

Author: Jon Barnett
 Date: 12 August 2003

1 Axis 1.1 and JBoss 3.2.x configuration

JBoss 3.2.x provides a bundled Web services implementation. This bundled service is called JBoss.NET and incorporates Axis as a plug-in service for the JBoss microkernel. Deployment of Web services uses the built-in JBoss deployment system that understands the WSR packaging.

However, some users may wish for a standard Axis deployment since this is supported across most application server platforms. At this time, not many other application servers support a WSR deployment and this hinders portability of a Web service implementation from JBoss to other application servers. This short technical note provides the basics for configuring a JBoss 3.2.x installation for Axis 1.1. For this installation to function, you will need an embedded servlet container such as Jetty or Tomcat.

Axis is an implementation of the Simple Object Access Protocol (SOAP). SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment and the protocol is XML-based. The protocol consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

1.1 Concepts

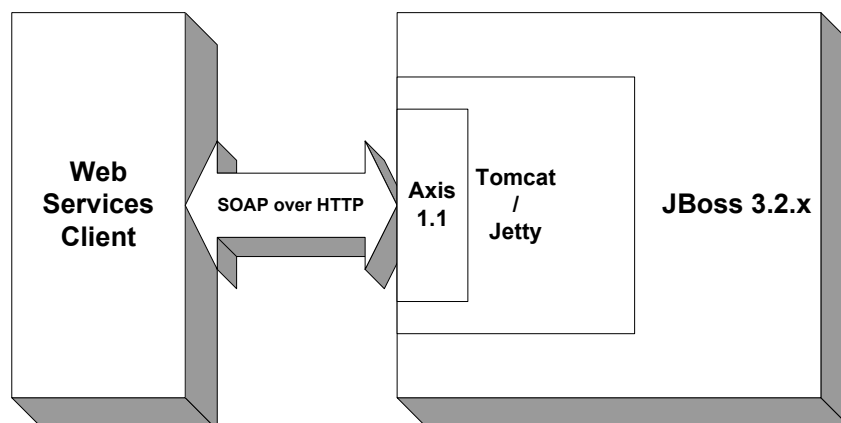
Axis 1.1 can deliver Web services over HTTP or over JMS as described in the SOAP protocol binding definition. It is for this reason that you will require an embedded servlet container to deliver the content of a deployed Web service over HTTP. JBoss 3.2.x also provides a JMS implementation so you may use this as an alternative delivery mechanism for your Axis-based Web service.

The packaging of Axis 1.1 is designed for deployment in a servlet container such as Tomcat. The Axis service is intended to be a web application, delivering the Web services typically over HTTP and also managing the publishing and directory listings over HTTP.

Axis can convert an EJB into a Web service by providing the interface plumbing and the service delivery. More information on Axis can be found here - <http://ws.apache.org/axis/>.

1. JBoss 3.2.x manages the embedded servlet container as a service in the JBoss microkernel.
2. JBoss 3.2.x is not directly involved in serving the content over HTTP to the Web services client.
3. The JBoss 3.2.x JMS is involved in serving the content over JMS to the Web services client.

A typical Web service over HTTP deployment within a JBoss context would appear as shown below.



1.2 Installing Axis 1.1

Install and configure your JBoss 3.2.x distribution. It is assumed that you have a basic mastery of JBoss 3.2.x configuration and you can successfully perform the basic operational installation of JBoss 3.2.x. The installation of Axis is largely independent of the embedded servlet container that you use – it can be either Jetty or Tomcat.

Axis 1.1 can be obtained from http://ws.apache.org/axis/dist/1_1/. You can either obtain the Axis 1.1 source and build the binary distribution from the source or obtain the binary distribution directly.

In either case, the end result will include a *webapps* directory. Within this is a subdirectory called *axis*. Copy this subdirectory to the appropriate deployment directory for your JBoss 3.2.x run-time instance, **JBOSS_HOME/server/instance/deploy** where *instance* is the run-time instance. Normally this would be **JBOSS_HOME/server/default/deploy** but adjust this for your situation. Change the subdirectory name from *axis* to *axis.war*. This allows JBoss 3.2.x to identify the deployment as a web application and appropriately delegate run-time deployment to the servlet container.

The Axis 1.1 includes most of the necessary libraries required to support Web services into the WEB-INF/lib directory of the WAR. These include saaj.jar and jaxrpc.jar. In a standalone Tomcat 4.1.x distribution, these libraries would need to be situated **CATALINA_HOME/common/lib** due to the Tomcat constraints for loading classes in the java.* and javax.* hierarchy. Notes regarding this were included in the earlier Axis 1.1 Release Candidates (RCs). With JBoss 3.2.x, users have discovered that Axis will not correctly deploy if saaj.jar and jaxrpc.jar are contained in the **axis.war/WEB-INF/lib** directory while using the standard Java 2 compliant class-loading scheme in JBoss.

We have been using Axis 1.1 in both JBoss and Tomcat standalone distributions since RC 1, and have adopted a standard distribution with saaj.jar and jaxrpc.jar located with the standard Tomcat/Jetty libraries. Switching to a JBoss context, we moved saaj.jar and jaxrpc.jar to **server/instance/deploy/jbossweb-tomcat.sar** for JBoss-Tomcat distributions and to **server/instance/deploy/jbossweb-jetty.sar** for JBoss-Jetty distributions. You may need to adjust this for naming changes in later releases. We have had no problems with Axis 1.1 in JBoss 3.2.x with this configuration.

The Axis 1.1 WAR also includes log4j-1.2.8.jar in WEB-INF/lib. Since JBoss 3.2.x already has variants of the Apache log utilities, we omit the library from the JBoss 3.2.x deployment of Axis 1.1. This saves having extra classes loaded in memory and prevents any possible confusion. We have not experienced any problems to date with any Web services delivered in this configuration.

Following these steps will create an operational, vanilla Axis 1.1 service. You may tailor your installation by removing the sample classes and associated Web service definitions for the samples from the Axis installation. You can check the installation of components by using the Happy Axis test page. This is bundled as part of the Axis WAR. You can access it locally on your JBoss system via <http://localhost:8080/axis/happyaxis.jsp>. This test page will notify you if there are any missing components. There should not be any problems with missing libraries since JBoss includes most, if not all of the optional libraries for Axis 1.1.

For those who wish to remove the samples from the Axis 1.1 distribution:

1. Remove the **axis.war/WEB-INF/classes/samples** subdirectory
2. Remove the **axis.war/*.jws** Web service definitions

An alternative approach to moving the saaj.jar and jaxrpc.jar libraries is to disable Java 2 classloading compliance in JBoss 3.2.x for the servlet containers. This is controlled in the META-INF/jboss-service.xml in the SAR for the relevant servlet container – jbossweb-tomcat.sar, jbossweb-jetty.sar or similar. The required setting would be:

```
<attribute name="Java2ClassLoadingCompliance">true</attribute>
```

Should you use saaj.jar or jaxrpc.jar definitions beyond a web application context, you can also locate these either in **JBOSS_HOME/server/instance/lib** or **JBOSS_HOME/lib**, depending on the scope of usage.