Web Server HOWTO

Because Spk provides its services via the World Wide Web, an important component of the overall system is the web server. This document describes the installation and configuration of web server software to support Spk.

To complete the configuration for secure https connections, you will need two authentication certificates. If you do not have these certificates, it might be a good idea to obtain them before proceeding with the web server configuration. See the Authentication and Encryption HOWTO [1] for details.

Table of Contents

Introduction 1
Tomcat Instances and Web Applications 1
Installing Tomcat 3
Defining Environment Variables 3
Creating Four Run-Time Instances 3
Configuring User Groups and Permissions 4
Configuring the Instances 4
Setting the Instances to Run Automatically 8
Setting Authentication Certificates 9
Deploying the Spk Web Applications 10
Stopping and Starting 11

Introduction

The client component of Spk, called the Model Design Agent (MDA), is written in the Java language. It communicates with the rest of the system via the Internet, using the *http* and *https* protocols. On the server side of this communication is a web server, which is software that supports World Wide Web functionality and runs continuously in a server machine. The fact that the MDA is a Java application makes it very desirable that the components that communicate with the MDA from the server side be also written in Java. Server-side Java components such as those in Spk are known as *Java servlets*. A closely related technology, called *Java Server Pages (JSP)*, is an extensible set of extensions to HTML which can be used to generate servlets.

In order to run servlets and JSP within a web server, software known as a *servlet/JSP container* is required. Tomcat, a free product of the Catalina project, which in turn is part of the Jakarta project sponsored by Apache, the leading developer of web server software, serves this purpose for Spk. Tomcat can be added to an Apache web server, in which case it provides only the servlet/JSP container function, or it can be installed as a complete stand-alone web server and servlet/JSP container. For the purposes of Spk, the latter is the simpler solution.

The rest of this document describes the installation and configuration of tomcat for Spk.

Tomcat Instances and Web Applications

It is possible to run several copies of Tomcat simultaneously. Each independently running copy is called an *instance*. As will be explained, shortly, an Spk installation requires four Tomcat instances.

A Tomcat instance expects to find its configuration information, as well as its applications, in a file hierarchy of a certain, predetermined structure. One of the directories, called `webapps`, contains one or more web applications, also known as *contexts*.

Like all web servers, Tomcat receives requests formatted either to the standards of the *http* protocol or to the encrypted *https* protocol. The latter is also known as the *secure socket layer (SSL)*. A Tomcat instance can designate one tcp port for http and another one for https. Most often, port 80 is used for http and port 443 for https, but any other port numbers can be used instead. For Spk, we use 80 and 443 for the production ports; 8080 and 8443 for test ports.

From a reliability point of view, it is absolutely essential to have a web for testing that is identical to the web server used for production. The simplest way to achieve this is to have a production instance and a test instance of the same software running on the same machine. This is the approach that we take with Spk.

Most of the Spk web site must be run in the secure SSL mode, in order to protect the privacy of patient data and proprietary nature of some of the PK models. We absolutely do not want users to be able to access certain web pages using only the ordinary http protocol. With Tomcat, unfortunately, there is no way (or at least none that we have discovered) to keep a user from accessing, with http, the pages intended only for https, if both http and https ports are configured for the same instance. For this reason, an Spk installation requires four instances, rather than two.

The four instances are:

1. Production Instance, receiving http requests on port 80
2. Production SSL Instance, receiving https requests on port 443
3. Test Instance, receiving http request on port 8080
4. Test SSL Instance, receiving https requests on port 8443

When changes are made to a Java web application, it is very easy to deploy the modified application to a Tomcat instance if the application is organized in a certain predetermined hierarchy of directories and files. A special compressed archive format, known as a Web Archive (war), has been defined for compressing such an archive into a single file, for easy transport and storage. These files normally carry a `.war` file name suffix, and are very similar to the more familiar `.zip`, `.tar` and, especially, `.jar` files.

When a `.war` file is copied into the `webapps` directory of a Tomcat instance, it is automatically parsed, translated, compiled and linked by Tomcat itself, replacing the existing version of the application, without any need to interrupt operations.

The web portion of Spk is divided into two web applications:

1. *init*, which consists of static content, informative to everyone with an interest in Spk. The authorship and maintenance of this application is primarily the responsibility of the RFPK Science Team.
2. *user*, which consists of dynamic content, including user login, MDA launch, status and historical information about the jobs of a particular user. The *user* application is the responsibility of the RFPK Software Team.

Both the *init* context and the *user* context are accessible exclusively via https. The archives `init.war` and `user.war` must be installed in the `webapps` directory of either of the production SSL instance or the test SSL instance, but never in the other two instances.

The two non-SSL instances contain only a trivial application which redirects the user's browser to the *init* context on the associated SSL instance. This redirection is transparent and immediate, and should not normally be noticed by the user. The redirection, however, allows users to initiate contact with Spk via the standard non-secure URL http://spk.rfpk.washington.edu [2]. This application can be found in the `ROOT` context in the `webapps` directory of the production and test non-SSL instances.

Installing Tomcat

In a terminal window, as an ordinary user, download the tomcat tarball from whitechuck, then **su** to become *root*:

```
scp 'whitechuck:/opt/download/jakarta-tomcat*.tar.gz' .
su
```

If you do not already have a directory called `/usr/local/tomcat`, create one:

```
mkdir /usr/local/tomcat
```

Move the tarball to `/usr/local/tomcat`, and expand it:

```
mv jakarta-tomcat*.tar.gz /usr/local/tomcat
cd /usr/local/tomcat
tar xvzf *
```

Defining Environment Variables

It is convenient to define an environment variable, CATALINA_HOME, which contains the path to the directory in which tomcat is installed. Edit your `~/.bash_profile` file, adding lines to define CATALINA_HOME and to export it.

For example, suppose that the tarball expanded to `jakarta-tomcat-5.0.18`. Then you would add these lines to `~/.bash_profile`:

```
CATALINA_HOME=/usr/local/tomcat/jakarta-tomcat-5.0.18
export CATALINA_HOME
```

After saving your modified `~/.bash_profile`, restart your desktop by logging out and then logging back in again.

Creating Four Run-Time Instances

In this section, we will set things up so that four independent instances of tomcat can run simultaneously. As explained above, the four instances are production, test, production SSL and test SSL.

The following steps will create the file hierarchies for the two instances:

```
su
cd $CATALINA_HOME/..
mkdir instance
cd instance
mkdir prod
mkdir test
mkdir prodssl
mkdir testssl
cp -r $CATALINA_HOME/conf prod
cp -r $CATALINA_HOME/conf test
cp -r $CATALINA_HOME/conf prodssl
cp -r $CATALINA_HOME/conf testssl
cd ../prod
mkdir logs temp webapps work
cd test
mkdir logs temp webapps work
cd ../prodssl
mkdir logs temp webapps work
cd testssl
mkdir logs temp webapps work
```

Configuring User Groups and Permissions

At present, it is not possible to run the production instance or the production SSL instance with any user other than root, because only root can open ports with numbers less than 1024. It might be more secure to run under an ordinary user, as does Apache, but that capability is not yet available. On the other hand, Tomcat may be less vulnerable to certain exploits, such as buffer overflows, because of Java's built-in protection of against such problems.

Although root must be the tomcat process owner, it is preferable for its files to belong to a user or users other than root. That way, ordinary users can be given the authority to modify these files without having to know the root password.

Note that in the following, we are using a concept called *user private groups.* We use the same set of conventions for ownership and permissions in the CVS repository. For additional information, see the RedHat Enterprise Linux Reference Guide [3].

First we create a user and a group. The user *tomcat* will be the owner of the files, while the group *webadmin* will have group access to them.

```
su
/usr/sbin/useradd -r tomcat
/usr/sbin/groupadd -r webadmin
/usr/bin/gpasswd -a tomcat webadmin
```

At this point, you should also add *your user name* to the webadmin group:

```
/usr/bin/gpasswd -a your-user-name webadmin
```

Now set permissions on the file hierarchies for the instances:

```
cd $CATALINA_HOME/../instance
chown -R tomcat.webadmin *
find . -type d -exec chmod 2775 {} \;
find . -type f -exec chmod g+rw {} \;
```

Configuring the Instances

In each instance, we need to make changes to `server.xml`, which is the principal configuration file for tomcat.

Changing the Shutdown Port

In the default configuration, tomcat receives shutdown requests on port 8005. We will leave 8005 as the shutdown value for the test server, but select different ports for each of the other instances.

You should be able to make the following changes as an ordinary user, assuming that you have set up your permissions correctly, as describe above.

```
cd /usr/local/tomcat/instance
```

Before editing, each of the files `*/conf/server.xml` will look like this:

```
<Server port="8005" shutdown="SHUTDOWN" debug="0">
```

Edit `prod/conf/server.xml` to look like this:

```
<Server port="8006" shutdown="SHUTDOWN" debug="0">
```

Edit `testssl/conf/server.xml` to look like this:

```
<Server port="8007" shutdown="SHUTDOWN" debug="0">
```

Edit `prodssl/conf/server.xml` to look like this:

```
<Server port="8008" shutdown="SHUTDOWN" debug="0">
```

Configuring Request Ports for the Instances

In the `server.xml` configuration files, a tcp port and its attributes is defined by an element called *Connector*. In the following subsections, we edit each of these files to specify either one non-SSL connector or one SSL connector.

Non-SSL Connectors

Before editing, the non-SSL connector definition in each of the `server.xml` files looks like this:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="8080"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" redirectPort="8443" acceptCount="100"
   debug="0" connectionTimeout="20000"
   disableUploadTimeout="true" />
```

The file `test/conf/server.xml` does not need to be changed. The default values fit its requirements. The `server.xml` files for the other three instances need to be changed, however, as indicated in the next several paragraphs.

`prod/conf/server.xml` after editing:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 80 -->
<Connector port="80"
          maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
          enableLookups="false" redirectPort="443" acceptCount="100"
          debug="0" connectionTimeout="20000"
          disableUploadTimeout="true" />
```

Note that the port number in the comment was changed from 8080 to 80, and the value of the redirectPort attribute was changed from 8443 to 443.

`prodssl/conf/server.xml` after editing:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<!--
<Connector port="80"
          maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
          enableLookups="false" redirectPort="443" acceptCount="100"
          debug="0" connectionTimeout="20000"
          disableUploadTimeout="true" />
-->
```

Note that the entire Connector element has been commented out.

`testssl` after editing:

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<!--
<Connector port="80"
          maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
          enableLookups="false" redirectPort="443" acceptCount="100"
          debug="0" connectionTimeout="20000"
          disableUploadTimeout="true" />
-->
```

Note here, too, the entire Connector element has been commented out.

SSL Connectors

Before editing, the element for defining the SSL connector will look like this in all four `server.xml` files. Note that the Connector element is commented out.

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<!--
<Connector port="8443"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" disableUploadTimeout="true"
   acceptCount="100" debug="0" scheme="https" secure="true"
   clientAuth="false" sslProtocol="TLS" />
-->
```

The files `prod/conf/server.xml` and `test/conf/server.xml` do not have to be changed, because they should not have SSL connectors, hence the fact that the SSL Connector element is commented out by default fits their requirements. The remaining two `server.xml` files need to be edited as shown in the following paragraphs.

`prodssl/conf/server.xml` after editing:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 443 -->

<Connector port="443"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" disableUploadTimeout="true"
   acceptCount="100" debug="0" scheme="https" secure="true"
          keystoreFile="/root/.keystore" keystorePass="2bon2btit?"
   clientAuth="false" sslProtocol="TLS" />
```

Note that the comment brackets have been removed, and the port number has been changed from 8443 to 443 in both the introductory comment and as the value of the *port* attribute. Not also that new two entries: `keystore="/root/.keystore" keyPass="2bon2btit?"`.

`testssl/conf/server.xml` after editing:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->

<Connector port="8443"
   maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
   enableLookups="false" disableUploadTimeout="true"
   acceptCount="100" debug="0" scheme="https" secure="true"
          keystore="/root/.keystore" keyPass="2bon2btit?"
   clientAuth="false" sslProtocol="TLS" />
```

Note that only the comment brackets have been removed. Note also that new two entries: `keystore="/root/.keystore" keyPass="2bon2btit?"`.

Disabling the Apache Connector for All Instances

For all four instances, we will comment out the element in `server.xml` which defines a connector between tomcat and apache, because we are using tomcat as a complete stand-alone server that does not depend on apache.

Before editing, all four files files,

- `prod/conf/server.xml`
- `test/conf/server.xml`
- `prodssl/conf/server.xml`
- `testssl/conf/server.xml`

will look like this:

```
<!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
<Connector port="8009"
   enableLookups="false" redirectPort="8443" debug="0"
   protocol="AJP/1.3" />
```

Afterward, all four should look like this:

```
<!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8009 -->
<!--
<Connector port="8009"
   enableLookups="false" redirectPort="8443" debug="0"
   protocol="AJP/1.3" />
-->
```

Note that the Connector element has been commented out.

Setting the Instances to Run Automatically

Tomcat should be run automatically from the **init** process whenever the system boots, and be capable of being stopped or restarted manually from the command line by the *root* user, just like other system services.

In the `r2` CVS repository, in the directory `r2/admin/tomcatd`, there are four shell scripts, called **tomprod**, **tomtest**, **tomprodssl**, and **tomtestssl**.

tomprod looks like this

```
#!/bin/bash
#
# Init file for production tomcat
#
# chkconfig: 2345 63 37
# description: tomcat server for port 80
#

JAVA_HOME=/usr/java/j2sdk1.4.2_03
CATALINA_HOME=/usr/local/tomcat/jakarta-tomcat-5.0.18
CATALINA_BASE=$CATALINA_HOME/../instance/prod
export JAVA_HOME CATALINA_HOME CATALINA_BASE

exec $CATALINA_HOME/bin/catalina.sh $*
```

Note the line

```
# chkconfig: 2345 63 37
```

It contains values for the **/sbin/chkconfig** utility, which manages the configuration of services in the Linux operating system and in many Unix operating systems, so that they are started up and shutdown automatically when the system enters or leaves the appropriate run levels and in the correct order. The line above says that **tomprod** should be started whenever the system enters any of the run levels 2, 3, 4, or 5, and that compared to other services, its priority for starting is 63. It also says that upon leaving any of the given run levels, the instance should be stopped with priority 37.

The other three scripts differ from the one above only in the start priority. The starting order of the four instances are the following:

- 63: tomprod
- 64: tomtest
- 65: tomprodssl
- 66: tomtestssl

Whenever the system reboots, the four instances are started in the above order. If you use the command pipeline **ps -ef | grep tom** to show their run status, you will see them in the order that they were started. This can be useful if, for some reason, the **kill** command has to be used on one of them.

To install the four scripts, they must be copied from the CVS repository to the directory `/etc/rc.d/init.d` and then added to the system initialization structures with the help of **/sbin/chkconfig**.

```
cd /tmp
cvs checkout r2
cvs update -dP
cd r2/admin/tomcatd
chmod +x tom*
su
cp tom* /etc/rc.d/init.d
/sbin/chkconfig --add tomprod
/sbin/chkconfig --add tomtest
/sbin/chkconfig --add tomprodssl
/sbin/chkconfig --add tomtestssl
cd /tmp
rm -rf r2
```

Setting Authentication Certificates

Installing for The First Time

If you have not obtained the certificates that tomcat will need for https connections, get them by following the instructions in the Authentication and Encryption Howto [4]. If you obtained the certificates on a workstation rather than on the web server, transfer a copy of the `cert` directory from the ~root directory on the workstation to the ~root directory on the server. To install the the certificates, do the following:

```
su
cd cert
$JAVA_HOME/bin/keytool -import -alias root -keystore keystore -trustcacerts -file uw_ro
$JAVA_HOME/bin/keytool -import -alias tomcat -keystore keystore -trustcacerts -file spk
```

Renewal

You are going to request a new SPK certificate from C& and replace the old entry in `/root/.keystore`. You do not need to do anything about the UW root certificate.

First, generate a certificate request from the existing private/public key pair as following:

```
su -
cd cert
$JAVA_HOME/bin/keytool -certreq -alias tomcat -keyalg RSA -file certreq.csr
```

A certificate request, `certreq.csr`, is generated using the existing key pair in `/root/.keystore`, which is the default location that keytool looks for, in `cert` directory. WARNING! It is very & very important to have the key pair and the requested certificate matched!

Submit the request to C&C, following the procedures described in Request a Signed Certificate[5] and Retrieve your Certificate[6]. Once you retrieved the SPK certifiate, do the following to replace the old certificate entry registered in `/root/.keystore`.

```
$JAVA_HOME/bin/keytool -delete -alias tomcat
$JAVA_HOME/bin/keytool -import -alias tomcat -trustcacerts -file spk_cert.pem
```

The second command, i.e. import, will issue a message saying the certifiate you imported was determined as a reply to the certificate request you generated earlier.

Deploying the Spk Web Applications

On your development workstation, do the following process in the directory that contains your working copy of the `r2` CVS repository.

```
cd r2/src/apps/spk/webapps
deploy info prod
deploy info test
deploy user prod
deploy user test
cd ROOT
jar cvf ROOT.jar *
scp ROOT.jar 192.168.2.2:/usr/local/tomcat/instance/prod/webapps
scp ROOT.jar 192.168.2.2:/usr/local/tomcat/instance/test/webapps
```

Now you should be ready to start tomcat manually, and to verify that the installation is working.

```
su
/etc/rc.d/init.d/tomprod start
/etc/rc.d/init.d/tomtest start
/etc/rc.d/init.d/tomprodssl start
/etc/rc.d/init.d/tomtestssl start
```

should get all four instances of tomcat running. You can check that that they are running by using the **ps -ef | grep tom** command pipeline.

Once the four instances are running, you can test that the web applications are accessible from the web. If you are outside the firewall, try the URL

https://spk.rfpk.washington.edu/info/ [7]

https://spk.rfpk.washington.edu:8443/info/ [8]

to access *info* application running in the production and the test SSL instances, respectively.

If you are behind the firewall, the corresponding URLs are

https://192.168.2.2/info/ [9]

https://192.168.2.2:8443/info/ [10]

To access the *user* web application, click the **MySpk** link on any of the pages of the *info* application.

The *ROOT* application, which does nothing more than redirect non-SSL requests to the associated SSL instance, can only be tested from outside of the firewall. The URLs http://spk.rfpk.washington.edu [11]

http://spk.rfpk.washington.edu:8080 [12]

access the production and test instances respectively.

Stopping and Starting

Tomcat is a daemon. It will run until the *root* user tells it to stop, or until the computer is rebooted.

To tell, for example, the test SSL tomcat to stop (as root):

```
/etc/rc.d/init.d/tomtestssl stop
```

Before you try to restart an instance that you have commanded to stop, be sure that it has actually terminated by using the piped command **ps -ef | grep tom** and counting entries. Sometimes it can take a while for an instance to clean up everything that it needs to before quitting. If it has not died after several minutes, you can use the **kill -9 pid** command, where *pid* is the process id provided by **ps**. Because instances are started at boot time in the order **prod**, **test**, **prodssl**, **testssl**, and hence are assigned monotonically increasing pids, it should ordinarily be possible to decide which process to kill. If an instance has already been stopped and restarted, however, it will have a pid greater than any of the processes that have been running since boot time. Whenever you restart an instance, it is a good idea to write down its pid, in order not to be confused later on. In practice, **testssl** is the most likely instance to need restarting, and in that case there will not be a problem, because of the four instances, **testssl** has the greatest pid after boot.

Notes

1. ../authentication/authentication.html
2. http://spk.rfpk.washington.edu
3. https://www.redhat.com/docs/manuals/enterprise/RHEL-3-Manual/ref-guide/s1-users-groups-private-groups.html
4. ../authentication/authentication.html
5. file://192.168.2.2:8080/soft/howto/authentication/authentication.html#AEN77
6. http://192.168.2.2:8080/soft/howto/rhel3/authentication/authentication.html#AEN81
7. https://spk.rfpk.washington.edu/info/
8. https://spk.rfpk.washington.edu:8443/info/
9. https://192.168.2.2/info/
10. https://192.168.2.2:8443/info/
11. http://spk.rfpk.washington.edu
12. http://spk.rfpk.washington.edu:8080