

# CESNET Technical Report 16/2009

## Shibboleth IdP cluster using Terracotta

IVAN NOVAKOV

Received 12.12.2009

### Abstract

The article describes how to deploy Shibboleth IdP in cluster environment using Terracotta for session replication. The text is suitable for skilled Shibboleth IdP administrators with general knowledge of Apache web server, Tomcat servlet container and networks in general.

*Keywords:* Shibboleth idp, cluster, high availability, load balancing, terracotta

## 1 Introduction

The identity provider is one of the most crucial services running on the organization's network. If the service is down, users will not be able to access most of the internal network applications and will not be able to use federated applications and services at all.

With the increasing number of network applications requiring authentication and the increasing number of users using them, the working load of the identity provider increases too, which may result in longer response times or even nasty time-outs.

The ability of a system to ensure a certain degree of operating continuity is called high availability. The most common way to achieve high availability is building a cluster of nodes, where one node is active and others serve as backup.

If a network service is highly utilized the load can be distributed evenly across two or more identical nodes running the same software. This technique is known as load balancing. The hardware or software component responsible for load distribution is called load balancer.

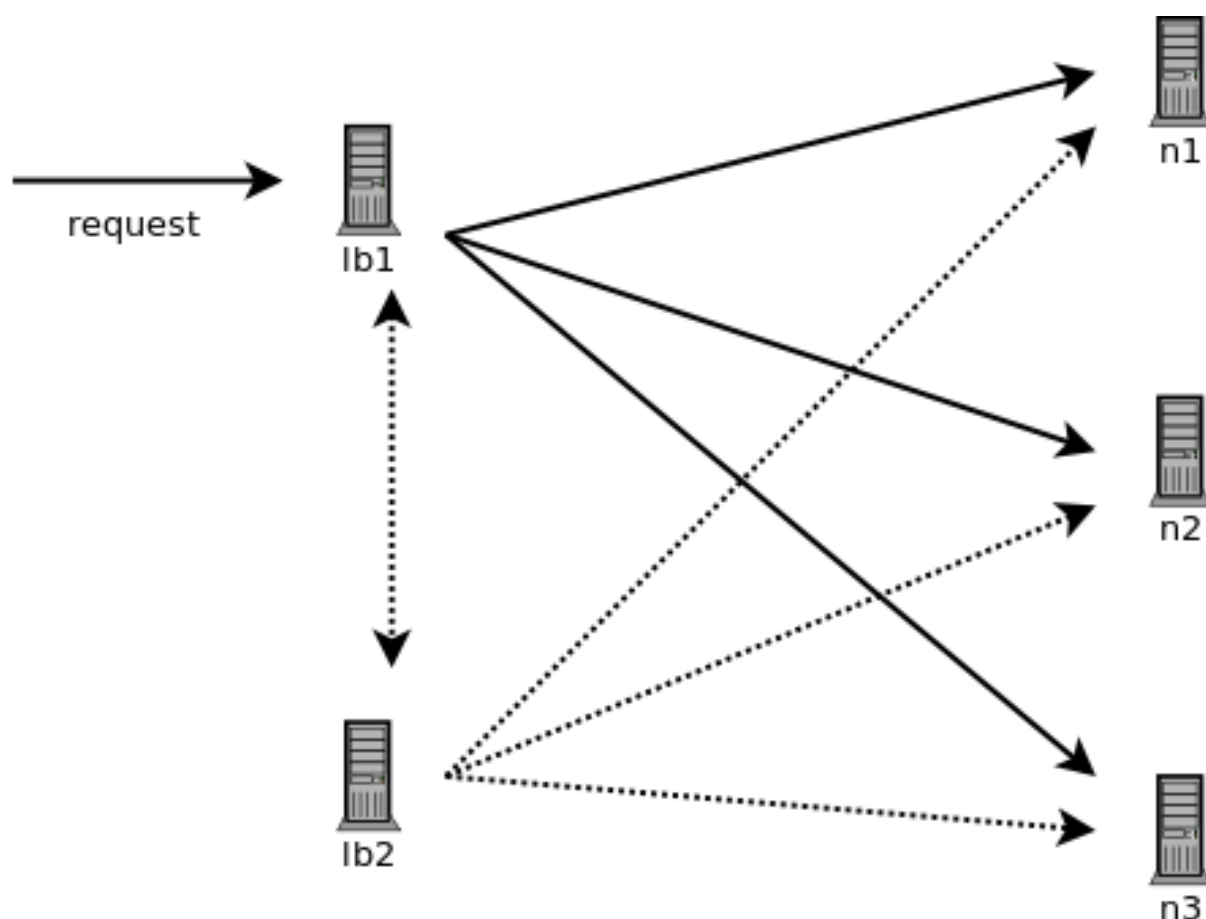
High availability and load balancing are different things, although they are often implemented and deployed together. One of the common solutions is a set-up with two (or more) nodes running the application with two (or more) load balancers in high availability mode in front of the nodes (Figure 1). Only one load balancer is active at a time. It dispatches the incoming requests and distributes them evenly across the nodes. If a node fails, it is automatically excluded from the pool of active nodes and the load balancer stops addressing it with requests. If the load balancer fails, it is immediately replaced by a backup load balancer.

## 2 Shibboleth IdP cluster overview

Our Shibboleth IdP cluster follows the design of the set-up described in the introduction. We have two nodes running Shibboleth<sup>1</sup> IdP under Tomcat and two load

---

<sup>1</sup> <http://shibboleth.internet2.edu/>



**Figure 1.** A cluster with two load balancers and three nodes

balancers running Apache. The load balancers use common high availability mode set-up (heartbeat), which is beyond the scope of this article.

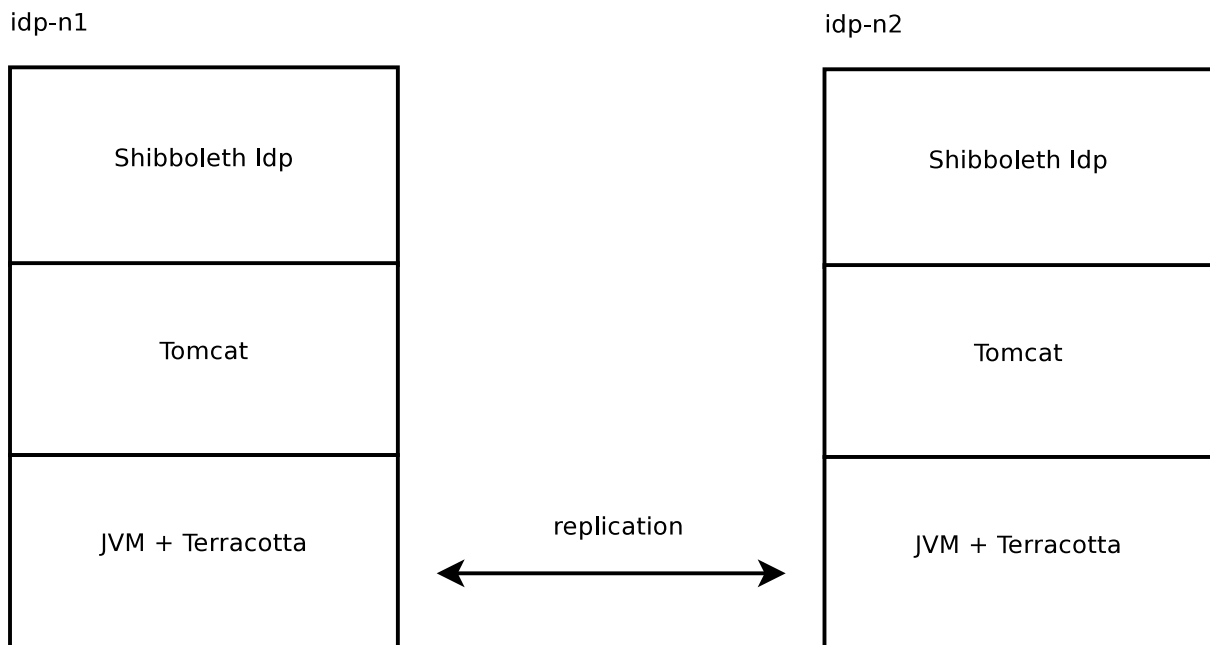
Similarly to the single server set-up, Apache serves as a front-end with SSL offload. But instead of passing the requests to a Tomcat instance installed locally, Apache uses the `mod_proxy_balancer`<sup>2</sup> module to distribute the requests across the Tomcat nodes. Like in the single server set-up, Apache communicates with Tomcat nodes over the AJP protocol (`mod_proxy_ajp`<sup>3</sup>).

The nodes itself need to share their session storages in order to provide seamless functionality of the IdP cluster. For example, if the initial authentication request is served by node N1, after successful authentication a login session is stored at node N1. Then, if a subsequent authentication request is served by node N2, the single sign-on functionality will be broken, since node N2 has no login session stored.

The recommended method is to use Terracotta to replicate the in-memory state between nodes. Terracotta functions as a distributed cache at the JVM level. That means, the replication is transparent and hidden from the application itself and developers need not modify the application's code to use it (Figure 2).

<sup>2</sup> [http://httpd.apache.org/docs/2.2/mod/mod\\_proxy\\_balancer.html](http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html)

<sup>3</sup> [http://httpd.apache.org/docs/2.2/mod/mod\\_proxy\\_ajp.html](http://httpd.apache.org/docs/2.2/mod/mod_proxy_ajp.html)



**Figure 2.** Terracotta in-memory state replication

### 3 The network

The load balancer and the nodes must be connected through a secure network. The AJP connections between Apache and Tomcat are not secured in any way. Replication data transmitted by Terracotta are not encrypted too.

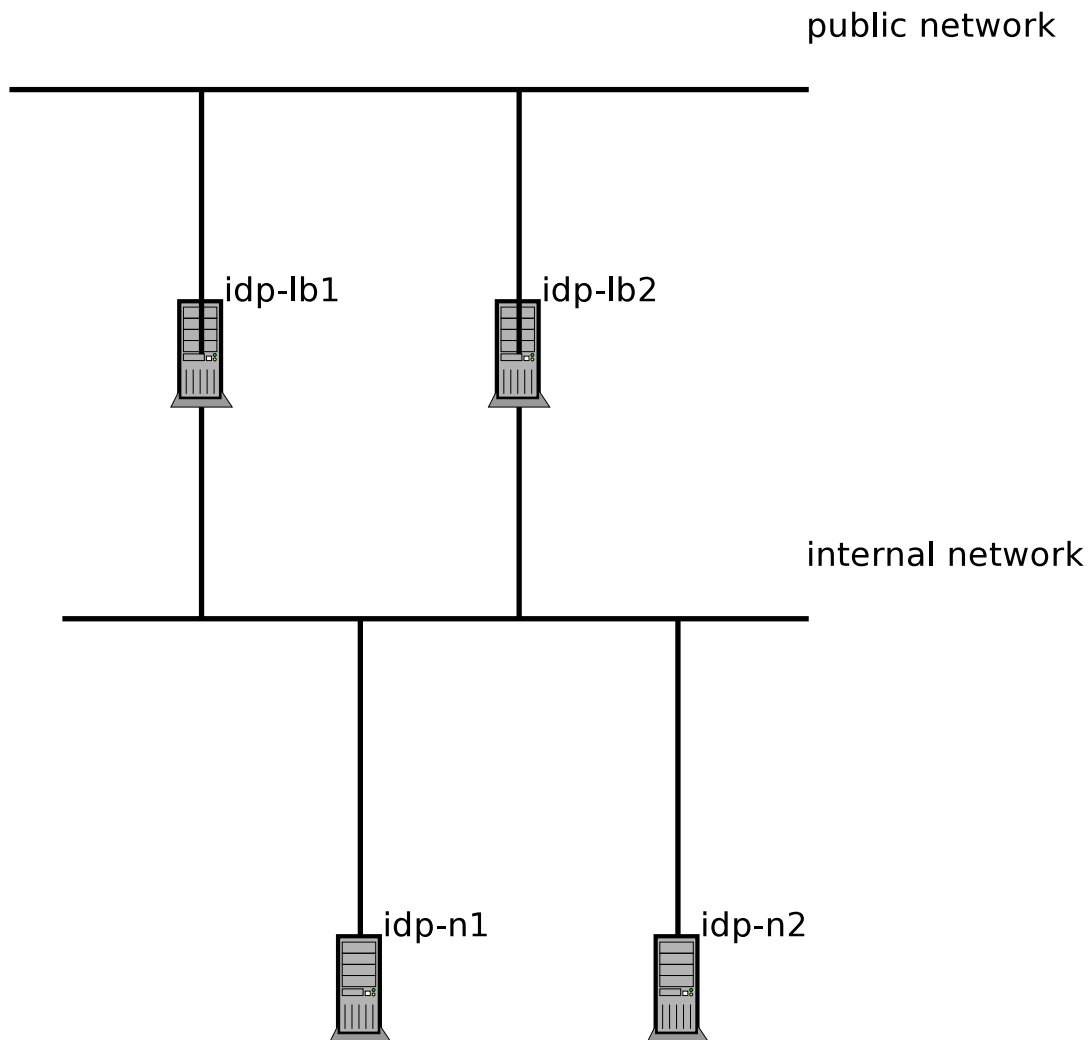
The best way to secure the communication is to physically separate the “internal” network from the “public” network. That means, the load balancers need a second network interface and a dedicated switch has to be deployed to connect the components together (Figure 3).

That is not always possible, especially if the nodes are situated in different geographical locations. One option is to use a virtual private network (VPN). A viable option is to use an OpenVPN<sup>4</sup> server at the load balancer and OpenVPN clients at the nodes.

Another option is to use IPsec in transport mode to secure the routes between the load balancers and the nodes. IPsec is based on open standards and is widely supported across the different operating systems and platforms [3].

Also, it is possible to use a separate VLAN if you have enough control over VLAN creation and maintenance. And finally, you may secure the communicating ports of all boxes with appropriate firewall settings. The last two options (VLAN and firewall) are not recommended, because they require additional measures and conditions to be really secure. For example, under normal conditions everyone, who has access to the network infrastructure, might be able to view the communication.

<sup>4</sup> <http://www.openvpn.net/>



**Figure 3.** Network representation of the cluster

## 4 Apache as load balancer

For our set-up we need Apache with SSL support and the following modules enabled: `mod_proxy`, `mod_proxy_ajp` and `mod_proxy_balancer` modules. The basic configuration is fairly simple. We need to create a SSL enabled virtual host and place there the balancer related configuration as follows:

```
<VirtualHost idp.example.com:443>
  [..]
  <Proxy balancer://idpcluster/>
    BalancerMember ajp://node-n1:8009
    BalancerMember ajp://node-n2:8009
  </Proxy>
  ProxyRequests Off
  ProxyPass /idp/ balancer://idpcluster/idp/
  ProxyPreserveHost On
  [..]
</VirtualHost>
```

Here we have two nodes – *node-n1* and *node-n2* (hostnames or IP addresses). Every request beginning with */idp/* will be passed to the balancer, which will then choose a suitable node to pass the request to.

## 5 Shibboleth IdP

On each node we have a standard Shibboleth IdP installation under the Tomcat container. In most cases, the configuration should be the same as if it is a single IdP server. Of course, all the nodes should use the same configuration and the same certificate.

It is important to realize, that all the nodes are hidden behind the load balancer. So we must configure all the IdP instances with the hostname of the load balancer. All we need to do is to enter the right hostname during the installation process.

If the nodes have no direct access to the public network, they will not be able to download federation metadata, if using a remote metadata provider. In that case, an alternative method of providing the metadata should be deployed. For example, some kind of reverse proxy may be set up at the load balancer, so the nodes will be able to download federation metadata.

Also, we have to make sure, the AJP connector in Tomcat is listening on the right interface.

## 6 Terracotta

### 6.1 Overview

Terracotta is composed of two parts. Terracotta servers that are responsible for lock management and moving data between clients and Terracotta clients which run in the IdP software and publish/receive changed data to/from the server [1].

It is recommended to run a Terracotta server on each node. Terracotta servers run in high availability mode. They share the same configuration and know how to connect to each other. Initially, through a distributed algorithm one server is selected to run in active mode, while the rest run in passive mode. If the active server fails, the passive servers detect that and select another server to be active.

That model is fairly robust, but it also has some design weaknesses. For example, if we restart all servers at the same time (due to a power outage for instance) and a former passive server comes up before the server which was active before the restart, the cluster will not be able to recover correctly [4]. It is recommended to restart only one server at a time and restart the next one, after the previous one has joined the pool. Or, if all the servers are down, the server, which has been active before, should be started first.

### 6.2 Configuration file

The latest Shibboleth IdP distribution packages contain a sample Terracotta server configuration file – *tc-config.xml* placed in *IDP\_HOME/conf/*. But it may be out-

dated, so it is recommended to use the latest version published at the Internet2 Wiki site [1].

The configuration file contains a definition of each Terracotta server in the cluster, some other configurations like logging settings and a bit complex structure of what has to be replicated. In our sample configuration file *tc-config.xml* we can use the example server configuration, which is at the beginning of the file. We need to copy it as many times as the number of our Terracotta servers and edit the following attributes for each instance:

- **name** - a unique name for this server (e. g. idpNode1)
- **host** - the hostname or IP address of the server
- **bind** - (optional) specify the IP address to which the server should bind (defaults to 0.0.0.0)

We may also adjust the paths to the log files to suit our set-up.

```
<server name="idpNode1" host="idp-n1.example.org" bind="192.168.1.15">
  <dso>
    <persistence>
      <mode>permanent-store</mode>
    </persistence>
  </dso>
  <logs>$IDP_HOME/cluster/server/logs</logs>
  <data>$IDP_HOME/cluster/server/data</data>
  <statistics>$IDP_HOME/cluster/server/stats</statistics>
</server>
```

All Terracotta servers and clients should use the same configuration file.

### 6.3 Installation

We need to download the open source distribution of Terracotta 3.x<sup>5</sup> and extract the archive in a suitable directory (*/opt* for instance). For the installation we have to set the proper environment variables.

```
# cd /opt
# ln -s terracotta-3.1.1 terracotta
# export TC_HOME=/opt/terracotta
# export TC_INSTALL_DIR=$TC_HOME
```

Then we are ready to install the Terracotta Integration Module (TIM) used by the IdP:

```
# $TC_HOME/bin/tim-get.sh install tim-vector 2.5.1 org.terracotta.modules
```

And the TIM for Tomcat:

```
# $TC_HOME/bin/tim-get.sh install tim-tomcat-6.0 2.0.1
```

Just to be sure, we have to check if our *tc-config.xml* file refers to the right versions of the both TIM components:

<sup>5</sup> <http://www.terracotta.org/>

```

<clients>
  <logs>${IDP_HOME}/cluster/client/logs-%i</logs>
  <statistics>${IDP_HOME}/cluster/client/stats-%i</statistics>
  <modules>
    <module name="tim-vector" version="2.5.1" group-id="org.terracotta.modules"/>
    <module name="tim-tomcat-6.0" version="2.0.1"/>
  </modules>
</clients>

```

Then we can generate the JAR file used by Tomcat. This JAR is specific to the type and version of our JVM. We will run the following command and write down the path of the JAR file. We will need it to start the IdP:

```

# $TC_HOME/bin/make-boot-jar.sh -fIDP_HOME/conf/tc-config.xml
2009-11-22 14:21:17,531 INFO - Terracotta 3.1.1, as of 20091009-141004
(Revision 13788 by cruise@su10mo5 from 3.1)
2009-11-22 14:21:18,143 INFO - Configuration loaded from the file at
'/opt/shibboleth-idp/conf/tc-config.xml'.
2009-11-22 14:21:28,235 INFO - Creating boot JAR file at
'/opt/terracotta-3.1.1/lib/dso-boot/dso-boot-hotspot_linux_160_12.jar'...
2009-11-22 14:21:28,739 INFO - Successfully created boot JAR file at
'/opt/terracotta-3.1.1/lib/dso-boot/dso-boot-hotspot_linux_160_12.jar'.

```

## 6.4 Starting the servers

We must start the servers before we start the IdP. The Terracotta servers are started with the *start-tc-server.sh* script located in *\$TC\_HOME/bin*. The script takes two parameters:

- **-n** - the unique name of the server as written in the Terracotta configuration file
- **-f** - the fully qualified file system path or URL to the Terracotta configuration file

For example:

```
# $TC_HOME/start-tc-server.sh -n idpNode1 -f /opt/shibboleth-idp/conf/tc-config.xml
```

Or with a URL:

```
# $TC_HOME/start-tc-server.sh -n idpNode1 \
> -f http://idp-lb.example.org/idp-cluster/tc-config.xml
```

## 6.5 Starting the IdP and the client

To start the Terracotta client within the JVM, we have to add the following three arguments to the JVM command line used to start Tomcat:

- **-Dtc.install-root=\$TC\_HOME**
- **-Dtc.config=<config file path or URL>**

- `-Xbootclasspath/p:<path to the JAR>`

Usually it will be enough to set the `JAVA_OPTS` environment variable, for example:

```
JAVA_OPTS="-Dtc.install-root=/opt/terracotta \
-Dtc.config=http://idp-bl.example.org/idp-cluster/tc-config.xml \
-Xbootclasspath/p:/opt/terracotta/lib/dso-boot/dso-boot-hotspot_osx_160_12.jar"
```

This setting may be placed in the Tomcat start-up script. Or, in a Debian-based linux distribution we may modify the `/etc/defaults/tomcat6` file appropriately.

Then we are ready to start Tomcat.

## 7 Running the cluster in production mode

Running a Shibboleth IdP cluster raises some specific issues typical for a distributed environment. First of all, along with Tomcat, Apache and Shibboleth, another piece of software is being used – Terracotta. Terracotta is a complex server-client architecture with its own configuration, network communication and logging. The configuration files for Shibboleth IdP and the Terracotta server should be the same on all nodes. So it is a good idea to deploy a suitable way of distribution of the configuration files across the nodes. Another issue are the logs – both for Shibboleth and Terracotta. Having separate logs on each node is not very convenient. It is recommended to gather them at one place, where they can be easily viewed and analysed.

Shibboleth IdP itself depends on other services, that provide authentication and attribute storage. For the purpose of making the cluster really efficient, those services should be clustered as well. In most cases, LDAP is used for authentication and attribute resolution. The Shibboleth IdP configuration provides enough options for using multiple LDAP servers both as authentication authority and attribute storage.

## References

- [1] LA JOIE, C. et al. *Configuring IdP Clustering*. Internet2, 14 October 2009 [cit. 2009-12-12]. Available online<sup>6</sup>.
- [2] LA JOIE, C. et al. *IdP Clustering Configuration*. Internet2, 4 February 2009 [cit. 2009-12-12]. Available online<sup>7</sup>.
- [3] IPsec. In *Wikipedia, The Free Encyclopedia*. [cit. 2009-12-12]. Available online<sup>8</sup>.
- [4] GAIRIFO. `com.tc.l2.ha.ClusterIDMismatchException`. In *Terracotta Discussion Forums*, 14 September 2009, 06:26:53 [cit 2009-12-12]. Available online<sup>9</sup>.

<sup>6</sup> <https://spaces.internet2.edu/display/SHIB2/IdPCluster>

<sup>7</sup> <https://spaces.internet2.edu/display/SHIB2/IdPClusterIntro>

<sup>8</sup> <http://en.wikipedia.org/wiki/Ipsec>

<sup>9</sup> <https://forums.terracotta.org/forums/posts/list/2509.page>