# Tomcat Configuration − A Step By Step Guide

Once you get Tomcat up and running on your server, the next step is configuring its basic settings. Your initial configuration process will consist of two tasks, which are explained in detail in this article. The first is editing Tomcat's XML configuration files (/tcat/tomcat-configuration#xml_config), and the second is defining appropriate environment variables (/tcat/tomcat-configuration#env_variables).

## XML Configuration Files

The two most important configuration files to get Tomcat up and running are called server.xml and web.xml. By default, these files are located at TOMCAT-HOME/conf/server.xml and TOMCAT-HOME/conf/web.xml, respectively.

Don't do the same configuration work twice. Try Tcat (/misc/forms/download/tcat-lbox-form.php) − server profiles let you save common configurations and apply them to multiple Tomcat instances with a single click.

### SERVER.XML

The server.xml file is Tomcat's main configuration file, and is responsible for specifying Tomcat's initial configuration on startup as well as defining the way and order in which Tomcat boots and builds. The elements of the server.xml file belong to five basic categories − Top Level Elements, Connectors, Containers, Nested Components, and Global Settings. All of the elements within these categories have many attributes that can be used to fine-tune their functionality. Most often, if you need to make any major changes to your Tomcat installation, such as specifying application port numbers, server.xml is the file to edit.

You can find comprehensive documentation for these options on Apache's Tomcat Documentation pages, but here's some information on some of the most important elements to get you started with your configuration!



# Top Level Elements

## Server



This element defines a single Tomcat server, and contains the Logger and ContextManager configuration elements. Additionally, the Server element supports the "port", "shutdown", and "className" attributes.

The port attribute is used to specify which port Tomcat should listen to for shutdown commands. The shutdown attribute defines the command string to be listened for on the specified port to trigger a shutdown. The className attribute specifies which Java class implementation should be used.

## Service

This element, which can be nested inside a Server element, is used to contain one or multiple Connector components that share the same Engine component. The main function of this component is to define these components as a single service. The name of the service that will appear in logs is specified using the Service element's "name" attribute.

## Connectors

By nesting one Connector (/tomcat-connector) (or multiple Connectors) within a Service tag, you allow Catalina to forward requests from these ports to a single Engine component for processing. Tomcat allows you to define both HTTP and AJP connectors.

## HTTP Connector

This element represents an HTTP/1.1 Connector, and provides Catalina with stand-alone web server functionality. This means that in addition to executing servlets and JSP (/tomcat-jsp) pages, Catalina is able to listen to specific TCP ports for requests. Each Connector you define represents a single TCP port Catalina should listen to for HTTP requests. When configuring your HTTP connectors, pay close attention to the "minSpareThreads", "maxThreads", and "acceptCount" attributes. The "maxThreads" attribute is of particular importance. This attribute controls the maximum number of threads that can be created to handle requests exceeding the number of available threads. Setting this value too low will cause requests to stack inside the server socket, which will begin refusing connections once it is full. Comprehensive testing will help you avoid this problem.

## AJP Connector

This element represents a connector that is able to communicate with the AJP protocol. The main role of this element is to help Tomcat integrate with an installation of Apache. The most common reason why you would want this functionality is if you plan to use Apache to serve static content in front of Tomcat. This technique is intended to free up more power for dynamic

page generation and load balancing, so if fast performance (/tomcat-performance) is a concern for your application, this is something to consider. AJP Connectors can also be used to expose Apache's SSL (/tomcat-ssl) processing functionality to Tomcat.



## Containers

These elements are used by Catalina to direct requests to the correct processing apparatus.



## Context



This element represents a single web application, and contains path information for directing



requests to the appropriate application resources. When Catalina receives a request, it



attempts to match the longest URI to the context path of a given Context (/tomcat-context) until it finds the correct element to serve the request. The Context element can have a maximum of one nested instance per element of the utility elements Loader, Manager (/tomcat-manager), Realm, Resources, and WatchedResource. Although Tomcat allows you to define Contexts within "TOMCAT-HOME/conf/server.xml", this should generally be avoided, as these central configuration settings cannot be reloaded without restarting Tomcat, which makes editing Context attributes more invasive than necessary.

## Engine

This element is used in conjunction with one or more Connectors, nested within a Service element, and is responsible for processing all requests associated with its parent service. The Engine element can only be used if it is nested within a Service element, and only one Engine element is allowed within a given Service element.Pay close attention to the "defaultHost" attribute, which defines the Host element responsible for serving requests for host names on the server that are not configured in server.xml. This attribute must match the name of one of the Host elements nested inside the Engine element in question. Also, it's important to assign a unique, logical name to each of your Engine elements, using the "name" attribute. If a single Server element in your server.xml file includes multiple Service elements, you are required to assign a unique name to every Engine element.

## Host

This element, which is nested inside of the Engine element, is used to associate server network names with Catalina (/tomcat-catalina) servers. This element will only function properly if the virtual host in question is registered with the managing DNS of the domain in question.

One of the most useful features of the Host element is its ability to contain nested Alias elements, which are used to define multiple network names that should resolve to the same virtual host.

## Cluster

The Cluster (/tcat/tomcat-cluster) element is used by Tomcat to provide context attribute replication, WAR deployment, and session replication, and can be nested within either the Engine or the Host element. The Manager, Channel, Valve, Deployer, and ClusterListener elements are nested inside of it. More information on these elements and how they are used can be found on Apache's Tomcat Configuration page. Although this element is highly configurable, the default configuration is generally enough to meet most users' needs.



## Nested Components



These elements are nested inside of container elements to define additional functionalities.



## Listeners

These elements, which can be nested inside Server, Engine, Host, or Context elements, point to a component that will perform an action when a specific event occurs.

While most components possess the className attribute, to select different implementations of the element, the Listener element is unique in that there are a number of unique implementations other than the default, and as of Tomcat 6.0, all of these implementations require that the Listener element be nested within a Server element. Thus, setting this attribute correctly is important. The implementations currently available are an APR Lifecycle Listener, a Jasper Listener, a Server Lifecycle Listener, a Global Resources Lifecycle Listener, a JMX (/tcat/tomcat-jmx) Remote Lifecycle Listener, and a JRE Memory (/tomcat-memory) Leak Prevention Listener.

## Global Naming Resources

This element is used to specify global Java Naming and Directory Interface (JNDI) resources for a specific Server, distinct from any per-web-application JNDI contexts. If you wish, you can declare JNDI resource lookup characteristics for <resource-ref> and <resource-env-ref> within this element by defining them and linking to them using <ResourceLink>. The results of this method are equivalent to including <resource-ref> elements in an application's "/WEB-INF/web.xml" file. If using this technique, be sure to define any additional parameters necessary to specify and configure the object factory and its properties.

## Realm

This element, which can be nested inside of any Container element, defines a database containing usernames, passwords, and roles for that Container. If nested inside a Host or Engine element, characteristics defined in the Realm element are inherited by all lower-level containers by default. It is important to set the "className" attribute of this element correctly, as a variety of implementations exist, to provide different types of Container Managed Security (/tomcat-security). These implementations are used to expose Catalina to other systems of user security management such as JDBC (/tomcat-jdbc), JNDI, and DataSource.

## Resources

This element has one simple job − directing Catalina to static resources used by your web applications. These resources include classes, HTML, and JSP files. Utilizing this element allows Catalina to access files contained in places other than the filesystem, such as resources contained in WAR archives or JDBC databases. It is vital to remember that this technique of allowing web applications access to resources contained off-filesystem can only be used if the application in question does not require direct access to resources stored on the filesystem.

### Valve

Valve components are nested inside Engine, Host, and Context elements to insert specific functionalities into the request processing pipeline. This is a very versatile element. Types of Valve elements range from authenticators to filters to fixes for WebDAV errors. Many of these types of Valves can only be nested within specific elements. Needless to say, paying attention to this element's "className" attribute is essential. Extensive documentation on the types of Valve elements and their uses is available on Apache's Tomcat Configuration page.<

## Web.XML

The web.xml file is derived from the Servlet (/tomcat-servlet) specification, and contains information used to deploy (/tomcat-deploy) and configure the components of your web applications. When configuring Tomcat for the first time, this is where you can define servlet mappings for central components such as JSP. Within Tomcat, this file functions in the same way as described in the Servlet specification. The only divergence in Tomcat's handling of this file is that a user has the option of utilizing TOMCAT-HOME/conf/web.xml to define default values for all contexts. If this method is utilized, Tomcat will use TOMCAT-HOME/conf/web.xml as a base configuration, which can be overwritten by application-specific WEB-INF/web.xml files.

## Other important configuration files

A few other configuration files will be important as you get Tomcat up and running for the first time. Default lists of roles, users, and passwords that Tomcat's UserDatabaseRealm will use for authentication can be found in tomcat-users.xml. If you want to access any of the administrative tools that are packaged with Tomcat, you can edit this file to add admin (/tcat/tomcat-admin) and manager access. Default context settings applied to all deployed contexts of your Tomcat installation can be adjusted in the context.xml file. The catalina.policy file, which replaces the java.policy file packaged with your chosen JDK, contains permissions settings for Tomcat's elements. You can edit this file by hand or with policytool, an application packaged with any Java distribution 1.2 or later.

## Environmental variables

Finally, when configuring Tomcat for the first time, there are several environment variables that should be modified to suit your needs.

### JAVA_OPTS

(https://www.mulesoft.com)

Using this variable you can define the heap size of the JVM (/tcat/tomcat-jvm). Setting an appropriate value for this variable is crucial when deploying a new application that may require more or less heap size to function properly. Finding the proper values for these settings can help eliminate or reduce OOME messages.

Features (/tcat/leading-enterprise-apache-tomcat-application-server)

Contact (https://www.mulesoft.com/contact)

Resources (/tcat/understanding-apache-tomcat)

Support (/tcat/support-apache-tomcat)

**Free trial (https://anypoint.mulesoft.com/login/#/signup?apintent=generic)**

Documentation (https://docs.mulesoft.com/tcat-server/7.1/overview-of-tcat-server)

Login (https://anypoint.mulesoft.com/login/#/signin?apintent=generic)

## CATALINA_HOME

This variable specifies the location of your Tomcat installation. Tomcat's startup scripts will attempt to guess the value of this variable, but it is a good idea to simply set it to the correct value yourself to avoid any problems.

## CATALINA_OPTS

This variable is used to set various Tomcat-specific options. This variable can be used to set environmental variables that override your JAVA_OPTS settings for Tomcat only, which is useful if you are running multiple Java applications on a single JVM.

### Related articles

- Tomcat 6 - New Features, migration and Tomcat 7 (/tcat/tomcat-6)
- The Tomcat webapp quick reference guide (/tcat/tomcat-webapp)
- Latest release of Metasploit testing framework includes Tomcat modules (/tcat/tomcat-targeted-by-vulnerability-tester)
- Tcat -- Tomcat monitoring & diagnostics (/tcat/server-tomcat-monitoring-diagnostics)
- The pros and cons of using Tomcat with JMX for administration (/tcat/tomcat-jmx)

(https://www.mulesoft.com)

Sign up for our newsletter

Email >

Features (/tcat/leading-enterprise-apache-tomcat-application-server)
Developers (https://developer.mulesoft.com/) Blog (https://blogs.mulesoft.com) Terms (/content/terms-service)

Privacy (https://www.salesforce.com/company/privacy/)

Privacy Shield (https://www.mulesoft.com/content/dam/web/en_us/www/documents/legal/Privacy/privacy-shield-notice.pdf)

Contact (/contact) Resources (/tcat/understanding-apache-tomcat) Support (/tcat/support-apache-tomcat)

Cookie preferences Contact (/contact) 1-415-229-2009 (tel:+14152292009)

Free trial (https://anypoint.mulesoft.com/login/#/signup?apintent=generic)

Documentation (https://docs.mulesoft.com/tcat-server/7.1/overview-of-tcat-server)

Login (https://anypoint.mulesoft.com/login/#/signin?apintent=generic)

MuleSoft provides a widely used integration platform (https://www.mulesoft.com/platform/enterprise-integration) for connecting applications, data, and devices in the cloud and on-premises. MuleSoft's Anypoint Platform™ is a unified, single solution for iPaaS (https://www.mulesoft.com/lp/reports/gartner-magic-quadrant-ipaas) and full lifecycle API management (https://www.mulesoft.com/platform/api-management). Anypoint Platform, including CloudHub™ (https://www.mulesoft.com/platform/saas/cloudhub-ipaas-cloud-based-integration) and Mule ESB™ (https://www.mulesoft.com/platform/soa/mule-esb-open-source-esb), is built on proven open-source software for fast and reliable on-premises and cloud integration without vendor lock-in.

---



English Full site