

Sakai Admin Guide - Advanced Tomcat (and Apache) Configuration

Created by Tony Atkins, last modified by Neal Caidin on May 25, 2016

Advanced Tomcat Configuration

Tomcat as configured out of the box is adequate for development and very basic exploration of Sakai. If you want to move beyond this point, you will need to satisfy a few additional concerns, namely handling SSL and scaling beyond a single server.

Redirecting Traffic to the Portal

The default root context for tomcat provides documentation for tomcat itself when users visit the root of your site. When running Sakai in production, you should redirect traffic to your portal of choice by removing or renaming the existing content in TOMCAT/webapps/ROOT and replacing it with an index.html file with contents like:

```
<html>
<head>
<title>Redirecting to /portal</title>
<meta http-equiv="Refresh" content="0:URL=/portal">
</head>
<body bgcolor="#ffffff" onLoad="javascript:window.location='/portal';">
<div style="margin:18px;width:288px;background-color:#cccc99;padding:18px;border:thin solid #cccc99">
<p style="margin-top:0px">
You are being redirected to the Sakai portal. If you are not automatically redirected, use
<a href="/portal">Take me to the Sakai portal</a>
</p>
</body>
</html>
```

Configuring tomcat to handle SSL

SSL can be handled by tomcat itself, by putting tomcat behind an apache front end, or by putting tomcat (with or without apache) behind a dedicated load balancer. Tomcat standalone can either handle SSL natively, or can now be configured to use the tomcat-native jars and APR libraries to handle http and https.

Configuring tomcat to handle SSL natively

If you want to expose your Sakai installation to end users, you will need to set up SSL support using a certificate signed either by a local or external certificate authority. If you do not already have a certificate, you can create a self-signed certificate by running the following command as the user that will run sakai:

```
keytool -genkey -alias tomcat -keyalg RSA
```

If you need to obtain and install a certificate signed by a trusted authority, review the tomcat documentation [here](#).

shboard / Documentation / ... / Sys Admin Guide

You can also take an existing X509 certificate and private key and import them into a keystore using the following utility:

http://www.comu.de/docs/tomcat_ssl.htm

Once you have a .keystore file in your service user's root directory, you will need to uncomment the SSL connector in your tomcat's server.xml file:

```
<Connector port="8443" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
```

Configuring tomcat to use APR

If you take the time to install APR and the tomcat native libraries, tomcat can be made to work with an X509 certificate in PEM format without conversion. Those instructions can be found at:

<http://tomcat.apache.org/tomcat-5.5-doc/apr.html>

Configuring tomcat to force SSL for appropriate content

At a minimum, any components (login, xlogin) that transmit username and password information should be transmitted using SSL. To avoid problems with session cookie hijacking, it is recommended that an entire Sakai installation be secured with security being relaxed only as needed.

Securing Individual Tomcat Contexts

To secure an individual tomcat context, you must edit its WEB-INF/web.xml and add the following inside the <web-app> tags:

```
<!-- redirect all traffic to the SSL port -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Automatic SLL Forwarding</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

At a minimum, you should secure The ROOT and login contexts.

Securing All Tomcat Contexts

If you wish to secure the entire tomcat installation (which can be done regardless of how you provide SSL), add the following to TOMCAT_HOME/conf/web.xml inside the web-app tags:

```
<!-- redirect all traffic to the SSL port -->
<security-constraint>
  <web-resource-collection>
```

```

:      <web-resource-name>Automatic SLL Forwarding</web-resource-name>
      <url-pattern>/*</url-pattern>
</web-resource-collection>
<user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
</security-constraint>

```

DAV issues when requiring SSL site-wide

When using the above configuration to require SSL for your entire tomcat installation, you must ensure that the serverURL specified in your sakai.properties file specifies the https protocol, including the port if the port is not 443. If you fail to do this, the DAV functionality built into Sakai may not function properly. (You'll also be requiring users to suffer through a redirect for most if not all URLs they follow within your Sakai installation.)


Relaxing Security Restrictions for a Single Context

If you have content that is required to be transmitted using http (for example, a tool that exports subscribable iCal calendars), you may want to relax the overall security restriction for a single context. This can be done by adding the following to a context's WEB-INF/web.xml file:

```

<security-constraint>
  <web-resource-collection>
    <web-resource-name>SSL Requirement Disabled</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

```

 This will only ensure that SSL is not required. You can still of course use SSL to communicate with this context.

Configuring Apache to proxy connections to tomcat

The following instructions will describe the basic options for setting up tomcat behind apache 2.2. NGinx and other frontends also work great, and their configuration is similar. Apache offers two main methods to proxy connections to tomcat: mod_proxy and mod_proxy_ajp.

Configuring mod_proxy

mod_proxy proxies connections to tomcat using plain http or https. The following configuration directives in your apache server's httpd.conf will map all contexts from a stock tomcat installation running on port 8080 to the root of the apache install:

```

#On some servers (like Ubuntu) you have to enable mod_proxy and mod_rewrite (a2enmod)
ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080/

#Requires that header module be enabled (sudo a2enmod headers)
RequestHeader set X-Forwarded-Proto "https"

```

```

:
ProxyRequests Off
ProxyPreserveHost On

<Proxy *>
    Order deny,allow
    Allow from all
</Proxy>

#Redirects portal
RewriteEngine on
#RewriteRule ^/$ /portal/ [L,R]
#This rewrite rule is probably better
RewriteCond %{SERVER_PORT} !^443$
RewriteRule ^/(.*) https://%{HTTP_HOST}/$1 [NC,R,L]

```

If you are proxying an SSL connection to an SSL connection or a non-SSL connection to a non-SSL connection, you will need to reconfigure tomcat's connectors so that they are aware of the ports that end users are using. Otherwise the URLs tomcat constructs internally will reflect the wrong port numbers. This is accomplished by modifying the connectors defined in TOMCAT_HOME/conf/server.xml along these lines:

```

<Connector port="8080" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" redirectPort="443" acceptCount="100"
    connectionTimeout="20000" disableUploadTimeout="true"
    proxyName="{YOUR SERVICE NAME}"
    proxyPort="80"
/>

<Connector port="8443" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" disableUploadTimeout="true"
    acceptCount="100" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    proxyName="{YOUR SERVICE NAME}"
    proxyPort="443"
/>

```

Additionally, because of recent browser updates with enhanced Mixed Content protection you *may* need to add a valve to Tomcat to allow it to know about the forwarded protocol. In the Engine block for localhost you'd want to add this block. This would match the request header you're passing in Apache/Nginx.

```

<Engine name="Catalina" defaultHost="localhost">
    . . .
    <Valve className="org.apache.catalina.valves.RemoteIpValve" protocolHeader="X-Forwarded-Proto" />
    . . .
</Engine>

```

⚠ For older Sakai instances running tomcat 5.5 (2.8) you have to get the jar from <https://code.google.com/p/xebia-france/wiki/RemotelpValve>. Then put this jar in server/lib and use the className: *org.apache.catalina.connector.RemotelpValve*

The direct location to the jar (since it's kind of buried in the page) is <http://xebia-france.googlecode.com/files/xebia-tomcat-extras-tc55-1.0.0.jar>

The key changes are updating the redirectPort for non-SSL ports, and adding the ProxyName and ProxyPort attributes. Note that this must be done for each active connector that is being mapped using mod_proxy.

If you use mod_proxy to proxy an SSL connection to a non-SSL tomcat connector (such as the one that runs on port 8080 by default), you will need to add the force.url.secure parameter to your sakai.properties file, typically with a value like:

```
force.url.secure=443
```

Configuring mod_proxy_ajp

AJP, or the The Apache Jserv protocol is the protocol by which tomcat and some other proxying service (Apache or a dedicated hardware load balancer) communicate. If you intend to use apache to provide other static content or to handle SSL, you may want to configure Tomcat to listen for AJP requests, and the have Apache proxy requests for tomcat using AJP rather than HTTP or HTTPS. To do this, you will need to do the following:

1. enable the AJP connector for one or more tomcat installations by editing TOMCAT_HOME/conf/server.xml and uncommenting the AJP connector included with the default distribution and adding the correct URIEncoding:

```
<Connector enableLookups="false" port="8009" protocol="AJP/1.3" redirectPort="8443" URI
```

⚠ If you are load balancing multiple tomcat instances on the same server, you will need to ensure that each one listens for AJP requests on a different IP address and/or port.

2. You can load balance requests between tomcat instances by adding something like the following to your Apache httpd.conf:

```
ProxyPass / balancer://sakaiCluster/ stickysession=JSESSIONID nofailover=On
<Proxy balancer://sakaiCluster>
BalancerMember ajp://localhost:8009
BalancerMember ajp://localhost:8019
BalancerMember ajp://localhost:8029
</Proxy>
```

⚠ You must have mod_proxy, mod_proxy_ajp, and mod_balancer enabled to use the above configuration options. It has also been noted that versions of mod_proxy_balancer prior to 2.2.4 have errors with this configuration.

Configuring mod_jk

mod_jk is an earlier (and still supported) apache module that supports proxying connections via AJP. mod_jk is available from tomcat.apache.org:

<http://tomcat.apache.org/download-connectors.cgi>

```
# Set the location of the worker.properties file
JkWorkersFile    /etc/apache2/worker.properties

# mount the root and all subdirectories associated with our Sakai installation
JkMount /* sakaiWorker
```

You will then need to create a worker.properties file at the location specified in your httpd.conf which contains lines like the following:

```
# We only have one worker, so this is a short list.
worker.list=sakaiWorker

# The minimum properties for the sakaiWorker are set here
worker.sakaiWorker.type=ajp13
worker.sakaiWorker.host=localhost
worker.sakaiWorker.port=8009
```

For more information on configuring mod_jk (including examples that demonstrate using mod_jk to load balance between multiple tomcat instances), consult the [Tomcat connector documentation](#).


Configuring Apache to Use Compression

If you are proxying connections to Tomcat through Apache, you may wish to look at enabling and configuring compression in Apache. This will allow Apache to greatly reduce the size of generated content prior to delivery to end users. This appeals particularly to sites whose users download content over dial-up or otherwise bandwidth-limited connections. Below are sample settings to be placed in httpd.conf (thanks to Stephen Marquard for those):

```
# mod_deflate (compress output for browsers that support it)
AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css application/x-javascript

# Some adjustments for IE browsers (c/f
http://www.robertswarthout.com/rswarthout/2007/05/ie-6-apache-mod\_deflate-blank-pages/)

BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4\.0[678] no-gzip
BrowserMatch \bMSIE\s7 !no-gzip !gzip-only-text/html
BrowserMatch \bMSIE\s8 !no-gzip !gzip-only-text/html
```

 Some sites have experienced problems using compression with Apache 2.0. Apache 2.2 or higher is recommended when using compression.

Configuring Tomcat Standalone to Work with a Load Balancer

If you have a software or hardware load balancer (see the [Sakai Admin Guide - Load Balancing and Scaling](#) section of the admin guide for more information), you can typically configure it to remap all traffic for the normal http and https ports associated with

Dashboard / Documentation / ... / Sys Admin Guide

As above, if you proxy an SSL connection to Sakai to a non-SSL connector, you will need to add the `force.url.secure` parameter to your `sakai.properties` file, typically with a value like:

```
force.url.secure=443
```

References

For more information, visit the following pages:

http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
<http://tomcat.apache.org/tomcat-5.5-doc/config/ajp.html>
<http://tomcat.apache.org/tomcat-3.3-doc/AJPv13.html>
http://edocs.bea.com/wls/docs61/webapp/web_xml.html

adminguide2_4

3 Comments



Dominic Hargreaves

A `mod_proxy_balancer` configuration (which requires `nofailover=on` due to the lack of session replication in sakai) doesn't behave very well with older versions of `mod_proxy_balancer`. In particular, Apache 2.2.3 (ships with eg Debian etch) suffers from the bug documented at https://issues.apache.org/bugzilla/show_bug.cgi?id=38962 - when a backend is marked as having an error, sessions already on that worker always end up with 503s afterwards. This is fixed in 2.2.4 and the patch is simple and applies easily to 2.2.3 (which I've done for our installation).



jacques pignon

Configuring `mod_proxy_ajp`, don't forget to add `:URIEncoding="UTF-8"` in the AJP connector : file `TOMCAT_HOME/conf/server.xml`
`<Connector port="8009"`
`enableLookups="false" redirectPort="8443" protocol="AJP/1.3"`
`URIEncoding="UTF-8" />`
usefull for resources uri with spaces



*Additionally, because of recent browser updates with enhanced Mixed Content protection you *may* need to add a valve to Tomcat to allow it to know about the forwarded protocol. In the Engine block for localhost you'd want to add this block. This would match the request header you're passing in Apache/Nginx.*

```
<Engine name="Catalina" defaultHost="localhost">
. . .
    <Valve className="org.apache.catalina.valves.RemoteIpValve" protocolHeader="X-Forwarded-Proto" />
. . .
</Engine>
```

I'm not sure if this would be the same as described above, but we are using a load balancer and we needed to apply additional properties to the valve as described in the comments in KNL-1382

```
<Valve className="org.apache.catalina.valves.RemoteIpValve"
    internalProxies="128\.6\.210\.69"
    remoteIpHeader="x-forwarded-for"
    proxiesHeader="x-forwarded-by"
    protocolHeader="x-forwarded-proto" />
```

Powered by a free **Atlassian Confluence Open Source Project License** granted to Apereo Foundation. Evaluate Confluence today.

This Confluence installation runs a Free Gliffy License - Evaluate the Gliffy Confluence Plugin for your Wiki!

