Chapter No. 2
"Configuration and Deployment"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.2 "Configuration and Deployment"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Tanuj Khare has been a professional in IT for over six years. He is involved in process improvements using the ITIL framework and techniques such as Lean Six Sigma. He is MCSA and ITIL certified, and has expertise in handling critical production server issues. He also has a track record of dealing with many complex problems. His quick resolution to issues faced in the production environment has helped his team and clients in a big way.

Tanuj has Subject Matter Expertise (SME) in Tomcat, WebLogic, and JBoss server administration. His experience includes working with large-enterprise web hosting environments for J2EE containers with small teams, and his quick turnaround time gave on-time delivery. Apart from this, his technical expertise in Root Cause Analysis, Problem Management, Migration of enterprise applications, and upgrade of web application servers are commendable. Up to now, he has migrated more than 100 enterprise applications and upgraded J2EE web applications. He has also managed environments with over 1000 middleware instances.

Apart from work life, Tanuj enjoys playing Table Tennis and exploring new technologies. He is a good dancer. This is his first book.

For More Information:
Apache Tomcat 7 Essentials

This book will help you resolve these issues and boost your confidence in handling Apache Tomcat 7 administration using the tips, tricks, and best practices used by various industry experts to maintain their middleware infrastructure. The best thing that the author did while designing the content is a practical solution, with a detailed description of why we are doing this solution.

Apache Tomcat (or Jakarta Tomcat, or simply Tomcat) is an open source servlet container developed by The Apache Software Foundation. The latest major stable release, Apache Tomcat version 7, implements the Servlet 3 and JavaServer Pages 2 specifications from the Java Community Process. It includes many additional features that make it a useful platform for developing and deploying web applications and web services.

Apache Tomcat 7 Essentials follows a practical approach to explain installing, configuring, and maintaining Tomcat. It helps you to understand the middleware architecture to host multiple websites and also provides the confidence to implement middleware support. It imparts to you the capacity to resolve migration issues and also provides regular maintenance solutions. This is the first, and only, book to cover upgrading to Tomcat 7 from the previous versions.

The journey of the reader starts at the beginner's level and ends at the expert level. The content is designed in such a way that it balances the theory and practical approach for understanding concepts related to handling middleware and web issues.

In this book, you will go through a three-phase life cycle. The first cycle consists of the installation, configuration of Tomcat 7 on different OSes, other configurations related to the JDBC, port, deployment, and so on.

The second phase deals with the building of an enterprise application setup and high availability architectures (clustering and load balancing). The third and critical phase will teach you to handle critical issues, performance tuning, and the best practices for various environments such as Dev/QA/Stage/Production.

This book gives you a wider vision of using Tomcat 7 in web technologies and the skill to optimize its performance using Apache Tomcat 7.

For More Information:
What This Book Covers

Chapter 1, Installation of Tomcat, covers the Apache Tomcat history and the new features introduced in Tomcat 7. The step-by-step installation of Tomcat 7 on Windows and Linux operating systems and common problems that may arise during the installation and their possible solutions are also discussed.

Chapter 2, Configuration and Deployment, covers the configuration of Tomcat including the DataSource configuration for different databases such as Oracle, MySQL, and PostgreSQL, and Context Path creation using an application. Various ways to perform deployment including deployment using Tomcat Manager for a sample application and troubleshooting common issues are also discussed.

Chapter 3, Performance Tuning, explains the different ways of performance improvement and techniques in Apache Tomcat 7. Step-by-step configurations for Connectors, JVM performance tuning, and OS parameter optimization are also covered.

Chapter 4, Integration of Tomcat with the Apache Web Server, explains the integration of Apache/IIS with Tomcat 7, integration of various components such as mod_jk, mod_proxy, and real-time issues which may arise during integration, along with their solutions.

Chapter 5, Securing Tomcat 7, explains the various policies of Tomcat 7 and its functionalities such as the Catalina policy and System level policy. Measures to enable security and their benefits such as SSL, best practices used in real-time industries to secure Tomcat 7 in the production environment, by doing the configuration change and SSL implementation are also covered.

Chapter 6, Logging in Tomcat 7, explains the different methods of enabling logs in Tomcat 7—log4j and JULI. Also, the best practices used for log analysis, tips, and tricks are discussed.

Chapter 7, Troubleshooting in Tomcat, covers different issues faced by the application/web administrators in a real-time environment, how to avoid these issues in the production environment using different techniques with errors and their solutions, thread dump analysis and tools used for thread dump analysis, memory issues, steps for troubleshooting real-time problems, and web server benchmarking.

Chapter 8, Monitoring and Management of Tomcat 7, explains various processes of monitoring in Tomcat 7, components using Tomcat Manager and JConsole, such as different ways of monitoring, how monitoring is done in Tomcat 7, JConsole, and how it is used.

For More Information:
Chapter 9, *Clustering in Tomcat 7*, explains clustering of Tomcat 7 and its implementation technique. Topics included are clustering architecture, horizontal and vertical clusters and their benefits, implementation of horizontal and vertical clustering on Tomcat 7, and verification of clusters.

Chapter 10, *Tomcat Upgrade*, explains various strategies used in the upgrade from Tomcat 6 to Tomcat 7 and the various steps followed during the upgrade process such as the life cycle of the upgrade, upgrade configuration of Tomcat 7, DataSource configuration, and discussions on various ITIL processes used during upgrade.

Chapter 11, *Advanced Configuration for Apache Tomcat 7*, explains the advanced configuration of Tomcat 7 and its optimization parameters. Key points covered in the environment are virtual hosting, features of Development/QA/Stage/Production, Tomcat as a service, and running Tomcat as a non privileged user.

For More Information:
In the previous chapter, you have installed Apache Tomcat 7 on DOS (Windows) and non-DOS (Linux/Unix) operating systems. Now, it's time to discuss the different configuration and deployment strategy tools used by different IT industries.

In this chapter, we will discuss the following topics:

- Configuration of Tomcat
- Configuration of the virtual directory
- Deployment of an application on Tomcat 7

Configuration files and their usage

Apache Tomcat 7 comes with a default setup, which can be directly used for a QA environment. We can customize Tomcat based on the environment specification; components such as Services, Servers, Engine, Connectors, Realm, and Valve can be configured. The Tomcat configuration files are available in the conf folder. Let's discuss the configuration properties and their usage.

Tomcat 7, by default, comes with seven configuration files (usually in XML format), and these files are very useful in order to customize Tomcat, based on the environment needs. We shall install in the production or development environment.

For More Information:
Configuration and Deployment

The following screenshot shows the directory structure of the configuration directory for Tomcat 7:

```
root@localhost conff]# ls -l
total 136
-rwxr-xr-x 3 root root 4096 May 16 21:03 catalina.properties
-rwxr-xr-x 1 root root 11888 Apr 1 18:15 catalina.policy
-rwxr-xr-x 1 root root 5069 Apr 1 18:15 catalina.properties
-rwxr-xr-x 1 root root 1428 Apr 1 18:15 context.xml
-rwxr-xr-x 1 root root 3213 Apr 1 18:15 logging.properties
-rwxr-xr-x 1 root root 6645 Apr 1 18:15 server.xml
-rwxr-xr-x 1 root root 1566 Apr 1 18:15 tomcat-users.xml
-rwxr-xr-x 1 root root 53273 Apr 1 18:15 web.xml
```

It's very important from an administrator's point of view, to know about the configuration files and their usage in the Tomcat environment. Let's discuss the configuration properties one-by-one, as follows:

- `catalina.policy`: This file describes the security policy permissions for Tomcat 7. It enforces the security policy permissions by JVM on the web application.

  > When `catalina` is executed with the `-security` option, the security policy mentioned in the `catalina` file is used and the web application security policy also gets executed.

- `catalina.properties`: This file contains the shared definition of the server, shared loader, and JARs, which need to be scanned at the time of the server startup.

- `server.xml`: This is one of the important configuration files of Tomcat. It holds critical information, such as the IP address, port, virtual host, context path, and so on.

- `tomcat-users.xml`: This file is used for authentication, authorization, and role-based definitions. It is used to implement a database of users/passwords/roles for authentication and container-managed security. To add/remove users or assign/unassign roles to existing users, edit this file.

- `logging.properties`: As the name suggests, it defines the logging properties of the Tomcat instances (such as startup logs).

- `web.xml`: This defines the default values for all web applications loaded into this instance of Tomcat, at the time of startup of the Tomcat instance. If a web application has its own deployment descriptor, its content will always override the configuration settings specified in this default descriptor.

• context.xml: The contents of this file will load with every application. Configuration of parameters such as session persistence, Comet connection tracking, and so on, are done here.

Any changes made in the server.xml file will be in effect after restarting the Tomcat instance.

Application level resources are not defined in the web.xml of the configuration folder. It would be better to define these in the application web.xml.

Configuration of Tomcat 7

Until now, we have discussed the various configuration files of Tomcat 7. Now, the interesting part starts while implementing these in practical, or on live systems.

Before we learn the details of the Tomcat server configuration, let’s quickly understand how the web application works from the following steps:

1. Whenever you hit the URL (for example, www.abc.com), the browser will contact the DNS server.
2. The DNS server will contact the ISP for the required information.
3. Once the web server accepts the request from the client browser, it will redirect it to the database server.
4. In turn, the database server will retrieve the query and respond it back to the web server.
5. The web server then forwards the same response to the client browser, and finally, the client browser will display the content to the user.

For More Information:
That's how the web browser gets the content generated by the web server. The following figure explains the web application functionality and different components, which play a vital role for the application to work:

**DataSource configuration**

For any web application, the database plays a very vital role as it's the backbone for an enterprise application. For an application to perform well, the correct datasource configuration is necessary at the application layer.

Before moving further, let's quickly discuss how the web application gets the response from the database server.

1. Whenever you hit the URL (for example, www.abc.com), the request goes to the web server.

For More Information:
2. Once the web server accepts the request from the client browser, it will analyze the request based on the query. If it requires the database (DB) response, then it redirects the request to the database server.

3. Based on the query, the database server will retrieve the content and respond to the web server. The web server then forwards the response from the database server to the client browser.

This process flow is also explained in the following figure:

![Diagram showing the process flow of database request in a web application.]

After all the previous discussions, we now understand how the database request flows in the web application. Now, it's time to do the real-time configuration of the datasource of Tomcat 7. Some of the terminologies used in the database connectivity are explained in the following content.

For More Information:
**JDBC**

Java Database Connectivity (JDBC) is a Java-based data access technology that provides an API through which the client accesses the server database. It is oriented towards a relational database and provides a way to query and update the database.

**JNDI**

Java Naming and Directory Interface (JNDI) services are an API for the Java platform, which provides naming and directory functionalities to applications written using the Java programming language.

**DataSource**

It is a Java object used to access relational databases through the JDBC API. It works well when integrated with the JNDI and after a datasource object is registered with a JNDI naming service. Objects can be accessed by the application itself and connect to the database.

The following are the parameters required for any database server to connect Tomcat 7 with the database and are also the prerequisites for datasource configuration:

- IP address
- Port number
- JNDI name
- Database user ID/password

Database servers in production

The applications which are hosted on the Internet, their web servers are always configured in the Demilitarized Zone (DMZ). For more information on the DMZ zone, please refer to [http://en.wikipedia.org/wiki/DMZ_(computing)](http://en.wikipedia.org/wiki/DMZ_(computing)).

Database servers are placed in an internal network. In this situation, the firewall port needs to be open between the web servers and the database server for communication.

**Database Connection Pool (DBCP)** configuration is located in the TOMCAT_HOME or CATALINA_HOME/lib/tomcat-dbcp.jar. This specific JAR is responsible for connection pooling. The following screenshot shows the location of tomcat-dbcp.jar. The following are the built-in properties of the Tomcat 7 server for accomplishing a connection with the database:

For More Information:

- Database Connection pool
- Common DBCP properties

```bash
root@localhost lib]# cd /opt/apache-tomcat-7.0.12/lib/
root@localhost lib]# ls -l tomcat-dbcp.jar
rw-r--r-- 1 root root 234639 Apr 1 18:15 tomcat-dbcp.jar
root@localhost lib]#
```

- Configuration of the database server details in `server.xml`
- The database specific JAR or JDBC driver needs to be placed in the `lib` directory
- The JNDI should be defined in the application `web.xml` file
- Application code should have proper JNDI configuration defined

There are many databases available in the market and every DB has its own advantage and disadvantage. We will discuss the most common databases used in the enterprise application and how to configure a datasource for these databases.

DataSource configuration consists of four major steps, irrespective of the database used.

**DataSource for Oracle**

The Oracle database holds a major share in the IT market because of its features. Following are the steps which you need to perform on the datasource configuration of Tomcat.

1. By default, the definition of datasource values are defined in the global section of `server.xml`. The following screenshot shows the datasource details in `server.xml`:

```xml
<!-- Global JNDI resources Documentation at /docs/jndi-resources-howto.html-->
<GlobalNamingResources>
<!-- Editable user database that can also be used by UserDatabaseRealm to authenticate users-->
<Resource name="jdbc/tomcat7" auth="Container"
type="javax.sql.DataSource"
driverClassName="oracle.jdbc.OracleDriver"
url="jdbc:oracle:thin:@127.0.0.1:1521:test"
description="test database for tomcat 7"
```

For More Information:
2. Oracle JDBC driver classes should be placed in the CATALINA_HOME/lib/ folder of the Tomcat instance. For Oracle, either class 12.jar or ojdbc14.jar is used.

   By default, Tomcat accepts only *.jar. If the driver is in ZIP format, then rename it to .jar and then deploy it in the jar directory. Based on the version used in the environment, you can download the Oracle JAR for free using the link, http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-10201-088211.html.

   In case you have installed the Oracle database version 9i, then you should use the oracle.jdbc.driver.OracleDriver class for JDBC connections, and for versions above 9i, you should use oracle.jdbc.OracleDriver class. Oracle.jdbc.driver.OracleDriver is deprecated and support for this driver will be discontinued from the next major release.

3. It's always mandatory to define the Document Type Definition (DTD) for the resource in the application web.xml. There is always a question that comes to the mind of the administrator, why can't we define the application specific DTD in the server web.xml? The answer to that question is very tricky. When the application is deployed, it will reference the application web.xml for the resource, but not for the server web.xml. The server web.xml should be used only for the server properties changes, such as the session parameter and so on, which references to the web/application server specific.

   <resource-ref>
       <description>Oracle Datasource for tomcat</description>
       <res-ref-name>jdbc/tomcat7</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <res-auth>Container</res-auth>
   </resource-ref>

For More Information:
4. After the previous step, the developer has to reference the JNDI in their code file and connect it to the database.

**DataSource for MySQL**

MySQL is one of the biggest open source databases currently supported by Oracle. It follows the same process as Oracle, but a few parameters vary. The following steps are to be performed to configure the datasource for MySQL:

1. The following lines of code provide the definition of datasource in `server.xml`. By default, these values are defined in the global section.

   ```xml
   <Resource name="jdbc/tomcat7" auth="Container"
            type="javax.sql.DataSource"
            maxActive="100" maxIdle="30" maxWait="10000"
            username="tomcatuser" password="tomcat"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/tomcat7"/>
   ```

2. The following lines of code provide the `web.xml` configuration for the application. This should be placed on the `WEB-INF/web.xml` for the application-specific content.

   ```xml
   <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                               http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
            version="2.4">
      <description>Tomcat 7 test DB</description>
      <resource-ref>
         <description>DB Connection</description>
         <res-ref-name>jdbc/tomcat7</res-ref-name>
         <res-type>javax.sql.DataSource</res-type>
         <res-auth>Container</res-auth>
      </resource-ref>
   </web-app>
   ```

3. The MySQL JDBC driver is deployed in the `CATALINA_HOME/lib/` folder of Tomcat. MySQL 3.23.47 or Connector/J 3.0.11-stable are the most common and widely used JAR files.

   You can download the MySQL JAR freely from the open source website, http://dev.mysql.com/downloads/.

---

**For More Information:**

4. One of the most important points which the Tomcat administrator should keep in mind is that, in MySQL, the DB should be configured with all privileges for the DB server user. Log in to the MySQL prompt and run the following command to grant the privileges:

```sql
mysql> GRANT ALL PRIVILEGES ON *.* TO tomcatuser@localhost IDENTIFIED BY 'tomcat7' WITH GRANT OPTION;
mysql> create database tomcat7;
mysql> use tomcat7;
mysql> create table testdata ( id int not null auto_increment primary key, foo varchar(25), bar int);
```

If you create the MySQL user without password, then the JDBC driver will fail to connect and you will have an authentication error in catalina.out.

**DataSource for PostgreSQL**

PostgreSQL is an open source and relational database. It is one of the oldest databases (15 years old). It can be installed on multiple OSes, such as Windows, Unix, MAC, and so on.

It has a four step configuration rule similar to Oracle as follows:

1. The following code provides the definition of datasource in server.xml. By default, these values are defined in the global section.

   ```xml
   <Resource name="jdbc/tomcat7" auth="Container"
           type="javax.sql.DataSource"
           driverClassName="org.postgresql.Driver"
           url="jdbc:postgresql://127.0.0.1:5432/tomcat7"
           username="tomcat7" password="tomcat" maxActive="20" maxIdle="10"
           maxWait="-1"/>
   ``

2. The PostgreSQL JDBC driver is deployed in the CATALINA_HOME/lib/postgresql-9.0-801.jdbc3.jar folder of Tomcat.

   Based on the version, the JDBC driver should be downloaded. For more reference on the driver version, refer to [http://jdbc.postgresql.org/download.html](http://jdbc.postgresql.org/download.html).
3. For the web.xml configuration of the application, use the following lines of code. This should be placed in the WEB-INF/web.xml for the application-specific content.

```xml
<resource-ref>
    <description>postgreSQL Tomcat datasource</description>
    <res-ref-name>jdbc/tomcat7</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

At the end of these steps, the developer will reference the JNDI in his/her code file and connect to the database.

**Comparison of the datasource for common databases**

Until now, we have seen how the datasource is configured on different databases. Let's quickly compare and find out what are the different syntaxes for each database:

- **Oracle**: The following mentioned code describes the datasource parameter for the Oracle database:

```xml
<Resource name="jdbc/tomcat7" auth="Container"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.OracleDriver"
    url="jdbc:oracle:thin:@127.0.0.1:1521:test"
    description="test database for tomcat 7"
    username="admin" password="admin" maxActive="20"
    maxIdle="10" maxWait="-1"/>
```

- **MySQL**: The following mentioned code describes the datasource parameter for the MySQL database:

```xml
<Resource name="jdbc/tomcat7" auth="Container"
    type="javax.sql.DataSource"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://127.0.0.1:5432/tomcat7"
    username="tomcat7" password="tomcat" maxActive="20" maxIdle="10"
    maxWait="-1"/>
```

For More Information:
Configuration and Deployment

- **PostgreSQL**: The following mentioned code describes the datasource parameter for the PostgreSQL database:

  ```xml
  <Resource name="jdbc/tomcat7" auth="Container" type="javax.sql.DataSource"
  driverClassName="org.postgresql.Driver"
  url="jdbc:postgresql://127.0.0.1:5432/tomcat7"
  username="tomcat7" password="tomcat" maxActive="20" maxIdle="10"
  maxWait="-1"/>
  ```

In the previous figure, we have defined each `driverClassName`, port, and JDBC driver for each database and tabulated to conclude that if you know the details for connectivity with the database, you can configure any new database very easily.

<table>
<thead>
<tr>
<th>Database</th>
<th>oracle</th>
<th>Mysql</th>
<th>Postgresql</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>1521</td>
<td>3306</td>
<td>5432</td>
</tr>
<tr>
<td>JDBC driver</td>
<td>ojdbc14.jar</td>
<td>MySQL 3.23.47</td>
<td>postgresql 9.0-801.jdbc3.jar</td>
</tr>
</tbody>
</table>

Every vendor has a predefined set of libraries through which you can connect to its database. If you need to connect to any other database, which is not mentioned here, then you can visit the vendor websites for support information.

**Tomcat Manager configuration**

The Tomcat Manager is a very powerful tool for Tomcat administration. In production server issues, it's not possible to be in the data center at all times. Sometimes, we have to connect to Tomcat remotely to resolve the issues and that is when the Tomcat Manager is very useful for handing a critical issue. It comes with the following features:

- Deployment of a new application remotely
- Idle session clearing
- Undeployment of an application without restarting the container
- Analysis of memory leaks
- JVM status
- Server status

For More Information:
Enabling the Tomcat Manager

By default, the Tomcat Manager is disabled in Tomcat 7. To enable the Tomcat Manager, you have to do the configuration in the default file, that is, `tomcat-users.xml` in the `conf` folder of Tomcat 7.

In this file, user roles and their authentication are configured. Let's quickly discuss the configuration parameters for enabling the Tomcat Manager.

Before enabling the Tomcat Manager, an authentication window will pop-up while browsing the Tomcat page, as shown in the following screenshot:

The following screenshot shows the `tomcat-users.xml` section before enabling the user properties:

```xml
<tomcat-users>
<!--
NOTE: By default, no user is included in the "manager-gui" role required to operate the "/manager/html" web application. If you wish to use this as you must define such a user - the username and password are arbitrary. -->
<!--
NOTE: The sample user and role entries below are wrapped in a comment and thus are ignored when reading this file. Do not forget to remove <!...> that surrounds them. -->
<!--
</tomcat-users>
```

For More Information:

After enabling the Tomcat Manager, the user will get a message in the command prompt, as shown in the following screenshot:

By default, Tomcat 7 comes with two users, tomcat and role1. If you want to add more users based on your system requirement, you can add here and define the role. Once you enable the Tomcat user configurations, this configuration will be in effect after the Tomcat recycle.

You can browse the Tomcat Manager using the URL http://localhost:8080/ and click on Manager App, as shown in the the following screenshot:

Once the authentication page is displayed, provide the user ID/password (user = admin, password = admin) as it's already defined in tomcat-users.xml. Click on OK. The pop-up will redirect it to the Tomcat Manager console, as shown in the following screenshot:
Through this console, we can deploy the new application or modify the current application's state to stop, undeploy, start, reload, clear sessions, and so on. Also, we can check the current status of the server by clicking on the Server Status, as shown in the following screenshot:
The following screenshot shows the **Server Status**:

The server status will define the following details:

- **JVM status**
  - Max memory
  - Total memory
  - Free memory

- **Connection of AJP port 8009**
  - Connection state
  - Data sent
  - Data received
  - Client
  - Virtual host

For More Information:

- Connection on HTTP port 8080
  - Connection state
  - Data sent
  - Data received
  - Client
  - Virtual host

- Server information
  - Tomcat version
  - OS version
  - JVM version
  - System architecture

**Context path**

The context path is a key element of a web application. It's also used for a virtual host. Virtual hosting can be defined as a method through which you can host multiple domain names on the same web server or a single IP.

The context path is also used to define the URL mapping for the .war files.

Many people ask why we need the context path. Instead, can we deploy the application on one root directory? The answer is, by defining the context path, we minimize the load on the server. When the server gets the request with the URL, it will check the server.xml or context path for the defined URL. If it's found, then the URL will be served from here, otherwise the server has to search all the deployed WAR files. Hence, the context path reduces the CPU cycle.

The second important advantage is, it gives us freedom to customize the application based on our requirement, such as logging, appBase, DB connection, and so on.

Let's consider a scenario for a large enterprise where a single application needs to be deployed on 100 Tomcat servers. Now it's impossible to deploy the application on every server, and so, in that case Common NAS share is used for the application deployment.

For More Information:
Enabling the context path

The context path in Tomcat can be enabled in two ways:

- GUI using the Tomcat Web Application Manager
- Command-line configuration in `server.xml`

GUI using the Tomcat Web Application Manager

For enabling the context path in the Tomcat Manager, you have to first log in to the Tomcat Manager app using the URL `http://localhost:8080`. Then click on Manager App, as shown in the following screenshot:

It then displays the Tomcat Web Application Manager console and its features, as shown in the following screenshot:
You can create the context path using the **Deploy** tab. Click on **Browse** and select the required WAR file. Then click on **Deploy**. It will take 10 to 15 seconds to deploy the application and you will see a page similar to the following screenshot:

---

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

---

The following screenshot shows the application deployment status and administrative controls such as **Stop**, **Reload**, and **Undeploy**:

---

For More Information:

Configuration and Deployment

Once the application is deployed successfully, you can browse the application using the URL http://localhost:8080/sample, as shown in the following screenshot:

![Sample "Hello, World" Application](image)

To prove that they work, you can execute either of the following links:
- To a JSP page
- To a servlet

Command-line configuration in server.xml

Another way of adding the context path in Tomcat 7 is by editing server.xml. But, you need to have a good understanding of XML. Let's quickly discuss the changes that need to be done on the Tomcat server.

```
<Context path="/sample" docBase="/opt/" reloadable="true"
swallowOutput="true">
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <Logger className="org.apache.catalina.logger.FileLogger"
      prefix="www-sample-com-log." suffix=".txt"
      timestamp="true"/>
</Context>
```

Now, it's time to discuss the parameters defined in the context path. The previous screenshot shows the details for the context path.

- **path="/sample"**: It defines the path URL for the server request, for example, http://localhost:8080/sample.
- **docBase="/opt/"**: It defines the document root for the context path. In simple language, this parameter defines the place from where the deployment .war file gets picked up.
- `reloadable="true"`: If this parameter is `true`, then every change done on the WAR file will be in effect automatically without a Tomcat recycle.
- `swallowOutput="true"`: If this parameter is set to `true`, then the output for `System.out` and `System.err` will be redirected to the application log.

It's always recommended to take the backup of the existing configuration file before performing changes in Tomcat.

**Deployment in Tomcat 7**

Deployment is basically defined as the installation of the WAR files in the web application. In other words, we can define the unpacking of the WAR file in the Tomcat `webapps` directory.

**Structure of the WebArchive**

You develop your web application within a specified directory structure so that it can be archived and deployed on Tomcat 7. All servlets, classes, static files, and other resources belonging to a web application are organized under a directory hierarchy. The root of this hierarchy defines the document root of your web application. All files under this root directory can be served to the client, except for files under the special directory `WEB-INF`, located under the root directory. The name of your web application is used to resolve requests for components of the application.

Always place private files (files which are not required to serve to the client) in the `WEB-INF` directory, under the root directory. All files under `WEB-INF` are private, and are not served to the client.

- `WebApplicationName/:` In this directory (or a subdirectory), all the static files, such as HTML and JSP files are placed. This directory is the document root of your web application.
- `/WEB-INF/web.xml`: It contains the deployment descriptor for the web application. Resources specific to the application are placed here.
- `/WEB-INF/classes`: This contains all the server-side classes or your application-specific third-party classes.
- `/WEB-INF/lib`: This directory contains JAR files used for the JSP completion.

For More Information:

Configuration and Deployment

- **web.xml**: It contains the details of all your dynamic files (servlets and JSP) and also other configuration-related information such as session time out and defining the datasource (access to DB).

  ```xml
  <servlet>
    <servlet-name>classB</servlet-name>
    <servlet-class>class.classB</servlet-class>
  </servlet>

  In the previous snippet, we are mapping the name to the servlet class (when Tomcat 7 starts, it will create an object of the class and map it to the name we have provided in the `servlet-name` field).

  ```java
  classB = new class.classB();
  ```

  ```xml
  <servlet-mapping>
    <servlet-name> classB </servlet-name>
    <servlet-name> classB </servlet-name>
  </servlet-mapping>
  ```

### Archive Files

In most production environments, you receive a deployment unit as an archive file from the developer. An archive file is a single file that contains all of an application or module's classes, static files, directories, and deployment descriptor files. Archive files are typically created by using the JAR utility or Ant JAR tool.

Deployment units that are packaged using the JAR utility have a specific file extension depending on the type, as explained in the following points:

- EJBs are packaged as `.jar` files
- Web applications are packaged as `.war` files
- Resource adapters are packaged as `.rar` files
- Enterprise applications are packaged as `.ear` files, and can contain any combination of EJBs, web applications, and resource adapters
- Web services can be packaged either as `.ear` files or as `.war` files

### Exploded archive directories

An exploded archive directory contains the same files and directories as a JAR archive. However, the files and directories reside directly in your filesystem and are not packaged into a single archive file with the JAR utility.

A deployment unit should be deployed as an exploded archive directory, rather than a single archive file, in the following circumstances:

- You want to perform partial updates to a deployed application without redeploying the entire application.

For More Information:  
• You want to use the Tomcat Manager to dynamically edit and persist selected deployment descriptor values for the deployment.

It's not possible to edit deployment descriptor values in the console for deployments from the archive files or .war files.

• You are deploying a web application that contains static files that you will periodically update. In this case, it is easier to deploy the application as an exploded directory, because you can update and refresh the static files without re-creating the archive.

Deployment operations
The deployment tools provide support for performing these common deployment operations:

• **Deploy**: It makes deployment source files available to target servers and loading classes into class loaders so that applications are available to clients.

• **Redeploy**: It updates a deployment unit or part of a deployment unit (for example, a WAR, a module within a WAR, or a static file in a Web Application) that is currently deployed and available to clients. When redeploying an entire application, all of the application's modules must redeploy successfully or the entire application is stopped.

An application becomes unavailable to clients during redeployment. The Tomcat 7 server doesn't guarantee the operation of the application and deployment task if there is an access from the client at this time. For this reason, redeployment is not recommended for use in a production environment.

• **Stop**: This unloads an application's classes and makes an application unavailable to clients. Stopping still leaves the deployment files and deployment name available to the target servers for subsequent redeployment or starting.

• **Start**: It reloads an application's classes into class loaders and makes the application available to clients. Starting requires that the deployment files be available on the target servers as a result of an earlier deployment.

• **Undeploy**: This stops a deployment unit and then removes its deployment files and the deployment name from the target servers.

For More Information:
An application becomes unavailable to clients during undeployment. The Tomcat 7 server doesn't guarantee the operation of the application and deployment task if there is an access from the client at this time.

**Types of deployment**

The deployment staging mode determines how deployment files are made available to the target servers that must deploy an application or standalone module. The Tomcat 7 server provides three different options for staging files listed as follows:

- Stage mode
- Nostage mode
- External_stage mode

The following table describes the behavior and best practices for using the different deployment staging modes:

<table>
<thead>
<tr>
<th>Deployment Staging Mode</th>
<th>Behavior</th>
<th>When to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage</td>
<td>The Tomcat administrator first copies the deployment unit source files to the staging directories of the target servers and then the target servers deploy them using their local copy of the deployment files.</td>
<td>Deploying small or moderate-sized applications to multiple Tomcat 7 server instances. Deploying small or moderate-sized applications to a cluster.</td>
</tr>
<tr>
<td>Nostage</td>
<td>The Tomcat administrator does not copy the deployment unit files. Instead, all servers deploy using the same physical copy of the deployment files, which must be directly accessible by the Tomcat administrator and target servers. Nostage deployments of exploded archive directories is not recommended.</td>
<td>Deploying to a single server instance. Deploying to a cluster on a multi-homed machine. Deploying very large applications to multiple targets or to a cluster where deployment files are placed on the server.</td>
</tr>
</tbody>
</table>
### Deployment Staging Mode

<table>
<thead>
<tr>
<th>Deployment Staging Mode</th>
<th>Behavior</th>
<th>When to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>External_stage</td>
<td>The Tomcat administrator does not copy the deployment files. Instead, the administrator must ensure that deployment files are distributed to the correct staging directory location before deployment (for example, by manually copying files prior to deployment). With external_stage deployments, the Tomcat administrator requires a copy of the deployment files for validation purposes. Copies of the deployment files that reside in the target servers’ staging directories are not validated before deployment.</td>
<td>Deployments where you want to manually control the distribution of deployment files to the target servers. Deploying to the server instance where third-party applications or scripts manage the copying of deployment files to the correct staging directories. Deployments that do not require a dynamic update of selected deployment descriptors via the Tomcat Manager (not supported in external_stage mode). Deployments that do not require partial redeployment of the application components.</td>
</tr>
</tbody>
</table>

## Ways of application deployment in Tomcat 7

Deployment of applications can be done in many ways in Tomcat 7. There are five different ways which are widely known and accepted in the various industries displayed in the following figure:

| War deployment | Tomcat Deployer (TCD) | Unpacked deployment | Tomcat 7 Deployer | Using ANT scripts | Tomcat Manager |

For More Information:
Configuration and Deployment

- **War deployment**: You can deploy the WAR file in the CATALINA_BASE directory of Tomcat and restart Tomcat to view the application. This approach is widely used in the production environment.

- **Unpacked deployment**: In this deployment method, the WAR file is extracted on the CATALINA_BASE directory for the instance. This method is commonly used in the development server.

- **Tomcat Manager**: It's a very good tool which is widely used in the production environment, mainly in remote infrastructure deployment. You can log in to the Tomcat browser from your system and deploy. Then click on the new web application deployment, as shown in the next screenshot:

  ![Tomcat Web Application Manager](image)

  You can create the context path using the **Deploy** tab. Click on **Browse** and select the required WAR file. Then click on **Deploy**. It will take 10 to 15 seconds to deploy the application and you will see a page similar to the following screenshot:

  ![WAR file to deploy](image)

  For More Information:
The following screenshot shows the application deployment status and administrative control such as Stop, Reload, and Undeploy:

```
Example None specified Hello World Application true 0
```

Once the application is deployed successfully, as shown in the following screenshot, you can browse the application using the URL, http://localhost:8080/sample:

- **Using ANT scripts**: You can also deploy the application using the ANT scripts. These scripts contain the information of the source/destination and target file. For doing this deployment, the Tomcat instance should be running.

- **TCD (Tomcat Deployer)**: It is a tool which is used for application deployment. ANT should be installed for the TCD to be working and the Tomcat instance should be running. There is no need to install the TCD on the Tomcat instance.

### Common issues in deployment, configuration, and their troubleshooting

There are multiple issues which may arise after the deployment and configuration on Tomcat. Let's discuss the different issues:

**Scenario 1:**

**Issue**: Users complain that after the deployment, they can still view the old code.

**Troubleshooting steps:**

- Check if the latest file is present on the doc base.
- Check the `catalina.out` in the `logs` directory of Tomcat 7 and whether the `WAR` filename is deployed or not.

For More Information:

• If both are checked and the issue still persists, then stop the Tomcat service and clear the content of the temp directory under the work/Catalina/localhost using the following command:

```bash
cd /opt/apache-tomcat-7.0.12/temp/
rm -rf ../temp/*
cd /opt/apache-tomcat-7.0.12/work/Catalina/localhost/
rm -rf ../localhost/*
```

• Restart the Tomcat service and ask the user to test the application.

Scenario 2:

**Issue:** Users complaining that they can view the current deployed code on one node and the other node still displays the previous version of the code.

**Troubleshooting steps:**

• Check if the latest file is present on the doc base.

• Check the catalina.out in the logs directory of Tomcat 7 and whether the WAR filename is deployed or not.

If both are checked and the issue still persists, then stop the Tomcat service on node2. Replicate the code from node1 and clear the content of the temp directory under the work/Catalina/localhost using the following command:

```bash
cd /opt/apache-tomcat-7.0.12/temp/
rm -rf ../temp/*
cd /opt/apache-tomcat-7.0.12/work/Catalina/localhost/
rm -rf ../localhost/*
```

• Restart the Tomcat service and ask the user to test the application. Also, check the database status on node1 and node2, if they are in replication.

• Connect the database from both the nodes.

Scenario 3:

**Issue:** The Tomcat instance is not coming up after the changes made to server.xml.

**Troubleshooting steps:**

• Go to the Tomcat bin directory.

• Then, run the configtest.sh. It will give you the following output:

```bash
[root@localhost ~]# cd /opt/apache-tomcat-7.0.12/bin/
[root@localhost bin]# ./configtest.sh
Using CATALINA_BASE: /opt/apache-tomcat-7.0.12
```

For More Information:
Using CATALINA_HOME: /opt/apache-tomcat-7.0.12
Using CATALINA_TMPDIR: /opt/apache-tomcat-7.0.12/temp
Using JRE_HOME: /opt/jdk1.6.0_24
Using CLASSPATH: /opt/apache-tomcat-7.0.12/bin/bootstrap.jar:/opt/apache-tomcat-7.0.12/bin/tomcat-juli.jar

Error:-
org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:435)
Caused by: java.net.BindException: Address already in use
   at java.net.PlainSocketImpl.socketBind(Native Method)
   at java.net.PlainSocketImpl.bind(PlainSocketImpl.java:383)
   at java.net.ServerSocket.bind(ServerSocket.java:328)
   at java.net.ServerSocket.<init>(ServerSocket.java:194)
   at java.net.ServerSocket.<init>(ServerSocket.java:150)

- It means that Tomcat is already running. Then, stop the web server and clear the temp directory.
- Restart the services again.

Summary
In this chapter, we have discussed the configuration of Tomcat including data source configuration for the different databases (Oracle, MySQL, and PostgreSQL) and the context path creation using a sample application, various ways to perform deployment including deployment using the Tomcat Manager for the sample application. We also discussed troubleshooting of common issues.

In the next chapter, we will discuss performance tuning for Tomcat 7 in terms of the JVM and OS level.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.