**App server, Web server: What's the difference?**

A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.

## **The Web server**

A Web server handles the HTTP protocol. When the Web server receives an HTTP request, it responds with an HTTP response, such as sending back an HTML page. To process a request, a Web server may respond with a static HTML page or image, send a redirect, or delegate the dynamic response generation to some other program such as CGI scripts, JSPs (JavaServer Pages), servlets, ASPs (Active Server Pages), server-side JavaScripts, or some other server-side technology. Whatever their purpose, such server-side programs generate a response, most often in HTML, for viewing in a Web browser.

Understand that a Web server's delegation model is fairly simple. When a request comes into the Web server, the Web server simply passes the request to the program best able to handle it. The Web server doesn't provide any functionality beyond simply providing an environment in which the server-side program can execute and pass back the generated responses. The server-side program usually provides for itself such functions as transaction processing, database connectivity, and messaging.

While a Web server may not itself support transactions or database connection pooling, it may employ various strategies for fault tolerance and scalability such as load balancing, caching, and clustering—features oftentimes erroneously assigned as features reserved only for application servers.

## **The application server**

As for the application server, according to our definition, an application server exposes business logic to client applications through various protocols, possibly including HTTP. While a Web server mainly deals with sending HTML for display in a Web browser, an application server provides access to business logic for use by client application programs. The application program can use this logic just as it would call a method on an object (or a function in the procedural world).

Such application server clients can include GUIs (graphical user interface) running on a PC, a Web server, or even other application servers. The information traveling back and forth between an application server and its client is not restricted to simple display markup. Instead, the information is program logic. Since the logic takes the form of data and method calls and not static HTML, the client can employ the exposed business logic however it wants.

In most cases, the server exposes this business logic through a component API, such as the EJB (Enterprise JavaBean) component model found on J2EE (Java 2 Platform, Enterprise Edition) application servers. Moreover, the application server manages its own resources. Such gate-keeping duties include security, transaction processing, resource pooling, and messaging. Like a Web server, an application server may also employ various scalability and fault-tolerance techniques.

An application server can be defined as the code, container or [framework](https://javapipe.com/hosting/blog/best-java-web-frameworks/) that sits between the Operating System and the application. It is usually charged with the responsibility of providing a suite of services for the application.

An example

As an example, consider an online store that provides real-time pricing and availability information. Most likely, the site will provide a form with which you can choose a product. When you submit your query, the site performs a lookup and returns the results embedded within an HTML page. The site may implement this functionality in numerous ways. I'll show you one scenario that doesn't use an application server and another that does. Seeing how these scenarios differ will help you to see the application server's function.

#### **Scenario 1: Web server without an application server**

In the first scenario, a Web server alone provides the online store's functionality. The Web server takes your request, then passes it to a server-side program able to handle the request. The server-side program looks up the pricing information from a database or a flat file. Once retrieved, the server-side program uses the information to formulate the HTML response, then the Web server sends it back to your Web browser.

To summarize, a Web server simply processes HTTP requests by responding with HTML pages.

#### **Scenario 2: Web server with an application server**

Scenario 2 resembles Scenario 1 in that the Web server still delegates the response generation to a script. However, you can now put the business logic for the pricing lookup onto an application server. With that change, instead of the script knowing how to look up the data and formulate a response, the script can simply call the application server's lookup service. The script can then use the service's result when the script generates its HTML response.

In this scenario, the application server serves the business logic for looking up a product's pricing information. That functionality doesn't say anything about display or how the client must use the information. Instead, the client and application server send data back and forth. When a client calls the application server's lookup service, the service simply looks up the information and returns it to the client.

By separating the pricing logic from the HTML response-generating code, the pricing logic becomes far more reusable between applications. A second client, such as a cash register, could also call the same service as a clerk checks out a customer. In contrast, in Scenario 1 the pricing lookup service is not reusable because the information is embedded within the HTML page. To summarize, in Scenario 2's model, the Web server handles HTTP requests by replying with an HTML page while the application server serves application logic by processing pricing and availability requests.

# Is Tomcat An Application Server?

Nov 8, 2017 | [1 comment](https://javapipe.com/hosting/blog/tomcat-application-server/#respond)

There is no simple yes or no answer to this question as it depends on where you stand in the mix of things.

Tomcat is normally defined as a reference implementation of the Java Servlet and the Java Server Page (JSP) Specifications. It basically executes Java servlets and renders web pages which include JSP coding.

The truth though is that Tomcat is frequently used as an application server.

As a matter of fact, there are lightweight applications where Tomcat does shine as an app server and then there are more complex service oriented architectures where one would be better off using what can be termed as a full-fledged Java Platform Enterprise Edition ([Java EE](http://www.oracle.com/technetwork/java/javaee/overview/index.html)) application server. It thus follows that to clearly find out if Apache Tomcat is an application server or not, then we need to gauge it against the Java EE specifications.

Java EE compliance is, therefore, an important cog for any app server. A compliant Java EE application is therefore expected to support a number of features; these include but are not limited to the following:

* JNDI capabilities
* Enterprise JavaBeans (EJB) container and server
* Java Transaction API (JTA) framework
* Java Message Service (JMS) framework
* J2EE Connector Architecture
* EAR files
* JavaMail
* JavaServer Pages or JavaServer Faces
* JAAS (Java Authentication and Authorization Service)
* JAF (JavaBeans Activation Framework)
* JPA (Java Persistence API)
* JTA (Java Transaction API)
* SAAJ(The SOAP with Attachments API for Java)
* JDBC (Java Database Connectivity) Framework
* JAXB (The Java Architecture for XML Binding)
* JAX-RPC (The Java API for XML-based RPC)

Further to the above Java EE does define containers for client applications, EJB components, and servlets. These containers normally provide structure as well as functionality that not only facilitate the deployment, but also the persistence and execution of all supported components.

J2EE does also define a standard architecture that’s used to connect Java EE applications and application servers to heterogeneous Enterprise Information Systems such as non-Java legacy applications, database systems, mainframes and ERP systems.

Simply put, the JavaEE specifications are a set of rules, they basically contain the interface. Thus, any JavaEE servers which need to comply with specifications need to have the implementation of these interfaces.

It is at this point where the major difference between a JavaEE server and Tomcat come in quite clearly with most people who believe it is solely a web server coming in hard.

This is mainly because if you are deploying any enterprise application, meaning that you have some technology such as EJB, JPA etc which are part of the JavaEE set to the server which is compliant, then your lib will not need to contain any API implementation jars. However, if you are using Tomcat for deployment, then you will have to add these API implementation jars.

This does explain why many Java based web applications are normally deployed to environments that support a wide array of technologies that are found in a web container/server such as JavaServer Pages (JSP), JDBC and servlets. In such a scenario, a Tomcat application server comes in very handy when compared against a Java EE application server which may find these environmental constraints a tad difficult to adapt to.

On the other hand, a Java Web application is normally said to combine a subset of Java Enterprise application components and technologies such as:

* JDBC (Java Database Connectivity) Framework
* Java Servlets
* WAR files
* Java Servlets

In a typical Java EE web application, HTML clients are required to simply post a request to a server where the request is handled by a web container of the application server being used. At this point, it is the web container that’s charged with the responsibility of invoking the Servlet that’s been configured to handle the specific or exact context of the request.

Going by the above-mentioned details, it follows that strictly speaking Tomcat should be referred to as Tomcat web server or a JavaServer/Servlet container since there are certain conditions and services of a commercial JEE Application server that it doesn’t offer its users.

However, it does cover for these faults by including the most widely used services and supports add-ons as well as plug-ins which make server enhancement quite easy. The main advantage of this server, therefore, lies in its architecture which allows users to leave out what they don’t need, use what they need and install what they may be lacking.

Because of this, Tomcat is often used as an application server for strictly web-based applications even though it doesn’t include the entire suite of capabilities that a standard Java EE application would have on offer.

## **Inherent Advantages of Apache Tomcat**

Apache Tomcat is a widely used application with slightly more than a million downloads a month. Its overall success has been due to its lightweight, effective and efficient nature.

This is because as earlier mentioned; Tomcat does implement JSP and Servlet which is basically what many applications tend to depend on.

Because Apache Tomcat is not a Java EE engine, it is possible to keep it lightweight and upload any add-ons on a need basis. This not only makes it versatile and flexible but also ensures that it can continuously meet the needs of most web applications.

Since Apache Tomcat is an open source product, it does enjoy the benefits of well thought out and thoroughly tested updates as many well trained and highly experienced developers and other volunteers are usually given an opportunity to offer their input.

This does give the product a distinct advantage since the skills- set that it borrows from is quite wide and diverse. It is also important to mention that since it is open source, getting updates and using it with other products is not only seamless but also quite simple and straightforward.

Another main advantage of the product is the ease of installing and configuring the application. Typically, this can be done in less than twenty (20) minutes. It is also worth mentioning that deploying web applications to Tomcat is also very easy and simple.

## **Apache Tomcat Growth and Upgrades**

Over the years, the Apache community has continued to introduce new components into the Tomcat. These include components such as cluster, web application, and high availability. The cluster component has been added to help in the management of large applications.

The cluster is mainly used for load balancing and can be achieved through a number of techniques. Currently, one is required to have JDK version 1.5 or later so as to support clustering.

The high availability feature has been added so as to facilitate the scheduling of system upgrades which includes but is not limited to change requests and new releases that don’t affect live environments.

This is usually achieved by simply dispatching live traffic requests to a different server which is located to a different port as the main server is upgraded or replaced on the main port. This is an indispensable feature when handling user requests on applications that are considered high traffic.

The web application comes for both system and user based application enhancements, all these make it pretty easy to deploy across a variety of environments, be it mobile, desktop or even remote based. The component does also manage applications as well as sessions across the network.

It is worth mentioning that there are a whole lot of components which users can either opt to build on their own or download from one of the numerous mirrors and then proceed to upload into the server. It is also important to note that by adding some administrative services from other vendors, it is possible to meet IT operations requirements; giving users a complete suite of services at any given time.

## **Conclusion**

As earlier mentioned, defining whether Apache Tomcat is an application server is dependent on who you are talking to.

Those who develop rather lightweight and not too complex web based applications will vouch for it as an application server since they use it quite effectively and efficiently on a daily basis as an application server. The few add-ons that they need to deploy from time to time notwithstanding.

However, those developers who for one reason or the other have failed once or severally in trying to use Apache Tomcat in more complex web-based applications will be quick to dismiss it as a web server and will also rightfully remind anyone willing to hear that Apache Tomcat doesn’t meet the Java EE specifications which are a pre-requisite for any application server that wants to interface with the Java family.

Whichever the case, the truth though is that even though Tomcat cannot be technically defined as an application server, it is continuously and successfully being used as an application server for millions of mission-critical applications on a daily basis. The jury is still out there.