|  |
| --- |
| Apache tomcat5.5  Tomcat is an open source web server developed by Apache Group. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies.  **Apache** *“The* Apache *software foundation provides support for the* Apache *community of open-source software projects. The* Apache *project are characterized by a collaborative, consensus based development process,* an *open and pragmatic software license, and d desire to create high quality software that leads the way in its field.’* Amongst the projects that come under the 'Apache” banner are the HTTP Web Server from which the whole Apache project has grown, and which is the container used for the majority of web sites worldwide, Ant (a build tool which allows the developer excellent control of the compiling and bundling processes), and Jakarta.  **Jakarta** *“The Jakarta Project creates and maintains open source solutions* on *the Java platform for distribution to the public at no charge. Jakarta products are developed by and distributed through various sub-projects.”* Jakarta is the name for the Apache project which deals with the provision of open source additions in lava. More than 20 such additions (known as sub-projects) are listed on their web site, including Struts1 and Tomcat.  **Tomcat** Tomcat is a servlet container for the JavaServlets and Java Server Pages. It provides a Java Virtual Machine and associated elements to give a complete Java Runtime Environment, and it also provides web server software to make that environment accessible on the Web. Configuration and management tools are also provided, with configuration data largely held in XML.  It's worth noting that Tomcat is much more than just an implementation of Sew- lets and JSPs, it's the official reference implementation and the standard against which all other suppliers of containers for Servlet’s and JSPs must measure their produces.. It means that developers know that if they develop code that works under Tomcat, that code should work under other containers that conform to the standards set.  **Standard Directory Structure**   Java web applications conform to a standard directory structure, specified in the Java EE specification, which designates exactly where specific Java objects should be placed within a program.  This method is used for three reasons.   First of all, standard directory structure is designed to integrate non-web Java directory idioms with the needs of a web container.  In Java, a correlation between class names and directory structure helps the class loader locate and load the libraries it needs.  In the Java [servlet](http://www.mulesoft.com/tomcat-servlet) specification, this idea is preserved and extended to make the deployment and loading of applications and classes on the server a more logical process. Document Root This is the root directory of the web application.  In addition to the WEB-INF directory, which we discuss below, this directory includes any resource that should be visible to the web container (in other other words, items that are directly accessible via a URL).  This includes items such as static HTML pages and images, JSP pages, client-side class files, and client-side JAR files.  These files can be loosely collected within the Document Root or organized into subfolders. WEB-INF This directory, which is contained within the Document Root, is invisible from the web container.  It contains all resources needed to run the application, from Java classes, to JAR files and libraries, to other supporting files that the developer does not want a web user to access. This folder also contains the application's deployment descriptor, an important XML file discussed below. WEB-INF/classes This directory, contained within WEB-INF, is a repository of all the Java classes and packages required by the application (other than those contained in Tomcat's common or shared class repositories).  These will be loaded when Tomcat starts the application, and are only visible to this application.  Class files can be organized into packages within this directory, as long as they are organized with the standard Java naming structure. WEB-INF/lib This directory contains JAR files required by the web application.  Classes contained in these files will be decompressed and loaded at application start up. WEB-INF/web.xml This file, known as the "deployment descriptor" is an important XML configuration file, which is read by Tomcat when the application is first deployed.  This file contains a list of all servlets and resources required by the application, in a hierarchical format that can be used to define security constraints, map servlets to specific URLs, and more. META-INF/context.xml In Tomcat, a Context represents a single web application.  Tomcat uses the Context configuration element to contain information about components required by a given application, such as databases, realms, or custom loaders.  Additionally, the Context element can be configured with a wide variety of attributes that control things such as [logging](http://www.mulesoft.com/tomcat-logging),[reload](http://www.mulesoft.com/tomcat-reload) permissions, caching, and more.  In older versions of Tomcat, Contexts had to be configured in Tomcat's central server.xml configuration file.  However, since Tomcat 5.x, Tomcat has supported the configuration of Contexts at [deploy](http://www.mulesoft.com/tomcat-deploy) time via a "context fragment" contained in the META-INF directory.  When the application is deployed from a WAR file, this file will be copied into the appropriate directory and renamed according to the application's context path.  For more information about the Context element, check out this article: Understanding the [Tomcat Context](http://www.mulesoft.com/tomcat-context). Building and Deploying Web Applications on Tomcat The process of packaging a Java web application and publishing it to an application server consists of two steps: build and deployment.  The purpose of the build process is to package the Java source of a project in a format that can be used on an application server.  Usually, this process is automated using a tool such as [Ant](http://ant.apache.org) or [Maven](http://www.mulesoft.com/tomcat-maven).    During the build process, the project's Java source code is compiled into class files (which may be packaged into JARs), a directory structure based on the standard structure described above is created, and any dependencies listed in the project are copied into the appropriate bins for Tomcat to find and load them.  Finally, the project is left in one of two formats.  If the application is under development, it may be left in an exploded format, so that its individual classes can be replaced and reloaded as changes are made, either manually or using an IDE.  If the application is ready for production, or if the developer is only making minor changes and finds it more efficient to simply redeploy the application after each build, the application will be packaged as a Web Application aRchive, or WAR file.  This file is then "deployed" to Tomcat, using either a static or dynamic method. Static Deployment Static deployment refers to the deployment of an application to a server while it is not running.  First, the packaged or exploded application is copied to the Tomcat "appBase", a directory which is configured on a per-Host basis.  The default appBase location is "$CATALINA\_BASE/webapps", or "$CATALINA\_HOME/webapps", if no base directory has been defined.  Next, if the application does not contain a Context fragment, one must be added within the appropriate Host element in Tomcat's server.xml configuration file.  Finally, the "deployOnStartup" attribute of the relevant Host element must be set to "true" in server.xml.  If all of these steps are completed correctly, Tomcat will deploy the application on [start](http://www.mulesoft.com/tomcat-start)-up, creating a context path as specified in the deployment descriptor and Context fragment. Dynamic Deployment Dynamic deployment refers to the deployment of an application to a running server.  Tomcat supports a number of different dynamic deployment methods. Deployment Using the Tomcat Manager Tomcat Manager, the management and administration tool included with all distributions of Tomcat, includes a number of deployment capabilities, accessible using either its URI commands or its web console.  Manager can deploy applications to a running server from remote and local directories, from Tomcat's appBase, or from a Context fragment that includes a docBase path, and can handle both exploded and compressed applications. autoDeploy In addition to the "deployOnStartup" attribute, the Host element also supports an attribute named "autoDeploy".  When this attribute is set to "true", Tomcat will enable dynamic deployment features for this Host.  Instead of waiting until start-up to check the appBase for new WARs and exploded applications, the Host uses a background process to check for new applications in the directory at a specified interval.  Thus, a new application can be deployed to a running server simply by dragging it into the appBase.  **Structure of tomcat projects:**    **Structure of tomcat** |