Tomcat deployment

Raúl Estrada

Octubre 2020

1. Overview

Apache Tomcat is one of the most popular web servers in the Java community. It ships as a **servlet container** capable of serving Web ARchives with the WAR extension.

It provides a management dashboard from which you can deploy a new web application, or undeploy an existing one without having to restart the container. This is especially useful in production environments.

In this article, we will do a quick overview of Tomcat and then cover various approaches to deploying a WAR file.

2. Tomcat Structure

Before we begin, we should familiarize ourselves with some terminology and environment variables.

2.1. Environment Variables

If you have worked with Tomcat before, these will be very familiar to you:

$CATALINA_HOME

This variable points to the directory where our server is installed.

$CATALINA_BASE

This variable points to the directory of a particular instance of Tomcat, you may have multiple instances installed. If this variable is not set explicitly, then it will be assigned the same value as *$CATALINA_HOME*.

Web applications are deployed under the *$CATALINA_HOME\webapps* directory.

2.2. Terminology

Document root. Refers to the top-level directory of a web application, where all the application resources are located like JSP files, HTML pages, Java classes, and images.

Context path. Refers to the location which is relative to the server's address and represents the name of the web application.

For example, if our web application is put under the *$CATALINA_HOME\webapps\myapp* directory, it will be accessed by the URL *http://localhost/myapp*, and its context path will be */myapp*.

WAR. Is the extension of a file that packages a web application directory hierarchy in ZIP format and is short for Web Archive. Java web applications are usually packaged as WAR files for deployment. These files can be created on the command line or with an IDE like Eclipse.

After deploying our WAR file, Tomcat unpacks it and stores all project files in the *webapps* directory in a new directory named after the project.

3. Tomcat Setup

The Tomcat Apache web server is free software that can be downloaded from their website. It is required that there is a JDK available on the user's machine and that the *JAVA_HOME* environment variable is set correctly.

3.1. Start Tomcat

We can start the Tomcat server by simply running the *startup* script located at *$CATALINA_HOME\bin\startup*. There is a *.bat* and a *.sh* in every installation.

Choose the appropriate option depending on whether you are using a Windows or Unix based operating system.

3.2. Configure Roles

During the deployment phase, we'll have some options, one of which is to use Tomcat's management dashboard. To access this dashboard, we must have an admin user configured with the appropriate roles.

To have access to the dashboard the admin user needs the *manager-gui* role. Later, we will need to deploy a WAR file using Maven, for this, we need the *manager-script* role too.

Let's make these changes in *$CATALINA_HOME\conf\tomcat-users*:

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="password" roles="manager-gui, manager-script"/>
```

More details about the different Tomcat roles can be found by following this official link.

3.3. Set Directory Permissions

Finally, ensure that there is read/write permission on the Tomcat installation directory.

3.4. Test Installation

To test that Tomcat is setup properly run the startup script (*startup.bat/ startup.sh*), if no errors are displayed on the console we can double-check by visiting *http://localhost:8080*.

If you see the Tomcat landing page, then we have installed the server correctly.

3.5. Resolve Port Conflict

By default, Tomcat is set to listen to connections on port *8080*. If there is another application that is already bound to this port, the startup console will let us know.

To change the port, we can edit the server configuration file *server.xml* located at *$CATALINA_HOME\conf\server.xml*. By default, the connector configuration is as follows:

```
<Connector port="8080" protocol="HTTP/1.1"
  connectionTimeout="20000" redirectPort="8443" />
```

For instance, if we want to change our port to *8081*, then we will have to change the connector's port attribute like so:

```
<Connector port="8081" protocol="HTTP/1.1"
  connectionTimeout="20000" redirectPort="8443" />
```

Sometimes, the port we have chosen is not open by default, in this case, we will need to open this port with the appropriate commands in the Unix kernel or creating the appropriate firewall rules in Windows, how this is done is beyond the scope of this article.

4. Deploy From Maven

If we want to use Maven for deploying our web archives, we must configure Tomcat as a server in Maven's *settings.xml* file.

There are two locations where the *settings.xml* file may be found:

- The Maven install: *${maven.home}/conf/settings.xml*
- A user's install: *${user.home}/.m2/settings.xml*

Once you have found it add Tomcat as follows:

```
<server>
    <id>TomcatServer</id>
    <username>admin</username>
    <password>password</password>
</server>
```

We will now need to create a basic web application from Maven to test the deployment. Let's navigate to where we would like to create the application.

Run this command on the console to create a new Java web application:

```
mvn archetype:generate -DgroupId=com.baeldung -DartifactId=tomcat-war-deployment
  -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

This will create a complete web application in the directory *tomcat-war-deployment* which, if we deploy now and access via the browser, prints *hello world!*.

But before we do that we need to make one change to enable Maven deployment. So head over to the *pom.xml* and add this plugin:

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <url>http://localhost:8080/manager/text</url>
        <server>TomcatServer</server>
        <path>/myapp</path>
    </configuration>
</plugin>
```

Note that we are using the Tomcat 7 plugin because it works for both versions 7 and 8 without any special changes.

The configuration *url* is the url to which we are sending our deployment, Tomcat will know what to do with it. The *server* element is the name of the server instance that Maven recognizes. Finally, the *path* element defines the **context path** of our deployment.

This means that if our deployment succeeds, we will access the web application by hitting *http://localhost:8080/myapp*.

Now we can run the following commands from Maven.

To deploy the web app:

```
mvn tomcat7:deploy
```

To undeploy it:

```
mvn tomcat7:undeploy
```

To redeploy after making changes:

```
mvn tomcat7:redeploy
```

5. Deploy With Cargo Plugin

Cargo is a versatile library that allows us to manipulate the various type of application containers in a standard way.

5.1. Cargo Deployment Setup

In this section, we will look at how to use Cargo's Maven plugin to deploy a WAR to Tomcat, in this case, we will deploy it to a version 7 instance.

To get a firm grip on the whole process, we will start from scratch by creating a new Java web application from the command line:

```
mvn archetype:generate -DgroupId=com.baeldung -DartifactId=cargo-deploy
  -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

This will create a complete Java web application in the *cargo-deploy* directory. If we build, deploy and load this application as is, it will print *Hello World!* in the browser.

Unlike the Tomcat7 Maven plugin, the Cargo Maven plugin requires that this file is present.

Since our web application does not contain any servlets, our *web.xml* file will be very basic. So navigate to the *WEB-INF* folder of our newly created project and create a *web.xml* file with the following content:

Since our web application does not contain any servlets, our *web.xml* file will be very basic. So navigate to the *WEB-INF* folder of our newly created project and create a *web.xml* file with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0">

    <display-name>cargo-deploy</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

To enable Maven to recognize Cargo's commands without typing the fully qualified name, we need to add the Cargo Maven plugin to a plugin group in Maven's *settings.xml*.

As an immediate child of the root *<settings></settings>* element, add this:

```
<pluginGroups>
    <pluginGroup>org.codehaus.cargo</pluginGroup>
</pluginGroups>
```

5.2. Local Deploy

In this subsection, we will edit our *pom.xml* to suit our new deployment requirements.

Add the plugin as follows:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.cargo</groupId>
            <artifactId>cargo-maven2-plugin</artifactId>
            <version>1.5.0</version>
            <configuration>
                <container>
                    <containerId>tomcat7x</containerId>
                    <type>installed</type>
                    <home>Insert absolute path to tomcat 7 installation</home>
                </container>
                <configuration>
                    <type>existing</type>
                    <home>Insert absolute path to tomcat 7 installation</home>
                </configuration>
            </configuration>
        </plugin>
    </plugins>
</build>
```

The latest version, at the time of writing, is *1.5.0*. However, the latest version can always be found here.

Notice that we explicitly define the packaging as a WAR, without this, our build will fail. In the plugins section, we then add the cargo maven2 plugin. Additionally, **we add a configuration section where we tell Maven that we are using a Tomcat container and also an existing installation.**

By setting the container type to *installed*, we tell Maven that we have an instance installed on the machine and we provide the absolute URL to this installation.

By setting the configuration type to *existing*, we tell Tomcat that we have an existing setup that we are using and no further configuration is required.

The alternative would be to tell cargo to download and setup the version specified by providing a URL. However, our focus is on WAR deployment.

It's worth noting that whether we are using Maven 2.x or Maven 3.x, the cargo maven2 plugin works for both.

We can now install our application by executing:

```
mvn install
```

and deploying it by doing:

```
mvn cargo:deploy
```

If all goes well we should be able to run our web application by loading *http://localhost:8080/cargo-deploy*.

5.3. Remote Deploy

To do a remote deploy, we only need to change the configuration section of our *pom.xml*. Remote deploy means that we do not have a local installation of Tomcat but have access to the manager dashboard on a remote server.

So let's change the *pom.xml* so that the configuration section looks like this:

```
<configuration>
    <container>
        <containerId>tomcat8x</containerId>
        <type>remote</type>
    </container>
    <configuration>
        <type>runtime</type>
        <properties>
            <cargo.remote.username>admin</cargo.remote.username>
            <cargo.remote.password>admin</cargo.remote.password>
            <cargo.tomcat.manager.url>http://localhost:8080/manager/text
              </cargo.tomcat.manager.url>
        </properties>
    </configuration>
</configuration>
```

This time, we change the container type from *installed* to *remote* and the configuration type from *existing* to *runtime*. Finally, we add authentication and remote URL properties to the configuration.

Ensure that the roles and users are already present in *$CATALINA_HOME/conf/tomcat-users.xml* just as before.

If you are editing the same project for *remote* deployment, first un-deploy the existing WAR:

```
mvn cargo:undeploy
```

If you are editing the same project for *remote* deployment, first un-deploy the existing WAR:

```
mvn cargo:undeploy
```

clean the project:

```
mvn clean
```

install it:

```
mvn install
```

finally, deploy it:

```
mvn cargo:deploy
```

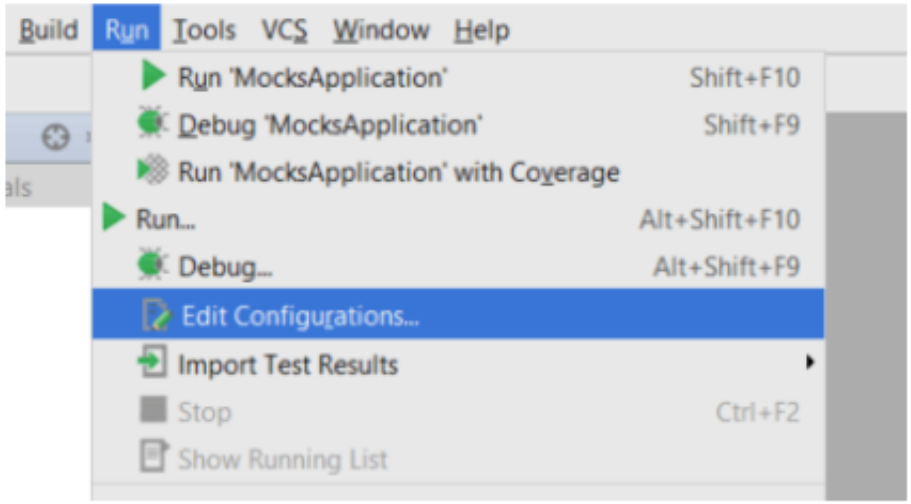
That's it.

7. Deploy From IntelliJ IDEA

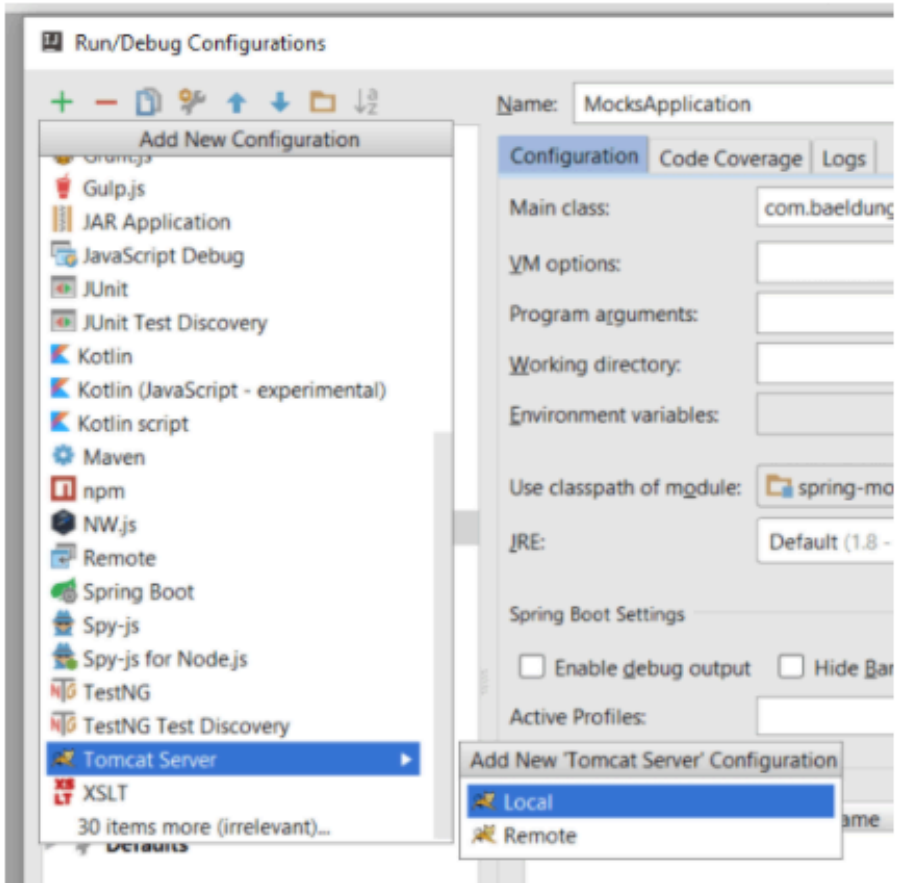
To deploy a web application to Tomcat, it must exist and have already been downloaded and installed.

7.1. Local Configuration

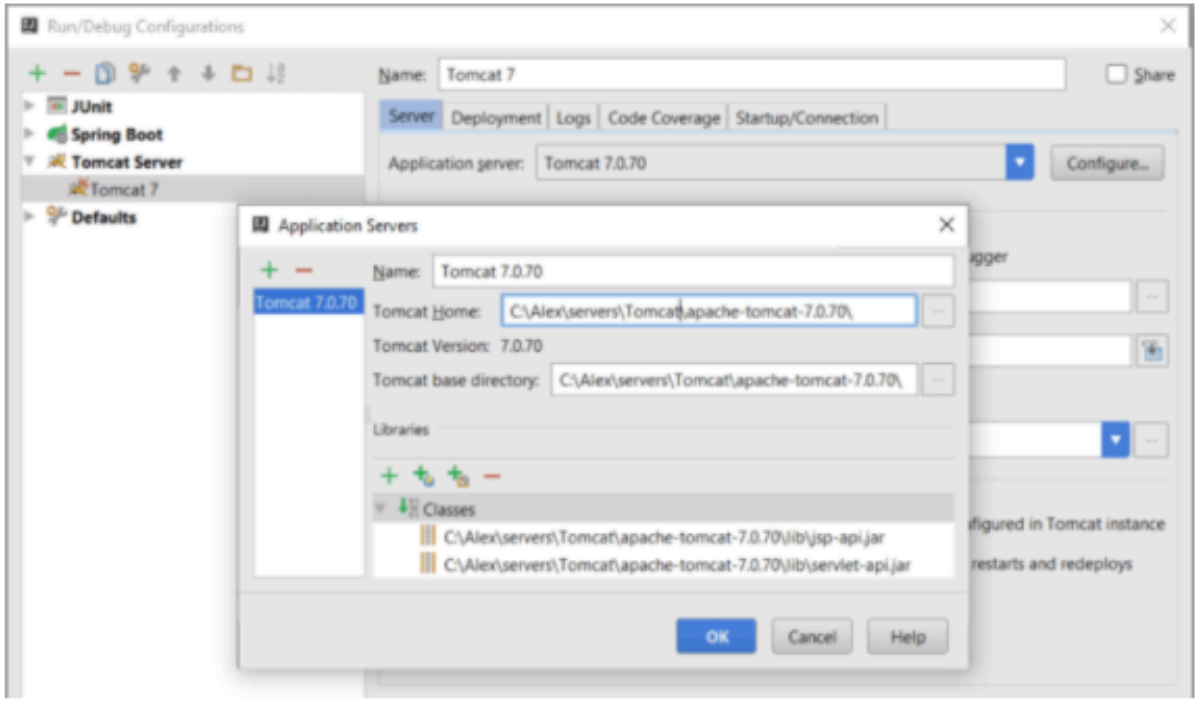
Open the *Run* menu and click the *Edit Configurations* options.



In the panel on the left search for *Tomcat Server*, if it is not there click the + sign in the menu, search for *Tomcat* and select *Local*. In the name field put *Tomcat 7/8* (depending on your version).

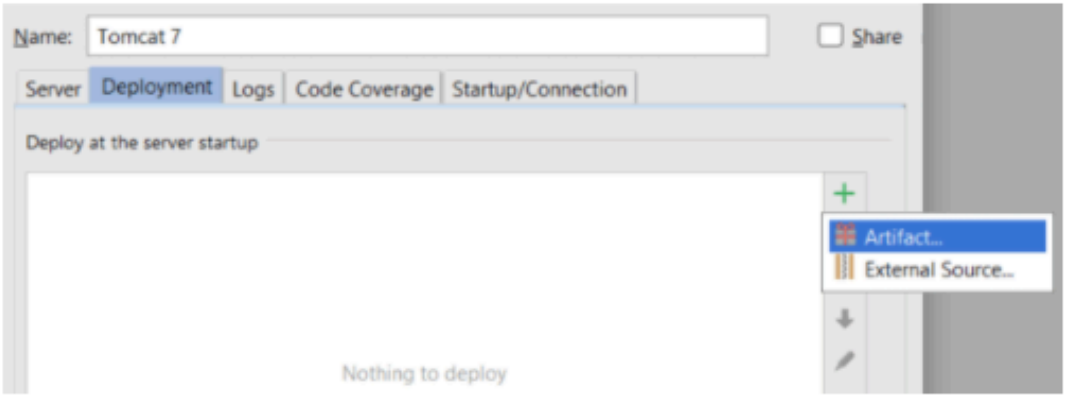


Click the *Configure...* button and in *Tomcat Home* field navigate to the home location of your installation and select it.



Optionally, set the *Startup* page to be *http://localhost:8080/* and *HTTP port: 8080*, change the port as appropriate.

Go to the *Deployment* tab and click on the + symbol, select artifact you want to add to the server and click OK



7.2. Remote Configuration

Follow the same instructions as for local Tomcat configurations, but in the server tab, you must enter the remote location of the installation.

8. Deploy by Copying Archive

We have seen how to export a WAR from Eclipse. One of the things we can do is to deploy it by simply dropping it into the *$CATALINA_HOME\webapps* directory of any Tomcat instance. If the instance is running, the deployment will start instantly as Tomcat unpacks the archive and configures its context path.

If the instance is not running, then the server will deploy the project the next time it is started.

9. Deploy From Tomcat Manager

Assuming we already have our WAR file to hand and would like to deploy it using the management dashboard. You can access the manager dashboard by visiting: *http://localhost:8080/manager*.

The dashboard has five different sections: *Manager*, *Applications*, *Deploy*, *Diagnostics*, and *Server Information.* If you go to the *Deploy* section, you will find two subsections.

9.1. Deploy Directory or WAR File Located on Server

If the WAR file is located on the server where the Tomcat instance is running, then we can fill the required *Context Path* field preceded by a forward slash "*/*".

Let's say we would like our web application to be accessed from the browser with the URL *http://localhost:8080/myapp*, then our context path field will have */myapp.*

We skip the *XML Configuration file URL* field and head over to the *WAR or Directory URL* field. Here we enter the absolute URL to the Web ARchive file as it appears on our server. Let's say our file's location is *C:/apps/myapp.war*, then we enter this location. Don't forget the WAR extension.

After that, we can click *deploy* button. The page will reload, and we should see the message:

```
OK - Deployed application at context path /myapp
```

at the top of the page.

Additionally, our application should also appear in the *Applications* section of the page.

9.2. WAR File to Deploy

Just click the *choose file* button, navigate to the location of the WAR file and select it, then click the *deploy* button.

In both situations, if all goes well, the Tomcat console will inform us that the deployment has been successful with a message like the following:

```
INFO: Deployment of web application archive \path\to\deployed_war has finished in 4,833 ms
```

10. Conclusion

In this writeup, we focused on deploying a WAR into a Tomcat server.