PingFederate®

Server Clustering Guide

PingIdentity®

Part Number 3007-354
Version 6.0
April, 2009
Ping Identity Corporation
1099 18th Street, Suite 2950
Denver, CO 80202
U.S.A.

Phone: 877.898.2905 (+1 303.468.2882 outside North America)
Fax: 303.468.2909
Web Site: http://www.pingidentity.com

Trademarks

Disclaimer

Contents

Introduction

PingFederate provides clustering features that allow a group of PingFederate servers to appear to browsers and partner federation servers as a single system. In this configuration, all client traffic normally goes though a load balancer, which routes requests to the PingFederate servers in the cluster. User-session states and configuration data are shared among the servers, enabling them to process requests as a single entity.

> **Note**: PingFederate provides separate failover capabilities specifically for Software-as-a-Service (SaaS) provisioning, which by itself does not require either load balancing or state management (see "Deploying SaaS-Provisioning Failover" on page 18).

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. Some of these choices are identified throughout this *Guide*.

General Architecture

PingFederate abstracts its runtime session-state management behind Java service interfaces. This enables PingFederate to use interface implementations without regard to underlying storage and sharing mechanisms. The abstraction also provides a well-defined point of extensibility in PingFederate (see "Runtime State-Management Services" on page 13).

Group RPC-Oriented Approach

The prepackaged state-management implementations are designed to accommodate a variety of deployments. The implementations leverage a remote-procedure-call (RPC) framework for reliable group communication, allowing PingFederate servers within a cluster to share state information.

Load Balancing

Clustered deployments of PingFederate for single sign-on (SSO) and logout transactions require the use of at least one load balancer, fronting multiple PingFederate servers.

When a client accesses the load balancer's virtual IP, the balancer distributes the request to one of the PingFederate servers in the cluster. The method that the balancer uses to select the appropriate server can vary from simple to highly complex, depending on deployment requirements. Specific balancing strategies, their strengths and weaknesses, as well as the impacts on PingFederate are discussed later (see "Managing Deployment Architecture" on page 8).

The PingFederate software distribution does not contain a load balancer. Numerous hardware or software products are available commercially, including free downloads.

> **Tip**: A load balancer is not required for setting up multiple servers to handle only SaaS-provisioning failover (see "Deploying SaaS-Provisioning Failover" on page 18).

Load balancers may incorporate SSL/TLS accelerators or work closely with them. Due to the high computational overhead of the SSL handshake, Ping Identity recommends terminating SSL/TLS on a

dedicated server external to PingFederate for deployments in which performance is a concern. You can still use SSL between the proxy or balancer and PingFederate, but as a separate connection.

Server Modes

In a cluster, you can configure each PingFederate instance, or *node*, as either an administrative console or a runtime engine. Runtime engines service federated-identity protocol requests, while the console server administers policy and configuration for the entire cluster (via the administrative console). A cluster may contain one or more runtime nodes but only one console node (see the next section).

Deploying Clustered Servers

Follow the steps below to configure and deploy clustered PingFederate servers.

> **Note**: Additional steps are required to set up failover for SaaS provisioning. Alternatively, if you are grouping servers *exclusively* to provide for provisioning failover, skip this section and refer to information under "Deploying SaaS-Provisioning Failover" on page 18.

To configure PingFederate servers in a cluster:

1. For each node in a cluster, follow the PingFederate installation procedures (see the "Installation" chapter in *Getting Started*).

   > **Tip:** You need only install one license file on any one machine in the cluster; the key will be pushed out to the other servers.

2. For each server instance, edit clustering properties in the `run.properties` file located in the `<pf_install>/pingfederate/bin` directory (see the next section, "Configuring Startup Properties").

3. (Optional) Adjust configuration files that control cluster architecture and runtime session-state management (see "Managing Deployment Architecture" on page 8 and "Runtime State-Management Services" on page 13).

4. (Optional) If SaaS provisioning is configured at your site and you want to provide failover capabilities, identify provisioning failover nodes (see "Deploying SaaS-Provisioning Failover" on page 18).

5. Start or restart all of the PingFederate servers.

   Refer to "Starting and Stopping PingFederate," in the PingFederate *Administrator's Manual*, if needed.

   > **Important:** Start the server containing the license file before starting the other servers.

6. After you configure (or reconfigure) PingFederate through the administrative console, replicate the configuration on all nodes in the cluster (see "Synchronizing Server Runtime Configuration" on page 19).

   The *Administrator's Manual* and online Help provide detailed information on using the administrative console.

Configuring Startup Properties

The `run.properties` in the `<pf_install>/pingfederate/bin` directory contains startup clustering configuration options. Configure the properties described in the following table according to your needs for each node in the cluster.

> **Note**: This table does not include information about properties used for SaaS-provisioning failover (see "Deploying SaaS-Provisioning Failover" on page 18).

Property	Description
pf.operational.mode	Controls the operational mode of the PingFederate server. Refer to the properties file for a list of valid values and descriptions. Note that the value `STANDALONE` should not be used in a cluster unless user-state management is not needed for any reason and configuration-archive deployment is used as the configuration synchronization method (see "Synchronizing Server Runtime Configuration" on page 19).
pf.cluster.node.index	Each server in a cluster must have a unique index number, which is used to identify peers and optimize inter-node communication. (Range: 0-65535) If no value is set for the node index, the system assigns a default index derived from the last two octets of the IP address. We recommend, however, that you assign static indexes.
pf.cluster.auth.pwd	Sets the password that each node in the cluster must use to authenticate when joining the group. This prevents unauthorized nodes from joining a cluster. (Value: any string, or blank) All nodes in a cluster must have the same value set for this property. The first node to start up will set the password for the cluster.
pf.cluster.encrypt	Indicates whether or not to encrypt network traffic sent between nodes in a cluster. (Values: true | false [default]) When set to true, communication within the cluster is encrypted with a symmetric key. The oldest member of the cluster generates a symmetric encryption key and distributes it to the other group members. Public/private key encryption is used to protect the initial key exchange. The symmetric key is renegotiated whenever group membership of the cluster changes. All nodes in a cluster must have the same value set for this property.

Property	Description
pf.cluster.bind.address	Controls the network interface to which the group communication should bind. For machines with more than one network interface, you can use this property to increase performance (particularly with UDP) as well as improve security by segmenting group-communication traffic onto a private network or VLAN. If left blank, the first available network interface is used. **Note:** Depending on your network-interface configuration, this field may be required for proper cluster functioning.
pf.cluster.bind.port	(Required) The port associated the bind-address property above.
pf.cluster.transport.protocol	Indicates the transport protocol used for group communication (values: `udp` | `tcp`). Use UDP multicast when IP multicasting is enabled in the network environment and the majority of group traffic is point-to-full-group. You must also provide values for the pf.cluster.mcast* properties described below. Use TCP for geographically dispersed servers or when multicast is not available or disabled for some other reason (routers discard multicast messaging). TCP may also be appropriate if your cluster configuration employs more point-to-point or point-to-few messaging than point-to-group. You must also provide a value for the pf.cluster.tcp.discovery.inital.hosts property described below. **Note**: This property is a reference to a protocol-stack XML configuration file located in the `<pf_install>/pingfederate/server/default/conf/` directory. Two stacks are provided: one for UDP multicast and one for TCP. Administrators can customize either stack or add to it as needed by modifying the associated configuration file.
pf.cluster.mcast.group.address	Defines the IP address shared among nodes in the same cluster for UDP multicast communication; required when UDP is set as the transport protocol. (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 nodes in a cluster must use the same address for this property and the port property below.
pf.cluster.mcast.group.port	Defines the port associated with the property above, pf.cluster.mcast.group.address.

Property	Description
pf.cluster.tcp.discovery.initial.hosts	Designates the initial hosts to be contacted for group membership information when discovering and joining the group; required when TCP is set as the transport protocol. The value is a comma-separated list of host names (or IPs) and ports. Example: `host1[7600],10.0.1.4[7600],host7[1033],10.0.9.45[2231]` Discovering and managing group membership is more difficult using TCP, which does not provide the built-in group semantics of IP multicast. Therefore, at least one of the members of the group must be known in advance and statically configured on each node. It is recommended that as many hosts as possible be included for this property on each cluster node, to increase the likelihood of new members finding and joining the group.

Managing Deployment Architecture

In a cluster consisting of a large number of nodes, it may be desirable to share data with only a subset of nodes (for performance reasons). To facilitate this, most of the group RPC-based service implementations (see "Runtime State-Management Services" on page 13) make use of a *preferred-nodes* concept, which allows each node to have a list of other nodes (identified by index) with which it shares data.

Each service implementation is controlled separately by a configuration file located in the `<pf_install>/pingfederate/server/default/conf` directory. The following table indicates the configuration file that applies to each implementation (refer to the indicated pages in this *Guide* for detailed information about each implementation):

Configuration File	RPC-Based Service Implementation
cluster-artifact.conf	Artifact-Message Persistence and Retrieval Service (see page 16)
cluster-assertion-replay-prevention.conf	Assertion Replay Prevention Service (see page 15)
cluster-idp-session-registry.conf	IdP Session Registry Service (see page 15)
cluster-inter-request-state.conf	Inter-Request State Management Service (see page 13)

Configuration File	RPC-Based Service Implementation
cluster-sp-session-registry.conf	Local Memory-based Tracking – In this alternative, the inter-request state of a user is tracked in the local memory of the processing server. **Note:** To implement this alternative, the load balancer must support sticky sessions to force all requests for the same user to be routed to the same server. To use this service implementation, set the class attribute for the `InterRequestStateMgmt` service in the `hivemodule.xml` file to the class name: `org.sourceid.saml20.service.impl.localmemory.InterReqStateMgmtMapImpl` SP Session Registry Service (see page 14)

The following table describes properties contained in the configuration files (the Artifact-Message Persistence and Retrieval Service uses only `rpc.timeout`):

Property	Description
preferred.node.indices	A comma-separated list of indices identifying the nodes with which this server shares session-state data for the associated service. If blank, all nodes in the cluster are used.
rpc.timeout	How long, in milliseconds, the server waits before timing out unresponsive RPC invocations.
synchronous.retrieve.majority.only	Indicates how many responses to wait for when making synchronous RPC calls (values: `true` | `false`). When `true` (the default), the server waits for the majority of recipients to respond. When `false`, it waits for all recipients to respond.

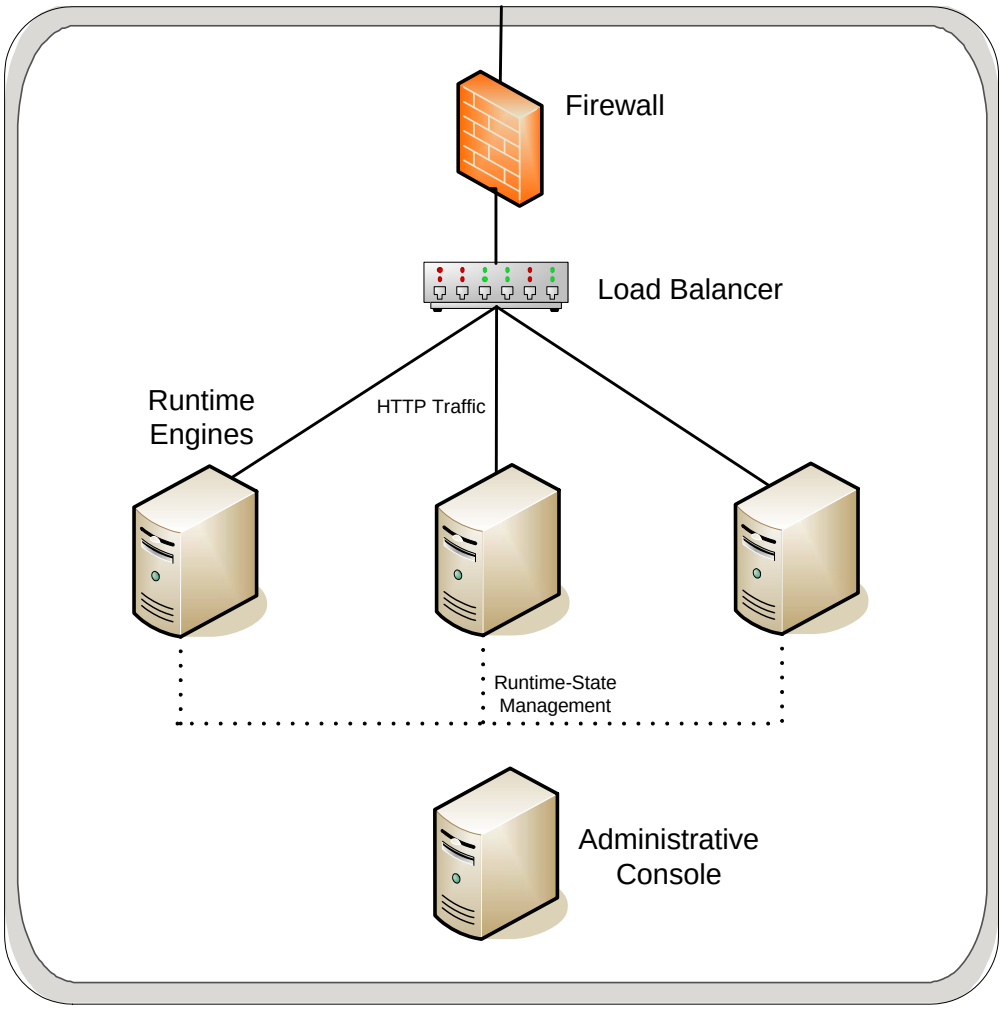
Note that within a single cluster deployment, individual services can use different preferred nodes. In other words, you can set different values for the preferred.node.indices property for each service.

The use of preferred nodes can translate into any number of deployment configurations. Three primary strategies are discussed in the following sections and may serve as touch points for you to consider in conjunction with your network requirements:

- Sharing All Nodes
- Designating State Servers
- Defining Subclusters

Sharing All Nodes

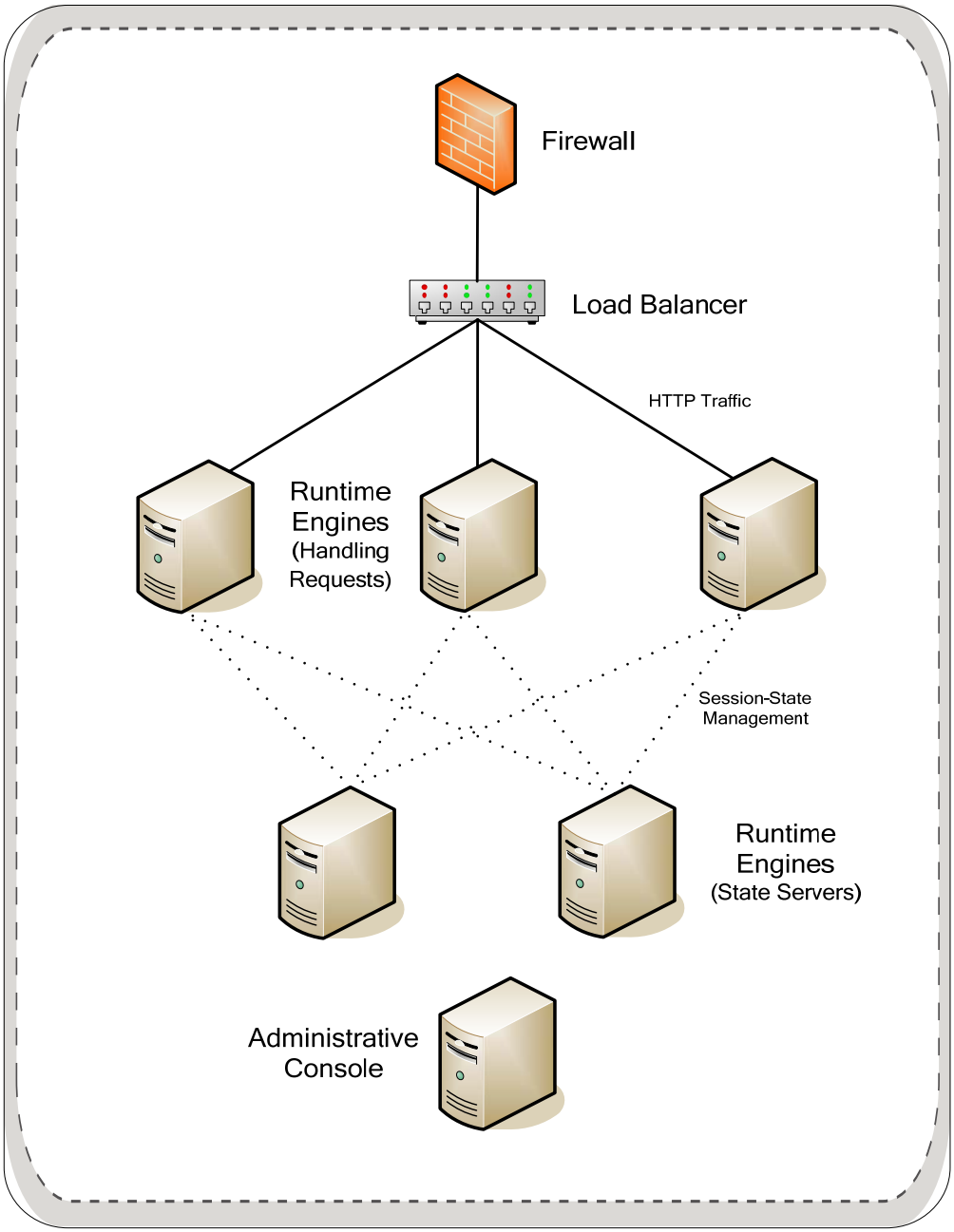
Leaving the preferred.node.indices property blank in all the cluster-configuration files provides a basic deployment case. An advantage of this approach is simplicity, including the option of using straightforward load-balancing strategies such as round robin. A disadvantage is that as additional nodes are added, the throughput improvement rate that clustering offers may decline as the state-replication overhead increases.



The diagram above illustrates the node-sharing approach. HTTP traffic is directed to all nodes.

Designating State Servers

You can select a few nodes as *state servers*. This deployment can be configured by setting the preferred-node indices of other servers in a group to those of the state servers. Load balancers should be configured to isolate the state-server nodes from end-user traffic. This approach scales better than the all-nodes approach because additional nodes do not require connections to every existing node; there need be only a connection between each server and each state server.



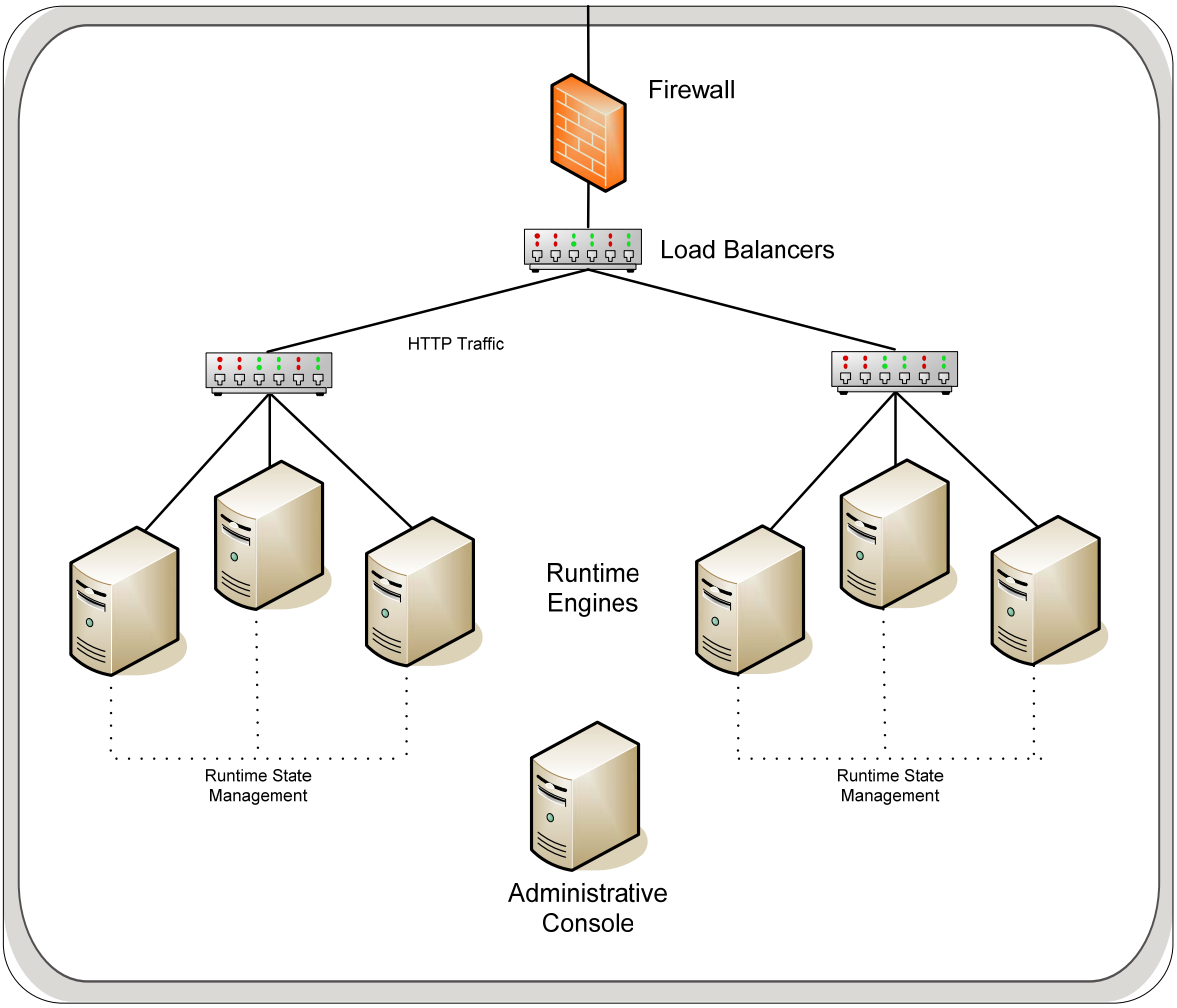
This diagram illustrates the state-server approach. If the two state-server nodes have indices of 1 and 2, then the preferred-nodes property of the other runtime nodes on would be:

```
preferred.node.indices=1,2
```

Defining Subclusters

You can use node indices to divide a cluster into subgroups, or *subclusters*, of a few nodes each. Using this configuration, each node in a subcluster shares data only with other members of the subcluster. This approach requires the load balancer to persist, or *stick*, user sessions to the same group so that each

subsequent request from the same user is directed to one of the nodes in that group. (A wide variety of persistence mechanisms exist and may vary by load balancer – consult the documentation specific to your product.) The advantage of this approach is that cluster throughput scales more linearly, because the creation of an additional subcluster will not degrade the performance of any other group.



The diagram above illustrates the subcluster approach. The preferred-nodes property of each server in the cluster lists the indices of all nodes in its subgroup (including itself). HTTP traffic is directed to all nodes, but the load balancer directs user sessions to the same subcluster.

Runtime State-Management Services

The Runtime State-Management Services are a collection of interfaces defining the contract that PingFederate uses with each service to manage session states. The implementation of each service interface is specified in the `hivemodule.xml` file in the `<pf_install>/pingfederate/server/default/conf/META-INF/` directory. This file does not need to be modified unless you wish to customize the way services are handled in a cluster.

By default, the interfaces listed in `hivemodule.xml` are proxies that select the best implementation for each service, based on the operational mode of the server. For example, if the server is in standalone mode, an in-memory-only implementation is used. If the server is in a clustered mode, a group RPC-based implementation is used. The proxies are provided for convenience; if you choose, you may specifically designate the implementation you desire for each service, as described in the following sections. Configuration files for the implementations that make use of preferred nodes are in the directory `<pf_install>/pingfederate/server/default/conf` (see "Managing Deployment Architecture" on page 8).

Inter-Request State Management Service

The PingFederate server tracks user-session state information between HTTP requests – for example, when PingFederate, acting as an Identity Provider (IdP), redirects a user's browser to another system for authentication. When the user's browser returns to PingFederate after authentication, the server needs access to the state associated with that user from before the redirection. Another example is keeping track of state between issuing a request to a partner and processing the response. Generally, this state is short-lived.

The `InterRequestStateMgmt` interface defines the methods needed to track this state. Three implementations are provided: cookie-based (the default), group RPC-based, and local memory-based.

> **Caution**: Depending on deployments, cookies used for state management in a cluster can become large enough to exceed certain limits. To avoid this limitation, where applicable, use either RPC-based or local memory-based tracking instead.

Cookie-Based Tracking – In a clustered mode, the service proxy uses an implementation that tracks short-lived session-state data with the help of browser cookies. The implementation works by compressing and encrypting the session-state data and sending it to the user's browser as a cookie. User data is then returned to the server on subsequent requests from the browser. No data for this service needs to be shared among nodes, and even the simplest round-robin load-balancing technique works. There is some overhead in the encryption and compression, but the advantage is that the session state must travel over the network only between the entities that actually need it – the user and the server that handles that user's next request.

> **Caution**: Cookie-based tracking, which can result in larger cookies, *should not* be used for configurations using account linking. (For more information about account linking, refer to the "Key Concepts" chapter of the PingFederate *Administrator's Manual*.)

This implementation does not use group RPC to share session-state data. However, RPC is used by default to handle dynamic distribution of the AES keys used for encryption and decryption (unless a static key is configured). In the `key-tracker.xml` file in the same `config-store` directory, you can

control the interval at which new keys are issued, the maximum number of keys to retain, and the key length. The oldest member of a cluster group is responsible for issuing and distributing new keys.

The file `inter-req-cookie-config.xml`, in the `<pf_install>/pingfederate/server/default/data/config-store` directory specifies the name of the cookie, which you may modify as needed. You can also configure a static AES encryption key by specifying its hex value (the key must set on each node). While a static key is somewhat less secure, it eliminates the need for key management between nodes via back-channel communication.

The `class` attribute for `InterRequestStateMgmt` in the `hivemodule.xml` file is set to use this implementation as the default. The class name is:

```
org.sourceid.saml20.service.impl.cookie.InterReqStateMgmtCookieImpl
```

Group RPC-based Session Tracking – A group RPC-based alternative to the cookie-based implementation is provided for situations where sending a significant additional amount of data to and from the user's browser is not appropriate.

The implementation utilizes the preferred-nodes concept; the configuration file is `cluster-inter-request-state.conf` in the `<pf_install>/pingfederate/server/default/conf/`directory. All three preferred-node approaches described under "Managing Deployment Architecture" on page 8 are possible with this implementation. The subgroup approach, however, is probably the most appropriate.

To use this service implementation, set the `class` attribute for the `InterRequestStateMgmt` service in the `hivemodule.xml` file to the class name:

```
org.sourceid.saml20.service.impl.grouprpc.InterRequestStateMgmtGroupRpcImpl
```

Local Memory-based Tracking – In this alternative, the inter-request state of a user is tracked in the local memory of the processing server.

> **Note:** To implement this alternative, the load balancer must support sticky sessions to force all requests for the same user to be routed to the same server.

To use this service implementation, set the class attribute for the `InterRequestStateMgmt` service in the `hivemodule.xml` file to the class name:

```
org.sourceid.saml20.service.impl.localmemory.InterReqStateMgmtMapImpl
```

SP Session Registry Service

PingFederate uses the SP (Service Provider) Session Registry Service to facilitate single logout by tracking assertions issued from IdP partners. This service is used only when the PingFederate server is acting in an SP role and supports single logout with one or more partner connections. (See *Getting Started* for information about federated-identity roles.)

When PingFederate is in clustered mode, the service proxy uses a group RPC-based, preferred-nodes implementation; the configuration file is `cluster-sp-session-registry.conf` in the `<pf_install>/pingfederate/server/default/conf` directory.

All preferred-node approaches are possible with this implementation (see "Managing Deployment Architecture" on page 8). However, if your site accepts single logout messages via SOAP, the subgroup approach may not succeed because the load balancer's sticky-session functionality cannot ensure that the SOAP message from a partner will be directed to the appropriate subgroup.

The service proxy uses the class:

```
org.sourceid.saml20.service.impl.grouprpc.SpSessionRegistryGroupRpcImpl
```

IdP Session Registry Service

PingFederate uses the IdP Session Registry Service to facilitate single logout by tracking assertions issued to SP partners. This service is used only when the PingFederate server is acting in an IdP role and supports single logout with one or more partner connections.

When PingFederate is in clustered mode, the service proxy uses a group RPC-based, preferred-nodes implementation; the configuration file is `cluster-idp-session-registry.conf` in the `<pf_install>/pingfederate/server/default/conf` directory.

As with the SP service, all three preferred-node approaches are possible with this implementation (see "Managing Deployment Architecture" on page 8), but inbound SOAP logout messages may not be handled correctly for subgroups.

The service proxy uses the class:

```
org.sourceid.saml20.service.impl.grouprpc.IdpSessionRegistryGroupRpcImpl
```

LRU Memory Management Schemes – During the normal course of transaction processing, the SP and IdP Session Registries and the Inter-Request State Management service manage memory as part of normal processing. However, it is common for end users to abandon Web sessions, resulting in orphaned data. To ensure that such data does not result in excessive memory usage, the data structures used by these services employ a least-recently-used (LRU) algorithm to purge old data. When a data structure reaches the maximum size, the oldest entries are automatically removed.

The maximum size of each data structure is configurable in the file `size-limits.conf` in the `<pf_install>/pingfederate/server/default/conf` directory.

Assertion Replay Prevention Service

The SAML (Security Assertion Markup Language) standards specify that when an SP receives assertions via the POST binding, the SP should keep track of that assertion for the duration of its validity to ensure that it is not replayed (that is, intercepted by a third party and reposted). PingFederate delegates that responsibility to the Assertion-Replay Prevention Service. This service is used only when PingFederate is acting as an SP and receives assertions via the POST binding (see *Getting Started* for information about SAML bindings).

When PingFederate is in clustered mode, the service proxy uses a group RPC-based, preferred-nodes implementation; the configuration file is `cluster-assertion-replay-prevention.conf` in the `<pf_install>/pingfederate/server/default/conf` directory.

Due to the nature of the threat that this service is intended to prevent, only the "Sharing All Nodes" and "Designating State Servers" deployment strategies are truly appropriate for this service (see after "Managing Deployment Architecture" starting on page 8).

The service proxy uses the class:

```
org.sourceid.saml20.service.impl.grouprpc.
     AssertionReplayPreventionServiceGroupRpcImpl
```

This service is unique in that it fulfills only a security condition of the federation specifications, rather than supporting normal SSO functionality. Because many other security requirements are in place (for example, SSL, no-cache directives, and digital signatures), there may be situations where the priority placed on cluster performance outweighs the priority placed on this security check. If you are in this situation, you may wish to change the implementation for the service point `AssertionReplayPreventionService` in the `hivemodule.xml` file to one of these classes:

- `org.sourceid.saml20.service.impl.localmemory.AssertionReplayPreventionSvcInMemoryImpl`

  This is the implementation used in standalone mode. It performs all the appropriate replay checks but does not share any data with other nodes. A replay attempt routed to the same server node would fail, but other nodes would not have sufficient information to stop the transaction.

- `org.sourceid.saml20.service.impl.localmemory.AssertionReplayPreventionServiceNullImpl`

  This implementation disables assertion-replay prevention; however, you may wish to use it, with caution, when performance is an absolute priority.

Artifact-Message Persistence and Retrieval Service

When the artifact binding is used to send messages to partners, the PingFederate server must keep track of the message referenced by the artifact until the partner makes a SOAP call to retrieve it.

Two options are available for using outbound artifacts in a cluster:

- Group RPC-Based Retrieval
- Using SAML 2.0 Indexing

Group RPC-Based Retrieval – When PingFederate is in clustered mode, the service proxy selects a group RPC-based implementation, which takes advantage of node indexing but does not use the preferred-nodes concept. Sticky-session load-balancing strategies are not effective for the artifact binding, because the requests that result in the issuance of the artifact and the artifact resolution request come from different locations.

This implementation works by having the node that issues an artifact encode its node index into the artifact using two bytes of the artifact message handle. When a server receives the artifact resolution request, it uses the encoded node index to determine the issuing node and makes an RPC call to get the associated message.

Although this implementation does not use the preferred-nodes concept, it does have a configuration file, `<pf_install>/pingfederate/server/default/conf/cluster-artifact.conf`, in which the RPC time-out can be configured.

The service proxy uses the class:

```
org.sourceid.saml20.service.impl.grouprpc.
     ArtifactPersistenceSvcGroupRpcEncodedNodeIdxImpl
```

SAML 2.0 Indexing (Local Memory) – Due to the implementation complexity involved in managing state with the artifact binding, the SAML 2.0 specification introduced the concept of indexed endpoints. A SAML 2.0 federation entity may support multiple artifact resolution services, each identified by a unique index number. Artifacts include this index, and a federation partner must send

the artifact resolution request to the appropriate endpoint for that index. This means that servers do not need to share information concerning the artifact.

The downside to this approach is that partners must know about each of your back-end servers. Generally, this means providing partners with metadata that includes multiple artifact resolution service endpoints with the corresponding indices.

For example, if you have four servers in a cluster, the metadata might look like this:

```
<ArtifactResolutionService Binding="..."
Location="https://node1/idp/ARS.saml2" index="1"/>
<ArtifactResolutionService Binding="..."
Location="https://node2/idp/ARS.saml2" index="2"/>
<ArtifactResolutionService Binding="..."
Location="https://10.0.1.72/idp/ARS.saml2" index="3"/>
<ArtifactResolutionService Binding="..."
Location="https://node4/idp/ARS.saml2" index="4"/>
```

In this case, the index corresponds to the node index configured in the `run.properties` file on each individual server. This service encodes the node index in the artifact handle when running in a clustered mode (it will always use an index of zero in standalone mode).

Not only do partners need to know about each back-end server, they need direct access to each ARS endpoint. This may require more complicated configuration of load balancers, proxies, and firewalls. Another drawback to this approach is that it cannot be used for SAML 1.x, or with adapters that utilize PingFederate's artifact-data management.

To use this approach for SAML 2.0 federation deployments, edit the `hivemodule.xml` file and change the implementation for the `ArtifactStore` service point to the class name:

```
org.sourceid.saml20.service.impl.localmemory.
     ArtifactPersistenceServiceMapImpl
```

Extensibility and Other Services

Custom implementations of all state management services described in this G*uide* can be plugged into PingFederate as needed for special deployments. Software developers can refer to the Javadoc reference:

```
<pf_install>/pingfederate/sdk/doc/index.html
```

Two other services may need consideration when running PingFederate in a cluster, depending on SAML 2.0 federation deployment needs:

Account Linking Service – This service stores the association between the external and internal identifiers of an end user when account linking is used as an SP identity-mapping strategy (see "Key Concepts" in the *Administrator's Manual* for more information). The default, standalone implementation uses a JDBC interface to an embedded database within PingFederate. No information from the embedded database is shared across the cluster. Therefore, when account linking is used for an IdP connection deployed in a cluster, the default implementation will not work properly. In such cases, the pointer must be adjusted for cluster use by pointing the service to an external database. Developers can refer to the Javadoc reference describing the `AccountLinkingServiceDBImpl` class for more information.

Pseudonym Service – This service references the method needed by PingFederate to generate or look up a pseudonym for a user (see "Key Concepts" in the *Administrator's Manual* for more information). The service is used only if your site is acting in an IdP role and produces assertions containing pseudonyms as subject identifiers. The default implementation uses a message digest to produce the value so that no session-state synchronization is required. However, it may be desirable in some situations to implement pseudonym handling differently. Developers can refer to the Javadoc reference describing `PseudonymService` interface for more information.

Deprecation Note: HTTP Session Replication

Versions of PingFederate before 5.0 replicate session-state information in the HTTP session object. The Inter-Request State Manager has been updated in the current PingFederate version with methods that supersede HTTP-session replication, allowing greater deployment flexibility.

Some legacy adapters rely on the previous functionality to work in a clustered environment. To re-enable session replication on PingFederate, edit the `jboss-service.xml` file in the `<pf_install>/pingfederate/server/default/deploy/jetty.sar/META-INF/` directory and uncomment the section near the bottom that refers to the Distributed Session Manager. You must also edit the `protocolStack` field and set `mcast_addr` and `mcast_port` to the multicast group address used in your cluster for session replication.

Deploying SaaS-Provisioning Failover

If you are using PingFederate as an Identity Provider (IdP) and have enabled and configured SaaS provisioning, then you can set up one or more failover PingFederate servers specifically for provisioning backup.

> **Note**: SaaS provisioning is available separately from Ping Identity. Contact sales@pingidentity.com for more information.

Provisioning runtime processing and failover is independent of SSO/SLO runtime processing and server clustering described in the preceding sections. However, if you are already deploying, or have deployed, a cluster for federation-protocol runtime processing, then you can use a subset of those servers for SaaS-provisioning failover. Alternatively, you can mix the configuration or set up provisioning-failover servers independently.

> **Important**: Hypersonic databases, including the default database bundled with PingFederate, cannot be used in a failover configuration. Each server in the failover network must be configured to use the same relational database (for example, Oracle). (For more information, see the "Key Concepts" chapter in the PingFederate *Administrator's Manual.)*

To deploy failover for SaaS provisioning:

1. Identify two or more runtime instances of PingFederate to configure for provisioning failover.

   Install new instances of PingFederate if needed (see the "Installation" chapter in *Getting Started*).

   > **Note**: Ensure that a license file enabling SaaS provisioning is installed in each server's `<pf_install>/pingfederate/server/default/conf` directory.

2. For each server instance, edit provisioning properties in the `run.properties` file located in the `<pf_install>/pingfederate/bin` directory (see the next section, "Configuring Runtime Properties").
3. Start or restart all of the PingFederate servers.

   Refer to "Starting and Stopping PingFederate," in the PingFederate *Administrator's Manual*, if needed.
4. After you configure (or reconfigure) SaaS provisioning in the administrative console, replicate the configuration on each server (see "Synchronizing Server Runtime Configuration" on page 19).

   The *Administrator's Manual* and online Help provide detailed information on using the administrative console to configure provisioning.

Configuring Runtime Properties

The `run.properties` in the `<pf_install>/pingfederate/bin` directory contains runtime failover settings. Configure the properties described in the following table for each server in the failover network.

Property	Description
pf.provisioner.mode	Enables or disables provisioning. Allowed values are: `OFF` – SaaS provisioning is disabled (the default). `STANDALONE` – Provisioning is enabled, without failover. `FAILOVER` – Provisioning is enabled, with failover. **Important**: The `STANDALONE` setting is not used for failover configuration; primary and secondary server(s) must be set to `FAILOVER`.
provisioner.node.id	Each server must have a unique index number (from 1 to *n*), which is used to prioritize which server is currently active and which is next in line in case of a failure.
provisioner.failover.grace.period	The time interval (in seconds) between the first indication that a node is dead and failover to the next server in line. The time period should be greater than the Synchronization Frequency set in the administrative console (see the PingFederate *Administrator's Manual).*

Synchronizing Server Runtime Configuration

All nodes in a cluster must have the same configuration settings (as set via the administrative console). This information includes such settings as partner-connection endpoints, data stores, and certificates, as well as SaaS provisioning configuration data, when applicable (see the PingFederate *Administrator's Manual* for complete information).

For a server cluster, you can use either of the following methods to ensure that configuration data is synchronized on all cluster nodes:

- Console Configuration Push
- Configuration-Archive Deployment

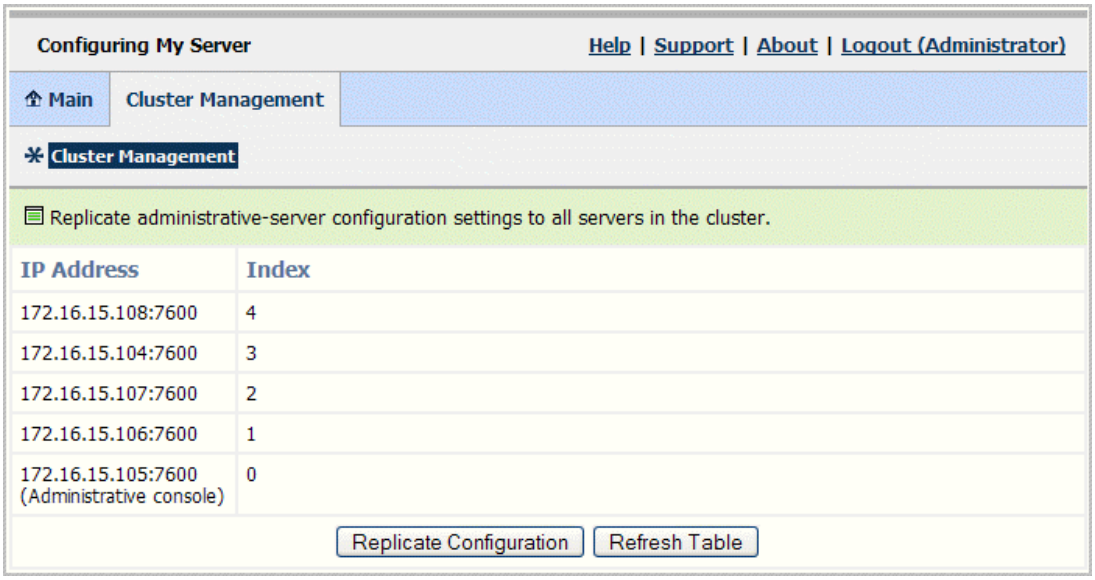
> **Note**: PingFederate also provides a Web Service for synchronizing clustered servers, as part of the Connection Management Service. For information, refer to the "Web Service Interfaces" appendix in the PingFederate *Administrator's Manual.*

For exclusive or mixed-use SaaS-provisioning failover, when the primary and/or backup server(s) are not otherwise part of a cluster, you must use the configuration-archive method (see the previous section).

> **Note**: Changes made directly to configuration files (for example, in `<pf_install>/pingfederate/bin/run.properties`) must be replicated manually across the cluster, as applicable, and PingFederate servers must be restarted if running.

Console Configuration Push

When running in cluster mode, the administrative console provides a **Cluster Management** link on the Main Menu under Administrative Functions.



From this screen you can replicate the current console configuration to all the server nodes in the cluster. The configuration is sent over the network to all nodes in the cluster.

This screen is also useful for verifying that nodes have joined the cluster.

Configuration-Archive Deployment

After you configure or reconfigure the console, you can also update cluster nodes by downloading a configuration archive and then deploying it (either manually or via a scripted process) to each cluster

node or provisioning-failover server. For more information about creating and deploying configuration archives, refer to the "System Administration" chapter of the *Administrator's Manual*.

A configuration archive contains the same information sent during the configuration push from the administrative console described in the previous section. However, this option provides for scheduling and scripting cluster synchronization.

Note: You must still configure each node's session-state services, if you are using settings other than defaults (see "Managing Deployment Architecture" on page 8). The archive does not contain information about clustering; it contains only information about server settings and partner configuration.