The Florida State University
College of Arts and Sciences

An Evaluation of Web Application Server Technology

BY

Anatoly A. Reznik

TR-020404

April 2002

A project submitted to the Department of Computer Science
In partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Project Committee Members:
Dr. Greg Riccardi (Major Professor)
Dr. Dan Schwartz
Dr. Robert Van Engelen
TABLE OF CONTENTS

Acknowledgements.................................................................................................................3
Abstract.....................................................................................................................................4
Chapter 1 - Introduction.............................................................................................................5
Chapter 2 - Web Application Architecture................................................................................6
Chapter 3 - Apache Application Server Environment...............................................................8
  3.1 HTTP Server.....................................................................................................................8
  3.1.1 HTTP Server Installation...........................................................................................8
  3.1.2 HTTP Server Configuration.......................................................................................9
  3.2 Java-enabled Servers ......................................................................................................10
  3.2.1 Java Servlets.............................................................................................................10
  3.2.3 Java Server Pages......................................................................................................11
  3.3 Tomcat Application Server............................................................................................13
  3.3.1 Tomcat Installation..................................................................................................13
  3.3.2 Tomcat Web Applications.........................................................................................14
  3.3.3 Tomcat/Apache Integration.....................................................................................17
     3.3.3.1 Tomcat 3.X.......................................................................................................17
     3.3.3.2 Tomcat 4.0.......................................................................................................18
  3.3.4 Tomcat Security........................................................................................................19
Chapter 4 - Portals ..................................................................................................................21
  4.1 Portal Structure..............................................................................................................22
  4.2 Web Services..................................................................................................................22
  4.3 Jetspeed Portal Engine...................................................................................................23
     4.3.1 Jetspeed Installation.............................................................................................23
     4.3.2 Jetspeed Administration.......................................................................................24
        4.3.2.1 Adding Portlets..............................................................................................24
     4.3.2.2 Portlet API.......................................................................................................26
     4.3.3 Jetspeed Customization.........................................................................................27
Chapter 5 - XML Web Publishing............................................................................................29
  5.1 Cocoon............................................................................................................................30
     5.1.1 Cocoon Installation...............................................................................................30
     5.1.2 Cocoon Structure..................................................................................................32
     5.1.3 Adding a Pipeline..................................................................................................32
Chapter 6 – Websphere Application Server Environment......................................................34
  6.1 Application Server Concepts.........................................................................................34
     6.1.1 Enterprise Java Beans Modules............................................................................35
     6.1.2 Web and Application Client Modules..................................................................36
  6.2 Websphere Installation.................................................................................................37
     6.2.1 DB2 Installation.....................................................................................................37
     6.2.2 Application Server Installation............................................................................38
     6.2.3 Deploying an Application....................................................................................39
        6.2.3.1 Application Assembly Tool...........................................................................39
        6.2.3.2 Administrative Server................................................................................42
Chapter 7 – Conclusions..........................................................................................................45
References................................................................................................................................47
Acknowledgements

First, I would like to thank Dr. Ricardi for giving me the opportunity to work with him and his research group on something that I found very interesting and new to me. I would also like to thank him for being my major professor and giving me the guidance needed to finish this project. I would like to thank Dr. Dennis and his help in providing the necessary expertise in this area and always being there to assist in any way he could. Next, I would like to thank my committee members Dr. Dan Schwartz and Dr. Robert Van Engelen for being interested in my research topic. Finally, I would like to thank my parents for their encouragement in pursuing my education.
Abstract

This paper discusses a project that evaluates various Web tools and software packages that aid in the creation, installation, and maintenance of dynamic Web pages and Web applications. The project primarily focuses on two distinct sets of Web tools. The first set is a compilation of tools available from the Apache Software Foundation. The tools that were evaluated are parts from three projects within the Apache Software Foundation. These projects were

- HTTP Server Project
- Jakarta Project
- XML Project

The second set of evaluated tools includes software created by the IBM Corporation. Through the Scholar's Program available from IBM, I was able to acquire technologies used to create state-of-the-art Web pages and services. The tools that were used were picked to serve similar purposes as the Apache tools to demonstrate some of the advantages and disadvantages in both.

The major scope of the project involves understanding the installation, configuration, and basic maintenance of each of the tools involved. The major advantages and disadvantages of all tools were compared to come up with a recommendation on which ones were the best and easiest to use.
Chapter 1 - Introduction

The objective of this project is to create a guide that aids in the creation of dynamic Web pages and Web Applications. There are two main goals that support the objective. One of the goals is to evaluate and recommend specific tools that assist developers and administrators. The second goal was to write a manual to show proper installation and configuration of the selected tools. This will allow developers to concentrate on the task of creating their applications.

The project was broken down into three main steps. The first step involved understanding the major issues in Web application development. The next phase consisted of finding the right tools to handle the issues. The final part involved the installation and configuration of the selected tools to make them productive. The created environments were then used as the basis for the guide in this paper.

In order to evaluate the tools, it was necessary to install them, configure them to operate properly, and use them to verify their capabilities. The beginning phase of the project dealt with finding specific tools and learning about them. Once the tools were selected, I began my evaluation of specific characteristics for each tool. I created two separate environments for the development and deployment of dynamic Web applications. The first one is comprised of the tools that are available from the Apache Software Foundation. The second environment consists of tools from the IBM Websphere Project.

The tools were evaluated on such issues as ease of installation and administration, functionality, and robustness. I kept a diary of the steps needed to configure and set up each tool to be useful. This diary served as the basis for the guide that is presented in this paper. The final phase involved testing the tools by creating and running sample examples to verify the tools' effectiveness.

The successful installation and configuration of all tools allowed me to thoroughly evaluate them and create a guide for using them. The evaluations identified key problems and concerns for each tool, as well as major advantages. I also created a guide for developers and administrators to help them choose the right tools. The guide explains a step-by-step process of installing and administering each tool. This is very important because it reduces the time and effort for developers need to spend on a similar activity before creating their applications.
Chapter 2 - Web Application Architecture

The process of creating simple static Web pages has been replaced by more sophisticated Web application development. The growth of the Web has allowed it to become the favorite interface of users and has created opportunities to run new types of applications. Due to this, many Web developers are now facing the same types of problems as regular application developers [13]. These problems include such issues as code management, maintenance costs, and basic control over the application. These are all important because the Web applications are becoming very large and complex. Websites also need to be flexible so any changes will be easy to manage. Web developers also need to concern themselves with page caching and database access management to handle the large amounts of information being requested.

The development and deployment of applications to the Web can be quite tricky. Applications have to be scalable, manageable, and provide security. Applications have to communicate with different tools already deployed and fit together to provide maximum functionality. A typical Web application environment consists of four main layers – the browser, the Web server layer, the application layer, and the database layer. Figure 1 shows the basic structure of the application server model.

Begin Figure 1.

![Diagram of application server model](image)

Figure 1. The Application Server Model

The Web server is responsible for sending and receiving information from the browser. It acts as the Hypertext Transport Protocol (HTTP) server that carries out the protocol in order to successfully send data over the Internet. However, the Web server needs to relay some HTTP requests to a Web component because they require more complex operations. We are most interested in server-side Java applications. These can include Java Servlets, Java Server Pages and others, which will be discussed in more detail in Chapter 3.
The rapid growth of the Web as a means of providing information and services has created the need for more powerful tools and environments. Web servers are allowing more dynamic content and include complex applications running on them. An application server manages the development and deployment of services that create dynamic pages and services. The application server also communicates with specific tools to handle incoming requests, process them, and output results to the client. A well-configured Web server environment is important to the creation and maintenance of state-of-the-art Websites and Web applications.

The purpose of the application layer is to provide a way of developing these applications, deploying them, and making sure they are consistent with older legacy data sources that still exist. One great feature of some application servers is that they provide very powerful application development environments. These environments allow for faster completion of the projects and aid in their transition and deployment to the Web. The application development environments will be discussed later in the report to show how their features are beneficial to Web development.

Finally, the database layer is responsible for storing all the information needed to generate the Web pages. The application layer connects the Web and database layers and provides an intermediary step to generating HTML pages. The application layer receives the request from the Web server layer, connects to the database, extracts, and updates the information from it. After that, the application layer generates HTML and other presentation formats which are transmitted to the Web layer for broadcast. Many Web application servers either provide a connection scheme to a database of your choice or actually require a database application to exist before installation.

During my project I identified a number of areas within Web application development. The following parts all serve an important role in development and deployment of major Websites:

- Web Servers and Application Servers
- Portal Development
- Extensible markup Language (XML) Web Publishing
- Integrated Development Environments

The application server manages the deployment and maintenance of the Websites. It also connects various Web tools and integrates them efficiently. Portal development tools are necessary for presentation of remote and local data in customized layouts. XML Web publishing separates the presentation and information of Web pages and allows for quicker transformation of the data. Integrated Development Environments are also very important because they aid in development of code and simplify testing.
Chapter 3 - Apache Application Server Environment

The Apache Software Foundation [2] is one of the leading developers of open-source application servers available today. From the very beginning the Apache HTTP Server has been one of the most robust and reliable Web servers on the Internet. As of today 56% of all Web sites on the Internet are using Apache to serve their pages [2]. Over the last few years as Java became more popular, applications began to be deployed on the Web with more advanced tools. The Apache group has moved in that direction as well. In addition to the regular Web server, the group is working on a number of projects, which provide the necessary tools to develop some of the more advanced applications. As will be talked about later, these projects range from Java engines to XML publishing to Portal Development. There are a number of subprojects that are going on at this moment to add the tools necessary for Apache to become the only application server needed to handle any demand. The three subprojects that were used as a part of this project include the HTTP Server Project, the Jakarta Project, and the XML Project.

3.1 HTTP Server

This is the project responsible for the Web server. Although the features of the Web server in Apache have greatly increased in the last few years, for the most part it is still just a regular Web server and does not need to be talked about in great depth for a regular user, which is what the intent of my project is. The goal is to provide the simplest and most efficient way to install and setup a system in such a way that all of the new application tools are available for use and deployment. Therefore, although there is room to go into much specific detail I will concentrate on the basic ones only.

Before beginning the installation of all these packages it is important to setup a place where every tool will be placed. Under /usr/local I recommend creating a directory which will contain all the Apache tools that will be used in Web development. Once that is completed the next step is downloading the necessary files for installation. As far as the Web server is concerned the installation is pretty simple. As of the writing of this report the latest version of the server is 1.3.22. Make sure to get the .tar file for that version.

3.1.1 HTTP Server Installation

After untarring the downloaded file into the previously created directory to store the Apache tools here are the major steps that need to be followed. Perhaps the only thing that should be looked into at depth is the configuration of the install script. This is the time where all the modules will be loaded for later use. Many times the user will just follow the basic command and simply type `./configure` to run it. This will work fine, but it will not enable the necessary modules that might possibly be of use later on. Especially important is the Dynamic Shared Object (DSO), which allows loading additional modules at
runtime to the HTTP Server. This will be necessary to install Tomcat 4.0 as the Java engine.

These will be talked about later, but it's important to understand that these are bottom layers that will allow Java applications to run on the server. It does not take much longer to set up DSO during installation so it's recommended doing so even if one is unsure whether it will be needed later. The commands below will properly install the Web server and its features:

1. `./configure --prefix=PREFIX --enable-module=all`  
   `--enable-shared=all`
2. `make`  
3. `make install`

(PREFIX is the directory under which to install - ex. /usr/local/Apache)

3.1.2 HTTP Server Configuration

After installing the server there are a couple of important things that need to be mentioned before actually running it. There is one major configuration file that controls exactly how the server will operate. It is located under the directory in which Apache was installed, in the conf directory. The file is named `httpd.conf` and is responsible for loading all the modules, setting up which files will be served, and connecting to other application engines that might be installed on the system. At this point I will not go into the details of the last option because I have not talked about Tomcat. But it is important to know that this will be the place to integrate the two servers. Make sure to look over the configuration file and fix or edit any lines that seem to need a change. During installation Apache will try to guess the default values for all options specified in the configuration file. For the most part it does a pretty good job with that, but sometimes a user might change a few things. Every line in the configuration file is commented and is self-explanatory, therefore, editing it should not be much of a problem. Finally, when the server is ready to be started the following command needs to be run (/usr/local/apache is assumed to be the directory under which the server was installed):

```
/usr/local/apache/bin/apachectl start
```
3.2 Java-enabled Servers

With the growth of Web application development and the need for more sophisticated page generation, Java has become one of the leading tools used in this process. The portability among different platforms and the capabilities that Java presents allows developers the freedom to design more impressive Websites that not only offer more features, but are much easier to maintain and still only require users to reference the pages through a simple Web browser. To support Java many new Java servers have emerged to allow for behind-the-scenes processing to occur. These servers range from being completely integrated into the more extensive Web application servers, to more simple Java Web servers, to much more simple java engines that require a connection with a Web server to run on the system. Some of the new Java technology components include Enterprise Java Beans, Java Servlets, and Java Server Pages all of which have their own benefits and coexist as parts of the J2EE platform [8].

3.2.1 Java Servlets

A servlet is a method of extending a Web server by providing a mechanism to create Web-based applications [9]. Servlets run on the server side instead of the client during the application. Servlets can be quite powerful because they have access to all Java APIs and a number of HTTP-related methods. Below is an example of a simple servlet, which can serve as a foundation for more complicated ones.

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

public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML><HEAD>MyServ!</HEAD></HTML>");
    }
}
```

Begin Figure 2
Like all servlets, the first few lines need to add the packages needed to import the required classes needed to compile the application. Line 5 basically defines your servlet class. It's important that your servlet extends the HTTPServlet class to inherit all HTTP-related methods. Lines 6-8 call the doGet method, which has the HTTP Get request passed to it. Line 9 sets the content to be displayed as HTML. Finally, lines 10 and 11 are simply printing out the information to the screen. Once the servlet is compiled, by linking the browser to it would result in MyServlet! appearing on the screen.

There are two main parameters that get passed to any servlet – the request and response. These are in HTTP format and need to be converted into requests and response objects that the servlet engine can handle.

3.2.3 Java Server Pages

JSP is a fast way of serving dynamically generated Web pages. Together with Java Servlet technology they provide an efficient and more attractive way of running and displaying interactive pages. It is a scripting language that allows developers to insert Java code directly into the HTML code to create dynamic page creation. JSP uses XML-like tags as code inserted into HTML. To get the most out of JSP, the code written in Java is a lot more powerful; however, other scripting languages such as JavaScript and VBScript are also allowed [5].

JSP provide a much better alternative to dynamic page creation that has been and still is being used today. CGI scripts are old and don't come close to the functionality and robustness of JSP. ASP offers similar results as JSP; however, it is not platform-independent because it requires a Microsoft Web server and platform to serve the pages.

Below is a very simple example of how to embed JSP code in the HTML code.
Begin Figure 3

```
1  <HTML>
2  <BODY>
3  The current time is <%= new java.util.Date() %>
4  </BODY>
5  </HTML>
```

Figure 3. JSP to show current time

The Java technology presented is a new step in Web development. Through servlets and JSP not only can the content and presentation of pages be separated, but also its delivery to the client is much faster. The Java servers guarantee that once a page is displayed that process is loaded and does not need to be reloaded every time a page is refreshed. This is a huge improvement over CGI where the execution time was much greater after every time the page was refreshed. There are many types of servers out there that do many different things. Some support the whole J2EE platform, meaning they are capable of running EJB, servlets, and JSP. Some only support one or two of those and have to be configured with a Web server to run the applications.
3.3 Tomcat Application Server

Tomcat is the implementation of the Java Servlet and Java Server Pages technologies under the Apache Software Foundation Project [6]. It is a part of the Jakarta Project, which tries to create and maintain solution under the Java platform. Tomcat allows servlets and JSP to run on the application servers since it provides support for them. Once these are written and compiled they can be stored as Web applications under the tomcat directory. The applications are then interpreted by the engine and served whenever the browser calls them. Since these run on the server side they only need to be compiled once and the application is stored in Tomcat’s cache. Unlike CGI this allows for much quicker execution.

3.3.1 Tomcat Installation

The easiest way of getting Tomcat up and running on your system is to simply download the binary distribution for the system’s operating system. This works just as well as actually building the application yourself and takes much less time. The downloaded file simply needs to be unzipped into a directory, preferably under one that contains the Apache Web server since these will need to be configured to act as one application server. Since Tomcat is a Java application the JSDK environment needs to be installed on the system. Once this is done, two more things need to be set up before trying to start Tomcat. JAVA_HOME and TOMCAT_HOME variables need to be set in the classpath so the startup and shutdown scripts know where to find the necessary class files. Set JAVA_HOME to be the directory where Java is installed, ex. /usr/java and TOMCAT_HOME where the Tomcat distribution is located, ex. /usr/local/tomcat.

Even though Tomcat is meant to be just a Java server engine it can act as a stand-alone Web server as well. Before going through the trouble of configuring it with the Apache Web server it is worth the time to try to run it by itself. Not only is this a good way to see that everything was installed properly and runs correctly, but also to get familiar with how Tomcat works.

In the bin directory under TOMCAT_HOME there exist all the script necessary to start and stop Tomcat. By default after the installation Tomcat will be set up to run on port 8080 on the machine. Changing to the bin directory and typing `/startup.sh’ will start the Tomcat engine. If Java is found on the system the startup will go through a few steps and output the information to the console when the application has started. Similarly, by typing `/shutdown.sh’ the Tomcat engine can be stopped at any time. To see if Tomcat started up properly, open up a browser and simply type ‘localhost:8080’. If everything is working properly, the following is a screen shot of what should appear if Tomcat 4.0 is started. Tomcat 3.X looks very similar just a different color and format.
Figure 4. Tomcat Application Server

Tomcat comes with a number of examples that help explain how servlets work and how they are handled and interpreted by the server. This is a good way of learning how to create servlets and JSP and adding them to be served by the engine. The Tomcat documentation is linked in HTML format, which will take you step by step from developing an application and packaging it to be deployed to the Web.

3.3.2 Tomcat Web Applications

Tomcat serves Web applications built in Java, but the setup has to be followed in a specific way by the guidelines set in the Tomcat specification. The beauty of Tomcat is that it allows for the fastest execution of these applications, but the downside is that the directory
structure for each Web application has to follow the same exact pattern. Each Web application is separated in Tomcat into its own root directory under the Tomcat/webapps directory.

I will briefly go into a short explanation of the structure that each application needs to have before being deployed by Tomcat. The main directory under which all files will be stored is called the document root of the application. All of the HTML, JSP, and other client visible files must be stored in that directory. JavaScript, XML, and stylesheet files would also go into this directory. Under this directory, it's important to setup a WEB-INF directory under which the actual Java code and the configuration file will be placed. The WEB-INF/classes directory will contain any compiled Java class files that area part of the application. The WEB-INF/lib directory will contain the jar files that the application uses. Both will be added to the classpath when Tomcat starts up. Finally, the WEB-INF directory needs to contain a web.xml file. This is the deployment descriptor for the application where all the servlets are named and the basic setup for the Web application is set. Below is an example web.xml file that comes with the Tomcat documentation to give developers a template to work from.

Begin Figure 5

<!DOCTYPE Web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN" "http://java.sun.com/j2ee/dtds/Web-app_2_2.dtd">

<Web-app>

<display-name>My Web Application</display-name>
<description> My Test Application </description>

<context-param>
    <param-name>Webmaster</param-name>
    <param-value>myaddress@mycompany.com</param-value>
    <description> EMAIL address </description>
</context-param>

<servlet>
    <servlet-name>controller</servlet-name>
    <description> Controller Role </description>
    <servlet-class> com.mycompany.mypackage.ControllerServlet </servlet-class>
</servlet>

<init-param>
<param-name>listOrders</param-name>
  <param-value>com.mycompany.myactions.ListOrdersAction</param-value>
</init-param>

<init-param>
  <param-name>saveCustomer</param-name>
  <param-value>
    com.mycompany.myactions.SaveCustomerAction
  </param-value>
</init-param>

<load-on-startup>5</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>30</session-timeout>
  <!-- 30 minutes -->
</session-config>
</Web-app>

Figure 5. Sample Web.xml file

Once the application is created and all the necessary directories and files are setup, there is a very easy way to deploy it in Tomcat. The first step is to package the application as a WAR (Web archive) file. Most Java development environments allow you to do this. Then, copy the WAR file to the TOMCAT_HOME/webapps directory. Finally, restart Tomcat and the application will be automatically deployed. This is one of the best features of Tomcat since deployment is many times the hardest thing to accomplish. Let’s say the application is packaged in a school.war file and deployed in Tomcat. Once the server is restarted the context path will be created automatically under school as the application name and the application will be reachable through a browser by simply going to the localhost:8080/school URL.
### 3.3.3 Tomcat/Apache Integration

Although Tomcat can act as a stand-alone Web server and have the applications run, it is really not meant for that. It lacks most of the features and options of a real Web server like Apache and therefore is better to be used to just serve servlets and JSP files. Therefore, there needs to be a step where the integration of those two services takes place. The process itself is not very complicated if the previous steps in installation have been followed properly until this point. For Tomcat 3.X it is quite simple and for Tomcat 4.0 there a few more steps involved and require that Apache be DSO-enabled.

#### 3.3.3.1 Tomcat 3.X

This is a two-step process to completely integrate the two servers together. The whole idea behind this is that Apache will continue serving as the Web server, but transfer certain directories to be served by Tomcat. These directories will be the Web applications that are installed under the Tomcat/Webapps directory. The mod-jk.conf file under the TOMCAT_HOME/conf directory contains all the Web applications that need to be served by Tomcat. The only step needed here is to simply add the code for the new Web application. Below is the code written to serve the examples directory that comes with the Tomcat installation. Simply replace the word examples with the name of the Web application being added.

```
Begin Figure 6

# The following makes Apache aware of the /examples context
Alias /examples "/$USER/local/portal/tomcat-3.3-m2/Webapps/examples"
<Directory "/$USER/local/portal/tomcat-3.3-m2/Webapps/examples">
   Options Indexes FollowSymLinks
</Directory>

#
# The following line mounts all JSP files and the /servlet/ url to
# tomcat
#
JkMount /examples/servlet/* ajp13
JkMount /examples/*.jsp ajp13
```
The only step left is to actually modify the httpd.conf file, which is the main configuration file for Apache. The httpd.conf file needs a directive to include the above mod-jk.conf file to transfer the directories in that to Tomcat. Adding the following line to the bottom of httpd.conf can do this:

Include TOMCAT_HOME /conf/mod_jk.conf

After Apache is restarted the changes should be evident.

3.3.3.2 Tomcat 4.0

The integration of Tomcat 4.0 with Apache is slightly more complicated because it requires a module to be loaded dynamically in Apache at run time. The required module is the Webapp module. The important step here is to make sure that the module exists in the libexec directory of the Apache installation. If it's not present at the time of integration the file should be downloaded and built, making sure that the shared object file – mod_webapp.so – is copied into the libexec directory. The next step is to load the module into the httpd.conf file. This can be done with the following two lines inserted in the code when all the other modules are being loaded.

LoadModule Webapp_module libexec/mod_Webapp.so
AddModule mod_Webapp.c

The module enables Apache to create a Web Connection, which will allow certain directories to be handled by Tomcat. Here is the code to setup the Web Connection:

1. <IfModule mod_Webapp.c>
2.     WebAppConnection warpConnection warp localhost:8008
3.     WebAppDeploy examples warpConnection /examples
4. </IfModule>

The warp Connection allows Apache to listen to Tomcat through port 8008 and have it serve the directories specified. To specify which Web applications are to be handled by Tomcat a WebAppDeploy statement needs to be added for each one. The above line adds the examples directory; similar statements can be added for each directory. After Apache is restarted the changes should be evident.
3.3.4 Tomcat Security

Security in Tomcat is handled by so-called roles. Each application can specify each own number of users and the roles that each user has. A role is Tomcat is used to specify who can gain access to different part of the application. Some roles may be limited to simply viewing some information, while others may actually have access to servlets that modify something. Roles and users for each Web applications are specified in one file. Below is an example of what the file should look like:

Begin Figure 7

1. < users>
2. <user name="user1" password="user1" roles="role1, role3" /> 
3. <user name="user2" password="user2" roles="role2" /> 
4. <user name="user3" password="user3" roles="role2,role3" /> 
5. </ users>

Figure 7. Sample users file

There can be as many users as needed in each file and as many roles specified as needed also. One user can be granted as many roles as needed as well. Once the file is saved it needs to be referenced by the applications to have access to the users and roles. This is accomplished by directing a SimpleRealm tag within the ContextManager in the server.xml file to the users file. Below is the line needed to be inserted:

<SimpleRealm filename="Path to users file " />

Once the setup of the file is finished the next step would be to actually modify each application that wishes to have the security setup as part of it. This involves modifying the web.xml file to setup either basic or form-based authentication. Below is the code necessary to be inserted:

Begin Figure 8

<security-constraint>
<Web-resource-collection>
  <Web-resource-name>Protected Area
</Web-resource-name>
</Web-resource-collection>

<!-- Define the context-relative URL(s) to be protected -->
<url-pattern>/*</url-pattern>

<!-- If you list http methods, only those methods are protected -->
<http-method>DELETE</http-method>
<http-method>GET</http-method>
<http-method>POST</http-method>
<http-method>PUT</http-method>
</Web-resource-collection>

<auth-constraint>
  <!-- Anyone with one of the listed roles may access this area -->
  <role-name>role1</role-name>
  <role-name>role2</role-name>
</auth-constraint>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Example Basic Authentication Area</realm-name>
</login-config>

Figure 8. Basic Authentication in web.xml

The above code specifies which parts of the applications need to be secured, which methods to secure, and which roles are allowed to gain access. The <url-pattern> tag specifies which directories require user authentication. The code above shows that the whole application is set up that way, but it is allowed to have as many tags as necessary to protect each specific directory. Then the <http-method> tags are used to list which methods of access need to be protected. Finally, the <role-name> tags are used to state which roles can gain access to the above directories.

In addition to specifying which directories require certain roles, each servlet written can require it as well. It's a very simple step that only needs one check to be added in the code. UserInRole(role) is a method that is a part of the Servlet API which accomplishes that. By taking the httpRequest parameter and checking it for the needed role provided another step of authentication. Below is a code that uses the above method:
if(request.isUserInRole("role1"))

Chapter 4 - Portals

The primary focus of portals is to integrate the access to data and applications for Web users. It can act as a central hub that connects any resources or data needed by a group or individual. Avi Saha describes a portal as “a single integrated point of comprehensive, ubiquitous, and useful access to information (data), applications, and people [3].” Below is a simple diagram of how a portal connects and interacts with people and the data and applications they need to use.

Begin Figure 9

![Portal Interaction Model](image)

Figure 9. Portal Interaction Model

The most attractive feature of portals is that it allows used personalization and customization. Users can select which applications and data sources they want to appear on their page and have that content brought to them. Some of the more important advantages of portals are that they can increase productivity through secure, integrated access to relevant information. Business processes can be carried out much easier and efficiently because of the constant flow of data. Deployment overhead for application services can also be greatly reduced.

Portals are a relatively new concept, but have already made a large impact on the Web. Portals can be found in a number of different places with different functions. Some of the uses of portals are:

- Corporate employee portals to help company communication
- Personal portals, such as Yahoo or AOL, for personalization
- E-marketplace portals to connect buyers and sellers
4.1 Portal Structure

For a site to be able to set up portals for its users or employees, there needs to be a specific portal server configured with the Web server. The purpose of this portal server is to serve all the information related to the portal being served and leave the rest of the site to be maintained by the actual Web server. Once a portal server is installed and configured with a Web server, then a portal can be accessed by a specific address within the Web server. Creating a separate directory within the file structure is usually the solution. Whenever this directory is referenced by the browser control is transferred from the Web server to the portal server.

A portal is made up of a number of portlets that produce content to be combined in pages. Portlets are small, connecting components that conform to a specific portlet API. These portlets can provide a number of services. These can range anywhere from simply displaying a collection of links, accessing some data from another source, providing a search function, or even connect to applications running on the server. Portlets are very similar in to servlets or applets in a sense that they are also written in Java and have to follow a specific API.

4.2 Web Services

A new way of providing information over the Internet is through a Web service. Web services allow objects to be spread out around the Internet from where clients can access them. Any Web site can be registered as a Web service offering a particular service that a client might want to use. There are a number of standards that exist which are responsible for registering, communicating, and providing a formal description of Web services. Universal Description, Discovery, and Integration (UDDI [11]) is a directory where Web services can be registered and clients looking for particular services can search for them. The Simple Object Access Protocol (SOAP [10]) describes the communication between Web services. Finally, the Web Services Description Language (WSDL [14]) provides a formal way of describing Web service interfaces.

Portals and Web services are very closely related to each other. Portals aim to connect people with data and applications in the easiest and most efficient way possible. Web services actually allow people to have access to that data. Portlets are usually set up that they either receive information from a Web service or actually serve a Web service themselves. Typically, a Web service will provide a client with specific information such as news or scientific data of some kind. That information will be sent over to a portal, usually as an xml document. Then, it's the portal's responsibility to render that document by a specific portlet. Not only is this flow of information very efficient, but it also allows the portal server to present the information independently of how it is transferred or presented on other sites.
4.3 Jetspeed Portal Engine

The Apache Software Foundation provides a great portal server that can help to deploy and maintain portals with little difficulty. Jetspeed is part of the Jakarta project that is an implementation of an information portal built in Java and XML. It acts as a central station where information and content from different sites is gathered and presented. It is meant as a tool for both portal developers and interface designers because it allows for customization of the way information is presented at the same time that it serves as a development tool. It is a great tool for beginners because it's rather simple to learn how to use it and setup a personal portal. However, it does not provide much support in actually writing the portlets, which make up the larger portal, which can provide difficulty in the beginning.

4.3.1 Jetspeed Installation

One of the greater advantages of this tool is that it is very simple to get it up and running in a matter of minutes. Since it is part of the Jakarta project, which also supports Tomcat, the integration of the two is taken care by the software itself and only requires minor configuration steps to start providing content to the Web. After downloading the Jetspeed.tar file and unpacking it into a directory, building the program follows the same pattern as most other Linux programs. The three main steps are:

1. `./configure`
2. `make`
3. `make install`

The end result is a jetspeed.war file that contains the Jetspeed application. Because it is written entirely in Java it can be packaged as a Web application, which can then be handled by the Tomcat Servlet Engine. As any other Web applications that need to be deployed in Tomcat, the .war file needs to be copied into the TOMCAT_HOME/Webapps directory. After performing that step and restarting Tomcat, Jetspeed will be initialized and a directory for it will be created. The portal is now up and running and can be accessed via the Web. If Tomcat is running as a stand-alone server then Jetspeed can be referenced by the jetspeed directory on the port that Tomcat is running on. If Apache and Tomcat have been integrated with the static content being served by the Web server, then the steps outlines in Section 3.3.3 need to be followed. The main screen of the Jetspeed application is provided below to show that the installation has been completed successfully.
4.3.2 Jetspeed Administration

Once the site is up and running the role of the administrator is to maintain it by making sure the necessary portlets are available and adding new ones whenever necessary. The customization and layout of the pages is done by Jetspeed itself, which saves a lot of time for the developer. There are two default users that are already setup – admin and turbine. The admin user will have access to administrative portlets that allow for system information and different security features. Some of these features include setting privileges for users, groups, and roles. The turbine user has little importance as far as maintenance with one exception. The customization that is set up for the turbine user will serve as the default customization for new users on the system until they change it.

4.3.2.1 Adding Portlets

A portal server will have little use if there were no portlets to add to it. Jetspeed provides a number of example portlets in its installation, but a developer needs to
understand the basics of creating and adding portlets to the Website to build a unique portal. There are three main types of portlets that can be created – instance, ref, and abstract. An instance portlet is one that can be instantiated and is usually some sort of an application that the developer creates to run in the portal. An abstract portlet provides the framework that other portlets can extend, but cannot itself be instantiated. Finally, the ref portlet actually extends some other type of portlet that already exists.

The easiest portlets to create are the ref types because they require no Java code to create. These are usually portlets that extend some of the basic portlets that already run on Jetspeed, including a simple Web Page or Content feed portlet. Many times a portal will just be a collection of information and links. These classes that come with Jetspeed allow for developers to quickly jump in and create portlets that reference other Web pages or gather information from feeds from all over the Internet. It is much more appealing to actually have a portlet which shows a snapshot of the Web page that it links to rather than just providing a link to it. It’s also easier to have feeds coming in to list all of the articles available from a certain provider rather than trying to search for them on a page. These are the simplest portlet classes to start with, however, Jetspeed actually provides a number of portlet classes that can be extended. The following classes are included in the org.Apache.Jetspeed.portal.portlets package:

Begin Figure 11

<table>
<thead>
<tr>
<th>AbstractPortlet</th>
<th>ApplicationsPortlet</th>
<th>ClearPortlet</th>
<th>CustomizerVelocityPortlet</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileServePortlet</td>
<td>FileWatchPortlet</td>
<td>JetSpeedContentPortlet</td>
<td>JspPortlet</td>
</tr>
<tr>
<td>LinkPortlet</td>
<td>NewRSSPortlet</td>
<td>PortletInfoPortlet</td>
<td>RSSPortlet</td>
</tr>
<tr>
<td>ServletInvokerPortlet</td>
<td>TurbineScreenPortlet</td>
<td>VelocityPortlet</td>
<td>WebPagePortlet</td>
</tr>
<tr>
<td>WMLFilePortlet</td>
<td>XMLPortlet</td>
<td>XSLPortlet</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11. Abstract Portlet Classes

All of the portlets that are available to be viewed in the portal have to be registered with Jetspeed. This is done through the Portal Structure Markup Language (PSML) and the Jetspeed portlet registry. The registry is made up of a number of .xreg files that specify the portlets that need to be included and any parameters that they might have. By default portlets from the installation are registered in the portlets.xreg file, but I recommend creating separate registry files for each portlet. There are two reasons for this. First, is that each time an upgrade to the Jetspeed server is performed a new portlets.xreg file will be written. Second, it’s much easier to keep track of each portlet that way, and any additions or deletions would eliminate accidentally changing other portlets’ specifications.

To illustrate what needs to be in a registry file I added a simple portlet that showed the contents of the Florida State University Web page in my portal. The code that is needed in the registry is given below:
1. <portlet-entry type="ref" parent="WebPagePortlet"
2.     name="FSUHome">
3.     <url>http://www.fsu.edu</url>
4.     <meta-info>
5.     <title>FSU Portlet</title>
6.     <description>FSU Home Page</description>
7.     </meta-info>
8. </portlet-entry>

The registry entry would very similar if other portlets were used as a ref type. For examples, if the portlet was a content feed from some other place, the only difference would be that the parent attribute would be RSS instead of a WebPagePortlet. The url tag would still be used to point to the place where the feed was coming from.

### 4.3.2.2 Portlet API

To write a portlet yourself it is necessary to follow the Portlet API. This comes with a number of classes and methods that can be used with Jetspeed to create applications. The way of doing that is to extend one of the classes given in Figure 10. These are broad enough to serve any number of applications that need to be developed, ranging from XML manipulation, to file server portlets, to even Web page creation with template languages such as Velocity or JSP. Below is a code to create a simple portlet that just prints out a greeting to the user.

2. import org.Apache.turbine.util.RunData
3. import org.Apache.ecs.ConcreteElement;
4. import org.Apache.ecs.StringElement;
5. public class Greetings extends AbstractPortlet
6. {
7.     public ConcreteElement getContent(RunData aRunData)
8.     {
9.         StringBuffer buf = new StringBuffer("Greetings ");
10.        String name = aRunData.getUser().getFirstName();
11.        buf.append(name).append("!");
12.        return new StringElement(buf.toString());
13.     }
The portlet would print out a message with “Greetings username!”, where username is the person who is logged in.

### 4.3.3 Jetspeed Customization

One of the greatest features of Jetspeed is that the user has a great amount of control on what he wants to see on his personalized page and how it should be laid out. This takes a lot of the Web developer’s worries out of his hands. There is no need for sophisticated interface design because the presentation will be taken care of by Jetspeed. Once the administrator creates all the necessary portlets that he wants to allow users to see and possibly adds some graphics or styles to the list of possible menu bars and panes that the user can choose from his job in that area is over.

Once the user is logged in there are a number of useful customization tools that are provided by Jetspeed. This includes everything from choosing exactly which portlets the user wants to be displayed to choosing what their layout on the page will be. The user can create their own panes to show different portlets on and group those portlets by any way he chooses. Below is a typical screen which shows all the available portlets to the user from which he simply needs to check which ones he chooses to view.

Begin Figure 12

![Figure 12. Jetspeed Portlet Customization.](image-url)
Once the portlets are selected the user is then given the choice of selecting how to lay them out on the screen and which panes to put them in. Some standard and defined colors and schemes are also available to change the look and feel.

Begin Figure 12

![Figure 12. Jetspeed Layout Customization](image)

Figure 12. Jetspeed Layout Customization
Chapter 5 - XML Web Publishing

As XML gains more and more popularity and its importance is increasingly recognized, there is a new movement in developing applications that help in Web publishing. Not only is XML important in how data is being transmitted from place to place, but it also has its advantages in presentation, especially for Web-based applications. Instead of spending hours trying to change old HTML code every time a Web page needs to be updated, there is a new, much more efficient way of doing this. XML documents only contain the data and the Extensible Stylesheet Language (XSL) files contain the code to present that data in a specific format. Therefore, changes are only needed to the XSL document to transform its look. There is still a major problem that exists. It would be very tedious and unnecessary to manually transform every XML document, especially when XML documents are automatically generated through an application. This creates the need for a Web-publishing tool that automatically applies the transformation to the XML document and presents it to the Web.

Figure 13. XML Web Publishing Framework Structure

The previous figure represents how a Web-publishing engine handles the transformations. Instead of applying the transformation manually to each document, the
engine can be set up in a way that every XML document, or a group of XML documents, can be transformed through the same XSLT transformation. The whole idea behind the Web-publishing framework is that it will automatically publish a file after a request from the client. The client will not see the raw data, which will typically be stored as an XML document, but when the request is made it will be published in a specific format [7]. The type of transformation that will be applied to the XML document will specify the format. This can range anywhere from a simple HTML document, to a PDF file, and even to a WML document which is used in wireless devices.

5.1 Cocoon

Cocoon is another tool from the Apache Software Foundation that aids in the development and presentation of dynamic pages and Web applications. Its goal is to allow information, which can be generated in XML, to be transformed into other presentable formats to the client. This includes simple transformations into HTML and more complicated formats such as WML, PDF, or SVG. The most useful part of this whole process is that patterns can be setup so that all files with be transformed automatically by specifying the step that they need to go through rather than doing it manually every time.

5.1.1 Cocoon Installation

The distribution of Cocoon comes as a Web application that needs to be plugged into Tomcat. This has both advantages and disadvantages to it. The advantages are that it is very easy to install and provides a number of working examples that are available immediately. However, the disadvantages are that because it needs to run under Tomcat it as to be configured in a specific way. Unlike Jetspeed, there are a few steps that need to be taken before Cocoon will operate correctly on the server. Once the distribution was downloaded and built, a cocoon.war was created which would simply need to be plugged in as a Web application. By simply copying the file and restarting Tomcat, the Cocoon application would not properly load. The reason has to do with certain class files that are used by both Tomcat and Cocoon. Tomcat contains an older version of the JAXP parser, which is replaced by the xerces parser in the Cocoon distribution. To fix the problem there are three steps that need to be taken prior to copying the cocoon.war file to the Tomcat/webapps directory.

- Delete Tomcat/lib/jaxp.jar
- Rename Tomcat/lib/parser.jar Tomcat/lib/zparser.jar
- Copy cocoon/lib/xerces.jar & cocoon/lib/xmlapis.jar to Tomcat/lib

After installing the cocoon application and restarting Tomcat, test the installation by pointing the browser to localhost:8080/cocoon. The following screen should appear:
The above example will only work if the system has an Xserver running on it. For those systems that do not, there is a work-around that needs to be performed. The Cocoon distribution requires that there be an Xserver to support some of the graphic Java libraries that are in it. The simplest solution is to install an in-memory virtual Xserver that does not actually require any graphics hardware to run. Xvfb is such a tool that often comes with a standard Linux distribution or can be downloaded for free from the Website. I modified the Tomcat startup script to start the virtual xserver first before continuing with the other Web applications. This allows for Cocoon to display its files properly. The following is the modification that was placed in the Tomcat/bin/startup.sh file:

```
xvfb :1 -screen 0 800x600x8 &
export DISPLAY=:1
```
5.1.2 Cocoon Structure

Cocoon handles the processing of all documents through a pipeline process that involves a number of steps specifying how the documents are generated, transformed, and serialized. The pipeline for all documents, as well as the list of all components that make up Cocoon are specified in the sitemap. The sitemap is an xml-like document that acts as the main configuration file for Cocoon. Any changes to an application need to be documented in this sitemap to reflect them and specify proper processing. The sitemap file is located in the main cocoon directory under Tomcat/webapps. To properly understand the sitemap I briefly need to explain the tree main parts that define any pipeline – the generator, the transformer, and the serializer.

The generator acts as the initialization point of the pipeline process. It is used to create an XML structure from some input source, such as a file or directory. It applies certain transformation to an XML document, creates a SAX structure out of it, and sends the compiled XML structure to the next step in the pipeline.

The transformer is the next step in the pipeline and is responsible to transforming the incoming XML structure into another one. Usually this is accomplished by associating a certain transformer with the incoming structure and changing it a new compiled structure. This step adds the parts necessary for the pipeline to transform the SAX structure into some kind of output format.

The final part of the pipeline is the serializer. It actually takes the XML structure that it receives and renders it in the appropriate format. This is an important step because it performs the actual step where the data is changed into some viewable form that the user can see. The usual transformers output the data in HTML, WML, or other formats. Cocoon comes with a number of already compiled generators, transformers, and serializers, but it is also possible to create new ones and add them to Cocoon. They are all written in Java, which is why it is possible to write new ones and have Tomcat actually serve them. The only thing that needs to be done is to let Cocoon know where to find these new structures. This is done by adding appropriate lines in the sitemap to include the newly created files. The following is an addition to the sitemap file to specify a new generator:

```xml
<map:generator name="my"
src="org.Apache.cocoon.generation.MyGenerator"/>
```

5.1.3 Adding a Pipeline

The hardest thing for new developers starting out is to actually start adding their own files to be transformed. The sitemap is a rather complicated file and has a vast number of options that can be used to configure it. However, the basic principle lies in the addition of new pipelines to serve the files that are located in the cocoon directory structure. The first example that I will go through is to how add a single file transformation from XML to HTML. Assuming that the XML and XSL files are ready for deployment I need to tell the
sitemap where to find them and how to transform them into an HTML file. The files that I want to add are called *sample.xml* and *sample.xsl*.

```xml
1. <map:match pattern="sample.html">
2. <map:generate src="test/sample.xml"/>
3. <map:transform src="test/sample.xsl"/>
4. <map:serialize type="html"/>
5. </map:match>
```

I have placed my two files in the *test* directory and want to output an HTML file that is rendered from the XML and XSL files. The first line creates a new pattern that specifies how the file to be referenced is identified. It's needed to tell Cocoon what to do whenever the user tries to open the sample.html file or is linked to from another page. Line 2 selects which file we want to generate. Line 3 shows which file is going to be used to transform the original generated file. Finally, Line 4 outputs what type of a serializer will be used, in this case being a simple HTML file. Even though I do not have a sample.html file anywhere in my directory, it will be automatically generated by Cocoon.

To show a little more complicated example and to get closer to what the real benefits of Cocoon are, I will present a pattern in the pipeline, which allow for multiple transformations of files. One of the major needs for Cocoon is that it will allow us to transform similar documents without us actually specifying all the files in the sitemap. This is particularly useful when XML files are generated and need to be transformed to be output to the client. It would be impossible and unnecessary to create a match pattern for every such document; of which there could be thousands. This is where the pattern I create can become very useful.

```xml
1. <map:match pattern="*.html">
2. <map:generate src="test/{1}.xml"/>
3. <map:transform src="test/{1}sample.xsl"/>
4. <map:serialize type="html"/>
5. </map:match>
```

The above code will be able to transform any number of HTML documents that are requested on the site. The asterisk acts as a wildcard and will take a name for the file from the XML file that is being generated in Line 2. Every XML file that is generated will be applied an XSL transformation stated in the *sample.xsl* file and rendered as an HTML document. This is obviously a very timesaving feature that can create many dynamic pages very quickly.
Chapter 6 – Websphere Application Server Environment

The second environment that I created consisted of tools from the IBM Websphere project. The tools in this project are meant for commercial Web development so it was a great comparison to the Apache tools, which are distributed as a free alternative. Being a commercial piece of software the amount of knowledge needed to install it and understand how to use is great. The extensive documentation and easy-to-use environments make that process a little easier, however. In this chapter I will talk about a few topics relevant to this application environment. The first issue will deal with the basic concept and terminology that are used. The second main issue will deal with the correct installation and configuration of the application server. The final topic will discuss how to use the application server to develop and deploy applications to the Web.

6.1 Application Server Concepts

The IBM environment is quite different from the one described in the Apache environment. The tools responsible for serving Java applications and XML are already built into the application server. The main difference is that Web applications have to be structured and deployed in a specific format consistent with the Websphere ideology. Fig. 15 shows the basic structure of the Websphere Application Server as described by IBM.

Applications that are deployed in Websphere are known as enterprise applications. Enterprise applications are specially packages files that contain specific elements. Those smaller elements are combined to form a large application. An enterprise application can contain Enterprise Java Beans modules, Web modules, application client modules, and any standalone Java classes packaged in .jar files. These concepts will be discussed in the next few sections. Once an enterprise application is packaged it is saved in an .ear file. The Application server then known how to handle those enterprise application files and properly deploy them to the Web server.
Begin Figure 15.

Figure 15, Websphere Application Server Model.

6.1.1 Enterprise Java Beans Modules

EJB modules are basically server-side components for the J2EE platform, which allow for a much simple and faster development of middleware applications. They also enable applications to be much more portable and secure [4]. The developer no longer needs to worry about the transaction management aspects of the applications because the EJB takes care of it internally through its objects and method calls.

EJB has rapidly grown in popularity recently because it allows the developer to concentrate more on the business logic and not worry so much about data processing management. Some of the major advantages of EJB include:

- Freeing the developer from transaction and thread management
• Platform-independent solutions
• Compatibility with other Java APIs and non-Java applications

There are two types of enterprise beans – entity and session. The main difference is that entity beans require a database connection, while session beans do not. It's also possible for session beans to access entity beans if database connection is necessary.

Enterprise beans do not communicate with the application server directly. The middle layer that connects the two is the enterprise bean container. The container provides the interface between the beans and the server. The container is what manages the transactions and data storage for the beans it contains.

6.1.2 Web and Application Client Modules

Web modules contain the Web applications. Web applications contain a number of different files. These can be regular static HTML files, servlets, or JSP files. Just like the Apache environment Web applications also need a deployment descriptor to properly configure them with the application server. Web modules are packaged in .war files and the deployment descriptor is named web.xml. The structure is very similar to the way Web applications are deployed in Tomcat. This shows the underline premise that Java is the basis for all these technologies.

Just like enterprise beans cannot communicate with the application server directly so do Web applications. A Web container is necessary to serve the requests for servlets, JSP files, and all Java server-side applications. A Web container is another name for a Java-enabled server, which was talked about in Chapter 3. Tomcat can be viewed as a type of a Web container. Some of the main responsibilities of a Web container include creating servlet instances, managing request and response objects, and overseeing basic servlet functionality.

The final piece of an enterprise application is an application client module. It's a standalone Java program that does not run in a browser. This is less common and less important to this project, but should be mentioned because it is a part of Websphere. It works similarly to the Web modules in that it combines Java files and a deployment descriptor. The packaged files are stored in a .jar file. An application client container is configured with the application server and allocates all the necessary resources for the client to run the application.
6.2 Websphere Installation

The installation of Websphere can be somewhat tricky because there are a lot of steps to properly configure it. I will go into the installation of Websphere on Linux Red Hat since that is the environment that I created the installation on. There are a number of prerequisites that need to be considered before installation. It's important that there is enough disk space to fit the application server because it is quite large. Second, the machine needs to have a lot of memory, at least 512MB, for the application server to run properly. Finally, it's necessary to have a database system installed on the machine. I will describe the basic installation of IBM DB2 needed to ensure proper operation of the application server. Oracle, MySQL, and other database can be used also, however, I recommend using DR2 because it's an IBM product also. This allows for quicker configuration and is documented well in case problems arise.

6.2.1 DB2 Installation

The presence of a database system is a must before installing the Websphere Application Server. The following section shows a guide that installs DB2 and sets up the necessary configurations.

1. Type ./db2_install at the distribution top directory. The following message will be shown on the screen:

   Specify one or more of the following keywords, separated by spaces, to install DB2 products.
   DB2.PERS - DB2 Personal Edition
   DB2.SDK - DB2 Application Development Client
   DB2.CAE - DB2 Administration Client
   Enter "help" to redisplay the product names, enter "quit" to exit.
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

2. Select one or more of the options and press 'Enter' to install the rpm's. The software will be installed in the /usr/IBMdb2/V7.1 directory.

3. Create three groups, one for instance users, one for fenced users, and one for administrator server users.

4. Create three users – an instance owner user, a fenced user, and administrator server user. Place them in appropriate groups created in step 3.

5. Create a db2 instance by typing the following:
   /usr/IBMdb2/V7.1/instance/db2crt -u fenced_user instance_name

6. Create a db2 Administration Server by typing the following:
   /usr/IBMdb2/V7.1/instance/dasicrt AdminServerName
The installation of DB2 should now be complete. To test the installation simply log in as the instance owner and start the database server. The database server can be started by typing .\db2 at the command prompt. This is all the explanation of DB2 that I will go into because that's all that is needed to install the Websphere application server. DB2 is a powerful and complex database server that offers many options. To fully understand those I recommend going through the documentation that comes with it.

6.2.2 Application Server Installation

One of the great features of the Websphere application server is that one installation installs all the necessary components on the machine. The Java servlet and JSP server is automatically installed, as well as the XSL Transformer to handle XML document transformations. The IBM HTTP server also comes in this distribution, but does not have to be used if the Apache Web server is already installed. They both work exactly the same so the option is up to the user.

To start the installation log in as root and navigate to the top directory of the distribution. Simply run the installation script by typing .\install.sh and wait for the installation screen to appear. Unlike the Apache tools, this installation has a graphical interface, which makes it easier to follow. There is also a progress indicator to show how far into the installation the process is. Click Next on the welcome screen to start the installation. Select Typical Installation to install the necessary components.

The next screen configures the DB2 server with the application server to be installed. The fields to be filled out are pretty self-explanatory so there is no need to go in depth. The only point I want to make is that the DB2 needs to have information of the instance owner to be properly configured. Once that information is entered and the directory for the software is selected the installation should complete rather quick.

At this point, the installation is finished. There are three main components that have been installed – the HTTP server, the administrative server, and the application assembly tool. The later two will be discussed in more detail in the next few sections. It's important to test that the HTTP server is operating properly. Start the Web server and test it in the browser window to verify proper installation.

The final step before the application server can be started is the creation of a database instance to interact with the server. To accomplish this, log in as the instance owner user. Navigate to the bin directory of the application server installation. This step involves running a small script to actually create the database that interacts with the application server. Type ./createdb2 and type in the information that the script asks for. This will be the username and password of the instance owner and the name that the database should be given. At the completion, log in as root user to test the application server.

As I mentioned earlier, the application server has both an administrative server and client to perform its functions. The client is a graphical interface to the application server, which allows for the actual deployment of applications. In order to start the client, the administrative server has to be running on the system. The following command at the bin directory starts up the server - ./adminserver.sh It will take a bit of time for the
server to start up properly. If the installation has been successful and the server starts up properly the message 'Server open for e-business' will appear in the console. At this point the application client can be started by typing ./adminclient.sh

Once the client starts the final step of testing the installation is to start the local server and tests if the sample applications that come with the distribution can be deployed. Expand the tree to WebSphere Administrative Domain->Nodes->localhost->Application Servers->Default Server. At this point, start the default server by right clicking on it and selecting start from the popup menu. An information window will appear when the server has started. Open the browser and enter the address for a snoop servlet as follows:

http://localhost/servlet/snoop

If the information about the servlet appears in the browser that means that everything is working properly.

6.2.3 Deploying an Application

As I described in section 6.1 the applications deployed in Websphere have to be assembled in a specific manner. All applications have to be enterprise applications and contain enough information to specify how they should be deployed. The enterprise applications are packaged in .ear files and contain any components that are necessary to run them. The Websphere application server has two separate tools for this process. The Application Assembly Tool packages the needed components into modules and then into enterprise applications. Then the Administrative Server deploys those applications to the Web server.

6.2.3.1 Application Assembly Tool

As I mentioned earlier all code has to be deployed as enterprise applications within the Websphere environment. To properly package the files and get them ready for deployment it's necessary to use the Assembly Tool that comes with the installation. This is a great tool that has an easy-to-use graphical interface. It provides a number of wizards that automate the task of creating the modules and adding those modules into an enterprise application.

I will go into the basic steps to create a module and then combining several modules into a larger application. This will provide the basic steps necessary to deploy an application from scratch after the code has been written. It will also allow installing some sample tutorials that can be downloaded from the Websphere Information Center.

Start the Assembly tool by typing the following command in the bin directory of the Websphere installation:

./assembly
The picture below shows a typical screen shot of a module loaded by the tool. The properties and parameters of the module are displayed and are easily modified through textboxes and tabs. The module shown below is a sample EJB module I installed during the testing of the application server.

Begin Figure 16

![Application Assembly Tool](image)

Figure 16. Application Assembly Tool.

The creation of applications and modules using this tool is very simple once all the java code and static files have been produced. The wizards guide you through the process making sure that every important piece of the application is covered. I will describe the process of adding a Web module. Creating an EJB module or an application client module is almost identical. It simply uses a different wizard to setup slightly different properties and save the module with it's own appropriate extension.
To create a module using a wizard, click on **File->Wizards**. There is a wizard for creating all the modules and one for creating an application. For the Web module click **Create Web Module Wizard**. The wizard will go through about twenty different screens to gather all the necessary information. Most of those are optional because they set specific parameters, icons, and mappings if the module requires it.

The required information that needs to be entered, deals with the files and how they will be served. Make sure to specify a unique name for each Web module. The next step involves adding all the class files that will be used by the Web module. These can be files already packaged in .jar files or not.

After a few optional screens the most important part of the Web module is created. This is the part where the actual Web components such as servlets or JSP files are added. Make sure to create a unique name for each Web component, add the files for it, and specify whether it's a servlet or JSP. This step is necessary because the Web plug-ins that the application server uses to serve each type is different.

The wizard continues going through the rest of the setup. Some of the options include adding resource references, context parameters, tag libraries, and EJB references that can be used by the Web module. It also asks for a URL to specify the servlet mapping. This URL will be used by the browser to access the Web module.

Once the wizard finishes the Web modules and it's components will be displayed in the assembly tool for manual editing. Additional properties or files can be added at any time through the interface. Finally, save the Web module as a .war file and it will be ready for packaging into an enterprise application.

An enterprise application is packaged very similarly to individual modules. The easiest way is to run a wizard to accomplish this task. There are seven primary steps during the wizard that need to be performed.

1. Specify an application name and file
2. Add additional files not included in component modules.
3. Add EJB Modules
4. Add Web Modules
5. Add Application Client Modules
6. Define Security Roles
7. Generate Deployment Code

I have already described how to create the individual files, so the next thing in this list is security. Security within the Websphere application server is automated as well. Unlike the Apache environment where I had to manually create files to authenticate users, the code here will be created automatically. The really great feature is that security can be added to individual modules as well as the entire application.

The wizard allows the administrator to create security roles just like I did with the Tomcat Server. Roles will be used to gain access to specific parts of the application. Once the roles are created, the users can be specified also. Binding the users and the roles can be
performed through the graphical interface and will automatically assign roles for individuals. This is a much easier process than the Apache environment. The final step is assigning the particular roles to the methods within the code. Those are bound by the tool as well and leave the tedious act of adding the code yourself out of the picture.

The final step involves the generation of deployment code for the application. This code is used by the Administrative Server when the application is installed. The deployment code generation is made up of a few steps. It compiles all container implementation classes. It validates all the code under J2EE regulations. Finally, it creates persistence code for entity beans.

Once all the steps have been performed by the wizard, the file is saved with an .ear extension. The Application Assembly Tool can now be closed because the next part of deployment is performed with the Administrative Server.

6.2.3.2 Administrative Server

The Administrative server is a tool that allows for the configuration of applications and their deployment to the server. It’s responsible for installing applications and modules, updating their properties, and administering the servers running on the machine. As I mentioned earlier it’s comprised of two parts – the server and the client. The client is just a graphical user interface for the server. This allows for much faster configuration and deployment of web applications. It’s important to first start the server and then the client. The client will not start if the server is not running on the system.

To start the server:

./adminserver.sh

To start the client:

./adminclient.sh

The Administrative Server has many different functions and capabilities. It can set up virtual hosts and many nodes that can be configured to the application server. Some of its functions also include setting up resources, security references, and database connectivity support. The client allows this to be accomplished at a much easier rate because of its graphical interface. Figure 17 shows a screenshot of the Administration Client.
Begin Figure 17.

![Websphere Advanced Administrative Console](image)

Figure 17. Administration Client.

I will briefly describe the process of installing an application with the client. It’s rather simple and only takes a few minutes once the enterprise application has been packages with the Assembly Tool. Once again, a handy wizard is available for accomplishing this step. The main step in this process is actually specifying a name for the installation to be performed and inserting an .ear file into it. Some optional properties can be modified as well during the installation, but they are not obligatory.

During the installation it's important to specify which node the application will run on. Since the server can handle numerous nodes it's important to specify the right one. This will typically be the default server, which is probably the machine that the application server is installed on.

Once the application is installed and appears among the Enterprise Applications in the tree-view of the Websphere Server, two more steps have to be performed. The first step deals with the regeneration of Web plug-ins. As I mentioned earlier, plug-ins are responsible for the configuration between the Web server and the application server. Whenever changes
are made, the plug-ins need to be regenerated. This is done by simply right-clicking on the default server and choosing Regen Webserver Plugin as the option. Finally, the server needs to be restarted to make the application available to the Web. This can also be done by right-clicking on the default server and choosing the Start/Stop options.

This chapter can serve as a basic guide to getting started using the IBM Websphere Application Server. It provides enough information on how to package the files and deploy them to the Web. It is by no means a comprehensive documentation of the whole application server, but it should be enough to get a brand new user starting to deploy applications on it.
Chapter 7 – Conclusions

The goal of this paper is to present a guide for developers and administrators to using the tools evaluated during this project. During this project I came up with evaluations for the two environments that I created. As a conclusion to this paper I would like to share these evaluations and provide some recommendations. I will talk about the advantages and disadvantages of both sets of tools that I encountered.

The Apache environment is great for beginners because it does not require in-depth knowledge. A new user can quickly become acquainted with the server and the other tools to begin developing applications on it. A really attractive feature is that all of the tools from the Apache Software Foundation are free. The real tricky part is that there is not a great deal of documentation available. I had to find most of the solutions to my problems through user lists and posts. Not having a graphical interface to the configurations really slows down the process of administering applications.

The IBM environment is a great commercial product. It’s meant for experienced professionals who are ready to take their Web applications to the next level. Since it is a commercial product it is quite expensive, but it makes up for it with its functionality. The graphical interfaces and the wizards help to automate standard tasks. A beginner could also sit down and begin using this product by going through extensive tutorials that are presented.

From my project I have realized that these tools are really meant for quite separate things. The Apache environment is meant for Web developers and administrators that are interested in having complete control over the site. Sometimes this means sacrificing some time to configure everything properly. The IBM tools on the other hand are developed primarily for e-commerce. They provide easy-to-manage structures and faster deployment. EJB module support also provides transaction and security support independent of the applications.

Fig. 18 represents the main issues that I find important when dealing with application servers and tools. I listed the benefits and disadvantages of both environments in a table to easily see some of the main differences.
Begin Figure 18.

<table>
<thead>
<tr>
<th></th>
<th>Apache Server</th>
<th>IBM Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Installation</strong></td>
<td>Easy</td>
<td>Quite Complicated</td>
</tr>
<tr>
<td><strong>Configuration With</strong></td>
<td>Quite a bit of work,</td>
<td>Automated, configured</td>
</tr>
<tr>
<td><strong>Development Tools</strong></td>
<td>including manipulation of several files</td>
<td>during installation</td>
</tr>
<tr>
<td><strong>Price</strong></td>
<td>Free</td>
<td>Very Expensive, depends on edition also</td>
</tr>
<tr>
<td><strong>Graphical Interfaces</strong></td>
<td>None</td>
<td>Very easy to navigate</td>
</tr>
<tr>
<td><strong>Servlet Support</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>JSP Support</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>EJB Support</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>XML Publishing</strong></td>
<td>With Cocoon</td>
<td>Integrated</td>
</tr>
<tr>
<td><strong>Deployment of Applications</strong></td>
<td>Simple once a .war file has been created</td>
<td>A little complex, but automated with installation wizard</td>
</tr>
</tbody>
</table>
References

1. "A Brief Introduction to Servlets and JSP", Harshal Deo, Web Developer's Virtual Library, April 5 2000

2. “Apache Software Foundation”, web site,
   http://www.apache.org

   IBM Software Strategy, IBM November 1999

4. "Enterprise JavaBeans Technology", web site,
   http://java.sun.com/products/ejb/

5. “IBM Websphere Info Center”, web site,

6. "Introduction to Java Server Pages", J.J. Kuslich,

7. “Jakarta Tomcat”, web site,
   http://jakarta.apache.org/tomcat/

8. “Java and XML”, Brett McLaughlin, O'Reilly,
   1st Edition June 2000

9. "Java Servers", web site,
   http://www.javaskyline.com/servlet.html

10. "Java Servlet Technology", web site,
    http://java.sun.com/products/servlet/
   http://www.w3.org/TR/SOAP

    http://www.uddi.org

13. "Web Application Servers", web site,
    http://www.appserver-zone.com/articles/wa1198/wa1198.asp

14. “Web Application Servers”, John Garris,
    PC Magazine May 14, 1999
    http://www.zdnet.com/products/stories/reviews/0,4161,2129538,00.html

15. “Web Services Description Language (WSDL) 1.1”, Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, 2000
    http://www.w3.org/TR/wsd1