Enterprise Java for Linux HOWTO

Table of Contents

Table of Contents

Table of Contents

Enterprise Java for Linux HOWTO

Gary Meyer `gary@meyer.net`

v 0.1, January 1999

How to set up an Enterprise Java environment on Linux including a Java Development Kit, a Web server, supporting Java servlets, accessing a database via JDBC, and supporting Enterprise Java Beans (EJBs).

1.Introduction

2.How to Setup the Java Development Kit

3.How to Setup the Web Server

4.How to Setup Java Servlet Support

5.How to Setup Java Server Pages (JSP) Support

6.How to Setup JDBC Support

7.How to Setup Enterprise Java Bean (EJB) Support

1.Introduction

1.1 Background

This document was started January, 1999 by Gary Meyer (gary@meyer.net) after several weeks of installing various open source and proprietary Enterprise Java products for Linux. "Enterprise Java" is defined as using the Java Enterprise APIs.

1.2 Audience

This HOWTO is intended to benefit software professionals who are interested in evaluating, developing, or deploying Enterprise Java on Linux. Limited knowledge or experience in either Linux or Java is assumed.

1.3 New Versions

The newest version of this document can be found at the author's website at http://24.8.19.113/ej4linux.html.

1.4 Copyrights and License

1.5 Disclaimers

The suggestions in this document are provided to help you get a Enterprise Java environment on Linux up and running as quickly as possible. The suggestions are not product recommendations or endorsements. As you become familiar with the options available, you can do you own product evaluations and determine what options are best for your particular purpose.

For the purpose of this HOWTO, "Enterprise Java" is defined as using the Java Enterprise APIs. This HOWTO does not address scalability, availability, manageability, and other such aspects of software that are often associated with the word "enterprise."

1.6 Potential Future Sections

This HOW has focused on the most popular aspects of Enterprise Java. The following sections may be added to this HOWTO.

- Integrated Development Environments (IDEs) that Support Enterprise Java for Linux
- Java Naming and Directory Interface (JNDI) Support
- Java Mail API (JMAPI) Support
- Java Transaction Service (JTS) Support
- Java Interface Definition Language (JIDL) Support
- Java Messaging Service (JMS) Support
- Common Object Request Broker Architecture (CORBA) Support

Interested in authoring a section?

Please contact the author, Gary Meyer, at (gary@meyer.net).

1.7 Other Resources

The App−Serv Center website at http://www.app−serv.com/.

Java Enterprise in a Nutshell by David Flanagan et al at http://www.oreilly.com/catalog/jentnut/.

1.8 Feedback

Please submit all additions and corrections to the author, Gary Meyer, at (gary@meyer.net).

2.How to Setup the Java Development Kit

There are several Java Development Kits available for Linux. These include:

- Blackdown JDK
- IBM Java Developer Kit
- Sun J2SE
- Kaffe

If you are going to try just one JDK, I suggest you initially try the Sun J2SE, unless you are recommended otherwise by specific software you are using or want to use. Additionally, if you are interested in an open source implementation, you will need to use Kaffe.

2.1 Blackdown JDK

Background

The Blackdown JDK is a port of Sun JDK to Linux. As of the time of this writing, the Blackdown JDK is current with JDK 1.2.2 on the Intel architecture and 1.1.8 on the PowerPC.

In December 1999, Sun announced availability of the Java 2 Platform, Standard Endition (J2SE) on Linux. This Sun release has significant impact on Blackdown because Blackdown is a port. In a press release, Sun states, "This week's announcement would not have been possible without the collaboration of Blackdown, a group of developers and enthusiasts around the globe. Since its inception, Blackdown has been a provider of Java technology for the Linux platform. Their dedicated effort over a number of years has laid the foundation for this release of the Java 2 platform port to Linux; in particular their effort was critical to the success of this release."

Additionally, the Sun press release continues, "Blackdown.org continues to be a valuable source for Java technology for Linux, including JDK 1.1.x releases."

Download

The Blackdown JDK can be obtained from http://www.blackdown.org.

>From the Blackdown home page, select download and a mirror site.

Select the version of the JDK you want. If other software that you are wanting to use does not recommend a specific version, I suggest the most current, which is at the time of this writing, JDK 1.2.2.

Select the machine architecture you are installing on. For Intel architecture, select i386.

Select the release candidate you want. If other software that you are wanting to use does not recommend a specific release candidate, I suggest the most recent or final version if available.

For the Blackdown JDK, there are possibly a number of different files available in different packaging formats. Additionally you have to be sure you get support for the right libc for your Linux distribution.

The files available include:

- *jdk* – The Java Development Kit contains everything you need to compile, run, and debug Java. It does not contain international character converters.
- *jre* – The Java Runtime Environment, including international character converters.
- *rt* – A minimal Java Runtime Environment that does not include international character converters.
- *i18n* – The internationalization font mappings and a JAR containing the international character converters.
- *native* – Additional binaries providing native thread support.

I suggest downloading only the jdk for Java development in English.

When downloading the Blackdown files, you may need to select between libc5 and glibc as well as potentially a specific version of glibc. The libc options include:

- libc5 – The older, and still most common, Linux libc is libc5.
- glibc – The new Linux libc.

If you are using a newer distribution of Linux, you will most likely have glibc. I suggest initially trying glibc.

Installation

I suggest installing files in the /usr/local directory. After downloading the files, run:

```
mkdir /usr/local/blackdown
```

```
mv jdk* /usr/local/blackdown
```

If you downloaded the tarball format, run:

```
tar zxvf [filename].tar.gz
```

Where [filename] is the name of the file.

Under the /usr/local/blackdown directory, you now should see a directory such as jdk1.2.2.

The above example shows JDK 1.2.2 release candidate 3 for the Intel architecture. Substitute the file name, version number, release candidate number, and architecture as appropriate. You will need to open each distribution package file in the above manner.

Setting up Your Environment

The environment variables to set up are:

- JAVA_HOME
- PATH
- CLASSPATH

The JAVA_HOME environment variable references the home directory of your JDK installation. Set your JAVA_HOME environment variable to the directory into which you just installed a version of the Blackdown JDK.

```
export JAVA_HOME=/usr/local/blackdown/jdk1.2.2
```

The $JAVA_HOME/bin directory has the Java compiler (javac) and the Java Virtual Machine (java) as well as other necessary programs for development. Add $JAVA_HOME/bin to your PATH.

```
export PATH=$JAVA_HOME/bin:$PATH
```

Note that $JAVA_HOME/bin was added to the front of the PATH so that the installed JDK will be used rather than any JDK that might have come with your Linux distribution.

To confirm that your PATH is correctly set up, check which Java compiler and JVM will be used.

```
which javac
which java
```

The output should reference javac and java in your $JAVA_HOME/bin directory.

The CLASSPATH environment variable references all JARs and directories that you will need to compile and run Java programs.

For JDK 1.2.2, you don't need to initially add any JARs to your CLASSPATH. JARs can be packaged in either .jar or .zip files.

```
export CLASSPATH=$CLASSPATH:.
```

Confirming Your Installation

You are now ready to compile and run a simple application. Create the following program.

```
class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Compile the program with the Java compiler.

```
javac HelloWorld.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java HelloWorld
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, World!
```

Congratulations, you have installed, set up an environment for, and tested the Blackdown JDK on Linux.

More Information

For more information on the Blackdown JDK, see the Blackdown website at http://www.blackdown.org. There is an excellent FAQ available.

2.2 IBM Java Developer Kit

Background

The IBM Java Developer Kit and Runtime Environment pass Sun's Java compatibility test and include the latest maintenance. (From the IBM website.)

As of the time of this writing, the IBM Java Developer Kit is current with JDK 1.1.8 and is available only on the Intel architecture.

Download

The IBM Java Developer Kit can be obtained from http://www.ibm.com/java/jdk/118/linux.

In order to download, you will have to register with the IBM website and agree to the license online.

The files available include:

- *ibm−jdk−l118−linux−x86.tgz* − The Java Development Kit contains everything you need to compile, run, and debug Java.
- *ibm−jre−l118−linux−x86.tgz* − The Java Runtime Environment contains everything you need to run Java.

Since you will be doing Java development, I suggest downloading the ibm−jdk tarball file.

Installation

I suggest installing files in the /usr/local directory. After downloading the files, run:

```
mkdir /usr/local/ibm
mv ibm-jdk-l118-linux-x86.tgz /usr/local/ibm
```

You can now open the distribution package. To do this, type:

```
tar zxvf ibm-jdk-l118-linux-x86.tgz
```

Under the /usr/local/ibm directory, you now should see the jdk118 directory.

The above example shows JDK 1.1.8 for the Intel architecture. Substitute the filenames as appropriate.

Setting up Your Environment

The environment variables to set up are:

- JAVA_HOME
- PATH
- CLASSPATH

The JAVA_HOME environment variable references the home directory of your JDK installation. Set your JAVA_HOME environment variable to the directory into which you just installed a version of the IBM Java Developer Kit.

```
export JAVA_HOME=/usr/local/ibm/jdk118
```

The $JAVA_HOME/bin directory has the Java compiler (javac) and the Java Virtual Machine (java) as well as other necessary programs for development. Add $JAVA_HOME/bin to your PATH.

```
export PATH=$JAVA_HOME/bin:$PATH
```

Note that $JAVA_HOME/bin was added to the front of the PATH so that the installed JDK will be used rather than any JDK that might have come with your Linux distribution.

To confirm that your PATH is correctly set up, check which Java compiler and JVM will be used.

```
which javac
which java
```

The output should reference javac and java in your $JAVA_HOME/bin directory.

The CLASSPATH environment variable references all JARs and directories that you will need to compile and run Java programs.

Initially I suggest adding the following JARs to your CLASSPATH. JARs can be packaged in either .jar or .zip files.

For instance:

```
export CLASSPATH=$JAVA_HOME/lib/classes.zip
export CLASSPATH=$CLASSPATH:.
```

Confirming Your Installation

You are now ready to compile and run a simple application. Create the following program.

```
class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Compile the program with the Java compiler.

```
javac HelloWorld.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java HelloWorld
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, World!
```

Congratulations, you have installed, set up an environment for, and tested the IBM Java Developer Kit on Linux.

More Information

For more information on the IBM Java Developer Kit, see the IBM Java website at http://www.ibm.com/java.

2.3 Kaffe

Background

Kaffe is a cleanroom, open source implmentation of a Java Virtual Machine and class libraries. As of the time of this writing, Kaffe "mostly complies with JDK 1.1, except for a few missing parts." And "parts of it are already JDK 1.2 (Java 2) compatible." (From the Kaffe website.)

Kaffe may have already been shipped with your Linux distribution because of its open source license.

Download and Installation

Rather than downloading from Kaffe, I suggest you initially try the Kaffe that most likely came with your Linux distribution.

Alternatively, Kaffe can be obtained from http://www.kaffe.org.

>From the Kaffe home page, select the current release. At the time of this writing, the current release is 1.0.5. The Kaffe version number has no relationship to JDK specification version numbers.

Setting up Your Environment

The environment variables to set up are:

- PATH
- CLASSPATH

To confirm that your PATH is correctly set up, check which Java compiler and JVM will be used.

```
which javac
which java
```

The CLASSPATH environment variable references all JARs and directories that you will need to compile and run Java programs.

Initially I suggest you add the following JARs to your CLASSPATH. JARs can be packaged in either .jar or .zip files.

For instance:

```
export CLASSPATH=/usr/local/share/kaffe/Klasses.zip
export CLASSPATH=$CLASSPATH:.
```

Confirming Your Installation

You are now ready to compile and run a simple application. Create the following program.

```
class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Compile the program with the Java compiler.

```
javac HelloWorld.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java HelloWorld
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, World!
```

Congratulations, you have installed, set up an environment for, and tested Kaffe on Linux.

More Information

For more information on Kaffe, see the Kaffe website at http://www.kaffe.org.

2.4 Sun J2SE

Background

The Sun Java 2, Standard Edition (J2SE) is Sun's production release of the Java 2 Platform for the Linux operating system. As of the time of this writing, J2SE is current with JDK 1.2.2 on the Intel architecture.

Download

J2SE can be obtained from http://developer.java.sun.com/developer/earlyAccess/j2sdk122.

You will need to register with Sun and agree to the license online before downloading.

Installation

I suggest installing files in the /usr/local directory. After downloading the files, run:

```
mkdir /usr/local/sun
mv jdk1_2_2rc1-linux-i386.tar.gz /usr/local/sun
```

You can now open the distribution package. To do this, type:

```
tar zxvf jdk1_2_2rc1-linux-i386.tar.gz
```

Under the /usr/local/sun directory, you now should see the jdk1.2.2 directory.

The above example shows JDK 1.2.2 release candidate 1 for the Intel architecture. Substitute the filenames as appropriate.

Setting up Your Environment

The environment variables to set up are:

- JAVA_HOME
- PATH
- CLASSPATH

The JAVA_HOME environment variable references the home directory of your JDK installation. Set your JAVA_HOME environment variable to the directory into which you just installed a version of J2SE.

```
export JAVA_HOME=/usr/local/sun/jdk1.2.2
```

The $JAVA_HOME/bin directory has the Java compiler (javac) and the Java Virtual Machine (java) as well as other necessary programs for development. Add $JAVA_HOME/bin to your PATH.

```
export PATH=$JAVA_HOME/bin:$PATH
```

Note that $JAVA_HOME/bin was added to the front of the PATH so that the installed JDK will be used rather than any JDK that might have come with your Linux distribution.

To confirm that your PATH is correctly set up, check which Java compiler and JVM will be used.

```
which javac
which java
```

The output should reference javac and java in your $JAVA_HOME/bin directory.

The CLASSPATH environment variable references all JARs and directories that you will need to compile and run Java programs.

For JDK 1.2.2, you don't need to initially add any JARs to your CLASSPATH. JARs can be packaged in either .jar or .zip files.

```
export CLASSPATH=$CLASSPATH:.
```

Confirming Your Installation

You are now ready to compile and run a simple application. Create the following program.

```
class HelloWorld {
  public static void main (String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Compile the program with the Java compiler.

```
javac HelloWorld.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java HelloWorld
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, World!
```

Congratulations, you have installed, set up an environment for, and tested the Sun J2SE for Linux.

More Information

For more information on Sun J2SE, see the Sun Java website at http://java.sun.com. There are excellent discussion forums available where you might be able to find answers to various questions.

3.How to Setup the Web Server

There are several Web Servers available for Linux. These include:

- Apache
- IBM Domino
- IBM HTTP Server

If you are going to try just one Web Server, I suggest you initially try Apache, principally because it comes with most major Linux distributions and may already be installed, and perhaps running, on your system.

3.1 Apache

Background

Apache is the most popular HTTP server on the Internet. It was originally based upon the NCSA httpd and has since been completely rewritten. It is Open Source licensed. (From the Apache website.)

Download, Installation, and Setting up Your Environment

Rather than downloading from Apache, I suggest you initially try the Apache that most likely came with your Linux distribution.

Alternatively, Apache can be obtained from http://www.apache.org.

Confirming Your Installation

To confirm that Apache is installed and running on your computer, open your web browser, and enter the URL: "http://127.0.0.1". (127.0.0.1 is the IP address for the localhost.)

You should see a web page to the effect of "It Worked!"

If it did not work, you can confirm that Apache is installed by typing the following on a RedHat Package Manager (RPM)−based Linux distribution.

```
rpm −q | grep apache
```

To start Apache, type:

```
cd /etc/rc.d/init.d
./httpd start
```

Note: The httpd script used at boot time may be in a different location on other Linux distributions.

For more assistance, I suggest you look into the Apache FAQ at http://www.apache.org/docs/misc/FAQ.html.

3.2 IBM Domino

To be written.

See http://www.lotus.com/dominolinuxfor more information.

3.3 IBM HTTP Server

Background

The IBM HTTP Server is an IBM repackaging of Apache. You might consider using the IBM HTTP Server if you plan on working with IBM WebSphere.

At the time of this writing, the most recent version is 1.3.6.1.

Download

The IBM HTTP Server can be obtained from
http://www−4.ibm.com/software/webservers/httpservers/download.html.

Click on the download link and select 56−bit or 128−bit SSL encryption.

You will need to register with IBM, fill out a marketing survey, and accept the license agreement before downloading. The IBM HTTP Server requires glibc either version 2.0 or 2.1. glibc is the new Linux libc. If you have an older distribution that is based upon libc5, you will not be able to use the IBM HTTP Server.

On an RedHat Package Manager (RPM)−based Linux distribution, you can run:

```
rpm −qa | grep libc
```

You will see output such as:

```
glibc−2.1.2−11
libc−5.3.12−31
```

This will show you which versions of libc5 and glibc you have installed on your Linux distribution. In my above example I have both glibc and libc5 installed on my system. glibc is version 2.1, so I would want to download the files for glibc2.1.

I suggest downloading all of the tar files for the glib version of Linux that you have as they are relatively small. However, minimally you will need the server file. For RedHat 6.0 and distributions derived from Redhat 6.0 you will also need the redhat60only. For glibc2.0 based distributions you will also need the libstdc file.

Installation

I suggest installing files in the /usr/local directory. After downloading the files, run:

```
mkdir /usr/local/ibm
mv HTTPServer.linux.* /usr/local/ibm
```

You can now open the distribution package file or files.

```
tar xvf [filename].tar
```

Where [filename] is the name of the file.

Under the /usr/local/ibm directory, you now should see the directory IHS.

Use the RedHat Package Manager (RPM) to install the rpm files that the tar file produced. If you needed the libstdc file, you will need to install that file first. Then you would install the server RPM file such as:

```
cd IHS
rpm -i IBM_HTTP_Server-1.3.6-2.i386.rpm
```

The above example shows version 1.3.6 for the Intel architecture. Substitute the filename as appropriate.

After installing you can delete the rpm files as they can be easily recreated from the tar files.

Setting up Your Environment

The environment variables to set up are:

- PATH

The IBM HTTP Server installed itself into /opt/IBMHTTPServer. You need to add its bin directory to your PATH.

```
export PATH=/opt/IBMHTTPServer/bin:$PATH
```

Note that /opt/IBMHTTPServer/bin was added to the front of the PATH so that the installed Web Server will be used rather than any Web Server that might have come with your Linux distribution.

To confirm that your PATH is correctly set up, check which Apache controller will be used. Type:

```
which apachectl
```

The output should reference apachectl in the /opt/IBMHTTPServer/bin directory.

Note: because the IBM HTTP Server is based upon Apache, it uses the Apache controller to start and stop it. When you have both the IBM HTTP Server and Apache installed on a computer, take particular care to your PATH to make sure you are working with the correct server.

You may need to modify the IBM HTTP Server configuration file. The configuration file was installed in /opt/IBMHTTPServer/conf/httpd.conf. The most common two entries that need to be changed are the ServerName and the Listen port number. Look for the key words "ServerName" and "Listen" in the httpd.conf file. The ServerName should be set to either your hostname or IP address. If your computer uses DHCP to acquire an IP address, the hostname is a better candidate. However, in order to use your computer's hostname, your computer's hostname and IP address must be properly registered in DNS.

Additionally if you are running another Web server on the computer you need to assign the IBM HTTP Server to another port so you can run both Web servers simultaneously if necessary.

The following is an example entry in httpd.conf.

```
ServerName 192.168.0.4
Listen 3000
```

Confirming Your Installation

To start the IBM HTTP Server, type the following:

```
/opt/IBMHTTPServer/bin/apachectl start
```

To confirm that the IBM HTTP Server is installed and running on your computer, open your web browser, and enter the URL: "http://192.168.0.4:3000" substituting the correct IP address and port number entered into httpd.conf.

You should see a web page to the effect of "Welcome to the IBM HTTP Server". Contratulations, you have installed, set up an environment for, and tested the IBM HTTP Server for Linux.

More Information

For more information, I suggest you look into the IBM HTTP Server Support page at http://www−4.ibm.com/software/webservers/httpservers/support.html.

4.How to Setup Java Servlet Support

There are several Web Server plug−ins and Application Servers available for Linux that provide support for Java Servlets. These include:

- Allaire JRun
- Apache JServ
- BEA WebLogic
- Enhydra
- Locomotive
- IBM Websphere

If you are going to try just one, I suggest you initially try Apache JServ.

4.1 Allaire JRun

To be written.

See http://www.allaire.com/products/jrun/for more information.

4.2 Apache JServ

Background

Apache JServe is a 100% pure Java servlet engine fully compliant with the Java Servlet 2.0 specification. Apache JServ is part of the Java Apache Project. (From the Apache Java Project website).

Download

Apache JServ can be obtained from http://java.apache.org/jserv/index.html.

>From the Apache JServ Project home page, follow the Download Apache JServ link.

Currently RPM distributions are available for RedHat Linux. For other Linux distributions you will have to build from source. The following example describes how install the RPM for RedHat 6x.

As of the time of this writing, the current version is 1.1b3.

Installation

I suggest installing files in the /usr/local directory. After downloading the files, run:

```
mkdir /usr/local/apachejserv
mv ApacheJServ*.rpm /usr/local/apachejserv
```

For RedHat and RedHat–derived distributions, use the RedHat Package Manager (RPM) to install the rpm file such as:

```
rpm -i ApacheJServ-1.1-b2_RH6x.i386.rpm
```

The above example shows version 1.1–b2 for the RedHat 6x on the Intel architecture.

Setting up Your Environment

You will need to stop, set your Java environment variables, and restart Apache to register Apache JServ.

To stop Apache, type:

```
cd /etc/rc.d/init.d
./httpd stop
```

Note: The httpd script used at boot time may be in a different location on other Linux distributions

To set you Java environment, see the How to Install the JDK section of this document, specifically for the JDK you intend to use. You need to set several properties in the jserv.properties file installed in /etc/httpd/conf/jserv. Specifically, look for:

- wrapper.bin – to reference the JDK you installed
- wrapper.classpath – to minimally include /usr/lib/apache/ApacheJServ.jar and /home/httpd/classes/servlet–2.0.jar
- bindaddress=localhost
- port=8007

To restart Apache, type:

```
cd /etc/rc.d/init.d
./httpd start
```

Confirming Your Installation

To confirm that the Apache JServ is installed and running on your computer, open your web browser, and enter the URL: "http://127.0.0.1/servlet/IsItWorking" substituting the correct IP address if you are browsing from another machine.

You should see a web page to the effect of "Yes, It's Working!". Contratulations, you have installed, set up an environment for, and tested the Apache JServ for Linux.

For more assistance, I suggest you look into the Apache JServ website at

http://java.apache.org/jserv/index.html.

Now, to compile and run your own servlet. Enter the following Java servlet program.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
  public void service (HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
  }
}
```

More Information

For more information, I suggest you look into the Java Apache Project website at http://java.apache.org/.

4.3 BEA WebLogic

See BEA WegLogic below.

4.4 Enhydra

To be written.

See http://www.enhydra.orgfor more information.

4.5 IBM WebSphere

To be written.

See http://www–4.ibm.com/software/webservers/appserv/linux.htmlfor more information.

4.6 Locomotive

To be written.

See ://www.locomotive.org/for more information.

5.How to Setup Java Server Pages (JSP) Support

To be written.

5.1 Apache Jakarta

To be written.

See ://jakarta.apache.com/for more information.

5.2 Caucho Resin

To be written.

See ://www.caucho.com/for more information.

6.How to Setup JDBC Support

There are several databases that run on Linux that also support a JDBC interface. These include:

- IBM DB2
- MiniSQL
- MySQL
- Oracle
- PostgreSQL
- Sybase

If you are going to try just one DBMS, I suggest you initially try PostgreSQL, principally because it comes with most major Linux distributions and may already be installed on your system.

6.1 IBM DB2

To be written.

See http://www−4.ibm.com/software/data/db2/linux/for more information.

6.2 MiniSQL

To be written.

See http://www.hughes.com.au/for more information.

6.3 MySQL

To be written.

See http://www.mysql.org/for more information.

6.4 Oracle

To be written.

See http://platforms.oracle.com/linux/

6.5 PostgreSQL

Background

PostgreSQL is a sophisticated Object−Relational DBMS, supporting almost all SQL constructs, including subselects, transactions, and user−defined types and functions. It is the most advanced open−source database available anywhere. Commercial Support is also available from PostgreSQL, Inc. The current version is 6.5.3 and is available at any of the many mirror sites or on CD. (From the PostgreSQL website.)

PostgreSQL may have already been shipped with your Linux distribution because of its open source license.

Download and Installation

Rather than downloading from PostgreSQL, I suggest you initially try the PostgreSQL that most likely came with your Linux distribution.

Alternatively, PostgreSQL can be obtained from http://www.postgresql.org.

To confirm that PostgreSQL is installed on your computer, type:

```
rpm −qa | grep postgresql
```

or

```
which postmaster
which psql
```

You need the postgresql, postgresql−server, and postgresql−java packages installed to use Java with PostgreSQL.

Make sure PostgreSQL is running. Type:

```
ps −f −u postgres
```

You should see postmaster, the PostgreSQL daemon, running.

If postmaster is not running, there will probably be a Sys V Init script that you can use to start it. In many distributions it is located in /etc/rc.d/init.d. To start PostgreSQL, type:

```
cd /etc/rc.d/init.d
./postgresql start
```

You can use the above "ps" command to confirm that PostgreSQL is running.

Note: To use JDBC, PostgreSQL needs to have been started with the '−i' parameter indicating support for TCP/IP connections rather than solely UNIX domain sockets. Confirm that postmaster> was started with the '−i' paramter.

Create a test database by typing:

```
su − postgres
createdb javatest
```

You should see no error messages.

Create a test table with one test row. First, log in to the interactive PostgreSQL tool and connect to the javatest database you just created by typing (as the postgres user):

```
psql javatest
```

You should see confirmation that you are connected to the database: javatest.

Then, create the test table by typing (within psql):

```
create table test (col1 varchar(255));
```

You should see the "CREATE" confirmation message.

Next, insert one row by typing (within psql):

```
insert into test (col1) values ('Hello, from PostgreSQL!');
```

You should see the "INSERT" confirmation message.

Finally, confirm that the row is there by typing (within psql):

```
select col1 from test;
```

You should see the row you just created.

You can exit psql by typing "\ q".

For more assistance on working with PostgreSQL, I suggest you look into the Database−SQL−RDBMS HOW−TO document for Linux (PostgreSQL Object Relational Database System) at http://metalab.unc.edu/mdw/HOWTO/PostgreSQL−HOWTO.html.

You will need to add the appropriate JAR to your CLASSPATH. The PostgreSQL JARs come in the *postgresql−jdbc* package.

```
export CLASSPATH=$CLASSPATH:/usr/lib/pgsql/jdbc6.5−1.2.jar
```

You may need to substitute the path depending you where PostgreSQL is installed on your system.

Confirming Your Installation

You are now ready to compile and run a simple JDBC application that uses PostgreSQL. Create the following program.

```
import java.sql.*;

class PostgreSQLTest {
  public static void main (String[] args) {
    try {
      Driver driver = (Driver)
        Class.forName("postgresql.Driver").newInstance();
      DriverManager.registerDriver(driver);

      String     url = "jdbc:postgresql:javatest";
      Connection con = DriverManager.getConnection(url, "postgres", "");
      Statement  stm = con.createStatement();

      stm.setQueryTimeout(10);
      ResultSet  rs  = stm.executeQuery("select col1 from test");

      rs.next();

      System.out.println(rs.getString(1));

    } catch (SQLException e) {

      System.out.println("Exception!");
      System.out.println(e.toString());
  }
}
```

Compile the program with the Java compiler.

```
javac PostgreSQLTest.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java PostgreSQLTest
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, from PostgreSQL!
```

Congratulations, you have installed, set up an environment for, and tested a JDBC interface to PostgreSQL.

More Information

For more information, I suggest you look into the PostgreSQL website at

http://www.postgresql.org/.

6.6 Sybase

Background

Sybase Adaptive Server Enterprise is a commericial RDBMS that is available for the Linux operating system. While Sybase has recently released version 12.0, version 11.9.2 is available for Linux.

According to the Sybase website, "By porting ASE to Linux, Sybase provides the Linux development community with the first highly scalable, high−performance database engine available for the platform. The package includes the standard features of Adaptive Server Enterprise and all related connectivity components. Adaptive Server Enterprise 11.9.2 is offered FREE for development."

Download

The Sybase ASE can be obtained from
http://www.sybase.com/products/databaseservers/linux/linux1192_reg.html.

In order to download, you will have to register with the Sybase website and agree to the license online.

The Sybase JDBC driver can be obtained from http://www.sybase.com/products/internet/jconnect/.

Select download jConnect 4.2/5.2.

If you have access to a Sybase server on the network, you only need to download and install the JDBC driver.

Installation

Installation of Sybase is beyond the scope of this HOWTO. This HOWTO will assume that Sybase has been correctly installed and configured and that you can get to Sybase using isql.

Log into isql as sa and create a test user and test database by typing:

```
create database javatest
go
sp_addlogin javatest, javatest, javatest
go
use javatest
go
sp_dbowner javatest
go
```

You should see no error messages.

Create a test table with one test row. First, log in to isql as the javatest user and type:

```
create table test (col1 varchar(255))
go
```

You should see no error messages.

Next, insert one row by typing:

```
insert into test (col1) values ('Hello, from Sybase!')
go
```

You should see no error messages.

Finally, confirm that the row is there by typing:

```
select col1 from test
go
```

You should see the row you just created.

You can exit isql by typing "exit".

For more assistance on working with Sybase, review the documentation that can be downloaded with Sybase.

You will need to add the appropriate JAR to your CLASSPATH.

```
export CLASSPATH=$CLASSPATH:/usr/local/sybase/jConnect-5_2/classes/jconn2.jar
```

You may need to substitute the path depending you where jConnect is installed on your system.

Confirming Your Installation

You are now ready to compile and run a simple JDBC application that uses Sybase. Create the following program.

```
import java.sql.*;

class SybaseTest {
  public static void main (String[] args) {
    try {
      Driver driver = (Driver)
        Class.forName("com.sybase.jdbc2.jdbc.SybDriver").newInstance();
      DriverManager.registerDriver(driver);

      String     host = "127.0.0.1";
      String     port = "4100";

      String     url = "jdbc:sybase:Tds:" + host + ":" + port;
      Connection con = DriverManager.getConnection(url, "javatest", "javatest");
      Statement  stm = con.createStatement();

      stm.setQueryTimeout(10);
      ResultSet  rs  = stm.executeQuery("select col1 from test");

      rs.next();

      System.out.println(rs.getString(1));

    } catch (SQLException e) {

      System.out.println("Exception!");
      System.out.println(e.toString());
  }
}
```

You will need to substitute the host and port number of you Sybase server as appropriate. See $SYBASE/interfaces and the $DSQUERY entry for what values to use for the host and port number.

Compile the program with the Java compiler.

```
javac SybaseTest.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Run the program with the JVM.

```
java SybaseTest
```

If the JVM produces errors, confirm your PATH and CLASSPATH.

You should see the following output:

```
Hello, from Sybase!
```

Congratulations, you have installed, set up an environment for, and tested a JDBC interface to Sybase.

More Information

For more information, I suggest you look into the Sybase jConnect website at http://www.sybase.com/products/internet/jconnect/.

7.How to Setup Enterprise Java Bean (EJB) Support

To be written.

7.1 BEA WebLogic

To be written.

See http://www.beasys.com/linux/for more information.

7.2 EJBoss

Background

The EJBoss organization strives at delivering a complete J2EE compliant platform to the open source web. (From the EJBoss website).

As of the time of this writing, the most recent version is 0.95.

Download

EJBoss can be downloaded from the Telkel website at http://www.telkel.com/ejboss.

As no implementation for JDK 1.3 is available for Linux, select ejboss095_jdk122.jar.

Installation

I suggest installing files in the /usr/local directory. After downloading, run:

```
mkdir /usr/local/ejboss
mv ejboss* /usr/local/ejboss
```

Unjar the file:

```
jar xvf ejboss095_jdk122.jar
```

You should see various files and directories created under /usr/local/ejboss.

The above example shows EJBoss 0.95 for JDK 1.2.2. Substitute the file names as appropriate.

Setting up Your Environment

The environment variables to set up are:

- CLASSPATH

The CLASSPATH environment variable references all JARs and directories that you will need to compile and run Java programs.

Include the EJBoss JAR and the beans/generated directory in your CLASSPATH.

```
export CLASSPATH=/usr/local/ejboss/lib/ejboss095_jdk122.jar:/usr/local/ejboss/beans/generated:$CLASS
```

Confiming Your Installation

You are now ready to compile and run a simple EJB application. Create the following three source files for the server.

First, the business interface.

```
// EJBTest.java

import javax.ejb.*;
import java.rmi.RemoteException;

public
 interface EJBTest extends EJBObject {
  public String greet() throws
 RemoteException;

}
```

Second, the home interface.

```
// EJBTestHome.java

import javax.ejb.*;
import java.rmi.RemoteException;

public
 interface EJBTestHome extends EJBHome {

  public EJBTest create() throws

   CreateException, RemoteException;
```

```
}
```

Third, the bean implementation class.

```
// EJBTestBean.java

import javax.ejb.*;
import java.rmi.RemoteException;

public
 interface EJBTestBean implements SessionBean {

  private SessionContext
 mContext = null;

  public void ejbPassivate() {
    System.out.println("EJBTestBean
 passivated.");
}

  public void ejbActivate() {
    System.out.println("EJBTestBean
 activated.");
}

  public void ejbCreate() {
    System.out.println("EJBTestBean
 created.");
}

  public void ejbRemove() {
    System.out.println("EJBTestBean
 removed.");
}

  public void setSessionContext() {
    System.out.println("EJBTestBean
 context set.");
    mContext = context;
}

  public String greet()
 {
    return "Hello, I'm an EJB!";
}

}
```

Compile the server source files with the Java compiler:

```
javac EJBTest*.java
```

If the compiler produces errors, double check the syntax and confirm your PATH and CLASSPATH.

Now that you have successfully written and compiled the server source files, you need to deploy your bean to EJBoss. Deploying a bean to EJBoss requires several steps that must be performed exactly.

First, create the file ejb–jar.xml.

```
<?xml version="1.0" encoding="Cp1252"?>
```

```
<ejb-jar ID="">

     <description></description>

     <display-name></display-name>

     <small-icon></small-icon>

     <large-icon></large-icon>

     <ejb-client-jar></ejb-client-jar>

     <enterprise-beans>

       <session>

         <description>Nextgen bean</description>

         <ejb-name>nextgen.EJBTest</ejb-name>

         <home>EJBTestHome</home>

         <remote>EJBTest</remote>

         <ejb-class>EJBTestBean</ejb-class>

         <session-type>Stateful</session-type>

         <transaction-type>Bean</transaction-type>

         <env-entry>

           <description></description>

           <env-entry-name></env-entry-name>

           <env-entry-type>java.lang.String</env-entry-type>

           <env-entry-value></env-entry-value>

         </env-entry>

         <resource-ref>

           <description></description>

           <res-ref-name></res-ref-name>

           <res-type></res-type>

           <res-auth>Container</res-auth>

         </resource-ref>

       </session>

     </enterprise-beans>

     <assembly-descriptor />
```

```
</ejb-jar>
```

The above file, which must be named ejb−jar.xml identifies the interface and class names of files that you just created as well as a name for the object.

Second, relative to the directory of the three class files you just created, create a META−INF directory.

```
mkdir META-INF
mv ejb-jar.xml META-INF
```

Third, package all four files into a jar.

```
jar cvf EJBTest.jar EJBTest*.class META-INF/ejb-jar.xml
```

You should see that it added the manifest as well as the three class files and the XML deployment descriptor file.

Fourth, put the JAR you just created in the EJBoss beans directory.

```
mv EJBTest.jar /usr/local/ejboss/beans
```

Fifth, move the class files you created to the EJBoss beans/generated directory.

```
mv EJBTest*.class /usr/local/ejboss/beans/generated
```

(This fifth step is redudant due to a bug in EJBoss 0.95.)

You are now ready to start the EJBoss server.

cd /usr/local/ejboss

sh server.sh

You should see the proxy files compile automatically and confirmation that your EJB is deployed.

You are now ready to write, compile and test the simple client applicaiton.

7.3 Bullsoft JOnAS EJB

To be written.

See http://www.bullsoft.com/ejb/for more information.