PingAccess®

Version 3.0

Deployment Guide

Copyright

PingAccess

Version 3.0
July, 2014

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

Trademark

Disclaimer

Document Lifetime

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

From the web site, you may also download and refresh this PDF if it has been updated, as indicated by a change on this date: **July, 2014**.

1. Deployment Guide . . . 2
1.1 Use Cases and Deployment Architecture . . . 2
1.1.1 Deploying for Gateway API Access Management . . . 2
1.1.1.1 API Access Management POC Deployment Architecture . . . 3
1.1.1.2 API Access Management Production Deployment Architecture . . . 4
1.1.2 Deploying for Gateway Web Access Management . . . 4
1.1.2.1 WAM Gateway POC Deployment Architecture . . . 5
1.1.2.2 WAM Gateway Production Deployment Architecture . . . 6
1.1.3 Deploying for Agent Web Access Management . . . 6
1.1.3.1 WAM Agent POC Deployment Architecture . . . 7
1.1.3.2 WAM Agent Production Deployment Architecture . . . 8
1.1.4 Deploying for Auditing and Proxying . . . 9
1.1.4.1 Audit and Proxy POC Deployment Architecture . . . 10
1.1.4.2 Audit and Proxy Production Deployment Architecture . . . 10
1.2 Port Requirements . . . 11
1.3 Performance Tuning . . . 12
1.3.1 Java Tuning . . . 12
1.3.2 Garbage Collector Configuration . . . 13
1.3.3 Resource Pools . . . 13
1.3.4 Logging and Auditing . . . 15
1.3.5 Agent Tuning . . . 15
1.4 Server Clustering . . . 15

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, to port requirements and clustering options. This section provides information to help you make the right decisions for your environment.

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

Use Cases and Deployment Architecture

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.

As of PingAccess 3.0 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 forward 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 so called *agent request* contains just the relevant parts of the client request so that PingAccess Policy Server can make the access control decision and respond with instructions to the agent regarding any modifications to the original client request that the agent should perform prior to forwarding the request. For example, the agent may add headers and tokens required by the target resource. Under the PingAccess Policy Server's control, the agent may perform a certain amount of caching of information in order to minimize the overhead of contacting the PingAccess Policy Server, thus maximizing response time.

In both gateway and agent deployment the response from the target resource is processed on the way to the original client. In an agent deployment, the amount of processing is more limited than in a gateway deployment. The agent does not make another request to the Policy Server, so response processing is based on the initial agent response. Consequently, the agent is not able to apply the request processing rules available to the gateway.

When designing a deployment architecture, many requirements and components must be identified for a successful implementation. Proper network configuration of routers/firewalls and DNS ensure that all traffic is routed through PingAccess for the Resources it is protecting and that alternative paths (for example, backdoors) are not available.

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

Deploying for Gateway API Access Management
Deploying for Gateway Web Access Management
Deploying for Agent Web Access Management
Deploying for Auditing and Proxying

Deploying for Gateway API Access Management

Deploying for Gateway API Access Management

A PingAccess API access management deployment enables an organization to quickly set up an environment that provides a secure method of controlling access to APIs while integrating with existing identity management infrastructure. Pressure from an ever-expanding mobile device and API economy can lead developers to hastily design and expose API’s outside the network perimeter. Standardized API access management leads to a more consistent, centrally-controlled model that ensures existing infrastructure and security policies are followed, thereby safeguarding an organization’s assets.

PingAccess Gateway sits at the perimeter of a protected network between mobile, in-browser, or server-based client applications and protected APIs and performs the following actions:

- Receives inbound API calls requesting protected applications. OAuth-protected API calls contain previously-obtained access tokens retrieved from PingFederate acting as an OAuth Authorization Server.
- Evaluates application and resource-level policies and validates access tokens in conjunction with PingFederate.
- Acquires the appropriate target site security token (Site Authenticators) from the PingFederate STS or from a cache (including attributes and authorized scopes) should an API require identity mediation.
- Makes authorized requests to the APIs and responses are received and processed.
- Relays the responses on to the clients.

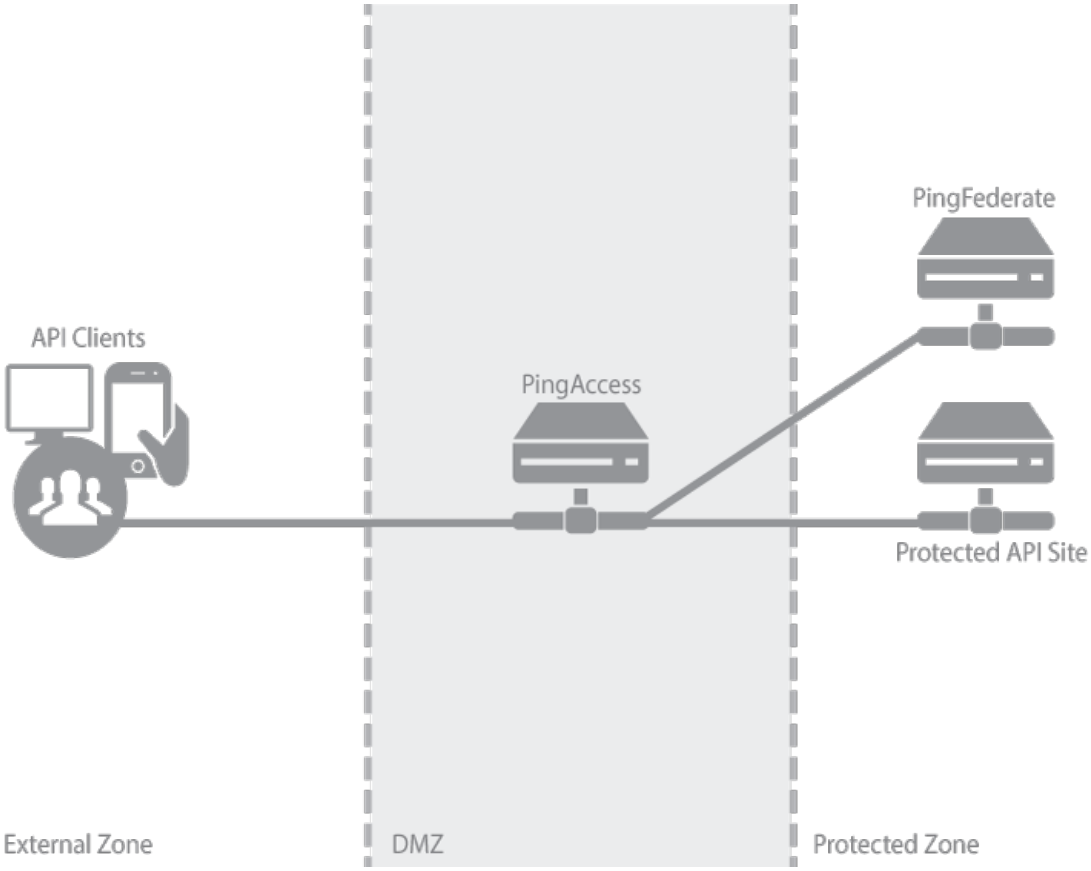
The following sections describe sample Proof of Concept and Production architectures for an API access management use case deployment.

- API Access Management POC Deployment Architecture
- API Access Management Production Deployment Architecture

API Access Management POC Deployment Architecture

API Access Management Proof of Concept Deployment Architecture

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



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

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

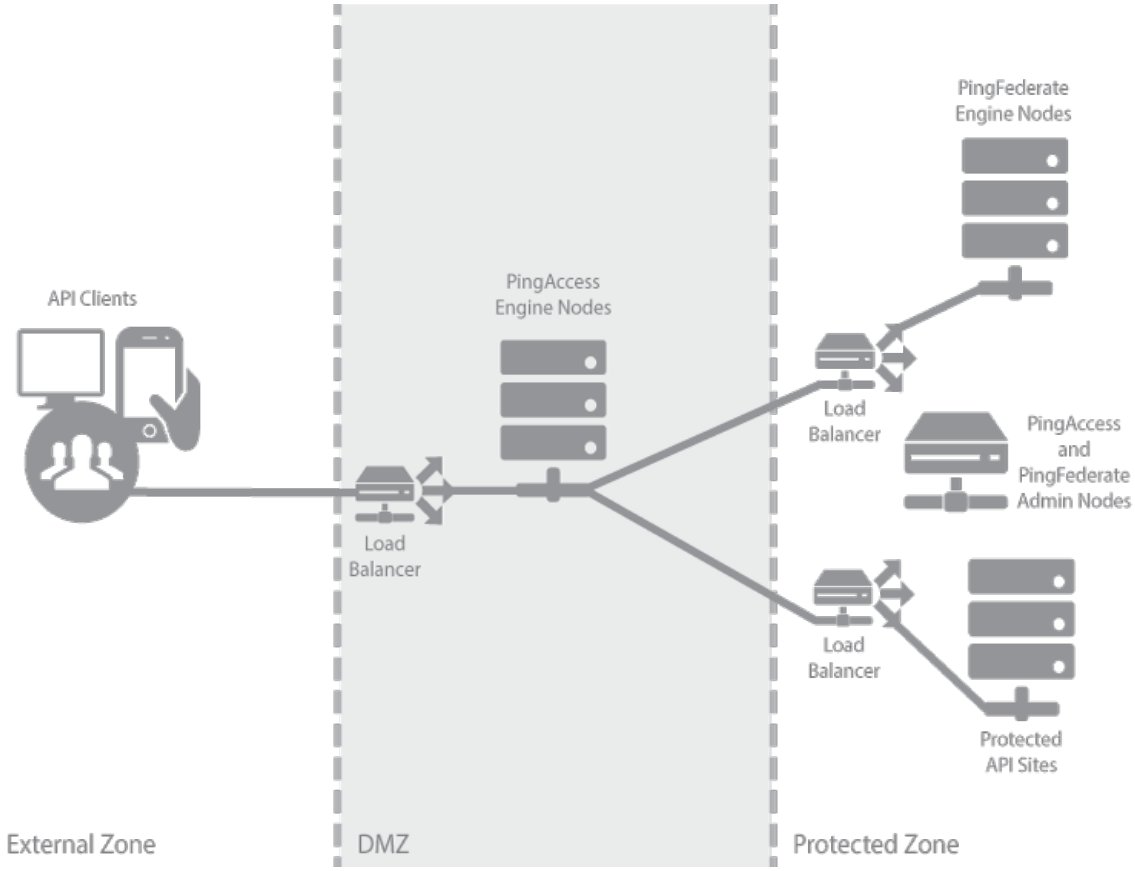
Protected Zone	Back-end controlled zone in which Sites hosting the protected APIs are located. All requests to these APIs must be designed to pass through PingAccess. PingFederate is accessible to API clients in this zone and is a standalone instance, serving as both a runtime and an administrative port.

API Access Management Production Deployment Architecture

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.

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.

Zone	Description
External Zone	External network where incoming API requests originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to API clients. A minimum of two PingAccess engine nodes will be deployed in the DMZ to achieve high availability. Depending on your scalability requirements, more nodes may be required.
Protected Zone	Back-end controlled zone in which Sites hosting the protected APIs are located. All requests to these APIs must be designed to pass through PingAccess. PingFederate is accessible to API clients in this zone. A minimum of two PingFederate engine nodes will be deployed in the protected zone. Administrative nodes for both PingAccess and PingFederate may be co-located on a single machine to reduce hardware requirements

Deploying for Gateway Web Access Management

Deploying for Gateway Web Access Management

A PingAccess Web access management (WAM) deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to Web-based applications while integrating with existing identity management infrastructure. With growing numbers of internal and external users, and more and more enterprise resources available online, it is important to ensure that qualified users can access only those applications to which they have permission. A WAM environment provides authentication and policy-based access management while integrating with existing infrastructure.

Deployed at the perimeter of a protected network between browsers and protected Web-based applications, PingAccess Gateway performs the following actions:

- Receives inbound calls requesting access to Web applications. Web Session protected requests contain a previously-obtained PA token in a cookie derived 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.

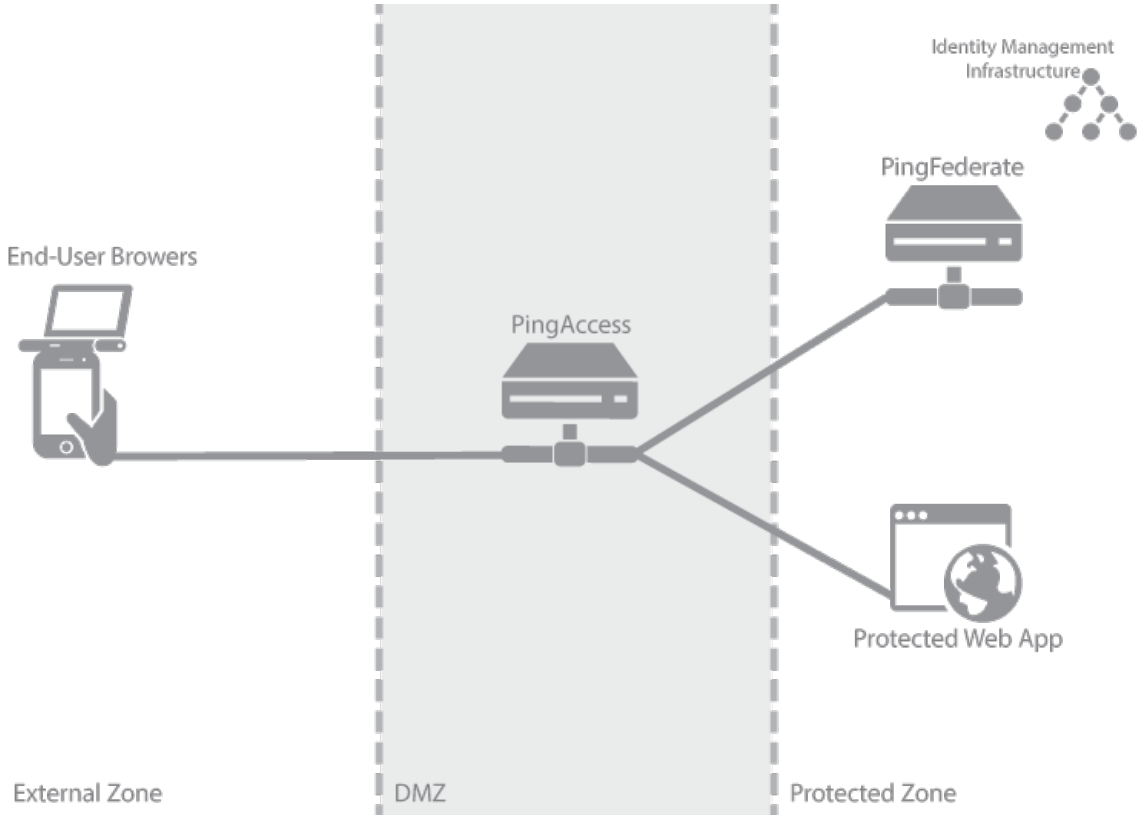
The following sections describe sample Proof of Concept and Production architectures for a WAM use case deployment.

- WAM Gateway POC Deployment Architecture
- WAM Gateway Production Deployment Architecture

WAM Gateway POC Deployment Architecture

Web Access Management Gateway Proof Of Concept Deployment Architecture

This environment is used to emulate the Production environment for testing purposes. In the test environment, PingAccess can be set up with the minimum hardware requirements. This environment example does not provide high availability and is not recommended for a Production environment.



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

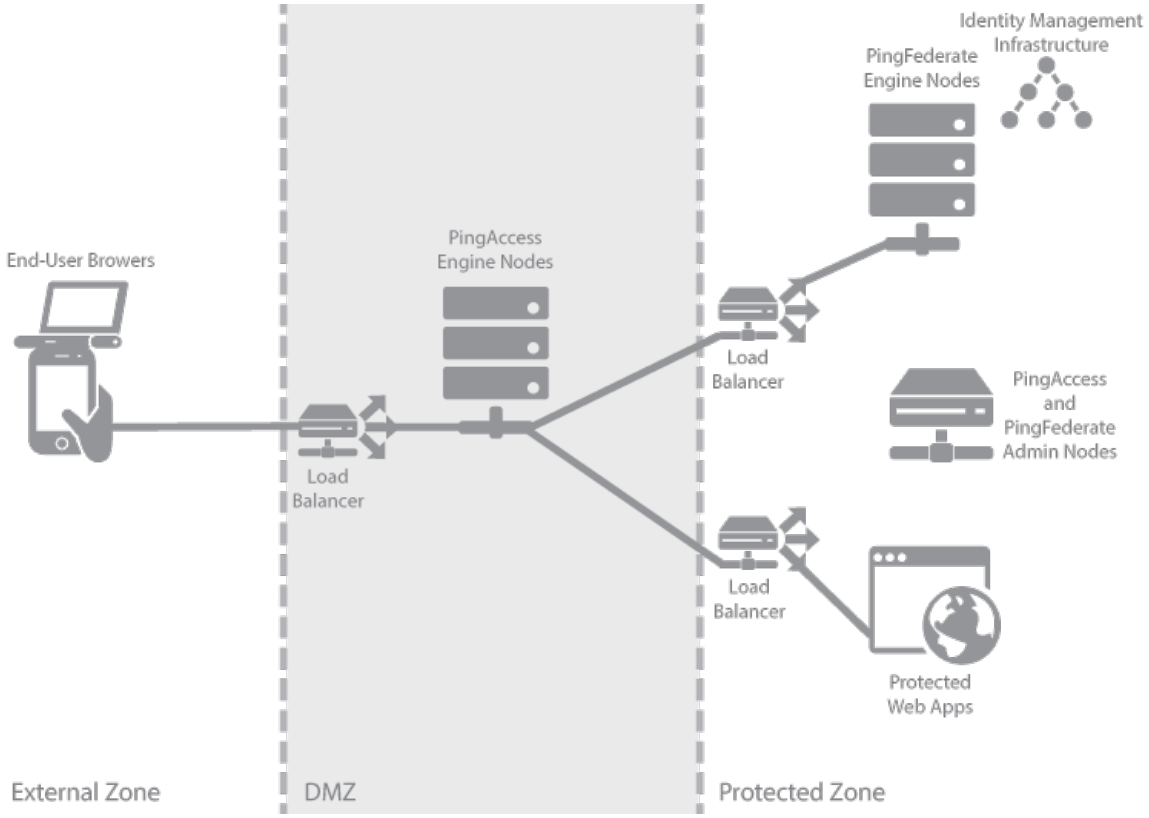
Zone	Description

External Zone	External network where incoming requests for Web applications originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to Web browsers. PingAccess is a standalone instance in this environment, serving as both a runtime and an administrative port.
Protected Zone	Back-end controlled zone in which Sites hosting the protected Web applications are located. All requests to these Web applications must be designed to pass through PingAccess. PingFederate is accessible to Web browsers in this zone and is a standalone instance in this environment, serving as both a runtime and an administrative port. PingFederate requires access to identity management infrastructure in order to authenticate users (depicted by the icon in the diagram).

WAM Gateway Production Deployment Architecture

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.



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

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to Web browsers. A minimum of two PingAccess engine nodes will be deployed in the DMZ to achieve high availability. Depending on your scalability requirements, more nodes may be required.
Protected Zone	Back-end controlled zone in which Sites hosting the protected Web applications are located. All requests to these Web applications must be designed to pass through PingAccess. PingFederate is accessible to Web browsers in this zone and requires access to identity management infrastructure in order to authenticate users (depicted by the icon in the diagram). A minimum of two PingFederate engine nodes will be deployed in the protected zone. Administrative nodes for both PingAccess and PingFederate may be co-located on a single machine to reduce hardware requirements.

Deploying for Agent Web Access Management

Deploying for Agent Web Access Management

A PingAccess Web access management (WAM) agent deployment enables an organization to quickly set up an environment that provides a secure method of managing access rights to Web-based applications while integrating with existing identity management infrastructure and minimal network configuration changes. With growing numbers of internal and external users, and more and more enterprise resources available online, it is important to ensure that qualified users can access only those applications to which they have permission. A WAM environment provides authentication and policy-based access management while integrating with existing infrastructure.

The PingAccess Agent plugin is installed on the Web server hosting the protected Web-based applications and configured to communicate with PingAccess Server also deployed on the network. When the agent intercepts a client request to a protected Web application resource it performs the following actions:

- Intercepts inbound requests to Web applications.
- Sends agent requests to the PingAccess Policy Server sending along relevant request information needed by Policy Server.
- Receives agent responses from Policy Server and follows the instructions from Policy Server, modifies the request as specified, and allows the request to proceed to the target resource.
- Intercepts responses from the application and modifies response headers as instructed in the initial agent request to Policy Server.
- Relays responses on to the browsers.

The PingAccess Policy Server listens for agent requests and performs the following actions:

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

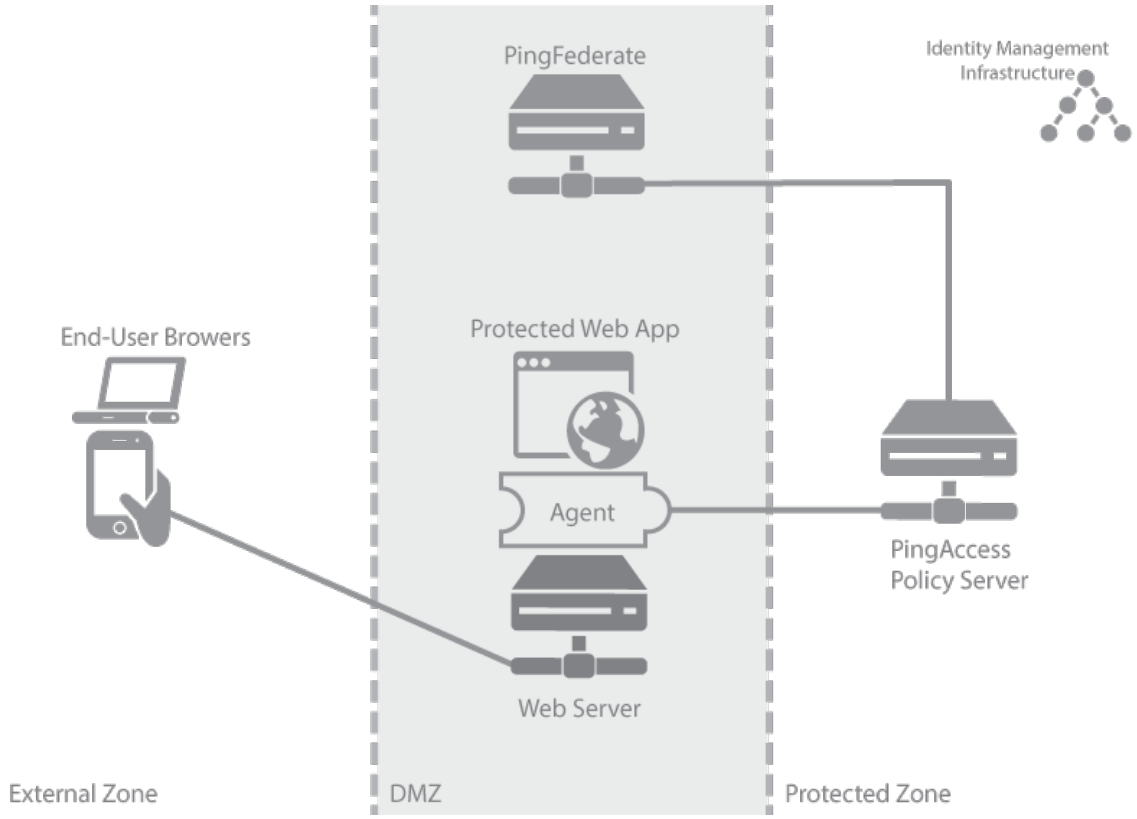
The following sections describe sample Proof of Concept and Production architectures for a WAM use case deployment.

- WAM Agent POC Deployment Architecture
- WAM Agent Production Deployment Architecture

WAM Agent POC Deployment Architecture

Web Access Management Agent Proof Of Concept Deployment Architecture

This environment is used to emulate the Production environment for testing purposes. In the test environment, PingAccess can be set up with the minimum hardware requirements. This environment example does not provide high availability and is not recommended for a Production environment.



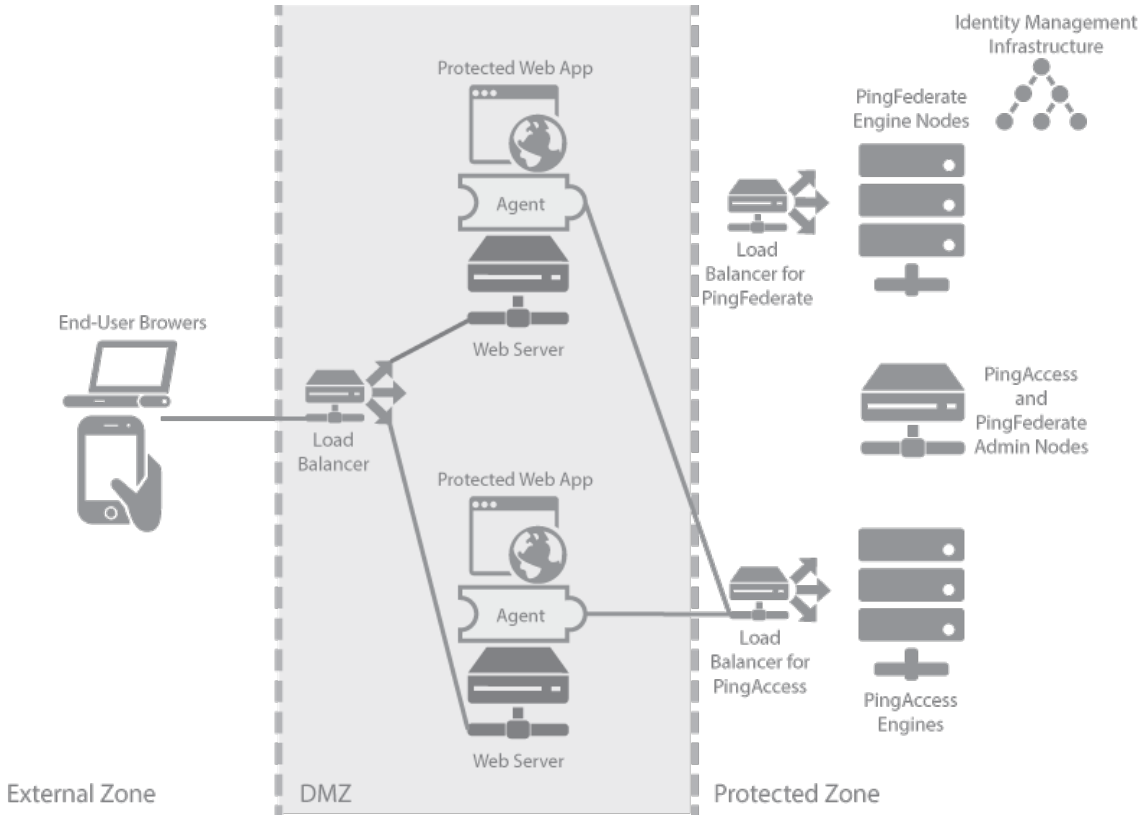
The following table describes the three zones within this proposed architecture.

Zone	Description
External Zone	External network where incoming requests for Web applications originate.
DMZ Zone	Externally exposed segment where application Web server is accessible to Web clients. PingAccess Agent is deployed as a plugin on this Web server. The agent interacts with PingAccess Policy Server in the Protected Zone. PingFederate is deployed as a standalone instance in this environment because during user authentication clients interact with PingFederate. PingFederate requires access to Identity Management Infrastructure in order to authenticate users.
Protected Zone	Back-end controlled zone with no direct access by Web clients. PingAccess Policy Server is deployed in this zone. PingAccess interacts with PingFederate in the DMZ Zone. Identity Management Infrastructure is deployed in this zone.

WAM Agent Production Deployment Architecture

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 Zone	Externally exposed segment where (possibly multiple) application Web servers are accessible to Web clients. PingAccess Agent is deployed as a plugin on these Web servers. Agents interact with PingAccess Policy Server in the Protected Zone.
Protected Zone	Back-end controlled zone with no direct access by Web clients. PingAccess Policy Server is deployed in a cluster in this zone with a separate administrative engine. PingFederate is also deployed in this zone in a cluster with its own separate administrative engine. PingFederate needs access to the Identity Management Infrastructure in order to authenticate users. Since during user authentication Web clients need to interact with PingFederate directly, a reverse proxy such as PingAccess Gateway is required to forward client requests through the DMZ. This aspect is not shown in the diagram.

Deploying for Auditing and Proxying

Deploying for Auditing and Proxying

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

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

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

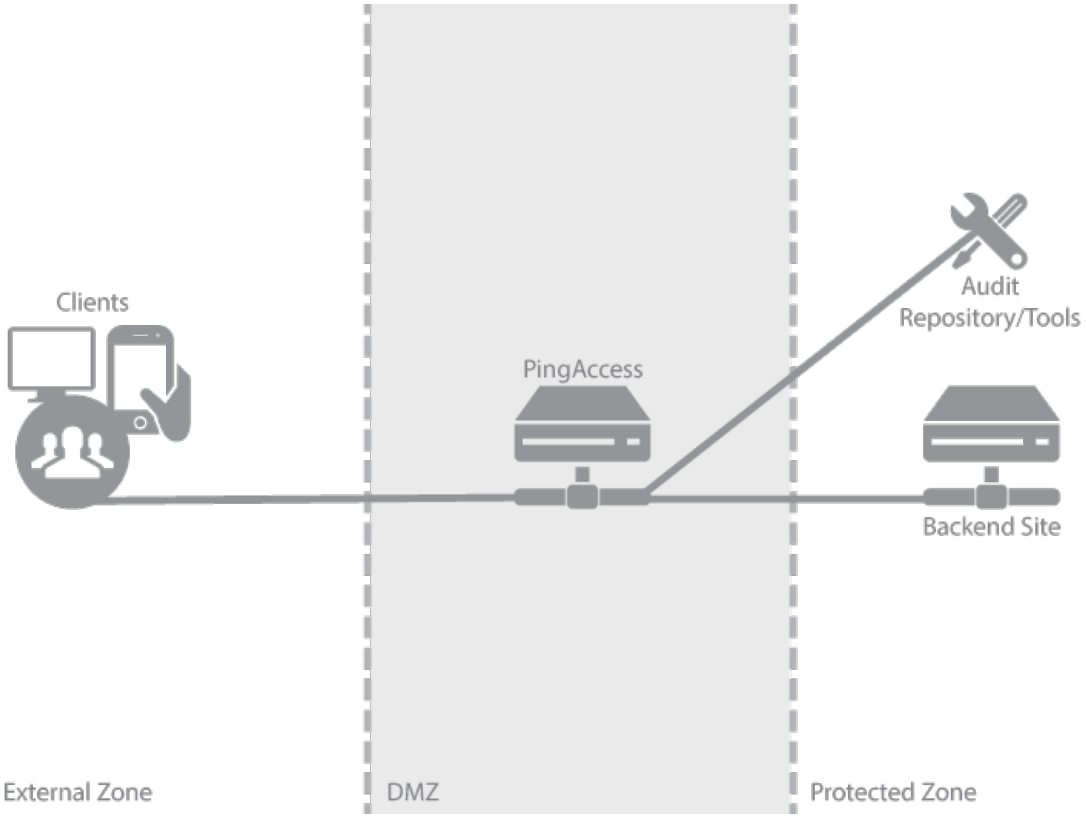
The following sections describe sample Proof of Concept and Production architectures for an auditing/proxying use case deployment.

Audit and Proxy POC Deployment Architecture
Audit and Proxy Production Deployment Architecture

Audit and Proxy POC Deployment Architecture

Auditing and Proxying Proof of Concept Deployment Architecture

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



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

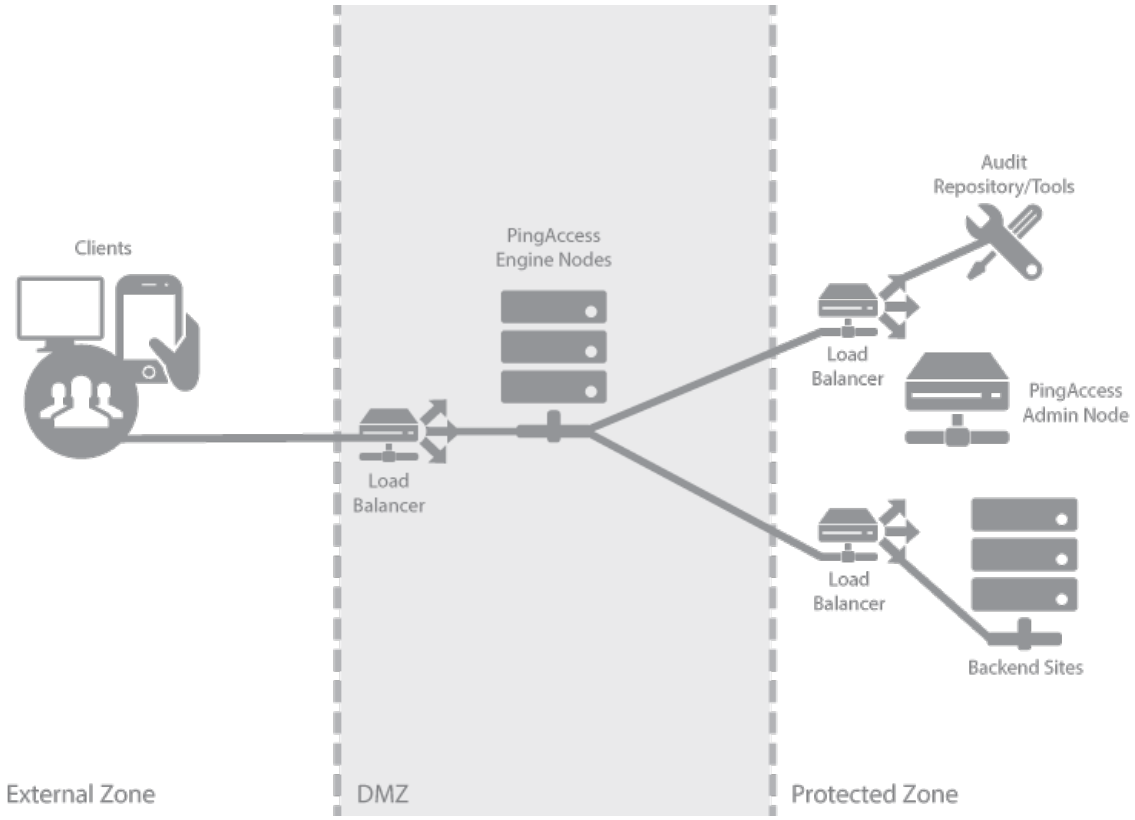
Zone	Description
External Zone	External network where incoming requests originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to clients. PingFederate and PingAccess are standalone instances in this environment, serving as both runtime and administrative ports.
Protected Zone	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.

Audit and Proxy Production Deployment Architecture

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.

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.

Zone	Description
External Zone	External network where incoming requests originate.
DMZ Zone	Externally exposing segment where PingAccess is accessible to clients. A minimum of two PingAccess engine nodes will be deployed in the DMZ. Depending on your scalability requirements, more nodes may be required.
Protected Zone	Contains back-end Sites audited and proxied through PingAccess. Audit results are sent to an audit repository or digested by reporting tools. Many types of audit repository tools are supported such as SIEM/GRC, Splunk, database, and flat files.

Port Requirements

Port Requirements

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

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

Service (Type of Traffic)	Protocol	TCP/UDP	Default Port	Source	Destination	Direction	Description
PingAccess Administrative Console	HTTPS	TCP	9000	PingAccess Administrator browser	PingAccess Administration Engine	Inbound	Used for incoming requests to the PingAccess administrative console. Configurable using the `admin.port` property in the run.properties file.

PingAccess Engine	HTTPS	TCP	3000	Client Browser, Mobile Devices, PingFederate Engine	PingAccess Engine	Inbound	Used for incoming requests to the PingAccess runtime engine. Configurable using the `engine.http.port` property in the run.properties file.
PingAccess Agent	HTTP	TCP	3030	PingAccess Agent	PingAccess Engine	Inbound	Used for incoming Agent requests to the PingAccess runtime engine. Configurable using the `agent.http.port` property of the run.properties file.
PingFederate Traffic	HTTPS	TCP	9031	PingAccess Engine	PingFederate	Outbound	Used to validate OAuth Access Tokens, ID Tokens, make STS calls for Identity Mediation, and return authorized information about a user. Configurable using the PingFederate Settings page within PingAccess.
PingAccess Cluster Traffic	JGroups	TCP	7600	PingAccess Engine	PingAccess Engine	Inbound	Used for communications between engine nodes in a cluster. Configurable using the run.properties file.
PingAccess Cluster Traffic	JGroups	TCP	7700	PingAccess Engine	PingAccess Engine	Inbound	Used by other nodes in the cluster as part of the cluster's failure-detection mechanism. Configurable using the run.properties file.
PingAccess Cluster Traffic	JGroups	UDP	7601	PingAccess Engine	PingAccess Engine	Inbound	Used by other nodes in the same cluster to share information. Configurable using the run.properties file.

Performance Tuning

Performance Tuning

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

> An additional document related to performance, the PingAccess Capacity Planning Guide, is also available to customers as a performance data reference. This document is available from the Customer Portal (https://www.pingidentity.com/support/customer-portal.cfm).

Java Tuning
Garbage Collector Configuration
Resource Pools
Logging and Auditing

Java Tuning

Java Tuning

Heap (How much memory)

One of the most important tuning options you can apply to the Java Virtual Machine (JVM) is to configure how much heap (memory for runtime objects) to use. The JVM grows the heap from a specified minimum to a specified maximum. If you have sufficient memory, best practice is to “fix” the size of the heap by setting minimum and maximum to the same value. This allows the JVM to reserve its entire heap at startup, optimizing organization and eliminating potentially expensive resizing.

By default, PingAccess fixes the Java heap at 512 megabytes (MB). This is a fairly small footprint and not optimal for supporting higher concurrent user loads over extended periods of activity. If you expect your deployment of PingAccess to serve more than 50 concurrent users (per PingAccess node if deploying a cluster), we recommend that you increase the heap size.

Modifying Heap Size

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

1. Edit the run script in the bin directory of the PingAccess install: run.sh on Linux or run.bat on Windows
2. Specify overall heap size by modifying the MINIMUM_HEAP and MAXIMUM_HEAP variables:

- Edit -Xms512m and -Xmx512m respectively
- Specify units as m (megabytes) or g (gigabytes)

3. Specify young generation size by modifying the MINIMUM_NEW and MAXIMUM_NEW variables:
   - Edit -XX:NewSize=256m and -XX:MaxNewSize=256m, respectively
   - Set values to 50% of MINIMUM_HEAP and MAXIMUM_HEAP, respectively

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

Windows Service

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

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

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

Linux Service

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

Garbage Collector Configuration

Garbage Collector Configuration

Selecting the appropriate garbage collector depends on the size of the heap and available CPU resources. The following is a table of available collectors and some general guidance on when and how to use them.

Garbage Collector	Description	Modifications
Parallel	-Best used with heaps 4GB or less -Full stop-the-world copying and compacting collector -Uses all available CPUs (by default) for garbage collection	Default collector for server JVM. No modification is required to the run.sh/run.bat scripts or the Windows Service configuration file or use.
Concurrent Mark Sweep (CMS)	-Best for heaps larger than 4GB with at least 8 CPU cores -Mostly a concurrent collector -Some stop-the-world phases -Non-Compacting -Can experience expensive, single threaded, full collections due to heap fragmentation	**Run.sh/run.bat scripts**: Set GARBAGE_COLLECTOR variable to **-XX:+UseConcMarkSweepGC** in the run script. **Note**: Quote delimiters required in run.sh, not run.bat. **Windows Service**: Set wrapper.java.additional.10 to **-XX:+UseConcMarkSweepGC** in the PingAccessService.conf file.
Garbage First (G1)	-Best for heaps larger than 6GB with at least 8 CPU cores -Combination concurrent and parallel collector with small stop-the-world phases -Long-term replacement for CMS collector (does not suffer heap fragmentation like CMS)	**Run.sh/run.bat scripts**: Set GARBAGE_COLLECTOR variable to **-XX:+UseG1GC** in the run script. **Note**: Quote delimiters required in run.sh, not run.bat. Also disable MINIMUM_NEW and MAXIMUM_NEW tuning. Explicit sizing adversely affects pause time goal. To disable, precede variables with **rem** in run.bat, # in run.sh. **Windows Service**: Set wrapper.java.additional.10 to **-XX:+UseG1GC** in the PingAccessService.conf file. Also disable wrapper.java.additional.11 and wrapper.java.additional.12. Explicit sizing adversely affects pause time goal. To disable, precede lines with #.

Resource Pools

Resource Pools

Acceptor Threads

PingAccess uses a pool of threads to respond to HTTP/S requests made to the TCP port(s) in use. This applies to both administrative and runtime engine listening ports. Acceptor threads read user requests from the administrative or runtime port and pass the requests to worker threads for processing. A best practice is to use at least two acceptors for performance. On larger multiple CPU core machines, more acceptors can be used. We recommend limiting to between two and 1/4th the number of available CPU cores.

To modify, open the `run.properties` file located in the `conf` directory of your PingAccess deployment and specify the number of acceptors you want to use on the following lines:

```
admin.acceptors=N
engine.http.acceptors=N
agent.http.acceptors=N
```

Where N represents the number of acceptor threads.

Worker Threads

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

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

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

and

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

Where `N` represents the number of worker threads.

Maintenance of the pool is such that if the number of threads in the pool exceeds the value of `engine.httptransport.coreThreadPoolSize`, threads idle for 60 seconds are terminated and removed from the pool. The idle timeout value is not modifiable. However, if the values of `engine.httptransport.coreThreadPoolSize` and `engine.httptransport.maxThreadPoolSize` are the same, a fixed sized pool is created and idle threads are not terminated and removed. Similarly for `agent.httptransport.coreThreadPoolSize` and `agent.httptransport.maxThreadPoolSize`.

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

Backend Server Connections

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

Max Connections

Connections to PingAccess are not explicitly connections to the proxied site. PingAccess creates a pool of connections, unlimited in size by default, that are multiplexed to fulfill client requests. Maintenance of the pool includes creating connections to the site when needed (if none are available) and removing connections when the **Keep Alive Timeout** is reached (see Keep Alive Timeout for more details).

In certain situations it can be advantageous to limit the number of connections in the pool for a given Web site. If, for example, the Web site is limited to the number of concurrent connections it can handle or has specific HTTP Keep Alive settings, limiting the number of connections from PingAccess can improve overall performance by not overloading the backend server. In the event that all connections in the pool are in use, a requesting thread waits for one to become available. Assuming that response time from the backend site is sufficiently fast, the time spent waiting for a connection is likely to be less than if the system becomes overloaded.

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

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

Keep Alive Timeout

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

Configuring PingAccess to timeout the connections before the proxied server ensures that use of “stale” connections to the Site is not attempted, causing failure and retry overhead. To improve efficiency, keep the timeout value of PingAccess connections as close as possible to the timeout value of the proxied server without matching or going over that value. This depends on the time granularity afforded by the proxied HTTP server's configuration (time set in minutes, seconds, milliseconds, etc.) and may take some testing to fully optimize. As a starting point, we suggest 500 milliseconds (half a second) to one second as PingAccess transactions typically complete in less than a half a second on a properly-sized deployment. See Sites for information on setting **Keep Alive Timeout**.

Logging and Auditing

Logging and Auditing

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

Logging

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

Auditing

As with logging, auditing is provided by the same high performance, asynchronous logging framework. Furthermore, auditing messages can be written to a database instead of flat files, decreasing file I/O. If you do not require auditing for interactions with a Resource or between PingAccess and PingFederate, it is more efficient to disable audit logging. However, if you do require auditing services and have access to a Relational Database Management System (RDBMS), we recommend auditing to the database. You will see a decrease in disk I/O, which may result in increased performance depending on database resources.

Agent Tuning

Agent Tuning

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

Max Connections

Connections from the agent to PingAccess are limited by `agent.engine.configuration.maxConnections`. The default is set to `10`. In certain situations it can be advantageous to increase the number of connections. In the event that all connections in the pool are in use, a requesting thread waits for one to become available. Assuming that response time to PingAccess is sufficiently fast, the time spent waiting for a connection is likely to be less than if the system becomes overloaded. Note that this is the maximum number of connections per worker process, and not simply the total number of workers the agent has access to. Setting `agent.engine.configuration.maxConnections` value too low may create a bottleneck to PingAccess, and setting the value too high may cause PingAccess to become overloaded.

Max Tokens

By default, the maximum number of cached tokens in an agent is unlimited. In certain situations it can be advantageous to limit the size of the cache for the agent, as a smaller cache has a smaller memory footprint, freeing up memory available to the application for servicing requests. However, when the token cache limit is reached, the least recently used token-policy mapping will be removed from the cache. If that token-policy mapping happens to be needed again, the agent will have a cache miss, resulting in the need to obtain a new token-policy mapping from PingAccess.

Server Clustering

Server Clustering

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

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

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

Set up Your Cluster for High Availability

The following lists the steps required to facilitate high availability of critical services and increase performance and overall system throughput with your cluster. For detailed step information, see Configure PingAccess Servers into a Cluster.

1. Install PingAccess on multiple machines.
2. Edit the run.properties file accordingly for the cluster node type.
3. Optional. Edit run.properties on each engine node for sub-clustering.
4. Optional. Create a new Key Pair and self-signed certificate or import an existing secure certificate.
5. Assign the new certificate to the administrative console.
6. Register cluster nodes in the administrative console.
7. Start or restart each node.