CONTENTS:

1 Servlets and JSPs
   1.1 Servlets
   1.2 JSPs

2 Web Servers, Application Servers
   2.1 Tomcat Server
   2.2 IIS Server
   2.3 Sun Java Web Server
   2.4 JBoss Application Server
       2.4.1 Installation, Configuration

3 TOMCAT SERVER
   3.1 Installation of JDK, JRE
   3.2 Setting Java Environment
   3.3 Installing Tomcat
   3.4 As root user
   3.5 As non root user
   3.6 Starting and Stopping Tomcat

4 Deploying Tomcat
   4.1 Creating Directory Structure
   4.2 Creating web.xml File

5 Administering Tomcat Server
   5.1 Administration Tool Web Application
   5.2 The Manager Web Application
   5.3 The Status Web Application

6 Security
   6.1 Changing port 8080 to port 80
   6.2 Running on port 80 as non root
   6.3 Tomcat Security Manager

7 Useful Links
1.1 SERVLETS:
Servlets are memory-resident Java programs, running inside a servlet container (e.g., Tomcat!). Because they're memory-resident, they can quickly respond to requests, as they do not incur the overhead of process creation and subsequent cleanup, unlike CGI-based scripting, e.g. perl, etc.

"The Java™ Servlet API provides web developers with a simple, consistent mechanism for extending the functionality of a web server and for accessing existing business systems. A servlet can almost be thought of as an applet that runs on the server side -- without a face."

1.2 JSPs:
"JSP technology is an extension of the servlet technology created to support authoring of HTML and XML pages. It makes it easier to combine fixed or static template data with dynamic content."

JSP is comparable to other technologies such as PHP and ASP, which combine programming/scripting with a markup language like HTML. The key difference is the programming language of choice.

For example, PHP uses a C/C++/Java hybrid, ASP uses VBScript, and JSP utilizes the full power of the Java programming language. There have been many comparisons of these technologies, and each has its place in the astute developer's toolbox.

2 WEB SERVER:
A web server is a simple server that understands HTTP requests and responds using HTTP, mainly deals with sending HTML for display in a Web browser.

The web server on receiving http request which can be handled by a servlet, passes it on to servlet which will handle it & may generate an HTML page which web server delivers to webbrowser.

JAVA WEB SERVER:
Java-enabled web server means it has JVM & servlet engine which can run java servlets or JSP.
APPLICATION SERVER:
An application server provides access to business logic for use by client application programs. It will usually have web server capability, servlet container capability, and a container that implements other fun stuff too - like EJBs - plus they usually have XML libraries and web services too. An application server is an adaptation on the webserver to suit modern requirements for modern web applications.

2.1 TOMCAT WEB SERVER:
Tomcat is an open-source HTTP server from the Apache Foundation, written in Java, that supports Servlets and JSP. Tomcat is primarily a servlet/JSP container. It's written in Java. It can serve static content, too, but its main purpose is to host servlets and JSPs. Although it's possible to get Tomcat to run Perl scripts and the like, you wouldn't use Tomcat unless most of your content was Java.

2.2 Microsoft IIS WEB SERVER:
IIS runs only on Windows OS. The most obvious advantage of IIS is its tight integration with Microsoft and Windows technologies. It is easy to install/use and has instant access to databases. This strong integration of IIS with Windows has led hackers to develop endless streams of viruses that target Windows desktops via IIS.
It is more system resource hungry.

2.3 Sun’s Java Web Server:
Latest Release 6.1 boasts of Security Features:

**Header Masking:** This feature hides Web server header details. It also removes file extensions.
Runs in user space, not kernel space.
This design element prevents potential Web server exploits from accessing operating system services.
Creating custom error pages (e.g., 404 errors) for all of your Web applications eliminates another potential identifying feature of the Java System Web Server.

2.4 JBoss Application Server:

JBoss (pronounced Jay Boss) is an open source, Java based application server. Because it is Java based, JBoss can be used on any operating system that supports Java. It is an open source project.
JBoss AS is the #1 Java application server on the market. What makes it even better is the fact that it integrates **Hibernate**, **Apache Tomcat**, **EJB 3.0**, and **JBoss Cache** into its microkernel foundation so you don't have to.

To run JBoss, you will need the following tools:

- Java 2 Standard Edition SDK 1.3.1 or higher

JBoss is available at Jboss.org in two flavors: the standard bundle, which includes Jetty for web pages and servlets, and the Tomcat bundle, which includes (you guessed it) Tomcat instead of Jetty.

JBoss 3.2.1 is the latest stable release.

### 2.4.1 Installation

To install JBoss, first unzip the distribution file to a directory of your choice. A directory named JBoss-version should be created, and it should contain the following directories:

- bin -- scripts
- client -- client jars
- docs -- documentation
- lib -- server jars
- server -- server configurations

Set the following environment variables:

- JBOSS_HOME - Set to JBoss installation directory.
- JAVA_HOME - Set to Java 2 SDK home directory

### 2.4.2 Server Configurations

**JBoss comes with three server configurations:**

- Minimal - This configuration includes only logging, the JNDI service, and the URL deployment scanner. You would want to use this configuration for starting services that don't require J2EE or as the base for a custom configuration.
- Default - This configuration is the default. It includes all of the J2EE services exception RMI/IIOP and clustering.
- All - This configuration includes all JBoss services.
Startup

To start JBoss:

1. Open a shell.
2. cd $JBOSS_HOME/bin
3. ./run [-c default|minimal|all]

Once the server is started successfully, you can verify that all of the J2EE services are up and running by opening http://localhost:8080/jmx-console/ in your browser.

To shutdown JBoss in the window or shell where it is running, press Ctrl-C.

To shutdown JBoss from another window or shell:

- Open a shell.
- cd $JBOSS_HOME/bin
- ./shutdown -S or ./shutdown --server=\url

3 TOMCAT SERVER:

Why use Tomcat?

There are 3 reasons why I use and recommend Tomcat:

1. It is free. Free for commercial and non-commercial use under the Apache license.

2. It has a large installed base of users. If you are using proprietary software, people are not so willing to share expertise and you will probably end up shelling out money to resolve issues.

3. It is a reference implementation. The good thing about reference implementations is that, if you know how to deploy applications in this servlet engine, you should have no problems with other servlet engines, commercial or otherwise.

NOTE:

If you are using Fedora Core 2, you will need to install the upgrade kernel-2.6.6-1.427 or better. The stock i686 kernel that comes with the FC2 distribution will hang your system when anything Java is run on it.
Installation

Prerequisites

1. Java Developer Kit
2. Apache Tomcat 5.x
3. Apache Web Server (optional)

3.1 Installing the Java Developer Kit

Installing the j2sdk
As root user,

```
$ cd /downloads
$ ./j2sdk*.bin - (This will extract the RPM file)
$ rpm -ivh j2sdk* - (This will install in the /usr/java directory)
```

3.2 Installing the jre
As root user,

```
$ cd /downloads
$ ./jre*.bin - (This will extract the RPM file)
$ rpm -ivh jre* - (This will install in the /usr/java directory)
```

Setting the java environment
As root user,

```
$ cd /usr/java
$ cd /etc/profile.d

Create a file called java.sh in the profile.d directory:

-------- java.sh --------
# Set java environment

export JAVA_HOME= /usr/java/java
export PATH= $PATH: $JAVA_HOME/bin
export CLASSPATH= $JAVA_HOME/lib
```
3.3 Setting Environment Variables
As root user,

```
JAVA_HOME=/usr/java/j2sdk xx
TOMCAT_HOME=/usr/local/tomcat
```

3.4 Installing Tomcat as root

There are many ways that you can get Tomcat installed on your Linux system. You could use the rpm's that are now available on jpackage.org or the tarballs from the Jakarta site. This document covers only the Jakarta tarballs.

If you have already downloaded the binary, simply copy it to the directory you want to install it under and expand the file. For example, if I downloaded the Tomcat tar file into /downloads.

```
$ tar -zxvf jakarta-tomcat-5.x.xx.tar.gz
$ ln -s jakarta-tomcat-5.x.xx tomcat //Create symbolic link to tomcat directory
```

Insert the following lines inside /etc/profile

```
export JAVA_HOME=usr/java/j2sdk xx
export CATALINA_HOME=usr/local/tomcat
```

We will need to ensure that CATALINA_HOME and JAVA_HOME are correctly set. Check the ENV variables typing in the following.

```
echo $JAVA_HOME
echo $TOMCAT_HOME
```

3.6 Starting and Stopping Tomcat
Start Tomcat with the following command, as root,
To stop Tomcat, as root,

```bash
# $CATALINA_HOME/bin/shutdown.sh
```

### 3.5 RUNNING TOMCAT AS NON ROOT:

At this stage, the Tomcat packages, files and binaries are owned by root. We will first need to create a Tomcat user and group that will own these files, and under which Tomcat will run.

**Tomcat User :: tomcat, Tomcat Group :: tomcat**

```bash
groupadd tomcat
useradd -g tomcat -d /usr/local/tomcat
passwd tomcat
```

Now put everything in `/usr/local/tomcat` under Tomcat user and group. As root,

```bash
chown -R tomcat:tomcat /usr/local/tomcat
```

Verify that JAVA_HOME and CATALINA_HOME environment variables are setup for tomcat user, and you should be good to go. Once the Tomcat binaries are under Tomcat user, the way you invoke it will be different.

### 3.6 Starting and Stopping Tomcat

To start Tomcat

```bash
su - tomcat -c /opt/tomcat/bin/startup.sh
```

To stop Tomcat

```bash
su - tomcat -c /opt/tomcat/bin/shutdown.sh
```
4 DEPLOYING WEB APPLICATION

A Java servlet/JSP engine is not really useful without servlets and JSPs, and deploying servlets can sometimes be a difficult and error-prone process, because, in addition to writing and compiling the servlet, you will need to edit at least 2 XML files to get the servlet properly deployed. There is a Manager Web Application which could be used to deploy web applications.

We will write and compile a simple "HelloWorld" servlet and deploy it, using this example to "bootstrap" future deployments.

**Creating the HelloWorld Servlet**

First, open your favorite text editor or Java IDE and create the following file:

```java
// Filename : HelloWorld.java
// Description : This servlet merely says hello!

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

public class HelloWorld extends HttpServlet {
    public void doGet ( HttpServletRequest request, HttpServletResponse response )
            throws ServletException, IOException    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>Hello, Cruel World!</title></head>");
        out.println("<body>");
        out.println("<h1>Hello, Cruel World !</h1>");
        out.println("This is my first servlet.");
        out.println("</body>");
        out.println("</html>");
    } // end doGet

}///:~
```

Now, we save the file as HelloWorld.java and compile it.
4.1 Deploying "HelloWorld"
Deployment of a web application in Tomcat consists of the following tasks:
1. Creating the directory structure
2. Creating the Context Descriptor file
3. Creating a web.xml file for the web application
4. Copying the servlets, JSPs and support files to their respective directories

4.1.1 Creating the directory structure
Before you begin deployment, you will need to create a directory structure under the $CATALINA_HOME/webapps directory that conforms to the servlet specifications. For our example "HelloWorld" servlet, we are going to create a directory called "MyFirst", and we are going to save our "HelloWorld" servlet under that directory.
As tomcat user,

```
$ cd $CATALINA_HOME/webapps
$ mkdir MyFirst
$ mkdir MyFirst/WEB-INF
$ mkdir MyFirst/WEB-INF/classes
$ mkdir MyFirst/WEB-INF/lib
```

The final web application should be laid out as shown below:

```
/opt/tomcat/
  webapps/
    MyFirst/
      index.html
      somepage.jsp
    WEB-INF/
      web.xml
    classes/
      HelloWorld.class
    lib/
      doc_drivers.jar
```
4.2 Creating a web.xml file for the web application

The file web.xml is sometimes known as a deployment descriptor, and it is basically the configuration file for your web application. In this file, you determine, among other things:

- the URL of the servlets in your web application
- the authentication method you wish to use
- Your filter definitions

For now, we will simply register HelloWorld as a servlet in the web application called MyFirst. We create a new web.xml file with the following contents inside it:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>

</web-app>
```

The two things done here are:

- Associate the servlet class file (in this case, HelloWorld.class) with a handler (HelloWorld)
- Define how the servlet is accessed via a URL

In the web.xml file above, it is stated that the servlet called HelloWorld.class is referred to as HelloWorld, and that the URL by which this servlet is to be referred to is http://hostname.domain.com:8080/MyFirst/HelloWorld.
Copying the servlets, JSPs and support files to their respective directories

Now we can copy the Java class file that we compiled earlier, into the deployment directory. Because this is a servlet, it will go into $CATALINA_HOME/webapps/MyFirst/WEB-INF/classes directory.

There is no need to copy the source file into that directory.

For more complicated deployments, such as servlets that require additional classes, such as JDBC drivers, or libraries, these additional files must be stored inside the /WEBINF/lib subdirectory of the web application directory. If you wish to make these libraries available to all web applications in Tomcat, save them inside $CATALINA_HOME/common/lib/ directory.

For JavaServer Pages (JSPs), the *.jsp files go in the same directory as the static HTML files, that is, under $CATALINA_HOME/webapps/My_Web_Application/.

Saying “Hello World”

Testing your servlet:

We are now ready to test our HelloWorld servlet. Since this is the first servlet we are testing, and we are not really sure what we will see, X Windows and opening several terminal windows to display error and information messages, as they occur is easier. Here's how we're going to test and see if our Tomcat deployment can indeed say "HelloWorld"!

1. Start X Windows and the desktop environment of your choice.
2. Log in as root
3. Open 3 terminal windows
4. In the first terminal window, execute the following commands:
5. 
   # cd $CATALINA_HOME/bin
   # ./startup.sh
6. In the second terminal window, execute the following commands:

```bash
# cd $CATALINA_HOME/logs
# tail -f catalina.out
```

This will display the messages that Tomcat writes to the log file catalina.out, which is automatically created, if it does not already exist.

7. In the third terminal window, execute the following commands:

```bash
# cd $CATALINA_HOME/logs
```

At starts up, you can see the messages it writes to the log files
If Tomcat started without errors (or even if it did have errors), we can test the servlet with our browser.

**Checking with a browser**

Open Netscape or Mozilla on the local machine

For the URL, key in: http://localhost:8080/MyFirst/HelloWorld
5 Administering Tomcat Server

There are three administration tools bundled with Tomcat 5. They are:

- Server Status application
- Tomcat Administration Tool
- Tomcat Manager

5.1 The Administration Tool Web Application

The Administration Tool web application is the GUI tool for administering Tomcat. Before we can use it, or any of the GUI tools, we will first need to create a Tomcat administrator account, a Tomcat manager account and two roles: "admin" and "manager". For example, we will create one user "TomcatAdmin" who has both manager and admin privileges.

**Administration Users and Roles**

<table>
<thead>
<tr>
<th>Tomcat Administrator Role</th>
<th>admin</th>
<th>This is a built-in role.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tomcat Manager Role</td>
<td>manager</td>
<td>This is a built-in role.</td>
</tr>
<tr>
<td>Tomcat Administrator Username</td>
<td>TomcatAdmin</td>
<td></td>
</tr>
<tr>
<td>Tomcat Administrator Password</td>
<td>tcpass</td>
<td></td>
</tr>
</tbody>
</table>

Launch the editor and add the following lines in green to $CATALINA_HOME/conf/tomcat-users.xml.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <role rolename="manager"/>
    <role rolename="admin"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
    <user username="TomcatAdmin" password="tcpass" roles="admin,manager"/>
</tomcat-users>
```
Save the file, start Tomcat and open a browser. You should be able to access the Administration Tool web application by clicking on the "Tomcat Administration" link on the left sidebar.
You will be presented with a login screen.
Type in your Tomcat Administrator username and password, and you should be able to enter the administration tool GUI.
The Administration Tool has collapsible menus on the left. Click on any of the icons in the left pane to display information and settings in the right pane. Depending on the specifications of your server, the display of settings and information in the right pane can be very slow! You have to be patient and restrain your urge to keep clicking if the system seems unresponsive.
The GUI is shown below.

![Tomcat Administration Tool](image)

**Tomcat Administration Tool**
5.2 The Manager Web Application

You can access the Manager Web Application by clicking the "Tomcat Manager" link in the left sidebar of the default Tomcat page. This will display a GUI that allows you to view the status of your installed web applications and even deploy new web applications. There is also a link to the "HTML Manager Help" which can give you more information on using the Manager Web Application.

Tomcat Manager Web Application

The Manager web application also provides a way to install a new web application, reload it, start and stop it, and many other functions by merely passing a URL. For example, if we wanted to list the applications currently deployed in Tomcat, we could open a browser and key in http://hostname.domain.com:[port]/manager/list, and it will return the list that we require.
5.3 Status Web Application

Finally, we have the Status web application, which displays the status of the Tomcat server, such as the memory utilization of the JVM, the version of JVM being run and the number and status of threads, among other parameters. This tool is new in Tomcat 4 and can be used to check if your Tomcat server is running low on system resources.
6 SECURITY:

6.1 Changing port from 8080 to 80
In Tomcat web server, the standard default port to avoid conflicts with other web servers is set to port 8080.

If using Tomcat in standalone mode (i.e. as a complete web server, not just a servlet and JSP engine integrated within another web server), it is more convenient to use port 80. That way you donot have to type in the port number on every URL you type in the browser.


Modifying port number involves editing install_dir/conf/server.xml file, changing the port attribute of the connector element from 8080 to 80.

However in Linux, as a security precaution built into the kernel, only the root user can bind to a port below 1024. Thus, any webserver, ftp server etc needs to start up as root if only to bind to their standard socket numbers. On the other hand, having a running network server as root is in itself a bad idea, so most server software only remain as root for the time it takes them to establish the initial "listening" socket, then drop to a non-privileged user for the remainder of their operation.

6.2 To run on port 80 as non root

The IPTables solution

This solution allows Tomcat to run as an independent web server, using a non-privileged user, and still respond to port 80 on the server. It requires Linux with a kernel v2.4.x and iptables installed.

IP Tables also allows you to redirect all incoming packets on, say, port 80 (the default www port) to another port such as port 8080 (Tomcat's default port).

To do this redirection, two things are needed: To ensure that the firewall allows incoming requests to port 8080, and to redirect packets from port 80 to port 8080.
The first step is needed only if you have configured your firewall to protect your machine (and if you haven't done so, you should). Use your firewall-configuration software to accept incoming packets of type tcp and destination-port 8080. Essentially, you need a duplicate of the standard rule(s) for port 80, using port 8080.

The second step is to instruct IPTables to redirect all packets which arrive on port 80, to port 8080. This is done using the following command (as root):

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j REDIRECT --to-port 8080
```

Unfortunately this will not redirect local requests, since these bypass the PREROUTING chain. Thus, any browsers or other client software running on the server itself will either have to connect directly to port 8080, or something else must be used to redirect these local requests.

**RINETD Solution:**
Use a dedicated port redirector, such as rinetd.
This is a tiny program that simply receives a packet and redirects it to a different address or port. Since this processes packets after they've passed through IPtables, it can handle locally-originating requests too. All you have to do is install it, create a file /etc/rinetd.conf containing the line "xx.xx.xx.xx 80 xx.xx.xx.xx 8080" (where xx.xx.xx.xx is your IP address), and start it as a service. Rinetd will catch and redirect those requests which bypass the IP tables redirection rule.

Unfortunately rinetd has a significant problem - it makes all packets appear as if they originate on the local machine. This means that, in the web server access logs for instance, it will be impossible to differentiate between different visitors, or to set permissions based on IP addresses.

Used together, iptables and rinetd allow tomcat to run as a non-privileged user, while still serving requests arriving via port 80. Since rinetd will only be handling locally-originating requests, it doesn't affect the originating IP-address.
6.3 Tomcat Security Manager

The Java SecurityManager protects your server from trojan servlets, JSPs, JSP beans, and tag libraries.

The security policies implemented by the Java SecurityManager are configured in the $CATALINA_HOME/conf/catalina.policy file.

The Exceptions are thrown when it detects violation.

7. USEFUL LINKS

   http://www.linux.org
   http://jakarta.apache.org/tomcat/faq/