edit an engine**

1. Navigate to **Settings > System > Clustering**.
2. Expand the engine you want to edit, then click .
3. Edit the engine **Name** or **Description**, as appropriate.
4. If a new public key is needed, click **Save & Download**
5. If a new public key is not needed, click **Save**.




**Remove an engine**

1. Navigate to **Settings > System > Clustering**.
2. Expand the engine you want to delete and click  to permanently remove all references to the engine from the cluster.
3. Click **Delete** in the confirmation window.

## Administrative nodes

### Configure the primary administrative node

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

-  **Warning:** If you are promoting a replica admin node to primary, remove the bootstrap properties file from the replica admin node.
  -  **Note:** This procedure allows you to specify an HTTP or HTTPS proxy. If proxy configuration is defined in a properties file (`bootstrap.properties` or `run.properties`), it will take precedence over UI or API configuration.
  -  **Note:** If a proxy is configured on an replica administrative node, when failing over and before removing the `bootstrap.properties` file, the administrative node should have the same proxy configuration.
1. Enter the host and port for the administrative console. The default is `localhost:9000`.
  2. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.
  3. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.
  4. Click **Save**.

### Configure the Replica Administrative Node

When using a replica administrative node, it is necessary to define a key pair to use for the CONFIG QUERY 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.

In addition to the configuration below, the Replica Administrative node includes a status indicator that indicates the health of the node and a read-only **Last Updated** field that indicates the date and time of the last update. The status indicator can be green (good status), yellow (degraded status), or red (failed status).

The status is determined by using the value for `admin.polling.delay` as an interval to measure health:

#### Green (good status):


The replica administrative node contacted the primary administrative node on the last pull request.

#### Yellow (degraded status):

The replica administrative node contacted the primary administrative node between 2 and 10 intervals.

#### Red (failed status):

The replica administrative node has either never contacted the primary administrative node, or it has been more than 10 intervals since the nodes communicated.

-  **Note:** If a Replica Administrative Node is being configured in the environment, that must be done prior to configuring the engines.

1. Go to **Settings > System > Clustering > Administrative Nodes**.
2. Configure the **Host** value. This name and port pair must match either a subject alternative name in the key pair or be considered a match for the wildcard specified if the key pair uses a wildcard in the common name.
3. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.
4. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.
5. Specify the **Replica Administrative Node Trusted Certificate** to use for cases where a TLS-terminating network appliance, such as a load balancer, is placed between the engines and the admin node.
6. Click **Save & Download** to download the `<replicaname>_data.zip` file for the replica administrative node. PingAccess automatically generates and downloads a public and private key pair into the `bootstrap.properties` file for the node. The **Public Key** is indicated on this screen.
7. Copy the downloaded file to the replica administrative node's `PA_HOME` directory and unzip it.
8. **Conditional:** If the Replica Administrative Node is running on a Linux host, execute the command `chmod 400 conf/pa.jwk`.

9. Edit `PA_HOME/conf/run.properties` on the replica administrative node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`.
10. Start the replica node.
11. 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".

## License

View the details of your PingAccess license or upload a new license.

### Upload or view a PingAccess license

The **Settings > System > License** page displays the current license attribute list. If an admin wants to update an existing license, it may be done via this page. An admin can preview a new license's attributes here, and compare them with the current license before installing it. The UI will display a warning ribbon on the page in certain cases. For example, if the expiration date of the uploaded license is sooner than the current license, the UI will flag that as a warning. If the new license is acceptable, the admin may commit it, and the new license will replace the old.

1. Navigate to **Settings > System > License** to view the attributes of the installed license.



**Note:** If the installation has a running configuration, and the admin shuts down the server, removes the license file from the filesystem, and restarts the server, the existing runtime configuration will continue to work. However, the admin will have to install a new license file on the filesystem, or upload one via the UI, to enable accessing and applying changes via the UI.

2. To upload a license:

- a) Scroll down to the **Import License** section.
- b) Click **Choose a File** to select a license file.



**Warning:** The UI will display a warning ribbon for the following cases:

- **Expiration date:**  
The new license is set to expire on a date earlier than that of your current license.
- **Expired:**  
The new license is already expired.
- **License version:**  
The UI will check and alert when the major version of the license doesn't match the current version of PingAccess.
- **Max Applications:**  
The new license is limited to support fewer applications than your current license.

- c) Click **Import** to import the selected license.



**Note:** If you selected the wrong license, you can either click **Remove** to remove the selected license from the **Import License** section of the page, or you can click **Choose File** to select a different license file.

- d) Click **Confirm** to install the selected license.

## Token Provider

PingAccess can be configured to use PingFederate as the token provider or may be configured to use a common provider via the OpenID Connect protocol.

### Manage Token Provider

This document contains the steps required to configure the token provider. You can choose between PingFederate or another common provider via the OpenID Connect protocol. The following tasks are available:

## PingFederate

- [Configure PingFederate runtime](#) on page 69
- [Configure PingFederate Administration](#) on page 70
- [Configure the OAuth Resource Server](#) on page 70
- [Configure PingFederate for PingAccess SSO](#) on page 71

## Common Token Provider

- [Configure OpenID Connect](#) on page 72
- [Configure OAuth Authorization Server](#) on page 73

## PingFederate

### Configure PingFederate runtime

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.

➔ **Info:** For information on setting PingFederate up as an OAuth Authorization Server, see [Enabling the OAuth AS](#) and [Authorization Server Settings](#) in the PingFederate documentation.

Once the PingFederate Runtime connection is saved, PingAccess will test the connection to PingFederate. If the connection cannot be made, a warning will be displayed in the admin interface.

### Configure the connection to the PingFederate Runtime

1. Navigate to **Settings > System > Token Provider > Runtime**
2. Enter the **Host** name or IP address for the PingFederate Runtime.
3. Enter the **Port** number for PingFederate Runtime.
4. Enter the **Base Path**, if needed, for PingFederate Runtime. This field is optional. It must start with a slash - for example: `/federation`.
5. Select **Audit Level** 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.
6. Select **Secure** if PingFederate is expecting HTTPS connections.
7. From the **Trusted Certificate Group** list, select the certificate group the PingFederate certificate is in. This field is available only if you select **Secure**.
8. To configure advanced settings, click **Show Advanced**.
9. Click **Add Back Channel Server**.
10. Enter one or more `hostname:port` pairs in the **Back Channel Servers** list.
11. **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.
12. **Conditional:** If the back channel uses an alternate base path, enter the path in the **Back Channel Base Path** field.
13. **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.
14. **Conditional:** If hostname verification is required, enter the hostname PingAccess should expect in the **Expected Hostname** field.
15. To use a configured proxy for back channel requests, select the **Use Proxy** checkbox.



**Note:** If the node is not configured with a proxy, requests are made directly to PingFederate.

16. Select **Use Single-Logout** to enable single logout. To use this feature, SLO must be configured on the OIDC provider.

17. Click **Save**.

- ➔ **Info:** Once you save this configuration and [Configure the OAuth Resource Server](#) on page 70, a PingFederate Access Validator is available for selection when you define OAuth-type rules in Policy Manager.

### *Configure PingFederate Administration*

For information on the PingFederate Administration API see [PingFederate Administrative API](#) in the PingFederate documentation.

Once the PingFederate Administration configuration is saved, PingAccess will test the connection to PingFederate. If the connection cannot be made, an error will be displayed in the admin interface, and the configuration will not be saved.

1. Navigate to **Settings > System > Token Provider > Administration**.
2. Enter the **Host** name or IP address for access to the PingFederate Administrative API.
3. Enter the **Port** number for access to the PingFederate Administrative API.
4. Optional: **Optional:** Enter the **Base Path** for the PingFederate Administrative API.

The **Base Path** must start with a slash (/).

For example: /path

5. Enter the **Admin Username**.  
This username only requires Auditor (read only) permissions in PingFederate.
6. Enter the **Admin Password**.
7. Select **Audit Level** 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.
8. Enable **Secure** if PingFederate is expecting HTTPS connections.
9. From the **Trusted Certificate Group** list, select the group of certificates to use when authenticating to PingFederate. PingAccess requires that the certificate in use by PingFederate anchor to a certificate in the associated Trusted Certificate Group. This field is available only if you enable **Secure**.
10. To configure advanced settings, click **Show Advanced**.
11. To use a configured proxy for API requests, select the **Use Proxy** checkbox.



**Note:** If the node is not configured with a proxy, requests are made directly to PingFederate.

12. Click **Save** to save your changes.

### *Configure the OAuth Resource Server*



Prior to configuring this option, PingFederate Administration Configuration must be completed.

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.







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

1. Navigate to **Settings > System > Token Provider > OAuth Resource Server**
2. Enter the OAuth **Client ID** you defined when creating the PingAccess OAuth client in PingFederate.
  - ➔ **Info:** When you configure an OAuth client in PingFederate, be sure to select [Access Token Validation](#) as the allowed grant type. For more information, see [Configuring a Client](#) in the *PingFederate Administrator's Manual*.
3. Enter the **Client Secret** you defined when creating the PingAccess OAuth client within PingFederate.

4. Select **Cache Tokens** to retain token details for subsequent requests. This option reduces the communication between PingAccess and PingFederate.
5. If **Cache Tokens** is enabled, specify the **Token Time To Live** by entering the number of seconds to cache the access token. The default value of -1 means no limit. This value can be -1 or above and must be less than the PingFederate Token Lifetime.
6. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes. 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.
7. If multiple Access Token Managers are configured in PingFederate, select the **Send Audience** option to send the URI the user requested as the `aud` OAuth parameter to select an Access Token Manager.
  -  **Note:** Use of this option requires that the Access Token Management instances be configured with appropriate Resource URIs. Matching of the Resource URI is performed on a most-specific match basis.
8. To enable the use of OAuth 2.0 token introspection select the **Use Token Introspection Endpoint** option.
  -  **Note:** This option is only supported with PingFederate 8.2 or later.
9. Click **Save** to save your changes.

### Configure PingFederate for PingAccess SSO

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.

-  **Note:** 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).
-  **Note:** You must complete the configuration for connecting to the PingFederate OAuth AS instance you plan to use (see *Configure PingFederate Administration* on page 70).

### 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

-  **Info:** 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

➔ **Info:** 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.

### Common token provider

#### *Configure OpenID Connect*

The following steps allow you to configure OpenID Connect as the token provider.

1. Navigate to **Settings > System > Token Provider**.
2. Click **Change Token Provider Type**.
3. When prompted to confirm the change, click **Confirm**.
4. Select **OpenID Connect**.
5. In the **Issuer** field, enter the OpenID Connect provider's issuer identifier.
6. In the **Description** field, enter a description for the token provider.
7. Select the **Audit Level** checkbox to record requests to OpenID Connect provider to the audit store.
8. In the Trusted Certificate Group list, select the group of certificates to use when authenticating to OpenID Connect provider. PingAccess requires that the certificate in use by OpenID Connect provider anchor to a certificate in the associated Trusted Certificate Group.
9. If required, click **Add Query Parameter** and enter custom query parameter name and value pairs used by the OpenID Connect provider in the **Query Parameters** section.
10. To configure advanced settings, click **Show Advanced**.
11. To use a configured proxy, select the **Use Proxy** checkbox.
  -  **Note:** If the node is not configured with a proxy, requests are made directly to the token provider.
12. Select **Use Single-Logout** to enable single logout. To use this feature, SLO must be configured on the OIDC provider.
13. Optionally, configure *token provider specific options*.
14. Click **Save**.

#### Configure token provider specific options

Token provider specific options are plugins that perform particular functions for the selected token provider type. In the case of the PingAccess for Azure AD solution, the plugin addresses the following problems:

- **Transform data** - The format of data returned from the OIDC UserInfo endpoint results in some unexpected JSON formatting. This data is transformed into a format that PingAccess can easily digest.

- **Use the Azure AD Graph API** - If the groups attribute contains more than 200 groups, the id\_token contains a level of indirection that points to a URL in the Azure AD Graph API. Through the creation of a simple purpose-driven application, you can communicate with the Azure ID Graph API to retrieve the complete list of groups.
- **Retrieve group display names** - The groups attribute is a list of GUIDs. The groups for a user are only provided as GUIDs since User-friendly names for AAD groups are not globally unique. Configure the Graph API call to include the group names along with the GUID for creation of more robust policies.

1. First, [Create the Azure AD Graph API application](#)
2. Next, [Configure token provider specific options](#)

### Prerequisite: Create the Azure AD Graph API application

To use the Azure AD Graph API, an application must exist whose purpose is to provide an **Application ID** and **Key** that PingAccess will use as the **Client ID** and **Client Secret** for communication with the Graph API. Create the application in Azure AD via the **App Registrations** blade using the following criteria:

- **Name** - Enter a unique name for the application, for example: *Graph API app*
- **Application Type** - Web app / API
- **Sign-on URL** - This field is not relevant for this particular use case, but is required by Azure AD. Enter the PingAccess host.

1. When the application has been created, navigate to the application in the list.
2. Select **Required permissions** and click **Add**.
3. Choose the **Windows Azure Active Directory API**, select the following permission, and click **Save**.
  - Application Permissions - Read directory data
4. Copy the **Application ID**.
5. Generate and copy a **Key**.

### Configure token provider specific options

1. Navigate to **Settings > System > Token Provider**. Scroll to the **Token Provider Specific Options** section.
2. From the **Type** list, select **Azure Active Directory**.
3. Select the **Use Azure AD Graph API** checkbox.
4. In the **Client ID** field, enter the **Application ID** you copied from the Azure AD API application you created.
5. In the **Client Secret** field, paste the **Key** you copied.
6. Select **Retrieve Group Display Names**.




**Important:** To retrieve groups data for a particular application in the token, the manifest for that application must be modified to include a Group Membership Claim. In the **App Registrations** blade, select the application and click the **Manifest** button. Locate the "groupMembershipClaims" entry and specify a group type, for example "SecurityGroup".

7. Click **Save**.

### Configure OAuth Authorization Server

The following steps allow you to configure the OAuth Authorization Server.

1. Navigate to **Settings > System > Token Provider**.
2. Select **OAuth Authorization Server**.
3. In the **Description** field, enter a description for the authorization server.
4. In the **Targets** field, enter one or more `hostname:port` pairs for the OAuth Authorization Server. Click **Add Target** to add additional targets.
5. In the **Introspection Endpoints** field, specify the OAuth endpoint through which the token introspection operation is accomplished.
6. Optional: Select the **Audit Level** checkbox to record requests to OAuth Authorization server to the audit store.

7. Select the **Secure** checkbox if OAuth Authorization server is expecting HTTPS connections. When selected, use the **Trusted Certificate Group** list to select the group of certificates to use when authenticating to OAuth Authorization Server. PingAccess requires that the certificate in use by OAuth Authorization Server anchor to a certificate in the associated Trusted Certificate Group.
8. In the **Client ID** field, enter the unique identifier assigned when you created the PingAccess OAuth client within your OAuth Authorization Server.
9. In the **Client Secret** field, enter the Client Secret associated with the Client ID.
10. Optional: Select the **Cache Tokens** to retain token details for subsequent requests. This option reduces the communication between PingAccess and OAuth Authorization Server. When selected, use the **Token Time To Live** checkbox to enter the number of seconds to cache the access token. A value of -1 means there is no limit. This value should be less than the OAuth Authorization Server Token Lifetime.
11. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes. At runtime, the attribute's value is used as the Subject field in audit log entries for API Resources with policies that validate access tokens.
12. Select the **Send Audience** checkbox to send the URI the user requested as the `aud` OAuth parameter for PingAccess to the OAuth 2.0 Authorization Server.
13. To configure advanced settings, click **Show Advanced**.
14. To use a configured proxy, select the **Use Proxy** checkbox.  
 **Note:** If the node is not configured with a proxy, requests are made directly to the token provider.
15. Click **Save**.

# Install PingAccess

---

## Install PingAccess

---

This document provides instructions to install PingAccess. PingAccess can be installed on:

- [Linux](#)
- [Red Hat Enterprise Linux](#)
- [Windows](#)

After you install PingAccess, you:

- [Start PingAccess](#) on page 80
- [Access the admin console for the first time](#) on page 81
- [Change configuration database passwords](#) on page 82

You can also:

- [Stop PingAccess](#) on page 82
- [Run PingAccess as a service](#) on page 83
- [Uninstall PingAccess](#) on page 85

## Installation requirements

The following sections detail system, hardware, and port requirements for installing PingAccess:

- [System requirements](#) on page 75
- [Hardware Requirements](#) on page 76
- [Port requirements](#) on page 77

### 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.



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

### Operating systems



**Note:** 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
- Microsoft Windows Server 2016
- Red Hat Enterprise Linux ES 6.8
- Red Hat Enterprise Linux ES 7.2
- Red Hat Enterprise Linux ES 7.3
- SUSE Linux Enterprise 11 SP4

- SUSE Linux Enterprise Server 12 SP2

### 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.

- ➔ **Info:** 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 SE Runtime Environment (Server JRE) 8 (64-bit)

### Supported PingFederate

- PingFederate 8.1
- PingFederate 8.2
- PingFederate 8.3
- PingFederate 8.4
- PingFederate 9.0

### Supported browsers for end users

- Chrome
- Firefox
- Safari
- Edge
- Internet Explorer 9 and higher
- Android
- iOS

### Supported browsers for admin console

- Chrome
- Firefox
- Internet Explorer 11 and higher

### Audit event storage (external database)

- Oracle 11g R2
- Oracle 12c
- MS SQL Server 2012
- MS SQL Server 2016

### Hardware Requirements

- ➔ **Info:** 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

## 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.

- ➔ **Info:** *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	Port details	Source	Description
PingAccess Administrative Console	Protocol: HTTPS Transport: TCP Default port: 9000 Destination: PingAccess Administrative Engine Direction: Inbound	PingAccess Administrator browser, PingAccess administrative API REST calls, PingAccess Replica Admin and clustered Engine nodes	Used for incoming requests to the PingAccess administrative console. Configurable using the <code>admin.port</code> property in the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information. This port is also used by clustered engine nodes and the replica admin node to pull configuration data using the admin REST API.
PingAccess Cluster Communications Port	Protocol: HTTPS Transport: TCP Default port: 9090 Destination: PingAccess Administrative Engine Direction: Inbound	PingAccess Administrator browser, PingAccess administrative API REST calls, PingAccess Replica Admin and clustered Engine nodes	Used for incoming requests where the clustered engines request their configuration data. Configurable using the <code>clusterconfig.port</code> property in the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information. 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	Protocol: HTTP/HTTPS Transport: TCP Default port: 3000 <sup>1</sup> Destination: PingAccess Engine Direction: Inbound	Client Browser, Mobile Devices, PingFederate Engine	Used for incoming requests to the PingAccess runtime engine. Configurable using the <code>Listeners</code> configuration page. See the <a href="#">PingAccess user interface reference guide</a> for more information.
PingAccess Agent	Protocol: HTTP/HTTPS Transport: TCP Default port: 3030	PingAccess Agent	Used for incoming Agent requests to the PingAccess runtime engine. Configurable using


Service	Port details	Source	Description
	Destination: PingAccess Engine Direction: Inbound		the <code>agent.http.port</code> property of the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information.
PingFederate Traffic	Protocol: HTTPS Transport: TCP Default port: 9031 Destination: PingFederate Direction: Outbound	PingAccess Engine	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. See the <a href="#">PingAccess user interface reference guide</a> for more information.
PingAccess Cluster Traffic	Protocol: JGroups Transport: TCP Default port: 7610 Destination: PingAccess Engine Direction: Inbound	PingAccess Engine	Used for communications between engine nodes in a cluster. Configurable using the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information.
PingAccess Cluster Traffic	Protocol: JGroups Transport: TCP Default port: 7710 Destination: PingAccess Engine Direction: Inbound	PingAccess Engine	Used by other nodes in the cluster as part of the cluster's failure-detection mechanism. Configurable using the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information.
PingAccess Cluster Traffic	Protocol: JGroups Transport: UDP Default port: 7500 Destination: PingAccess Engine Direction: Inbound	PingAccess Engine	Used by other nodes in the same cluster to share information. Configurable using the <code>run.properties</code> file. See the <a href="#">Configuration file reference guide</a> for more information.

<sup>1</sup> In addition to port 3000, additional engine listener ports defined in the configuration need to be open as well.


## Install PingAccess on Linux

Prior to starting the installation, the following prerequisites must be satisfied:

- Ensure the [installation requirements](#) are met.
- Ensure you are logged on to your system with appropriate privileges to install and run an application.

 **Note:** On Linux, we recommend that you install and run PingAccess as a non-root user.


- The 64-bit Oracle JRE must be installed.
  - The System or User Environment Variable `JAVA_HOME` must exist and be set to a value that represents the location of your Java installation (ie; `usr/java/jdk 1.8.0_74`).
  - The JRE `/bin` directory (ie; `usr/lib64/jvm/jre/bin`) path must be added to the `PATH` variable so it is available for scripts that depend on it.
  - You must have a `pingaccess.lic` license file. If you do not have one, you can request an evaluation key at the [Request a License Key page](http://www.pingidentity.com/content/pic/en/products/request-license-key.html) ([www.pingidentity.com/content/pic/en/products/request-license-key.html](http://www.pingidentity.com/content/pic/en/products/request-license-key.html)).
1. Download the distribution ZIP file.
  2. Extract the distribution ZIP file into your installation directory.

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


## Install PingAccess on Red Hat Enterprise Linux

Prior to starting the installation, the following prerequisites must be satisfied:


- Ensure the [installation requirements](#) are met.
- Ensure you are logged on to your system with appropriate privileges to install and run an application.

 **Note:** On RHEL, we recommend that you install and run PingAccess as a non-root user.


- The 64-bit Oracle JRE must be installed.
- The System or User Environment Variable `JAVA_HOME` must exist and be set to a value that represents the location of your Java installation (ie; `usr/java/jdk 1.8.0_74`).
- The JRE `/bin` directory (ie; `usr/lib64/jvm/jre/bin`) path must be added to the `PATH` variable so it is available for scripts that depend on it.
- You must have a `pingaccess.lic` license file.

 **Note:** If you do not have one, you can request an evaluation key at the [Request a License Key page](http://www.pingidentity.com/content/pic/en/products/request-license-key.html) ([www.pingidentity.com/content/pic/en/products/request-license-key.html](http://www.pingidentity.com/content/pic/en/products/request-license-key.html)). During the first run of PingAccess, you will be prompted to upload the license file via the UI.

1. Download the installation script.
2. **Optional:** Download the distribution tar file into your installation directory.

 **Note:** Downloading the distribution file before running the script may save time. If you do not download the distribution file, the installer downloads the required files during the installation process.


3. Launch the installation script: `sudo -E ./pa-install-<version>.sh`

 **Note:** On supported RHEL versions, the newer **systemd** scripting service is used during installation. On earlier versions, the installation is performed using the older **systemv** scripting service.

4. Follow the prompts on the screen, enter your preferred options, or accept the default values. The installation completes by starting up PingAccess according to the parameter values you supplied. Wait until you see the "Starting" and "Started" messages.
5. Open a browser and enter the machine and PingAccess admin port in the URL, for example:

`https://yourhost:9000`

6. Finalize your setup.

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

## Install PingAccess for Windows

Prior to starting the installation, the following prerequisites must be satisfied:



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



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

- The 64-bit Oracle JRE must be installed.
  - The System Environment Variable JAVA\_HOME must exist and be set to a value that represents the location of your Java installation (ie; *C:\Program Files\Java\jre 1.8.0\_91*).
  - The *javapath* directory path (ie; *C:\Program Files\Oracle\Java\javapath*) must be added to the PATH variable.
1. Download the PingAccess Windows installer.
  2. Double-click on the icon to launch the PingAccess Setup Wizard.
  3. Click **Next** and follow the prompts to complete the installation using the following information for the **Operational Mode** that you select.

Operational Mode	Requirements
Standalone	<p><b>Ports</b></p> <ul style="list-style-type: none"> <li>• <b>PingAccess administrative console:</b> TCP 9000</li> <li>• <b>PingAccess agent protocol:</b> TCP 3030</li> </ul>
Clustered Admin Node	<p><b>Ports</b></p> <ul style="list-style-type: none"> <li>• <b>PingAccess administrative console:</b> TCP 9000</li> <li>• <b>Configuration query port:</b> TCP 9090</li> </ul>
Clustered Replica Admin Node	<p><b>Ports</b></p> <ul style="list-style-type: none"> <li>• <b>PingAccess administrative console:</b> TCP 9000</li> <li>• <b>Configuration query port:</b> TCP 9090</li> </ul> <p><b>Prerequisites</b></p> <ul style="list-style-type: none"> <li>• A Clustered Admin Node must be installed and configured.</li> <li>• A configuration data archive file for the Replica Admin Node must be available. Consult PingAccess clustering documentation for more information.</li> </ul> <p> <b>Note:</b> We recommend that you install the Clustered Replica Admin Node on a separate machine in the same network.</p>
Clustered Engine Node	<p><b>Ports</b></p> <ul style="list-style-type: none"> <li>• <b>PingAccess agent protocol:</b> TCP 3030</li> </ul> <p><b>Prerequisites</b></p> <ul style="list-style-type: none"> <li>• A Clustered Admin Node must be installed and configured.</li> <li>• A configuration data archive file for the Clustered Engine Node must be available. Consult PingAccess clustering documentation for more information.</li> </ul>

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

## Start PingAccess



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

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

Wait for the script to execute. PingAccess is started when you see the message “PingAccess running...” in the command window.

## Access the admin console for the first time

---

1. Launch your browser and go to: `https://<DNS_NAME>:9000`  
 <DNS\_NAME> is the fully-qualified name of the machine running PingAccess.
  - ➔ **Info:** If you have not yet installed a PingAccess license, the server will redirect you to the **License Upload** screen outside of the main UI. See the [PingAccess User Interface Reference Guide](#) for more information.
2. Sign in with the default username and password:
  - Username:** Administrator
  - Password:** 2Access
3. Read and accept the license agreement.
4. Change the default administrator password on the **First Time Login** page, then click **Continue**.
  - ➔ **Info:** 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.

Upon a successful login, PingAccess creates a backup of the current configuration to allow the administrator to revert any changes made. This backup is stored in `PA_HOME/data/archive`. The number of backup files can be controlled using the `pa.backup.filesToKeep` property in `run.properties`.

- ⚠ **Caution:** As the backup file contains your complete PingAccess configuration, ensure the file is protected with appropriate security controls in place.

## Access the PingAccess administrative API

---

Use this task to access the PingAccess Administrative API.

Send HTTP request to URL `https://<host>:<admin-port>/pa-admin-api/v3/<api-endpoint>`.

- 📄 **Note:** 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/v3/applications
```

- The `-u Administrator:Password1` parameter sends Basic Authentication header with the username Administrator and password Password1

- The `-k` parameter specifies to ignore HTTPS certificate issues
- The `-H "X-Xsrf-Header: PingAccess"` parameter sends an X-XSRF-Header with value PingAccess

## Access the interactive administrative API documentation

1. Launch your browser and go to URL `https://<host>:<admin-port>/pa-admin-api/v3/api-docs/`. For example, `https://localhost:9000/pa-admin-api/v3/api-docs/`.
2. The browser may prompt for credentials. Enter the administrator username and password.
3. Use the administrative API to perform a variety of administrative tasks, including those to gather information as seen in the following example that uses the interactive Administrative API documentation to see all defined applications:
  - a) Click on the `/applications` endpoint to expand it.
  - b) Click on the GET method (`GET /applications`) to expand it.
  - c) Enter parameters values or leave all blank.
  - d) Click **Try It Out** button.
  - e) The Request URL, Response Body, Response Code, and Response Headers appear.

## Change configuration database passwords

---

The PingAccess configuration database is protected by two passwords - a file password and a user password. These passwords both default to `2Access`, but should be changed for production environments.

Changing either password requires PingAccess be shut down.

1. Open a terminal window and change to the `<PA_HOME>/bin` directory.
2. Ensure that the `JAVA_HOME` environment variable is set correctly by executing the command `echo $JAVA_HOME`.
3. Ensure that the proper Oracle Java executable is in your path. Enter the command `java -version`. If this command returns a value indicating that the Java executable is not a supported version of Oracle Java, correct this issue before continuing.
4. Shut down PingAccess.
5. Optional: (Optional) To change the database file password, use the following commands:
  - On Windows: `dbfilepasswd.bat old_password new_password`
  - On Linux: `./dbfilepasswd.sh old_password new_password`
6. Optional: **Conditional:** If you changed the database file password, update the `pa.jdbc.filepassword` property in `PA_HOME/conf/run.properties` with the obfuscated password output from the command used in the preceding step.
7. Optional: (Optional) To change the database user password, use the following commands:
  - On Windows: `dbuserpasswd.bat file_password old_password new_password`
  - On Linux: `./dbuserpasswd.sh file_password old_password new_password`
8. Optional: **Conditional:** If you changed the database user password, update the `pa.jdbc.password` property in `PA_HOME/conf/run.properties` with the obfuscated password output from the command used in the preceding step.

## Stop PingAccess

---


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

## Run PingAccess as a service


PingAccess can run as a service on Linux and Windows 64-bit operating systems. This enables PingAccess to start automatically when the operating system is started. The service runs as the `root` (Linux) and `System` (Windows) user by default.

The following tasks allow you to manage PingAccess as a service:

- [Configure PingAccess to run as a Linux systemd service](#) on page 83
- [Configure Multiple Instances of PingAccess as Linux services](#) on page 84
- [Remove the PingAccess Linux service](#) on page 84
- [Configure PingAccess to run as a Windows service](#) on page 84
- [Remove the PingAccess Windows service](#) on page 85

 **Tip:** Before performing the following procedures, ensure that PingAccess runs normally by manually starting the server. See [Run PingAccess for the First Time](#) for more information.

### Configure PingAccess to run as a Linux systemd service

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

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`

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

**start**

Start the PingAccess Service

**stop**

Stop the PingAccess Service

**restart**

Restart the PingAccess Service

**status**

Show the status of the PingAccess service and the service PID

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

### Configure PingAccess to run as a Linux systemd service

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

1. Copy the configuration file from `PA_HOME/sbin/linux/pingaccess.service` to `/etc/systemd/system/pingaccess.service`.
2. In the `pingaccess.service` file, replace the following variables:
  - `PA_HOME`
  - `PA_USER`
  - `JAVA_HOME`
3. Allow read/write activity on the service using the command `chmod 644 /etc/systemd/system/pingaccess.service`.
4. Load the systemd service using the command `systemctl daemon-reload`.
5. Enable the service using the command `systemctl enable pingaccess.service`.
6. Start the service using the command `systemctl start pingaccess.service`.

## Configure Multiple Instances of PingAccess as Linux services

For hosts running multiple instances of PingAccess that need to be started as a service, follow the procedure used for [Configuring PingAccess as a Linux Service](#), but make the following modifications to the script for each service:

- 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

## Remove the PingAccess Linux service



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

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

## Configure PingAccess to run as a Windows service

➔ **Info:** Before performing this procedure, ensure that PingAccess runs normally by manually starting the server (see [Run PingAccess for the First Time](#)).



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

This configuration allows you to start PingAccess automatically when Windows starts.

1. Install PingAccess.
  - ➔ **Info:** Ensure `JAVA_HOME` is set as a system variable.
2. Ensure you are logged in with full Administrator privileges.
3. Start a Command Prompt as an Administrator.
4. In the Command Prompt, run `install-service.bat`. This script is located in `PA_HOME\sbin\windows`.
5. Open 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.)

## Remove the PingAccess Windows service

To remove the PingAccess Windows Service, perform the following steps as an Administrator:

1. Open a Command Prompt
2. Change the current directory to `PA_HOME\sbin\windows`
3. Run `uninstall-service.bat`
4. When the script has finished, remove the `PA_HOME` environment variable from the system.

## Uninstall PingAccess

---


1. [Stop PingAccess](#) on page 82.
2. Delete the PingAccess installation directory.

# Upgrade PingAccess

---

## Upgrade a PingAccess standalone version

---

-  **Important:** In release 5.0, there are potentially breaking changes to the SDK for Java, Groovy scripts, and the Administrative API. For information on these changes and the actions administrators may need to take, review the [PingAccess Release Notes for release 5.0](#), in addition to any release notes for releases between your current version and the target version.

This document provides instructions for upgrading a standalone instance of PingAccess.


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

- The PingAccess Upgrade Utility archive
- The PingAccess distribution zip file
- Your PingAccess 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.
- The version of PingAccess you are upgrading from must be running
- Administrator credentials for the running PingAccess instance

 **Important:**


If your installation uses custom plugins, they will need to be rebuilt against the new (5.0) SDK. You will then run the Upgrade Utility manually with the new "-i" command-line option to include their plugin directory. To migrate your custom plugins, see the [PingAccess Addon SDK for Java Migration Guide](#).

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


-  **Important:** If you have set `security.overridePropertiesFile=false` in `$JAVA_HOME/jre/lib/java.security`, the upgrade utility may fail, as the PingAccess Upgrade Utility uses an override to enable deprecated ciphers and protocols during the upgrade process.

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

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

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

Use the PingAccess Upgrade Utility to upgrade from PingAccess 2.1 or later to the most recent version.

-  **Attention:** Before you upgrade, create a backup of your existing PingAccess configuration. In the event of an upgrade failure, you can restore the backup configuration.

1. Unpack the upgrade utility zip file.
2. Change to the upgrade utility's `bin` directory.
3. Run the PingAccess Upgrade Utility:
  - On Windows: `run.bat [-p <admin_port>] [-i <directory>] <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`
  - On Linux: `./run.sh [-p <admin_port>] [-i <directory>] <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`

For example: `./run.sh -p 9002 -i MyJARDir pingaccess-4.3 pingaccess-5.0 pingaccess-5.0.zip pingaccess50.lic`

### Parameter Definitions

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

Parameter	Value description
<code>&lt;admin_port&gt;</code>	Optional port to be used by the temporary PingAccess instance run during the upgrade. The default is 9001.
<code>&lt;directory&gt;</code>	A directory containing additional library JAR files (e.g. plugins, JDBC drivers) to be copied into the target installation
<code>&lt;sourcePingAccessRootDir&gt;</code>	The PA_HOME for the source PingAccess version
<code>&lt;outputDir&gt;</code>	The target directory which will contain the unpacked PingAccess distribution
<code>&lt;pingaccessZip&gt;</code>	The PingAccess distribution for the target version
<code>&lt;newPingAccessLicense&gt;</code>	The path to the PingAccess license file to use for the target version



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

## Upgrade a PingAccess cluster

**Important:** In release 5.0, there are potentially breaking changes to the SDK for Java, Groovy scripts, and the Administrative API. For information on these changes and the actions administrators may need to take, review the [PingAccess Release Notes for release 5.0](#), in addition to any release notes for releases between your current version and the target version.

**Important:** It is important that all nodes in a clustered PingAccess be running the same software release.

**To upgrade a cluster, perform the following steps:**

**Attention:** Before you upgrade, create a backup of your existing PingAccess configuration. In the event of an upgrade failure, you can restore the backup configuration.


1. Run the upgrade utility on the administrative console
2. Change the upgraded administrative console's `admin.port` and `clusterconfig.port` values 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




## Upgrade Windows using the installer

---


You can upgrade your Windows PingAccess installation using the installer.

 **Important:** If additional JAR files (such as custom plugins and JDBC drivers) have been added to the existing PingAccess /lib directory, the 5.0-Beta installer cannot be used to perform the upgrade. Instead, the Upgrade Utility should be run manually, using the "-i" command-line option to specify the JAR files to be included.

1. Download the installer.
2. Start the installer. The existing installation is detected.
3. Choose **Yes** to upgrade the installation.
4. Choose a license file and specify a temporary admin port. Click **Next**.

 **Note:** The temporary admin port is not required when upgrading a cluster node.

5. Specify the administrator credentials. Click **Next**.


 **Note:** Administrator credentials are not required when upgrading a cluster node.


6. Click **Finish**.
7. Complete required post-upgrade tasks.

## Upgrade RHEL using the installer

---

You can upgrade your RHEL PingAccess installation using the installer.


 **Note:** On supported systems, you are given the option to upgrade your existing **systemv** scripting service to the newer **systemd** service. Upgrading the service will replace the existing service.

 **Important:** If additional JAR files (such as custom plugins and JDBC drivers) have been added to the existing PingAccess /lib directory, the 5.0-Beta installer cannot be used to perform the upgrade. Instead, the Upgrade Utility should be run manually, using the "-i" command-line option to specify the JAR files to be included.


1. Download the installer script.
2. Launch the script using the command `sudo -E ./pa-install-<version>.sh`.
3. Specify the instance you want to upgrade.

 **Important:** If you are upgrading a PingAccess cluster, always upgrade the console first.

4. Confirm the selection.
5. Specify a license file.
6. Specify the temporary admin port.

 **Note:** The temporary admin port is not required when upgrading a cluster node.

7. Specify whether you have additional plugin or JDBC driver JAR files to include during upgrade.

 **Note:** This step is required when plugin JARs are present and you are upgrading to a version that is not SDK backward-compatible. For more information, see [Upgrade a PingAccess standalone version](#) on page 86 and the [PingAccess Addon SDK for Java Migration Guide](#).

8. **Optional:** If you selected `yes` to include additional JAR files, enter the path to the `include` directory. The JAR files in this directory will be copied into the new PingAccess installation instead of migrating additional library JARs from the existing installation.
9. Provide the administrator credentials.



**Note:** Administrator credentials are not required when upgrading a cluster node.

10. Allow the upgrade to complete and perform any required post-upgrade tasks.

## Complete the upgrade

At the end of an 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 the new PingAccess server to complete the manual portion of the upgrade. To see details about the migrated configuration data, examine `log/audit.log`.

After the upgrade has completed, perform the following steps:

1. Review any warnings returned by the upgrade utility and take the actions indicated in the table below.
2. 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.
3. Review the HTTP Requests 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.

Warning text	Steps to take
Resource ' <i>ResourceName</i> ' 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, <code>/(app1 app2)</code> . This can be translated to a single resource in 3.1.1 with path prefixes of <code>/app1</code> and <code>/app2</code> .
Resource ' <i>ResourceName</i> ' 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 <code>/(?i)</code> 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 ' <i>ResourceName</i> ' requires a case-insensitive path. This conflicts with its containing Application, which requires a case-sensitive path. Manual intervention may be required.	This is the same as the previous setting, but with the requirement being for a case-insensitive path rather than a case-sensitive one.
Resource ' <i>ResourceName</i> ' is disabled in the source version. Resources can no longer be individually disabled. Application ' <i>ApplicationName</i> ' 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.

Resource '*ResourceName*' could not be migrated to the target version due to Application context root conflicts. Manual intervention is required.

Application '*ApplicationName*' contains OAuth rules, but authenticates users with a web session. Unexpected results may occur.

The resource order for Virtual Host '*VirtualHostName*' has changed 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.

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.

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, however, an API application and web application are mutually exclusive so the semantics of this particular configuration cannot be preserved.

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

Application '*ApplicationName*' is no longer associated with an Identity Mapping. A Web Session or an Authorization Server is required to use Identity Mappings.

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.

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.

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.

Upgrade created 'Availability Profile for Site '*SiteName*'. A more descriptive name may be required.

Application '*ApplicationName*' and associated Resources were not migrated. The context root of /pa is reserved. Manual intervention may be required.

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 OAuth Groovy Script Rule no longer controls the realm in the response sent for an unauthorized OAuth request.

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.

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.

Indicates a misconfiguration in the source version. Check whether the OAuth Rule is necessary to implement the desired Access Control policy.

Indicates a misconfiguration in the source version. Check whether the OAuth RuleSet is necessary to implement the desired Access Control policy.

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.

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.

The /pa context root was allowed as a valid context root in PingAccess 3.0 and is no longer allowed.

The /pa path prefix was allowed as a valid path prefix in PingAccess 3.0 and is no longer allowed.

With PingAccess 3.2, Realms have been moved to the Application. The Realm can still be set using the PingAccess Admin API interface. With the change in context for how realms are applied, it is necessary to check existing OAuth Groovy Rules to ensure that they behave as expected. This message is shown if any OAuth Groovy Rules exist in the migrated configuration.

The property '*PropertyName*' was set to a blank value to maintain compatibility. However, it is recommended that this be set to '*PropertyName=PropertyValue*'

As a security enhancement, the default value of '*CipherList*' has changed with this version of PingAccess. Your existing ciphers remain unchanged. However, it is recommended to use the default value: '*PropertyName=CipherList*'.

The property '*PropertyName*' was set to a blank value to maintain compatibility. However, it is recommended that this be set to '*PropertyName=CipherList*'

The host for VirtualHost *VirtualHost:Port* already has a KeyPair associated with it. The KeyPair previously associated with this VirtualHost was removed. Only one KeyPair can be associated with a given host.

Application with name '*ApplicationName*' not migrated as the context root '*Path*' was a reserved path.

Resource with name '*ResourceName*' not migrated as the path '*Path*' was a reserved path.

The CIDR Rule with name '*RuleName*' is associated with an Agent Application named '*ApplicationName*' and overrides the IP source configuration. A new Agent rule should be created that does not override the IP source.

Require HTTPS option on Application '*ApplicationName*' was set to *Setting* as Virtual Host had port *Port*. Please verify this setting is correct.

VirtualHost '*VirtualHost*' was not migrated. An existing VirtualHost existed with the same logical name '*VirtualHost*'.

New Security Headers properties values are not set during an upgrade in order to preserve the behavior from the source release in the upgrade. If there is no reason not to in your environment, update *run.properties* with the recommended setting.

This message applies to the *admin.ssl.ciphers*, *engine.ssl.ciphers*, and *agent.ssl.ciphers* lists. This message is displayed if the upgrade source version cipher lists are changed from the defaults. We recommend updating the configuration with the new default value if possible.

This message applies to the *site.ssl.protocols*, *site.ssl.ciphers*, *pf.ssl.protocols*, and *pf.ssl.ciphers* settings. The upgrade utility sets these values as empty values in order to maintain backwards compatibility, but the recommended value should be used if possible.

If a Virtual Host has more than one key pair associated with it, only one key pair will be associated with it after the upgrade completes. This message is displayed to indicate which key pair was used.

If an Application's context root is a reserved PingAccess path, the application will not be migrated. The indicated application will need to be created with a context root that does not conflict with the reserved path.

If an Resource path is a reserved PingAccess path, the application will not be migrated. The indicated application will need to be created with a context root that does not conflict with the reserved path.

With changes in IP Source header handling, additional options are available to override the headers used to identify the source address. When an Agent is involved, the changes in IP source handling may cause the specified rule to not behave as expected.

The upgrade utility attempts to set the **Require HTTPS** option based on the virtual host associated with an application during an upgrade. This message is an advisory to just verify that the setting was properly detected.

Virtual Host names are now case-insensitive. During the upgrade, after making the names case-insensitive, a duplicate Virtual Host was identified. It will be necessary to either recreate the virtual host with a new name, or to modify the configuration so the proper Virtual Host is migrated to the upgraded system.

Renamed Virtual Host's Hostname from 'VirtualHost' to 'NewVirtualHost' due to virtual host spec compliance issue

Removed Http Request Rule with name 'RuleName', this Rule must be converted to a groovy script rule. Manual intervention may be required.

The property 'PropertyName' uses a customized value. "Your original value has not been modified. You may encounter startup or connection problems if this value is not supported by the JVM.

Rule with ID *RuleId* and name 'RuleName' was not migrated as matcher was invalid for the Groovy rule type.

Invalid rules were removed from RuleSet 'RuleSetName' which resulted in an empty set.

The RuleSet was removed. Please check your policy configuration.

Invalid rules were removed from RuleSet 'RuleSetName'. Please check your policy configuration.

If a Virtual Host name contains an underscore ( `_` ) character, that does not conform to host naming requirements. In this instance, the underscore will be renamed to the string a-z. For example, if a Virtual Host named `my_virtual_host` is migrated, the new name will be `mya-zvirtuala-zhost`.

When an HTTP Request Rule is migrated from an earlier release of PingAccess, rules that specify a source of Body are not migrated. A Groovy script rule can be used to perform a similar match, but the details of such a Groovy script require administrator intervention.

A simple Groovy script rule that would perform a similar function might be:

```
requestBodyContains('value')
```

We advise, however, that a script be constructed that performs additional validation in order to ensure the rule passes only when desired; a generic match like this could lead to unexpected results depending on what content might be in the request body.

When migrating SSL settings between versions of PingAccess that use different JVM or JDK versions, custom settings may not be compatible. If the protocols or ciphers used are not compatible with the target JVM or JDK, this message indicates which settings need to be manually updated.

The *PropertyName* value can be any of the following values:

- `site.ssl.protocols`
- `site.ssl.ciphers`
- `pf.ssl.protocols`
- `pf.ssl.ciphers`
- `admin.ssl.protocols`
- `admin.ssl.ciphers`
- `engine.ssl.protocols`
- `engine.ssl.ciphers`
- `agent.ssl.protocols`
- `agent.ssl.ciphers`

These messages may be displayed if the source PingAccess installation has misconfigured Groovy Rules.

This indicates that you are not permitted to add an OAuth rule to an Application of type Web, by editing an existing Rule Set.

Groovy or OAuth Groovy rules will not be migrated for the following reasons:

- The OAuth Groovy rule was applied to a Web application.

Invalid Rules were removed from Application '*ApplicationName*'. Please check your policy configuration.

Invalid RuleSets were removed from Application '*ApplicationName*'. Please check your policy configuration.

Invalid Rules were removed from Resource '*resource name*' on Application '*ApplicationName*'. Please check your policy configuration.

Invalid RuleSets were removed from Resource '*resource name*' on Application '*ApplicationName*'. Please check your policy configuration.

Rule with name '*RuleName*' has been removed from RuleSet with name '*RuleSetName*'. Multiple Rate Limiting Rules with the same Policy Granularity cannot be included in a RuleSet."

Rule with name '*RuleName*' has been removed from RuleSet with name '*RuleSetName*'. Multiple Cross-Origin Request Rules cannot be included in a RuleSet."

One or more notifications were issued while migrating from version *SOURCE* to version *TARGET*

Setting clusterconfig.enabled to false

The new configuration query port feature has been disabled for backward compatibility. Please refer to the PingAccess clustering documentation before enabling this feature.

One or more notifications were issued while migrating from version *SOURCE* to version *TARGET*

For backward compatibility, when connecting to a protected, TLS SNI-enabled Site, PingAccess will set the SNI server\_name to the configured target host and not the HTTP request Host header value. Please refer to PingAccess' upgrade documentation for more information.

Localization property '*{property name}*' was added to pa-

- The Groovy or OAuth Groovy uses a matcher that is not appropriate for the application type.

Check the policy configuration.

The upgrade utility supports migrating a RuleSet containing multiple CORS or Rate Limiting rules with the same Policy Granularity. The upgrade utility will generate new action items, indicating that rules were removed from a RuleSet.

These messages indicate that if both rules exist, there is a restriction to a single Rate Limiting or CORS rule. Please check to confirm that you have applied the correct rule to the policy.

The new cluster config query port is enabled by default for new PingAccess 4.0 installations when running in CLUSTERED\_CONSOLE or CLUSTERED\_CONSOLE\_REPLICA mode.

During the upgrade process to version 4.0, the new cluster config query port is disabled. Messages are written to upgrade.log and audit.log to indicate this cluster configuration change was made.

Please refer to the PingAccess clustering documentation before enabling this feature.

During upgrades to release 4.0 and higher, the Upgrade Utility sets the value of pa.site.tls.sni.legacyMode to true to maintain compatibility with existing configurations. This property is controlled in the run.properties file and is not enabled on new installs.

This message will appear if new language properties are added between the source and target PA versions and you have added additional language files or modified

messages.properties. Any customized localization files should be updated.

Localization property '{*property name*}' in pa-messages.properties was modified. Any customized localization files should be updated.

Localization property '{*property name*}' was removed from pa-messages.properties. This property can be removed from any customized localization files.

WebSessionManagement contained an invalid cookie name. Replaced '{*old cookie name*}' with '{*new cookie name*}'. Please validate your configuration.

Legacy authentication requirements policy evaluation has been enabled to maintain backward compatibility with earlier versions of PingAccess. To disable this setting, remove the pa.policy.eval.acr.v42 property from run.properties.

the en or en\_US files. Update any customized files as required.

This message will appear if the language properties have changed between the source and target PA versions and you have added additional language files or modified the en or en\_US files. Update any customized files as required.

This message will appear if the language properties have been removed between the source and target PA versions and you have added additional language files or modified the en or en\_US files. Update any customized files as required.

This message will appear if the WebSessionManagement has an invalid cookie name. Invalid characters are replaced with an underscore. Update any references as required.

This message will appear on upgrade to release 4.3 or higher if you have one or more authentication requirements rules. You can make adjustments to configured rules so you can remove this property or you can maintain the property to leave existing rules unaffected.

## Restore a PingAccess configuration backup

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



**Caution:** This operation will replace your current configuration settings.

### To Restore a PingAccess Configuration Backup:

1. Stop PingAccess.
2. Unzip the backup file to *PA\_HOME*.
3. Restart PingAccess.

Your PingAccess configuration will now be reverted to the state in the backup archive that was restored.



# Configure session management

---

## Configure session management

---

This document provides information regarding session management using PingAccess. Use this document to learn about the concepts involved and to configure PingAccess for server-side session management using PingFederate.

### 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 the following tasks to configure secure Web Sessions for use with specific applications and to configure global Web Session settings.

### Application scoped Web Sessions

PingAccess Tokens can be configured to have their Web Sessions scoped to a specific application. This improves the security model of the session by preventing unrelated applications from impersonating the end user.

Several controls exist to scope the 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.

➔ **Info:** 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.

## Configure server-side session management

---

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

- PingAccess can reject a PingAccess cookie associated with a PingFederate session that has been invalidated as a result of an end-user driven logout.
- 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 for session management

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

1. Log in to the PingFederate Administrative Console
2. Navigate to **Server Configuration > 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, navigate to **OAuth Settings > Authorization Server Settings** and configure the 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 enabled, then 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**.



**Note:** This setting is the main setting that enables the server-side session management feature in PingFederate.

9. Click **Save** to save your changes.

## Configure PingFederate for user-initiated single logout

1. Log in to the PingFederate administrative console
2. Navigate to **OAuth Settings > Authorization Server Settings**
3. Select **Track User Sessions for Logout** under **OpenID Connect Settings**
4. Click **Save** to save this change
5. Navigate to **OAuth Settings > OpenID Connect Policy Management** and click an existing policy
6. Click **Manage Policy**, then enable **Include Session Identifier in ID Token**. (For more information about configuring an OpenID Connect Policy, see [Configuring OpenID Connect Policies](#) in the *PingFederate Administrator's Manual*.)
7. Click **Save** to save this change
8. Navigate to **OAuth Settings > Client Management** and select the client to be used by PingAccess
9. In the OpenID Connect section of the client's configuration page, enable **PingAccess Logout Capable**.



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

10. Click **Save** to save this change

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

## Configure PingAccess for server-side session management

To configure PingAccess:

1. Log in to the PingAccess administrative console.
2. Click **Settings > Access > Web Sessions**.
3. Either [create a new web session](#) or [edit an existing web session](#).
4. In the **Client ID** field, enter the client name defined in PingFederate.

5. Enter the client secret associated with the specified Client ID.
6. Click the **Advanced** tab.
7. Select **Validate Session** to enable the server-side session management feature.
8. Click **Save**.

# Configure logging

---

## Configure logging

---

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

## Security audit logging

---

The PingAccess audit logs record a selected subset of transaction log information at runtime plus additional details, intended to facilitate security auditing and regulatory compliance. The logs are located in *PA\_HOME/log/*. Elements recorded in these logs are described in the table below, and are configured in *conf/log4j2.xml*.

PingAccess generates these audit logs:

### **pingaccess\_engine\_audit.log**

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

### **pingaccess\_agent\_audit.log**

Records transactions between PingAccess Agents and the PingAccess Engine.

### Audit log configuration

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

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

Item	Description
AUDIT.respSentMillisec	Time in milliseconds (since 1970) that a response was sent back to the client
AUDIT.roundTripMS	The respSentMillisec time minus the reqReceivedMillisec time. This represents the total number of milliseconds it took PingAccess to respond to a client's request (including the proxyRoundTripMS).
AUDIT.proxyRoundTripMS	The respReceivedMillisec time minus the reqSentMillisec time. This represents the total number of milliseconds PingAccess was waiting for another entity to respond to a backchannel call or proxy request.
AUDIT.userInfoReqSentMillisec	The time in milliseconds (since 1970) that the engine sent a request to the token provider's OIDC UserInfo endpoint.
AUDIT.userInfoRespReceivedMillisec	The time in milliseconds (since 1970) that the engine received a response from the token provider's OIDC UserInfo endpoint.
AUDIT.userInfoRoundTripMS	The userInfoRespReceivedMillisec minus the userInfoReqSentMillisec time. This represents the total number of milliseconds it took PingAccess to receive a response from the token provider's UserInfo endpoint.

## Logging

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

- ➔ **Info:** Audit logs are also configurable in `conf/log4j2.xml`. These logs record a selected subset of transaction log information at runtime plus additional details (see [Security Audit Logging](#)).

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

- Changing the log file name:


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

- Setting the maximum log size:

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

- Setting the maximum number of log files:

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

-  **Tip:** The default log level is `DEBUG`. We recommend that once PingAccess is configured and is in use in a production environment, logging be configured to use the lowest, most appropriate level for your needs.

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

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

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

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



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

## Configure log levels

Define log levels for specific package or class names in order to get more (or less) logging from a class or group of classes. If the log level is not specified for a particular package or class, the settings for the root logger are inherited.

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

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

3. Modify the `level` attribute to set the desired log level.  
Valid values are `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`.
4. Save the modified file. PingAccess will automatically make the changes effective within 30 seconds.

## Configure a class or package log level

1. Open `conf/log4j2.xml` in an editor. Class or package loggers are defined in the `<AsyncLogger>` name attribute. For example, cookie logging is enabled using the line:

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

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

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

For example, to apply `TRACE` level logging for the `com.pingidentity` package, locate the following line:

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

and change it to:

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

3. Save the file.

## Enable cookie logging

Cookie logging is an optional feature in the `TRACE` log level.

1. Edit `conf/log4j2.xml` and uncomment the following section:

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

2. Save the file.

## Append log messages to syslog and the console

---

Additional output destinations (called *appenders*) are available. Console and syslog appenders are pre-configured in `log4j2.xml`, but are disabled by default.

To enable additional appenders, perform the following steps:

1. Open `conf/log4j2.xml` in an editor.
2. Locate the following lines in the `<Loggers>` element:

```
<AsyncLogger name="com.pingidentity" level="DEBUG" additivity="false"
  includeLocation="false">
  <AppenderRef ref="File"/>
  <!--<AppenderRef ref="CONSOLE" />-->
  <!--<AppenderRef ref="SYSLOG" />-->
</AsyncLogger>
```



**Note:** If you have customized logging to enable logging for additional classes, locate the `<AsyncLogger>` element that is relevant to the class in question. This class is defined in the `<AsyncLogger>` name attribute.

3. Uncomment the `<AppenderRef>` element that applies to the appender you wish to enable.



**Note:** PingAccess will rescan the logging configuration within 30 seconds and make the change active automatically.

4. Save the file.

## Write logs to other formats

---

PingAccess provides the option of writing the audit logs to an *Oracle or SQL Server database*.

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

### Write logs to databases

You can enable database logging for the API, engine, and agent audit logs in `conf/log4j2.db.properties`. Scripts are provided in `conf/log4j/sql-scripts` to create the necessary tables. PingAccess supports logging to Oracle, SQL Server, and PostgreSQL databases.

1. Ensure that your database driver JAR file is installed in the `PA_HOME/lib` directory. Restart PingAccess after installing the driver.
2. In `conf/log4j2.xml`, uncomment one or more of the preset appender configurations listed below:

- **Oracle**

- **For Administrative API audit logging:** Uncomment the `<JDBC>` element with the `name="ApiAuditLog-Database"` attribute specified, along with the following `<RollingFile>` and `<PingAccessFailover>` elements.



- **For Engine audit logging:** Uncomment the <JDBC> element with the name="EngineAuditLog-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
- **For Agent audit logging:** Uncomment the <JDBC> element with the name="AgentAuditLog-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
- **SQL Server**
  - **For Administrative API audit logging:** Uncomment the <JDBC> element with the name="ApiAuditLog-SQLServer-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
  - **For Engine audit logging:** Uncomment the <JDBC> element with the name="EngineAuditLog-SQLServer-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
  - **For Agent audit logging:** Uncomment the <JDBC> element with the name="AgentAuditLog-SQLServer-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
- **PostgreSQL**
  - **For Administrative API audit logging:** Uncomment the <JDBC> element with the name="ApiAuditLog-PostgreSQL-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
  - **For Engine audit logging:** Uncomment the <JDBC> element with the name="EngineAuditLog-PostgreSQL-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.
  - **For Agent audit logging:** Uncomment the <JDBC> element with the name="AgentAuditLog-PostgreSQL-Database" attribute specified, along with the following <RollingFile> and <PingAccessFailover> elements.



**Note:** The <PingAccessFailover> element is used to define how PingAccess logging fails over if a connection to the primary database is not accessible. Use the `retryIntervalSeconds` attribute to specify the number of seconds that must pass before retrying the primary JDBC appender.

3. In `conf/log4j2.db.properties`, replace the placeholder parameter values for each enabled appender with valid values to provide access to the database.



**Info:** You can obfuscate the password used to access the database by running either `obfuscate.sh` or `obfuscate.bat`, located in `PA_HOME/bin`. Use the database password as an argument, then copy the output into the password configuration property for the appender in `PA_HOME/conf/log4j2.db.properties`.

4. In `conf/log4j2.xml`, uncomment the <AppenderRef> elements in each respective <Logger> section as shown in the following examples:

#### Oracle:

```
<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
  <AppenderRef ref="APIAuditLog-File"/>
  <AppenderRef ref="ApiAuditLog-Database-Failover"/>
  <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>-->
  <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
  <AppenderRef ref="EngineAuditLog-File"/>
  <AppenderRef ref="EngineAuditLog-Database-Failover"/>
  <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>-->
  <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
</Logger>
```

```

<Logger name="agentaudit" level="INFO" additivity="false">
  <AppenderRef ref="AgentAuditLog-File"/>
  <AppenderRef ref="AgentAuditLog-Database-Failover"/>
  <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>-->
  <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
</Logger>

```

## SQL Server

```

<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
  <AppenderRef ref="APIAuditLog-File"/>
  <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
  <AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>
  <!--<AppenderRef ref="ApiAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
  <AppenderRef ref="EngineAuditLog-File"/>
  <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
  <AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>
  <!--<AppenderRef ref="EngineAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
  <AppenderRef ref="AgentAuditLog-File"/>
  <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
  <AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>
  <!--<AppenderRef ref="AgentAuditLog-PostgreSQL"/>-->
  <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
</Logger>

```

## PostgreSQL

```

<!-- Audit Log Configuration-->
<Logger name="apiaudit" level="INFO" additivity="false">
  <AppenderRef ref="APIAuditLog-File"/>
  <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
  <!--<AppenderRef ref="ApiAuditLog-SQLServer-Database-Failover"/>-->
  <AppenderRef ref="ApiAuditLog-PostgreSQL"/>
  <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
  <AppenderRef ref="EngineAuditLog-File"/>
  <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
  <!--<AppenderRef ref="EngineAuditLog-SQLServer-Database-Failover"/>-->
  <AppenderRef ref="EngineAuditLog-PostgreSQL"/>
  <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
  <AppenderRef ref="AgentAuditLog-File"/>
  <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
  <!--<AppenderRef ref="AgentAuditLog-SQLServer-Database-Failover"/>-->
  <AppenderRef ref="AgentAuditLog-PostgreSQL"/>
  <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
</Logger>

```

5. Create the database tables. Scripts to create database tables are located in `conf/log4j/sql-scripts`.



**Note:** The scripts are written to handle the default list of elements for the relevant database log appender. Any changes to the list require corresponding changes to the SQL table creation script, or to the table itself

if it already exists. For more information on working with these scripts, see Oracle, PostgreSQL, or MS SQL Server documentation.



**Important:** For PostgreSQL database scripts, use of the default **public** schema is not recommended. To run the scripts against a different schema, choose one of the following options:

- Prepend the schema before the table name. For example, `api_audit_log` would become `my_schema.api_audit_log`
- Run the script via **psql** and specify an options parameter to define the schema. For example:

```
psql postgresql://<user>@<db_hostname>:5432/<db_name>?options=--
search_path=<schema> -f api-audit-log-postgresql.sql
```

## Write 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 `conf/log4j2.xml`, uncomment the `<RollingFile>` and `<AppenderRef>` elements for the Splunk appenders you wish to enable:

```
<!--
<RollingFile name="ApiAudit2Splunk"
    fileName="${sys:pa.home}/log/pingaccess_api_audit_splunk.log"
    filePattern="${sys:pa.home}/log/pingaccess_api_audit_splunk.
%d{yyyy-MM-dd}.log"
    ignoreExceptions="false">
    <PatternLayout>
        <pattern>%d{ISO8601} exchangeId="%X{exchangeId}"
trackingId="%X{AUDIT.trackingId}" subject="%X{AUDIT.subject}"
authMech="%X{AUDIT.auth
Mech}" client="%X{AUDIT.client}" method="%X{AUDIT.method}"
requestUri="%X{AUDIT.requestUri}" responseCode="%X{AUDIT.responseCode}"
%n</pattern>
    </PatternLayout>
    <Policies>
        <TimeBasedTriggeringPolicy />
    </Policies>
</RollingFile>
-->
```



**Note:** `<RollingFile>` elements are also present for `EngineAudit2Splunk` and `AgentAudit2Splunk`. They are structured similarly to the `ApiAudit2Splunk` example shown above.

```
<Logger name="apiaudit" level="INFO" additivity="false">
    <AppenderRef ref="APIAuditLog-File"/>
    <!--<AppenderRef ref="ApiAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="ApiAudit2Splunk"/>-->
</Logger>
<Logger name="engineaudit" level="INFO" additivity="false">
    <AppenderRef ref="EngineAuditLog-File"/>
    <!--<AppenderRef ref="EngineAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="EngineAudit2Splunk"/>-->
</Logger>
<Logger name="agentaudit" level="INFO" additivity="false">
    <AppenderRef ref="AgentAuditLog-File"/>
    <!--<AppenderRef ref="AgentAuditLog-Database-Failover"/>-->
    <!--<AppenderRef ref="AgentAudit2Splunk"/>-->
</Logger>
```

2. Save the file.



**Note:** PingAccess automatically updates its configuration within 30 seconds; a restart is not required for this change to be effective.

3. Download and install the Splunk Universal Forwarder on the machine running PingAccess.

4. Configure the Universal Forwarder to monitor `logs/pingaccess_api_audit_splunk.log`, `logs/pingaccess_agent_audit_splunk.log`, or `logs/pingaccess_engine_audit_splunk.log`.



**Info:** For detailed installation and configuration instructions, consult the Splunk documentation accompanying the Universal Forwarder.

# Manage APIs

---

## 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.

### *Auth Token Management endpoint on page 110*

Endpoints used for Authentication Token Management.

### *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.

➔ **Info:** 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
<code>\$monitor.getTotalJvmMemory('bytes' 'KB' 'MB' 'GB')</code>	Returns the total memory in the JVM. Specify 'bytes', 'KB', 'MB', or 'GB' to specify the units. 'bytes' is the default if not specified.
<code>\$monitor.getUsedJvmMemory('bytes' 'KB' 'MB' 'GB')</code>	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
<code>\$monitor.getFreeJvmMemory('bytes' 'KB' 'MB' 'GB')</code>	Returns the free memory in the JVM. Specify 'bytes', 'KB', 'MB', or 'GB' to specify the units. 'bytes' is the default if not specified.
<code>\$monitor.getTotalPhysicalSystemMemory('bytes' 'KB' 'MB' 'GB')</code>	Returns the total system memory. Specify 'bytes', 'KB', 'MB', or 'GB' to specify the units. 'bytes' is the default if not specified.
<code>\$monitor.getTotalUsedPhysicalSystemMemory('bytes' 'KB' 'MB' 'GB')</code>	Returns the total used system memory. Specify 'bytes', 'KB', 'MB', or 'GB' to specify the units. 'bytes' is the default if not specified.
<code>\$monitor.getTotalFreePhysicalSystemMemory('bytes' 'KB' 'MB' 'GB')</code>	Returns the total free system memory. Specify 'bytes', 'KB', 'MB', or 'GB' to specify the units. 'bytes' is the default if not specified.
<code>\$monitor.getHostname()</code>	Returns the hostname for the system running PingAccess.
<code>\$monitor.getNumberOfCpus()</code>	Returns the number of CPU cores in the system.
<code>\$monitor.getCpuLoad('###.##')</code>	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 <a href="#">Java DecimalFormat</a> 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".
<code>\$monitor.getOpenClientConnections()</code>	Returns the current number of clients connected to PingAccess.
<code>\$monitor.getNumberOfVirtualHosts()</code>	Returns the current number of configured virtual hosts in PingAccess.
<code>\$monitor.getNumberOfApplications()</code>	Returns the current number of configured applications in PingAccess.
<code>\$monitor.getNumberOfSites()</code>	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 <b>Server Clustering</b> documentation. This value is not included in the default template, but can be added by the system administrator if desired.
<code>\$monitor.getLastRefreshTime('yyyy/MM/dd HH:mm:ss')</code>	Returns the time the PingAccess configuration was last refreshed. The parameter specifies the date format to use; if no value is specified, the <i>ISO 8601</i> date format is used. If the parameter is specified, the format used comes from the <i>Joda DateTimeFormat</i> 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.

- ➔ **Info:** 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 PingFederate documentation 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 be used in conjunction with the configuration of a JWT token processor instance in PingFederate. For more information on configuring a JWT, see PingFederate documentation.

### `/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.

## Auth Token Management endpoint

---

This page describes endpoints used for Authentication Token Management. These endpoints are available on the `engine.http.port` port defined in `PA_HOME/conf/run.properties`.

### `/pa/authtoken/JWKS`

This endpoint is used by backend sites to validate the signature of a JWT. For more information on the use of JWT, see the [OpenID Connect 1.0 Developers Guide](#).

## 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` at path `/pa-admin-api/v3/api-docs/` (`https://<PA_HOME>:<PORT>/pa-admin-api/v3/api-docs/`).



**Note:** For enhanced API security, you must include `X-XSRF-Header: PingAccess` in all requests and use the `application/json` content type for PUT/POST requests.



# Perform a zero downtime upgrade

## Introduction

This document describes the steps required to perform a zero downtime upgrade of a PingAccess cluster to version 5.0 or higher. A zero downtime upgrade allows you to upgrade your environment with no impact to resource availability or existing user sessions.

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

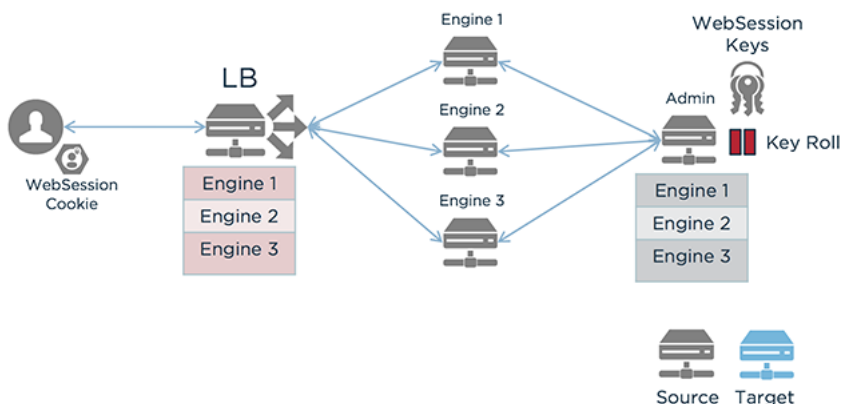
Note that there are some steps, particularly those related to working with a load balancer, that are dependent on your environment. It is expected that you are familiar with the tasks required by these steps and, accordingly, this document does not attempt to offer detailed instruction on performing these tasks.

**!** **Important:** In order to achieve a successful upgrade, perform the steps in this document in the order that they are presented. Deviation from these steps may result in a failed upgrade and/or system downtime.

To begin the upgrade process, [Disable key rolling](#) to prevent active sessions from being invalidated.

## Step 1: Disable key rolling

In this step, you will disable key rolling to prevent active sessions from being invalidated during the upgrade process. This is a temporary modification that you will address when the upgrade is complete. Note that there are different procedures at this stage depending if your source version is [PingAccess 5.0](#) or an [earlier version of PingAccess](#).



### Source version of PingAccess 5.0 and higher

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

### Source is an earlier version of PingAccess

1. Navigate to **Settings > Access > Identity Mappings**.
2. In the **Auth Token Management** section, specify a **Key Roll Interval** of 240 (10 days).

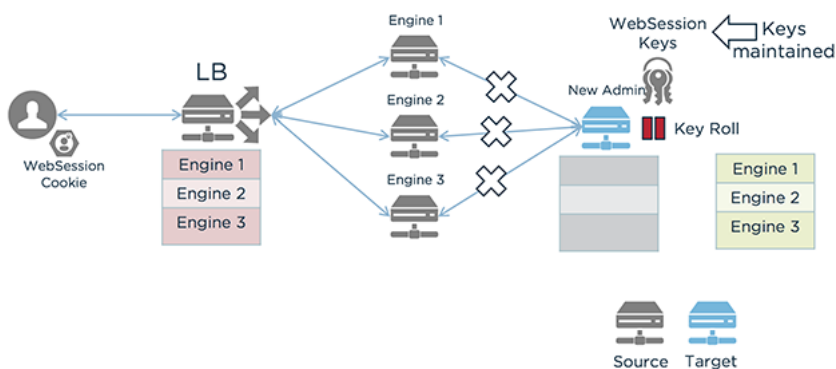
3. Click **Save**.
4. Navigate to **Settings > Access > Web Sessions**.
5. In the **Web Session Management** section, specify a **Key Roll interval** of 240 (10 days).
6. Click **Save**.

Next, you will [upgrade the Admin node](#).

## Step 2: Upgrade the Admin node

In this step, you will upgrade the PingAccess administration node using the PingAccess Upgrade Utility. As part of this step, you will use the `-r` switch to disable configuration replication on the target version.

For more information on upgrading PingAccess, see [Upgrade PingAccess](#).



### Prerequisites

Prior to beginning the upgrade process, make sure you have:

- Ensured PingAccess is running
- Downloaded and unpacked the PingAccess [Upgrade Utility](#)
- Downloaded the PingAccess [distribution ZIP file](#)
- The PingAccess license
- Administrator credentials
- Basic Authentication enabled

**Warning:** Failure to enable basic authentication can result in losing access to the Administration console.

### Upgrade the PingAccess administrative node

1. To run the Upgrade Utility, use a command line to change to the Upgrade Utility's `/bin` directory. For example:

```
cd /pa-upgrade-5.0.1/bin
```

2. Run the Upgrade Utility:

- Linux: `./run.sh -r <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`

For example:

```
./run.sh -r ../pingaccess-4.3.0 ../pingaccess-5.0.1 ../pingaccess-5.0.1.zip ../pingaccess.lic
```

- Windows: `run.bat -r <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`

For example:

```
run.bat -r ../pingaccess-4.3.0 ../pingaccess-5.0.1 ../
pingaccess-5.0.1.zip ../pingaccess.lic
```

**!** **Important:** The `-r` switch will disable configuration replication on the Admin node. You will re-enable configuration replication after the Admin and all engine nodes have been upgraded.

3. Stop the existing PingAccess admin instance.
4. Start the new PingAccess admin instance.
5. Upgrade the Admin Replica node using the same steps, but omitting the `-r` switch.

**!** **Important:** If PingAccess is running as a service:

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

After you have upgraded the Admin and Replica Admin nodes, you can begin [upgrading the engines](#).

## Step 3: Upgrade engines

---

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

**!** **Important:** It is imperative that you perform these steps on one engine at a time to maintain availability. Engines are identified by the engine name. Ensure that the engine that you remove from the load balancer aligns with the engine definition you import.

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

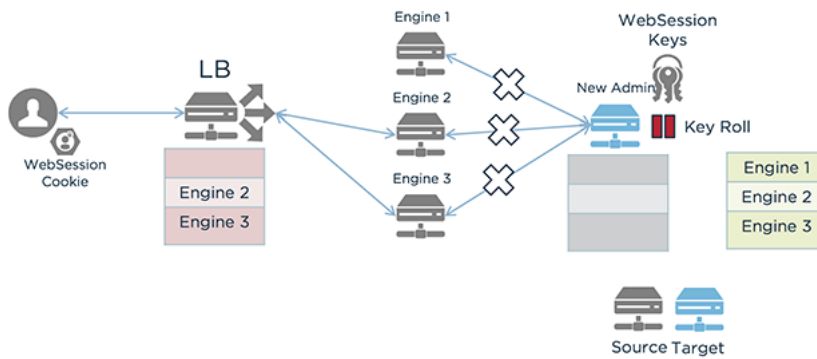
- [Remove the engine from the load balancer](#)
- [Upgrade the engine](#)
- [Resume configuration replication](#) on page 116
- [Add the engine to the load balancer](#)

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

### Remove the engine from the load balancer configuration

This step requires you to remove the engine from the load balancer configuration. Since this step is dependent on your environment, no specific instruction will be provided. It is assumed that you are familiar with the steps required to temporarily remove the engine from your load balancer configuration.

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



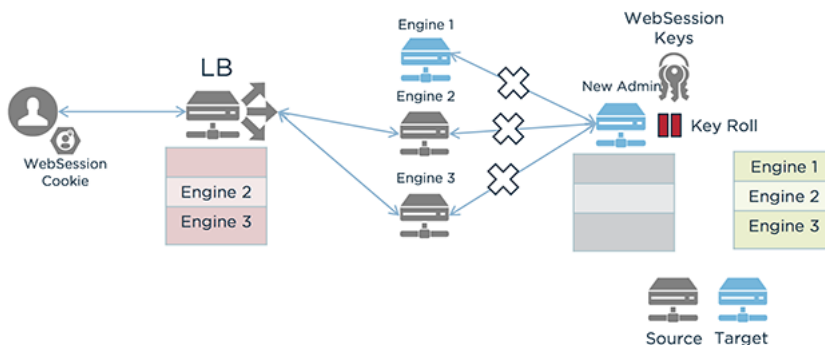
### Remove the engine from the load balancer configuration

1. Identify and note the engine you want to upgrade. Ensure you have the engine definition for this engine available.
2. Locate the entry for this engine in your load balancer configuration.
3. Depending on your environment, you may be removing this entry or you may be commenting the entry to remove it. Ensure that you make note of any entries you remove or modify so that you can reverse this operation later in [Add the engine to the load balancer configuration](#) on page 117.
4. Remove the engine entry from the load balancer.
5. Restart the load balancer.

### Upgrade the engine

In this step, you will use the PingAccess Upgrade Utility to upgrade the engine.

For more information on upgrading PingAccess, see [Upgrade PingAccess](#).



### Prerequisites

Prior to beginning the upgrade process, make sure you have:

- Ensured the PingAccess engine is running
- Downloaded and unpacked the PingAccess [Upgrade Utility](#)
- Downloaded the PingAccess [distribution ZIP file](#)
- The PingAccess license

### Upgrade the PingAccess engine

1. To run the Upgrade Utility, use a command line to change to the Upgrade Utility's `/bin` directory. For example:

```
cd /pa-upgrade-5.0.1/bin
```

2. Run the Upgrade Utility:

- Linux: `./run.sh <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`

For example:

```
/run.sh ../pingaccess-4.3.0 ../pingaccess-5.0.1 ../
pingaccess-5.0.1.zip ../pingaccess.lic
```

- Windows: `run.bat <sourcePingAccessRootDir> <outputDir> <pingaccessZip> <newPingAccessLicense>`

For example:

```
run.bat ../pingaccess-4.3.0 ../pingaccess-5.0.1 ../
pingaccess-5.0.1.zip ../pingaccess.lic
```

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

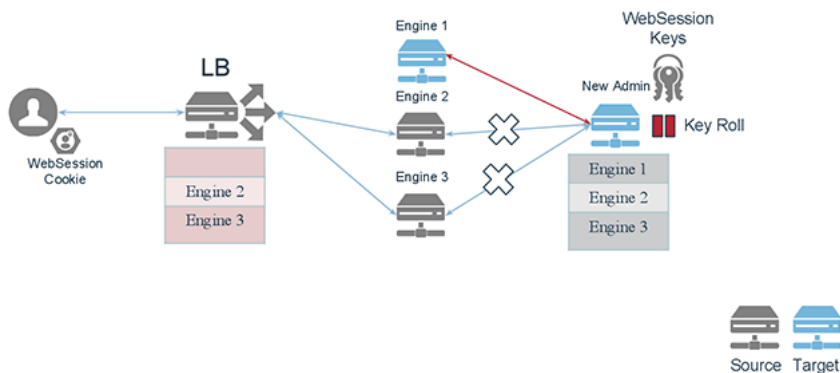


**Important:** If PingAccess is running as a service:

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

## Resume configuration replication

In this step, you will resume configuration replication that was disabled by the Upgrade Utility. You will perform this step for the Admin replica and all engines in the cluster. You will use the PingAccess Admin API to GET and PUT the relevant configuration data for each of these items.



### To resume configuration replication:




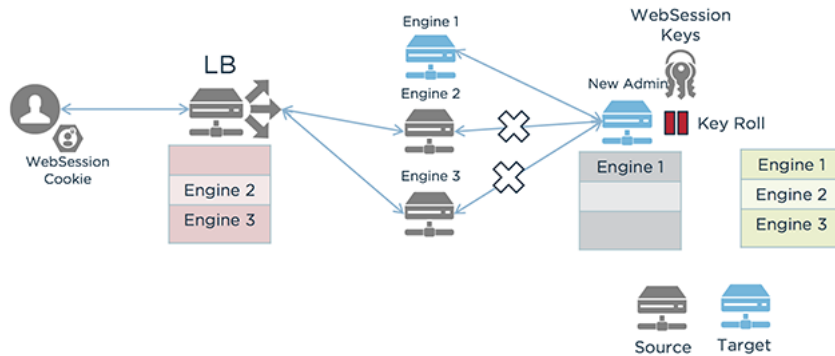
**Note:** Perform the following steps for the Replica Admin node and for each engine in the cluster.

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

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

11. Repeat the previous steps for each engine.
12. In the PingAccess administration console, navigate to **Settings > System > Clustering**.
13. Ensure the engines are displayed and reporting. A healthy engine shows a green status indicator.

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



14. Expand the `/adminConfig/replicaAdmins` endpoint.
15. Click the **GET /adminConfig/replicaAdmins** operation.
16. Click **Try it out!** and note the `id` for the replica admin.
17. Click the **GET /adminConfig/replicaAdmins/{id}** operation.
18. Enter the `id` of the replica admin you want to update and click **Try it out!**
19. Copy the **Response Body**.
20. Click the **PUT /adminConfig/replicaAdmins/{id}** operation and enter the `id` of the replica admin you want to update.
21. Paste the **Response Body** you copied and change `"configReplicationEnabled"` to `true`.
22. Click **Try it out!**

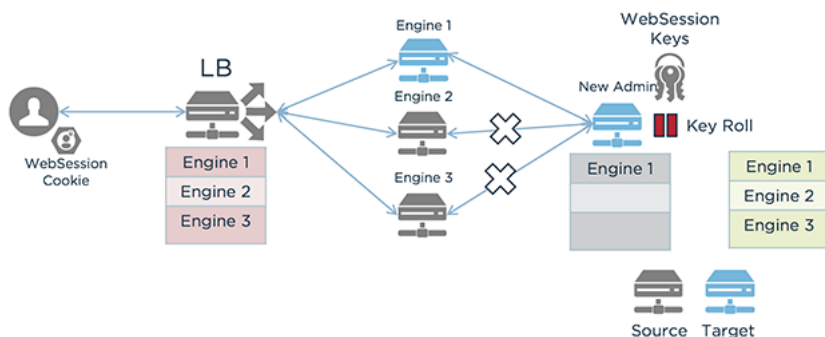
If the operation is successful, you will receive a **Response Code** of **200**.

23. In the PingAccess administration console, navigate to **Settings > System > Clustering**.
24. Ensure the **Replica Administrative Node** displayed and reporting on the **Administrative Nodes** screen. A healthy node shows a green status indicator.

## Add the engine to the load balancer configuration

This step requires you to add the engine back to the load balancer configuration. Since this step is dependent on your environment, no specific instruction will be provided. It is assumed that you are familiar with the steps required to add the engine back to the load balancer configuration.

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



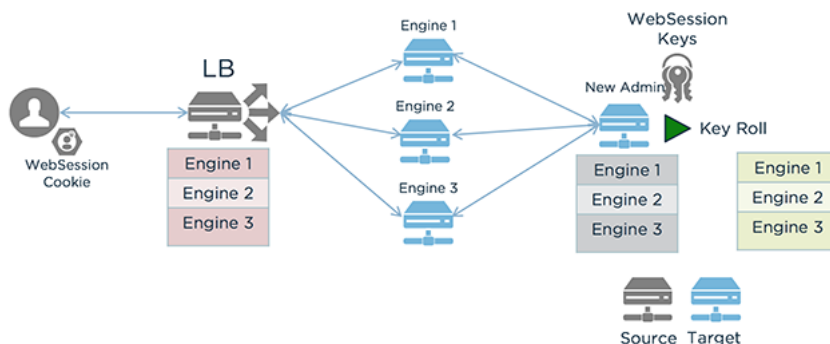
### Add the engine to the load balancer configuration

1. To add the engine to the load balancer configuration, simply reverse the steps you took in [Remove the engine from the load balancer configuration](#) on page 114 to remove the engine.
2. Restart the load balancer.

Repeat the [Step 3: Upgrade engines](#) on page 114 process until each engine has been upgraded. When all engines have been upgraded, added to the load balancer configuration, and are reporting to PingAccess, you can move on to the final step, [Enable key rolling](#), to complete the zero downtime upgrade process.

## Step 4: Enable key rolling

In this step, you will resume key rolling. Note that there are different procedures at this stage depending if your source version is [PingAccess 5.0](#) or an [earlier version of PingAccess](#).



### Source version of PingAccess 5.0 and higher

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

### Source is an earlier version of PingAccess

1. Navigate to **Settings > Access > Identity Mappings**.
2. In the **Auth Token Management** section, specify a **Key Roll Interval** of your choosing. The default value is 24 (1 day).
3. Click **Save**.
4. Navigate to **Settings > Access > Web Sessions**.

5. In the **Web Session Management** section, specify a **Key Roll interval** of your choosing. The default value is 24 (1 day).
6. Click **Save**.

## Recovering from a failed upgrade

---

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

To recover your PingAccess cluster in the event of a failure, you would utilize the former installation by:

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



# Protect a web-based application using PingFederate and PingAccess in a proxy deployment

---

## Protect a web-based application in a proxy deployment

---

This document describes the basic steps necessary to protect a web-based application using the combination of **PingAccess** as the access manager and **PingFederate** as the token provider.

Along with a description of the configuration steps that are required, this document includes a functioning example using the **PingAccess QuickStart** application. You can perform the example configuration to achieve a working result and become familiar with this solution, or you can substitute your own data.

Note that this is a basic configuration featuring:

- a PingFederate HTML Form Adapter with an instance of a Simple Password Credential Validator
- the PingFederate Access Token Manager (ATM) using the Internally Managed Reference Token data model

Your configuration may call for additional settings and components to satisfy network and/or security requirements. Links to additional resources that explain these additional settings and components are offered throughout this document.

This document does not detail the usage of identity mappings or authentication requirements that are common components of a typical configuration. As these components are simple to configure and beyond the scope of this document, you can read more about them and other features in the [PingAccess User Interface Reference Guide](#).

### Prerequisites

To complete the example configuration, the following prerequisites should be satisfied:

1. You have downloaded and performed a basic installation of [PingAccess 5.0](#) and [PingFederate 9.0](#).
2. You have downloaded the [PingAccess QuickStart \(Login required\)](#) application, unzipped the file, and copied the `\pingaccess-quickstart-5.0.0\pf-dist\PingAccessQuickStart.war` file to `\Program Files\Ping Identity\pingfederate-9.0.0\pingfederate\server\default\deploy`.
3. The PingAccess QuickStart application is hosted and available at `https://mypingfedserver:9031/PingAccessQuickStart`.

To begin, [Configure PingFederate for PingAccess connectivity](#) on page 120.

## Configure PingFederate for PingAccess connectivity

---

This document provides the sequence of steps for configuring PingFederate components for PingAccess connectivity as required for this solution. In this configuration procedure, you will:

1. [Enable PingFederate roles and protocols](#) on page 121
2. [Create a password credential validator](#) on page 121
3. [Configure an IdP adapter](#) on page 121
4. [Define the scope](#) on page 122
5. [Create an access token manager](#) on page 122
6. [Configure an IdP adapter mapping](#) on page 122
7. [Configure an access token mapping](#) on page 123
8. [Create an OpenID Connect policy](#) on page 123
9. [Create a resource server client](#) on page 123
10. [Create a web session client](#) on page 124
11. [Create and export a certificate](#) on page 124

**Important:**

1. These steps assume you have installed PingFederate. The example assumes that your PingFederate instance is available at `https://mypingfedserver`, using ports 9031 and 9999 respectively for the runtime and administration functions.
2. These steps assume you have installed PingAccess. This example assumes that your PingAccess instance is available at `https://mypingaccessserver` and that 3000 is the default listening port.

**Enable PingFederate roles and protocols**

Ensure that PingFederate is configured to respond to OAuth and OIDC requests using the following steps. For more information on PingFederate Roles and Protocols, see [Choose roles and protocols](#).

1. Navigate to **Server Configuration > System Settings > Server Settings**.
2. Click **Roles & Protocols** and ensure the following items are selected:
  - **OAuth** (role) and **OpenID Connect** (protocol)
  - **IdP** (role) and **SAML 2.0** (protocol)
3. On the **Federation Info** screen, enter the URL of your PingFederate environment and your SAML 2.0 entity ID. For example:
  - **My Base URL:** `https://mypingfedserver:9031`
  - **SAML 2.0 Entity ID:** `https://mypingfedserver/idp`
4. Click **Save**.

**Create a password credential validator**

Create a password credential validator, then create a username and password to authenticate with. For more information on Password Credential Validators, see [Manage password credential validators](#).

1. Navigate to **Server Configuration > Authentication > Password Credential Validators**.
2. Click **Create New Instance**.
3. Specify an **Instance Name** and **Instance ID** of your choosing. For example:

```
Instance Name: My_PCV
Instance ID: mypcv
```

4. In the **Type** list, select **Simple Username Password Credential Validator**.
5. On the **Instance Configuration** page, click **Add a new row to 'Users'**.
6. Create a **Username**, then create and confirm a **Password**.
7. Click **Update**.
8. On the **Summary** page, click **Done**.
9. Click **Save**.

**Configure an IdP adapter**

Configure an IdP adapter to look up session information and provide user identification to PingFederate. This example uses an instance of the HTML Form Adapter with an instance of the Simple Password Credential Validator. For more information, see [Manage IdP adapters](#).

1. Navigate to **IdP Configuration > Application Integration > Adapters**.
2. Click **Create New Instance**.
3. Specify an **Instance Name** and **Instance ID** of your choosing. For example:

```
Instance Name: My_IdP
Instance ID: myidp
```

4. In the **Type** list, select **HTML Form IdP Adapter**.
5. On the **IdP Adapter** page, click **Add a new row to 'Credential Validators'**.

6. In the **Password Credential Validator Instance** list, select the password credential validator you created (Example: *My\_PCV*) and click **Update**.
7. On the **Adapter Attributes** page, next to the `username` attribute, click to select the **Pseudonym** checkbox
8. On the **Summary** page, click **Done**.
9. Click **Save**.

### Define the scope

Use the Scope Management screen to define the default scope. For more information, see [Define scopes](#).

1. Navigate to **OAuth Settings > Authorization Server > Scope Management**.
2. Enter a description, for example: `default scope`.
3. Enter the following scope values and their descriptions one at a time, clicking **Add** with each entry:

Scope Value	Scope Description
address	address
email	email
openid	openid
phone	phone
profile	profile

4. Click **Save**.

### Create an access token manager

Create an access token that is used to grant access and control access parameters. This sample configuration uses an instance of the Access Token Manager (ATM) using the Internally Managed Reference Tokens data model. For more information, see [OAuth access token management](#).

1. Navigate to **OAuth Settings > Token & Attribute Mapping > Access Token Management**.
2. Click **Create New Instance**.
3. Specify an **Instance Name** and **Instance ID** of your choosing. For example:

```
Instance Name: General Access Token
Instance ID: GeneralAccessToken
```

4. In the **Type** list, select **Internally Managed Reference Tokens**.
5. On the **Access Token Attribute Contract** page, in the **Extend the Contract** field, enter `UserName` and click **Add**.
6. On the **Summary** page, click **Save**.

### Configure an IdP adapter mapping

Configure an IdP adapter mapping to map attributes. For more information, see [Manage IdP adapter mappings for OAuth](#).

1. Navigate to **OAuth Settings > Token & Attribute Mapping > IdP Adapter Mapping**.
2. In the **Source Adapter Instance** list, select the adapter you created.
3. Click **Add Mapping**.
4. On the **Contract Fulfillment** page, in the **Source** lists, select **Adapter** for both `USER_KEY` and `USER_NAME`.
5. In the **Value** lists, select `username` for both items.
6. On the **Summary** page, click **Save**.

## Configure an access token mapping

Configure an access token mapping to map attributes to be requested from the OAuth resource server with the access token. For more information, see [Manage access token mappings](#).

1. Navigate to **OAuth Settings > Token & Attribute Mapping > Access Token Mapping**.
2. In the **Context** list, select `Default` or select your IdP adapter instance.
3. In the **Access Token Manager** list, select the access token you created. For example:

```
GeneralAccessToken
```

4. Click **Add Mapping**.
5. On the **Contract Fulfillment** page, in the **Source** list, select `Persistent Grant`.
6. In the **Value** list, select `USER_KEY`.
7. On the **Summary** page, click **Save**.

## Create an OpenID Connect policy

Configure an OpenID Connect policy so you can define OpenID Connect policies for client access to attributes mapped according to OpenID specifications. For more information, see [Configure OpenID Connect policies](#).

1. Navigate to **OAuth Settings > Token & Attribute Mapping > OpenID Connect Policy Management**.
2. Click **Add Policy**.
3. Specify a **Policy ID**, for example: `OIDC`
4. Specify a **Name**, for example: `OIDC`
5. In the **Access Token Manager** list, select the access token you created. For example:

```
GeneralAccessToken
```

6. On the **Attribute Contract** page, delete all items beneath **Extend the Contract**.
7. On the **Contract Fulfillment** page, in the **Source** list, select `Access Token`.
8. In the **Value** list, select `UserName`.
9. On the **Summary** page, click **Done**.
10. Click **Save**.
11. Beside the policy you created, click **Set as Default**.
12. Click **Save**.

## Create a resource server client

Configure an OAuth client for use with PingFederate token provider resource server configuration in PingAccess. For more information, see [Manage OAuth clients](#).

1. Navigate to **OAuth Settings > Clients > Manage All**.
2. Click **Add Client**.
3. Specify a **Client ID**. For example:

```
pa_rs
```

4. Specify a **Name**. For example:

```
PingAccessResourceServer
```

5. Select **Client Secret**.
6. Generate a secret by clicking **Generate Secret**. Copy this secret to a secure location so that you can use it in PingAccess configuration.
7. In the **Redirection URIs** field, add the OIDC callback redirect to the PingAccess server. For example:

```
https://mypingaccessserver:3000/pa/oidc/cb
```

8. Click **Add**.
9. For the **Allowed Grant Type** setting, select the **Access Token Validation (Client is a Resource Server)** check box.
10. Click **Save**.
11. Click **Save**.

### Create a web session client

Configure an OAuth client for use with web session configuration in PingAccess. For more information, see [Manage OAuth clients](#).

1. Navigate to **OAuth Settings > Clients > Manage All**.
2. Click **Add Client**.
3. Specify a **Client ID**, for example:

```
pa_wam
```

4. Specify a **Name**. For example:

```
PingAccessWebAccessManagement
```

5. Select **Client Secret**.
6. Generate a secret by clicking **Generate Secret**. Copy this secret to a secure location so that you can use it in PingAccess configuration.
7. In the **Redirection URIs** field, add the OIDC callback redirect to the PingAccess server, for example:

```
https://mypingaccessserver:3000/pa/oidc/cb
```

8. Click **Add**.
9. Select the **Bypass Authorization Approval** check box.
10. For the **Allowed Grant Type** setting, select the **Authorization Code** check box.
11. Click **Save**.
12. Click **Save**.

### Create and export a certificate

Create and export a certificate for the PingFederate server that you will import to PingAccess to establish trust. For more information, see [Manage SSL server certificates](#).

1. Navigate to **Server Configuration > Certificate Management > SSL Server Certificates**.
2. Click **Create New**.
3. In the **Common Name** field, enter the PingFederate server address. For example:

```
mypingfedserver
```


4. Complete the remaining fields as required.
5. Click **Next**.
6. Click **Done**.
7. Under the **Action** heading, click **Activate for Runtime Server**.
8. **Export** the certificate only.
9. Click **Save**.

Next, [Connect to PingFederate and configure an application in PingAccess](#) on page 125.

## Connect to PingFederate and configure an application in PingAccess

This document provides the sequence of steps for configuring the PingAccess components required for this solution. In this configuration procedure, you will:

1. [Import certificates and create a trusted certificate group](#) on page 125
2. [Configure the token provider](#) on page 125
3. [Create a web session](#) on page 126
4. [Create a virtual host](#) on page 126
5. [Create a site](#) on page 127
6. [Create an application](#) on page 127

 **Important:** The examples in this procedure assume that the PingAccess QuickStart application is available at the following address:

```
https://mypingfedserver:9031/PingAccessQuickStart
```

### Import certificates and create a trusted certificate group

Import a certificate for the PingFederate server to establish trust. For more information, see [Certificates](#).

1. Navigate to **Settings > Security > Certificates**.
2. Click + to the right of the **Certificates** subheading.
3. Enter an **Alias** for the certificate. For example:

```
PingFed
```

4. Click **Choose File** to select the certificate.
5. Click **Add** to import the certificate. A new certificate row appears on the Certificates page.
6. Click + to the right of the **Trusted Certificate Groups** heading.
7. Drag a certificate onto the box that appears.
8. Enter a **Name** for the group in the box that appears. For example:

```
PingFed
```

9. Click **Add**.

### Configure the token provider

Establish communication with the token provider, PingFederate. For more information, see [Manage Token Provider](#).

1. Navigate to **Settings > System > Token Provider > Runtime**.
2. Enter the **Host** name or IP address for the PingFederate Runtime. For example:

```
mypingfedserver
```

3. Enter the **Port** number for PingFederate Runtime. For example:

```
9031
```

4. Select **Secure**.
5. From the **Trusted Certificate Group** list, select the PingFed certificate group.
6. Click **Save**.
7. Navigate to **Settings > System > Token Provider > Administration**.

8. Enter the **Host** name or IP address for access to the PingFederate Administrative API. For example:

```
mypingfedserver
```

9. Enter the **Port** number for access to the PingFederate Administrative API. For example:

```
9999
```

10. Enter the PingFederate **Admin Username**.

11. Enter the **Admin Password**.

12. Select **Secure**.

13. From the **Trusted Certificate Group** list, select the PingFed certificate group.

14. Click **Save**.

15. Navigate to **Settings > System > Token Provider > OAuth Resource Server**

16. Enter the OAuth **Client ID** you defined when creating the PingAccess OAuth client in PingFederate. For example:

```
pa_rs
```

17. Enter the **Client Secret** you defined when creating the PingAccess OAuth client within PingFederate.

18. In the **Subject Attribute Name** field, enter the attribute you want to use from the OAuth access token as the subject for auditing purposes. For example:

```
username
```

19. Click **Save**.

## Create a web session

Create a web session to control access. For more information, see [Web Sessions](#).

1. Navigate to **Settings > Access > Web Sessions**

2. On the Web Session page, click **Add Web Session**.

3. Enter a unique **Name** for the web session, up to 64 characters, including special characters and spaces. For example:

```
PF_WAM
```

4. Select **Encrypted JWT** for **Cookie Type**.

5. Specify the **Audience** that the PA Token is applicable to, represented as a short, unique identifier between 1 and 32 characters. For example:

```
global
```

6. Specify the **OpenID Connect Login Type** **CODE**.

7. Specify the **Client ID**. For example:

```
pa_wam
```

8. Specify the **Client Secret**

9. Click **Save**.

## Create a virtual host

Create a virtual host that PingAccess will respond to. For more information, see [Virtual Hosts](#).

1. Navigate to **Settings > Access > Virtual Hosts**.

2. Click **Add Virtual Host**.

3. Enter the **Host** name for the Virtual Host. For example:

```
mypingaccessserver
```

4. Enter the **Port** number for the Virtual Host. For example:

```
3000
```

5. Click **Save**.

### Create a site

Create a site to define the location of the application that PingAccess is protecting. For more information, see [Sites](#).

1. Navigate to **Main > Sites > Sites**.
2. Click **Add Site**.
3. Specify a **Name**. For example:

```
QuickStart
```

4. Specify the **Target**. The format for this is `hostname:port`. For example:

```
mypingfedserver:9031
```

5. Select **Secure** and choose the PingFed **Trusted Certificate Group**.
6. Click **Save**.



**Note:** If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

### Create an application

Create an application to define the resource that PingAccess is protecting. For more information, see [Applications](#).

1. Navigate to **Main > Applications**.
2. Click **Add Application**.
3. Provide a unique name for the application. For example:

```
QuickStart
```

4. Specify the context at which the application is accessed at the site. For example:

```
/PingAccessQuickStart
```

5. Specify the virtual host for the application. For example:

```
mypingaccessserver:3000
```

6. Specify the application type **Web** and select the **Web Session** for the application. For example:

```
PF_WAM
```

7. Specify the application destination type **Site** and select the **Site** requests are sent to when access is granted. For example:

```
PingAccessQuickStart
```

8. Select the **Require HTTPS** option.
9. Select the **Enabled** checkbox.
10. Click **Save**.



## Test the configuration

---

To test the solution after configuration is complete:

1. Open your browser.
2. Navigate to the application using a combination of the Virtual Host and Context Root that you defined in PingAccess. For example:

```
https://mypingaccessserver:3000/PingAccessQuickStart
```

3. Login using the credentials you created for the PingFederate *password credential validator*.

# Customize and localize PingAccess

## Customization of 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_HOME/conf/template` directory.

You can modify most of these pages in a text editor to suit the particular branding and informational needs of your PingAccess installation. (Cascading style sheets and images for these pages are included in the `PA_HOME/conf/static/pa/assets` subdirectory.) Each page contains both Velocity constructs and standard HTML. The Velocity engine interprets the commands embedded in the template page before the HTML is rendered in the user's browser. At runtime, the PingAccess server supplies values for the Velocity variables used in the template.

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.
exchangeId	A value that identifies the request/response pair. This can be used to locate messages in the PingAccess logs.
trackingId	A value that identifies either the tracking ID (identified with a <code>tid:</code> prefix) or an Access Token ID (identified with a <code>atid:</code> prefix). This can be used to identify the session in the PingAccess and PingFederate logs.

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
<code>admin.error.page.template</code>	Indicates an error occurred while the admin console was processing a request	Error	Consult <code>PA_HOME/log/pingaccess.log</code> to determine the underlying cause of the issue.
<code>general.error.page.template</code>	Indicates that an unknown error has occurred and provides an error message.	Error	Consult <code>PA_HOME/log/pingaccess.log</code> to determine the underlying cause of the issue.
<code>general.loggedout.page</code>	Displayed when a user logs out of PingAccess.	Normal	User should close the browser.
<code>oauth.error.json</code>	Indicates that Rule evaluation has failed and provides an optional error message. To customize	Normal	If necessary, consult the audit logs in <code>PA_HOME/log</code> for details about

Template File Name	Purpose	Type	Action
<code>policy.error.page.template</code>	Indicates that Rule evaluation has failed and provides an optional error message. To customize this information, see Error-Handling Fields for OAuth Rules documentation.	Normal	why the policy denied the request.  If necessary, consult the audit logs in <code>PA_HOME/log</code> for details about why the policy denied the request.



**Note:** The templates stored in `PA_HOME/conf/template/system` are system templates, and should not be modified.

## Localization of user-facing pages

In addition to the use of Velocity templates to change the look and feel of user-facing pages, administrators can provide localized versions of user-facing status messages generated by PingAccess.

In `PA_HOME/conf/localization/`, properties files contain the messages to be returned to the client in various languages; by default, only English language messages are provided, using the default `pa-messages.properties` file. This file serves as a fallback for any message not found in other files in the directory.

The selection of a messages file is determined based on several different factors:

- The browser's `Accept-Language` header, based on a best-match first check against the `pa-messages` files
- The value of a cookie named `ping-accept-language`, which can be defined by the protected application
- A custom-developed PingAccess add-on that can customize the order of localization resolution

The default behavior allows the `ping-accept-language` cookie to override the browser preferences, and if that cookie is not set, then to use the `Accept-Language` header preference order, starting with the highest priority preference and trying to match the locale exactly. If none of the specified locales cannot be matched exactly, a more generic locale will be used, starting with the highest priority value.

If no matches are found, then the value in the `pa-messages.properties` file is used.

For example, suppose your browser had the following `Accept-Language` header:

```
Accept-Language: fr-CA;q=0.9, en-US;q=0.8
```

and PingAccess attempted to display a localized version of the message for:

```
pa.response.status.service.unavailable
```


The order in which PingAccess searches for the string to display is:

1. `pa-messages_fr_CA.properties`
2. `pa-messages_en_US.properties`
3. `pa-messages_fr.properties`
4. `pa-messages_en.properties`
5. `pa-messages.properties`

If the `ping-accept-language` cookie is set by the protected application to the value `en-US`, then the above list would be ignored, and PingAccess would search for the string in:

1. `pa-messages_en_US.properties`

2. `pa-messages_en.properties`
3. `pa-messages.properties`

 **Important:** Most browsers support the use of an ordered list of languages, however, Safari is an exception to this. Even though the system supports an ordered list of languages, only the preferred language is sent with its requests.

## Groovy development guide

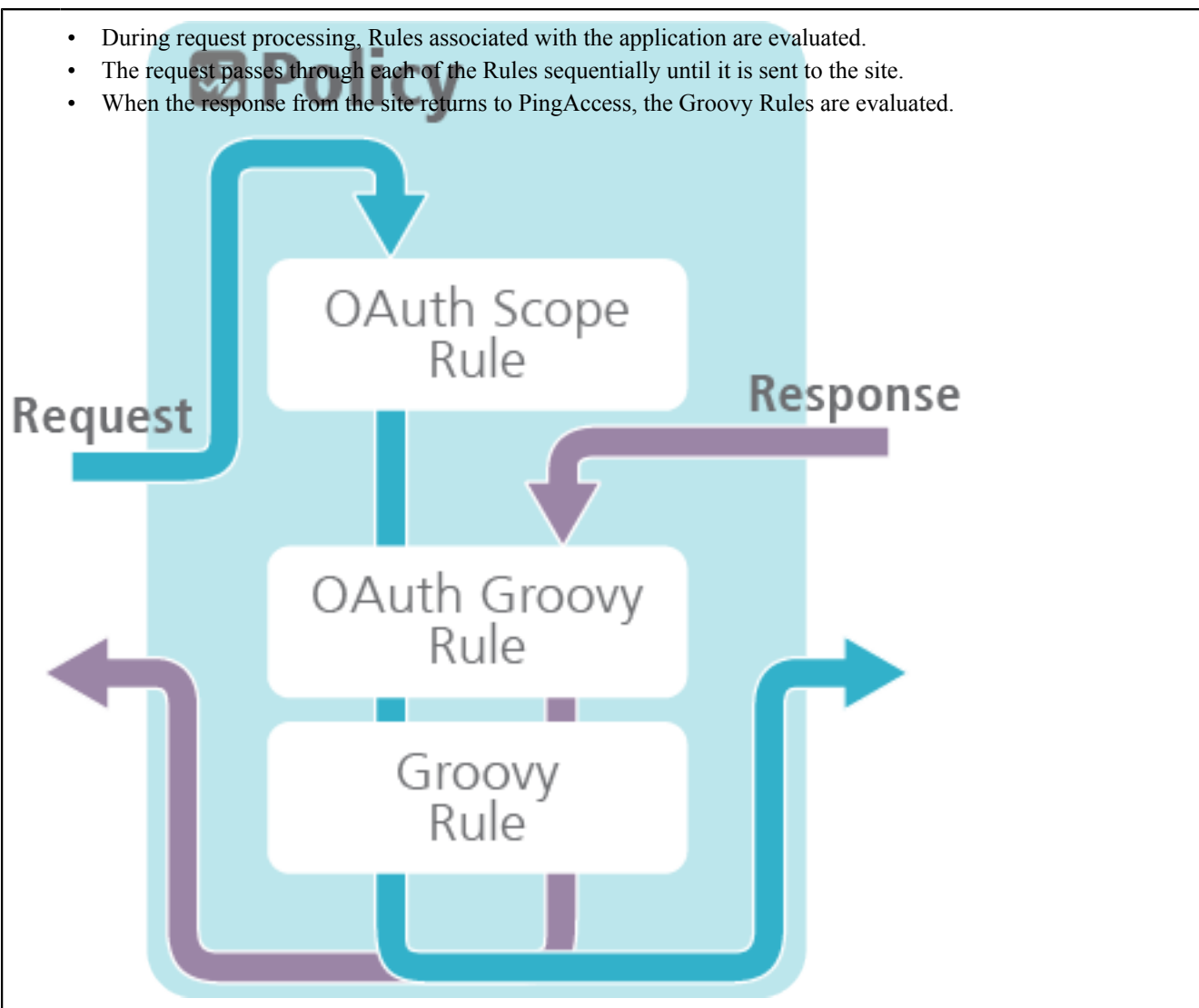
---

### 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.



## 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.



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

### Matchers

Groovy scripts must end execution with a *Matcher* instance. Matchers provide a framework for establishing declarative Rule matching objects. You can use a *Matcher* from the list of *PingAccess matchers* or from the *Hamcrest library*.

The following are Hamcrest method examples for constructing access control policies with the Web Session Attribute Rule using evaluations such as an OR group membership evaluation.

**allOf** - Matches if the examined object matches ALL of the specified matchers. In this example, the user needs to be in both the sales and managers groups for this rule to pass.

```
allOf(containsWebSessionAttribute("group", "sales"),
      containsWebSessionAttribute("group", "managers"))
```

**anyOf** - Matches any of the specified matchers. In this example, the rule passes if the user is in any of the specified groups.

```
anyOf(containsWebSessionAttribute("group", "sales"),
      containsWebSessionAttribute("group", "managers"),
      containsWebSessionAttribute("group", "execs"))
```

**not** - Inverts the logic of a matcher to not match. In this example, the rule fails if the user is in both the sales and the managers groups.

```
not(allOf(containsWebSessionAttribute("group", "sales"),
          containsWebSessionAttribute("group", "managers")))
```

See *Matchers* for more information.

### Objects

The following objects are available in Groovy. 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.

### *Logger Object*

Configure and view the state of logging.

### *MediaType Object*

Contains information related to the media type.

## **Debugging/troubleshooting**

Groovy Script Rules are evaluated when saved to ensure that they are syntactically valid. If a Groovy Script Rule fails to save, 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.], {}]

```

➔ **Info:** These error messages are only logged if the DEBUG level output is enabled for the `com.pingidentity` logger.

## **Body object**

---

### **Purpose**

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.

## Exchange object

### Purpose

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"))
}
```

## Method summary

Method	Description
Identity getIdentity()	Obtains the PingAccess representation of the identity associated with the request. This object will be null for requests to an unprotected application or an unauthenticated request to an anonymous resource.
Request getRequest()	Obtains the PingAccess representation of the request. This request is sent to the site with any changes that might be made in a Groovy script.
Response getResponse()	Obtains the PingAccess representation of the response. If the site has not been called, the response is null. This field is not available in scripts used with an Agent.



Method	Description
long getTimeReqSent()	Obtains the time, in milliseconds, when the request was sent to the site. This field is not available in scripts used with an Agent.
long getTimeResReceived()	Obtains the time, in milliseconds, when the response was received from the site. This field is not available in scripts used with an Agent.
String getRequestURI()	Returns the PingAccess URI that received the request.
String getRequestScheme()	Obtains the scheme used by the browser or other user agent that made the request.
Object getProperty(String key)	Returns the value of a custom property.
void setProperty(String key, Object value)	Sets a custom property.
SslData getSslData()	Obtains information established in the TLS handshake made with PingAccess.

## Headers object

### Purpose

Accesses the Headers object in Groovy `exc?.request?.header` or `exc?.response?.header`

The Headers object contains the HTTP Header information from the request made to an application or the HTTP Header from a site response. The *Request* HTTP Header is sent on to the site after the Rules are evaluated. The *Response* HTTP Header is returned to the client after the response Rules are evaluated.

Use the Headers object to add custom HTTP headers for site.

### Groovy sample

```
//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.  ➔ <b>Info:</b> 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.

Method	Description
void setAuthorization(String value)	Sets authentication credentials for HTTP Authentication.
String getConnection()	Returns the connection type preferred by the User-Agent.
void setConnection(List<String> values)	Sets the connection type preferred by the User-Agent.
int getContentLength()	Returns the request body content length.
void setContentLength(int length)	Sets the request body content length.
MediaType getContentType()	Returns media type of Header with Content type
void setContentType(String)	Sets the request body MIME type.
Map <String, String[]> getCookies()	Returns all cookies sent with the request.
void setCookie(String)	Sets a cookie.
String getFirstCookieValue(String)	Returns the first cookie in the Cookie header.
String getFirstValue(String)	Returns the first value of the HTTP header specified by the name.
void setDate(Date date)	Sets the date of the message in the Date HTTP header.
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.
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.

## Identity object

---

### Purpose

The Identity object contains information about the authenticated identity associated with the current HTTP request.

### Groovy sample

```
// Only allow access for an identity with subject "user"
def subject = exc?.identity?.subject

if ("user".equals(subject)) {
    pass()
} else {
    fail()
}
```

### Method summary

Method	Description
String getSubject()	Returns the subject of the identity.
String getMappedSubject()	Returns the subject set by the identity mapping. If there is no identity mapping associated with the application, the return value will be null. If there is an identity mapping associated with the application, but the identity mapping did not determine a subject to map, the returned value may be the empty string.
String getTrackingId()	Returns the tracking identifier used in PingAccess logs. This value is not guaranteed to be globally unique and should be used for diagnostic purposes only
String getTokenId()	Returns the unique ID for the associated authentication token. This value may change when new tokens are issued for the same identity.
Date getTokenExpiration()	Returns a Date object representing the time at which the authentication token expires. This may be null if the authentication provider did not indicate an expiry.
JsonNode getAttributes()	Returns a JsonNode object representing the attributes of the identity.

## JsonNode object

---

### Purpose

The JsonNode object represents the attributes of an identity.

### Groovy sample

```
// Only allow access if the user is in the group "staff"
def groups = exc?.identity?.attributes?.get("groups")
```

```

foundGroup = false
if (groups) {
    for (group in groups) {
        if ("staff".equals(group.asText())) {
            foundGroup = true
            break
        }
    }
}

if (foundGroup) {
    pass()
} else {
    fail()
}

```

### Method summary

Method	Description
JsonNode get(String fieldName)	Gets the JsonNode representing a field of this JsonNode. This method will return null if no field exists with the specified name.
boolean has(String fieldName)	Returns true if this JsonNode has a field with the specified name.
java.util.Iterator<String> fieldNames()	Returns an java.util.Iterator providing access to the names of all the fields of this JsonNode.
boolean isTextual()	Returns true if this JsonNode represents a string value.
String asText()	Returns a string representation of this JsonNode. If this JsonNode is an array or object, this will return an empty string.
int intValue()	Returns an integer representation of this JsonNode. If this JsonNode does not represent a number, 0 is returned.
boolean isArray()	Returns true if this JsonNode is an array.
boolean isObject()	Returns true if this JsonNode is an object.
int size()	For an array JsonNode, returns the number of elements in the array. For an object JsonNode, returns the number of fields in the object. 0 otherwise.
java.util.Iterator<JsonNode> iterator()	Returns an java.util.Iterator over all JsonNode objects contained in this JsonNode. For an array JsonNode, the returned java.util.Iterator will iterate over all the elements in the array. For an object JsonNode, the returned java.util.Iterator will iterate over all field values in the object.

### Remarks

A JsonNode implements java.lang.Iterable<JsonNode> so a for loop can be used to iterate over all the elements in an array JsonNode or the field values in an object JsonNode.

## Logger object

---

### Purpose

Accesses the Logger object.

### Method summary

Method	Description
void trace(String format, Object... arguments)	Logs a TRACE level message based on the specified format and arguments.
void debug(String format, Object... arguments)	Logs a DEBUG level message based on the specified format and arguments.
void info(String format, Object... arguments)	Logs an INFO level message based on the specified format and arguments.
void warn(String format, Object... arguments)	Logs a WARN level message based on the specified format and arguments.
void error(String format, Object... arguments)	Logs an ERROR level message based on the specified format and arguments.
boolean isTraceEnabled()	Checks if the logger instance is enabled for the TRACE level.
boolean isDebugEnabled()	Checks if the logger instance is enabled for the DEBUG level.
boolean isInfoEnabled()	Checks if the logger instance is enabled for the INFO level.
boolean isWarnEnabled()	Checks if the logger instance is enabled for the WARN level.
boolean isErrorEnabled()	Checks if the logger instance is enabled for the ERROR level.

## MediaType object

---

### Purpose

Accesses the MediaType object.

### Method summary

Method	Description
Map getParameters()	Returns a list of parameters.
String getBaseType()	Returns the media base type.
String getSubType()	Returns the media sub type.
String getParameter(String)	Returns a string containing the value of the request parameter.
String getPrimaryType()	Returns the primary media type.

## Method object

---

### Purpose

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()
}
```

### Method summary

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

## OAuth Token object

---

### Purpose

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()
```

### Method summary

Method	Description
Instant getExpiresAt()	Contains the expiration instant of the OAuth access token.
Instant getRetrievedAt()	Contains the instant that the OAuth access token was retrieved from PingFederate.
String getTokenType()	Contains the type of OAuth access token. (Bearer, JWT).
String getClientId()	Contains the client ID associated with the OAuth access token.
Set getScopes()	Contains the set of scopes associated with the OAuth access token.
Map<String, List<String>> getAttributes()	Contains a map of identity attributes specific to the user.

## PolicyContext object

---

### Purpose

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")
```

### Method summary

Method	Description
Map<String, Object> getContext()	Container for the <i>OAuthToken object</i> .
Exchange getExchange()	Returns the exchange a message relates to.

## Request object

---

### Purpose

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
def contentType = request?.headers?.getContentType()
def containsJson = contentType?.matchesBaseType("application/json")
//Check to make sure the request body contains JSON
if (!containsJson) {
  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.
void setUri(String)	Sets the PingAccess URI.

## Method summary

Method	Description
<i>Method</i> getMethod	Contains the HTTP method information from the request sent to the application.
<i>Header</i> getHeader	Contains the HTTP header information from the request sent to the application.  <b>Warning: Warning:</b> Previously executed custom Rules can modify these values.
<i>Body</i> getBody	Contains the HTTP body information from the request sent to the application. This field is not available in scripts used with an Agent.  <b>Warning: Warning:</b> Previously executed custom Rules can modify these values.
Map<String, String[]> getRequestParams() throws java.net.URISyntaxException	Parses the query string parameters from the request. If the query string parameters cannot be parsed due to formatting errors, this method will throw a java.net.URISyntaxException. Groovy scripts that use this method are not required to catch this exception. Scripts that choose not to catch this exception will fail if the query string parameters are invalid.
Map<String, String[]> getPostParams()	Parse the form parameters from the body content of the request, assuming the content is encoded using the encoding defined by the application/x-www-form-urlencoded content type.
void setBodyContent(byte[] content)	Replaces the body content of the request. This method will also adjust the Content-Length header field to align with the length of the specified content.



## Response object

### Purpose

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?.getStatusCode() == 500)
{
    not(anything("A server error occurred"))
}
else
{
    anything()
}
```

### Field summary

Field	Description
<code>int getStatusCode()</code>	Contains the HTTP response status code.
<code>void setStatusCode(int)</code>	Sets the status code from an integer.
<code>String getStatusMessage()</code>	Contains the HTTP response status message.
<code>void setStatusMessage(String)</code>	Sets the status message from a string.

### Method summary

Method	Description
<code>boolean isRedirect()</code>	Returns true if the status code is in the 300's.
<i>Header</i> <code>getHeader</code>	Contains the HTTP header information from the response.   <b>Warning:</b> Previously executed custom Rules can modify these values.
<i>Body</i> <code>getBody</code>	Contains the HTTP body information from the response.   <b>Warning:</b> Previously executed custom Rules can modify these values.
<code>void setBodyContent(byte[] content)</code>	Replaces the body content of the response. This method will also adjust the Content-Length header field to align with the length of the specified content.

## SslData object

---

### Purpose

The SslData object provides access to information established in the TLS handshake with PingAccess.

### Groovy sample

```
// Force TLS client authentication
def certChain = exc?.sslData?.clientCertificateChain
if(certChain && !certChain.isEmpty())
{
    pass();
}
else
{
    fail();
}
```

### Method summary

Method	Description
List<String> getSniServerNames()	Returns a list of SNI server_names sent by the user agent in the TLS handshake. Empty if the user agent did not utilize the SNI TLS extension.
List<java.security.cert.X509Certificate> getClientCertificateChain()	Returns the certificate chain presented by the user agent in the TLS handshake. Empty if the user agent did not utilize TLS client authentication.

## Groovy script examples

---

### 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")
}
```

Set an exchange property named "com.pingidentity.policy.error.info" so the value will be available in \$info variable in policy.error.page.template.html for Web applications.

```
exc?.setProperty("com.pingidentity.policy.error.info", "this value will be
  passed to the template in $info variable")
not(anything())
```

Create a white listing rule for certain characters.

```
if (!exc?.request?.uri?.matches("[\\p{Po}\\p{N}\\p{Z}\\p{L}\\p{M}\\p{Zs}\\./
_\\-\\(\\)\\{\\}\\[\\]}*"))
{
  fail()
}
else
{
  pass()
}
```

Add a cookie to the response.

```
List cookies = new java.util.ArrayList()
com.pingidentity.pa.sdk.http.SetCookie cookieRep = new
  com.pingidentity.pa.sdk.http.SetCookie()
// Set the cookie name and value along with the path and as secure for the
  Response
cookiecookieRep.name("ResponseTestCookie")
cookieRep.value("this is on response")
cookieRep.secure()
cookieRep.path("/")
// Add the cookie to the cookies array
cookies?.add(cookieRep.toHeaderField())
// Set the cookie on to the response
exc?.response?.header?.setSetCookie(cookies)
```

Comb an AND and OR, invoking an existing rule matcher.

```
if ((anyOf(containsWebSessionAttribute("engineering",
  "true"), containsWebSessionAttribute("marketing", "true"))) &&
  (containsWebSessionAttribute("manager", "true")))
{pass()
}
else{
fail()
}
```

## 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
<code>inIpRange(String cidr)</code>	<p>Validates the source IP address of the request against the <code>cidrString</code> parameter in CIDR notation. When Source IP headers defined in the <a href="#">HTTP Requests</a> page are found, the source IP address determined from those headers is used as the source address.</p> <p>For agents, this value is also potentially controlled by the override options on the Agent settings.</p> <p><b>Example:</b> <code>inIpRange("127.0.0.1/8")</code></p>
<code>inIpRange(java.net.InetAddress ipAddress, int prefixSize)</code>	<p>Validates the source IP address against the <code>ipAddress</code> and the <code>prefixSize</code> parameters specified individually. When Source IP headers defined in the <a href="#">HTTP Requests</a> page are found, the source IP address determined from those headers is used as the source address.</p> <p>For agents, this value is also potentially controlled by the override options on the Agent settings.</p> <p><b>Example:</b></p> <pre>inIpRange(InetAddress.getByName("127.0.0.1"), 8) is equivalent to inIpRange("127.0.0.1/8")</pre>
<code>inIpRange(String cidr, String listValueLocation, boolean fallBackToLastHopIp, String... headerNames)</code>	<p>Validates the source IP address in the first of the specified <code>headerNames</code> using the <code>cidr</code> value. Can be specified as part of a Groovy Script as a means of overriding the configuration stored in PingAccess for a specific Groovy Script rule.</p> <p>Valid values for the <code>listValueLocation</code> parameter are <code>FIRST</code>, <code>LAST</code>, and <code>ANY</code>. This parameter controls where, in a multivalued list of source IP addresses, the last source should be taken from. If <code>ANY</code> is used, if any</p>

Matcher	Description
<pre>inIpRange(java.net.InetAddress address, int prefixSize, String listValueLocation, boolean fallBackToLastHopIp, String... headerName)</pre>	<p>of the source IP addresses in a matching header match the CIDR value, the matcher evaluates to true.</p> <p><b>Example:</b> <code>inIpRange("127.0.0.1/8", "LAST", true, "X-Forwarded-For", "Custom-Source-IP")</code></p> <p>Validates the source IP address in the first of the specified headerNames using the address and prefixSize values. In all other respects, this matcher behaves the same as the version that uses a cidr value for comparison.</p> <p><b>Example:</b>  <code>inIpRange(InetAddress.getByName("127.0.0.1"), 8, "LAST", true, "X-Forwarded-For", "Custom-Source-IP")</code></p>
<pre>requestXPathMatches(String xpathString, String xpathValue)</pre>	<p>Validates that the value returned by the xpathString parameter is equal to the xpathValue parameter.</p> <p><b>Example:</b> <code>requestXPathMatches("//header[@name='Host']/text()", "localhost:3000")</code></p>
<pre>inTimeRange(String startTime, String endTime)</pre>	<p>Validates that the current server time is between the startTime and endTime parameters.</p> <p><b>Example:</b> <code>inTimeRange("9:00 am", "5:00 pm")</code></p>
<pre>inTimeRange24(String startTime, String endTime)</pre>	<p>Validates that the current server time is between the specified 24-hour formatted time range between the startTime and endTime parameters.</p> <p><b>Example:</b> <code>inTimeRange24("09:00", "17:00")</code></p>
<pre>requestHeaderContains(String field, String value)</pre>	<p>Validates that the HTTP header field value is equal to the value parameter.</p> <p><b>Example:</b> <code>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")</code></p>
<pre>requestHeaderContains(Map&lt;String, String&gt; fieldValuesMap, boolean caseSensitive)</pre>	<p>Validates that at all of the HTTP header fields maps to the associated value. The first fieldValuesMap string contains the HTTP header name, and the second string contains the value to compare the incoming request header value with.</p> <p>The caseSensitive parameter determines whether a case-sensitive comparison is performed on the value.</p> <p>The second string in the fieldValuesMap supports Java Regular Expressions.</p>

Matcher	Description
<pre>requestPostFormContains (Map&lt;String, String&gt; fieldValuesMap, boolean caseSensitive)</pre>	<p>If multiple pairs of strings are present in the <code>fieldValuesMap</code> parameter, then all conditions must be met in order for the matcher to pass.</p> <p><b>Example:</b> <code>requestHeaderContains(['User-Agent': 'Mozilla/5.0', 'Cookie': 'JSESSIONID'], false)</code></p> <p>Validates that all of the HTTP form fields maps to the associated value. The first <code>fieldValuesMap</code> string contains the form header name, and the second string contains the value to compare the incoming request header value with.</p> <p>The <code>caseSensitive</code> parameter determines whether a case-sensitive comparison is performed on the value.</p> <p> <b>Note:</b> This matcher determines whether to use fields passed in the URL or forms with a content-type header of <code>application/x-www-form-urlencoded</code>.</p> <p>The second string in the <code>fieldValuesMap</code> supports Java Regular Expressions.</p> <p>If multiple pairs of strings are present in the <code>fieldValuesMap</code> parameter, then all conditions must be met in order for the matcher to pass.</p> <p><b>Example:</b>  <code>requestPostFormContains(['email': '@example.com', 'phonenumber': '720'], false)</code></p>
<pre>requestHeaderDoesntContain (String field, String value)</pre>	<p>Validates that the HTTP header field value is not equal to the value parameter.</p> <p><b>Example:</b>  <code>requestHeaderDoesntContain("User-Agent", "InternetExplorer")</code></p>
<pre>requestBodyContains (String value)</pre>	<p>Validates that the HTTP body contains the value parameter.</p> <p><b>Example:</b>  <code>requestBodyContains("production")</code></p>
<pre>requestBodyDoesntContain (String value)</pre>	<p>Validates that the HTTP body does not contain the value parameter.</p> <p><b>Example:</b>  <code>requestBodyDoesntContain("test")</code></p>
<pre>containsWebSessionAttribute (String attributeName, String attributeValue)</pre>	<p>Validates that the PA Token contains the attribute name and value.</p> <p><b>Example:</b>  <code>containsWebSessionAttribute("sub", "sarah")</code></p>
<pre>containsACRValues (String value)</pre>	<p>Validates that the PA token contains a matching ACR value.</p>

The following table lists the matchers available to only the OAuth Groovy Rule.

Matcher	Description
<code>hasScope(String scope)</code>	Validates that the OAuth access token contains the scope parameter. <b>Example:</b> <code>hasScope("access")</code>
<code>hasScopes(String... scopes)</code>	Validates that the OAuth access token contains the list of scopes. <b>Example:</b> <code>hasScopes("access", "portfolio")</code>
<code>hasAttribute(String attributeName, String attributeValue)</code>	Checks for an attribute value within the current OAuth2 policy context. <b>Example:</b> <code>hasAttribute("account", "joe")</code>


# Addon SDK for Java

---

## 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.

 **Important:** A restart of PingAccess is required after the deployment of any custom plugins written in Java.

### 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
- Identity mappings
- Load balancing strategies
- Locale override service

### 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_HOME/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.

### IdentityMappingPlugin

An interface for developing custom Identity Mappings to provide user identity information to an Application within PingAccess.

### LoadBalancingPlugin

An interface for developing custom Load Balancing strategies that provide the logic for load balancing requests to Target Hosts configured for a Site.

### LocaleOverrideService

An interface for developing custom logic for resolving the locale of a request used for localization.

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](https://ping.force.com/Support) (ping.force.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.



## Get started with the SDK

---

This section describes the directories and build components that comprise the SDK and provides instructions for setting up a development environment.

### SDK directory structure

The PingAccess SDK directory ( `PA_HOME/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.
- `/samples/IdentityMappings` – Contains a maven project with example plug-in implementations for Identity Mappings. You may use these examples for developing your own implementations.
- `/samples/IdentityMappings/README.md` – Contains the details of the IdentityMappings samples.
- `/samples/LoadBalancingStrategies` – Contains a maven project with example plug-in implementations for Load Balancing Strategies. You may use these examples for developing your own implementations.
- `/samples/LoadBalancingStrategies/README.md` – Contains the details of the LoadBalancingStrategies samples.
- `/samples/LocaleOverrideService` – Contains a maven project with example plug-in implementations for the Locale Override Service. You may use these examples for developing your own implementations.
- `/samples/LocaleOverrideService/README.md` – Contains the details of the LocaleOverrideService samples.
- `/apidocs/` – Contains the SDK Javadocs. Open `index.html` to get started.

### SDK 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, 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>4.0.1.3</version>
  <scope>system</scope>
  <systemPath><PA_HOME>/lib/pingaccess-sdk-4.2.0.0.jar</systemPath>
</dependency>
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>1.0.0.GA</version>
  <scope>system</scope>
  <systemPath>PA_HOME/lib/validation-api-1.0.0.GA.jar</systemPath>
</dependency>
<dependency>
```

```

    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.4</version>
    <scope>system</scope>
    <systemPath>PA_HOME/lib/slf4j-api-1.7.4.jar</systemPath>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.4</version>
    <scope>system</scope>
    <systemPath>PA_HOME/lib/slf4j-log4j12-1.7.4.jar</systemPath>
</dependency>

```

Replace *PA\_HOME* with the path to the PingAccess installation.

## How to install the SDK 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\_HOME/sdk/samples/Rules*
  - For the Site Authenticators samples, navigate to *PA\_HOME/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\_HOME/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]
-----

```

## Create your own Plugins

---

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

 **Important:** A restart of PingAccess is required after the deployment of any custom plugins written in Java.

### Create an identity mapping

- For details on how to create an identity mapping, reference the javadoc at: [PA\\_HOME/sdk/apidocs/com/pingidentity/pa/sdk/identitymapping/IdentityMapping.html](#)
- Add a Java class to `/sdk/samples/IdentityMappings/src` that implements `com.pingidentity.pa.sdk.identitymapping.IdentityMappingPlugin` and is annotated by `com.pingidentity.pa.sdk.identitymapping.IdentityMapping`. The base classes `com.pingidentity.pa.sdk.identitymapping.IdentityMappingPluginBase` and

`com.pingidentity.pa.sdk.identitymapping.header.HeaderIdentityMappingPlugin` are available to simplify implementing an `IdentityMapping`.

- Add the class name of the new class to `/sdk/samples/IdentityMappings/src/main/resources/META-INF/services/com.pingidentity.pa.sdk.identitymapping.IdentityMappingPlugin`. Execute `maven install` on the `IdentityMappings` sample pom.

## Create a load balancing strategy

- For details on how to create a load balancing strategy, reference the javadoc at: `PA_HOME/sdk/apidocs/com/pingidentity/pa/sdk/ha/lb/LoadBalancingPlugin.html`
- Add a Java class to `/sdk/samples/LoadBalancingStrategies/src` that implements `com.pingidentity.pa.sdk.ha.lb.LoadBalancingPlugin` and is annotated by `com.pingidentity.pa.sdk.ha.lb.LoadBalancingStrategy`. A base class `com.pingidentity.pa.sdk.ha.lb.LoadBalancingPluginBase` is available to simplify implementing a `LoadBalancingStrategy`.
- Add the class name of the new class to `/sdk/samples/Rules/src/main/resources/META-INF/services/com.pingidentity.pa.sdk.ha.lb.LoadBalancingPlugin`. Execute `maven install` on the `LoadBalancingStrategies` sample pom.

## Create a locale override service

- For details on how to create a locale override service, reference the javadoc at: `PA_HOME/sdk/apidocs/com/pingidentity/pa/sdk/localization/LocaleOverrideService.html`
- Add a Java class to `/sdk/samples/LocaleOverrideService/src` that implements `com.pingidentity.pa.sdk.localization.LocaleOverrideService`.
- Replace the existing content in `/sdk/samples/LocaleOverrideService/src/main/resources/META-INF/services/com.pingidentity.pa.sdk.localization.LocaleOverrideService` with the class name of the new class. Execute `maven install` on the `LocaleOverrideService` sample pom.

## Create a rule

- For details on how to create a Rule, reference the javadoc at: `PA_HOME/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.


## Create a site authenticator

- For details on how to create a Site Authenticator, reference the javadoc at: `PA_HOME/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.SiteAuthenticatorInterceptor` and 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.

 **Important:** A restart of PingAccess is required after the deployment of any custom plugins written in Java.

### Logging

Use the SLF4j API for logging activities in your module. Documentation on using SLF4j is available on the [SLF4j website](#).

### Lifecycle

The plugins and the implementation of a PluginConfiguration can be instantiated for a number of reasons and at many times. For example, with a RuleInterceptor here is what happens before the RuleInterceptor is available to process user requests:

- 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}`.

# Agent SDK for Java

---

## Preface

---

This document provides technical guidance for using the PingAccess Agent SDK for Java. Developers can use this guide along with the Javadocs for the Java Agent API and sample source code to implement the PingAccess Agent Protocol in custom agents.

### Intended audience

This guide is intended for application developers and system administrators responsible for implementing a Java PingAccess Agent. The reader should be familiar with Java software-development principles and practices. It describes the use of the SDK within a sample Java Servlet Filter.

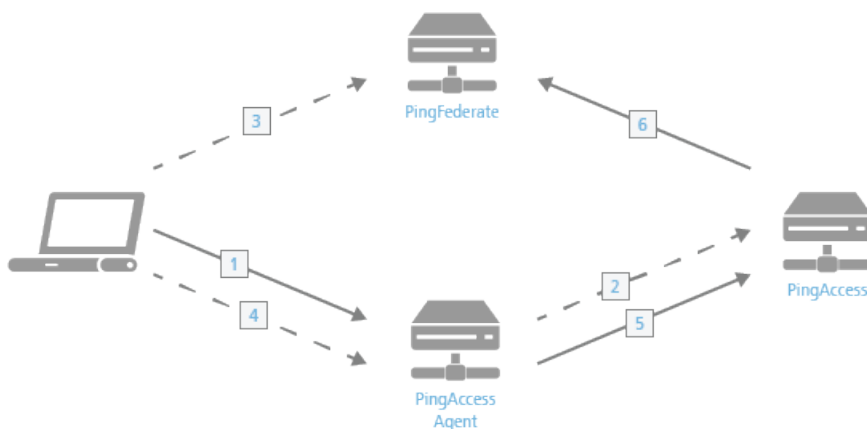
### Additional documentation

The Java Agent API Javadocs provide detailed reference information for developers. After unzipping the `pingaccess-agent-java-sdk-1.1.2.zip` package, the Javadocs can be accessed with a web browser by viewing the file `<AGENT_SDK_JAVA_HOME>/apidocs/index.html`.

## Introduction

---

The PingAccess Agent SDK for Java provides an API and sample code to enable developers to build agents for Java-based application and web servers. Agents provide access management features to their containing server by relying on central PingAccess servers over the PingAccess Agent Protocol. The [PingAccess Agent Protocol Specification](#) is available from the Ping Identity support portal.



The process used when a PingAccess Agent is added to the policy decision process is as follows:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user logs in, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.
6. If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

The PingAccess Agent SDK for Java consists of the following components:

### Java Agent API (Java Agent)

`pingaccess-agent-java-api-1.1.2.0.jar`: The Java Agent API is a set of classes that implement the PingAccess Agent Protocol.

### PingAccess Agent SDK for Java

`pingaccess-agent-java-sdk-1.1.2.zip`: The PingAccess Agent SDK for Java package.

### Servlet Filter Sample

`<AGENT_SDK_FOR_JAVA_HOME>/sample`: The Servlet Filter Sample demonstrates how the Java Agent API integrates into a Java Servlet container. The provided source code, logging configuration and deployment descriptor provide a functional example for how to integrate the Java Agent API into an existing web application. The sample can be modified in place and recompiled using Maven to test customizations to the Servlet Filter Sample code for your environment.



**Note:** This sample code demonstrates how to implement a servlet filter and has been qualified on Apache Tomcat 7. The filter itself is production quality and can be used either as-is or as a starting point for further development. Application configuration within the sample demonstrates how to associate the filter with a servlet (namely, in `web.xml`). Further hardening of this file or the application server configuration may be required.

If you need assistance using the PingAccess Agent SDK for Java, visit the Ping Identity [Support Center](https://ping.force.com/Support) ([ping.force.com/Support](https://ping.force.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.

## Agent SDK directory structure

---

The PingAccess Agent SDK for Java directory (`pingaccess-agent-java-sdk-1.1.2`) contains the following:

### `/apidocs`

The Javadocs for the Java Agent API. Open `index.html` in this directory to access the Javadocs content.

### `/dist`

The directory containing `pingaccess-agent-java-api-1.1.2.0.jar`

### `/sample`

A directory containing `src` and `target` directories for building a Java Servlet Filter. This filter uses the Java Agent API, an `agent.properties` configuration exported from PingAccess, and the `init-params` from the web application `web.xml` file to enforce resource policy decisions configured in PingAccess.

## Agent SDK prerequisites

---

Before you start, ensure you have the Java SDK, [Apache Maven](https://maven.apache.org) ([maven.apache.org](https://maven.apache.org)) and an application server (e.g. Apache Tomcat) installed. The sample uses Apache Maven and assumes that the Java Agent API can be referenced as a dependency. It references Ping Identity's public Maven repository, located at:

```
http://maven.pingidentity.com/release
```

If Internet access is unavailable, there are two other ways to reference the Java Agent API. First, once Apache Maven is installed, install the Java Agent API into your local dependency repository by executing the following command:

```
mvn install:install-file -Dfile=<AGENT_SDK_JAVA_HOME>/dist/pingaccess-agent-java-api-1.1.2.0.jar -DgroupId=com.pingidentity -DartifactId=pingaccess-agent-java-api -Dversion=1.1.2.0 -Dpackaging=jar
```



Alternatively, update the dependency in your `pom.xml` to point to the local installation:

```
<dependency>
  <groupId>com.pingidentity</groupId>
  <artifactId>pingaccess-agent-java-api</artifactId>
  <version>1.1.2.0</version>
  <scope>system</scope>
  <systemPath><AGENT_SDK_JAVA_HOME>/dist/pingaccess-agent-java-
api-1.1.2.0.jar</systemPath>
</dependency>
```

With either of these options, replace `<AGENT_SDK_JAVA_HOME>` with the absolute path to the unzipped `pingaccess-agent-java-sdk-1.1.2.0` directory.

## How to install the servlet filter sample

Ensure you have the PingAccess Agent SDK for Java, Apache Maven, and Apache Tomcat. These instructions assume that you are using Apache Tomcat.

- The servlet filter sample is installed under `<AGENT_SDK_JAVA_HOME>/sample`.
- A deployed version of the servlet filter is under `<AGENT_SDK_JAVA_HOME>/sample/target/agent-sample`.

For the initial setup of the web application, we assume you already have Tomcat or another application server set up on the same machine hosting PingAccess. Out of the box, PingAccess generates self-signed server certificates for listeners servicing runtime ports with the hostname `localhost`. By default, the servlet filter sample configures the Java Agent (Java Agent API) to use "strict" certificate checking for communications with PingAccess. The Java Agent will not be able to communicate with PingAccess over HTTPS if it is not also on `localhost` because of strict hostname checking. If PingAccess already has a server certificate configured with a valid hostname other than `localhost`, then you can deploy the Java Agent into a container on another system.

If you cannot setup the application server on the same system as an existing PingAccess service, and that PingAccess deployment still uses the default `localhost` server certificate for the Agent port, there is another option. You can change the default `strict` certificate checking in `agent-sample/WEB-INF/web.xml` to `test`. Please see the comments in `agent-sample/WEB-INF/web.xml` for more detail.

The `agent-sample` (servlet filter sample) web application is meant to demonstrate the features of the Java Agent within the context of a functional, standalone sample application. The servlet filter sample uses the Java Agent to intercept requests bound for sample servlet and will accept or reject them based on the configured PingAccess policy. The sample servlet only prints out headers, cookies, and other parameters it receives in the request.

1. In the Tomcat `webapps` directory, create a directory called `ROOT`.
2. Copy the `WEB-INF`, `META-INF`, and `assets` contents from `/sample/target/agent-sample/` into `webapps/ROOT`.

This sample servlet filter must run as `/` to properly carry out the OpenID Connect workflow.

3. In the Tomcat `bin` directory, create a script called `setenv.sh` (Linux) or `setenv.bat` (Windows) with the following contents:

- For Linux:

```
export CATALINA_OPTS="-Dlog4j.configurationFile=<PATH_TO_TOMCAT_ROOT>/
webapps/ROOT/WEB-INF/logs/log4j2.xml -
Dserver.log.file=<PATH_TO_TOMCAT_ROOT>/webapps/ROOT/WEB-INF/logs/
server.log"
```

- For Windows:

```
set CATALINA_OPTS=="-Dlog4j.configurationFile=<PATH_TO_TOMCAT_ROOT>/
webapps/ROOT/WEB-INF/logs/log4j2.xml -
```

```
Dserver.log.file=<PATH_TO_TOMCAT_ROOT>/webapps/ROOT/WEB-INF/logs/
server.log"
```

The Agent servlet filter logging is configured in `webapps/ROOT/WEB-INF/logs/log4j2.xml`, and outputs to `webapps/ROOT/WEB-INF/logs/server.log`

4. **Conditional:** If running Tomcat on Linux, execute the command `chmod a+x setenv.sh` to make this script executable.
5. Configure a PingAccess Agent.
6. Configure an Application and associate the new Agent with it.
7. When configuring an Agent through the PingAccess administration console, it automatically exports the agent properties file. Copy the downloaded properties file to `webapps/ROOT/WEB-INF/agent-config/agent.properties`.
8. Start Tomcat
9. Start a browser and navigate to `http://<HOST>:<PORT>/sample`

The values for `<HOST>` and `<PORT>` here need to match the Tomcat configuration in use.



**Note:** If your Tomcat server is not set up to use HTTPS, ensure that any related Web Sessions do not have the **Secure** option enabled.

## PingAccess Agent SDK for Java Release History

---

### Version 1.1.2 - June 2017

#### Resolved issues

- Fixed an issue where the Java agent was handling the PingAccess `set-cookie` header incorrectly.
- Fixed an issue where the Java agent wasn't correctly processing multiple `set-cookie` headers sent by PingAccess.
- Fixed an issue where the SDK sample implementation was not correctly enforcing the PingAccess Agent Protocol directives when the `ClientHttpRequest` `getCookies` method was called. This resulted in a discrepancy between the cookie request headers returned from the `getHeader*` methods and the `getCookies` method.

### Version 1.1.1 - April 2017

#### Resolved issues

- Fixed an issue where unknown attributes should be ignored
- Fixed an issue where percent-encoded sequences in resource paths were being handled incorrectly
- Fixed an issue where an "Index out of bounds" exception occurs if a cookie value is "".

### Version 1.1 - August 2015

#### Resolved issues

- Fixed an issue where OAuth API response headers were getting trimmed
- Fixed an issue where the Java Agent enforced the requirement of a username and shared secret
- Fixed an issue where the Agent was not handling a 477 response correctly

### Version 1.0 - June 2015

- Initial Release

# Agent SDK for C

---

## Preface

---

This documentation provides technical guidance for using the PingAccess Agent SDK for C. Developers can use this guide along with the API documentation for the SDK and sample source code to implement custom agents that use the PingAccess Agent Protocol to integrate with a PingAccess policy server.

### Intended Audience

This guide is intended for application developers and system administrators responsible for implementing a C-based PingAccess agent. The reader should be familiar with C software development principles and practices. It describes the use of the SDK within a sample Agent for Apache.

### Additional documentation

The SDK documentation provides detailed reference information for developers. After unzipping the `pingaccess-agent-c-sdk-version.zip` package, the API documentation can be accessed with a web browser by viewing the file `AGENT_SDK_C_HOME/apidocs/index.html`. The current version of the API documentation may also be found online at <https://developer.pingidentity.com/content/dam/developer/documentation/pingaccess/agent-c-sdk/latest/index.html>

## Introduction

---

The PingAccess Agent SDK for C provides an API and sample code to enable developers to build agents for C or C++-based application and web servers.

### Supported platforms

- RedHat Enterprise Linux Server 6 (32 bit)
- RedHat Enterprise Linux Server 6 (64 bit)
- RedHat Enterprise Linux Server 7 (32 bit)
- RedHat Enterprise Linux Server 7 (64 bit)
- SUSE Linux Enterprise Server 11 SP4 (64 bit)
- SUSE Linux Enterprise Server 12 SP2 (64 bit)

The PingAccess Agent SDK for C provides an API and sample code to enable developers to build agents for C-based application and web servers. Agents provide access management features to their containing server by relying on central PingAccess servers over the PingAccess Agent Protocol. The *PingAccess Agent Protocol Specification* is available from the Ping Identity support portal.

The process used when a PingAccess Agent is added to the policy decision process is as follows:

1. The client accesses a resource. If the user is already authenticated, this process continues with step 5.
2. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then redirects the client to PingFederate to establish a session.
3. The user logs in, and PingFederate creates the session.
4. The client is then redirected back to the resource.
5. The agent asks PingAccess for instructions. PingAccess checks the URL policy and determines that it is a protected resource. PingAccess then checks the session token and determines that it is valid.

- If session revocation is enabled, PingAccess checks and updates the central session revocation list. If the session is valid, the agent is instructed to set identity HTTP headers.

The PingAccess Agent SDK for C consists of the following components:

### SDK (C Agent)

The SDK is a set of C header files that represent the interface to the library that implements the PingAccess Agent Protocol.

### C Agent libraries

The C libraries implement the PingAccess Agent Protocol. There are binaries for Red Hat Enterprise Linux 6/7 as well as for Windows.

### PingAccess Agent SDK for C API documentation

Each of the interfaces defined in the header files is fully documented.

### Apache Agent Sample

`<AGENT_SDK_FOR_C_HOME>/sample` : The Apache Agent Sample demonstrates how the SDK integrates into Apache as an Apache module that is integrated with the Apache request processing workflow. The provided source code and module configuration provide a functional example for how to integrate the SDK into an existing web application. The sample can be modified in-place and recompiled using `make` to test customizations to the Sample code for your environment.



**Note:** This sample code demonstrates how to implement the PingAccess Agent as an Apache module and has been qualified in the following environments:

- Red Hat Enterprise Linux 6 (RHEL6), 64-bit
- Red Hat Enterprise Linux 7 (RHEL7), 64-bit

The Apache Agent itself is production-quality and can be used either as-is or as a starting point for further development. While Ping Identity provides this as a sample, the only versions that are fully supported in production are the precompiled versions available from the Ping Identity download site.

The sample includes instructions for how to configure the sample as a PingAccess Agent to protect websites within its scope. Note that further hardening of the Apache server configuration or of the sample configuration file may be required.

If you need assistance using the PingAccess Agent SDK for C, visit the Ping Identity [Support Center](http://ping.force.com/Support) (ping.force.com/Support) to see how we can help you with your application. You may also engage the Ping Identity Professional Services team for assistance with developing customizations.

## Getting Started with the PingAccess Agent SDK for C

---

### Agent SDK for C directory structure

The PingAccess Agent SDK for C directory contains the following subdirectories:

/

This directory contains the Agent SDK for C `README.md`, which contains information developers will need in order to develop agents using the SDK. It also contains `ReadMeFirst.pdf` and `Legal.pdf`, which contain general information about the kit and third-party licenses used by components of the SDK.

**/apidocs**

API documentation for the SDK. Open `index.html` to access the API documentation content.

**/include**

Agent SDK header files.

**/lib**

32-bit and 64-bit libraries for Red Hat Enterprise Linux 6/7 and Windows, including third-party dependencies required by the SDK.

## /sample

Sample source code for an agent for Apache. This sample agent uses the SDK, and includes a sample configuration file for Apache to use the sample agent to enforce authentication and access control policies.

## Agent SDK for C sample code

The Agent SDK for C sample code is available both in the SDK distribution and on github at <https://github.com/pingidentity/pa-agent-c-sdk-sample-apache>.

Before building the sample code, ensure you have the PingAccess Agent SDK for C archive, the GNU `make` utility and associated compiler utilities installed with your compiler, and Apache and its development libraries.

The sample uses Apache and assumes that the PingAccess Agent SDK for C can be referenced as a dependency. For more details about specific dependencies and requirements, as well as instructions on how to build the sample code, see `AGENT_SDK_C_HOME/sample/readme.md`.

## PingAccess Agent SDK for C release history

---

- **Version 1.1.2 - March 2017**
  - Added support for:
    - SUSE Linux Enterprise Server 11 SP4 (x86\_64)
    - SUSE Linux Enterprise Server 12 SP2 (x86\_64)
- **Version 1.1.1 - January 2017**
  - Established a workaround for a *known issue* in the Network Security Services library that results in a memory leak when the agent closes a HTTPS connection to a PingAccess policy server. For more information, see [this KB article](#).
  - Fixed an issue where duplicate headers were being included in the backend request to the PingAccess Engine causing the request for content to be blocked by the agent.
- **Version 1.1 - November 2016**
  - Added policy server failover support. Policy server failover support is only provided by the SDK when using the libcurl HTTP client.
- **Version 1.0.2 - September 2016**
  - Fixed an issue where agents could not communicate with PingAccess servers using a certificate that is signed by a certificate authority because the CRL Distribution Point extension is missing. This issue is limited to agents on Windows deployments.
  - Addressed a potential security vulnerability. This issue is limited to Windows deployments.
- **Version 1.0.1 - May 2016**
  - Fixed an issue with ZeroMQ policy cache where a terminated process could cause a condition that resulted in unexpected CPU utilization.
- **Version 1.0 - April 2016**
  - Initial Release.

# Addon SDK for Java migration guide

---

## PingAccess Addon SDK for Java Migration Guide

---

Plugins built against the Java Addon SDK for PingAccess 5.0 are now required to be built with JDK 8, as the SDK for PingAccess 5.0 uses Java 8 features. Previously, plugins could be built with either JDK 7 or JDK 8.

The following sections provide a detailed description of the changes, organized by package. Where relevant, code examples show how to port existing code to account for the changes in the SDK APIs.

- [General changes](#) on page 165
- [com.pingidentity.pa.sdk.http](#) on page 168
- [com.pingidentity.pa.sdk.identity](#) on page 185
- [com.pingidentity.pa.sdk.identitymapping.header](#) on page 185
- [com.pingidentity.pa.sdk.policy](#) on page 186
- [com.pingidentity.pa.sdk.services](#) on page 192
- [com.pingidentity.pa.sdk.siteauthenticator](#) on page 192
- [com.pingidentity.pa.sdk.ui](#) on page 193
- [com.pingidentity.pa.sdk.user](#) on page 193
- [com.pingidentity.pa.sdk.util](#) on page 193

### General changes

#### Prevent modification to Request in Response chain

Starting in PingAccess 5.0, any modifications made to a Request or its header fields during response processing will now result in a warning log message and the modification operation being ignored. Previously, PingAccess would log a warning message about the modification but still allow the modification operation to complete.

#### Retrieving Key Pair and Trusted Certificate Group configuration data

In the previous version of the SDK, a SDK plugin accessed the configuration data of a Key Pair or Trusted Certificate Group configured via the Administrative API by annotating a field in the plugin's `PluginConfiguration` class with a `JsonDeserialize` annotation, specifying the appropriate custom deserializer from the SDK. For example:

```
public class Configuration extends SimplePluginConfiguration
{
    @JsonDeserialize(using = PrivateKeyDeserializer.class)
    KeyStore.PrivateKeyEntry keyPair;

    @JsonDeserialize(using = TrustedCertificateGroupDeserializer.class)
    Collection<X509Certificate> certificateGroup;
}
```

In the current version of the SDK, this mechanism has changed to be less error-prone as well as to provide access to more properties of the Key Pairs and Trusted Certificate Groups. The previous `Configuration` class should be ported to the following:

```
public class Configuration extends SimplePluginConfiguration
{
    KeyPairModel keyPair;

    TrustedCertificateGroupModel certificateGroup;
}
```

The `KeyPairModel#getPrivateKeyEntry` method provides access to the `KeyStore.PrivateKeyEntry` object for the corresponding Key Pair in the administrative configuration. The `TrustedCertificateGroupModel#getCertificates` method provides access to the Collection of `X509Certificate` objects in the corresponding Trusted Certificate Group in the administrative configuration. Refer to the JavaDoc for each of these classes for more information.

Related to this change, the provided implementations of `ConfigurationModelAccessor`, `PrivateKeyAccessor` and `TrustedCertificateGroupAccessor`, have been updated to use these new classes. Both classes have also been moved to new packages. `PrivateKeyAccessor` has also been renamed to `KeyPairAccessor`.

#### BEFORE:

```
import com.pingidentity.pa.sdk.accessor.PrivateKeyAccessor;
import com.pingidentity.pa.sdk.accessor.TrustedCertificateGroupAccessor;

// ... class definition omitted ...

private void invokePrivateKeyAccessorGet (
    PrivateKeyAccessor accessor,
    String id)
{
    KeyStore.PrivateKeyEntry keyPair = accessor.get(id);
}
private void invokeTrustedCertificateGroupAccessorGet (
    TrustedCertificateGroupAccessor accessor,
    String id)
{
    Collection<X509Certificate> certificates = accessor.get(id);
}
```

#### AFTER:

```
import
    com.pingidentity.pa.sdk.accessor.certgroup.TrustedCertificateGroupModel;
import com.pingidentity.pa.sdk.accessor.keypair.KeyPairAccessor;

// ... class definition omitted ...

private void invokePrivateKeyAccessorGet (
    KeyPairAccessor accessor,
    String id)
{
    KeyStore.PrivateKeyEntry keyPair = accessor.get(id)

    .map(KeyPairModel::getPrivateKeyEntry)
    .orElse(null);
}

private void invokeTrustedCertificateGroupAccessorGet (
    TrustedCertificateGroupAccessor accessor,
    String id)
{
    Collection<X509Certificate> certificates = accessor.get(id)
    .map(TrustedCertificateGroupModel::getCertificates)
    .orElse(null);
}
```

#### Changes to validation of PluginConfiguration instances

In the previous version of the SDK, the `ConfigurablePlugin#configure` method was invoked and passed a `PluginConfiguration` instance. The `ConfigurablePlugin` was expected to assign the specified `PluginConfiguration` instance to a field annotated with the `javax.validation.Valid` annotation. After the `configure` method returned, `PingAccess` passed the `ConfigurablePlugin` instance to a `javax.validation.Validator` for further validation.

If setup correctly, this logic allows `javax.validation.Constraint` annotations to be used to declare the validation to be applied to fields in a `PluginConfiguration` implementation, ensuring the configuration was valid as well as providing validation error message to `PingAccess` to provide to administrators using the Administrative API or UI.

However, if the `ConfigurablePlugin#configure` method needed to post-process the specified `PluginConfiguration` instance, the method needed to duplicate all the validation declared on the fields of the `PluginConfiguration`.

To remove the need for this duplication of validation logic, `PingAccess` will now validate the `PluginConfiguration` instance with a `javax.validation.Validator` prior to passing the instance to the `ConfigurablePlugin#configure` method.

Further, the `ConfigurablePlugin` no longer needs to annotate the field used to hold the `PluginConfiguration` instance. The field is still necessary to implement the `ConfigurablePlugin#getConfiguration` method.

The following example `ConfigurablePlugin` implementation demonstrates this change:

```
public class ValidationExample
    implements ConfigurablePlugin<ValidationExample.Configuration>
{
    // @Valid annotation no longer required
    private Configuration configuration;

    @Override
    public void configure(Configuration configuration) throws
    ValidationException
    {
        this.configuration = configuration;

        // With the previous version of the SDK, these assertions were not
        // guaranteed to be true, despite the javax.validation.Constraint
        // annotations enforcing these conditions.
        //
        // In the current version of the SDK, these assertions are guaranteed
        // to be true because they are enforced by the
    javax.validation.Constraint
    // annotations on the fields in the PluginConfiguration class, and
    the
        // PluginConfiguration validation is performed before invoking the
        // configure method.
        //
        // The end result is that plugins can remove duplicated validation
        // logic from the configure method if further post-processing of the
        // configuration needs to be performed.
        assert(configuration.getAttributeName() != null);
        assert(configuration.getAttributeName().length() > 0);
        assert(configuration.getAttributeName().length() <= 16);
        assert(configuration.getAttributeValue() != null);

    }

    @Override
    public Configuration getConfiguration()
    {
        return configuration;
    }

    static class Configuration extends SimplePluginConfiguration
    {
        @NotNull
        @Size(min = 1,
            max = 16,
            message = "Attribute name length must be between 1 and 16
characters")
        private String attributeName;
    }
}
```



```

    @NotNull
    private String attributeValue;

    public String getAttributeName()
    {
        return attributeName;
    }

    public void setAttributeName(String attributeName)
    {
        this.attributeName = attributeName;
    }

    public String getAttributeValue()
    {
        return attributeValue;
    }

    public void setAttributeValue(String attributeValue)
    {
        this.attributeValue = attributeValue;
    }
}

```

### **com.pingidentity.pa.sdk.http**

#### **com.pingidentity.pa.sdk.http.Body**

The Body interface has changed to require an explicit read of data before invoking methods to obtain that data. Previously, methods to obtain the data would result in an implicit read of the data. The following code examples illustrate this change in semantics.

As the updated JavaDoc for the Body interface indicates, plugins should avoid interrogating a Body object unless absolutely necessary because reading a Body object's data into memory can impact the scalability of PingAccess. As plugin code is updated, evaluate whether the Body object needs to be used by the plugin.

#### *Using the Body#read method*

##### **BEFORE:**

```

private void invokeRead(Body body) throws IOException
{
    body.read();
}

```

##### **AFTER:**

```

private void invokeRead(Body body) throws AccessException
{
    try
    {
        body.read();
    }
    catch (IOException e)
    {
        throw new AccessException("Failed to read body content",
            HttpStatus.BAD_GATEWAY,
            e);
    }
}

```

#### *Using the Body#getContent method:*

**BEFORE:**

```
private void invokeGetContent(Body body) throws IOException
{
    byte[] content = body.getContent();
}
```

**AFTER:**

```
private void invokeGetContent(Body body) throws AccessException
{
    invokeRead(body); // see the Body#read code example for this method
    byte[] content = body.getContent();
}
```

Using the Body#getBodyAsStream method:**BEFORE:**

```
private void invokeGetBodyAsStream(Body body) throws IOException
{
    InputStream stream = body.getBodyAsStream();
}
```

**AFTER:**

```
private void invokeGetBodyAsStream(Body body) throws AccessException
{
    invokeRead(body); // see the Body#read code example for this method
    InputStream stream = body.newInputStream();
}
```

**Note** the rename of the method from `getBodyAsStream` to `newInputStream`.

Using the Body#write method:**BEFORE:**

```
private void invokeWrite(Body body, BodyTransferrer bodyTransferrer)
    throws IOException
{
    body.write(bodyTransferrer);
}
```

**AFTER:**

This functionality is no longer supported. To obtain the content of the Body, read the content into memory using the `Body#read` method and then invoke `Body#getContent` or `Body#newInputStream`.

Using the Body#getLength method**BEFORE:**

```
private void invokeGetLength(Body body) throws IOException
{
    int length = body.getLength();
}
```

**AFTER:**

```
private void invokeGetLength(Body body) throws AccessException
{
    invokeRead(body); // see the Body#read code example for this method

    int length = body.getLength();
}

```

Using the `Body#getRaw` method:

**BEFORE:**

```
private void invokeGetRaw(Body body) throws IOException
{
    byte[] rawBody = body.getRaw();
}

```

**AFTER:**

This functionality is no longer supported. This method used to provide access to the content as it appeared on the wire, which required complicated handling if the body content used a chunked Transfer-Encoding. Use `Body#getContent` instead.

**com.pingidentity.pa.sdk.http.BodyFactory**

Using the `BodyFactory#continuousBody` method

**BEFORE:**

```
private void invokeContinuousBody(BodyFactory bodyFactory, byte[] content)
{
    Body body = bodyFactory.continuousBody(content);
}

```

**AFTER:**

```
private void invokeContinuousBody(BodyFactory bodyFactory, byte[] content)
{
    Body body = bodyFactory.createInMemoryBody(content);
}

```

**BEFORE:**

```
private void invokeContinuousBody(BodyFactory bodyFactory, InputStream in)
{
    Body body = bodyFactory.continuousBody(in);
}

```

**AFTER:**

A `Body` instance can no longer be created from an `InputStream` using the `BodyFactory` class. Instead, a plugin should read the contents of the `InputStream` into a byte array and provide the byte array to `BodyFactory#createInMemoryBody`.

**com.pingidentity.pa.sdk.http.Constants**

The constants available from this class have been removed from the SDK. Plugins using these constants should maintain their own constants with the needed values.

**com.pingidentity.pa.sdk.http.Exchange**

A handful of methods have been removed from the Exchange.

Further, the mechanism for storing data on the Exchange via properties has been enhanced to make it easier to write type-safe code when working with Exchange properties.

*Using the Exchange#getCreationTime method*

**BEFORE:**

```
Calendar creationTime = exchange.getCreationTime();
```

**AFTER:**

```
Calendar creationTime = Calendar.getInstance();
creationTime.setTime(Date.from(exchange.getCreationTime()));
```

**NOTE:** If a Calendar object is not required, consider using the Instant object returned from the `getCreationTime` method directly instead of converting it into a Calendar object.

*Using the Exchange#getDestinations method*

**BEFORE:**

```
List<String> destinations = exchange.getDestinations();
```

**AFTER:**

This functionality is no longer supported. Consider using the `Exchange#getTargetHosts` method to obtain similar information from the Exchange.

*Using the Exchange#getOriginalHostHeader method*

**BEFORE:**

```
String originalHostHeader = exchange.getOriginalHostHeader();
```

**AFTER:**

This functionality is no longer supported. Consider using the `Exchange#getUserAgentHost` method to obtain similar information from the Exchange. The `getUserAgentHost` method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

*Using the Exchange#getOriginalHostHeaderHost method*

**BEFORE:**

```
String host = exchange.getOriginalHostHeaderHost();
```

**AFTER:**

This functionality is no longer supported. Consider using the `Exchange#getUserAgentHost` method to obtain similar information from the Exchange. The `getUserAgentHost` method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

*Using the Exchange#getOriginalHostHeaderPort method*

**BEFORE:**

```
String port = exchange.getOriginalHostHeaderPort();
```

**AFTER:**

This functionality is no longer supported. Consider using the `Exchange#getUserAgentHost` method to obtain similar information from the Exchange. The `getUserAgentHost` method leverages the PingAccess HTTP requests configuration to determine the Host header value sent by the user agent.

Using the `Exchange#getOriginalRequestBaseUri` method

**BEFORE:**

```
String originalRequestBaseUri = exchange.getOriginalRequestBaseUri();
```

**AFTER:**

This functionality is no longer supported. A possible replacement is as follows:

```
String originalRequestBaseUri = exchange.getUserAgentProtocol() +
    "://" +
    exchange.getUserAgentHost();
```

Using the `Exchange#getProperties` method

**BEFORE:**

```
Map<String, String> properties = exchange.getProperties();
```

**AFTER:**

This functionality is no longer supported. Properties should be obtained individually from the Exchange.

Using the `Exchange#getRequestBaseUri` method

**BEFORE:**

```
String requestBaseUri = exchange.getRequestBaseUri();
```

**AFTER:**

This functionality is no longer supported. A possible replacement is as follows:

```
String requestBaseUri = exchange.getUserAgentProtocol() +
    "://" +
    exchange.getUserAgentHost();
```

Using the `Exchange#getRequestScheme` method

**BEFORE:**

```
String requestScheme = exchange.getRequestScheme();
```

**AFTER:**

This functionality is no longer supported. A possible replacement is as follows:

```
String requestScheme = exchange.getUserAgentProtocol() + "://";
```

Using the `Exchange#getUser` method

**BEFORE:**

```
User user = exchange.getUser();
```

**AFTER:**

The User interface is no longer supported. Use the Identity interface instead. It can be retrieved via the Exchange#getIdentity method.

Using the Exchange#setUser method

**BEFORE:**

```
private void invokeSetUser(Exchange exchange, User user)
{
    exchange.setUser(user);
}
```

**AFTER:**

This functionality is no longer supported. The identity associated with an Exchange cannot be replaced.

Using the Exchange#setSourceIp method

**BEFORE:**

```
private void invokeSetSourceIp(Exchange exchange, String sourceIp)
{
    exchange.setSourceIp(sourceIp);
}
```

**AFTER:**

This functionality is no longer supported. This value cannot be changed.

Using the Exchange#setProperty method

**BEFORE:**

```
private void invokeSetProperty(Exchange exchange, String propertyKey,
    String value)
{
    exchange.setProperty(propertyKey, value);
}
```

**AFTER:**

```
private void invokeSetProperty(Exchange exchange,
    ExchangeProperty<String> propertyKey,
    String value)
{
    exchange.setProperty(propertyKey, value);
}
```

See the JavaDoc for ExchangeProperty for instructions on creating an ExchangeProperty object.

Using the Exchange#getProperty method

**BEFORE:**

```
private void invokeGetProperty(Exchange exchange, String propertyKey)
{
    Object propertyValueObj = exchange.getProperty(propertyKey);
    if (propertyValueObj instanceof String)
    {
        String propertyValue = (String) propertyValueObj;
    }
}
```

**AFTER:**

```
private void invokeGetProperty(Exchange exchange, ExchangeProperty<String>
    propertyKey)
{
    String propertyValue = exchange.getProperty(propertyKey).orElse(null);
}
```

**NOTE:** Exchange#getProperty now returns an Optional object instead of the Object directly.

**com.pingidentity.pa.sdk.http.Header**

This deprecated class has been replaced by the Headers interface. A Headers object can be created via a HeadersFactory obtained from the ServiceFactory#headersFactory method. The majority of methods on Header have counterparts on the Headers interface. See the JavaDoc for the Headers interface for more information.

**com.pingidentity.pa.sdk.http.HeaderField**

This class is now final and cannot be extended.

*Constructing a HeaderField***BEFORE:**

```
private HeaderField createHeaderField(String line)
{
    return new HeaderField(line);
}
```

**AFTER:**

```
private HeaderField createHeaderField(String line)
{
    String name = line.substring(0, line.indexOf(':'));
    String value = (line.substring(line.indexOf(":") + 1)).trim();

    return new HeaderField(name, value);
}
```

**NOTE:** Parsing an HTTP header field line can be error prone, consider if the plugin can be avoid having to parse an HTTP header field line.

*Using the HeaderField#setHeaderName method:***BEFORE:**

```
private void invokeSetHeaderName(HeaderField field)
{
    field.setHeaderName(new HeaderName("X-Custom"));
}
```

**AFTER:**

This functionality is no longer supported. A HeaderField's name is set upon construction and cannot be changed.

*Using the HeaderField#getApproximateSize method:***BEFORE:**

```
int approximateSize = field.getApproximateSize();
```

**AFTER:**

This method has been removed. The value returned by the method can still be computed:

```
int approximateSize = 2 * (4 +
    field.getHeaderName().toString().length() +
    field.getValue().length());
```

### **com.pingidentity.pa.sdk.http.Headers**

A few methods on the Headers interface have been updated to use the Instant class, instead of Date.

#### Using the Headers#getDate method

##### **BEFORE:**

```
Date date = headers.getDate();
```

##### **AFTER:**

```
Date date = Date.from(headers.getDate());
```

#### Using the Headers#setDate method

##### **BEFORE:**

```
private void invokeSetDate(Headers headers, Date date)
{
    headers.setDate(date);
}
```

##### **AFTER:**

```
private void invokeSetDate(Headers headers, Date date)
{
    headers.setDate(date.toInstant());
}
```

#### Using the Headers#getLastModified method

##### **BEFORE:**

```
SimpleDateFormat format = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss
z",
    Locale.ENGLISH);

String lastModified = headers.getLastModified();
if (lastModified != null)
{
    Date lastModifiedDate = format.parse(lastModified);
}
```

##### **AFTER:**

```
Date lastModifiedDate = Date.from(headers.getLastModified());
```

#### Using the Headers#setLastModified method

##### **BEFORE:**

```
private void invokeSetLastModified(Headers headers, Date date)
{
    SimpleDateFormat format = new SimpleDateFormat("E, dd MMM yyyy
HH:mm:ss z",
```



```

Locale.ENGLISH);
headers.setLastModified(format.format(date));
}

```

**AFTER:**

```

private void invokeSetLastModified(Headers headers, Date date)
{
    headers.setLastModified(date.toInstant());
}

```

**com.pingidentity.pa.sdk.http.HeadersFactory**

*Using the HeadersFactory#createFromRawHeaderFields method*

**BEFORE:**

```

private void invokeCreateFromRawHeaderFields(HeadersFactory factory,
                                             List<String> fields) throws
    ParseException
{
    Headers headers = factory.createFromRawHeaderFields(fields);
}

```

**AFTER:**

This functionality is no longer supported. Consider if the plugin can create HeaderFields directly and utilize the HeadersFactory#create method.

**com.pingidentity.pa.sdk.http.HttpStatus**

The HttpStatus enum was converted to a final class. Common HttpStatus instances are defined as constants on HttpStatus.

*Using the HttpStatus#getLocalizationKey method*

**BEFORE:**

```
String localizationKey = status.getLocalizationKey();
```

**AFTER:**

This functionality is no longer supported. Instead, a HttpStatus contains a LocalizedMessage instance that encapsulates the localization of the status message for use in error templates.

**com.pingidentity.pa.sdk.http.MimeType**

The constants available in this class are now available as constant MediaType instances in the class com.pingidentity.pa.sdk.http.CommonMediaTypes.

**com.pingidentity.pa.sdk.http.MediaType**

This class is now final and cannot be extended.

*Constructing a MediaType*

**BEFORE:**

```

private void createMediaType(String mediaTypeString)
{
    MediaType mediaType = new MediaType(mediaTypeString);
}

```

**AFTER:**

```
private void createMediaType(String mediaTypeString)
{
    MediaType mediaType = MediaType.parse(mediaTypeString);
}
```

**com.pingidentity.pa.sdk.http.Message**

A number of methods have been removed from the Message interface.

*Using the Message#getBodyAsStream method*

**BEFORE:**

```
InputStream bodyStream = message.getBodyAsStream();
```

**AFTER:**

This functionality is no longer supported. However, the following code snippet could be used to maintain semantics of the old method.

```
Body body = message.getBody();
try
{
    body.read();
}
catch (IOException | AccessException e)
{
    throw new RuntimeException("Could not get body as stream", e);
}

InputStream bodyStream = body.newInputStream();
```

While this snippet maintains semantics, it is recommended that a plugin propagate errors as an AccessException instead of a RuntimeException.

*Using the Message#getCharset method*

**BEFORE:**

```
Charset charset = message.getCharset();
```

**AFTER:**

This functionality is no longer supported. However, the following code snippet could be used to maintain semantics of the old method.

```
Charset charset = message.getHeaders().getCharset();
if (charset == null)
{
    charset = StandardCharsets.UTF_8;
}
```

While this snippet maintains semantics, a plugin should consider how to handle the case where a Charset is not specified by a Message's header fields. Assuming a Charset of UTF-8 could lead to issues in some cases.

*Using the Message#getHeader method*

**BEFORE:**

```
Header header = message.getHeader();
```

**AFTER:**

This functionality is no longer supported. Instead, use `Message#getHeaders` and the `Headers` interface instead of `Header`.

*Using the `Message#setHeader` method*

**BEFORE:**

```
private void invokeSetHeader(Message message, Header header)
{
    message.setHeader(header);
}
```

**AFTER:**

This functionality is no longer supported. Instead, use `Message#setHeaders` and the `Headers` interface instead of `Header`.

*Using the `Message#isDeflate` method*

**BEFORE:**

```
boolean deflate = message.isDeflate();
```

**AFTER:**

This method has been removed. However, the value can still be computed with the following code snippet:

```
List<String> contentEncodingValues =
    message.getHeaders().getContentEncoding();
boolean deflate = contentEncodingValues.stream().anyMatch(v ->
    v.equalsIgnoreCase("deflate"))
    && contentEncodingValues.size() == 1;
```

*Using the `Message#isGzip` method*

**BEFORE:**

```
boolean gzip = message.isGzip();
```

**AFTER:**

This method has been removed. However, the value can still be computed with the following code snippet:

```
List<String> contentEncodingValues =
    message.getHeaders().getContentEncoding();
boolean gzip = contentEncodingValues.stream().anyMatch(v ->
    v.equalsIgnoreCase("gzip"))
    && contentEncodingValues.size() == 1;
```

*Using the `Message#isHTTP10` method*

**BEFORE:**

```
boolean http10 = message.isHTTP10();
```

**AFTER:**

This method has been removed. However, the value can still be computed with the following code snippet:

```
boolean http10 = message.getVersion().equals("1.0");
```

*Using the `Message#isHTTP11` method*

**BEFORE:**

```
boolean http11 = message.isHTTP11();
```

**AFTER:**

The method has been removed. However, the value can still be computed with the following code snippet:

```
boolean http11 = message.getVersion().equals("1.1");
```

Using the Message#read method**BEFORE:**

```
private void invokeRead(Message message,
                        InputStream inputStream,
                        boolean createBody) throws IOException
{
    message.read(inputStream, createBody);
}
```

**AFTER:**

This functionality is no longer supported. A Request attached to an Exchange can no longer be completely replaced, but individual components can be replaced, such as the method, uri, headers and body. A Response attached to an Exchange can be replaced by using Exchange#setResponse.

Using the Message#setVersion method**BEFORE:**

```
private void invokeSetVersion(Message message, String version)
{
    message.setVersion(version);
}
```

**AFTER:**

This functionality is no longer supported. The version of a Message cannot be changed.

Using the Message#write method**BEFORE:**

```
private void invokeWrite(Message message,
                        OutputStream output) throws IOException
{
    message.write(output);
}
```

**AFTER:**

This functionality is no longer supported. However, the following code snippet can be used to perform the equivalent operation:

```
private void invokeWrite(Message message,
                        OutputStream output) throws IOException,
                        AccessException
{
    Body body = message.getBody();
    body.read();

    output.write(message.getStartLine().getBytes(StandardCharsets.ISO_8859_1));
}
```

```

output.write(message.getHeaders().toString().getBytes(StandardCharsets.ISO_8859_1));
output.write("\r\n".getBytes(StandardCharsets.ISO_8859_1));
output.write(body.getContent());
output.flush();
}

```

### **com.pingidentity.pa.sdk.http.Method**

The Method interface has been converted to a final class. Additionally, the related Methods enum has been merged into the Method class. The Method class provides common Method instances as class-level constants.

#### Obtaining a common Method instance

##### **BEFORE:**

```
Method get = Methods.GET
```

##### **AFTER:**

```
Method get = Method.GET;
```

#### Using the Method#getMethodName method

##### **BEFORE:**

```
String methodName = method.getMethodName();
```

##### **AFTER:**

```
String methodName = method.getName();
```

### **com.pingidentity.pa.sdk.http.Request**

A few methods have been removed from the Request interface.

#### Using the Request#getPostParams method

##### **BEFORE:**

```
private void invokeGetPostParams(Request request) throws IOException
{
    Map<String, String[]> postParams = request.getPostParams();
}

```

##### **AFTER:**

```
private void invokeGetPostParams(Request request) throws AccessException
{
    Body body = request.getBody();
    try
    {
        body.read();
    }
    catch (IOException e)
    {
        throw new AccessException("Failed to read body content",
            HttpStatus.BAD_GATEWAY,
            e);
    }

    Map<String, String[]> postParams = body.parseFormParams();
}

```

Using the Request#isMultipartFormPost method**BEFORE:**

```
boolean multipartFormPost = request.isMultipartFormPost();
```

**AFTER:**

This method has been removed from the Request interface. However, the value can still be calculated using the following code snippet:

```
Headers headers = request.getHeaders();

boolean multipartFormPost =
    request.getMethod() == Method.POST
    && headers.getContentType() != null
    && headers.getContentType().getBaseType().equals("multipart/form-
data")
    && headers.getContentType().getParameter("boundary") != null;
```

**com.pingidentity.pa.sdk.http.ResponseBuilder**

A handful of methods were removed from ResponseBuilder. Additionally, a handful of methods have changed their semantics, particularly those that included an HTML message payload. See the updated JavaDoc for ResponseBuilder for more info.

Using the ResponseBuilder#badRequestText method**BEFORE:**

```
Response response = ResponseBuilder.badRequestText(message).build();
```

**AFTER:**

```
Response response = ResponseBuilder.newInstance(HttpStatus.BAD_REQUEST)
    .contentType(CommonMediaTypes.TEXT_PLAIN)
    .body(message)
    .build();
```

**NOTE:** This approach does not localize the response body. Using a TemplateRenderer is recommended instead.

Using the ResponseBuilder#contentLength method**BEFORE:**

```
Response response =
    ResponseBuilder.newInstance().contentLength(length).build();
```

**AFTER:**

This functionality is no longer supported. Consider using one of the ResponseBuilder#body methods instead of explicitly setting the content length. This ensures that the body content of the Response aligns with the Content-Length header field.

Using the ResponseBuilder#continue100 method**BEFORE:**

```
Response response = ResponseBuilder.continue100().build();
```

**AFTER:**

```
Response response =
    ResponseBuilder.newInstance(HttpStatus.CONTINUE).build();
```

Using the ResponseBuilder#forbiddenText method**BEFORE:**

```
Response response = ResponseBuilder.forbiddenText().build();
```

**AFTER:**

```
Response response = ResponseBuilder.newInstance(HttpStatus.FORBIDDEN)
    .contentType(CommonMediaTypes.TEXT_PLAIN)
    .body(HttpStatus.FORBIDDEN.getMessage())
    .build();
```

**NOTE:** This approach does not localize the response body. Using a `TemplateRenderer` is recommended instead.

Using the ResponseBuilder#forbiddenWithoutBody method**BEFORE:**

```
Response response = ResponseBuilder.forbiddenWithoutBody().build();
```

**AFTER:**

```
Response response =
    ResponseBuilder.newInstance(HttpStatus.FORBIDDEN).build();
```

**BEFORE:**

```
Response response = ResponseBuilder.forbiddenWithoutBody(message).build();
```

**AFTER:**

```
Response response =
    ResponseBuilder.newInstance(HttpStatus.FORBIDDEN).build();
```

**NOTE:** In the original method, the `String` message parameter was not used.

Using the ResponseBuilder#htmlMessage method**BEFORE:**

```
String message = ResponseBuilder.htmlMessage(caption, text);
```

**AFTER:**

This functionality is no longer supported. Plugins that used this method will need to construct the HTML message without this method. Consider using the `TemplateRenderer` utility class in place of this method.

Using the ResponseBuilder#internalServerError method**BEFORE:**

```
Response response = ResponseBuilder.internalServerError(message).build();
```

**AFTER:**

```
Response response =
    ResponseBuilder.internalServerError().body(message).build();
```

**NOTE:** This approach does not localize the response body. Using a `TemplateRenderer` is recommended instead.

*Using the `ResponseBuilder#internalServerErrorWithoutBody` method*

**BEFORE:**

```
Response response =
    ResponseBuilder.internalServerErrorWithoutBody().build();
```

**AFTER:**

```
Response response = ResponseBuilder.internalServerError().build();
```

*Using the `ResponseBuilder#newInstance` method*

The no-arg `newInstance` method has been removed. A `HttpStatus` is required to create an instance of `ResponseBuilder` and the required `HttpStatus` object should be passed to the `newInstance` method that accepts a `HttpStatus`.

**BEFORE:**

```
Response response = ResponseBuilder.newInstance().build()
```

**AFTER:**

```
Response response =
    ResponseBuilder.newInstance(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

*Using the `ResponseBuilder#noContent` method*

**BEFORE:**

```
Response response = ResponseBuilder.noContent().build();
```

**AFTER:**

```
Response response =
    ResponseBuilder.newInstance(HttpStatus.NO_CONTENT).build();
```

*Using the `ResponseBuilder#notFoundWithoutBody` method*

**BEFORE:**

```
Response response = ResponseBuilder.notFoundWithoutBody().build();
```

**AFTER:**

```
Response response = ResponseBuilder.notFound().build();
```

*Using the `ResponseBuilder#serverUnavailable` method*

**BEFORE:**

```
Response response = ResponseBuilder.serverUnavailable(message).build();
```



**AFTER:**

```
Response response =
    ResponseBuilder.serviceUnavailable().body(message).build();
```

**NOTE:** This approach does not localize the response body. Using a `TemplateRenderer` is recommended instead.

*Using the `ResponseBuilder#serviceUnavailableWithoutBody` method*

**BEFORE:**

```
Response response =
    ResponseBuilder.serverUnavailableWithoutBody().build();
```

**AFTER:**

```
Response response = ResponseBuilder.serviceUnavailable().build();
```

*Using the `ResponseBuilder#status` method*

The status methods have been removed. Instead the status should be specified to the `newInstance` method as it is now required.

**BEFORE:**

```
Response response =
    ResponseBuilder.newInstance().status(HttpStatus.OK).build();
```

**AFTER:**

```
Response response = ResponseBuilder.newInstance(HttpStatus.OK).build();
```

*Using the `ResponseBuilder#unauthorizedWithoutBody` method*

**BEFORE:**

```
Response response = ResponseBuilder.unauthorizedWithoutBody().build();
```

**AFTER:**

```
Response response = ResponseBuilder.unauthorized().build();
```

**com.pingidentity.pa.sdk.http.Response**

A few methods were removed from the `Response` interface.

*Using the `Response#isRedirect` method*

**BEFORE:**

```
boolean redirect = response.isRedirect();
```

**AFTER:**

```
boolean redirect = response.getStatusCode() >= 300
    && response.getStatusCode() < 400;
```

*Using the `Response#setStatusCode` method*

**BEFORE:**

```
response.setStatusCode(HttpStatus.OK.getCode());
```

**AFTER:**

```
response.setStatus (HttpStatus.OK) ;
```

*Using the Response#setStatusMessage method*

**BEFORE:**

```
response.setStatusMessage (HttpStatus.OK.getMessage () ) ;
```

**AFTER:**

```
response.setStatus (HttpStatus.OK) ;
```

**com.pingidentity.pa.sdk.identity****com.pingidentity.pa.sdk.identity.Identity**

The getTokenExpiration method was updated to use an Instant instead of Date.

*Using the Identity#getTokenExpiration method*

**BEFORE:**

```
Date expiration = identity.getTokenExpiration () ;
```

**AFTER:**

```
Date expiration = Date.from (identity.getTokenExpiration ()) ;
```

**com.pingidentity.pa.sdk.identity.OAuthTokenMetadata**

The OAuthTokenMetadata methods now use an Instant instead of a Date.

*Using the OAuthTokenMetadata#getExpiresAt method:*

**BEFORE:**

```
Date expiresAt = metadata.getExpiresAt () ;
```

**AFTER:**

```
Date expiresAt = Date.from (metadata.getExpiresAt ()) ;
```

*Using the OAuthTokenMetadata#getRetrievedAt method:*

**BEFORE:**

```
Date retrievedAt = metadata.getRetrievedAt () ;
```

**AFTER:**

```
Date retrievedAt = Date.from (metadata.getRetrievedAt ()) ;
```

**com.pingidentity.pa.sdk.identitymapping.header**

ClientCertificateMapping has been removed from the SDK, as it was not required to create an IdentityMappingPlugin implementation.

Plugins utilizing this class should create their own version of this class.

**com.pingidentity.pa.sdk.policy****com.pingidentity.pa.sdk.policy.AccessExceptionContext**

The nested Builder class has been removed from `AccessExceptionContext` and instead `AccessExceptionContext` is a builder, that can be initially created with the new `AccessExceptionContext#create` method.

The `LocalizedMessage` interface has been introduced to simplify the configuration of a localized message for use in an error template. A `LocalizedMessage` has three implementations provided in the SDK: `FixedMessage`, `BasicLocalizedMessage` and `ParameterizedLocalizedMessage`. See the following code examples for more information on using these new classes.

*Constructing an AccessExceptionContext:***BEFORE:**

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                    Throwable
    cause)
{
    return AccessExceptionContext.builder()
        .cause(cause)
        .httpStatus(httpStatus)

        .exceptionMessage(httpStatus.getMessage())

        .errorDescription(httpStatus.getLocalizationKey())
            .errorDescriptionIsKey(true)
            .errorDescriptionSubstitutions(new
String[0])
        .build();
}
```

**AFTER:**

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                    Throwable
    cause)
{
    return AccessExceptionContext.create(httpStatus)
        .cause(cause)

        .exceptionMessage(httpStatus.getMessage())

        .errorDescription(httpStatus.getLocalizedMessage());
}
```

**BEFORE:**

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                    String
    localizationKey,
                                                    String[]
    substitutions)
{
    return AccessExceptionContext.builder()
        .httpStatus(httpStatus)
        .errorDescription(localizationKey)
        .errorDescriptionIsKey(true)
```

```

        .errorDescriptionSubstitutions(substitutions)
            .build();
    }

```

**AFTER:**

```

private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                                String
    localizationKey,
                                                                String[]
    substitutions)
{
    LocalizedMessage localizedMessage =
        new ParameterizedLocalizedMessage(localizationKey,
    substitutions);

    return AccessExceptionContext.create(httpStatus)
        .errorDescription(localizedMessage);
}

```

**BEFORE:**

```

private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                                String
    localizationKey)
{
    return AccessExceptionContext.builder()
        .httpStatus(httpStatus)
        .errorDescription(localizationKey)
        .errorDescriptionIsKey(true)
        .build();
}

```

**AFTER:**

```

private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus,
                                                                String
    localizationKey)
{
    LocalizedMessage localizedMessage = new
    BasicLocalizedMessage(localizationKey);
    return AccessExceptionContext.create(httpStatus)
        .errorDescription(localizedMessage);
}

```

**BEFORE:**

```

private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus)
{
    return AccessExceptionContext.builder()
        .from(httpStatus)

        .httpStatusDescription(httpStatus.getLocalizationKey())
        .httpStatusDescriptionIsKey(true)
        .templateFile("template.html")
        .contentType("text/html");
}

```

```
}

```

**AFTER:**

```
private AccessExceptionContext createAccessExceptionContext(HttpStatus
    httpStatus)
{
    return AccessExceptionContext.create(httpStatus)

    .errorDescription(httpStatus.getLocalizedMessage());
}

```

**NOTE:** this example demonstrates that it is no longer possible to set a template file and its associated content type on an `AccessExceptionContext`. To generate an error response from a template file, use the `TemplateRenderer` class. See the JavaDoc for the `TemplateRenderer` class for more information.

**com.pingidentity.pa.sdk.policy.AccessException**

The changes to `AccessExceptionContext` apply to the creation of `AccessException` because the creation of an `AccessException` requires an `AccessExceptionContext`.

In addition to these changes, obtaining information from `AccessException` has also changed. See the code examples below for more information.

Finally, `AccessException` no longer derives from `IOException` and derives directly from `Exception` instead.

*Constructing an AccessException:***BEFORE:**

```
private void throwAccessException(String errorDescription,
    Throwable throwable) throws
    AccessException
{
    throw new AccessException(errorDescription, throwable);
}

```

**AFTER:**

```
private void throwAccessException(String errorDescription,
    Throwable throwable) throws
    AccessException
{
    LocalizedMessage templateMessage = new
    FixedMessage(errorDescription);
    throw new
    AccessException(AccessExceptionContext.create(HttpStatus.INTERNAL_SERVER_ERROR)

    .exceptionMessage(errorDescription)

    .cause(throwable)

    .errorDescription(templateMessage));
}

```

**BEFORE:**

```
private void throwAccessException(String errorDescription) throws
    AccessException
{
    throw new AccessException(errorDescription);
}

```

**AFTER:**

```
private void throwAccessException(String errorDescription) throws
    AccessException
{
    LocalizedMessage templateMessage = new FixedMessage(errorDescription);
    throw new
    AccessException(AccessExceptionContext.create(HttpStatus.INTERNAL_SERVER_ERROR)
        .exceptionMessage(errorDescription)
        .errorDescription(templateMessage));
}
```

**BEFORE:**

```
private void createAccessException(int statusCode,
    String statusMessage,
    String errorDescription) throws
    AccessException
{
    throw new AccessException(statusCode, statusMessage,
        errorDescription);
}
```

**AFTER:**

```
private void createAccessException(int statusCode,
    String statusMessage,
    String errorDescription) throws
    AccessException
{
    HttpStatus httpStatus = new HttpStatus(statusCode, statusMessage);
    LocalizedMessage templateMessage = new FixedMessage(errorDescription);
    throw new AccessException(AccessExceptionContext.create(httpStatus)
        .exceptionMessage(errorDescription)
        .errorDescription(templateMessage));
}
```

**BEFORE:**

```
private void throwAccessException(int statusCode,
    String statusMessage,
    String errorDescription,
    Throwable throwable) throws
    AccessException
{
    throw new AccessException(statusCode, statusMessage, errorDescription,
        throwable);
}
```

**AFTER:**

```
private void throwAccessException(int statusCode,
    String statusMessage,
    String errorDescription,
    Throwable throwable) throws
    AccessException
{
    HttpStatus httpStatus = new HttpStatus(statusCode, statusMessage);
```

```

    LocalizedMessage templateMessage = new FixedMessage(errorDescription);
    throw new AccessException(AccessExceptionContext.create(httpStatus)

.exceptionMessage(errorDescription)

.errorDescription(templateMessage)

.cause(throwable));
}

```

**BEFORE:**

```

private void throwAccessException() throws AccessException
{
    throw new AccessException(AccessExceptionContext.builder()
        .statusCode(403)

.httpStatusMessage("Forbidden")

.build());
}

```

**AFTER:**

```

private void throwAccessException() throws AccessException
{
    throw new
    AccessException(AccessExceptionContext.create(HttpStatus.FORBIDDEN));
}

```

Using the `AccessException#getExceptionContext` method**BEFORE:**

```

AccessExceptionContext context = accessException.getExceptionContext();

```

**AFTER:**

This functionality is no longer supported. The information that used to be provided by the `AccessExceptionContext` is now provided directly by an `AccessException`.

Using the `AccessException#getHttpStatusCode` method**BEFORE:**

```

int statusCode = accessException.getHttpStatusCode();

```

**AFTER:**

```

int statusCode = accessException.getErrorStatus().getCode();

```

Using the `AccessException#getHttpStatusMessage` method**BEFORE:**

```

String statusMessage = accessException.getHttpStatusMessage();

```

**AFTER:**

```

String statusMessage = accessException.getErrorStatus().getMessage();

```

Using the `AccessException#setHttpStatusCode` method

**BEFORE:**

```
accessException.setHttpStatusCode(statusCode);
```

**AFTER:**

This functionality is no longer supported. The status code associated with an `AccessException` is fixed once it is constructed.

*Using the `AccessException#setHttpStatusMessage` method*

**BEFORE:**

```
accessException.setHttpStatusMessage(statusMessage);
```

**AFTER:**

This functionality is no longer supported. The status message associated with an `AccessException` is fixed once it is constructed.

**com.pingidentity.pa.sdk.policy.RuleInterceptor**

The `handleRequest` and `handleResponse` methods on a `RuleInterceptor` no longer throw an `IOException`. Instead, they throw an `AccessException`, which no longer derives from `IOException`.

*Accounting for the `RuleInterceptor#handleRequest` method signature change*

**BEFORE:**

```
@Override
public Outcome handleRequest(Exchange exchange) throws IOException
{
    Outcome outcome = applyPolicy(exchange);

    return outcome;
}
```

**AFTER:**

```
@Override
public Outcome handleRequest(Exchange exchange) throws AccessException
{
    Outcome outcome = applyPolicy(exchange);

    return outcome;
}
```

*Account for the `RuleInterceptor#handleResponse` method signature change*

**BEFORE:**

```
@Override
public void handleResponse(Exchange exchange) throws IOException
{
    applyPolicyToResponse(exchange.getResponse());
}
```

**AFTER:**

```
@Override
```



```
public void handleResponse(Exchange exchange) throws AccessException
{
    applyPolicyToResponse(exchange.getResponse());
}
```

### **com.pingidentity.pa.sdk.policy.error.InternalServerErrorCallback**

This class has been removed. Use `LocalizedInternalServerErrorCallback` instead.

### **com.pingidentity.pa.sdk.services**

#### **com.pingidentity.pa.sdk.services.ServiceFactory**

This class is now final and cannot be extended.

### **com.pingidentity.pa.sdk.siteauthenticator**

#### **com.pingidentity.pa.sdk.siteauthenticator.SiteAuthenticatorInterceptor**

This interface is no longer a `RequestInterceptor` or `ResponseInterceptor`, but it still defines the `handleRequest` and `handleResponse` methods:

```
public interface SiteAuthenticatorInterceptor<T extends
    PluginConfiguration>
    extends DescribesUIConfigurable, ConfigurablePlugin<T>
{
    void handleRequest(Exchange exchange) throws AccessException;
    void handleResponse(Exchange exchange) throws AccessException;
}
```

Additionally, these methods now only throw an `AccessException` instead of an `IOException` or `InterruptedException`.

#### *Accounting for the SiteAuthenticatorInterceptor#handleRequest method signature change*

##### **BEFORE:**

```
@Override
public Outcome handleRequest(Exchange exc)
    throws RuntimeException, IOException, InterruptedException
{
    // Site authenticator implementation //

    return Outcome.CONTINUE;
}
```

##### **AFTER:**

```
@Override
public void handleRequest(Exchange exc) throws AccessException
{
    // Site authenticator implementation //
}
```

#### *Accounting for the SiteAuthenticatorInterceptor#handleResponse method signature change*

##### **BEFORE:**

```
@Override
public void handleResponse(Exchange exc) throws IOException
{
    // Site authenticator response implementation //
}
```

**AFTER:**

```
@Override
public void handleResponse(Exchange exc) throws AccessException
{
    // Site authenticator response implementation //
}
```

**com.pingidentity.pa.sdk.ui****com.pingidentity.pa.sdk.ui.ConfigurationType**

The deprecated PRIVATEKEY enum value has been removed. Instead use a ConfigurationType of ConfigurationType#SELECT and specify the PrivateKeyAccessor.class instance to ConfigurationBuilder#dynamicOptions or UIElement#modelAccessor.

**com.pingidentity.pa.sdk.user****com.pingidentity.pa.sdk.user.User**

This class has been removed from the SDK. Use the Identity interface instead. An instance of Identity can be retrieved from the Exchange, similar to the User interface.

**com.pingidentity.pa.sdk.util****com.pingidentity.pa.sdk.util.TemplateRenderer**

The semantics of the renderResponse method have changed so it produces a Response and does not have any side-effects on the specified parameters.

*Using the TemplateRenderer#renderResponse method:*

**BEFORE:**

```
private void invokeRenderResponse(TemplateRenderer templateRenderer,
                                  Map<String, String> context,
                                  String templateName,
                                  Exchange exchange,
                                  ResponseBuilder builder)
{
    templateRenderer.renderResponse(context, templateName, exchange,
    builder);
}
```

**AFTER:**

```
private void invokeRenderResponse(TemplateRenderer templateRenderer,
                                  Map<String, String> context,
                                  String templateName,
                                  Exchange exchange,
                                  ResponseBuilder builder)
{
    Response response = templateRenderer.renderResponse(exchange,
    context,
    templateName,
    builder);
    exchange.setResponse(response);
}
```

# Performance tuning guide

---

## 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.

- ➔ **Info:** An additional document related to performance, the PingAccess Capacity Planning Guide, is also available to customers as a performance data reference. This document is available from the [Customer Portal](https://ping.force.com/Support) (ping.force.com/Support).

## Java tuning

---

### Tuning the Java heap

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.

### Configure JVM crash log in Java startup

This section describes how to modify JVM crash log settings. The JVM crash log location is specified in *run.bat* (Windows) or *run.sh* (Linux) and is enabled by default.

Use these steps to disable the JVM crash log or to change the location in which the crash log is stored.

1. Open `<PA_HOME>/bin/run.bat` (Windows) or `<PA_HOME>/bin/run.sh` (Linux) for editing.
2. To disable JVM crash log reporting, comment out the line that specifies the JVM crash log location. For example, `#ERROR_FILE=-XX:ErrorFile=$PA_HOME/log/java_error%p.log`.
3. To enable JVM crash log reporting, remove the comment tag and make the line active. For example, `ERROR_FILE=-XX:ErrorFile=$PA_HOME/log/java_error%p.log`.

### Configure memory dumps in Java startup

This section describes how to modify JVM memory dump settings. The JVM memory dump location is specified in *run.bat* (Windows) or *run.sh* (Linux) and is disabled by default.

Use these steps to enable JVM memory dump or to change the location in which the dump is stored.

1. Open `<PA_HOME>/bin/run.bat` (Windows) or `<PA_HOME>/bin/run.sh` (Linux) for editing.
2. To enable JVM memory dump, remove the comment tag on the line that specifies the JVM memory dump location. For example, `HEAP_DUMP=-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$PA_HOME/lo`.
3. To disable JVM memory dump, comment out the line. For example, `#HEAP_DUMP=-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=$PA_HOME/lo`.

## Modify the Java heap size

To modify the Java heap size for both Windows and Linux installations, including their services, do the following:

1. Open the `jvm-memory.options` file located in `PA_HOME/conf`.
2. Specify overall heap size by modifying the `#Minimum heap size` and `#Maximum heap size` parameters.
  - Modify `-Xms512m` to change the `#Minimum heap size value`
  - Modify `-Xmx512m` to change the `#Maximum heap size value`

Specify units as `m` (megabytes) or `g` (gigabytes).

3. Specify young generation size by modifying the `#Minimum size for the Young Gen space` and `#Maximum size for the Young Gen space variables`.
  - Modify `-XX:NewSize=256m` to change the `#Minimum size for the Young Gen space value`
  - Modify `-XX:MaxNewSize=256m` to change the `#Maximum size for the Young Gen space value`

Set values to 50% of `#Minimum heap size` and `#Maximum heap size`.

- ➔ **Info:** Not advisable if selecting the G1 collector (see [Garbage Collector Configuration](#) for more information).

## Operating system tuning

---

This section contains tuning recommendations for your operating system. The tuning recommendations provided here are particularly useful in preventing deployment issues in high capacity environments. See the included topics for guidance specific to your operating system.

### Linux tuning

This section describes tuning recommendations for the Linux operating system environment. Implement these recommendations to prevent deployment issues, particularly in high capacity environments. The following settings will increase the performance and capacity of the networking (particularly TCP) stack and file descriptor usage, respectively, enabling PingAccess to handle a high volume of concurrent requests.

#### Network/TCP tuning:

Add/Modify the following entries in `/etc/sysctl.conf`

```
##TCP Tuning##
# Controls the use of TCP syncookies (default is 1)
# and increase the number of outstanding syn requests allowed.
net.ipv4.tcp_syncookies=1
net.ipv4.tcp_max_syn_backlog=8192

# Increase number of incoming connections.
# somaxconn defines the number of request_sock structures allocated
# per each listen call.
# The queue is persistent through the life of the listen socket.
net.core.somaxconn=4096

# Increase number of incoming connections backlog queue.
# Sets the maximum number of packets, queued on the INPUT side,
# when the interface receives packets faster
# than kernel can process them.
net.core.netdev_max_backlog=65536

# increase system IP port limits
net.ipv4.ip_local_port_range=2048 65535
```

```
# Turn on window scaling which can enlarge the transfer window:
net.ipv4.tcp_window_scaling=1

# decrease TCP timeout
net.ipv4.tcp_fin_timeout=10

# Enables fast recycling of TIME_WAIT sockets.
# (Use with caution according to the kernel documentation!)
net.ipv4.tcp_tw_recycle=1

# Allow reuse of sockets in TIME_WAIT state for new connections
# only when it is safe from the network stack's perspective.
net.ipv4.tcp_tw_reuse=1

# Increase the read and write buffer space allocatable
# (minimum size, initial size, and maximum size in bytes)
net.ipv4.tcp_rmem = 4096 65536 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

# The maximum number of packets which may be queued
# for each unresolved address by other network layers
net.ipv4.neigh.default.unres_qlen=100
net.ipv4.neigh.eth0.unres_qlen=100
net.ipv4.neigh.em1.unres_qlen=100

# Default Socket Receive and Write Buffer
net.core.rmem_default=8388608
net.core.wmem_default=8388608
#####
```

### Increase File Descriptor limits:

Add/Modify the following lines in `/etc/security/limits.conf`:

```
pingAccessAccount soft nofile 10400
pingAccessAccount hard nofile 10400
```

where `pingAccessAccount` is the user account used to run the `PingAccess` java process (or `*` for all users).

## Windows tuning

This section describes tuning recommendations for the Windows (version 7 and up) operating system environment. Implement these recommendations to prevent deployment issues, particularly in high capacity environments. The following settings will increase the performance and capacity of network (specifically the TCP socket) connectivity, enabling `PingAccess` to handle a high volume of concurrent requests.

### Increase the number of available ephemeral ports

1. View ephemeral ports: `netsh int ipv4 show dynamicportrange tcp`
2. Increase ephemeral ports: `netsh int ipv4 set dynamicport tcp start=1025 num=64510`
3. Reboot the machine.
4. View and confirm updated port range: `netsh int ipv4 show dynamicportrange tcp`

### Reduce socket TIME\_WAIT delay

1. Click **Start** > **Run**, type `regedit` and click **OK** to open the Registry Editor.
2. Navigate to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters`.
3. Create a new DWORD value (32 bit) and provide the name `TcpTimedWaitDelay`.
4. Set a decimal value of 30.
5. Reboot the machine.

## 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.

Specify the garbage collector using the `jvm-memory.options` file located in `PA_HOME/conf`. Modify the parameter beneath `#Use the parallel garbage collector` using the information provided below.

Garbage Collector	Description	Modifications
Parallel	<ul style="list-style-type: none"> <li>• Best used with heaps 4GB or less</li> <li>• Full stop-the-world copying and compacting collector</li> <li>• Uses all available CPUs (by default) for garbage collection</li> </ul>	Default collector for server JVM. No modification is required.
Concurrent Mark Sweep (CMS)	<ul style="list-style-type: none"> <li>• Best for heaps larger than 4GB with at least 8 CPU cores</li> <li>• Mostly a concurrent collector</li> <li>• Some stop-the-world phases</li> <li>• Non-Compacting</li> <li>• Can experience expensive, single threaded, full collections due to heap fragmentation</li> </ul>	Set to <code>-XX:+UseConcMarkSweepGC</code> in <code>jvm-memory.options</code> .
Garbage First (G1)	<ul style="list-style-type: none"> <li>• Best for heaps larger than 6GB with at least 8 CPU cores</li> <li>• Combination concurrent and parallel collector with small stop-the-world phases</li> <li>• Long-term replacement for CMS collector (does not suffer heap fragmentation like CMS)</li> </ul>	Set to <code>-XX:+UseG1GC</code> in <code>jvm-memory.options</code> . Also disable <code>#Minimum heap size</code> and <code>#Maximum heap size</code> tuning. Explicit sizing adversely affects pause time goal. To disable, comment the lines out in the script.

## 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. For performance, only one acceptor thread need be used in most situations. On larger multiple CPU core machines, more acceptors may be used.

To modify, open the `run.properties` file located in the `conf` directory of your PingAccess deployment and specify the number of acceptors you want to use on the following lines:

```
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 the following option 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 they are closed by the backend server due to inactivity.

In certain situations it can be advantageous to limit the number of connections in the pool for a given 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.

- ➔ **Info:** 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 documentation for information on setting **Max Connections**.

## 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.

### 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 a 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.



# Clustering reference guide

---

## Clustering

---

PingAccess can be configured in a clustered environment to provide higher scalability and availability for critical services. While it is important to understand that there may be tradeoffs between availability and performance, PingAccess is designed to operate efficiently in a clustered environment.

PingAccess clusters are made up of three types of nodes:

### Administrative Console

Provides the administrator with a configuration interface

### Replica Administrative Console

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

### Clustered Engine

Handles incoming client requests and evaluates policy decisions based on the configuration replicated from the administrative console

Any number of clustered engines can be configured in a cluster, but only one administrative console and one replica administrative console can be configured in a cluster.

Configuration information is replicated to all of the clustered engine nodes and the replica administrative node from the administrative console. State information replication is not part of a default cluster configuration, but some state information can be replicated using PingAccess subclusters.

### PingAccess Subclusters

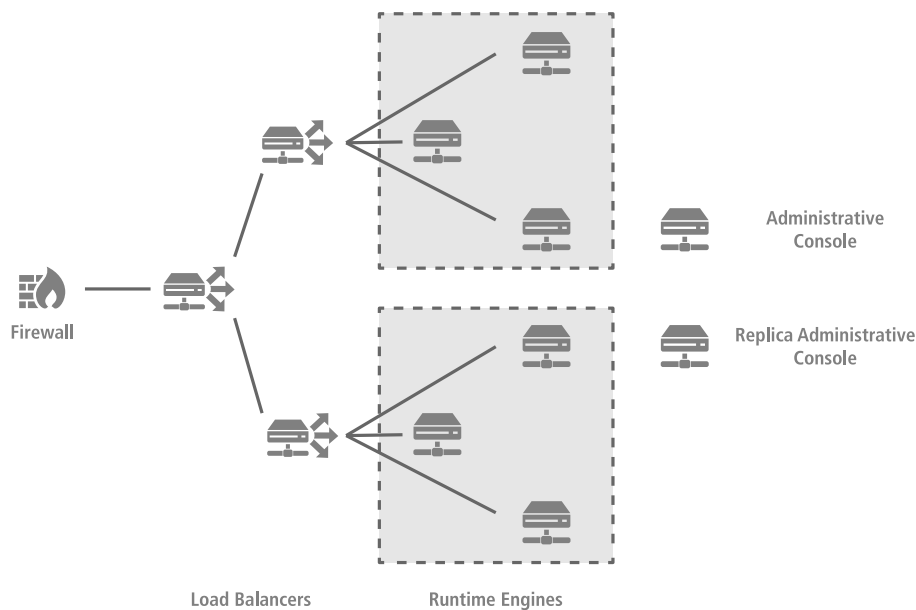
Subclusters are a method to provide better scaling of very large PingAccess deployments by allowing multiple engine nodes in the configuration to share certain information. A load balancer is placed in front of each subcluster in order to distribute connections to the nodes in the subcluster.

Subclusters serve three purposes:

- Providing fault-tolerance for mediated tokens if a cluster node is taken offline.
- Reducing the number of STS transactions with PingFederate when the front-end load balancer does not provide a sticky session.
- Ensure rate limits are enforced properly if the front-end load balancer does not provide a sticky session.

If token mediation and rate limiting are not used in your environment, subclustering is not necessary.

➔ **Info:** This cache can be tuned using the EHCACHE Configuration Properties listed in the **Configuration Properties** documentation.



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.

**Table 1: bootstrap.properties**

<code>engine.admin.configuration.host</code>	Defines the host where the administrative console is available. The default is localhost.
<code>engine.admin.configuration.port</code>	Defines the port where the administrative console is running. The default is 9000.
<code>engine.admin.configuration.userid</code>	Defines the name of the engine.
<code>engine.admin.configuration.keypair</code>	Defines an elliptic curve key pair that is in the JSON Web Key (JWK) format.
<code>engine.admin.configuration.bootstrap.truststore</code>	Defines the truststore, in JWK format, that is used for communication with the administrative console.

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.

- ➔ **Info:** 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**.

Engine nodes include a status indicator that indicates the health of the node and a **Last Updated** field that indicates the date and time of the last update. The status indicator can be green (good status), yellow (degraded status), or red (failed status).

The status is determined by using the value for `admin.polling.delay` as an interval to measure health:

**Green (good status):**

The replica administrative node contacted the primary administrative node on the last pull request.

**Yellow (degraded status):**

The replica administrative node contacted the primary administrative node between 2 and 10 intervals.

**Red (failed status):**

The replica administrative node has either never contacted the primary administrative node, or it has been more than 10 intervals since the nodes communicated.

## Configure cluster prerequisites

---


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

1. Install PingAccess on each cluster node.
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.
  - ➔ **Info:** 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 **Administrative Nodes** 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 > System > Clustering > Administrative Nodes** 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


## Configure a PingAccess cluster

---


Prior to configuring your cluster, PingAccess should be set up as a `STANDALONE` server. The initial node becomes the primary administrative console, and is used to configure the rest of the cluster.

1. Install PingAccess on each cluster node
  -  **Note:** The sequence of the configuration flow is significant: host names first, then generate keys, and then configure the cluster.
2. Navigate to **Settings > System > Clustering > Administrative Nodes** and define the **Primary Administrative Node** as a `host:port` pair.
 


The `host` must be a resolvable DNS name for the node or the node's IP address. The `port` is the TCP port PingAccess listens to for the administrative interface. The default port is 9090.
3. If a replica administrative node will be used in the cluster, perform the following steps:
  - a) Define a `host:port` pair for the replica node.
  - b) Click **Save & Download** to download the replica administrative node configuration file.
  - c) Copy the downloaded `replica1_data.zip` file to the replica administrative node.

 **Note:** If a replica administrative node is added after the cluster has been deployed, it will be necessary to update the configuration for each engine node when the replica administrative node is added.

4. Navigate to **Settings > Security > Key Pairs** and create a new key pair to assign to the `ADMIN` listener. The key pair needs to be valid for both the primary and replica administrative nodes.

 **Tip:** If a replica administrative node is not defined during the initial cluster configuration, you might still opt to define a subject alternative name for a future replica administrative node. This avoids having to reissue the keys when adding the replica administrative node in the future.

5. Open `conf/run.properties` in an editor and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE`.
6. Navigate to **Settings > Networking > Listeners** and assign the newly created key pair to the `CONFIG_QUERY` listener.

 **Note:** If `clusterconfig.enabled` is set to `false` in `run.properties`, the key pair needs to be assigned to the `ADMIN` listener instead. This property is set to `false` when a cluster has been upgraded from PingAccess 3.2 or earlier.

7. Restart PingAccess on the administrative node.

Perform steps 8-10 on the replica administrative node, if one has been configured.

8. Unzip `replica1_data.zip` in the `PA_HOME` directory.
9. Open `conf/run.properties` in an editor and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`.
10. Start PingAccess on the replica administrative node.

For each engine node, perform steps 11-16.

11. Navigate to **Settings > System > Clustering** and click **Add Engine**.
12. After defining the engine's parameters, click **Save & Download** to download the engine configuration zip file.
13. Copy `engine_name_data.zip` to the engine node.
14. On the engine node, unzip `engine_name_data.zip` in the `PA_HOME` directory.
15. On the engine node, open `conf/run.properties` in an editor and change the `pa.operational.mode` value to `CLUSTERED_ENGINE`.
16. Start PingAccess on the engine node.


Navigate to **Settings > System > Clustering** to check your cluster's status. If everything is configured properly, the cluster engine nodes and optional replica administrative node should show a green status icon, indicating that the cluster is operational.


You can optionally configure each node to run PingAccess as a service set to automatically run when the node is started. For more information about configuring PingAccess as a service, see installation documentation for more information.


## Configure the primary administrative node

---

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

 **Warning:** If you are promoting a replica admin node to primary, remove the bootstrap properties file from the replica admin node.

 **Note:** This procedure allows you to specify an HTTP or HTTPS proxy. If proxy configuration is defined in a properties file (`bootstrap.properties` or `run.properties`), it will take precedence over UI or API configuration.

 **Note:** If a proxy is configured on an replica administrative node, when failing over and before removing the bootstrap properties file, the administrative node should have the same proxy configuration.

1. Enter the host and port for the administrative console. The default is `localhost:9000`.
2. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.

3. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.
4. Click **Save**.

## Configure PingAccess subclusters

---

1. Modify `PA_HOME/conf/run.properties` and change the `pa.cluster.interprocess.communication` value from `none` to either `tcp` or `udp`.
  - ➔ **Info:** 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. Update the `pa.cluster.bind.address` with the IP address of the network interface that should handle the interprocess communication traffic for the cluster.
5. Place a load balancer in front of each subcluster to distribute the load across the subcluster nodes.
6. Restart the engine nodes.

## Configure the Replica Administrative Node

---

When using a replica administrative node, it is necessary to define a key pair to use for the CONFIG QUERY 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.

In addition to the configuration below, the Replica Administrative node includes a status indicator that indicates the health of the node and a read-only **Last Updated** field that indicates the date and time of the last update. The status indicator can be green (good status), yellow (degraded status), or red (failed status).

The status is determined by using the value for `admin.polling.delay` as an interval to measure health:

### Green (good status):

The replica administrative node contacted the primary administrative node on the last pull request.

### Yellow (degraded status):

The replica administrative node contacted the primary administrative node between 2 and 10 intervals.

### Red (failed status):

The replica administrative node has either never contacted the primary administrative node, or it has been more than 10 intervals since the nodes communicated.



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


1. Go to **Settings > System > Clustering > Administrative Nodes**.
2. Configure the **Host** value. This name and port pair must match either a subject alternative name in the key pair or be considered a match for the wildcard specified if the key pair uses a wildcard in the common name.
3. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.
4. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.

5. Specify the **Replica Administrative Node Trusted Certificate** to use for cases where a TLS-terminating network appliance, such as a load balancer, is placed between the engines and the admin node.
6. Click **Save & Download** to download the `<replicaname>.data.zip` file for the replica administrative node. PingAccess automatically generates and downloads a public and private key pair into the `bootstrap.properties` file for the node. The **Public Key** is indicated on this screen.
7. Copy the downloaded file to the replica administrative node's `PA_HOME` directory and unzip it.
8. **Conditional:** If the Replica Administrative Node is running on a Linux host, execute the command `chmod 400 conf/pa.jwk`.
9. Edit `PA_HOME/conf/run.properties` on the replica administrative node and change the `pa.operational.mode` value to `CLUSTERED_CONSOLE_REPLICA`.
10. Start the replica node.
11. 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".

## Manual fail 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.

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

1. Open `PA_HOME/conf/run.properties` in an editor.
2. Locate the `pa.operational.mode` line and change the value from `CLUSTERED_CONSOLE_REPLICA` to `CLUSTERED_CONSOLE`


This change is detected while the node is running, and does not require a restart of the node.

## Reinstate 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.

1. Install the new replica node.
2. Change the `run.properties` value for `pa.operational.mode` to `CLUSTERED_CONSOLE_REPLICA`
3. Go to **Settings > System > Clustering > Administrative Nodes** 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.

 **Tip:** 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 **Save & Download** 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.

## Configure an engine


---

For each engine:

1. Click **Settings > System > Clustering**
2. Click **Add Engine** to configure a new engine.
3. Enter a **Name** for the engine. Special characters and spaces are allowed.
4. Enter a **Description** of the engine.
5. If applicable, specify an **HTTP Proxy** for the engine. Click **Create** to create an HTTP proxy.
6. If applicable, specify an **HTTPS Proxy** for the engine. Click **Create** to create an HTTPS proxy.
7. Specify the **Engine Trusted Certificate** to use for cases where a TLS-terminating network appliance, such as a load balancer, is placed between the engines and the admin node.
8. 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.
9. 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.
  - ➔ **Info:** 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.
10. **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.
11. Start each engine.
  - ➔ **Info:** For information on configuring engine to share information with each other in a cluster, see [Configure a PingAccess cluster](#) on page 202.

## Edit an engine

---

1. Navigate to **Settings > System > Clustering**.
2. Expand the engine you want to edit, then click .
3. Edit the engine **Name** or **Description**, as appropriate.
4. If a new public key is needed, click **Save & Download**
5. If a new public key is not needed, click **Save**.


## Remove an engine's access to the Administrative Console

---

1. Navigate to **Settings > System > Clustering**
2. Expand the engine you wish to remove from the cluster and edit it.
3. Click **Delete** under the **Public Keys** heading to revoke engine access to the administrative console.
  - ➔ **Info:** 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. Navigate to **Settings > System > Clustering**.
2. Expand the engine you want to delete and click  to permanently remove all references to the engine from the cluster.
3. Click **Delete** in the confirmation window.



# Deployment guide

---

## 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. This section provides information to help you make the right decisions for your environment.

## 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 minimizing response time.

In both gateway and agent deployment the response from the target resource is processed on the way to the original client. In an agent deployment, the amount of processing is more limited than in a gateway deployment. The agent does not make another request to the Policy Server, so response processing is based on the initial agent response. Consequently, the agent is not able to apply the request processing rules available to the gateway.

When designing a deployment architecture, many requirements and components must be identified for a successful implementation. Proper network configuration of routers/firewalls and DNS ensure that all traffic is routed through PingAccess for the Resources it is protecting and that alternative paths (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.

## Deploy for Gateway Web Access Management

A PingAccess Web access management (WAM) deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to Web-based applications while integrating with existing identity management infrastructure. With growing numbers of internal and external users, and more and more enterprise resources available online, it is important to ensure that qualified users can access only those applications to which they have permission. A WAM environment provides authentication and policy-based access management while integrating with existing infrastructure.

Deployed at the perimeter of a protected network between browsers and protected Web-based applications, PingAccess Gateway performs the following actions:

- Receives inbound calls requesting access to Web applications. Web Session protected requests contain a previously-obtained 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](#)

## Deploy for Agent Web Access Management

A PingAccess Web access management (WAM) agent deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to Web-based applications while integrating with existing identity management infrastructure and minimal network configuration changes. With growing numbers of internal and external users, and more 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](#)

## Deploy for Gateway API Access Management

A PingAccess API access management deployment enables an organization to quickly set up an environment that provides a secure method of controlling access to APIs while integrating with existing identity management

infrastructure. Pressure from an 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](#)

## Deploy for auditing and proxying

A PingAccess deployment for auditing and proxying enables an organization to quickly set up an environment that provides a secure method of controlling access to 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](#)

## 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.

## 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.
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.

Step	Description
Configure the OpenID Connect Relying Party Client for PingAccess.	The client must be registered with PingFederate and the client credentials configured in PingAccess to identify PingAccess when requesting authentication for users trying to access Web applications.
Configure Web session details to enable protection of Web Resources.	Configures settings for secure Web sessions such as timeout values, cookie parameters, and cryptographic algorithms.
Generate or Import Key Pairs and configure HTTP Listeners.	Defines the certificates and keys used to secure access to the PingAccess administrative console and secure incoming HTTPS requests at runtime.
Set up your cluster for high availability.	Facilitates high availability of critical services, and increases performance and overall system throughput.
Add trusted CA certificates.	Defines trust to certificates presented during outbound secure HTTPS connections.
Create a trusted certificate group.	Provides a trusted set of anchor certificates for use when authenticating outbound secure HTTPS connections.
Define virtual servers for protected Resources.	Allows one server to share PingAccess Resources without requiring all Sites on the server to use the same host name. If SNI is available (Java 8), specific key pairs can be assigned to virtual hosts.

## API Access Management Gateway deployment

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
<a href="#">Configure the connection</a> to the PingFederate OAuth Authorization Server.	PingAccess uses this connection and credentials to validate incoming Access Tokens for securing API calls.
<a href="#">Configure the Resource Server OAuth Client.</a>	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

## 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.

## 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.

### Choose Between an Agent or Gateway deployment

PingAccess can be deployed using Agents, as a Gateway (or reverse proxy), or using a combination of both. Before deciding on a deployment, it is important to understand the pros and cons of each deployment scenario and determine how they impact your strategy.

#### Gateway

Pros:

- Fewer number of deployed components that require maintenance
- Independent of target application platform
- No impact on web/app server processing and performance
- Able to work with existing security token types (such as creating 3rd party WAM tokens)

Cons:

- Requires networking changes
- Requires strategy for securing direct access to backend web/app servers (network routing or service level authentication)

- Depending on the application, may require content / request/response rewriting
- Another layer that requires HA/DR planning

### Agents

#### Pros:

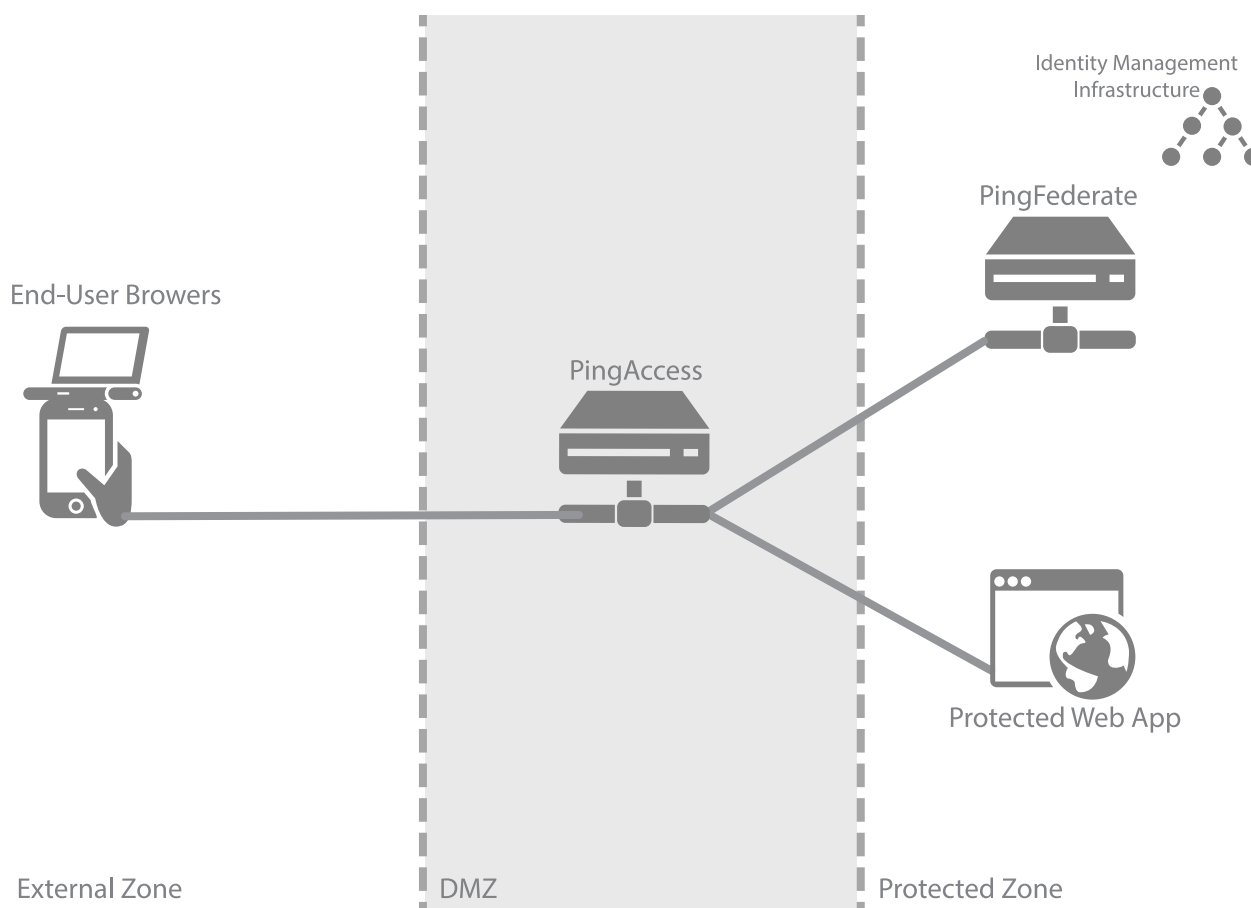
- No networking or server level authentication changes required
- Tight integration with web server handling requests
- Scales with application

#### Cons:

- High cost of ownership when many agent instances deployed, although should be upgradable/patchable independently of PingAccess (policy) server
- Policy evaluation is cached; although it is periodically flushed/re-evaluated (for new sessions, updates to session token, etc.) it isn't quite is "real time" as proxy
- Tight dependency on web server version & platform

## 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.

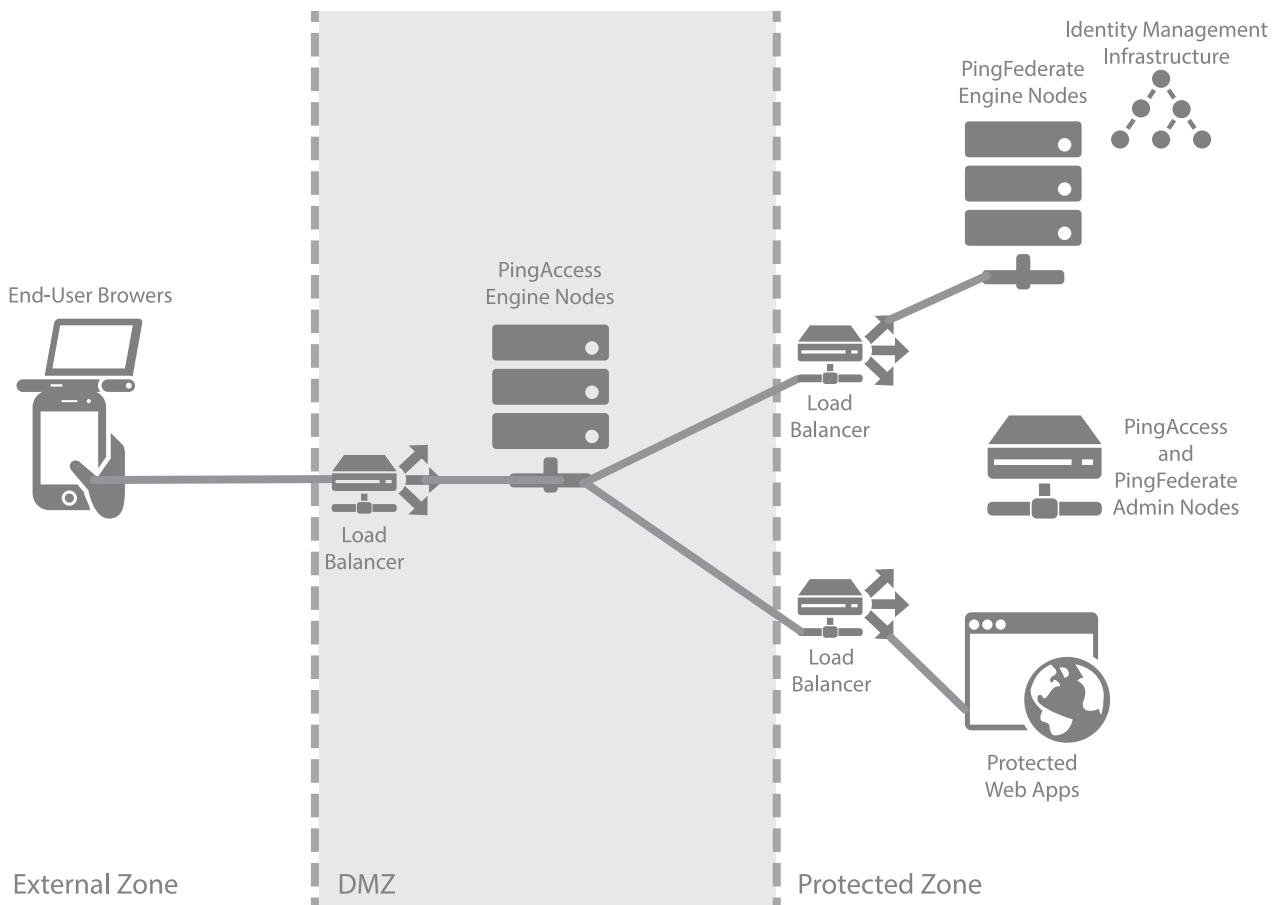
Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ	Externally exposing segment where PingAccess is accessible to Web browsers. PingAccess is a standalone instance in this environment, serving as both a runtime and an administrative port.
Protected 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.

- ➔ **Info:** 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.



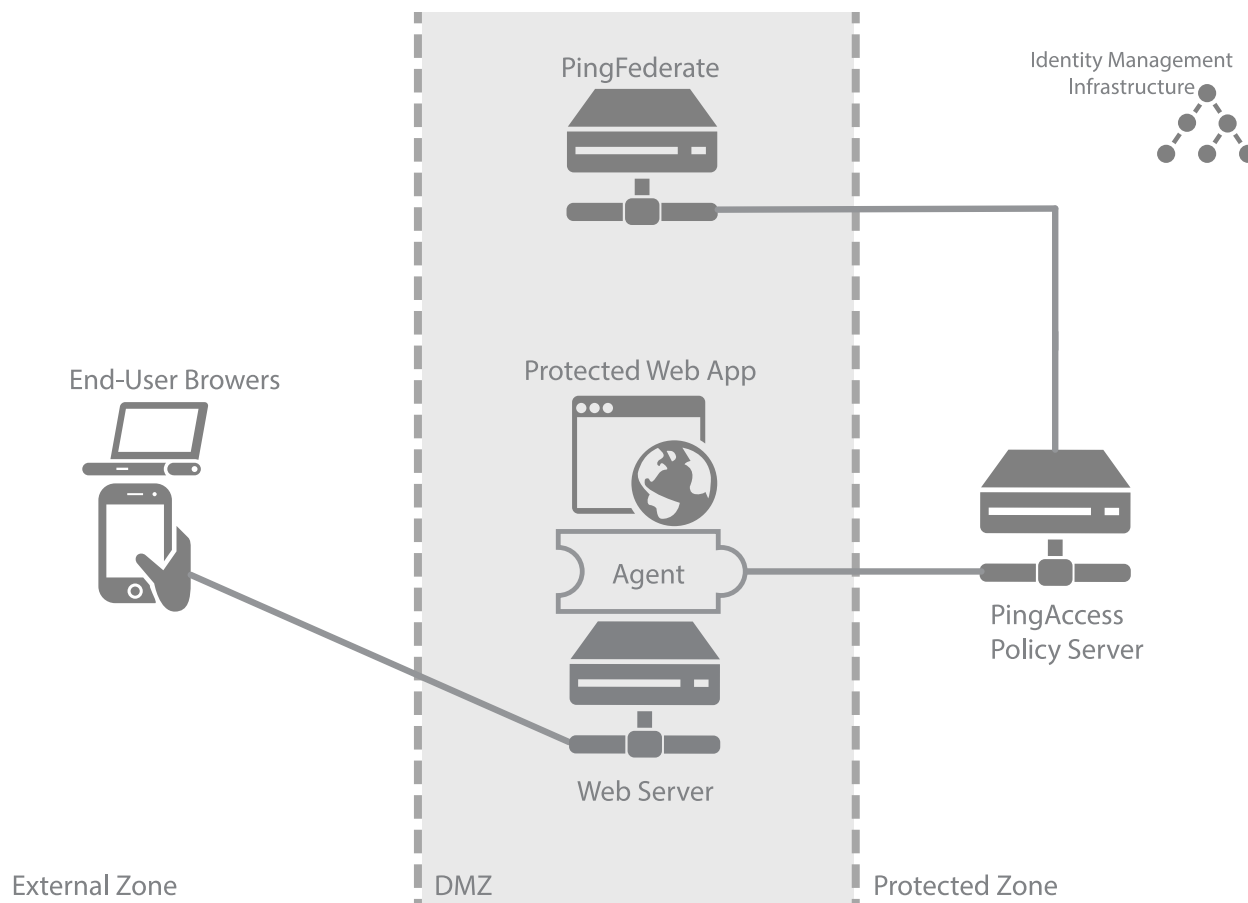


The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ	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.

## 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.

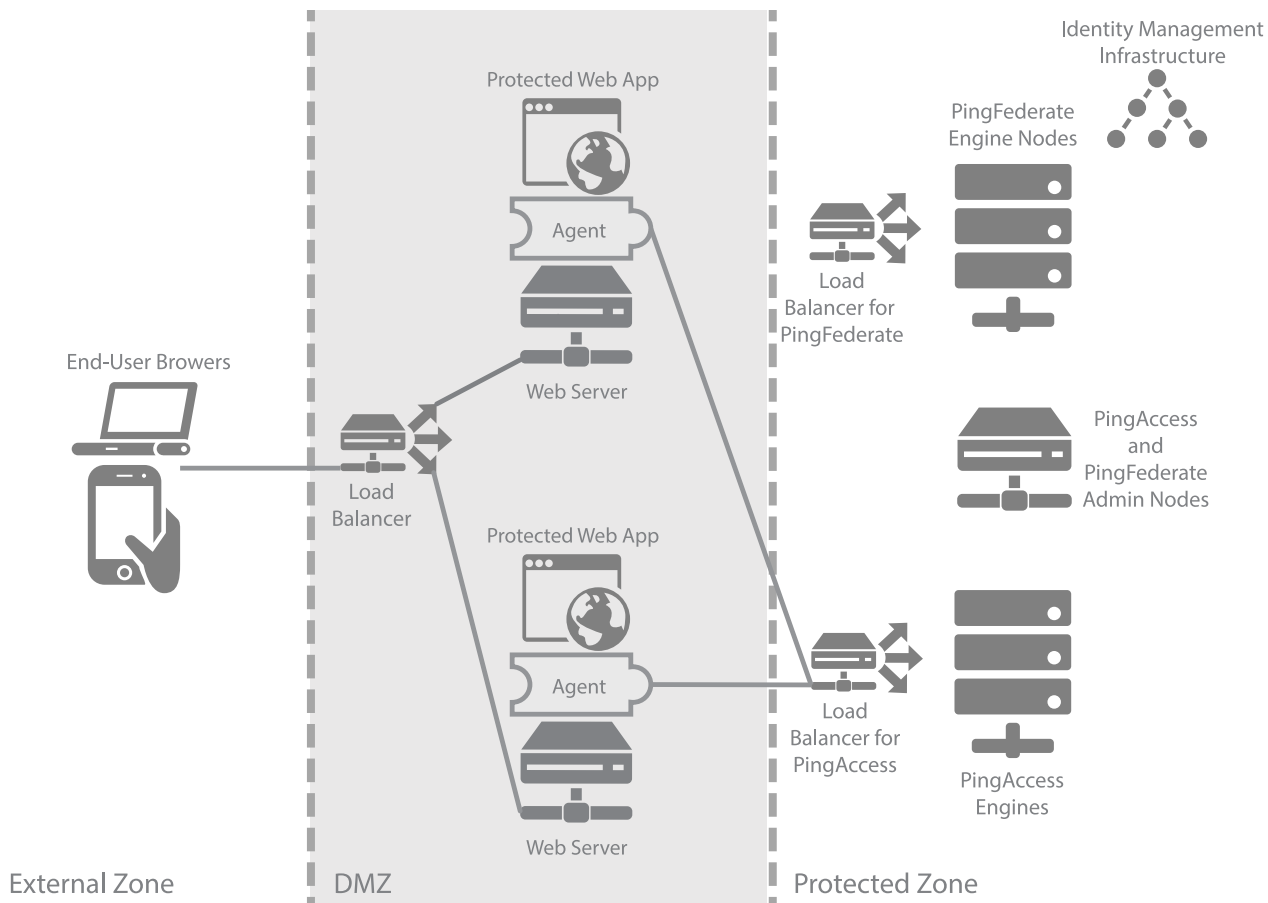


The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ	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.



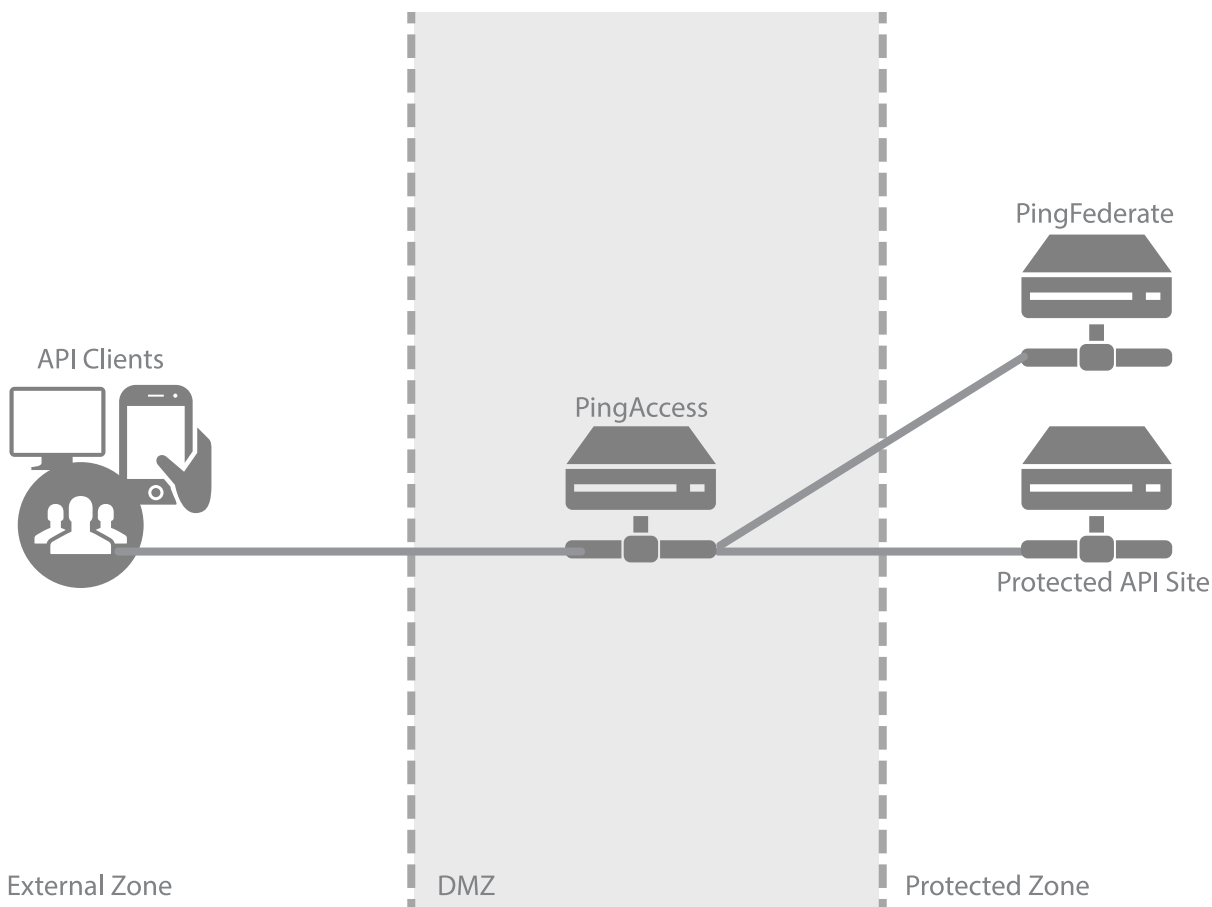
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ	Externally exposed segment where (possibly multiple) application Web servers are accessible to Web clients. PingAccess Agent is deployed as a plugin on these Web servers. Agents interact with PingAccess Policy Server in the Protected Zone.
Protected Zone	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

Zone	Description
	PingAccess Gateway is required to forward client requests through the DMZ. This aspect is not shown in the diagram.

## API Access Management proof of concept deployment architecture

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	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

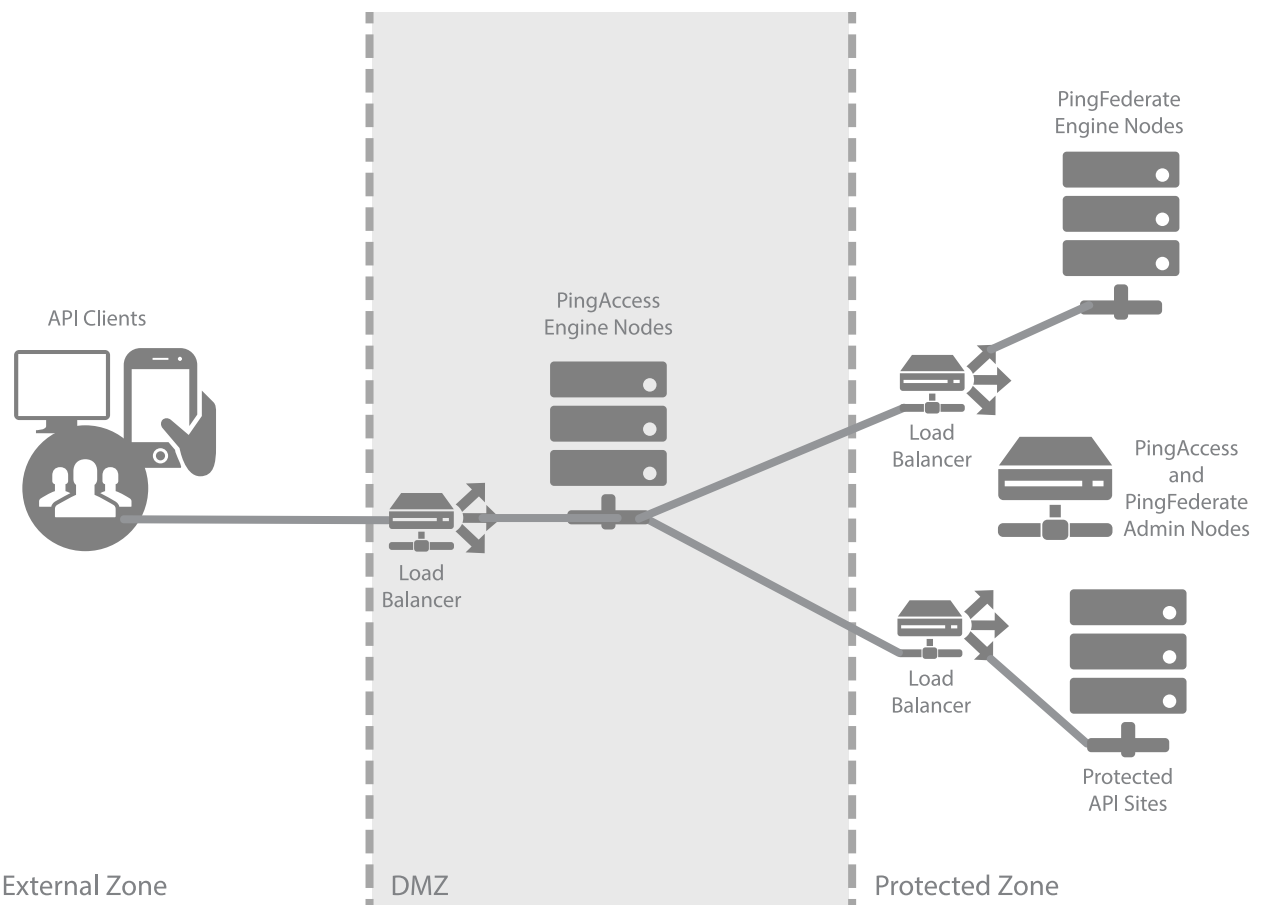
Zone	Description
	is a standalone instance, serving as both a runtime and an administrative port.

## 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.

➔ **Info:** 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.



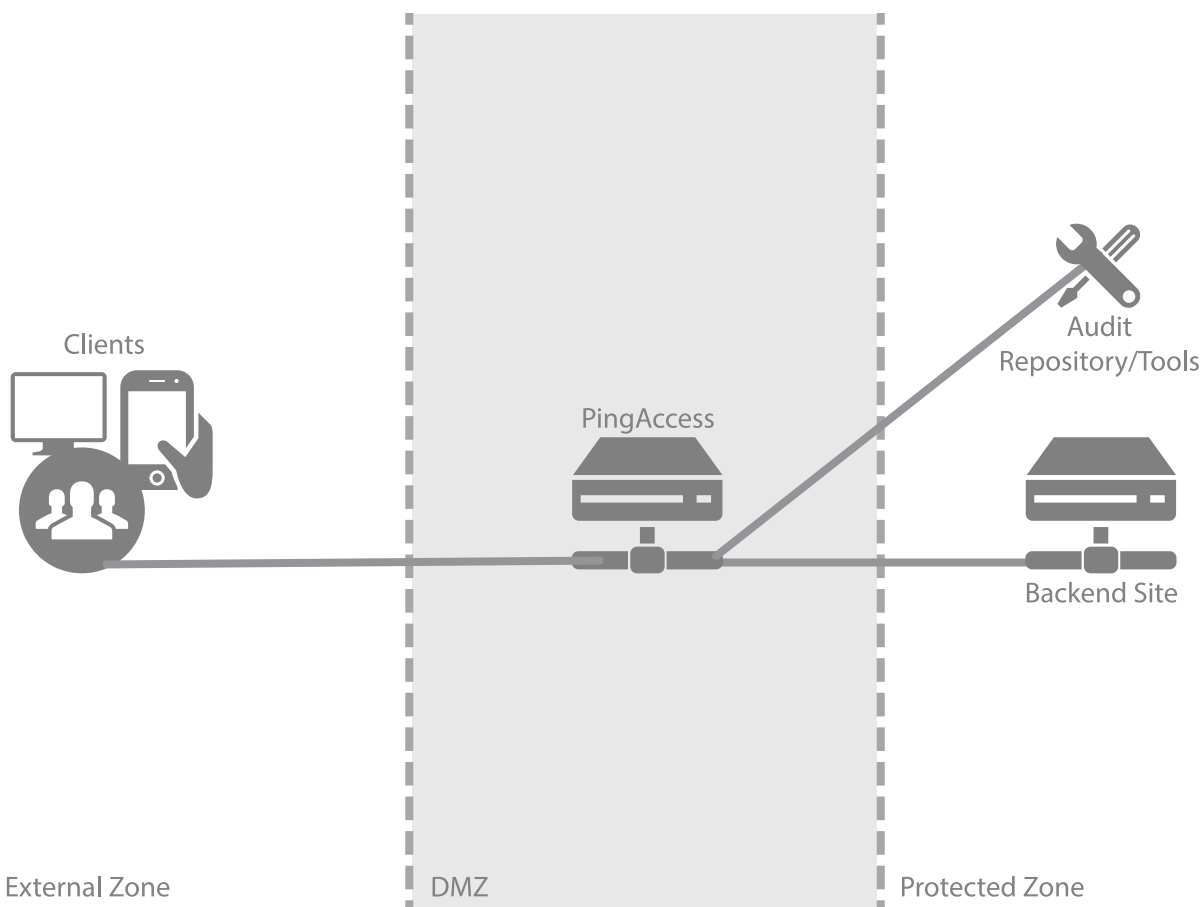
The following table describes the three zones within this proposed architecture.

External Zone	External network where incoming API requests originate.
DMZ	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

Protected Zone	<p>high availability. Depending on your scalability requirements, more nodes may be required.</p> <p>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</p>
----------------	---

## 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	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.

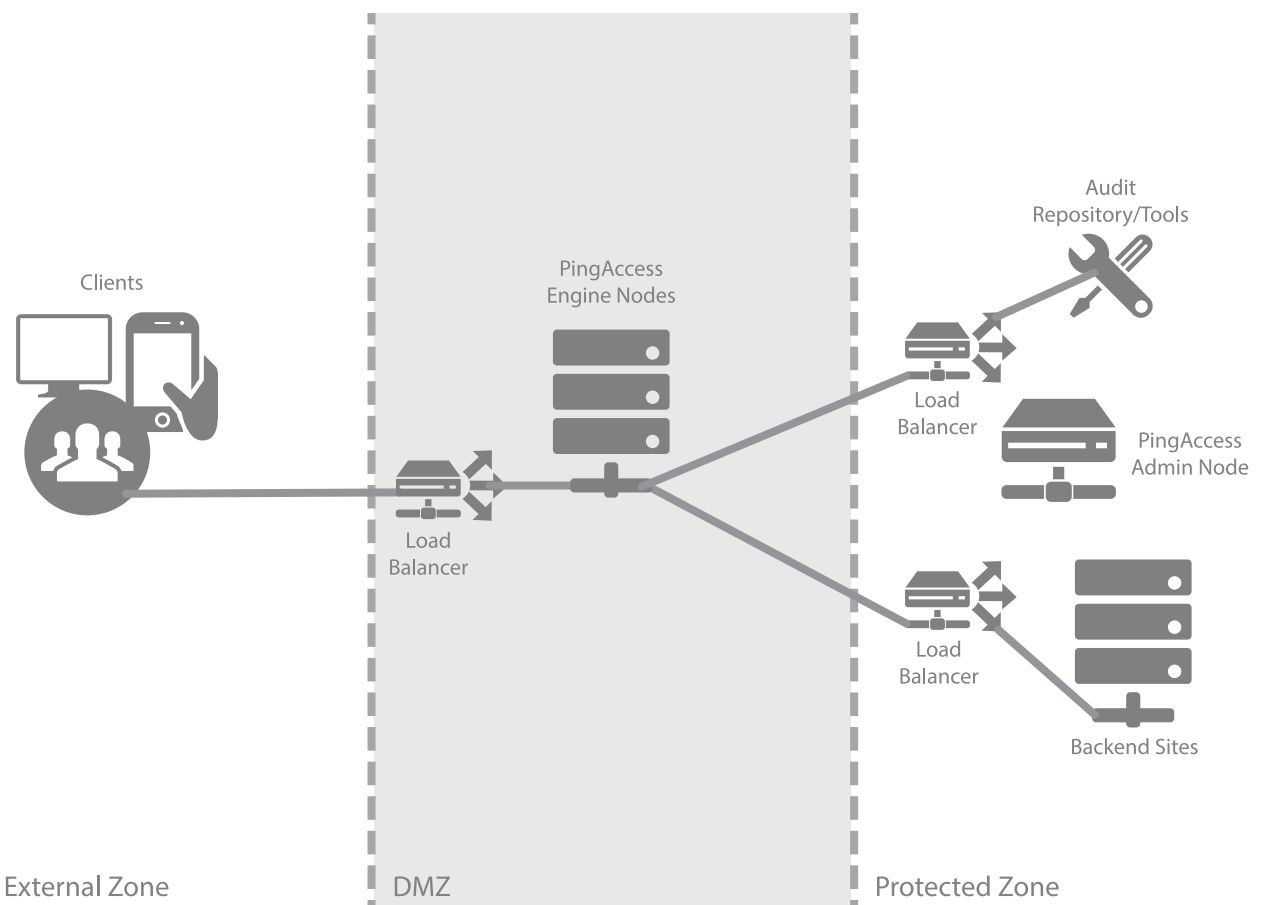
Zone	Description
Protected Zone	Contains back-end Sites audited and proxied through PingAccess. Audit results are sent to an audit repository or digested by reporting tools. Many types of audit repository/tools are supported such as SIEM/GRC, Splunk, database, and flat files.

## 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.

➔ **Info:** 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	Externally exposing segment where PingAccess is accessible to clients. A minimum of two PingAccess

**Protected Zone**

engine nodes will be deployed in the DMZ. Depending on your scalability requirements, more nodes may be required.

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.



# Configuration file reference guide

---

## Configuration file reference

---

This document provides a reference to configurable parameters used by PingAccess at runtime. These parameters are configured in the `run.properties` file located at `<PA_HOME>/conf/`.



**Note:** Changes made to the `run.properties` file will take effect after PingAccess is restarted.



**Tip:** When storing passwords in `run.properties`, we strongly recommend you obfuscate them using the `obfuscate.bat` or `obfuscate.sh` utility to mask the password value. This utility is located in the `PA_HOME/bin` folder.

### **account.locking.max.consecutive.failures**

Defines the maximum number of failed login attempts before locking the account when using basic authentication in the administrative UI or administrative REST APIs. The default value is 3.

### **account.locking.max.lockout.period**

Defines, in minutes, the amount of time to lock an account out from the administrative interfaces after exceeding the `account.locking.max.consecutive.failures`. The default value is 1.

### **admin.acceptors**

Defines the number of admin acceptor threads used to establish connections. The default value is 1.

### **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. Default value is `default`.

### **admin.backlog**

Defines the maximum queue length for incoming admin connection indications. The default value is 512.

### **admin.bindAddress**

Defines the IP address that `admin.port` will bind to. This is typically required on multihomed servers having multiple IP addresses. The default value of `0.0.0.0` means that the port will bind to all of the server's IP addresses.

### **admin.header.Strict-Transport-Security**

Sets the parameters for the Strict-Transport-Security response header sent to the browser when an administrator is interacting with the Admin UI.

### **admin.header.X-Content-Type-Options**

Sets the parameters for the X-Content-Type-Options response header sent to the browser when an admin is interacting with the Admin UI.

### **admin.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser when an admin is interacting with the Admin UI.

### **admin.header.X-XSS-Protection**

Sets the parameters for the X-XSS-Protection HTTP response header sent to the browser when an admin is interacting with the Admin UI.

### **admin.headers**

Additional headers added to responses from the PingAccess Administrator Console and the Administrator API interface. Header values are defined using the `admin.header` prefix.

### **admin.httptransport.coreThreadPoolSize**

Defines the number of threads to keep in the admin transport pool, even if they are idle. The default value is 5.

**admin.httptransport.ioThreads**

Defines the number of I/O threads for the admin host. A value of 0 is used to denote that PingAccess should automatically calculate the appropriate number of I/O threads for the host. The default value is 0.

**admin.httptransport.maxThreadPoolSize**

Defines the maximum number of threads for the admin transport pool. The default value is -1, which denotes no limit.

**admin.httptransport.socketTimeout**

Defines, in milliseconds, the admin socket timeout. The default value is 30000.

**admin.max.request.bodylength**

Defines, in megabytes, the maximum body length for a request to the administrative API endpoint. The default value is 15.

**admin.polling.delay**

Defines, in milliseconds, how long after the initial query to the administrative console that the replica administrative node begins querying for configuration information. The default is every 2000 milliseconds.

**admin.polling.initialdelay**

Defines, in milliseconds, how long after the replica administrative node starts up before it begins to poll the administrative console for configuration information. The default is 500.

**admin.port**

Defines the TCP port on which the PingAccess administrative console runs. Default is 9000.

**admin.reuseAddress**

When enabled, allows a process to bind to a port which remains in a TIME\_WAIT state for the admin transport. The default value is `true`.

**admin.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with administrative HTTPS ports.

**admin.ssl.protocols**

Defines the protocols for use with administrative HTTPS ports.

**admin.ui.max.sessions**

Defines the maximum number of sessions for the admin UI when admin SLO is not enabled.

**agent.assets.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser via the agent when responding to a request for an asset used by a PingAccess template.

**agent.assets.headers**

Additional headers added to responses from PingAccess Agents. Header values are defined using the `agent.assets.header` prefix.

**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.cache.invalidated.response.duration**

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 responses for the changed application. Default value is 900.

**agent.default.token.cache.ttl**

Defines, in seconds, the time to live for cached agent tokens.

**agent.error.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser via the agent when responding with a PingAccess error template.

**agent.error.headers**

Additional headers added to error responses from PingAccess Agents. Header values are defined using the `agent.error.header` prefix.

**agent.http.backlog**

Defines the maximum queue length for incoming admin connection indications. The default value is 512.

**agent.http.bindAddress**

Defines the address from which an engine listens for agent requests.

**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.reuseAddress**

When enabled, allows a process to bind to a port which remains in a `TIME_WAIT` state for the agent transport. The default value is `true`.

**agent.http.secure**

Defines whether the engine is using HTTPS for agent requests. Default is `true`.

**agent.httptransport.coreThreadPoolSize**

Defines the number of threads to keep in the agent transport pool, even if they are idle. The default value is 5.

**agent.httptransport.ioThreads**

Defines the number of I/O threads for the agent host. A value of 0 is used to denote that PingAccess should automatically calculate the appropriate number of I/O threads for the host. The default value is 0.

**agent.httptransport.maxThreadPoolSize**

Defines the maximum number of threads for the agent transport pool. The default value is -1, which denotes no limit.

**agent.httptransport.socketTimeout**

Defines, in milliseconds, the agent socket timeout. The default value is 30000.

**agent.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with agent HTTPS ports.

**agent.ssl.protocols**

Defines the protocols used for communication with agent HTTPS ports.

**as.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with authorization server HTTPS ports.

**as.ssl.protocols**

Defines the protocols used for communication with authorization server HTTPS ports.

**client.ioThreads**

Defines the number of threads for client connections to backend sites. A value of 0 means there is no limit. The default value is 0.

**clusterconfig.acceptors**

Defines the number of cluster configuration acceptor threads used to establish connections. The default value is 1.

**clusterconfig.backlog**

Defines the maximum queue length for incoming cluster configuration connection indications. The default value is 512.

**clusterconfig.bindAddress**

Defines the optional address used for cluster configuration.

**clusterconfig.enabled**

When enabled, uses the cluster configuration port for cluster replication. When disabled, the admin port is used for cluster configuration replication. The default value is `true`.



**Note:** This parameter is set to `false` by the PingAccess Upgrade Utility after a PingAccess cluster is upgraded from a version earlier than 4.0.

**clusterconfig.httptransport.coreThreadPoolSize**

Defines the number of threads to keep in the cluster configuration transport pool, even if they are idle. The default value is 5.

**clusterconfig.httptransport.ioThreads**

Defines the number of I/O threads for the cluster configuration host. A value of 0 is used to denote that PingAccess should automatically calculate the appropriate number of I/O threads for the host. The default value is 0.

**clusterconfig.httptransport.maxThreadPoolSize**

Defines the maximum number of threads for the cluster configuration transport pool. The default value is `-1`, which denotes no limit.

**clusterconfig.httptransport.socketTimeout**

Defines, in milliseconds, the cluster configuration socket timeout. The default value is 30000.

**clusterconfig.port**

Defines the optional port used for cluster configuration.

**clusterconfig.reuseAddress**

When enabled, allows a process to bind to a port which remains in a `TIME_WAIT` state for the cluster configuration transport. The default value is `true`.

**clusterconfig.secure**

When enabled, enables SSL communications for the cluster configuration port. The default value is `true`.

**clusterconfig.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with HTTPS ports in a clustered configuration.

**clusterconfig.ssl.protocols**

Defines the protocols used for communication with HTTPS ports in a clustered configuration.

**enable.detailed.heartbeat.response**

When enabled, this setting enables a customizable heartbeat response to be returned. When disabled, the heartbeat endpoint returns a `200 OK` response. The default value is `false`.

**engine.admin.configuration.audience**

Defines the audience used for cluster authentication. This property must be set to the same value on all nodes in a PingAccess cluster. The default value is `PingAccessAdminServer`.

**engine.assets.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser via the engine when responding to a request for an asset used by a PingAccess template.

**engine.assets.headers**

Additional headers added to responses from the PingAccess Engine. Header values are defined using the `engine.assets.header` prefix.

**engine.error.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options HTTP response header sent to the browser via the engine when responding with a PingAccess error template.

**engine.error.headers**

Additional headers added to error responses from the PingAccess Engine. Header values are defined using the `engine.error.header` prefix.

**engine.http.acceptors**

Defines the number of engine acceptor threads used to establish connections. The default value is 1.

**engine.http.backlog**

Defines the maximum queue length for incoming engine connection indications. The default value is 512.

**engine.http.bindAddress**

Defines the address for an engine in a clustered environment.

**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.reuseAddress**

When enabled, allows a process to bind to a port which remains in a TIME\_WAIT state for the engine transport. The default value is `true`.

**engine.httptransport.coreThreadPoolSize**

Defines the number of threads to keep in the engine transport pool, even if they are idle. The default value is 5.

**engine.httptransport.ioThreads**

Defines the number of I/O threads for the engine host. A value of 0 is used to denote that PingAccess should automatically calculate the appropriate number of I/O threads for the host. The default value is 0.

**engine.httptransport.maxThreadPoolSize**

Defines the maximum number of threads for the engine transport pool. The default value is -1, which denotes no limit.

**engine.httptransport.socketTimeout**

Defines, in milliseconds, the engine socket timeout. The default value is 30000.

**engine.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.

**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.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with engine HTTPS ports.

**engine.ssl.protocols**

Defines the protocols used with engine HTTPS ports.

**engine.websocket.maxConnections**

Sets the maximum number of allowed web socket connections. Default is -1 (unlimited).

**pa.admin.test.connections**

A boolean property that allows the PingAccess admin UI to make HTTP calls to validate that it can reach PingFederate and sites when the user configures them.

**pa.admin.user.password.error.message**

Defines the message returned when password complexity is not satisfied. The default value is Password must be at least 8 characters in length, contain one upper-case letter, one lower-case letter and one digit..

**pa.admin.user.password.regex**

Defines the regex that controls password complexity for the Administration Console. The default value is

```
((?=.**\d)(?=.**[a-z])(?=.**[A-Z]).{8,20})
```

**pa.auditing.unknown.resource**


When set to `true`, this setting causes PingAccess to audit requests for resources that are requested but not mapped to an Application or Resource. This setting can be used to help troubleshoot resource definition issues. The default is `false`.

**pa.backup.filesToKeep**

Defines the number of backup files to preserve when the Administrator authenticates to PingAccess. The default value is 25.

**pa.cluster.auth.pwd**

Sets the key that each engine in the cluster must use to authenticate when joining the group. This prevents unauthorized engines from joining a cluster. This key should be treated as a strong key rather than as a human-readable password value. (Values: any string or blank)

 **Important:** If `pa.cluster.encrypt` is `true`, `pa.cluster.auth.pwd` must not be blank.

**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 7610. Whether this is a TCP or UDP port depends on the value configured for the `pa.cluster.interprocess.communication` property (see above).

**pa.cluster.encrypt**

Indicates whether to encrypt network traffic sent between engines in a cluster. (Values: `true` or `false` [default])

 **Important:** If `pa.cluster.encrypt` is `true`, `pa.cluster.auth.pwd` must not be blank.

**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 7710. Whether this is a TCP or UDP port depends on the value configured for the `pa.cluster.interprocess.communication` property (see above).

**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.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. The default value is `239.16.96.69`.

**pa.cluster.mcast.group.port**

Defines the UDP port associated with the `pa.cluster.mcast.group.address` property above. The default value is 7611.

**pa.cluster.serverstate.replicationIntervalMilliseconds**

Defines, in milliseconds, how often Rate Limiting metadata is replicated within a subcluster. The default value is 1000.

**pa.cluster.serverstate.staleEntryEvictionIntervalSeconds**

Defines, in seconds, how often a PingAccess engine scans the Rate Limiting metadata to evaluate metadata to be removed from the cache, based on the `pa.cluster.serverstate.timeToIdleSeconds` value. The default value is 60.

**pa.cluster.serverstate.timeToIdleSeconds**

Defines, in seconds, how long metadata for the Rate Limiting rule is maintained by a PingAccess Engine after its last use. The default value is 86400.

**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]`.

**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.failedRetryTimeout**

Defines, in seconds, the amount of time to wait before retrying a failed host. The default is 60.

**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.pooledConnectionTimeout**

Defines, in milliseconds, the amount of time to wait before timing out the request for a pooled connection to the target site. The default is -1.

**pa.default.availability.ondemand.readTimeout**

Defines, in milliseconds, the amount of time to wait before timing out the read response for a target site. The default is -1.

**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.contentRewrite.buffer.default**

Defines, in bytes, the default buffer size when using a Rewrite Content rule to do a search and replace of content. The default value is 2048.

**pa.default.contentRewrite.buffer.min**

Defines, in bytes, the minimum buffer size used when using a Rewrite content rule. The default value is 1024.

**pa.default.limitRequestLine**

Defines the maximum number of bytes to read from the request line. The default value is 8192.

**pa.default.maxConnectionsPerSite**

Defines the maximum number of connections PingAccess will open to the PingFederate Admin or Engine. A value of -1 means there is no limit. The default is -1.

**pa.default.maxHeaderCount**

Defines the maximum number of headers to read from a request. The default value is 100.

**pa.default.maxHTTPHeaderSize**

Defines the maximum number of bytes to read when reading headers. The default value is 8192.

**pa.default.maxRequestBodySize**

Defines the maximum number of bytes to read from a request body. The default value is 204800.

**pa.default.session.cookie.attributes.httponly**

Defines the default setting for the **HTTP-Only Cookie** setting for newly-created web sessions. The default value is `true`.

**pa.default.session.cookie.attributes.secure**

Defines the default setting for the **Secure Cookie** setting for newly-created web sessions. The default value is `true`.

**pa.default.session.cookie.size.threshold**

Defines, in bytes, the default maximum session cookie size. The default value is 4093.

**pa.ehcache.AuthTokenCache.maxEntriesLocalHeap**

Defines the maximum size of the JWT identity mapping token cache used when sending tokens to a protected site. Default is 10000.

**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.

**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.PAWamUserAttributesCache.maxEntriesLocalHeap**

Defines the maximum number of entries in the local heap for the PA WAM user attribute cache. The default is 10000.

**pa.ehcache.PAWamUserAttributesCache.timeToIdleSeconds**

Defines, in seconds, the time an entry in the PA WAM user attribute cache can be idle before it is expired. The default is 120 seconds.

**pa.ehcache.PAWamUserAttributesCache.timeToLiveSeconds**

Defines, in seconds, the maximum time an entry can be in the PA WAM user attribute cache. The default 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.

**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.SessionStateCache.maxEntriesLocalHeap**

Defines the maximum size of the identity attribute entry cache when the user's attributes are stored on the server rather than as a cookie. Default is 10000.

**pa.interceptors.relativepath.decode.count**

Number of times the URL is decoded to check for path traversal characters. The default is 3.



**pa.interceptors.relativepath.decode.regex**

Defines the accepted URL regex pattern that administrators can customize based on their needs. The default value is: Defines the regular expression to use when checking for a valid path in an incoming request. The default value is

```
[\\p{Po}\\p{N}\\p{Z}\\p{L}\\p{M}\\p{Zs}\\. /_\\-\\|\\~()\\{\\}\\}\\[\\]]*
```



**Note:** This value is double-escaped as required by the `java.util.regex.Pattern` Java class.

**pa.interceptors.relativepath.strict**

When this property is set to `true`, the incoming URL is matched with the whitelist pattern defined in `pa.interceptors.relativepath.decode.regex`. All other request URLs are rejected. The default value is `false`.

**pa.jdbc.filepassword**

Defines the password used to encrypt the PingAccess configuration database. Default is `2Access`.

**pa.jdbc.password**

Defines the password for the database user of the PingAccess configuration database. Default is `2Access`.

**pa.jdbc.username**

Defines the username for accessing the PingAccess configuration database. Default is `sa`.

**pa.keystore.pw**

Defines the password for the `$JAVA_HOME/lib/security/cacerts` keystore.

**pa.localization.missing.message.placeholder**

Defines the message used when an error message is unresolvable. An error will be logged.

**pa.localization.resource.bundle.cache.enable**

When set to `false`, allows language files in `/conf/localization` to be added or modified. When `true`, enables caching of language files and properties.

**pa.oidc.logout.redirect**

A boolean property that defines whether or not to redirect the user to the token provider on logout. If `true`, the user is redirected to the token provider.

**pa.oidc.logout.redirectURI**

The URI that a user gets sent to when they log out.

**pa.oidc.post.preservation.encrypt**

When enabled, POST data preserved through a redirection to PingFederate for authentication is encrypted on the client to be used after the authentication is successful. The default value is `false`.

**pa.oidc.post.preservation.maxRequestBodySize**

Defines, in bytes, the maximum size of the post body for POST preservation. The default value is `8192`.

**pa.oidc.post.preservation.paramsAttributeName**

Used to store the encoded or encrypted POST payload in the browser session storage during POST preservation.

**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.
  - ➔ **Info:** Only one engine in a cluster can run the administrative console.
- `CLUSTERED_CONSOLE_REPLICA` - Use this value for the server instance you want to use as the backup administrative console server.
- `CLUSTERED_ENGINE` - Use this value to indicate a server engine.

**Note:**

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

- Define all 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.

**pa.uri.strict**

When enabled, this setting requires the raw input URI be in strict compliance with the URI spec implemented by `java.net.URI` when generating URIs. The default value is `false`.

**pf.api.keepAliveTimeout**

Defines, in milliseconds, the keep alive timeout for the PingFederate API. The default value is `30000`.

**pf.api.maxConnections**

Defines the maximum number of connections PingAccess will establish to the PingFederate API endpoint. A value of `-1` means there is no limit. The default value is `-1`.

**pf.api.maxRetries**

Defines the maximum number of retries PingAccess attempts to make to the PingFederate server before declaring the server unavailable. The default value is `0`.

**pf.api.readTimeout**

Defines, in milliseconds, how long the API will wait for responses from PingFederate when making calls to the PingFederate Admin API. The default value is `-1`.

**pf.api.socketTimeout**

Defines, in milliseconds, the socket timeout for the PingFederate API endpoint. The default value is `5000`.

**pf.redirect.header.X-Frame-Options**

Sets the parameters for the X-Frame-Options value that is sent when the user is redirected to PingFederate to authenticate.

**pf.redirect.headers**

Additional headers added to the redirection response that sends the client to PingFederate for authentication. Header values are defined using the `pf.redirect.header` prefix.

**pf.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with PingFederate HTTPS ports.

**pf.ssl.protocols**

Defines the protocols used for communication with PingFederate HTTPS ports.

**provider.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with Provider HTTPS ports.

**provider.ssl.protocols**

Defines the protocols used for communication with Provider HTTPS ports.

**rule.error.headers**

Additional headers added to responses that result from policy rule results. Header values are defined using the `rule.error.header` prefix.

**site.ssl.ciphers**

Defines the type of cryptographic ciphers available for use with Site HTTPS ports.

**site.ssl.protocols**

Defines the protocols used for communication with Site HTTPS ports.

**tls.default.cipherSuites**

Defines the default set of ciphers used for HTTPS communication.

**tls.default.protocols**

Defines the default protocols used for HTTPS communication.

# PingAccess for AWS: Solution setup guide

---

## Introduction

---

This document provides the steps required to perform a variety of tasks related to setting up and configuring the PingAccess for AWS solution. You can choose to deploy this solution in a demonstration environment that includes a basic PingFederate instance or you can choose to deploy with an *existing PingFederate instance*.

Review the *prerequisites* prior to setting up this solution.

After you have deployed the solution, you can *configure an application*.

## Setup and deployment

---

In this section, you will perform the tasks required to setup and deploy the PingAccess for AWS solution. Along with PingAccess, this solution includes the optional deployment and configuration of a basic PingFederate instance.

When setup is complete, you will be able to configure a protected application using PingFederate as the token provider.

To setup and deploy the solution, you will:

1. Review the setup *AWS prerequisites* on page 235
2. *Obtain the automation ZIP file* on page 237
3. *Create the PingAccess software repository* on page 237
4. *Populate the S3 bucket* on page 237
5. *Create the PingAccess key pair* on page 238
6. *Create the PingAccess VPC* on page 238
7. *Configure Security Groups* on page 243
8. *Access the PingAccess administration console* on page 243
9. *Access the PingFederate administration console* on page 244

## AWS prerequisites


Prior to beginning the setup of this solution, you must first satisfy some prerequisites. These prerequisites will ensure that you have details and resources that are required for use in the setup phase. You may have one or more of these items available as part of an existing deployment. Create the items you require.

Prerequisites include the following items:

1. *Create an Amazon Web Services account* on page 235
2. *Choose your AMIs*
3. *Register domain name with Route53*
4. *Create a certificate*


### Create an Amazon Web Services account

To make use of this solution, you need an *Amazon Web Services* account. Create an account if necessary.

 **Important:** While many of the components required for this solution are *Free Tier* eligible, following the steps in this document may lead to incurred costs.

## Choose your AMIs

Throughout the setup process, you will be asked to specify the ID of the Amazon Machine Image (AMI) that you want to use. AMIs are region-specific, so you will require an *AMI that is available to your region* with Signature Version 2 support.

 **Important:** AMIs available from AWS have default configurations that may not meet the security requirements of some network environments. Administrators are encouraged to assess the default configurations of the AMIs and make necessary modifications to satisfy the security requirements of their environment as needed.

At this time, you may only use the AMI types specified in this document. Your configuration may include the use of only a single Amazon Linux AMI ID for all required AMI IDs, but you may choose a combination where possible using the following supported types:


- **Bastion host AMI ID:** Amazon Linux
- **Admin AMI ID:** Amazon Linux or RHEL 7.4
- **Engine AMI ID:** Amazon Linux or RHEL 7.4
- **PingFederate AMI ID:** Amazon Linux

The EC2 instance type is specified by the templates you will use. For optimal performance, use the instance type specified.

To find the ID of the AMI you want to use:

1. In AWS, navigate to **Services > Compute > EC2**.
2. On the EC2 Dashboard, click **Launch Instance**. This page displays the AMIs available in your selected region.
3. Make note of the AMIs you want to use, along with their AMI ID. In the following example, the AMI ID is *ami-9fa343e7*:

**Red Hat Enterprise Linux 7.4 (HVM), SSD Volume Type - ami-9fa343e7**

 **Note:** If you are customizing the AMIs used by the setup templates, you must modify `/pingaccess/templates/security/internal-ca.yaml` to specify the AMI ID you want to use. By default, the specified AMI is `ami-bf4193c7`.


For more information on AMIs, see [AWS AMI documentation](#).

## Register a domain name with Route53

During configuration, you will be asked to specify a **Public hosted zone ID**. The public hosted zone ID maps to a domain registered with **AWS Route 53**. The domain will be used to form URLs needed for external access to a variety of components in this solution, including applications, and the PingAccess and PingFederate administration consoles.

You can purchase a domain name using Route 53, or you can transfer an existing domain to this service.

To learn more about AWS Route 53, including how to *register a domain* or transfer an existing domain, see [AWS Route 53 documentation](#).


 **Tip:** When your domain name is available in Route 53, make note of the associated **Hosted Zone ID** so you can use it during configuration.

## Create a certificate

During configuration, you will be asked to specify a Certificate ID. The certificate ID will map to a certificate issued for the domain you will use, as specified by the Public hosted zone ID.

You can request a certificate or you can import a certificate using the **AWS Certificate Manager**.

To learn more about AWS Certificate Manager, including how to request or import a certificate, see [AWS Certificate Manager documentation](#).

-  **Tip:** When your certificate is available in Certificate Manager, make note of the associated **ARN** for the certificate so you can use it during configuration.


## Obtain the automation ZIP file

The first step in creating your PingAccess environment in AWS is to obtain the ZIP file that includes templates and scripts used by the automation process.

Contact your Ping Identity sales representative to obtain the automation ZIP file.

## Create the PingAccess software repository

In order for the automation to run successfully, you must first create a stack that will be referenced by the setup scripts. You will submit a template to [CloudFormation](#) that creates the S3 bucket, and also creates policies that define access to the bucket that are referenced by provisioning functions.

-  **Important:** Do not manually create the S3 bucket. Use the templates described in this document.

**Prerequisite:** This procedure assumes you have downloaded and unzipped the automation ZIP file.

To create the PingAccess S3 bucket stack and required policies:


1. In the AWS console, navigate to **Services > Management Tools > CloudFormation**.
2. In the CloudFormation console, click **Create Stack**.
3. On the Select Template screen, click to select **Upload a template to Amazon S3**.
4. Click **Choose File**.
5. Navigate to the root of the unzipped archive, select the **create-repo.yaml** template, and click **Open**.
6. Click **Next**.
7. On the **Specify Details** screen, specify the **Stack Name**. For example, *pingaccess-s3-stack*.
8. Specify the **Bucket Name**. For example, *pingaccess-s3-bucket*.


-  **Important:** Bucket names must be unique *across all of AWS*. If your chosen bucket name is in use, the template will fail.

9. Click **Next**.
10. On the **Options** screen, click **Next**.
11. On the Review screen, click to select the **I acknowledge that AWS CloudFormation might create IAM resources** check box.
12. Click **Create**.
13. When stack creation is complete, view your S3 bucket at **Services > Storage > S3**.

## Populate the S3 bucket

After you create the PingAccess software bucket in S3, you will populate your bucket with the files needed for the automation process. The automation ZIP file you downloaded contains the required directory structure. You will add files to this directory structure and upload a set of folders to S3.

-  **Note:** Do not modify the directory structure. Modifying the directory structure will cause the setup process to fail.

-  **Note:** If you are planning to redeploy with an existing configuration, see [Redeploying with an existing configuration](#) on page 246.

### Add files to the directory structure

Prior to uploading to S3, add files that are required for setup. These files include:

- The **Java Server JRE** - Copy the Server JRE install package to `s3-bucket-root/jre/server-jre-<version>-linux-x64.tar.gz`.
- The **PingAccess distribution package** - Copy the PingAccess 5.0.0 distribution package to `s3-bucket-root/pingaccess/dist/pingaccess-5.0.0.zip`.

- The **PingAccess license file** - Copy the PingAccess license file to `s3-bucket-root/pingaccess/licenses/pingaccess.lic`.
- The **PingFederate distribution package** (Optional) - Copy the PingFederate 8.4.1 distribution package to `s3-bucket-root/pingfederate/dist/pingfederate-8.4.1.zip` if you are deploying the basic PingFederate instance for a demonstration environment.
- The **PingFederate license file** (Optional) - Copy the PingFederate license file to `s3-bucket-root/pingfederate/licenses/pingfederate.lic` if you are deploying the basic PingFederate instance for a demonstration environment.
- Additional **library JAR files** (Optional) - Copy any additional library JAR files (e.g. plugins) that should be included with the installation to `s3-bucket-root/automation-artifacts/<export-prefix>/pingaccess/lib`, where `<export-prefix>` is the **Export name prefix** you will supply to the [automation template](#).


### Upload the directory structure to S3

1. In the AWS console, navigate to **Services > Storage > S3**.
2. Select the bucket you created.
3. Click **Upload**.
4. In a file window, select the contents of the `s3-bucket-root` folder. The contents include the following folders and their contents:
  - `automation-artifacts`
  - `instance-init-util`
  - `internal-ca`
  - `jre`
  - `pingaccess`
  - `pingfederate`
5. Drag and drop the folders into the AWS **Upload** window.
6. Click **Next** (x3).
7. On the final screen, click **Upload**.


## Create the PingAccess key pair

This solution will require the use of a key pair for SSH access. You can import an existing key pair, use a key pair that already exists in your AWS environment, or you can create new key pair.

### Create a new key pair

1. In AWS, navigate to **Services > Compute > EC2**.
2. In the left menu bar, under the **Network & Security** heading, click **Key Pairs**.
  -  **Tip:** To import an existing key pair, select **Import Key Pair** on the Key Pairs page.
3. Click **Create Key Pair**.
4. Type the **Key pair name** and click **Create**.

The private key is automatically downloaded to your system, while the public key is stored in AWS.

 **Important:** Save the private key to a safe location.

## Create the PingAccess VPC

This document provides the steps required to create VPC environment. The automation script includes functions to create all of the required components for this deployment, including:

- PingAccess Administration instance
- PingAccess Engine instances
- Load balancers
- Routing configuration

- Security groups configuration
- PingFederate instance (Optional)

**!** **Important:** If you are planning to deploy the PingAccess for AWS solution along with an existing PingFederate instance, your PingFederate configuration must meet the components requirements expected by the automation. To review the PingFederate configuration requirements, see [PingAccess for AWS: PingFederate environment requirements](#).

**☰** **Note:** If you are redeploying with an existing configuration, see [Redeploying with an existing configuration](#) on page 246.

#### To create the complete VPC environment:

1. In AWS, navigate to your S3 bucket.
2. Navigate to `/pingaccess/templates` and click on the `create-pingaccess.yaml` template.
3. On the **Overview** tab, copy the template link.
4. Navigate to **Services > Management Tools > CloudFormation**.
5. Click **Create Stack**.
6. Select the option to **Specify an Amazon S3 template URL** and paste the template link you copied.
7. Click **Next**.
8. On the **Specify Details** page, provide the required information using the [Configuration options](#) table as a guide.
9. Click **Next**.
10. **(Optional)** To disable rollback on failure for troubleshooting purposes, on the **Options** page, expand the **Advanced** heading and set **Rollback on failure** to **No**.
11. Click **Next**.
12. On the **Review** page, scroll to the bottom and select **I acknowledge that AWS CloudFormation might create IAM resources with custom names**.
13. Click **Create**.

You will be taken to the **CloudFormation - Stacks** page where you can monitor setup progress. When the setup is successful, the parent stack and all child stacks will show a status of **CREATE\_COMPLETE**.

#### To deploy PingAccess into an existing VPC

You can deploy PingAccess into an existing VPC by following the same general steps as above while using the `pingaccess-deploy.yaml` template. If you wish, you can create a base VPC using the `create-base-vpc.yaml` template.

#### Configuration options

<b>Stack name</b>	The name of the parent stack. For example: <code>pingaccess-demo</code> .
<b>Export name prefix</b>	The prefix to be applied to all stack outputs imported and exported. For example: <code>pingaccess-stack</code> . You can create multiple clusters by using a unique <b>Export name prefix</b> for each.
<b>Ping S3 repo stack name</b>	The name of the stack used to establish the PingAccess software repository in S3. Use the <b>Stack Name</b> you specified in <a href="#">Create the PingAccess software repository</a> on page 237. For example: <code>pingaccess-s3-stack</code> .
<b>VPC name</b>	The value applied to the VPC Name label. For example: <code>pingaccess-vpc</code> .
<b>VPC CIDR block</b>	The unique network CIDR block to be covered by the created VPC.



<b>Number of public subnets specified</b>	The number of public subnet CIDR blocks to be specified.
<b>Public subnet CIDR blocks</b>	A comma-delimited list of public subnet CIDR blocks. Must be unique subnets of VPC CIDR block.
<b>Availability zones for public subnets</b>	A comma-delimited list of Availability Zones per public subnet. At least two subnets must be in separate Availability Zones.
<b>Number of private subnets specified</b>	The number of private subnet CIDR blocks to be specified.
<b>Private subnet CIDR blocks</b>	A comma-delimited list of private subnet CIDR blocks. Must be unique subnets of VPC CIDR block.
<b>Availability zones for private subnets</b>	A comma-delimited list of Availability Zones per private subnet. At least two subnets must be in separate Availability Zones.
<b>The allowed tenancy of instances launched into the VPC</b>	The allowed tenancy of instances launched into the VPC.
<b>Admin Node Private Host Name</b>	The admin node host name to associate with the internal ELB.
<b>Internal private domain name</b>	The domain name to use for the private hosted zone for Ping Identity infrastructure internal DNS names. For example: <code>pingaccess.aws</code> .
<b>Public hosted zone ID</b>	The public hosted zone for use with external Route 53 DNS. You must have a public domain name registered with Amazon Route53. For more information, see <a href="#">Route53 documentation</a> .  For example: <code>my-aws-domain.com</code> .
<b>Admin public host name</b>	The fully qualified domain name to associate with the internet facing ELB for the admin API. This should be a unique subdomain of the specified Public hosted zone ID. This entry <b>must</b> be followed by a period.  For example: <code>admin.my-aws-domain.com</code> .
<b>Applications public host name</b>	The fully qualified domain name to associate with the internet facing ELB for the engines. This should be a unique subdomain of the specified Public hosted zone ID. This entry <b>must</b> be followed by a period.  For example: <code>app.my-aws-domain.com</code> .
<b>PingAccess Admin API port</b>	The TCP/IP port of the PingAccess Admin API listener. Use the default of 9000.
<b>PingAccess configuration query port</b>	The TCP/IP port of the Admin Config Query listener. Use the default of 9090.
<b>SSH port</b>	The TCP/IP port for the VPC bastion host and PingAccess instance SSH access. Use the default of 22.

<b>EC2 keypair name</b>	The EC2 Key Pair for SSH access to the VPC bastion host and PingAccess instances. Use the key pair you created or imported.
<b>Bastion host AMI ID</b>	The Amazon Linux AMI ID for the bastion host.
<b>Bastion host instance type</b>	The EC2 instance type for the bastion host. Use the default of <code>t2.micro</code> or higher.
<b>Admin AMI ID</b>	The Amazon Linux or RHEL 7.4 AMI ID for Admin instances.
<b>Admin instance type</b>	The EC2 instance type for Admin nodes. Use the default of <code>m3.large</code> or higher.
<b>Administrative node root volume name</b>	The root volume device name for the specified Admin AMI. Commonly, Amazon Linux uses <code>/dev/xvda</code> and RHEL 7.4 uses <code>/dev/sda1</code> .
<b>Administrative node root volume size</b>	The size, in GB, for the root volume on the Admin instance. Use the default of 15 or higher.
<b>Engine AMI ID</b>	The Amazon Linux or RHEL 7.4 AMI ID for Engine node instances.
<b>Engine instance type</b>	The EC2 instance type for Engine nodes. Use the default of <code>m3.large</code> or higher.
<b>Engine node root volume name</b>	The root volume device name for the specified Engine AMI. Commonly, Amazon Linux uses <code>/dev/xvda</code> and RHEL 7.4 uses <code>/dev/sda1</code> .
<b>Engine node root volume size</b>	The size, in GB, for the root volume on Engine instances. Use the default of 16 or higher.
<b>Engine node root volume IOPS</b>	The provisioned IOPS for the root volume on Engine instances. The maximum ratio of IOPS to volume size is 50:1.
<b>Admin API certificate ID (ARN)</b>	The ARN of the certificate to use with Admin API ELB listener. Use the certificate you created or imported.
<b>Engines application certificate ID (ARN)</b>	The ARN of the certificate to use with Engine API ELB listener. Use the certificate you created or imported.
<b>Internal CA hostname</b>	The unqualified hostname for the Internal CA within the private hosted zone.
<b>Root CA Certificate TTL</b>	The amount of time that the Root CA certificate will be valid before it expires. The default is 20 years (20y).
<b>Certificate Generation Token TTL</b>	Time to live for Vault tokens issued for certificate generation. The default is 1 year (1y).
<b>Leaf Certificate Default TTL</b>	The default time to live for generated leaf certificates before they expire. The default is 30 days (30d).
<b>Leaf Certificate Max TTL</b>	The maximum time to live allowed for leaf certificate generation requests.
<b>Certificate Renewal Threshold</b>	The number of days before a certificate expires that it should be renewed.

<b>Certificate Renewal Schedule</b>	The schedule expression that determines how often checks for certificate renewal should occur. For more information, see <a href="#">AWS documentation for scheduled events</a> .
<b>Provision PingFederate</b>	Specify whether or not you want to provision a PingFederate instance. <ul style="list-style-type: none"> <li>• Select <b>True</b> to provision a PingFederate instance in the demonstration environment.</li> <li>• Select <b>False</b> to use an existing PingFederate instance. If you select <b>False</b>, you must ensure your PingFederate configuration meets the requirements expected by the automation. To review the required PingFederate setup, see PingAccess for AWS: PingFederate environment requirements.</li> </ul>
<b>PingFederate Version</b>	Indicates the PingFederate version expected in the S3 repository file name pingfederate- <b>&lt;PingFederateVersion&gt;</b> .zip. For example, pingfederate- <b>8.4.4</b> .zip.  This field is ignored if <b>Provision PingFederate</b> is set to <b>False</b> .
<b>PingFederate DNS Name</b>	The fully qualified domain name to associate with the internet facing PingFederate instance. This should be a unique subdomain of the specified Public hosted zone ID. This entry <b>must</b> be followed by a period.  For example: pf.my-aws-domain.com.  This field is ignored if <b>Provision PingFederate</b> is set to <b>False</b> .
<b>PingFederate Admin API Port</b>	The PingFederate Admin API port. Use the default of 9999.  This field is ignored if <b>Provision PingFederate</b> is set to <b>False</b> .
<b>PingFederate Runtime Port</b>	The PingFederate Runtime port. Use the default of 9031.  This field is ignored if <b>Provision PingFederate</b> is set to <b>False</b> .
<b>PingAccess Audit Logs Retention</b>	The number of days to retain CloudWatch audit logs.
<b>PingAccess Server Logs Retention</b>	The number of days to retain CloudWatch server logs.
<b>Lambda Logs Retention</b>	The number of days to retain CloudWatch Lambda logs.
<b>Java version</b>	Indicates the Java JRE version expected in the S3 repository file name server-jre-8u <b>&lt;JavaVersion&gt;</b> -linux-x64.tar.gz. For example, server-jre-8u <b>151</b> -linux-x64.tar.gz.
<b>PingAccess Version</b>	Indicates the PingAccess version expected in the S3 repository file name


pingaccess- <b>&lt;PingAccessVersion&gt;</b> .zip. For example, pingaccess-5.0.0.zip.
---

## Configure Security Groups

To access the PingAccess and PingFederate Administration consoles, as well as the PingFederate runtime, AWS Security Groups must be configured to allow ingress traffic on the required ports. The complete installation provides a set of Security Groups that you must configure.

You can:


- [Configure the included security groups for ingress access](#)
- [Create a new Security Group](#)

 **Important:** By default, all security groups created as part of this solution are configured to allow **all** egress traffic. Administrators may wish to review egress traffic requirements for their network and modify these settings. For more information, see [AWS documentation for Security Groups](#).

### Configure the included security groups for ingress access:

1. In AWS, navigate to **Services > Compute > EC2**.
2. In the left menu bar, under **Network & Security**, click **Security Groups**.
3. Select the Security Group you want to edit. See the table below for the list of groups.
4. In the bottom pane that appears, select the **Inbound** tab.
5. Click **Edit**.
6. In the **Edit inbound rules** window, click **Add Rule**.
  - a. Add a **Custom TCP** rule.
  - b. In the **Port Range** field, enter the port. See the table below for required ports.
  - c. Enter the CIDR or IP for the machine you want to use for access. For quick access, you can select the **Source of My IP**.
7. Click **Save**.

<b>PingAccess Admin API ingress</b>	Port 9000
<b>PingFederate</b>	Ports 9999 and 9031
<b>SSH Ingress (Type SSH)</b>	Port 22

 **Note:** Configuration of PingFederate Security Groups is only required if you are choosing to deploy with a demonstration PingFederate instance by selecting "True" for the ProvisionPingFederate parameter in the PingAccess for AWS CloudFormation template.

### Create a new Security Group:

1. In AWS, navigate to **Services > Compute > EC2**.
2. Create a Security Group and assign it to the VPC that will include the PingAccess and PingFederate installations.
3. Add 3 **Custom TCP** rules and one **SSH** rule:
  - Port Range: **9000**, Source: **My IP**
  - Port Range: **9999**, Source: **My IP**
  - Port Range: **9031**, Source: **My IP**
  - Post Range: **22**, Source **My IP**
4. Click **Create**.

## Access the PingAccess administration console

After the automated deployment is complete, you can access the PingAccess administrative console. The PingAccess administrative console will allow you to configure a token provider, create sites and applications, configure policies, and perform other related tasks.



**Important:** The PingAccess for AWS solution protects your admin node availability by ensuring a new instance of the admin node is created automatically in the event of a failure. This automated process deploys an instance of the PingAccess admin node based on the settings specified in `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`.

For this reason, it is **critical** to your deployment that you maintain this file. Any time changes are made to the PingAccess configuration, you **must** export the new configuration and replace the existing file at `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`. Failure to synchronize this file could lead to unexpected results if your admin node should fail for any reason.

### Access the PingAccess Administration console

To access the PingAccess Administration console in your browser, use a combination of the Admin public host name and the PingAccess Administration port.

For example: `https://pa-admin.mypublichostname.com:9000`

For the demonstration environment, use the following credentials:

```
Username: joe
Password: 2Federate
```

If you are using an existing PingFederate instance, login with the credentials you created for use with the Simple Password Credential Validator in [PingAccess for AWS: PingFederate environment requirements](#).

### Access the PingFederate administration console



**Note:** The following instructions only apply if you have chosen to deploy the PingAccess for AWS solution with a basic PingFederate instance for demonstration purposes.

After the automated deployment is complete, you can access the PingFederate administrative console. The PingFederate administrative console will allow you to configure clients and perform other related tasks.

#### Access the PingFederate Administration console

To access the PingFederate Administration Console in your browser, use a combination of the PingFederate public host name, the PingFederate Administration port, and the path to the PingFederate application.

For example: `https://pf-admin.mypublichostname.com:9999/pingfederate/app`

For this demonstration, use the default PingFederate credentials:

```
Username: Administrator
Password: 2Federate
```

## Configure SSH connectivity

---

Configure SSH connectivity to the PingAccess Administration and Engine nodes so you can perform tasks such as viewing logs. The PingAccess Administration and Engine nodes are not exposed directly to the internet, but are configured to be placed behind a Bastion host through which you can access these resources. You first connect to the Bastion host, and then establish a connection to the desired resource.

The instructions provided here are intended to detail just one of the methods you can use to establish this connectivity. You may choose to use another method.

### Prerequisites

This procedure requires that you have the following:

- The PingAccess private key
- The Public DNS hostname used by the Bastion host. Obtain this in EC2 by selecting the SSH Bastion instance and viewing the Description tab.

- The Private DNS hostname used by the PingAccess Admin Console and/or Engines. Obtain this in EC2 by selecting the desired instance and viewing the Description tab.

### Configure SSH connectivity

1. Add the PingAccess private key to the SSH authentication agent. Run this command from the directory containing the private key. For example:

```
ssh-add -K ~/.ssh/mypingaccesskeypair.pem
```

2. Add a host entry to your SSH configuration for the Bastion host. You will provide a name for the entry, the public DNS host name, and the user name used to access the host. For example, use the GNU nano editor to edit the host file with the following command:

```
nano ~/.ssh/config
```

Add an entry with the following information:

```
Host aws-bastion
    Hostname mypublicbastiondns.compute.amazonaws.com
    User ec2-user
```

Save the file.

3. Test the connection to the Bastion host:

```
ssh aws-bastion
```

4. Add one or more SSH host entries for the Administration or Engine nodes, as required. Use the ProxyCommand function to route the connection through the Bastion host. For example:

```
Host aws-console
    Hostname myinternaladmindns.compute.internal
    ProxyCommand ssh -q aws-bastion nc %h 22
    User ec2-user
```

Save the file.

### Establish a connection

Establish a connection to a configured PingAccess Administration or Engine node using the name you configured in the SSH host file. For example:

```
ssh aws-console
```

## View PingAccess log files

---

To assist with troubleshooting, you can view PingAccess Admin and Engine log files. You can view logs related to the admin or engine nodes, and you can view [AWS CloudWatch](#) logs. To view node log files, you must first [establish an SSH connection](#) to the required node.

On an Admin or Engine node, the PingAccess installation can be found at `/usr/local/pingaccess`. Log files are contained in the `log` directory.

### View log files

After you have established a connection to the node, navigate to the proper directory and open the file using an editor such as GNU nano. For example:


```
cd /usr/local/pingaccess/log
```

```
nano pingaccess.log
```

## Maintaining the S3 bucket

---

To maintain the S3 bucket for updated files, simply delete the existing files and upload any new files using the steps found in [Populate the S3 bucket](#) on page 237.


 **Note:** If you are planning to redeploy with an existing configuration, see [Redeploying with an existing configuration](#) on page 246.

## Delete the PingAccess VPC

---

If removal of the PingAccess VPC is required, you can perform this task by deleting the top level stack. To do this:

1. In AWS, navigate to **Services** > **Management Tools** > **CloudFormation**.
2. In the CloudFormation console, select the top level template.
3. Click **Actions** > **Delete Stack**.
4. When prompted, confirm the operation.

 **Note:** If you are planning to redeploy with an existing configuration, see [Redeploying with an existing configuration](#) on page 246.

## Redeploying with an existing configuration

---

You can redeploy the PingAccess for AWS solution using an existing configuration. You can specify to use an existing configuration by re-using the **Export name prefix**. Customers who wish to redeploy while maintaining an existing configuration should consider the following notes:

- You must first delete the PingAccess VPC
- When redeploying, you must use the same **Export name prefix**.
- Before you begin, you must delete the engine definitions present at `/automation-artifacts/<exportnameprefix>/pingaccess/engines`.
- If you intend to change certificate related parameters, you must delete the `/automation-artifacts/<exportnameprefix>/ca` folder.

## Troubleshooting VPC creation

---

For a variety of possible reasons, you can encounter errors or other issues during the creation of the PingAccess VPC. This document is intended to provide guidance on troubleshooting some of these issues.

To troubleshoot VPC creation, you should [disable Rollback on Failure](#) so that relevant logs and other helpful information remain available.

- [General](#)
- [Viewing logs on EC2 instances](#)
- [View CloudWatch logs](#)
- [View Lambda functions](#)

### General

Some of the more common errors come from the planning and/or preparation process. For your chosen deployment type, ensure that you have provided and specified:

- Valid license files for PingAccess and, if applicable, PingFederate

- Valid installation packages for PingAccess and, if applicable, PingFederate
- Valid Server JRE install package
- A set of *AMIs* available to your AWS region that support Signature Version 2
- One or more *certificates* available to your AWS region
- If you are redeploying and reusing the same Export Name Prefix, ensure you have followed the advice in [Redeploying with an existing configuration](#) on page 246.

### Viewing logs on EC2 instances

To view CloudFormation logs on your EC2 instances, [SSH to the instance](#) and view logs located at `/var/log`. CloudFormation logs begin with the `cfn-` prefix.

For instances that contain a PingAccess or PingFederate installation, you can view their respective log files at `/usr/local/pingaccess/log` and `/usr/local/pingfederate/log`.

### View CloudWatch logs

CloudWatch logs sometimes contain helpful information in the event of a failure. View CloudWatch logs by navigating to **Services > Management Tools > CloudWatch** and clicking **Logs**.

You can filter logs for easier viewing. To find applicable CloudWatch logs, filter using the string `/pingidentity/{ExportNamePrefix}` where `{ExportNamePrefix}` is replaced by the **Export name prefix** you specified during [VPC creation](#).

Click on a log group to view available log streams and their contents. To learn more about CloudWatch logs, see [AWS documentation for CloudWatch logs](#).

### View Lambda functions

To help narrow down causes for an issue, it may be helpful to view logged details for the lambda that is producing the error. To do this, navigate to **Services > Compute > Lambda** and filter by the VPC name. Locate and click the lambda function you are investigating. On the **Monitoring** tab, review the **Invocation errors** widget and click **Jump to Logs**.



# PingAccess for AWS: PingFederate environment requirements

---

## Introduction

---

In addition to deploying a demonstration environment that includes a basic PingFederate instance, you can configure the PingAccess for AWS solution to deploy with support for an existing PingFederate environment. You may choose this installation method for test or production deployments with a PingFederate environment that is external to your AWS VPC.

This document is a guide for administrators that describes the components the automation process is expecting to be configured in your PingFederate environment, how to configure those components, and how to include information about your environment to the automation.

While variations to these PingFederate requirements do exist, and while it is possible to deviate from these instructions in some cases, this document is intended to describe a minimum configuration and, as such, these instructions would be outside the scope of this document at this time.

At a minimum, your PingFederate environment requires the following components:

1. *Install the AWS Password Credential Validator*
2. *Configure the AWS PCV instance*
3. *Configure a Simple Password Credential Validator Instance*
4. *Create an IdP adapter*
5. *Add a Persistent Grant Extended Attribute*
6. *Configure default scope values*
7. *Configure an Access Token Manager instance*
8. *Configure an AWS PCV Resource Owner Credentials Mapping*
9. *Configure an IdP adapter mapping*
10. *Configure an IdP Adapter Access Token mapping*
11. *Configure an AWS PCV Access Token mapping*
12. *Create an OpenID Connect policy*
13. *Configure the PingAccess Admin SSO OAuth client*
14. *Configure the Resource Owner API Client OAuth client*
15. *Create and (optionally) export a PingFederate certificate*

## Create an IAM User in AWS

---


This document describes the steps required to create an IAM User that will be used by the AWS Password Credential Validator.



**Note:** If your PingFederate instance resides in Amazon EC2, do not use these steps. The preferred method is instead to create an IAM Role and assign the proper permissions:

1. Navigate to **Services > IAM**.
2. Click **Roles**.
3. Click **Create role**.
4. Select **AWS service > EC2** and specify the **EC2** use case.
5. Click **Next: Permissions**.
6. Filter for **IAMReadOnlyAccess** and select this item.
7. Click **Next: Review**.

8. Provide a **Role name** and optional **Role description**.
  9. Click **Create role**.
  10. Assign this role to your PingFederate EC2 instance by right clicking the EC2 instance and choosing **Instance Settings > Attach/Replace IAM Role**.
1. In AWS, navigate to **Services > Security, Identity & Compliance > IAM**.
  2. Click **Users**.
  3. Click **Add User**.
  4. Specify a **User name**. For example: `awspcv_api`
  5. For **Access type**, select **Programmatic access**.
  6. Click **Next**.
  7. Select **Attach existing policies directly**.
  8. Filter the list for **IAMReadOnlyAccess** and select this item.
  9. Click **Next**.
  10. Click **Create User**.
  11. Copy the **Access key ID** for use with the AWS PCV instance that you will create in PingFederate.
  12. In the **Secret access key** field, click **Show**.
  13. Copy the **Secret access key** for use with the AWS PCV you will create.
 

 **Important:** You must copy the Secret access key at this stage. The key will be masked permanently when you navigate away from this page.
  14. Click **Close**.

## PingFederate configuration requirements

---

The following sections describe the PingFederate configuration elements expected by the automation. At a minimum, your PingFederate environment should include these components.

### Install the AWS Password Credential Validator


The following steps describe how to install the AWS password credential validator (PCV).

1. Obtain the PCV file, `aws-password-credential-validator-<version>.jar`.
2. Copy the file to your PingFederate installation at `%PingFedHome%/pingfederate/server/default/deploy`.
3. Restart PingFederate.

### Configure the AWS PCV instance

The following steps describe how to configure the AWS PCV instance.

1. In PingFederate, navigate to **Server Configuration > Password Credential Validators**.
2. Click **Create New Instance**.
3. Specify the **Instance Name**: AWS PCV
4. Specify the **Instance ID**: `AwsPcv`
5. Select the **Instance Type**: AWS Username Password Credential Validator
6. Click **Next**.
7. Enter the **Access Key** and **Secret Key** for the *IAM user* you created.

 **Note:** If your PingFederate instance is in EC2, and you have *created an IAM Role*, leave these fields blank.

 **Note:** If your PingFederate instance is located in the same VPC, you can leave these fields blank.

8. Click **Next**.
9. Click **Done**.

10. Click **Save**.

### Configure a Simple Password Credential Validator Instance

The following steps describe how to configure the Simple PCV instance. This PCV will be assigned to an IdP adapter instance.



**Note:** You can use an existing PCV/Adapter configuration for Admin SSO if it meets these configuration requirements.

1. In PingFederate, navigate to **Server Configuration > Password Credential Validators**.
2. Click **Create New Instance**.
3. Specify the **Instance Name**: SimplePcv
4. Specify the **Instance ID**: SimplePcv
5. Select the **Instance Type**: Simple Username Password Credential Validator
6. Click **Next**.
7. Click **Add a new row to 'Users'**.
8. Specify a **Username** and **Password** to use for PingAccess Admin SSO, confirm the password, and click **Update**.  
For example: joe/2Federate.
9. Click **Next**.
10. Click **Done**.
11. Click **Save**.

### Create an IdP adapter instance

The following steps describe how to create an IdP adapter instance.

1. Navigate to **IdP Configuration > Adapters**.
2. Click **Create New Instance**.
3. Specify the **Instance Name**: HtmlFormIdpAdapter
4. Specify the **Instance ID**: HtmlFormIdpAdapter
5. Select the **Instance Type**: HTML Form IdP Adapter
6. Click **Next**.
7. Click **Add a new row to 'Credential Validators'**.
8. Select the **SimplePcv** PCV you created and click **Update**.
9. Click **Next** (x2).
10. On the **Adapter Attributes** screen, select the **Pseudonym** checkbox.
11. Click **Next** (x2).
12. Click **Done**.
13. Click **Save**.

### Add a Persistent Grant Extended Attribute

The following steps describe how to add a Persistent Grant Extended Attribute.

1. Navigate to **OAuth Settings > Authorization Server Settings**.
2. In the **Persistent Grant Extended Attributes** section, type **ACTIONS** and click **Add**.
3. Scroll to the bottom and click **Save**.

### Configure default scope values

The following steps describe how to add values to the default scope.

1. Navigate to **OAuth Settings > Scope Management**.
2. Specify a **Default Scope Description**: Default Permissions
3. Enter the following **Scope Value** and **Scope Description** items and click **Add** with each:

Scope Value	Scope Description
address	Address access
email	Email access
offline_access	Offline access
openid	OpenID Connect login
phone	Phone Number access
profile	Profile access

4. Click **Save**.

### Configure an Access Token Manager instance

The following steps describe how to create an Access Token Manager instance.

1. Navigate to **OAuth Settings > Access Token Management**.
2. Click **Create New Instance**.
3. Enter the **Instance Name**: Internally Managed Reference Tokens
4. Enter the **Instance ID**: default
5. Select the **Instance Type**: Internally Managed Reference Tokens
6. Click **Next** (x3).
7. On the **Access Token Attribute Contract** screen, extend the contract by adding attributes **Actions** and **Username**, clicking **Add** after each.
8. Click **Next** (x3).
9. Click **Done**.
10. Click **Save**.

### Configure an AWS PCV Resource Owner Credentials Mapping

The following steps describe how to create an AWS PCV Resource Owner Credentials Mapping.

1. Navigate to **OAuth Settings > Resource Owner Credentials Mapping**.
2. Select the AWS PCV from the **Source Password Validator Instance** list and click **Add Mapping**.
3. Click **Next**.
4. On the Contract Fulfillment screen, select the Source and Value for each Contract item as follows:

Contract	Source	Value
ACTIONS	Password Credential Validator	actions
USER_KEY	Password Credential Validator	rolename

5. Click **Next** (x2).
6. Click **Done**.
7. Click **Save**.

### Configure an IdP adapter mapping

The following steps describe how to configure an IdP adapter mapping.

1. Navigate to **OAuth Settings > IdP Adapter Mapping**.
2. In the **Source Adapter Instance** list, select the **HtmlFormIdpAdapter** you created.
3. Click **Add Mapping**.
4. Click **Next**.
5. On the **Contract Fulfillment** screen, configure entries as follows:

Contract	Source	Value
ACTIONS	Text	ssso
USER_KEY	Adapter	username
USER_NAME	Adapter	username

6. Click **Next** (x2).
7. Click **Save**.

### Configure an IdP Adapter Access Token mapping

The following steps describe how to configure an IdP Adapter Access Token mapping.

1. Navigate to **OAuth Settings > Access Token Mapping**.
2. Select the **IdP Adapter: HtmlFormAdapter** from the **Context** list, **Internally Managed Reference Tokens** from the **Access Token Manager** list, and click **Add Mapping**.
3. On the **Contract Fulfillment** screen, configure entries as follows:

Contract	Source	Value
Actions	Persistent Grant	ACTIONS
Username	Persistent Grant	USER_KEY

4. Click **Next** (x2).
5. Click **Save**.

### Configure an AWS PCV Access Token mapping

The following steps describe how to configure an AWS PCV Access Token mapping.

1. Navigate to **OAuth Settings > Access Token Mapping**.
2. Select the **Validator: AWS PCV** from the **Context** list, **Internally Managed Reference Tokens** from the **Access Token Manager** list, and click **Add Mapping**.
3. On the **Contract Fulfillment** screen, configure entries as follows:

Contract	Source	Value
Actions	Persistent Grant	ACTIONS
Username	Persistent Grant	USER_KEY

4. Click **Next** (x2).
5. Click **Save**.

### Create an OpenID Connect policy

The following steps describe how to create an OpenID Connect policy.

1. Navigate to **OAuth Settings > OpenID Connect Policy Management**.
2. Click **Add Policy**.
3. Enter the **Policy ID**: PA-WAM
4. Enter the **Policy Name**: PA-WAM
5. Select the **Access Token Manager**: Internally Managed Reference Tokens
6. Click **Next**.
7. Delete all items beneath **Extend the Contract**.
8. Click **Next** (x2).
9. For the **Attribute Contract** sub, select a **Source** of **Access Token**, and a **Value** of **Username**.
10. Click **Next** (x2).

11. Click **Done**.
12. Click **Set as Default**.
13. Click **Save**.

### Configure the PingAccess Admin SSO OAuth client

The following steps describe how to create the PingAccess Admin SSO OAuth client that is used by the automation.

1. Navigate to **OAuth Settings > Clients** and click **Create New**.
2. Enter the **Client ID**: pa\_sso
3. Enter the **Name**: PingAccess Admin SSO
4. For **Client Authentication**, select **None**.
5. Enter the following **Redirect URIs** and click **Add** after each:
  - https://\*:9000
  - https://\*:9000/\*
6. Select the following **Allowed Grant Types**:
  - Authorization Code
  - Refresh Token
  - Implicit
7. Click **Save**.

### Configure the Resource Owner API Client OAuth client

The following steps describe how to create the Resource Owner API Client OAuth client that is used by the automation.

1. Navigate to **OAuth Settings > Clients** and click **Create New**.
2. Enter the **Client ID**: api\_ro
3. Enter the **Name**: Resource Owner API Client
4. For **Client Authentication**, select **Client Secret** and specify a Secret of 2Access.
5. Select the following **Allowed Grant Types**:
  - Resource Owner Password Credentials
  - Access Token Validation (Client is a Resource Server)
6. Click **Save**.

### Create and (optionally) export a PingFederate certificate

Your PingFederate installation must use a valid certificate that contains the PingFederate hostname as either the Common Name or Subject Alternative Name. You can import a valid certificate into PingFederate or you can create a certificate using PingFederate.

The following steps describe how to use PingFederate to create and export a certificate for use with the automation.

1. Navigate to **Server Configuration > SSL Server Certificates**.
2. Click **Create New**.
3. Enter a **Common Name** that matches the URL for the PingFederate server. For example, if your PingFederate server is hosted at `mypingfed.myserver.com:9031`, the Common Name will be `mypingfed.myserver.com`.
4. Enter appropriate values for **Organization** and **Country**.
5. Click **Next**.
6. Choose to make the certificate active for both the runtime server and the admin console.
7. Click **Done**.
8. On the **Certificate Management** screen, locate the certificate you created and click **Export**.
9. Select **Certificate Only** and click **Next**.

10. On the **Export & Summary** screen, click **Export** to download the certificate.
11. Click **Done**.
12. Locate the certificate file and open in a text editor.
13. Copy and save the certificate data between the first and last lines. Do not copy the first and last lines that identify the certificate file. You will add this data to the *api.json file* used by the automation.

## Modify api.json to include details about your environment

---

Users who wish to deploy the PingAccess for AWS solution with an existing PingFederate instance must edit the `api.json` file that is included with the automation package and supply this file to the automation. You will provide details about your PingFederate environment that will be used to configure the PingAccess token provider and Admin Authentication components.

The following instructions assume that you have followed and meet the *PingFederate configuration requirements* on page 249.

**!** **Important:** It is important to note that if your AWS deployment fails for any reason while you are using a modified `api.json` file, you will need to replace the file in the S3 repository with the modified file. This is due to changes that are made to the file by the automation, such as the obfuscation of password data.

The `api.json` file is located in the automation package at `<automation package>/s3-bucket-root/pingaccess/conf`.

If you meet the PingFederate configuration requirements, the following changes will need to be made to the `api.json` file:

- `"pingFederate": "host":` - Enter the hostname for your PingFederate server. Do not include ports or prefixes.
- `"pingFederate": "trustedCertificateGroupName":` - Enter "Trust Any".
- `"pingFederateAdmin": "host":` - Enter the hostname for your PingFederate server. Do not include ports or prefixes.
- `"pingFederateAdmin": "trustedCertificateGroupName":` - Enter "Trust Any" for testing or debugging purposes, otherwise *Add a PingFederate Certificate object to the automation*. Using a PingFederate certificate object overrides this property.

If your PingFederate server is running on non-default ports, enter the appropriate ports in this file.

### (Optional) Add a PingFederate Certificate object to the automation

You can use an *existing PingFederate certificate* with the automation by adding the following top level object to `api.json`:

```
"pingFederateCertificate": {
  "alias": "PingFederate",
  "fileData": "<certificate_data>"
}
```

Replace `<certificate_data>` with the data you copied and saved in *Create and (optionally) export a PingFederate certificate* on page 253.

### Add the api.json file to the automation

After you have made the appropriate changes to the `api.json` file, upload the file to your S3 bucket at `<S3 bucket>/pingaccess/conf`.

# PingAccess for AWS: Configure an application


---

## Introduction

---

You can configure a protected application in the PingAccess for AWS environment. This document will provide instructions for configuring an application. These instructions are applicable for deployments that include a basic PingFederate instance, as well as deployments that make use of an existing PingFederate environment.

These instructions assume you have successfully deployed your environment and are able to access the PingAccess and PingFederate administration consoles.

 **Important:** The PingAccess for AWS solution protects your admin node availability by ensuring a new instance of the admin node is created automatically in the event of a failure. This automated process deploys an instance of the PingAccess admin node based on the settings specified in `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`.

For this reason, it is **critical** to your deployment that you maintain this file. Any time changes are made to the PingAccess configuration, you **must** export the new configuration and replace the existing file at `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`. Failure to synchronize this file could lead to unexpected results if your admin node should fail for any reason.

## Protect an application

---

This document provides the steps required to protect an application using PingAccess and PingFederate in the AWS environment.

To protect an application, complete the following steps:

1. [Configure PingFederate](#) on page 256
2. [Configure a PingAccess web application](#) on page 256
3. [Configure a PingAccess API application](#) on page 258
4. [Test the web application](#) on page 259
5. [Test the API application](#) on page 259

## Install and access the unprotected PingAccess QuickStart application

After the automated deployment is complete, you can install the PingAccess QuickStart application. The PingAccess QuickStart application can be used to test a protected PingAccess application.

### Install the PingAccess QuickStart application

To install the PingAccess QuickStart application:

1. Download the PingAccess QuickStart application.
2. Unzip the file.
3. Copy `\pingaccess-quickstart-<version>\pf-dist\PingAccessQuickStart.war` to `<PingFedHome>\pingfederate\server\default\deploy`.
4. Restart PingFederate.

### Access the unprotected PingAccess QuickStart application

To access the unprotected PingAccess QuickStart application in your browser, use a combination of the PingFederate public host name, the PingFederate runtime port, and the path to the QuickStart application.

For example: `https://pf-admin.mypublichostname.com:9031/PingAccessQuickStart`



## Configure PingFederate

If you have deployed a basic PingFederate instance as part of the automation process, most of the configuration is complete. Likewise, if you have deployed with an existing PingFederate instance, and meet the PingFederate environment requirements, most of the required configuration is complete.

This document describes the steps required to create a client that will be used for the PingAccess Web Session you will assign to a web based application in PingAccess.

### To create the web session client in PingFederate:

1. Log in to the PingFederate administration console.
2. Navigate to **Main > OAuth Settings > Clients** and click **Manage All**.
3. Click **Add Client**.
4. Specify a **Client ID**. For example:

```
pa_wam
```

5. Specify a **Name**. For example:

```
PingAccessWebAccessManagement
```

6. Select **Client Secret**.
7. Generate a secret by clicking **Generate Secret**. Copy this secret to a secure location so that you can use it in PingAccess configuration.
8. In the **Redirection URIs** field, add the OIDC callback redirect to the PingAccess server, for example:


```
https://mypingaccessserver.my-aws-domain.com/pa/oidc/cb
```

9. Click **Add**.
10. Select the **Bypass Authorization Approval** check box.
11. For the **Allowed Grant Type** setting, select the **Authorization Code** check box.
12. Click **Save** (x2).

## Configure a PingAccess web application

This document provides the steps required to configure the demo web application in PingAccess. In performing the steps required, you will:

1. [Create a web session](#) on page 256
2. [Create a virtual host](#) on page 257
3. [Create a site](#) on page 257
4. [Create an application](#) on page 257
5. [Configure resources](#)

 **Important:** The PingAccess for AWS solution protects your admin node availability by ensuring a new instance of the admin node is created automatically in the event of a failure. This automated process deploys an instance of the PingAccess admin node based on the settings specified in `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`.

For this reason, it is **critical** to your deployment that you maintain this file. Any time changes are made to the PingAccess configuration, you **must** export the new configuration and replace the existing file at `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`. Failure to synchronize this file could lead to unexpected results if your admin node should fail for any reason.

### Create a web session

Create a web session to control access. For more information, see [Web Sessions](#).

1. Navigate to **Settings > Access > Web Sessions**

2. On the Web Session page, click **Add Web Session**.
3. Enter a unique **Name** for the web session, up to 64 characters, including special characters and spaces. For example:

```
PF_WAM
```

4. Select `Encrypted JWT` for **Cookie Type**.
5. Specify the **Audience** that the PA Token is applicable to, represented as a short, unique identifier between 1 and 32 characters. For example:

```
global
```

6. Specify the **OpenID Connect Login Type** `CODE`.
7. Specify the **Client ID**. For example, the Client ID you created is:

```
pa_wam
```

8. Specify the **Client Secret** you copied in [Configure PingFederate](#) on page 256.
9. Click **Save**.

### Create a virtual host

Create a virtual host that PingAccess will respond to. For more information, see [Virtual Hosts](#).

1. Navigate to **Settings > Access > Virtual Hosts**.
2. Click **Add Virtual Host**.
3. Enter the **Host** name for the Virtual Host or specify a wildcard. For example: `app.my-aws-pingaccess.com` or `*`.
4. Enter the **Port** number for the Virtual Host. In this environment, AWS is expecting a secure connection, so you must specify the secure port. For example:

```
443
```

5. Click **Save**.

### Create a site

Create a site to define the location of the application that PingAccess is protecting. For more information, see [Sites](#).

1. Navigate to **Main > Sites > Sites**.
2. Click **Add Site**.
3. Specify a **Name**. For example:

```
QuickStart
```

4. Specify the **Target**. The format for this is `hostname:port`. For example:

```
pf.my-pf-instance.com:9031
```

5. Select **Secure** and choose the `Trust Any` **Trusted Certificate Group**.
6. Click **Save**.



**Note:** If the target site cannot be contacted, the site is saved and a warning is displayed indicating the reason the site was not reachable.

### Create an application

Create an application to define the resource that PingAccess is protecting. For more information, see [Applications](#).

1. Navigate to **Main > Applications**.

2. Click **Add Application**.
3. Provide a unique name for the application. For example:

```
QuickStart
```

4. Specify the context at which the application is accessed at the site. For example:

```
/PingAccessQuickStart
```

5. Specify the virtual host for the application. For example:

```
*:443
```

6. Specify the application type `Web` and select the **Web Session** for the application. For example:

```
PF_WAM
```

7. Specify the application destination type `Site` and select the **Site** requests are sent to when access is granted. For example:

```
QuickStart
```

8. Select the **Require HTTPS** option.
9. Select the **Enabled** checkbox.
10. Click **Save**.

### Configure resources



**Note:** This step is only required for the demonstration environment. Your resource access requirements may differ.

Configure application resources to allow anonymous access to the root resource and authenticated access to the `/headers` resource.

1. Navigate to **Main > Applications**.
2. Expand the QuickStart application and click the edit button.
3. Select the **Resources** tab.
4. Expand the **Root Resource** and click the edit button.
5. Select the **Anonymous** checkbox and click **Save**.
6. Click **Add Resource**.
7. Enter a **Name** for the resource. For example: `headers`.
8. Enter the **Path Prefix** of `/headers`.
9. Click **Save**.

## Configure a PingAccess API application

This document provides the steps required to configure the demo API application in PingAccess. Prior to completing these steps, ensure you have followed the steps in [Configure a PingAccess web application](#) on page 256 to create a **Virtual Host** and a **Site**.



**Important:** The PingAccess for AWS solution protects your admin node availability by ensuring a new instance of the admin node is created automatically in the event of a failure. This automated process deploys an instance of the PingAccess admin node based on the settings specified in `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`.

For this reason, it is **critical** to your deployment that you maintain this file. Any time changes are made to the PingAccess configuration, you **must** export the new configuration and replace the existing file at `automation-artifacts/<export_prefix>/pingaccess/conf/pingaccess.json`. Failure to synchronize this file could lead to unexpected results if your admin node should fail for any reason.

## Create an application

Create an application to define the resource that PingAccess is protecting. For more information, see [Applications](#).

1. Navigate to **Main > Applications**.
2. Click **Add Application**.
3. Provide a unique name for the application. For example:

```
QuickStart
```

4. Specify the context at which the application is accessed at the site. For example:

```
/PingAccessQuickStart/api
```

5. Specify the virtual host for the application. For example:

```
*:443
```


6. Specify the application type **API** and ensure the **Protected** option is selected.
7. Specify the application destination type **Site** and select the **Site** requests are sent to when access is granted. For example:

```
QuickStart
```

8. Select the **Require HTTPS** option.
9. Select the **Enabled** checkbox.
10. Click **Save**.

## Test the web application

This document describes the steps to test the demo web application.

-  **Tip:** For the most visible experience with this demo, perform these steps in a new private or incognito session to avoid the reuse of an existing session.

### To test the web application:

1. Open your browser and navigate to the application using a combination of the **Applications public host name** you created in [Create the PingAccess VPC](#) on page 238 and the application Context Root. For example:


```
https://app.my-aws-domain.com/PingAccessQuickStart
```

2. On the Web Access Management application, click **Try it Now**.
3. You will be prompted to authenticate. Enter the **Username** and **Password** combination included with the QuickStart demo application: `joe/2Federate`.

The **headers** page is displayed.

## Test the API application

This document describes the steps to test the demo API application.

-  **Tip:** For the most visible experience with this demo, perform these steps in a new private or incognito session to avoid the reuse of an existing session.

### To test the demo API application:

1. Open your browser and navigate to the application using a combination of the **Applications public host name** you created in [Create the PingAccess VPC](#) on page 238 and the application Context Root. For example:

```
https://app.my-aws-domain.com/PingAccessQuickStart
```

2. On the API Access Management application, click **Try it Now**.

3. Click the **Get Token** button.
4. You will be prompted to authenticate. Enter the **Username** and **Password** combination included with the QuickStart demo application: joe/2Federate.

The token is obtained and pre-populated along with the URL context.

5. Click **Send API Request**.

A **Response Code** of **200** indicates a successful test. **Response Headers** are displayed.

# PingAccess release notes

---

## 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 protects applications and APIs, enabling access control and identity-based auditing on incoming requests. Featuring a lightweight, highly scalable architecture, PingAccess complements PingFederate with centralized session management and URL-level authorization.

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

### PingAccess 5.0.1 - December 2017

PingAccess 5.0.1 is a cumulative maintenance release for PingAccess 5.0, which introduced several new features, including Ping Identity Cloud Automation Repo, Zero downtime upgrades, a new application type (Web +API), along with several other enhancements. For more information, see the [release notes for PingAccess 5.0](#).

 **Important:** For customers upgrading to the 5.0 release from an earlier version, please note that there are potentially breaking changes to plugins and Groovy scripts. Plugins built with JDK 7 may not function as expected and should be reviewed for migration to JDK 8. For more information, see the [PingAccess Addon SDK for Java Migration Guide](#).

Likewise, the Groovy API is enhanced for the addition of some objects. Groovy scripts should be reviewed for necessary changes. For more information on these changes, see the [Groovy Development Reference Guide](#).

Previous functionality that allowed Groovy scripts to leverage an undocumented capability to use the Addon SDK APIs has been removed. Methods and types added to the Groovy API align with the previous version of the Addon SDK. As a result, if existing Groovy scripts leverage Addon SDK APIs that are not available in the new Groovy API, they will need to be ported to only use the methods and types available in the new Groovy API before upgrading to PingAccess 5.0.

### Resolved issues

Ticket ID	Description
PA-8706	Fixed a performance issue that is encountered when you create and assign a large number of virtual hosts to an application.
PA-7936	Fixed an issue in the Upgrade Utility.

### Known issues and limitations

#### Known issues

- Use of TLSv1.0 has been maintained for use by legacy versions of Internet Explorer. Since continued use of TLSv1.0 is not recommended for security reasons, users should upgrade to the latest version of Internet Explorer to make use of the more secure TLSv1.1 or TLSv1.2.
- Reverse DNS lookup on localhost fails when making back-channel calls to PingFederate.
- Engines and admin replicas do not connect to admin console if a combination of IP addresses and DNS names are used.
- The token processor can't connect to a JWKS endpoint via SSL when an IP is used rather than a hostname. To workaround this issue, add the hostname as the subject alt name on the key pair.
- In some cases, a session may not be invalidated when the WebSessionManagement issuer is changed.

- When using Internet Explorer 11, you may not be able to view the full application or resource policy because the scroll bar is missing.
- The Linux installer does not accept a configuration zip file with a space in the file name. To work around this issue, remove the space from the file name.

### Known limitations

- The PingAccess for AWS solution cannot yet support agent deployments due to a limitation in the Amazon Elastic Load Balancer implementation.
- Internet Explorer and Firefox do not correctly support the HTML5 time tag. When using the Time Range rule, enter time in 24-hour format.
- When installing PingAccess as a Windows service using Windows PowerShell and Java 8, the error message "Could not find or load main class" can be safely ignored.
- POST Preservation is not supported with Safari Private Browsing.
- When using IE 11 to access the PingAccess admin console remotely, a fully qualified domain name or IP address must be specified. For example, `https://console.site.com:9000` and `https://172.17.8.252:9000` will work, while specifying only the host name, `https://console:9000`, will not.
- Incorrect handling for IPv6 literals in Host header. Note that IPv6 is not currently supported.

## Previous releases

To access previous release notes, select the applicable version from the table of contents.

### PingAccess 5.0 - December 2017

#### Enhancements

##### PingIdentity Cloud Automation Repo

An automated cloud deployment of PingAccess for Amazon Web Services. This solution, both for customers with and without existing AWS resources, allows administrators to easily configure a functioning environment using PingAccess and an optional basic PingFederate deployment. The templates included with the package will guide you through the setup of all necessary components, including the S3 bucket, EC2 instances, routing and security parameters, and naming conventions.

For more information on deploying this solution, see the [PingAccess for AWS: Solution setup guide](#).



**Note:** The PingAccess for AWS solution does not yet support agent deployments. For more information, see [Known limitations](#).

#### Zero downtime upgrades

Upgrade a cluster to PingAccess 5.0 with no downtime. This updated procedure takes advantage of new functionality that ensures availability during the entire upgrade process without impacting existing user sessions.

To learn about the updated Zero Downtime Upgrade process, see [PingAccess: Zero Downtime Upgrade](#).

#### Web + API applications

PingAccess now supports the concept of a heterogeneous application that allows configuration of both Web and API settings. The PingAccess runtime inspects the inbound request for the presence of a web session cookie or an OAuth token and applies the access control policy defined for the type of authentication token in the request. Administrators can specify a default fallback type for application requests that do not include a web session cookie or OAuth token.

#### Security enhancements for exported configuration data

By default, all sensitive configuration data exported from PingAccess is now encrypted with the system master key (the "host key"), helping to ensure confidentiality of data at rest. Additionally, when a `MasterKeyEncryptor` plugin is configured, the host key will be encrypted and included in the exported data

in order to support export/import of configuration data across clusters with different host keys. Administrators still have the less secure option of including a generated encryption key in the exported data. This feature can be configured using the `admin.export.encryption.mode` property in `run.properties`.

Configurable values for this property are:

- **MASTER\_KEY** - The host key is encrypted and included in exported data. This option requires that you have a master key encryptor configured to encrypt and decrypt the necessary data. Customers using this option must supply and configure the master key encryptor.
- **PORTABLE\_INSECURE** - The encrypted host key is included in exported data. In this case, the host key should be manually removed from data and stored safely to prevent unauthorized usage of the host key.

### Java Addon SDK modifications

Plugins built against the Java Addon SDK for PingAccess 5.0 are now required to be built with JDK 8, as the SDK for PingAccess 5.0 uses Java 8 features. Previously, plugins could be built with either JDK 7 or JDK 8.

- ⚠ **Important:** There are potentially breaking changes with this release. Plugins built with JDK 7 may not function as expected and should be reviewed for migration to JDK 8. For more information, see the [PingAccess Addon SDK for Java Migration Guide](#).

### Groovy API enhancements

The Groovy API is enhanced for the addition of some objects. Groovy scripts should be reviewed for necessary changes. For more information on these changes, see the [Groovy Development Reference Guide](#).

#### 📄 Note:

Previous functionality that allowed Groovy scripts to leverage an undocumented capability to use the Addon SDK APIs has been removed. Methods and types added to the Groovy API align with the previous version of the Addon SDK. As a result, if existing Groovy scripts leverage Addon SDK APIs that are not available in the new Groovy API, they will need to be ported to only use the methods and types available in the new Groovy API before upgrading to PingAccess 5.0.

### Enhanced engine and replica admin trusted certificate functionality

This release introduces enhanced engine and replica admin trusted certificate functionality that allows an administrator to choose the Trusted Certificate that gets exported for an engine or replica admin in the configuration zip file.

### Administrative API version 3

PingAccess Admin REST APIs have been updated to version 3 with several changes, including the addition and removal of some fields that may warrant a migration of existing code. For more information about these changes and information on how to migrate existing code, see [Administrative API updates](#) on page 264.

### Improved upgrade support for custom plugins

This feature adds a new `-i` command-line option to the PingAccess Upgrade Utility, as well as support by installers, that allows an administrator to specify a directory containing custom plugins compiled against the new SDK.

If you have custom plugins and wish to perform an upgrade, you will need to rebuild the plugins against the new Java SDK, then run the Upgrade Utility manually with the new `-i` command-line option to include their directory.

### Resolved issues

Ticket ID	Description
PA-6272	Fixed an issue where the installers did not validate administrator credentials before continuing.




Ticket ID	Description
PA-6975	Fixed an issue where, during active sessions, PingAccess did not enforce the presence of an access token in the PingAccess cookie when a web session configuration was modified to require the access token.
PA-7618	Fixed an issue where the Windows installer failed to install PingAccess as a service during an upgrade.
PA-7629	Fixed an issue where, during an upgrade, the Linux installer showed a status of FAILED if you chose not to replace the existing service.
PA-7931	Fixed an issue where PingAccess X-Forwarded-For IP address validation was not allowing port numbers.
PA-7958	Fixed an issue where a race condition would cause the engine to miss the auth token key roll.
PA-7971	Fixed an issue where, during an upgrade on RHEL 7 using the Linux installer, the install may show as FAILED due to an error starting the systemd PingAccess service.
PA-7975	Fixed an issue where Auth token management settings were not being properly migrated during an upgrade.
PA-8056	Fixed an issue where a race condition would cause the engine to miss configuration updates.
PA-8250	Fixed an issue where PingAccess was failing on a call to retrieve a token from the token provider.
PA-8380	Fixed a potential security issue.
PA-8382	Fixed an issue where the upgrade would fail if a third party token provider was enabled but not configured.
PA-8477	Fixed an issue where a disabled Site send token would cause the replaced Authorization header field to be removed.

### Administrative API updates

With PingAccess 5.0, the administrative API interface is updated to version 3. This update includes changes to some API endpoints.

Administrators who have written code that uses the API endpoints to automate administrative tasks will need to update their code to use the new endpoints.

 **Note:** The fundamental change in all administrative API calls will be a change in the endpoint URI from `/pa-admin-api/v2` to `/pa-admin-api/v3`. For example, if you made a call to `/pa-admin-api/v2/pingfederate/admin`, your code would need to be updated to point to `/pa-admin-api/v3/pingfederate/admin`

The following table identifies specific API changes that may need more updates than just the path to the API endpoint.

Endpoint	Change Description
<code>/applications</code>	This endpoint will now prevent creation of protected Applications if the corresponding token provider configuration is not set. For a protected Web Application, the PingFederate Runtime or the OpenID Connect provider must be configured to be able to create or update a protected Web Application via the <code>/applications</code> endpoint. For a protected API Application, the PingFederate OAuth Resource Server

Endpoint	Change Description
	<p>or OAuth Authorization Server must be configured to be able to create or update a protected API Application vis the /applications endpoint. The token provider configuration can still be cleared via the appropriate DELETE endpoints if protected Applications are configured but disabled. However, doing so will prevent future updates to existing protected Applications until the corresponding token provider configuration is set.</p> <p>The applicationType field has been updated to add support for Dynamic (Web + API) applications. Additionally, the defaultAuthType and identityMappingIds fields have been added in support of Web + API applications.</p> <p>The format of the policy field has been updated to include the application type. For example:</p> <pre data-bbox="865 730 1455 1119"> {   ...   "policy": {     "web": [       {         "type": "Rule",         "id": 20       }     ],     "API": []   }   ... } </pre>
/applications/{id}/resources	<p>This endpoint has been updated to include the defaultAuthTypeOverride field in support of Web + API application types. Use this field to override the default of an application for a resource. A value of null indicates no override.</p> <p>The format of the policy field has been updated to include the application type. For example:</p> <pre data-bbox="865 1409 1455 1797"> {   ...   "policy": {     "web": [       {         "type": "Rule",         "id": 20       }     ],     "API": []   }   ... } </pre>
/adminConfig/replicaAdmins/{id}/key/{keyId} and	The DELETE operation has been removed because the keyId field for public keys is no longer available.

Endpoint	Change Description
/engines/{id}/key/{keyId}	Public keys can be deleted by removing them from the <code>keys</code> field of a <code>ReplicaAdmin</code> or an <code>Engine</code> entity supplied to the <code>PUT /adminConfig/replicaAdmins/{id}</code> or <code>PUT /engines/{id}</code> operations, respectively.
/adminConfig/replicaAdmins and /engines	For all GET, PUT, and POST operations that return a <code>ReplicaAdmin</code> or an <code>Engine</code> entity, the <code>PublicKey</code> objects contained in the <code>keys</code> field no longer contain an <code>id</code> field.  In addition, the <code>value</code> field of <code>PublicKey</code> has been renamed to <code>jwk</code> to better represent the fact that the data is a JSON Web Key. The field type has also been changed from a <code>string</code> to a <code>JSON object</code> .
/ruleSets	The <code>RuleSet</code> type field has been renamed to <code>elementType</code> to better reflect the fact that the type represents what sort of elements are contained in the <code>RuleSet</code> .
/rulesets/types	This endpoint has been renamed to <code>/rulesets/elementTypes</code> to better reflect the fact that the types represent what sort of elements can be included in a <code>RuleSet</code> .
/config/export	The configuration data returned by a GET operation no longer contains an <code>encryptionKey</code> field (by default). Instead, the system master encryption key is used to encrypt sensitive data. If a <code>MasterKeyEncryptor</code> plugin is configured, a new <code>masterKeys</code> field will be included, which contains the encrypted master key.
/webSessions	The <code>consultServerDurationInSeconds</code> field has been removed as it is no longer used by PingAccess. The length of time an Agent caches <code>WebSession</code> policy decisions is dependent on the lifetime of the sessions as determined by <code>idleTimeoutInMinutes</code> and <code>sessionTimeoutInMinutes</code> and the intervals defined by <code>refreshUserInfoClaimsInterval</code> and <code>pfsessionStateCacheInSeconds</code> .
/auth/oidc	The <code>clientId</code> and <code>clientSecret</code> fields have been removed. Instead, use the <code>clientCredentials</code> object contained in the object <code>oidcConfiguration</code> to set the <code>clientId</code> and <code>clientSecret</code> fields. Further, the <code>oidcConfiguration</code> field is now required. Previously, it was optional.
/identityMappings	The <code>identityMappingPairs</code> field has been removed. In PingAccess 4.1, <code>/identityMappings</code> was converted to support custom <code>IdentityMappings</code> and as a result, uses an extensible configuration model to support arbitrary configuration for custom <code>IdentityMappings</code> . With this change, the <code>IdentityMapping</code> from previous versions become the <code>HeaderIdentityMapping</code> . The schema for configuring a <code>HeaderIdentityMapping</code>

Endpoint	Change Description
	can be retrieved from the <code>/identityMappings/descriptors/headeridentitymapping</code> endpoint. The data previously contained in a <code>identityMappingPair</code> will map to an element in the <code>attributeHeaderMappings</code> array. Specifically, the <code>IdentityMappingPair.nameToExtractValueFrom</code> field will map to the <code>attributeName</code> field in an <code>attributeHeaderMapping</code> , the <code>IdentityMappingPair.nameToInjectValueInfo</code> field will map to the <code>headerName</code> field in an <code>attributeHeaderMapping</code> , and the <code>IdentityMappingPair.subject</code> field will map to the <code>subject</code> field in an <code>attributeHeaderMapping</code> .
<code>/webSessionManagement</code>	The <code>maxNumberOfKeys</code> field has been removed. Previously, this field is no longer exposed in the Admin API.

### PingAccess 4.3.4 - December 2017

PingAccess 4.3.4 is a cumulative maintenance release for PingAccess 4.3, which introduced several new features, including client certificate authentication, admin session revocation, support for PostgreSQL audit logging, and support for standards based OIDC single logout. For more information, see the [release notes for PingAccess 4.3](#).

#### Resolved issues

Ticket ID	Description
PA-8706	Fixed a performance issue that is encountered when you create and assign a large number of virtual hosts to an application.
PA-7936	Fixed an issue in the Upgrade Utility.

### PingAccess 4.3.3 - October 2017

PingAccess 4.3.3 is a cumulative maintenance release for PingAccess 4.3, which introduced several new features, including client certificate authentication, admin session revocation, support for PostgreSQL audit logging, and support for standards based OIDC single logout. For more information, see the [release notes for PingAccess 4.3](#).

#### Resolved issues

Ticket ID	Description
PA-8380	Fixed a potential security issue.

### PingAccess 4.3.2 - September 2017

PingAccess 4.3.2 is a cumulative maintenance release for PingAccess 4.3, which introduced several new features, including client certificate authentication, admin session revocation, support for PostgreSQL audit logging, and support for standards based OIDC single logout. For more information, see the [release notes for PingAccess 4.3](#).

#### Resolved issues

Ticket ID	Description
PA-7629	Fixed an issue where, during an upgrade, the Linux installer showed a status of FAILED if you chose not to replace the existing service.
PA-8205	Fixed a potential security issue.
PA-8248	Fixed an issue where dbuserpasswd and dbfilepasswd .sh and .bat files would fail.
PA-8253	Fixed an issue where a race condition would cause the engine or admin replica to miss an auth token key roll or a web session key roll
PA-8254	Fixed an issue where a race condition would cause the engine to miss configuration updates.
PA-8255	Fixed an issue where an engine data.zip could not be downloaded after a configuration import.

### PingAccess 4.3.1 - August 2017

PingAccess 4.3.1 is a cumulative maintenance release for PingAccess 4.3, which introduced several new features, including client certificate authentication, admin session revocation, support for PostgreSQL audit logging, and support for standards based OIDC single logout. For more information, see the [release notes for PingAccess 4.3](#).

### Resolved issues

Ticket ID	Description
PA-7632	Fixed an issue where, during an upgrade on RHEL 7 using the Linux installer, the install may show as FAILED due to an error starting the systemd PingAccess service.
PA-7931	Fixed an issue where PingAccess X-Forwarded-For IP address validation was not allowing port numbers.
PA-7975	Fixed an issue where Auth token management settings were not being properly migrated during an upgrade.

### PingAccess 4.3 - June 2017

#### Enhancements

##### PingAccess for Azure AD

A version of the PingAccess license is available for premium users of Microsoft's Azure AD. This license permits the use of a limited feature set supporting integration with the Azure AD Application Proxy and provides secure access for up to 20 legacy on-premises applications. Users are able to take advantage of the full PingAccess feature set by purchasing a full license. To learn more, see:

- [PingAccess for Azure AD overview](#)
- [Getting started with PingAccess for Azure AD](#)

##### Token provider specific attribute transformation

PingAccess provides a mechanism that transforms incompatible attribute payload data received from Azure AD's OIDC UserInfo endpoint and retrieves the display names for groups from the Azure AD Graph API.

##### Client certificate authentication

PingAccess now facilitates client certificate authentication with protected applications that accept client certificates as HTTP headers. This feature works by retrieving client certificates sent by the browser during the SSL/TLS handshake with PingAccess and adding them, through the use of identity mappings, (JWT and Header), to the HTTP request headers sent to a site. An example scenario is using PingAccess to protect the public interfaces of PingFederate when user authentication requires a certificate to authenticate. For more information on PingFederate configuration, see [Defining Proxy Options](#).

### Administrator session revocation

PingAccess now supports full administrator SSO session revocation. When using PingFederate as the token provider, PingAccess can be configured to utilize admin SLO functionality that redirects PingAccess to the SLO endpoint to initiate the sign-out process. When using PingAccess basic authentication, server side session tracking is used to verify the session and deny access for invalidated cookies after sign-out.

### Support for systemd services

The Linux installer now supports installation of systemd services on RHEL 7, while maintaining System V service support on RHEL 6. The systemd init system offers more robust service management and a simpler configuration. When upgrading PingAccess on a system that supports systemd, you will be given the option to replace an existing System V service with a newer systemd service. A template service unit configuration file is also provided for manual installation of systemd services on non-RHEL systems.

### PingAccess support for PostgreSQL audit logging

In addition to existing audit logging options for Oracle and MS SQL Server databases, PingAccess now supports database audit logging for the API, engine, and agent to PostgreSQL databases. Configure PostgreSQL audit logging using the `/conf/log4j2.db.properties` and `/config/log4j2.xml` files available in the PingAccess installation directory.

### Support for standards based OIDC single logout

Support has been added for administrator single logout when using a compatible third party OpenID Connect provider. A compatible provider supports OpenID Connect Session Management 1.0.

### Improved configuration export and import

The format of the exported configuration data has been modified to more closely align with the PingAccess REST API and to provide a more user-friendly JSON structure. In addition, sensitive data is now obfuscated to help protect data at rest.

### PingAccess support on additional operating systems

PingAccess Server is now supported on the following systems:

- SUSE Linux Enterprise Server v12
- RedHat Enterprise Linux 7.3
- Windows Server 2016

### Resolved issues

Ticket ID	Description
Various	This release contains multiple UI improvements and enhancements.
PA-5208	Fixed an issue where reverse DNS lookup was failing on PingAccess startup.
PA-5683	Fixed an issue where authentication requirements rules were not processing in the correct order.
PA-6332	Fixed an issue where an OAuth scope and attribute policy would succeed on an unprotected application when <code>negate</code> is set to <code>true</code> .
PA-6396	Fixed an issue where logouts were not being handled correctly when using a third party token provider.
PA-6582	Fixed an issue where a warning was not issued when a rule that requires an identity was assigned to an unprotected application.
PA-6585	Fixed an issue where nonce cookies were not being removed after unsuccessful requests.
PA-6776	Fixed an issue where an admin was taken to the basic authentication page after their session expired if using admin SSO.

Ticket ID	Description
PA-6884	Fixed an issue where the policy screen did not allow scrolling when using Internet Explorer 11.
PA-6900	Fixed an issue where a value of 0 was not accepted as a valid Session Poll Interval.
PA-6965	Fixed an issue where the upgrade utility was failing when there are spaces in the path.
PA-6973	Fixed an issue where PingAccess was calculating the agent token cache TTL for a web session incorrectly which may have resulted in an agent not consulting the PingAccess policy server when either the PingFederate Session State Cache Interval or the Refresh User Attributes Interval as configured for the web session had elapsed.
PA-6981	Fixed an issue where sessions were not being invalidated after changing the Web Session Issuer.
PA-7033	Fixed an issue where no warning was displayed when modifying the Admin SSO configuration that may have resulted in unexpected lockout.
PA-7040	Fixed an issue where a JWT identity mapping attribute with a period in its name was not being mapped into a claim.
PA-7151, PA-7190	Fixed issues with the encoding of special characters used in searches.
PA-7350	Fixed an issue where a Client Certificate JWT claim name didn't support nested (dot notated) formatting.
PA-7603	Fixed an issue where the installer was hanging during upgrade because the service had not yet stopped.

### PingAccess 4.2.3 - April 2017

PingAccess 4.2.3 is a cumulative maintenance release for PingAccess 4.2, which introduced several new features, including improved admin UI workflow, certificate trust chains, event storage in MS SQL, agent policy server failover, unknown resource handling, forward proxy handling, and installer support for upgrades. For more information, see the [release notes for PingAccess 4.2](#).

### Resolved issues

Ticket ID	Description
PA-6929	Fixed an issue where multiple nonce cookies were created during abandoned authentication attempts.
PA-6943	Fixed an issue where PingAccess was not encoding some data correctly when communicating with the token endpoint.
PA-6978	Fixed an issue where rate limiting rules were not correctly enforcing the application scoped rate limit.
PA-7035	Fixed an issue where identities were being associated with authentication requests to anonymous resources.
PA-7038	Fixed an issue where there was no redirect or success banner following successful object creation.
PA-7042	Fixed an issue where warnings were being included in logs when an HTTP response write error occurs.

### PingAccess 4.2.2 - February 2017

PingAccess 4.2.2 is a cumulative maintenance release for PingAccess 4.2, which introduced several new features, including improved admin UI workflow, certificate trust chains, event storage in MS SQL, agent policy server

failover, unknown resource handling, forward proxy handling, and installer support for upgrades. For more information, see the [release notes for PingAccess 4.2](#).

## Resolved issues

Ticket ID	Description
PA-6437	Fixed an issue where some velocity variables were not present in some conditions.
PA-6551	Fixed an issue where ID Token attributes were incorrectly included in the PingAccess cookie if <code>Include User Info in ID Token</code> was enabled and <code>Cache User Attributes</code> is enabled in the web session.
PA-6577	Fixed an issue in Basic Administrator Authentication where the "Password change failed" message was being displayed even when the password change succeeded.
PA-6602	Fixed an issue where some <i>Warning</i> messages were not being displayed on the Policies page.
PA-6695	Fixed an issue where PingAccess was using the incorrect proxy to communicate with PingFederate.
PA-6877	Fixed an issue where a PingAccess upgrade from release 4.0 to release 4.2.1 was failing.

## PingAccess 4.2.1 - December 2016

PingAccess 4.2.1 is a cumulative maintenance release for PingAccess 4.2, which introduced several new features, including improved admin UI workflow, certificate trust chains, event storage in MS SQL, agent policy server failover, unknown resource handling, forward proxy handling, and installer support for upgrades. For more information, see the [release notes for PingAccess 4.2](#).

## Resolved issues

Ticket ID	Description
PA-6436	Fixed an issue where configured sites were unreachable after upgrade to PingAccess 4.2, build 4.2.0.6.
PA-6317	Fixed an issue where, in a clustered environment using admin SSO, if the admin console is configured to use a proxy, and if the admin replica is not configured to use a proxy, admin SSO would not succeed in a failover situation.
PA-6408	Fixed an issue where you could not add a new row to a content rewrite rule via the UI.

## PingAccess 4.2 - December 2016

### Enhancements

#### Improved Admin UI workflow

Improved configuration workflow that allows you to easily configure required elements without navigating away from a page or losing information.

#### Certificate trust chains

This feature allows you to add or modify a certificate chain by adding or removing intermediate and root certificates.

#### Audit event storage in MS SQL 2012 & 2016

Configure PingAccess to store logs in MS SQL Server 2012 & 2016.



**Agent policy server failover**

Configure a failover host and port on an agent definition to enable agent failover to a backup policy server in the event of a failure of the primary policy server.

**Unknown resource handling**

Specify handling of resources for which there is no configured application. This feature allows you to customize error pages for unknown resources and to enable request pass-through for PingAccess agents. This feature is useful when specifying a policy for an agent protected resource that has both secure and public content within the scope of the agent.

**Forward proxy support for outbound calls**

Specify and configure a forward proxy that may be required to make outbound calls.

**Installer support for upgrades**

You can now install and upgrade PingAccess on Windows and RHEL using the applicable installer package.

**Resolved issues**

Ticket ID	Description
PA-3407	Fixed an issue that caused poor performance on reload when there are applications with many groovy script rules.
PA-4671	Improved handling of unsupported Content-Types during POST preservation.
PA-5400	Fixed an issue where an exception was incorrectly added to logs during evaluation of an OAuth Attribute Rule on an unprotected API application.
PA-5405	Fixed an issue where an exception was encountered by the error response handler if the connection to PingFederate is misconfigured.
PA-5448	Fixed an issue where a session cookie was not reissued if a transaction failed.
PA-5500	Fixed an issue with the display of values when viewing an Identity Mapping in expanded view.
PA-5502	Fixed an issue that caused upgrades to fail under certain conditions.
PA-5538	Fixed an issue where Save and Discard Changes buttons were not visible when adding or making changes to IP Source or Host Source settings on the HTTP Requests page.
PA-5644	Fixed an issue where expired tokens were logging incorrect error messages.
PA-5791	Improved handling of errors that were occurring when trying to decrypt a state parameter with a key that no longer exists.
PA-6027	Fixed an issue where warning messages weren't being parsed correctly.
PA-6271	Fixed an issue where upgrades were failing because the target instance admin console was not responding.
PA-6273	Fixed an issue where runtime requests were failing when using Internet Explorer 9 or 10.

**PingAccess 4.1.3 - November 2016**

PingAccess 4.1.3 is a cumulative maintenance release for PingAccess 4.1, which introduced several new features, including support for standards-based OpenID Connect provider login, JWT based identity mapping, OAuth 2.0 token introspection, and session timeout synchronization with PingFederate. For more information, see the [release notes for PingAccess 4.1](#).

**Resolved issues**

Ticket ID	Description
PA-6014	Fixed an issue where the Upgrade Utility was not migrating the configuration for 3rd party OAuth server or 3rd party OIDC provider.
PA-6213	Fixed a potential security issue.

### PingAccess 4.1.2 - October 2016

PingAccess 4.1.2 is a cumulative maintenance release for PingAccess 4.1, which introduced several new features, including support for standards-based OpenID Connect provider login, JWT based identity mapping, OAuth 2.0 token introspection, and session timeout synchronization with PingFederate. For more information, see the [release notes for PingAccess 4.1](#).

#### Resolved issues

Ticket ID	Description
PA-5681	Fixed an issue where Heartbeat URL displays a blank page when Detailed Response is disabled.
PA-5828	Fixed an issue where agents using the same agent certificate would generate an <code>IllegalStateException</code> when replicating to a cluster of engines.
PA-5830	Fixed an issue where Identity Mappings no longer shows multiple attributes with the same name when there is a only difference in case.
PA-5876	Fixed an issue where the <code>agentCacheInvalidatedResponseDuration</code> field is incorrectly set to 0, causing the <code>vnd-pi-cache-invalidated</code> header to be missing from the agent response.
PA-5879	Fixed an issue where the <code>vnd-pi-sub</code> header is missing from the agent response.

### PingAccess 4.1.1 - September 2016

PingAccess 4.1.1 is a cumulative maintenance release for PingAccess 4.1, which introduced several new features, including support for standards-based OpenID Connect provider login, JWT based identity mapping, OAuth 2.0 token introspection, and session timeout synchronization with PingFederate. For more information, see the [release notes for PingAccess 4.1](#).

#### Resolved issues

Ticket ID	Description
PA-5665	Fixed an issue where you could not save changes to a Trusted Certificate Group.
PA-5654	Fixed an issue where upgrade would fail to migrate Authentication Requirements Rules.
PA-5651	Fixed an issue where a new Authentication Requirements Rule could not be created after upgrading to version 4.1.
PA-5646	Fixed an issue where RuleSets were always evaluated before Rules.
PA-5563	Fixed an issue where Admin API transaction logs were appearing in <code>agent_audit_log</code> and <code>engine_audit_log</code> tables when using the Log4j2 JDBCAppender.
PA-5556	Fixed an issue with Log4j2 JDBCAppender where a memory leak would create out of memory issues and log events were not added to the failover log file.
PA-5547	Fixed an issue with Engine and Agent Audit Logging where FailedRule entries were not appearing in logs.

Ticket ID	Description
PA-5451	Fixed an issue where a change to enable Skip Hostname Verification would not take effect until a restart of PingAccess.

## PingAccess 4.1 - August 2016

### Enhancements

#### Support for standards-based OpenID Connect provider login


PingAccess 4.1 introduces support for the use of a standards-based OpenID Connect provider as a source for authentication tokens.

There are several requirements and prerequisites that must be fulfilled before enabling this feature, including:

- All existing applications must be disabled prior to changing the token provider
- Only Basic Authentication for the administrator console must be enabled. Administrator SSO must be disabled.
- The third-party OpenID Connect token provider must support OpenID Connect issuer discovery

Once the token provider is switched, applications can be re-enabled, and Administrator SSO can be re-enabled. We recommend that if Administrator SSO is provided, you enable administrative roles on the **Single Sign-On** tab of the **Admin Authentication** configuration page and configure **required Administrator Attributes** to ensure that only administrators are permitted access to the administrative console.

To change the token provider type, navigate to **Settings > System > Token Provider**, then click **Change Token Provider Type**.

 **Note:** This feature only works with web applications protected by PingAccess; API applications are not supported, and cannot be enabled if they exist in the configuration.

#### Ability to create identity mappings based on contents of a JWT

Identity mappings can now be added as claims to a signed JWT as well as in HTTP request headers to the back-end server. These JWTs are signed using the issuer information set in **Settings > Access > Auth Token Management**. The token itself can be validated using the `/pa/authtoken/JWKS` endpoint.

#### OAuth 2.0 Token Introspection

[RFC7662](#) introduced a standard for querying an OAuth authorization server about the state of an OAuth 2.0 token and to determine meta-information about it.

This release of PingAccess provides an option to use the PingFederate 8.2 introspection endpoint.

 **Note:** This feature requires the use of PingFederate 8.2, and it only applies to API applications.

Enable Token Introspection by navigating to **Settings > System > Token Provider > OAuth Resource Server > Use Token Introspection Endpoint**.

#### PingFederate session timeouts can be synchronized with PingAccess

PingAccess now provides a means for PingFederate 8.2 to synchronize its session timeouts with PingAccess.

### Resolved issues

Ticket ID	Description
PA-4886	Fixed an issue when trying to generate a CSR for a certificate that includes a DNS name subject alternative name that starts with a wildcard (*).
PA-4230	Fixed an issue where Agent name can be invalid Unicode characters in HTTP headers.
PA-5306	Fixed an issue when configuring sites with MutualTlsSiteAuthenticator.

Ticket ID	Description
PA-5303	Fixed an issue where the status icon no longer appears for the replica administrative node.
PA-4892	Fixed an issue where a socket is not properly closed on error.
PA-4891	Fixed an issue where a socket is not properly closed on error.
PA-4882	Fixed an issue where the callback endpoint always returns a Forbidden when combining a Code WebSession and the use of X-Forwarded-Host.
PA-4878	Removed libraries that are no longer in use.
PA-4866	Fixed an issue where some plugin fields were not accepting input.
PA-4610	Modified run scripts to use <code>/dev/random</code> for cryptographic processing.
PA-4581	Fixed an issue where a slow backend site causes PingAccess to throw an OOM (Out of memory) exception.
PA-3429	Fixed an issue where local DB was not being set to a known state before applying replicated configuration.

### PingAccess 4.0.4 - July 2016

PingAccess 4.0.4 is a cumulative maintenance release for PingAccess 4.0, which introduced many new features, such as contextual authentication, WebSocket support, and a new administrative UI. For more information, see the [release notes for PingAccess 4.0](#).

#### Resolved issues

Ticket ID	Description
PA-5115	Resolved an issue with the automatic creation of configuration archives when the administrator logged in if the administrator authentication used Admin SSO.
PA-5189	The PingAccess Upgrade Utility no longer overwrites the certificate alias attribute with a GUID during an upgrade.

### PingAccess 4.0.3 - May 2016

PingAccess 4.0.3 is a cumulative maintenance release for PingAccess 4.0, which introduced many new features, such as contextual authentication, WebSocket support, and a new administrative UI. For more information, see the [release notes for PingAccess 4.0](#).

#### Resolved issues

Ticket ID	Description
PA-4862	Corrected an issue where upgrading an environment that has Admin SSO enabled could result in an admin authentication loop.
PA-4867	Addressed a potential security issue.
PA-4875	Fixed an issue where paginated collections of entities (sites, applications, etc.) could cause the administrative UI to hang while reloading.
PA-4877	Addressed an issue in the Windows service configuration that could impact performance.
PA-4889	Resolved a conflict between using the <code>code</code> OpenID Connect Login Type setting and the use of <code>X-Forwarded-Host</code> headers. In this situation, the <code>X-Forwarded-</code>

Ticket ID	Description
	Host value was not matching the PingFederate <code>redirect_uri</code> tied to the authorization code, and authentication would fail.

### PingAccess 4.0.2 - April 2016

PingAccess 4.0.2 is a cumulative maintenance release for PingAccess 4.0, which introduced many new features, such as contextual authentication, WebSocket support, and a new administrative UI. For more information, see the [release notes for PingAccess 4.0](#).

#### Resolved issues

Ticket ID	Description
PA-4639	Fixed a defect that prevented database audit logging from working.
PA-4635	Corrected an issue where changes to the <b>Update Token Window(s)</b> setting in the web sessions configuration did not persist.
PA-4627	Addressed an issue that could cause the logging buffer to fill, preventing PingAccess from processing requests.
PA-4604	For backchannel OAuth validation requests to PingFederate, the effective URL of the request is used. This effective URL now includes the URL scheme.
PA-4423	When parsing Set-Cookie header values, PingAccess now uses the <code>pa.uri.strict</code> property in <code>run.properties</code> in determining whether to allow or deny invalid characters in the query.

### PingAccess 4.0.1 - March 2016

PingAccess 4.0.1 is a cumulative maintenance release for PingAccess 4.0, which introduced many new features, such as contextual authentication, WebSocket support, and a new administrative UI. For more information, see the [release notes for PingAccess 4.0](#).

#### Resolved issues

Ticket ID	Description
PA-4602	Fixed an issue where editing a resource would remove policies assigned to that resource.
PA-4598	Addressed an issue where the PingAccess Upgrade Utility would reset agent shared secrets.
PA-4416	Modified logging so expired tokens do not generate ERROR or WARN level log messages, since token expiration is an expected and normal operational condition.
PA-4415	Modified logging so responses from the Token Mediator Site Authenticator do not generate an INFO level message for every transaction.
PA-4232	When a <code>%</code> character appears in a URI processed in some way by PingAccess, if the character is not clearly part of a URL-encoded string, PingAccess treats it as a literal <code>%</code> character, and URL-encodes it as <code>%25</code> for processing.

## PingAccess 4.0 - February 2016

### Enhancements

#### Contextual authentication

PingAccess 4.0 introduces the Authentication Requirements rule type. Authentication Requirements rules define the type of authentication required to gain access to a web application, allowing for step-up authentication for access to more sensitive parts of an application. Additionally, requirements can be combined with other access control criteria to enable contextual authentication use cases, such as re-evaluating a user's authentication status if they move to an untrusted network, or using attribute values to trigger a requirement for a stronger form of authentication.

#### WebSocket security

Applications that use the WebSocket protocol can now be protected by PingAccess, allowing PingAccess access control policies and processing rules to be used to enhance security. WebSocket-based applications are supported by PingAccess as a gateway or when using PingAccess agents.

#### New administrative user interface

PingAccess 4.0 includes a completely redesigned administrative user interface, providing a uniform administration experience for PingAccess, PingFederate, and PingOne administrators..

#### Auditor and administrator roles

PingAccess 4.0 introduces role-based access control for administrative interfaces. Users can be assigned to the Administrator role, allowing full access to view and change the PingAccess configuration, or they can be assigned to the Auditor role, which allows read-only access to the configuration. Role-based access control is applied for both Admin SSO to the administrative UI or OAuth authentication to access the administrative API.

#### New REST API version

The PingAccess REST APIs have been updated to version 2, bringing a number of enhancements to that administrative interface. See [Administrative API updates](#) on page 280 for detailed information about these changes and how to migrate to the new API version.

#### New logging system

PingAccess now uses log4j2 as the logging engine. As a consequence, you will need to migrate your existing blitz4j configuration to log4j2.

This change also brings several other enhancements, such as failover logging capabilities, improved logging performance, and the ability to have changes to the log level applied without requiring a restart of PingAccess.

#### Oracle JRE support

PingAccess 4.0 removes the need to deploy the full Oracle JDK on runtime servers. Oracle JRE provides all required services.

#### Windows and RHEL installers

We've now created an MSI-based installer for Windows and a CLI-based installation script for RedHat Enterprise Linux to simplify the installation of PingAccess.

Both installers guide you through the installation decision points with a wizard-style interface, and update the PingAccess configuration based on those decisions.

#### Upgrade utility

The Upgrade Utility is now the only tool required for performing upgrades, regardless of the source and target versions of PingAccess.

The PingAccess Upgrade Utility has also been enhanced to support migrating plugins developed with the Addon SDK.

## Other enhancements

There are numerous other usability and functionality enhancements in this release of PingAccess, including:

- In the PingAccess administrative user interface, **Description** fields have been added or modified such that they are now available and consistent across agent, engine, and application entities. These fields all support multi-line values, and entering their values is optional.
- In the PingAccess administrative console UI, the **Use Target Host Header** field is now cleared by default. The recommended practice is to send the public host header to the backend site.
- The administrator is now notified in the administrative user interface when PingAccess needs to be restarted after certain configuration changes.
- The Windows service name and description properties in `PA_HOME/sbin/windows/PingAccessService.conf` is now in the following format:

```
# Display name of the service
  wrapper.ntservice.displayname=PingAccess Service
# Description of the service
  wrapper.ntservice.description=PingAccess Service
```

- PingAccess validates the web session's configured cookie domain to ensure it does not conflict with any virtual hosts associated with the applications the session assigned to.
- A unique nonce cookie name is now generated, rather than reusing the same name, resolving a potential issue if multiple authentication requests are generated prior to being handled by PingFederate.
- URI fragments are now preserved through the authentication process when a user is forced to re-authenticate to a resource.
- Administrators can now create localized error messages for user-facing error pages displayed by PingAccess.
- For the Rewrite Cookie Domain rule and the Rewrite Response Header rule, administrators can now specify that the public-facing cookie domain or URI path be rewritten for all site target hosts, rather than needing to create a rule for each target host.
- The Content Rewrite rule now supports rewriting content from applications that do not specify a `Content-Type` header.
- The OAuth Attribute Value rule now supports multiple attributes and values, making it consistent with the Web Session Attribute rule.
- Error messages can now be defined based on the application the user accessed in addition to a generic system-wide message being used.
- Added validation to prevent a misconfiguration that could result in an authentication loop. Administrators need to be aware that the web session cookie will be sent to the configured domain. As part of the feature, validation is done in the admin console to prevent administrators from defining an invalid cookie domain.

## Resolved issues

Ticket ID	Description
PA-242	Validation for Groovy Rules that use inapplicable matchers
PA-1099	Fixed Database decryption bug
PA-1900	Fixed PingAccess Agent Protocol compliance issue
PA-1903	Scripts not using JAVA_HOME to locate Java installation
PA-2362	Token Mediation deleting 3rd party cookies
PA-2419	Fixed session expiration response
PA-2430	Fixed upgrade utility NPE when target PingAccess fails to start
PA-2696	OAuth Groovy Script Rule Is Not Executing During Response Processing
PA-2727	Improved PingAccess core processing to increase server stability.

Ticket ID	Description
PA-2815	Fixed Admin API validation issue
PA-2897	Stale post preservation cookie results in Internal Server Error from PA
PA-2899	Backup on authentication not created when using Admin SSO
PA-2900	Improved behavior when empty post body is sent to agent
PA-2907	Fixed error when OIDC token is revoked
PA-2926	Fixed admin re-authentication handling
PA-2950	Fixed NPE caused by specific Admin API call
PA-2972	Fixed NPE caused by specific Admin API call
PA-2974	PingAccess fails to start (on first attempt) on Windows
PA-2979	Fixed Admin UI NPE when importing CSR
PA-3008	Enhanced handling of values in imported keypairs to handle non-string SANs.
PA-3039	POST preservation fails with exception if *:3000 virtual host not defined
PA-3053	PingAccess linux service script doesn't work with non-root user
PA-3054	ehcache WARN level log messages in pingaccess.log on startup
PA-3101	PingFederate authorization server showing up as blank in admin UI
PA-3205	PUT requests where the request body is no longer available after an initial failure are no longer retried.
PA-3217	Admin replica failover does not take effect until after replica restart
PA-3224	OAuth Rule Deny Returns 401 to Client but Logs in Audit Log as 403
PA-3227	Unauthenticated response connection is not closed properly
PA-3253	Web session cookie domain needs validation against associated application / virtual hosts
PA-3293	The PingAccess Upgrade Utility now uses the address defined in the source PingAccess server's admin.bindAddress configuration.
PA-3329	Web Session edit requires re-entry of Client Secret
PA-3359	Secrets in /sharedSecrets not obfuscated by Admin API
PA-3398	Improvements have been made to the API responses so that they're not HTML-escaped, resolving occasional issues encountered with the configuration import and export functionality.
PA-3454	Web Session Data integrity violation when trying to change a web session name with one that already exists
PA-3469	500 internal server error returned if Protocol Source Header Name too long
PA-3480	Editing Rule Sets allows OAuth rules to be applied to Web Application
PA-3509	vnd-pi-resource cache grows unbounded
PA-3574	Reload race condition causes intermittent failure of reload
PA-3635	Off-by-one errors in header size and count limits
PA-3663	NPE if you try to save Rule with empty error response code
PA-3669	Introduced the pa.site.tls.sni.legacyMode configuration parameter to give the administrator control over how SNI server_name is set.




Ticket ID	Description
PA-3692	PingAccess no longer throws an exception when POST preservation data cannot be retrieved.
PA-4160	User is able to add the same cert multiple times to same Trusted Cert Group
PA-4197	Inline field error message not displayed
PA-4215	NPE when trying to Import CSR Response
PA-4219	Resolved an issue where POST Preservation used in conjunction with the Rewrite Content rule, a null pointer exception could occur.
PA-4234	RuleSet allows SuccessCriteria of SuccessIfAnyOneSucceedsLastHandlesError
PA-4244	Network Range Plugin: Error Response Content-Type should be required
PA-4453	PostPreservation pi.postsrv claim is not being cleared after encoded post
PA-4504	On startup, PF back-channel failover is broken

### Administrative API updates

With PingAccess 4.0, the administrative API interface has been updated to version 2. This means there were some fairly significant changes in some of the API endpoints.

Administrators who have written code that uses the API endpoints to automate administrative tasks will need to update their code to use the new endpoints.

 **Note:** The fundamental change in all administrative API calls will be a change in the endpoint URI from `/pa-admin-api/v1` to `/pa-admin-api/v2`. For example, if you made a call to `/pa-admin-api/v1/pingfederate/admin`, your code would need to be updated to point to `/pa-admin-api/v2/pingfederate/admin`

The following table identifies specific API changes that may need more updates than just the path to the API endpoint.

Endpoint	Change Description	How to Update
<code>/pingFederateAdmin/accessTokens</code>	This endpoint now accepts an <code>encryptedvalue</code> or a <code>value</code> . The <code>encryptedvalue</code> is a string that contains an obfuscated value, as obfuscated when using the <code>obfuscate.sh</code> script.	In the v1 API, the model template includes a <code>clientSecret</code> property. This is now replaced with a <code>clientSecret</code> property with a subschema that can contain either an <code>encryptedValue</code> string or a <code>value</code> string.
<code>/webSession</code>	The name of this endpoint has been changed to <code>/webSessions</code>	Modify any code that accesses the v1 <code>/webSession</code> endpoint to point to <code>/webSessions</code> instead.
<code>/webSessions</code>	As with the <code>/pingFederateAdmin/accessTokens</code> endpoint, the <code>/webSessions</code> endpoint now supports either an <code>encryptedValue</code> or a <code>value</code> .	This change is the same as for the <code>/pingFederateAdmin/accessTokens</code> endpoint.
<code>/auth/oauth</code>	As with the <code>/pingFederateAdmin/accessTokens</code> endpoint, the <code>/auth/oauth</code> endpoint	This change is the same as for the <code>/pingFederateAdmin/accessTokens</code> endpoint.

Endpoint	Change Description	How to Update
	now supports either an <code>encryptedValue</code> or a <code>value</code> .	
<code>/auth/basic</code>	When the <code>GET</code> method is used with this endpoint, it now returns whether basic authentication is enabled or disabled. Deprecated fields are no longer returned.	If your code performs actions based on the v1 response body for <code>GET</code> requests, it will need to be modified.
<code>/applications/{id}</code>	Removed the <code>resourceIds</code> array, as they are not managed as part of the <code>applications</code> endpoint, but instead are managed through the <code>/applications/{id}/resources</code> endpoint.	Code that depends on the <code>resourceIds</code> array returned by the <code>GET</code> method will need to be updated.
<code>/applications</code>	The <code>virtualHostId</code> property has been removed.	Use the first value in the <code>virtualHostIds</code> array instead.
<code>/sites</code>	The <code>host</code> and <code>port</code> fields have been combined into the <code>targets</code> array. These values were used in older versions of PingAccess, when a single target was defined for a <code>Site</code> object.	Update your code to use the <code>targets</code> array. Each element in this array is a <code>host:port</code> pair.
<code>/sharedSecrets,</code> <code>/engines,</code> and <code>/adminConfig/</code> <code>replicaAdmins</code>	These three endpoints have been updated to allow their respective configurations to be individually exported and imported.	This is new functionality; no changes to existing code is required.
<code>/users</code>	<p>This endpoint has been updated to facilitate future use of multiple users for basic authentication. In the v1 API, changes made through this endpoint applied to the <code>Administrator</code> user, as that was the only user. Changing passwords or toggling the <code>slaAccepted</code>, <code>firstLogin</code>, and <code>showTutorial</code> settings were handled through this endpoint.</p> <p>In the v2 API, the <code>GET</code> method on <code>/users</code> returns a user list, but the following additional endpoints have been created:</p> <ul style="list-style-type: none"> <li><code>/users/{id}</code></li> <li><code>/users/{id}/password</code></li> </ul> <p>A <code>GET</code> request to the first of these returns information for the specified user.</p> <p>A <code>PUT</code> to the first allows the user flags to be manipulated as they would have been in the <code>/users</code> endpoint in v1 of the API.</p>	Code that uses the <code>/users</code> endpoint will need to be updated to use the new endpoints. Where it was possible in the v1 API to change the flags and the password with a single API call, two calls now will need to be used - the first to <code>/users/{id}</code> , and the second to <code>/users/{id}/password</code> .

Endpoint	Change Description	How to Update
	To change the user password, use a PUT request on the second endpoint, specifying the old and new passwords in the payload.	

### PingAccess 3.2.6 - February 2016

PingAccess 3.2.6 addresses the following issues:

#### POST Preservation

PingAccess no longer throws an exception when POST preservation data cannot be retrieved.

#### PUT requests with On-Demand availability profiles

PUT requests where the request body is no longer available after an initial failure are no longer retried.

#### Server Name Indication

Corrected a `server_name` and `host` data/header mismatch when using SNI.

#### Non-string Subject Alternative Name values in imported keypairs

Enhanced handling of values in imported keypairs to handle non-string SANs.

#### Admin console not accessible after upgrade

Corrected an issue where an upgrade could result in the administrative console not being accessible.

#### Security fix

Fixed a security vulnerability (SECADV012).

### PingAccess 3.2.5 - December 2015

PingAccess 3.2.5 addresses the following issue:

#### Issues resolved:

Corrected a resource allocation issue when using PingAccess agents configured with caching enabled in the agent.

### PingAccess 3.2.4 - December 2015

PingAccess 3.2.4 addresses the following issues:

#### Authentication loop with cached user attributes

Resolved an issue where a misconfiguration in the web session user attribute caching settings could cause an authentication loop.

#### Security updates

Fixed a few minor potential security issues.

#### Improved expiration handling for AccessToken validation during API flow

When validating an access token in the API flow which did not contain an `expires_in` value, a null pointer exception would be thrown.

### PingAccess 3.2.3 - October 2015

PingAccess 3.2.3 addresses the following issues:

#### Security issues

Two minor potential security issues are addressed in this release.

#### Fixed upgrade utility SSLHandShakeException failures with Java 8 update 51 or later

When updating from versions of PingAccess that use deprecated RC4 ciphers, the PingAccess Upgrade Utility would fail.

**Fixed potential issues with POST preservation in certain configuration scenarios**

In certain rare cases, POST preservation would fail as a result of a configuration error. Additional checks have been added to prevent these configuration errors from occurring.

**Fixed issue that could lead to unexpected request failures when using token mediation**

If a protected application relies on cookies being sent with the mediated token, if the token mediation flow needed to retry, the additional cookies delivered with the token would be removed, resulting in unexpected application failures.

**Fixed issue where the Subject Alternative Name is missing from the CSR for key pairs created in PingAccess**

When a key pair was generated in PingAccess and a Certificate Signing Request was issued, the CSR was missing the Subject Alternative Names.

**Improvements to the `install-service.bat` script**

When executed from a directory with a space in the path, the `install-service.bat` script would fail. This script has been updated to correct the issue.

**Made the `dbfilepasswd` and `dbuserpasswd` scripts more robust**

These scripts now run from paths containing spaces, and use the Java executable identified by `JAVA_HOME`.

**Added validation to prevent a potential redirect loop**

When a web session cookie domain is configured incorrectly, a user's browser could end up in an endless redirection loop. PingAccess now includes additional validation to prevent this misconfiguration.

**PingAccess 3.2.2 - August 2015**

PingAccess 3.2.2 addresses the following issues:

**The Admin API should require `authnReqListId` be set to 0 for an anonymous user**

Authentication requirements should not be allowed to be added to an anonymous resource. The PingAccess UI enforces this constraint, but previously the Admin API did not.

**Configuration backup archive not created when using Admin SSO**

When an administrator user logs into the administrative console using SSO authentication, PingAccess does not create a configuration archive. The archive is created successfully when the administrator logs in using basic authentication.

**Unable to reorder PingAccess rules inside Rule Sets in UI**

Making a change in the order of rules within a Rule Set produces a success message, but after navigating to another page and returning to the Policy page, the rules are displayed in the previous order, as it was before the change.

**JWT signature error produced when integrating with PingFederate 7.2**

When using unsigned access tokens with PingAccess 3.2, authentication using the OpenID Connect authentication flow may fail, logging the following message in `<PA_HOME>/log/pingaccess.log` when logging is set to `DEBUG`:

```
The JWT has no signature but the JWT Consumer is configured to require one
```

**After Admin Console cookie expiration, the administrator is prompted for basic authentication instead of SSO**

The problem occurs when Admin SSO is enabled and the administrator is logged into the Admin Console, but the access cookie has expired, for example, due to no user activity for a certain time period. In this case, the Admin Console directs the user to the basic login prompt instead of to PingFederate for reauthentication using SSO.

**Concealed plugin fields are not handled correctly in the PingAccess UI**

When entering values in the Basic Authentication Site Authenticator form, if username and password values are filled and then the username is edited again, the password value is saved as null rather than the password value entered. This issue affects any plugin with a concealed field such as a password field.

### **New login required after change to the Admin SSO page even though SSO is disabled**

If the administrator changes the **Client ID** value in the Admin SSO page, but does not select the **Enabled** checkbox, the administrator is required to authenticate again. A new login should be required only if Admin SSO is enabled.

### **Admin SSO authentication fails when PingAccess 3.2 is installed with PingFederate 7.2**

When PingAccess 3.2 is used with PingFederate 7.2, Admin SSO does not work.

### **PingAccess load balancing not enforcing stickiness**

In some cases, PingAccess sticky sessions do not work with PingAccess load balancing.

### **Corrected a potential security vulnerability**

Refer to the Security Advisory SECADV010 on the [Customer Portal](#) for more information.

## **PingAccess 3.2.1 - July 2015**

PingAccess 3.2.1 includes the following fixes:

### **Stale `postprsv` Cookie Causes PingAccess Internal Server Error Response**

In certain circumstances when using POST Preservation, the `postprsv` cookie was not properly maintained, resulting in the cookie not containing the expected information.

### **Intermittent 403 Forbidden Responses when Browser Pre-fetches `favicon.ico`**

In some instances, a browser might attempt to retrieve `favicon.ico` before a session cookie was issued, resulting in access being denied to the resource.

## **PingAccess 3.2 - June 2015**

PingAccess 3.2 includes the following enhancements and new functionality:

### **Access Control Enhancements**

#### **Rate Limiting**

The **Rate Limiting** rule enables administrators to control the allowed request rate to a protected API or Web Application based on the identity, source IP address, OAuth client, or resource being accessed.

#### **New HTTP Request Rules**

The **HTTP Request Rule** has been refactored into two rules: The **HTTP Request Header** and **HTTP Request Parameter** rules. These rules provide additional flexibility for making policy decisions based on values passed either in HTTP headers sent by a client, or based on form data submitted by a client. Both of these new rule types can be configured to require multiple value matches.

#### **Cross-Origin Resource Sharing (CORS)**

Cross-Origin Resource Sharing is a method for allowing restricted resources on a web page to be requested from a domain that is different from the domain that hosts the requested web page. This release of PingAccess adds an access control rule that can be used to control this type of access request.

### **Engine Enhancements**

#### **HTTP and HTTPS Engine Listeners**

When multiple engine listeners are defined, PingAccess now allows the administrator to define, on a per-engine listener basis, whether the listener uses HTTP or HTTPS.

#### **Multiple Access Token Manager Support**

In instances where multiple Access Token Managers are configured in PingFederate to provide OAuth access tokens for API applications, the `aud` OAuth parameter is populated with the user-requested URI to enable PingFederate to select the proper Access Token Manager when PingAccess makes a validation request to PingFederate.

#### **POST Preservation**

When a user submits a POST form and the authentication session has timed out, the user is redirected to PingFederate in order to re-authenticate. The POST form data is now preserved through this redirection, and the

submitted form is automatically resubmitted after successful authentication. The preserved POST form data can optionally be encrypted in the user's browser with this feature.

### HTTP Header-Based Load Balancing

In addition to the Round Robin load balancing strategy, PingAccess now includes a header-based load balancing strategy. This feature allows an external load balancer in the infrastructure to inject a header (such as `X-Target-Host`) with a value for a hostname or IP address (defined as a target in the Site configuration) where the request should be routed after PingAccess processes it.

### Improvements to HTTP Security Headers Set for PingAccess Admin/Engine/Agent Listeners

PingAccess now includes a number of HTTP Security Headers for connections to the Administrative User Interface, Engine, and Agents. These headers are defined using options in `run.properties`, and additional headers can be added if desired.

## Auditing Enhancements

### Auditing of Unknown Resource Requests

When a client requests a resource that is not known, PingAccess does not record the request in the audit log for performance reasons. In PingAccess 3.2, the optional configuration property `pa.auditing.unknown.resource` can be enabled in order to help troubleshoot resource definition issues.

### Request Tracking

Audit logging now includes a default option to include either a Tracking ID or an Access Token ID along with an Exchange ID to make it easier to follow the audit information for a particular session and exchange in the log files.

## Administrative Enhancements

### JSON-Based Export/Import of PingAccess Configuration

The existing **Backup** section for the administrative user interface has been replaced with a JSON-based export/import capability, making it simpler to duplicate an environment for development purposes. The JSON-based import functionality also simplifies the procedure to restore an environment's backup by allowing the administrator to use the administrative user interface.

### Support for Wildcards in Virtual Hosts

Earlier releases of PingAccess permitted the use of a wildcard virtual host that would match any host (for example, `*:443`), but wildcards used in conjunction with an existing domain name were not supported. This feature adds the ability to specify a wildcard host with a domain (for example, `*.example.com:443`).

### Usability Enhancements

Several administrative interface usability enhancements are in this release, including:

- Moved Options to Advanced Sections on Some Configuration Pages
- Restructured the Settings Page
- Added Pagination, List View, and Filtering Options to Improve Administrative UI Scalability
- Test Connections to Sites and PingFederate When Saving Related Configuration
- Added Cluster Status Display Information to Clustering Configuration Page

### Administrative API Name-Based Search

When using the Administrative API at `/pa-admin-api/v1/` to retrieve information about named objects (for example, Agents or Engines), a `name` parameter can be specified to retrieve the object with the exact name specified.

## 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 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**

- 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**

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

**PingAccess 2.0 - September 2013**

- Initial General Availability (GA) release

**PingAccess 1.0 - April 2013**

- Limited release