What's New in JBoss AS 4.0

The JBoss Application Server (JBoss AS) 4.0 is a production-ready Java 2 Enterprise Edition (J2EE) application server. It builds on top of the highly successful JBoss 3.2 line of open source Java application servers with improved standards compliance and major feature enhancements. JBoss AS 4.0 offers the same level of quality and stability customers have grown to expect from JBoss 3.2. Key features of JBoss AS 4.0 include:

- Officially certified to be fully compliant to the J2EE 1.4 specification. JBoss AS 4.0 is the first production-ready J2EE 1.4 application server in the industry.
- Full support for J2EE Web Services and the Service Oriented Architecture (SOA).
- Supports the Aspect-Oriented Programming (AOP) model for developing middleware solutions. JBoss AOP greatly improves developer productivity.
- Tightly integrates with Hibernate, world's most popular object persistence framework developed by JBoss, inside the application server container.
- Improves clustering and distributed caching support with on a new internal caching architecture.

1. J2EE Certification and Standards Compliance

JBoss AS 4.0 is the industry's first officially certified J2EE 1.4 application server. The certification guarantees that JBoss AS 4.0 conforms to the formal J2EE specification. That allows developers to safely reuse J2EE components (e.g., Enterprise JavaBeans or EJBs) across different application servers. For example, a developer could easily migrate an EJB developed for WebLogic or WebSphere to JBoss. The certification makes JBoss 4.0 a safe upgrading choice for both existing JBoss users and users of other J2EE application servers.

Compared with JBoss AS 3.2, JBoss AS 4.0 implements the following new J2EE specifications in order to be J2EE 1.4 compliant:

- JBoss AS 4.0 supports J2EE Web Services including JAX-RPC (Java API for XML for Remote Procedure Call) and the Web Services for J2EE Architecture, which leverages standard J2EE components (e.g., EJBs) to provide a scalable and secure Web Service environment. It is the basis for implementing SOA in J2EE. The older JBoss.NET Web Services API in JBoss AS 3.2 is no longer supported. The new Web Services implementation is WS BasicProfile-1.0 compliant.
- JBoss AS 4.0 implements the JMS (Java Messaging Service) 1.1 specification instead of the JMS 1.0 in JBoss AS 3.2. In JMS 1.0, client programming for the Point-to-Point and Pub/Sub domains was done using similar but separate class hierarchies. In JMS 1.1, there is now a domain-independent approach to programming the client application.
- JBoss AS 4.0 implements the JCA (Java Connector Architecture) 1.5 specification instead of the JCA 1.0 in JBoss AS 3.2. The JCA 1.5 specification adds support for the life cycle management of resource adapters, worker thread management as well as transaction and message inflow from the resource adapter to the application server.

- JBoss AS 4.0 implements the new Java Authorization Contract for Containers (JACC) specification. JACC is a Java 2 permission-based mechanism for externalizing the authorization decision for accessing EJB methods and web resources. The new implementation is based on the JBoss AS 3.2 semantic of associating the J2EE declarative roles with the authenticated Subject as a by-product of the JAAS authentication phase. JBoss AS 4.0 maintains compatibility with the JBoss AS 3.2 security configuration.
- JBoss AS 4.0 implements the EJB 2.1 specification instead of the EJB 2.0 in JBoss AS 3.2. The EJB 2.1 specification extends the message-driven bean contracts to support other messaging types in addition to JMS. It supports stateless session beans as web service endpoints. It also includes a new container managed service called the EJB timer service.

2. Server Configurations

The `minimal` and `all` configurations in JBoss AS 4.0 have the same meanings as the `minimal` and `all` configurations in JBoss 3.2.

- The `minimal` configuration starts the JBoss microkernel, JMX MBean server and JNDI naming service.
- The `all` configuration starts all services including clustering.

However, the `default` configuration is different from JBoss AS 4.0.1+ and JBoss 4.0.0.

2.1. JBoss AS 4.0.1 and above

In JBoss AS 4.0.1, the `default` configuration is the same as the `default` configuration in JBoss AS 3.2. It starts all J2EE services in JBoss's unified class loader. It has optimized performance, when the components are deployed in the same JVM. But the deployed applications are less compartmentalized in this configuration. As a result, this configuration is not fully J2EE 1.4 compliant. To make it J2EE complaint, you need to change the configuration settings as follows to enable the scoped class loading behavior and call by value JNDI lookup behavior.

First, edit the `conf/jboss-service.xml` file and set the `NamingService` `CallByValue` to `true`:

```
<mbean code="org.jboss.naming.NamingService"
       name="jboss:service=Naming">
    <!-- The call by value mode. true if all lookups are unmarshalled using
      the caller's TCL, false if in VM lookups return the value by reference.
    -->
    <attribute name="CallByValue">true</attribute>

      ...
</mbean>
```

Second, edit the `deploy/ear-deployer.xml` file and set the `Isolated` and `CallByValue` attributes to `true`:

```
<server>
   <!-- EAR deployer, remove if you are not using ear deployments -->
   <mbean code="org.jboss.deployment.EARDeployer"
      name="jboss.j2ee:service=EARDeployer">
```

```
      <!-- A flag indicating if ear deployments should have their own scoped
      class loader to isolate their classes from other deployments.
      -->
      <attribute name="Isolated">true</attribute>
      <!-- A flag indicating if the ear components should have in VM call
      optimization disabled.
      -->
      <attribute name="CallByValue">true</attribute>
   </mbean>
</server>
```

Last, edit the `deploy/jbossweb-tomcat50.sar/META-INF/jboss-service.xml` file and set the `Java2ClassLoadingCompliance` and `UseJBossWebLoader` attributes to `false`:

```
<server>

   <mbean code="org.jboss.web.tomcat.tc5.Tomcat5"
      name="jboss.web:service=WebServer">

      <!-- Get the flag indicating if the normal Java2 parent first class
           loading model should be used over the servlet 2.3 web container first
           model.
      -->
      <attribute name="Java2ClassLoadingCompliance">false</attribute>

      <attribute name="LenientEjbLink">true</attribute>

      <!-- A flag indicating if the JBoss Loader should be used. This loader
           uses a unified class loader as the class loader rather than the tomcat
           specific class loader.
      -->
      <attribute name="UseJBossWebLoader">false</attribute>

        ...
```

That's it. Your JBoss AS 4.0.1 `default` configuration is now fully J2EE 1.4 compliant.

2.2. JBoss AS 4.0.0

In JBoss AS 4.0.0, the situation is a little confusing:

- The `standard` configuration in JBoss AS 4.0.0 has the same meaning as the `default` configuration in JBoss AS 3.2 and JBoss AS 4.0.1+. It is not fully J2EE 1.4 compliant due to the problems we mentioned above.
- The `default` configuration in JBoss AS 4.0.0 is the same as the `standard` configuration except that it is configured for J2EE compatibility.

We recommend you to use JBoss AS 4.0.1+ instead of 4.0.0 whenever possible.

3. New Services Types

JBoss AS 4.0 adds support for new types of server services. The `SARDeployer` now recognizes the `*.deployer` archives (both in expanded directories and in zip files) and the `*-deployer.xml` files as valid deployment options. The `.deployer` suffix is equivalent to the `.sar` suffix, and the `-deployer.xml` file name suffix is equivalent to the `-service.xml` descriptor file name suffix. These suffixes are sorted ahead of any other service types so that these `.deployer` services are started before other services. For example, the JBoss AOP services are deployed as a `.deployer` service archive (i.e., the `jboss-aop.deployer` archive in the `deploy` directory). That makes sure that the JBoss AOP services are started early on in the server start-up process.

4. JBoss AOP Support

Aspect-oriented middleware is a key innovation in JBoss AS 4.0. It drastically simplifies middleware application development and allows developers to extend the container services. In JBoss AS 4.0, you can deploy AOP-based services and applications directly into the application server. A detailed introduction to aspect-oriented programming and the JBoss AOP framework can be found on JBoss web site.

In JBoss AS 4.0.0's `standard` and `all` configurations (the `default` and `all` configurations in JBoss AS 4.0.1), AOP support is provided by the `jboss-aop.deployer` service. It is a new `.deployer` type service similar to `.sar` service (see Section 2). Key features supported by the `jboss-aop.deployer` service are as follows.

- By default, you have to instrument the bytecode of your AOP applications offline using the `aopc` utility before you can deploy them into the application server. But you can enable load-time bytecode instrumentation via a configuration attribute in the `jboss-aop.deployer/META-INF/jboss-service.xml` file.
- JBoss AS 4.0 is shipped with several pre-packaged aspects to support security, transaction and asynchronous threads on plain old Java objects (POJOs). There are a number of predefined annotation tags in the `base-aop.xml` file. You can use those annotation in your POJOs to take advantage of the pre-packaged aspect services.
- JBoss AS 4.0 defines a new XML deployment file type with file name `*-aop.xml`. The `*-aop.xml` file specifies the binding for user-defined aspect classes. The aspect and binding become available to applications on the server.
- JBoss AS 4.0 defines a new JAR archive file type with the `.aop` file name extension. The `.aop` file can be used to package user-defined aspects and their bindings. The `jboss-aop.xml` file must reside in the `META-INF` directory in the `.aop` archive. The `.aop` archive can be bundled inside other deployment archive files to provide aspect services to a specific application.

5. Hibernate Integration

Hibernate is a very popular object persistence framework developed by JBoss. It maps Java objects to tables in relational databases and vice versa. The object-relational mapping rules and data sources are specified in special Hibernate configuration files. In JBoss AS 4.0, Hibernate integration support is provided by the `jboss-hibernate.deployer` service, which is available in the `default`, `standard` and `all` configurations. The `jboss-hibernate.deployer` service provides Hibernate framework libraries to all application on the server.

For Hibernate applications, JBoss defines a new `.har` service archive type. You can package your Hibernate mapped Java objects and mapping configuration files in the `.har` archive. You can also specify a data source name

and an JNDI name for this particular Hibernate configuration in the `hibernate-service.xml` file in the `.har` archive. The benefit is that, in your applications, you only need to do a JNDI lookup to retrieve the correctly configured Hibernate `SessionFactory` object. There is no need to load the mapping and data source configuration files manually in the application via API calls. In addition, the configuration settings in the `hibernate-service.xml` file is manageable via the JBoss JMX management console.

The `.har` file can be bundled inside a `.ear` file or deployed standalone.

6. Clustering and Caching

Many of the JBoss AS 4.0 clustering and caching improvements have been backported and available in JBoss 3.2.3 to 3.2.7. In this document, we will consolidate and give an overview of those improvements.

- TreeCache, which is based on the JGroups technology, is officially adopted as the underlying distributed cache architecture for the clustering environment.
- `CacheLoader` support (store/read from secondary storage) for both shared and unshared backend stores is added. Currently, we have `CacheLoader` implementations for the Sleepycat Berkeley DB (`BdbjeCacheLoader`), generic JDBC datasources, and the file system (`FileCacheLoader`) respectively.
- The `HttpSession` object is replicated across clustered servers. So, if one server fails, the users would be moved to a fail-over server without losing their sessions.
- The Single Sign-On (SSO) security context is also replicated across clustered servers. This way, the user would not be required to re-login when a server fails.
- The new loadbalancer service provides reverse-proxy support with silent failover.

7. References

- The JBoss Application Server 4.0 changelogs http://sourceforge.net/docman/index.php?group_id=22866
- The JBoss AOP Reference Documentation http://docs.jboss.org/aop/aspect-framework/reference/en/html/index.html