IdPClusterIntro

IdP Clustering Configuration

Actually, we fooled you, this page contains nothing about configuring the IdP to run in a clustered fashion. However, please read **all** of this page because it provides critical information for setting up the environment into which you will deploy you clustered IdPs.

Terminology

Many individuals consider high availability, load balancing, and clustering to be the same thing. They are not. Within this document the terms are used and have the following definitions:

node - A single IdP instance.

cluster - A collection of nodes.

high availability - The ability for nodes to fail without loss of existing operational data (e.g. sessions).

fail over - The ability to detect a node failure and redirect work destined for a failed node to an operational node.

load balancing - The ability to (relatively) evenly distribute the cluster's workload amongst all nodes.

Note, the number of nodes within a cluster need not correlate to the number of servers (physical or virtualized) within the cluster. Each server may run more than one instance of the IdP software and thus count as more than one node. Also note that high availability, fail over, and load balancing are distinct features and not all solutions in this area provide all features. Don't worry though, the solution the Shibboleth team recommends provides all three.

Finally, be aware that fail over and load balancing are completely outside the control of the IdP. These require some mechanism (two described below) for routing traffic before it even reaches a node.

Application State and its Effect on Clustering

The IdP is a stateful application. That is it maintains operational information, between requests, that is required to answer subsequent requests. The IdP keeps this information in memory (for various reasons cookies cannot be used as they are in some web applications). Therefore, in order to achieve high-availability this information must be shared amongst all IdP nodes. By default the V2 release assumed (and documented) the use of Terracotta as the mechanism for doing this. This is still supported, but only on Java 6 because the Terracotta solution was not ported to support Java 7.

At present, the only options for clustering in-memory state with Java 7 are third party extensions, such as the memcached plugin. An alternative that relies on client-side state is mentioned below.

Like most applications that use this approach, each IdP node keeps the state it creates in memory in a form readily usable by the node but uses a more compact form when making it available to other nodes. Therefore, any load balancing solution used should route all subsequent requests to the same node that serviced the initial request. This prevents the IdP nodes from constantly reading/writing information to/from this more compact form (an expensive process). This is generally known as session affinity load balancing.

Two Common Clustering Approaches

Most environments use one of two methods for creating a cluster of nodes that look like one single service instance to the world at large.

DNS Round Robin

This is done by registering each cluster node under the same hostname. When a DNS lookup is performed for that hostname the DNS server returns back a list of IP addresses (one for each node) and the client chooses which one to contact.

Pros:

- Easy to setup

Cons:

- No guarantee on fail-over characteristics. If a client chooses an IP from the list and then continues to stick with that address, even if a node is unreachable, the service will appear as unavailable for that client until their DNS cache expires.
- No guarantee on load-balancing characteristics. Because the client is choosing which node to contact, clients may "clump up" on a single node. For example, if the client's method of choosing which node to use is to pick the one with the lowest IP address all requests would end up going to one node (this is an extreme example and it would be dumb for a client to use this method).
- If a client randomly chooses an IP from the list for each request it makes there will be a constant churn of state data between nodes.
- This approach can not be used to run multiple nodes on the same IP address since DNS does not provide port information.

The Shibboleth team **strongly** discourages this approach.

Hardware-Based Clustering

This is done by using specially dedicated hardware to intercept and route traffic the various nodes in a cluster (so the hardware basically becomes a switch in front of the nodes). This hardware is then given the host name for all the services provided by the clusters behind it.

Pros:

- Guaranteed high-availability, load-balancing, and fail-over characteristics
- Can increase the security of each node by making them accessible only to the clustering hardware and not to the outside world

Cons:

- More difficult to set up
- Requires purchase of equipment (some solutions can be very costly)
- Adds additional complexity to deployment environment

Because of the guaranteed characteristics provided by this solution, the Shibboleth team recommends this approach. Caution should be taken though to ensure that the load balancing hardware does not become a single point of failure (i.e. buy and run two of them).

Configuring the IdP for Clustering

The above description is mostly focused on the actual Identity Provider servlet itself when it comes to clustering. However, since the IdP relies on other services for authentication and attribute resolution, it is important to also make sure that these services are clustered as well. Therefore, one should make sure in a production environment to:

- Have the **IdP servlet** running on multiple cluster nodes. How to do this is described in the howto configure the IdP for Clustering.
- Have the **attribute resolver** configured to use multiple LDAP/AD servers. How this can be done is described in the LDAP Resolver description.
- Also protect the **authentication system** to make use of clustering. Have a look at the username/password authentication description.

When it comes to using LDAP (most common case) for attribute resolution and authentication, you may also have a look at the multiple LDAP configuration hints.

Stateless Clustering

Finally, if you're willing to give up some features and (in many cases) do some custom development, it's possible to deploy the IdP in a configuration that does not rely on shared runtime state. See this topic for more information.

Proxy Clustering

Further finally, if you front-end your server with Apache and are willing to give up some features, it's possible to cluster an IdP using Apache rewrite and proxy directives . See this topic for more information.