

Author: Jon Barnett
Date: 17 July 2003

1 Linux Apache 2.0 and JBoss 3.2.x JK2 configuration

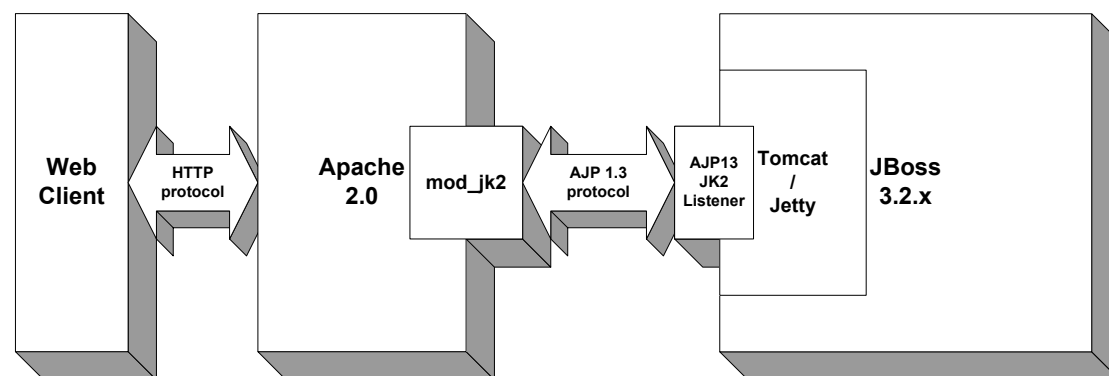
Questions are common about the integration of Apache 2.0 with JBoss 3.2.x embedded servlet containers. This short technical note provides the basics for connecting a Linux-based Apache 2.0 installation to a JBoss 3.2.x embedded servlet container. JBoss currently supports an embedded Tomcat or Jetty servlet container.

1.1 Concepts

The primary reason for connecting a web server to the Tomcat or Jetty servlet container is to enable static web content to be provided by the web server since the web server is faster for this task while still allowing dynamic content to be delivered by the servlet container. Both Tomcat and Jetty support delivery of static content so unless there are strong technical reasons for requiring a separate static content server, this additional delay element for dynamic pages should be avoided.

Should there be a need to combine the two systems for the purpose previously described; there are a few initial concepts that are important to successfully integrating the two systems.

1. JK2 is a refactoring of JK but we will not describe here the configuration elements for the Apache `mod_jk` installation.
2. JBoss 3.2.x manages the embedded servlet container as a service in the JBoss microkernel and manages the startup and the shutdown of the servlet container.
3. Apache will connect to the servlet container using the AJP13/JK2 connector from the `mod_jk2` module.
4. Apache will have a configuration file that defines the conditions under which Apache will pass a web request to the Tomcat or Jetty servlet container to service.
5. As with the standard HTTP and HTTPS listeners, the servlet container will passively await requests made to the AJP13/JK2 listener.



1.2 Configuring embedded Tomcat or embedded Jetty

By default, the Tomcat or Jetty AJP13 connectors are enabled in the JBoss 3.2.x distribution. Their connector configurations are located in `server/instance/deploy/jbossweb-tomcat.sar/META-INF/jboss-service.xml` and `server/instance/deploy/jbossweb-jetty.sar/META-INF/jboss-service.xml` respectively, where *instance* is the JBoss instance started via the `run.bat` or `run.sh` script. Normally, this instance is *default*. Some of the newer JBoss 3.2.x releases may have the instance *all* as the default. Also, the Tomcat deployment may have moved to `jbossweb-tomcat41.sar`.

AJP13/JK2 is simple with a JBoss 3.2.x embedded Jetty/Tomcat. You will not need the Java Native Interface (JNI). Apache will not be starting Tomcat within an Apache process. The embedded container is under the control of JBoss.

Similarly, Java Tomcat Connectors (JTC) is not a necessary consideration for this installation, at least from the servlet container perspective. The required listeners for AJP13/JK2 are already installed for the embedded Tomcat and the embedded Jetty.

Tomcat or Jetty, embedded in JBoss is a passive component in this configuration. It listens for incoming AJP13/JK2 requests and services those requests. There is nothing to be done on the servlet container side of things except configuring the AJP13/JK2 socket listener. By default, these are enabled in the distribution. If you are using only JK2, you may want to disable the standard HTTP/HTTPS listeners. The configuration files will be located in `server/instance/deploy/jbossweb-jetty.sar/META-INF/jboss-service.xml` and `server/instance/deploy/jbossweb-tomcat.sar/META-INF/jboss-service.xml` respectively for Jetty and Tomcat.

Jetty has the following:

```
<!-- - - - - - -->
<!-- Add and configure a HTTP listener to port 8080 -->
<!-- The default port can be changed using: java -Djetty.port=80 -->
<!-- - - - - - -->
<Call name="addListener">
  <Arg>
    <New class="org.mortbay.http.SocketListener">
      <Set name="Port"><SystemProperty name="jetty.port" default="8080"/></Set>
      <Set name="MinThreads">10</Set>
      <Set name="MaxThreads">100</Set>
      <Set name="MaxIdleTimeMs">30000</Set>
      <Set name="LowResourcePersistTimeMs">5000</Set>
    </New>
  </Arg>
</Call>

<!-- - - - - - -->
<!-- Add a HTTPS SSL listener on port 8843 -->
<!-- - - - - - -->
<!-- UNCOMMENT TO ACTIVATE -->
<Call name="addListener">
  <Arg>
    <New class="org.mortbay.http.SunJsseListener">
      <Set name="Port">8843</Set>
      <Set name="MinThreads">5</Set>
      <Set name="MaxThreads">100</Set>
      <Set name="MaxIdleTimeMs">30000</Set>
      <Set name="LowResourcePersistTimeMs">2000</Set>
      <Set name="Keystore"><SystemProperty
        name="jboss.server.home.dir"/>/conf/demokeystore</Set>
      <Set name="Password">OBF:1vny1z1olx8e1vvn1vn61x8g1z1ulvn4</Set>
      <Set name="KeyPassword">OBF:lu2ulwml1z7s1z7a1wn1lu2g</Set>
    </New>
  </Arg>
</Call>
-->

<!-- - - - - - -->
<!-- Add a AJP13 listener on port 8009 -->
<!-- This protocol can be used with mod_jk in apache, IIS etc. -->
<!-- - - - - - -->
<Call name="addListener">
  <Arg>
    <New class="org.mortbay.http.ajp.AJP13Listener">
      <Set name="Port">8009</Set>
      <Set name="MinThreads">5</Set>
      <Set name="MaxThreads">20</Set>
      <Set name="MaxIdleTimeMs">0</Set>
      <Set name="confidentialPort">443</Set>
    </New>
  </Arg>
</Call>
```

Note the AJP13 port is defined to be 8009. Also note that the comment is incorrect - it is for `mod_jk2` and `mod_jk`.

Tomcat has the following:

```
<!-- A HTTP/1.1 Connector on port 8080 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8080" minProcessors="3" maxProcessors="10"
  enableLookups="true" acceptCount="10" debug="0"
  connectionTimeout="20000" useURValidationHack="false" />

<!-- A AJP 1.3 Connector on port 8009 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8009" minProcessors="5" maxProcessors="75"
  enableLookups="true" redirectPort="8443"
  acceptCount="10" debug="0" connectionTimeout="20000"
  useURValidationHack="false"
  protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>
```

Note the AJP13 port is defined to be 8009.

Check your respective configurations and make sure you know where your AJP13 service is configured, and the port to which it is bound. The default in the JBoss 3.2.x distributions is for the AJP13/JK2 listeners to be enabled.

If it is not commented out, you will have nothing to do for the JBoss 3.2.x part of the install. Just make sure that JBoss is started before you start the Apache mod_jk2 service.

1.3 Building and installing mod_jk2 for Linux Apache 2.0

The module, mod_jk2 is a plug-in for Apache 2.0 that provides AJP13/JK2 connectivity so Apache can pass content requests to a supporting content server such as Tomcat or Jetty. The module documentation indicates it will work with Apache 1.3 but Apache 1.3 suffers from scalability issues in a multi-threaded, multi-processor environment and can prevent a server configuration taking full advantage of the JK2 enhancements. We would recommend using Apache 2.0 but only after testing performance first.

Get the JK2 module source from the Jakarta Apache website - there are no binaries. The URL is <http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk2/release/>. You will also need a Tomcat 4.1.x standalone distribution installed on the Linux system where you will be building mod_jk2. This is required as the build process needs to locate some of the Tomcat libraries. You can get the latest 4.1.x distribution from <http://www.apache.org/dist/jakarta/tomcat-4/binaries/>.

Unpack your Tomcat 4.1.x binary distribution on the Linux system where your Apache install is located so you can reference the Tomcat directory for the mod_jk2 build process. Do not start or use this Tomcat installation. It is installed only to allow the JK2 build process to complete. Unpack the JK2 source in your Linux build directory. Ensure that you can also reference the Apache directory.

Modify jk/native2/configure of the JK2 distribution as it tests whether the apache2 directory is a file.

The script fragment is this:

```
# Check whether --with-apache2 or --without-apache2 was given.
if test "${with_apache2+set}" = set; then
  withval="${with_apache2}"

  case "${withval}" in
    ""|"yes"|"YES"|"true"|"TRUE")
      ;;
    "no"|"NO"|"false"|"FALSE")
      echo "$as_me: error: valid apr source dir location required" >&2;
      { (exit 1); exit 1; }; }
      ;;
    *)
      tempval="${withval}"

      if ${TEST} ! -d ${tempval} ; then
        { { echo "$as_me:$LINENO: error: Not a directory: ${tempval}" >&5
        echo "$as_me: error: Not a directory: ${tempval}" >&2; }
        { (exit 1); exit 1; }; }
      fi

      if ${TEST} ! -f ${tempval};; then
```

The last line (line 8146 in the quoted distribution) should be replaced with this:

```
if ${TEST} ! -d ${tempval};; then
```

We are only interested in building the plugin for Apache 2.0 so we will skip most of the build. Go to `jk/native2` of the distribution and run the following:

```
./configure --with-tomcat40=/usr/local/tomcat-4.1 \
  --with-tomcat41=/usr/local/tomcat-4.1 \
  --with-apache2=/usr/local/apache2 --with-apxs2=/usr/local/apache2/bin/apxs
```

Use appropriate values for the location of your Apache install and the location of your Tomcat 4.1.x install. This will create the build configuration for the Apache 2.0 plugins.

When the configuration process has completed, just run *make* from the same directory - `jk/native2`:

```
make
```

Copy the completed files *.so from the build directory of the JK2 distribution to your Apache installation's *modules* directory. If you are still in the directory you performed the build (via make) and using the Apache install directory from my example, it would be:

```
cp ../build/jk2/apache2/mod*.so /usr/local/apache2/modules
```

Modify this for your requirements. It should copy *jkjni.so* and *mod_jk2.so* to your Apache modules directory. Now, just run *libtool* as per the instructions from the build. Again from this case example:

```
libtool --finish /usr/local/apache2/modules
```

Apache now has `mod_jni` and `mod_jk2` installed. Remember though that we don't want to run JNI.

1.4 Configuring `mod_jk2` for Linux Apache 2.0

With `mod_jk2` installed, you now just need to configure `conf/workers2.properties` in your Apache 2.0 installation and locate the module. The simplest installation consists of this:

```
[logger]
level=DEBUG

[config:]
file=${serverRoot}/conf/workers2.properties
debug=0
debugEnv=0

[uriMap:]
info=Maps the requests. Options: debug
debug=0

# Define the communication channel
[channel.socket:172.16.32.26:8009]
info=Ajpl3 forwarding over socket
tomcatId=172.16.32.26:8009

# Map the Tomcat examples webapp to the web server uri space
[uri:/test1/*]
info=Map the whole webapp

[shm:]
info=Scoreboard. Required for reconfiguration and status with multiprocess servers
file=${serverRoot}/logs/jk2.shm
size=1000000
debug=0
disabled=0
```

Note that I'm connecting via a socket to the JBoss/Tomcat or JBoss/Jetty AJP13/JK2 service located at the address 172.16.32.26 at port 8009. Change this to point to the location of your JBoss installation. There is expected to be a web application context on Tomcat/Jetty at `/test1` in this example. Map your URIs according to your requirements.

You can look up the meanings of the entries at <http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk2/release/v2.0.0/doc/jk2/configwebcom.html>.

There is also an example with the source distribution. It adds the JNI elements, which is just going to confuse most people who don't want Apache to start and stop Tomcat in an in-process manner. This is not going to work with a JBoss embedded web server as the system would need to start JBoss in an in-process manner and the `mod_jk2` interface has no definition of how to achieve this.

Remember to add the `LoadModule` directive for `mod_jk2` in your `httpd.conf` for Apache 2.0.

```
LoadModule jk2_module modules/mod_jk2.so
```

Make sure that your JBoss server is up and running. Restart Apache 2.0 and you should be able to get content from your servlet container.

In the example given here, we would be testing that the servlet container was delivering content for the request, `http://apacheserver/test1`.

That is all that is required for connecting via AJP13.

1.5 Configuring Tomcat for `jk2.properties`

Currently, Tomcat doesn't support embedded operation that well. Certain aspects such as locating the `jk2.properties` file are problematic. In most cases, you shouldn't need the `jk2.properties` file so this should not affect too many people.

A patch has been submitted to the Tomcat build but at this stage, it is uncertain when a build will be released containing that patch. If you wish to apply the patch yourself and build Tomcat, this is a copy of the submitted patch. It patches the `CoyoteConnector.java` code to use the JBoss home directory as the JK home directory.

```
    }
    protocolHandler.setAdapter(adapter);
-
+   IntrospectionUtils.setProperty(protocolHandler, "jkHome",
+   String jbosshome = System.getProperty("jboss.server.home.dir");
+   if (jbosshome != null) {
+       // set jkHome to jboss server home dir.
+       // (ex. $JBOSS_HOME/server/default)
+       IntrospectionUtils.setProperty(protocolHandler, "jkHome",
+                                     jbosshome);
+   } else {
+       IntrospectionUtils.setProperty(protocolHandler, "jkHome",
+                                     System.getProperty("catalina.base"));
-
+   }
+   // Set attributes
+   IntrospectionUtils.setProperty(protocolHandler, "port", "" + port);
+   IntrospectionUtils.setProperty(protocolHandler, "maxThreads",
```

With this patch, the expected location of `jk2.properties` would be `JBOSS_HOME/server/instance/conf/jk2.properties`.

An alternative to this involves adding some definitions to `server/instance/deploy/jbossweb-tomcat.sar/META-INF/jboss-service.xml`. The additions define `CatalinaBase` and `CatalinaHome` and are inserted into the MBean definition for the embedded Catalina service. This change may cause unanticipated side-effects so use this with caution.

```
<mbean code="org.jboss.web.catalina.EmbeddedCatalinaService41"
  name="jboss.web:service=WebServer">

  <attribute name="CatalinaHome">/usr/local/jboss/server/default/temp</attribute>
  <attribute name="CatalinaBase">/usr/local/jboss/server/default/temp</attribute>
```

When you start JBoss, it will create the specified directory with a `conf` sub-directory and a `work` sub-directory. You can also manually create the directories to avoid starting JBoss purely to create the directories. You place the `jk2.properties` file in the `conf` sub-directory. In this example, the full path would be `/usr/local/jboss/server/default/temp/conf/jk2.properties`.

1.6 Installing mod_jk2 for Windows Apache 2.0

There are some differences for installation on Apache 2.0 for Windows. The first thing is to obtain the Windows module for JK2 or build it from the source. Since we don't use Microsoft development tools, we cannot provide any specific information on the build process. We downloaded a working module from <http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk2/nightly/win32/>.

Place the mod_jk2.dll in the *APACHE2_HOME/modules* directory.

Remember to add the LoadModule directive for mod_jk2 in your *httpd.conf* for Apache 2.0.

```
LoadModule jk2_module modules/mod_jk2.dll
```

Some people have experienced problems due to having an incorrect module installed – *mod_jk* instead of *mod_jk2*. Whether this is due to a misnaming of the DLL is something we cannot verify. The module is the incorrect one if Apache reports the following:

```
No worker file and no worker options in httpd.conf  
use JkWorkerFile to set workers
```

Check the DLL for a string like “mod_jk2/2.0.0”. This is a good indication that the module is the mod_jk2 variant.

The configuration of the *workers2.properties* file is as per the instructions provided earlier.

1.7 Logging management for Tomcat

Logging for the Tomcat JK2 connector can be controlled through the *server/instance/conf/log4j.xml* configuration for your JBoss instance. For the relevant logger, insert the following filter:

```
<!-- ===== -->  
<!-- Limit categories -->  
<!-- ===== -->  
  
<category name="org.apache.jk">  
  <priority value="WARN"/>  
</category>
```

This can remove some verbose informational dumps generated by the Tomcat JK2 connector. The Jetty JK2 listener does not seem to generate the same level of information up to the current Jetty 4.2.14 plugin at the time of writing.

1.8 JK and JK2

Although JK2 is a re-factoring of the JK code and the listeners in the servlet containers are not affected by connection from mod_jk or mod_jk2, there are differences in the supporting configuration files, particularly on the Apache web server. This is the primary issue with ensuring that the correct modules are installed and configured.