JBoss Enterprise Application Platform Common Criteria Certification 5

Getting Started Guide

for Use with JBoss Enterprise Application Platform 5 Common Criteria Certification Edition 5.1.0

Last Updated: 2017-11-20

JBoss Enterprise Application Platform Common Criteria Certification 5 Getting Started Guide

for Use with JBoss Enterprise Application Platform 5 Common Criteria Certification
Edition 5.1.0

Red Hat Documentation Group

Legal Notice

Abstract

This Getting Started Guide documents information regarding the initial use of the JBoss Enterprise Application Platform 5.1.0.

Table of Contents

INTRODUCTION

JBoss Enterprise Application Platform is the open source implementation of the Java EE suite of services. It comprises a set of offerings for enterprise customers who are looking for preconfigured profiles of JBoss Enterprise Middleware components that have been tested and certified together to provide an integrated experience. Its easy-to-use server architecture and high flexibility makes JBoss the ideal choice for users just starting out with J2EE, as well as senior architects looking for a customizable middleware platform.

Because it is Java-based, JBoss Enterprise Application Platform is cross-platform, easy to install and use on any operating system that supports Java. The readily available source code is a powerful learning tool to debug the server and understand it. It also gives you the flexibility to create customized versions for your personal or business use.

1. HELP CONTRIBUTE

If you find a typographical error or if you have thought of a way to make this manual better, submit a report in JIRA: http://jira.jboss.com against the product *JBoss Enterprise Application Platform* and component *Documentation*.

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

CHAPTER 1. THE JBOSS SERVER - A QUICK TOUR

1.1. SERVER STRUCTURE

For a thorough explanation of the structure of the application server, see Migration chapter of the Installation Guide that accompanies this release of JBoss Enterprise Application Platform.

1.2. STARTING AND STOPPING THE SERVER

1.2.1. Start the Server

Move to **$JBOSS_HOME/server/$PROFILE/bin** directory and execute the **run.sh** (for Linux) script.

There is no **Server Started** message shown at the console when the server is started using the **production** profile. This message can be found in the **server.log** file located in the **$JBOSS_HOME/server/$PROFILE/production/log** subdirectory.

IMPORTANT

The JBoss Enterprise Application Platform now binds its services to localhost (127.0.0.1) by default, instead of binding to all available interfaces (0.0.0.0). This was primarily done for security reasons because of concerns of users going to production without having secured their servers correctly. To enable remote access by binding JBoss services to a particular interface, simply run JBoss with the **-b** option. To bind to all available interfaces and re-enable the legacy behaviour use **./run.sh -b 0.0.0.0** on Linux. In any case, be aware you still need to secure your server properly.

Using **-b** as part of the JBoss Server's command line is equivalent to setting these individual properties: -Djboss.bind.address, -Djava.rmi.server.hostname, -Djgroups.bind_addr and -Dbind.address. Passing -Djboss.bind.address to the Java process as part of the **JAVA_OPTS** variable in the run scripts will not work as it is a JBoss property not a JVM property.

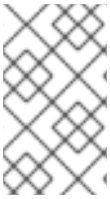
For more information including setting up multiple JBoss server instances on one machine and hosting multiple domains with JBoss, please refer to the Administration and Configuration Guide.

On starting your server, your screen output should look like the following (accounting for installation directory differences) and contain no error or exception messages:

```
[user@mypc bin]$ ./run.sh
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: unzip_locationjboss-as

  JAVA: java

  JAVA_OPTS: -Dprogram.name=run.sh -server -Xms1503m -Xmx1503m -
Dsun.rmi.dgc.client.
gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -
Djava.net.preferIPv4Stack=true
```

```
  CLASSPATH: unzip_location/jboss-as/bin/run.jar

========================================================================
```

More options for the JBoss Enterprise Application Platform **run** script are discussed in Section 1.2.2, "Start the Server With Alternate Configuration" below.



NOTE

There is no *Server Started* message shown at the console when the server is started using the **production** profile. This message may be observed in the **server.log** file located in the **server/production/log** subdirectory.

1.2.2. Start the Server With Alternate Configuration

Using **run.sh** without any arguments starts the server using the **default** server profile file set. To start with an alternate profile file set, pass the name of the server configuration file set (same as the name of the server configuration directory under **$JBOSS_HOME/server/$PROFILE/**) that you want to use, as the value to the **-c** command line option. For example, to start with the **minimal** profile file set you should specify:

```
[bin]$ ./run.sh -c minimal
...
...
...
15:05:40,301 INFO  [Server] JBoss (MX MicroKernel) [5.0.0 (build:
SVNTag=JBoss_5_0_0 date=200801092200)] Started in 5s:75ms
```

1.2.3. Using run.sh

The **run** script supports the following options:

```
usage: run.sh [options]
-h, --help                          Show help message
-V, --version                       Show version information
--                                  Stop processing options
-D<name>[=<value>]      Set a system property
-d, --bootdir=<dir>           Set the boot patch directory; Must be
absolute or url
-p, --patchdir=<dir>          Set the patch directory; Must be absolute or
url
-c, --configuration=<name>    Set the server configuration name
-B, --bootlib=<filename>      Add an extra library to the front
bootclasspath
-L, --library=<filename>      Add an extra library to the loaders
classpath
-C, --classpath=<url>         Add an extra url to the loaders classpath
-P, --properties=<url>        Load system properties from the given url
-b, --host=<host or ip>       Bind address for all JBoss services.
-g, --partition=<name>        HA Partition name (default=DefaultDomain)
-m, --mcast_port=<ip>         UDP multicast port; only used by JGroups
-u, --udp=<ip>                UDP multicast address
```

```
-l, --log=<log4j|jdk>         Specify the logger plugin type
```

1.2.4. Stopping the Server

To shutdown the server, you simply issue a **Ctrl-C** sequence in the console in which JBoss was started. Alternatively, you can use the **shutdown.sh** command.

```
[bin]$ ./shutdown.sh -S
```

The **shutdown** script supports the following options:

```
A JMX client to shutdown (exit or halt) a remote JBoss server.

usage: shutdown [options] <operation>

options:
-h, --help                      Show this help message (default)
-D<name>[=<value>]  Set a system property
--                              Stop processing options
-s, --server=<url>         Specify the JNDI URL of the remote server
-n, --serverName=<url>     Specify the JMX name of the ServerImpl
-a, --adapter=<name>       Specify JNDI name of the MBeanServerConnection
to use
-u, --user=<name>          Specify the username for authentication
-p, --password=<name>      Specify the password for authentication

operations:
-S, --shutdown                  Shutdown the server
-e, --exit=<code>          Force the VM to exit with a status code
-H, --halt=<code>          Force the VM to halt with a status code
```

Using the shutdown command requires a server configuration that contains the **jmx-invoker-service.xml** service. Hence you cannot use the shutdown command with the **minimal** profile.

1.2.5. Running As A System Service

It is possible to run the Application Server as a service under Windows, Linux, and UNIX. Refer to the post-installation chapter of the JBoss Enterprise Application Platform Installation Guide for instructions.

1.3. THE JMX CONSOLE

When the JBoss Server is running, you can get a live view of the server by going to the JMX console application at http://localhost:8080/jmx-console.

By default, the JMX console is secured and will prompt you for a username and password. If you installed JBoss Enterprise Application Platform using the graphical installer and you want to access the JMX console, you can use the username and password you provided when it was installed. If you installed using other modes such as .zip, go to the **$JBOSS_HOME/server/$PROFILE/conf/props/directory** and uncomment the admin userid and password code within the **jmx-console-users.properties** file. You can add other users as needed. This will allow the defined users access to the JMX console using the username and password combination specified within the **jmx-console-users.properties** file.

See Section 1.6.5, “Security Service” for further information about the security service in JBoss Enterprise Application Platform.

IMPORTANT

If you changed the **jmx-console-users.properties** file when the server was running, you may have to restart the server for the changes to take effect. In some cases, lazy loading can make this change live without restarting the server.

The JMX Console is the JBoss Management Console which provides a raw view of the JMX MBeans which make up the server. They can provide a lot of information about the running server and allow you to modify its configuration, start and stop components and so on.

For example, find the **service=JNDIView** link and click on it. This particular MBean provides a service to allow you to view the structure of the JNDI namespaces within the server. Now find the operation called **list** near the bottom of the MBean view page and click the **invoke** button. The operation returns a view of the current names bound into the JNDI tree, which is very useful when you start deploying your own applications and want to know why you can’t resolve a particular EJB name.

Look at some of the other MBeans and their listed operations; try changing some of the configuration attributes and see what happens. With a very few exceptions, none of the changes made through the console are persistent. The original configuration will be reloaded when you restart JBoss, so you can experiment freely without doing any permanent damage.

1.4. THE JNDIVIEW SERVICE

The JNDIView Service is enabled by default in the JBoss Enterprise Application Platform. This service is listed in the **jmx-console** (http://localhost:8080/jmx-console). Navigate to the **jboss:service=JNDIView** Mbean and click on that link. On the MBean operations page, you will find the **list** method. Click on the **Invoke** button adjacent to this **list** method.

The list operation will display the JNDI tree contents. The output will look something similar to this:

```
java: Namespace

+- securityManagement (class:
org.jboss.security.integration.JNDIBasedSecurityManagement)
+- comp (class: javax.namingMain.Context)
+- XAConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- JmsXA (class: org.jboss.resource.adapter.jms.JmsConnectionFactoryImpl)
+- policyRegistration (class:
org.jboss.security.plugins.JBossPolicyRegistration)
+- TransactionPropagationContextImporter (class:
com.arjuna.ats.internal.jbossatx.jta.PropagationContextManager)
+- app (class: org.jnp.interfaces.NamingContext)
|   +- Manager (class: javax.inject.manager.Manager)
+- ClusteredConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- Mail (class: javax.mail.Session)
+- TransactionPropagationContextExporter (class:
com.arjuna.ats.internal.jbossatx.jta.PropagationContextManager)
+- ProfileService (class:
org.jboss.system.server.profileservice.repository.AbstractProfileService)
```

```
+- DefaultDS (class: org.jboss.resource.adapter.jdbc.WrapperDataSource)
+- jaas (class: javax.naming.Context)
|   +- HsqlDbRealm (class:
org.jboss.security.plugins.SecurityDomainContext)
+- ClusteredXAConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- TransactionSynchronizationRegistry (class:

com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionSynchronizat
ionRegistryImple)
+- SecurityProxyFactory (class:
org.jboss.security.SubjectSecurityProxyFactory)
+- ConnectionFactory (class: org.jboss.jms.client.JBossConnectionFactory)
+- DefaultJMSProvider (class: org.jboss.jms.jndi.JNDIProviderAdapter)
+- TransactionManager (class:
com.arjuna.ats.jbossatx.jta.TransactionManagerDelegate)
+- timedCacheFactory (class: javax.naming.Context)
Failed to lookup: timedCacheFactory,
errmsg=org.jboss.util.TimedCachePolicy cannot be cast to
javax.naming.NamingEnumeration


Global JNDI Namespace

+- UserTransactionSessionFactory (proxy: $Proxy109 implements interface
org.jboss.tm.usertx.interfaces.UserTransactionSessionFactory)
+- UUIDKeyGeneratorFactory (class:
org.jboss.ejb.plugins.keygenerator.uuid.UUIDKeyGeneratorFactory)
+- HiLoKeyGeneratorFactory (class:
org.jboss.ejb.plugins.keygenerator.hilo.HiLoKeyGeneratorFactory)
+- SecureDeploymentManager (class: org.jnp.interfaces.NamingContext)
|   +- remote[link -> DeploymentManager] (class: javax.naming.LinkRef)
+- SecureManagementView (class: org.jnp.interfaces.NamingContext)
|   +- remote[link -> ManagementView] (class: javax.naming.LinkRef)
+- persistence.unit:unitName=jsfejb3.ear (class:
org.jnp.interfaces.NamingContext)
|   +- app.jar#helloworld (class: org.hibernate.impl.SessionFactoryImpl)
+- DeploymentManager (class: org.jboss.aop.generatedproxies.AOPProxy$4)
+- XAConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- topic (class: org.jnp.interfaces.NamingContext)
+- ClusteredConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- ProfileService (class: org.jboss.aop.generatedproxies.AOPProxy$2)
+- SecureProfileService (class: org.jnp.interfaces.NamingContext)
|   +- remote[link -> ProfileService] (class: javax.naming.LinkRef)
+- queue (class: org.jnp.interfaces.NamingContext)
|   +- DLQ (class: org.jboss.jms.destination.JBossQueue)
|   +- ExpiryQueue (class: org.jboss.jms.destination.JBossQueue)
+- ClusteredXAConnectionFactory (class:
org.jboss.jms.client.JBossConnectionFactory)
+- UserTransaction (class:
org.jboss.tm.usertx.client.ClientUserTransaction)
+- ConnectionFactory (class: org.jboss.jms.client.JBossConnectionFactory)
+- jmx (class: org.jnp.interfaces.NamingContext)
|   +- invoker (class: org.jnp.interfaces.NamingContext)
```

```
|   |   +- RMIAdaptor (proxy: $Proxy103 implements interface
org.jboss.jmx.adaptor.rmi.RMIAdaptor,interface
org.jboss.jmx.adaptor.rmi.RMIAdaptorExt)
|   +- rmi (class: org.jnp.interfaces.NamingContext)
|   |   +- RMIAdaptor[link -> jmx/invoker/RMIAdaptor] (class:
javax.naming.LinkRef)
+- TomcatAuthenticators (class: java.util.Properties)
+- console (class: org.jnp.interfaces.NamingContext)
|   +- PluginManager (proxy: $Proxy104 implements interface
org.jboss.console.manager.PluginManagerMBean)
+- ManagementView (class: org.jboss.aop.generatedproxies.AOPProxy$3)
```

This details the JNDI names to which your EJBs are bound.

1.5. HOT-DEPLOYMENT OF SERVICES IN JBOSS

Hot-deployable services are those which can be added to or removed from the running server. These are placed in the **JBOSS_DIST/jboss-as/server/<instance-name>/deploy** directory. Let's have a look at a practical example of hot-deployment of services in JBoss.

Start JBoss if it isn't already running and take a look at the **server/default/deploy** directory. Remove the **mail-service.xml** file and watch the output from the server:

```
13:10:05,235 INFO  [MailService] Mail service 'java:/Mail' removed from
JNDI
```

Then replace the file and watch JBoss re-install the service:

```
13:58:54,331 INFO  [MailService] Mail Service bound to java:/Mail
```

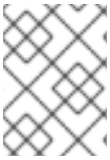
This is hot-deployment in action.

1.5.1. Hot-deployment configurations

Hot deployment of services in the server is controlled by the HDScanner MC bean configured in **$JBOSS_HOME/server/conf/deploy/hdscanner-jboss-beans.xml** file. For the **default** server configuration the scanPeriod is set to 5 seconds:

```
<bean name="HDScanner"
class="org.jboss.system.server.profileservice.hotdeploy.HDScanner">
 <property name="deployer"><inject bean="ProfileServiceDeployer"/>
</property>
 <property name="profileService"><inject bean="ProfileService"/>
</property>
 <property name="scanPeriod">5000</property>
 <property name="scanThreadName">HDScanner</property>
</bean>
```

The scanPeriod attribute controls the interval for thread which picks up the hot deployable changes.



NOTE

The changes to the **hdscanner-jboss-beans.xml** file itself are hot deployable. No server restart is needed.

1.5.2. Adding a custom deploy folder

JBoss server by default looks for deployments under the **JBOSS_DIST/jboss-as/server/<instance-name>/deploy** folder. However you can configure the server to even include your custom folder for scanning deployments. This can be done by configuring the **BootstrapProfileFactory** MC bean in **$JBOSS_HOME/server/$PROFILE/conf/bootstrap/profile.xml** file. The applicationURIs property of the **BootstrapProfileFactory** accepts a list of URLs which will be scanned for applications. You can add your custom deploy folder to this list. For example, if you want **/home/me/myapps** to be scanned for deployments, then you can add the following:

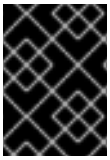
```
<bean name="BootstrapProfileFactory"
class="org.jboss.system.server.profileservice.repository.
StaticProfileFactory">
 ...
 <property name="applicationURIs">
  <list elementClass="java.net.URI">
   <value>${jboss.server.home.url}deploy</value>
   <value>file:///home/me/myapps</value>
  </list>
 ...
```

IMPORTANT

Modifying the **$JBOSS_HOME/server/$PROFILE/conf/bootstrap/profile.xml** requires a server restart, for the changes to take effect.

For performance reasons, adding a new deployment folder to the **BootstrapProfileFactory** also requires the same URL to be added to the **VFSCache** MC bean configuration in **$JBOSS_HOME/server/$PROFILE/conf/bootstrap/vfs.xml**. For example:

```
<bean name="VFSCache">
 ...
 <property name="permanentRoots">
  <map keyClass="java.net.URL"
valueClass="org.jboss.virtual.spi.ExceptionHandler">
   ...
   <entry>
    <key>file:///home/me/myapps</key>
    <value><inject bean="VfsNamesExceptionHandler"/></value>
   </entry>
  </map>
 </property>
 ...
```



IMPORTANT

Not adding the custom deployment folder to **VFSCache** might result in growing disk space usage by the server, over a period of time.

1.6. BASIC CONFIGURATION ISSUES

Now that we have examined the JBoss server, we will take a look at some of the main configuration files and what they are used for. All paths are relative to the server configuration directory (**server/default**, for example).

1.6.1. Setting your application as the default application on the server

JBoss server by default configures **$JBOSS_HOME/server/$PROFILE/deploy/ROOT.war** as the default application on the server. So accessing **http://localhost:8080/** results in displaying the index page of this application. If you want your application to be available as the default application, then you will wish to follow these steps:

- Rename **ROOT.war** in **$JBOSS_HOME/server/$PROFILE/deploy** to something else, for example, **jboss.war**.
- In your WAR file (the one which you want to be the default application), add a **jboss-web.xml**, in the **WEB-INF** folder, with a configuration for the context-root:

```
<?xml version="1.0"?>
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 5.0//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_5_0.dtd">

<jboss-web>
 <context-root>/</context-root>
 <!-- Other configurations as needed -->
</jboss-web>
```

By setting the context-root to **/** you are making your application the default application. Your application will now be available at **http://localhost:8080/**.

NOTE

Renaming the **ROOT.war** to **jboss.war** will make that application be available at **http://localhost:8080/jboss**

1.6.2. Bootstrap Configuration

The microcontainer bootstrap configuration is described by the **conf/bootstrap.xml** and the **conf/bootstrap/*.xml** it references. It is expected that the number of bootstrap beans will be reduced in the future. It is not expected that you would need to edit the bootstrap configuration files for a typical installation.

1.6.3. Legacy Core Services

The legacy core services specified in the **conf/jboss-service.xml** file are started just after server starts up the microcontainer. If you have a look at this file in an editor you will see MBeans for various

services including logging, security, JNDI, JNDIView etc. Try commenting out the entry for the **JNDIView** service.

NOTE

Eventually this file will be dropped as the services are converted to microcontainer beans or mbeans that are deployed as deploy directory services.

Note that because the mbeans definition had nested comments, we had to comment out the mbean in two sections, leaving the original comment as it was.

```
<!-- Section 1 commented out
<mbean code="org.jboss.naming.JNDIView"
    name="jboss:service=JNDIView"
    xmbean-dd="resource:xmdesc/JNDIView-xmbean.xml">
-->
    <!-- The HANamingService service name -->
<!-- Section two commented out
    <attribute name="HANamingService">jboss:service=HAJNDI</attribute>
</mbean>
-->
```

If you then restart JBoss, you will see that the **JNDIView** service no longer appears in the JMX Management Console (JMX Console) listing. In practice, you should rarely, if ever, need to modify this file, though there is nothing to stop you adding extra MBean entries in here if you want to. The alternative is to use a separate file in the **deploy** directory, which allows your service to be hot deployable.

1.6.4. Logging Service

In JBoss **log4j** is used for logging. If you are not familiar with the **log4j** package and would like to use it in your applications, you can read more about it at the Jakarta web site (http://jakarta.apache.org/log4j/).

Logging is controlled from a central **conf/jboss-log4j.xml** file. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. By default, JBoss produces output to both the console and a log file (**log/server.log**).

There are six basic log levels used: **TRACE**, **DEBUG**, **INFO**, **WARN**, **ERROR** and **FATAL**. The logging threshold on the console is **INFO**, which means that you will see informational messages, warning messages and error messages on the console but not general debug and trace messages. In contrast, there is no threshold set for the **server.log** file, so all generated logging messages will be logged there.

If things are going wrong and there doesn't seem to be any useful information in the console, always check the **server.log** file to see if there are any debug messages which might help you to track down the problem. However, be aware that just because the logging threshold allows debug messages to be displayed, that doesn't mean that all of JBoss will produce detailed debug information for the log file. You will also have to boost the logging limits set for individual categories. Take the following category for example.

```
<!-- Limit JBoss categories to INFO -->
<category name="org.jboss">
```

```
    <priority value="INFO"/>
</category>
```

This limits the level of logging to **INFO** for all JBoss classes, apart from those which have more specific overrides provided. By default the root logger in the **jboss-log4j.xml** is set to **INFO**. This effectively means that any **TRACE** or **DEBUG** logger from any logger categories will not be logged in any files or the console appender. This setting is controlled through the jboss.server.log.threshold property. By default this is **INFO**. If you were to change this to **DEBUG**, it would produce much more detailed logging output. In order to change this there are two options:

- You can pass the -Djboss.server.log.threshold=DEBUG parameter while starting the server:

  ```
  ./run.sh -Djboss.server.log.threshold=DEBUG
  ```

- You can edit the **$JBOSS_HOME/server/$PROFILE/conf/jboss-log4j.xml** file directly in order to set this property:

  ```
  <root>
   <!-- Let's comment this out to set our own value
   <priority value="${jboss.server.log.threshold}"/>-->
   <priority value="DEBUG"/>
   <appender-ref ref="CONSOLE"/>
   <appender-ref ref="FILE"/>
  </root>
  ```

NOTE

The **$JBOSS_HOME/server/$PROFILE/conf/jboss-log4j.xml** is scanned every 60 seconds (by default) to check for any changes. Changing this file does not require a server restart as the changes will be hot deployed within the next 60 seconds following the change.

As another example, let's say you wanted to set the output from the container-managed persistence engine to **DEBUG** level and to redirect it to a separate file, **cmp.log**, in order to analyze the generated SQL commands. You would add the following code to the **conf/jboss-log4j.xml** file:

```
<appender name="CMP"
class="org.jboss.logging.appender.RollingFileAppender">
    <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
    <param name="File" value="${jboss.server.home.dir}/log/cmp.log"/>
    <param name="Append" value="false"/>
    <param name="MaxFileSize" value="500KB"/>
    <param name="MaxBackupIndex" value="1"/>

    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d %-5p [%c] %m%n"/>
    </layout>
</appender>

<category name="org.jboss.ejb.plugins.cmp">
    <priority value="DEBUG" />
    <appender-ref ref="CMP"/>
</category>
```

This creates a new file appender and specifies that it should be used by the logger (or category) for the package **org.jboss.ejb.plugins.cmp**.

The file appender is set up to produce a new log file every day rather than producing a new one every time you restart the server or writing to a single file indefinitely. The current log file is **cmp.log**. Older files have the date they were written added to their filenames. Please note that the **log** directory also contains HTTP request logs which are produced by the web container.

By default the **server.log** appender is configured to retain log messages between server restarts. This is controlled by the Append property on the **FILE** appender which corresponds to the **server.log** file. By default this property is set to true; if you want the **server.log** contents to be wiped out on server restarts then you can edit the **$JBOSS_HOME/server/$PROFILE/conf/jboss-log4j.xml** file to set this property value to false. For example:

```
<appender name="FILE"
class="org.jboss.logging.appender.DailyRollingFileAppender">
 <errorHandler class="org.jboss.logging.util.OnlyOnceErrorHandler"/>
  <param name="File" value="${jboss.server.log.dir}/server.log"/>
  <param name="Append" value="false"/>
  ...
```

1.6.5. Security Service

The security domain information is stored in the file **conf/login-config.xml** as a list of named security domains, each of which specifies a number of JAAS [1] login modules which are used for authentication purposes in that domain. When you want to use security in an application, you specify the name of the domain you want to use in the application's JBoss-specific deployment descriptors, **jboss.xml** (used in defining jboss specific configurations for an application) and/or **jboss-web.xml** (used in defining JBoss for a Web application. We'll quickly look at how to do this to secure the JMX Console application which ships with JBoss.

Almost every aspect of the JBoss server can be controlled through the JMX Console, so it is important to make sure that, at the very least, the application is password protected. Otherwise, any remote user could completely control your server. To protect it, we will add a security domain to cover the application. This can be done in the **jboss-web.xml** file for the JMX Console, which can be found in **deploy/jmx-console.war/WEB-INF/** directory. Uncomment the **security-domain** in that file, as shown below.

```
<jboss-web>
    <security-domain>java:/jaas/jmx-console</security-domain>
</jboss-web>
```

This links the security domain to the web application, but it doesn't tell the web application what security policy to enforce, what URLs are we trying to protect, and who is allowed to access them. To configure this, go to the **web.xml** file in the same directory and uncomment the **security-constraint** that is already there. This security constraint will require a valid user name and password for a user in the **JBossAdmin** group.

```
<!--
   A security constraint that restricts access to the HTML JMX console
   to users with the role JBossAdmin. Edit the roles to what you want and
   uncomment the WEB-INF/jboss-web.xml/security-domain element to enable
   secured access to the HTML JMX console.
-->
```

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>HtmlAdaptor</web-resource-name>
        <description>
            An example security config that only allows users with the
            role JBossAdmin to access the HTML JMX console web application
        </description>
        <url-pattern>/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>JBossAdmin</role-name>
    </auth-constraint>
</security-constraint>
```

That's great, but where do the user names and passwords come from? They come from the **jmx-console** security domain we linked the application to. We have provided the configuration for this in the **conf/login-config.xml**.

```
<application-policy name="jmx-console">
    <authentication>
        <login-module
code="org.jboss.security.auth.spi.UsersRolesLoginModule"
                    flag="required">
            <module-option name="usersProperties">
                props/jmx-console-users.properties
            </module-option>
            <module-option name="rolesProperties">
                props/jmx-console-roles.properties
            </module-option>
        </login-module>
    </authentication>
</application-policy>
```

This configuration uses a simple file based security policy. The configuration files are found in the **conf/props** directory of your server configuration. The usernames and passwords are stored in the **conf/props/jmx-console-users.properties** file and take the form "**username=password**". To assign a user to the **JBossAdmin** group add "**username=JBossAdmin**" to the **jmx-console-roles.properties** file (additional roles on that username can be added comma separated). The existing file creates an **admin** user with the password **admin**. For security, please either remove the user or change the password to a stronger one.

JBoss will re-deploy the JMX Console whenever you update its **web.xml**. You can check the server console to verify that JBoss has seen your changes. If you have configured everything correctly and re-deployed the application, the next time you try to access the JMX Console, it will ask you for a name and password. [2]

The JMX Console is not the only web based management interface to JBoss. There is also the Web Console. Although it is a Java applet, the corresponding web application can be secured in the same way as the JMX Console. The Web Console is in the file **deploy/management/console-mgr.sar/web-console.war.**. The only difference is that the Web Console is provided as a simple WAR file instead of using the exploded directory structure that the JMX Console did. The only real difference between the two is that editing the files inside the WAR file is a bit more cumbersome.

1.6.6. Additional Services

The non-core, hot-deployable services are added to the **deploy** directory. They can be either XML descriptor files, ***-service.xml, *-jboss-beans.xml**, MC **.beans** archive, or JBoss Service Archive (SAR) files. SARs contains an **META-INF/jboss-service.xml** descriptor and additional resources the service requires (for example, classes, library JAR files or other archives), all packaged up into a single archive. Similarly, a **.beans** archive contains a **META-INF/jboss-beans.xml** and additional resources.

Detailed information on all these services can be found in the *JBoss Enterprise Application Platform: Aministration and Configuration Guide*, which also provides comprehensive information on server internals and the implementation of services such as JTA and the J2EE Connector Architecture (JCA).

1.7. THE SERVICE BINDING MANAGER

JBoss server uses various ports for the services that it provides (for example, port 8080 for HTTP, 1099 for JNDI). The Service Binding Manager (SBM) service provides a centralized location where settings for all services that need to bind to ports can be configured. SBM can be used to configure different sets of port bindings for a server instance. A system property on the SBM controls which named set (for example, ports-default, ports-01) is used by a particular server instance. If you want to run multiple server instances on the same system then you can configure the SBM on each instance to use a different named binding set. You can even use SBM to switch to a different binding set (for example, 8180 port for HTTP instead of the default 8080) for a server instance.

In a typical configuration, the **ports-default** set uses the standard ports (for example, JNDI on port 1099), with **ports-01** increasing each port value by 100 (for example, JNDI on 1199), **ports-02** by 200 and so on.

SBM is configured through the **$JBOSS_HOME/server/$PROFILE/conf/bindingservice.beans/META-INF/bindings-jboss-beans.xml** file. The configuration of the **ServiceBindingManager** involves three primary elements :

- A set of beans containing standard (default) binding configuration data. These are the base values (for example, JNDI on 1099) used to drive **ports-default**, **ports-01** and so on.
- A number of beans defining **ServiceBindingSets**, for example, **ports-default**, **ports-01**, **ports-02**. The sets of standard bindings are combined with each of these, along with an offset value (for example, 100 for **ports-01**) that should be applied to the standard port values to create the binding values for that set.
- The **ServiceBindingManager** service bean itself. This has the standard bindings and the **ServiceBindingSets** injected into it. It is also configured with the name of the binding set the particular server instance should use. The name of the binding set to be used is configurable from the command line by using the system property jboss.service.binding.set. The default value is **ports-default**.

```
<bean name="ServiceBindingManagementObject"
class="org.jboss.services.binding.managed.ServiceBindingManagementOb
ject">
 <constructor>

  <parameter>
   ${jboss.service.binding.set:ports-default}
  </parameter>
 ...
```

To switch to a different set of ports than the ones used by default, you can start the server by passing the -Djboss.service.binding.set property to the run command as follows:

```
./run.sh -Djboss.service.binding.set=ports-01
```

This will instruct the server to use the group of ports configured in the **ports-01** binding set.

[1] The Java Authentication and Authorization Service. JBoss uses JAAS to provide pluggable authentication modules. You can use the ones that are provided or write your own if you have more specific requirements.

[2] Since the username and password are session variables in the web browser you may need to restart your browser to use the login dialog window.

CHAPTER 2. USING OTHER DATABASES

In the previous chapters, we've been using the JBoss Enterprise Application Platform server default datasource in our applications. This datasource is configured to use the embedded Hypersonic database instance shipped by default with the distribution. This datasource is bound to the JNDI name **`java:/DefaultDS`** and its descriptor is named **`hsqldb-ds.xml`** under the deploy directory.

WARNING

The default persistence configuration works out of the box with Hypersonic (HSQLDB) so that the JBoss Enterprise Platforms are able to run "out of the box". However, *Hypersonic is not supported in production and should not be used in a production environment.*

Known issues with the Hypersonic Database include:

- no transaction isolation
- thread and socket leaks (**`connection.close()`** does not tidy up resources)
- persistence quality (logs commonly become corrupted after a failure, preventing automatic recovery)
- database corruption
- stability under load (database processes cease when dealing with too much data)
- not viable in clustered environments

In this chapter we will explain in detail how to configure and deploy a datasource to connect the JBoss Enterprise Application Platform to the most popular database servers available on the market today.

2.1. DATASOURCE CONFIGURATION FILES

Datasource configuration file names end with the suffix **`-ds.xml`** so that they will be recognized correctly by the JCA deployer. The **`docs/example/jca`** directory contains sample files for a wide selection of databases and it is a good idea to use one of these as a starting point. For a full description of the configuration format, the best place to look is the DTD file **`docs/dtd/jboss-ds_1_5.dtd`**. Additional documentation on the files and the JBoss JCA implementation can also be found in the JBoss Enterprise Application Platform Administration and Server Configuration Guide available at http://www.redhat.com/docs/en-US/JBoss_Enterprise_Application_Platform/.

Local transaction datasources are configured using the **`local-tx-datasource`** element and XA-compliant ones using **`xa-tx-datasource`**. The example file **`generic-ds.xml`** shows how to use both types and also some of the other elements that are available for things like connection pool configuration. Examples of both local and XA configurations are available for Oracle, DB2 and Informix.

If you look at the example files **`firebird-ds.xml`**, **`facets-ds.xml`** and **`sap3-ds.xml`**, you'll

notice that they have a completely different format, with the root element being **connection-factories** rather than **datasources**. These use an alternative, more generic JCA configuration syntax used with a pre-packaged JCA resource adapter. The syntax is not specific to datasource configuration and is used, for example, in the **$JBOSS_HOME/server/$PROFILE/deploy/messaging/jms-ds.xml** file to configure the JMS resource adapter.

Next, we'll work through some step-by-step examples to illustrate what's involved setting up a datasource for a specific database.

2.2. USING MYSQL AS THE DEFAULT DATASOURCE

The MySQL® database has become the world's most popular open source database thanks to its consistent fast performance, high reliability and ease of use. This database server is used in millions of installations ranging from large corporations to specialized embedded applications across every continent of the world. The official JDBC driver is called **Connector/J**. For this example we've used MySQL 5.1.31 and **Connector/J** 5.1.8. Both are available at http://www.mysql.com.

2.2.1. Creating a Database and User

We'll assume that you've already installed MySQL and that you have it running and are familiar with the basics. Run the MySQL client program from the command line so we can execute some administration commands. You should make sure that you are connected as a user with sufficient privileges (for example, by specifying the **-u** root option to run as the MySQL root user).

First create a database called **jboss** within MySQL for use by JBoss:

```
mysql> CREATE DATABASE jboss;

Query OK, 1 row affected (0.05 sec)
```

Then check that it has been created:

```
mysql> SHOW DATABASES;

+----------+
| Database |
+----------+
| jboss    |
+----------+
1 rows in set (0.00 sec)
```

Next, create a user called **jboss** with 'password' as the password to access the database:

```
mysql> GRANT ALL PRIVILEGES ON jboss.* TO jboss@localhost IDENTIFIED BY
'password';

Query OK, 0 rows affected (0.06 sec)
```

Again, you can check that everything has gone smoothly:

```
mysql> select User,Host,Password from mysql.User;
```

```
+-------+-----------+-------------------+
| User  | Host      | Password          |
+-------+-----------+-------------------+
| root  | localhost |                   |
| root  | %         |                   |
|       | localhost |                   |
|       | %         |                   |
| jboss | localhost | 5d2e19393cc5ef67 |
+-------+-----------+-------------------+
5 rows in set (0.02 sec)
```

2.2.2. Installing the JDBC Driver and Deploying the datasource

To make the JDBC driver classes available to the JBoss Enterprise Application Platform, copy the archive **mysql-connector-java-5.1.8-bin.jar** from the **Connector/J** distribution to the **lib** directory in the **default** server configuration (assuming that is the server configuration you're running).

Then create a file in the deploy directory called **mysql-ds.xml** with the following datasource configuration. Note that the databse user name and password corresponds to the MySQL user that we created in the previous section:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
 <local-tx-datasource>
  <jndi-name>MySqlDS</jndi-name>
  <connection-url>jdbc:mysql://localhost:3306/jboss</connection-url>
  <driver-class>com.mysql.jdbc.Driver</driver-class>
  <user-name>jboss</user-name>
  <password>password</password>
 </local-tx-datasource>
</datasources>
```

To ensure that you have correctly configured the datasource in **$JBOSS_HOME/server/$PROFILE/deploy** folder, start the server and you will notice messages like these in the logs:

```
INFO  [ConnectionFactoryBindingService] Bound ConnectionManager
'jboss.jca:service=DataSourceBinding,name=MySqlDS' to JNDI name
'java:MySqlDS'
```

NOTE

Configuring other datasources is a similar process.

2.2.3. Testing the MySQL DataSource

Using the test client described in Section 2.6, "Creating a JDBC client", you may now verify the proper installation of your datasource.

2.3. CONFIGURING A DATASOURCE FOR ORACLE DB

Oracle is one of the main players in the commercial database field and most readers will probably have come across it at some point. You can download it freely for non-commercial purposes from http://www.oracle.com/technology/products/database/xe/index.html

In this section, we'll connect the server to Oracle Database 11g Express Edition using the latest JDBC driver (11g) available at http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

2.3.1. Installing the JDBC Driver and Deploying the DataSource

To make the JDBC driver classes available to the JBoss Enterprise Application Platform, copy the archive **`ojdbc6.jar`** to the lib directory in the default server configuration (assuming that is the server configuration you're running).

Then create a text file in the **`deploy`** directory called **`oracle-ds.xml`** with the following datasource descriptor :

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
 <local-tx-datasource>
  <jndi-name>DefaultDS</jndi-name>
  <connection-url>jdbc:oracle:thin:@localhost:1521:xe</connection-url>
  <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
  <user-name>SYSTEM</user-name>
  <password>jboss</password>
  <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker</
valid-connection-checker-class-name>
  <metadata>
   <type-mapping>Oracle9i</type-mapping>
  </metadata>
 </local-tx-datasource>
</datasources>
```

The datasource is pointing at the database/SID called *xe* provided by default with Oracle XE.

Of course, you need to update the connection url attributes as well as the username/password combination to match your environment setup.

2.3.2. Testing the Oracle DataSource

Before you can verify the datasource configuration, Oracle XE should be reconfigured to avoid port conflict with the JBoss Enterprise Application Platform as by default they both start a web server on port 8080.

Open up an Oracle SQLcommand line and execute the following commands:

```
SQL> connect; Enter user-name: SYSTEM Enter password:
Connected.
SQL> begin 2 dbms_xdb.sethttpport('8090'); 3 end; 4 /
PL/SQL procedure successfully completed.
SQL> select dbms_xdb.gethttpport from dual;
GETHTTPPORT
-----------
8090
```

The web server started by Oracle XE to provide http-based administration tools is now running on port 8090. Start the JBoss Enterprise Application Platform server instance as you would normally do. You are now ready to use the test client to verify the proper installation of your datasource.

2.4. CONFIGURING A DATASOURCE FOR MICROSOFT SQL SERVER 200X

In this section, we'll connect the server to MS SQL Server 2005 using the latest JDBC driver (v2.0) available at http://msdn2.microsoft.com/en-us/data/aa937724.aspx.

2.4.1. Installing the JDBC Driver and Deploying the DataSource

To make the JDBC driver classes available to the JBoss Enterprise Application Platform, copy the archive **`sqljdbc.jar`** from the **`sqljdbc_2.0`** distribution to the **`lib`** directory in the default server configuration (assuming that is the server configuration you're running).

Then create a text file in the **`deploy`** directory called **`mssql-ds.xml`** with the following datasource descriptor :

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
  <jndi-name>DefaultDS</jndi-name>
  <connection-
url>jdbc:sqlserver://localhost:1433;DatabaseName=pubs</connection-url>
  <driver-class>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver-
class>
  <user-name>sa</user-name>
  <password>jboss</password>
  <check-valid-connection-sql>SELECT 1 FROM sysobjects</check-valid-
connection-sql>
  <metadata>
   <type-mapping>MS SQLSERVER2000</type-mapping>
  </metadata>
  </local-tx-datasource>
</datasources>
```

The datasource is pointing at a database **pubs** provided by default with MS SQL Server 2000.

Remember to update the connection url attributes as well as the username/password combination to match your environment setup.

2.4.1.1. Testing the datasource

Using the test client described in Section 2.6, "Creating a JDBC client", you may now verify the proper installation of your datasource.

2.5. CONFIGURING JBOSS MESSAGING PERSISTENCE MANAGER

The persistence manager of JBoss Messaging uses the default datasource to create tables to store messages, transaction data and other indexes. Configuration of "persistence" is grouped in **`xxx-persistence-service.xml`** files. JBoss Enterprise Application Platform ships with a default

hsqldb-persistence-service.xml file, which configures the Messaging server to use the Hypersonic database instance that ships by default with the JBoss Enterprise Application Platform.

You can view the **hsqldb-persistence-service.xml** file in configurations based on the *all* or *default* configurations:

```
<JBoss_Home>/server/all/deploy/messaging/hsqldb-persistence-service.xml
and
   <JBoss_Home>/server/default/deploy/messaging/hsqldb-persistence-
service.xml
```

WARNING

The Hypersonic database is not recommended for production environments due to its limited support for transaction isolation and its low reliability under high load

More information on configuring JBoss Messaging can be found in the Administration and Configuration Guide.

2.6. CREATING A JDBC CLIENT

When testing a newly configured datasource we suggest using some very basic JDBC client code embedded in a JSP page. First of all, you should create an exploded WAR archive under the deploy directory which is simply a folder named "**jdbcclient.war**". In this folder, create a text document named client.jsp and paste the code below:

```
<%@page contentType="text/html"
 import="java.util.*,javax.naming.*,javax.sql.DataSource,java.sql.*"
 %>
 <%

  DataSource ds = null;
  Connection con = null;
  PreparedStatement pr = null;
  InitialContext ic;
  try {
  ic = new InitialContext();
  ds = (DataSource)ic.lookup( "java:/DefaultDS" );
  con = ds.getConnection();
  pr = con.prepareStatement("SELECT USER_ID, PASSWD FROM JBM_USER");
  ResultSet rs = pr.executeQuery();
  while (rs.next()) {
  out.println("<br> " +rs.getString("USER_ID") + " | "
+rs.getString("PASSWD"));
  }
  rs.close();
  pr.close();
  }catch(Exception e){
  out.println("Exception thrown " +e);
```

```
  }finally{
  if(con != null){
  con.close();
 }
} %>
```

Open up a web browser and hit the url: http://localhost:8080/jdbcclient/client.jsp. A list of users and password should show up as a result of the JDBC query:

```
dynsub | dynsub
guest | guest
j2ee | j2ee
john | needle
nobody | nobody
```

APPENDIX A. REVISION HISTORY

Revision 5.1.0-110.33.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Revision 5.1.0-110.33 **July 24 2012** **Ruediger Landmann**

Rebuild for Publican 3.0

Revision 5.1-0 **Wed Sep 15 2010** **Laura Bailey**

Changed version number in line with new versioning requirements.

Revised for JBoss Enterprise Application Platform 5.1.0.GA.