JAVAWORLD

DEC 19, 2019 10:22 AM PST

About
Everything you need to know about Java programming tools and APIs, with code and examples.

TIP

What is Tomcat? The original Java servlet container

Everything you need to know about Tomcat: the high-availability Java application server for servlets, JSPs, and WebSockets

Apache Tomcat is a long-lived, open source Java servlet container that implements several core Java enterprise specs, namely the Java Servlet, JavaServer Pages (JSP), and WebSockets APIs.

An Apache Software Foundation project, Tomcat was first released in 1998, just four years after Java itself. Tomcat started as a reference implementation for the first Java Servlet API and the JSP spec. While it's no longer the reference implementation for either of these technologies, Tomcat remains the most widely used Java server, boasting a well-tested and proven core engine with good extensibility.

In this short introduction, you'll learn why many software shops choose Tomcat for running Java web apps. You'll get an overview of Tomcat and how it's used, as well as installation instructions for the most current version as of this writing.

Tomcat and the Java Servlet API
Tomcat 9 supports the Servlet 4.0 spec and requires JDK 8 or greater. Tomcat 8.5 supports several newer features like HTTP/2, but remains a Servlet 3 container.

What kind of server is Tomcat?

The Java ecosystem supports several kinds of application server, so let's disambiguate them and see where Tomcat fits in:

- A **servlet container** is an implementation of the Java Servlet specification, used primarily for hosting Java servlets.
- A **web server** is a server designed to serve files from the local system, like Apache.
- A **Java enterprise application server** is a full-blown implementation of the Java EE (now Jakarta EE) specification.

At heart, Tomcat is a servlet and JSP container. A Java servlet encapsulates code and business logic and defines how requests and responses should be handled in a Java server. JSP is a server-side view rendering technology. As the developer, you write the servlet or JSP page, then let Tomcat handle the routing.

Tomcat also contains the Coyote engine, which is a web server. Thanks to Coyote, it's possible to extend Tomcat to include a variety of Java enterprise specs and capabilities, including the Java Persistence API (JPA). Tomcat also has an extended version, called TomEE, that includes more enterprise features. I'll briefly introduce TomEE later in this article.

Let's begin with a look at using Tomcat to host servlets and JSPs.

[Learn Java from beginning concepts to advanced design patterns in this comprehensive 12-part course!]

Download and install Tomcat

Being a hoary ancient of the software world, a number of Tomcat versions are available. Information on version differences is available on the Tomcat homepage. You can usually just pick the latest stable version.

For our purposes, download the latest version of Tomcat, which currently is Tomcat 9. You have a choice of downloading Tomcat as an archive (`.zip` or `tar.gz`), or as an installed service. The best choice is up to you--unless of course you are not running on Windows, in which case you'll go for the archive. We'll use the archive for this article.

Windows installation for Tomcat

If you are running Windows and want to use the installer, simply download the .exe file and run it. Tomcat will install itself as a service with reasonable defaults. It will then inform you of where the install is, and you can proceed as though you had unpacked the archive there.

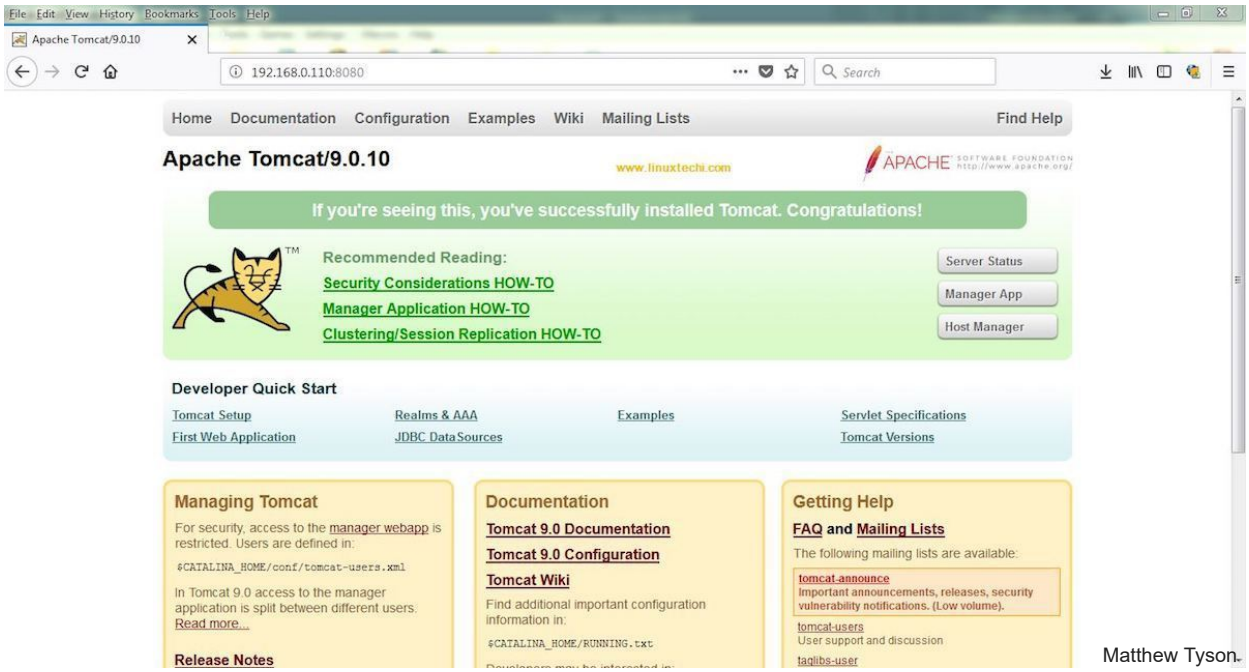
Step 1. Command-line installation

Go to the command-line and type `gunzip apache-tomcat-9.0.19.tar.gz` followed by `tar -xf apache-tomcat-9.0.19.tar`. This creates the following directories:

- **/bin** contains the scripts for executing Tomcat.
- **/webapps** is the location where you will deploy your apps.
- **/logs** is where Tomcat outputs its logs. Note that Tomcat's logs go into `/logs/catalina.out` by default. You can use this file to debug problems in conjunction with app-specific log files.
- **/lib** is where Tomcat looks for JARs. This is where you'll store additional packages not included with Tomcat, such as JPA.
- **/conf** is the config XML for Tomcat, where you can do things like adding users and roles for Tomcat.

Step 2. Start Tomcat

If you installed Tomcat as a service, it is already running. Otherwise, go ahead and start it up by entering `./catalina.sh start` at the command line. (Type `./catalina.sh` with no arguments to see all of the available commands). Now, you should be able to browse to Tomcat's welcome screen in a browser.



Matthew Tyson.

Figure 1. A screenshot of the Tomcat welcome page

Deploying applications in Tomcat

Tomcat's `webapps` directory is where you deploy an application. You can drop a `.war` file there and Tomcat will run it. A WAR file is the standard packaging for a web application resource: a JAR file with some additional files telling the container (in this case Tomcat) how to run it.

Aside from the standard packaging, there are three additional ways to deploy content in Tomcat.

Exploded deploy

An "exploded" web app is an application that isn't compressed into a WAR file, meaning it still contains all the elements laid out in directories and files. The Tomcat archive you unpacked shipped with several examples deployed in this manner, which you'll find in the `/webapps/examples` directory. The advantage of an exploded deploy is you can look at the files there without worrying about compression.

If you navigate to `http://localhost:8080/examples/`, you'll find a list of links. This page is rendered by Tomcat from the `/webapps/examples/index.html` file. Tomcat is serving an HTML file from the file system, which is an instance of Tomcat's Coyote engine acting as a web server.

You are free to explore the examples presented--they give you a good overview of Tomcat's capabilities for serving servlets, JSPs, and WebSockets.

Tomcat also includes a management app by default, found under the `/manager` path. Among other things, this app allows you to start, stop, and redeploy apps from a web console.

Serving static content

It's possible to serve files from the file system, or forward from Tomcat to another HTTP server like Apache. Another common setup is to put a file server like Apache or Nginx in front of Tomcat, and then forward your API requests into Tomcat. In these cases, the mod_JK library is used to configure Tomcat and Apache (or even another web server like IIS) to communicate.

For improved performance, primarily in delivering static content, Tomcat also offers native wrappers for Windows and Linux. This is known as *Tomcat APR* and more information is available here. These are not necessary for typical use cases, but good to know about.

Embedded Tomcat

For a long time, Jetty was the only server capable of running as an embedded server. That has changed, and now Tomcat can also run embedded. The idea in using an embedded server is that instead of the server containing the application files, as you've seen so far, you have an application with a main class (that is, a standalone Java app), that invokes the server capabilities from inside its code base. Overall, this offers a more simple and portable development model, and has rapidly become the norm. Spring Boot, for example, uses an embedded Tomcat instance running in dev mode.

Running an embedded server can net simplicity in terms of operations, since you are now dealing with just a single component (the app) instead of dealing with both the app and a server deployment. On the other hand, the setup where Tomcat runs as an independent host is still very common.

TomEE

It is possible to use more of the standard Java EE (or Jakarta EE) capabilities with Tomcat by adding those libraries to Tomcat itself, or to your application dependencies. Another option is the TomEE server. TomEE is the same Tomcat engine with additional Java enterprise support, including the popular JPA and CDI (Contexts and Dependency Injection) APIs. TomEE's spec is based on the Java EE web profile, so it gives you more than Tomcat but isn't a full-blown Java EE app server like WildFly or Glassfish.

High-availability and clustering
Tomcat supports high-availability and clustering. High availability is essentially the ability to *fail-over* to another instance of the server and re-create the session as though nothing had gone wrong. Clustering is the ability to create multiple versions of the same server to handle high-volume traffic.

Conclusion

Tomcat remains actively developed, keeping pace with change, and delivering a solid and reliable platform for deploying web apps. Both its continued popularity and choice as the default Java platform for many PaaS systems testify to its ongoing success.