

Step-by-step Jenkins Tomcat deploy of a WAR file

[Step-by-step Maven Tomcat WAR file deploy example](#)

The final stage of a continuous integration pipeline is often the deployment of a packaged WAR file to an application server, such as WebSphere Liberty, WildFly, Jetty or **Apache Tomcat**. In this Jenkins Tomcat deploy tutorial, we will take you through the various steps required to get a [Jenkins](#) pipeline to deploy to Tomcat after a build with a WAR file.

Jenkins Tomcat deploy prerequisites

The following pieces of software are required to follow this example of a Jenkins deployment of a WAR file to Tomcat:

- a Java web application that Jenkins can deploy to Tomcat;
- a Git source code management tool installation;
- a Java 8 or newer installation of the JDK;
- a build tool -- this tutorial [uses Maven](#), but any build tool, such as Ivy, ANT or Gradle, that can package a Java application into a WAR file will do; and
- a Jenkins [CI](#) tool installation.

A WAR file for Jenkins to deploy

If you need a web application for Jenkins to deploy to Tomcat, feel free to clone my rock-paper-scissors Java web app from GitHub:

```
git clone https://github.com/cameronmcnz/rock-paper-scissors.git
cd rock*
git checkout patch-1
```

Pay special attention to the last command, where you will switch to the patch-1 branch. The master branch creates [a JAR file](#), with Tomcat embedded within. But what we need is the patch-1 branch, which creates a WAR file that can be deployed to Tomcat by Jenkins.

Step 1: Add to Tomcat a user with deployment rights

For a successful Jenkins Tomcat deploy of a WAR file, you must add a new user to Tomcat with `manager-script` rights. You can do this with an edit of the `tomcat-users.xml` file, which can be found in Tomcat's `conf` directory.

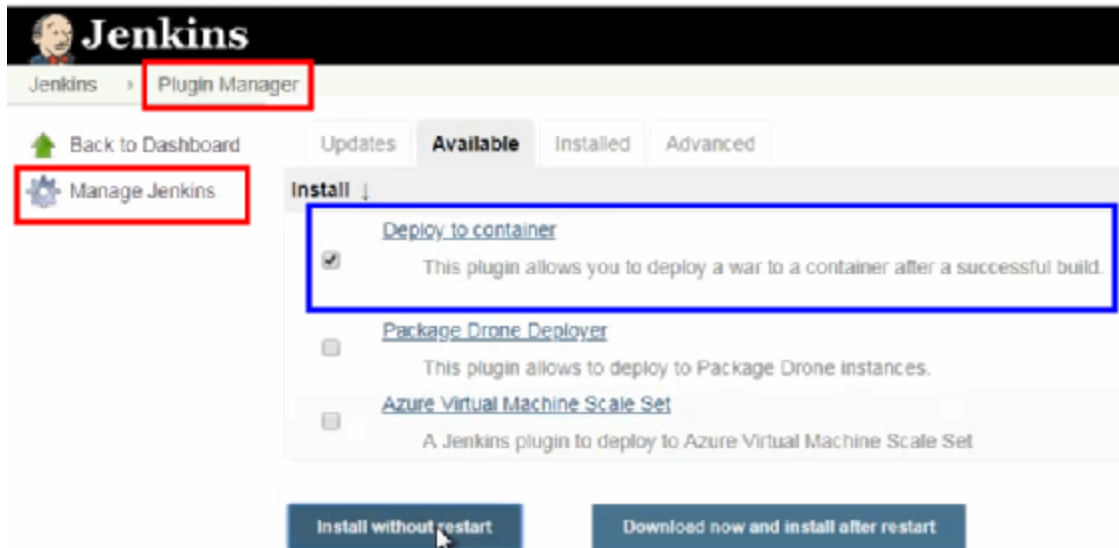
```
<!-- User to deploy WAR file from Jenkins to Tomcat -->
<user username="war-deployer" password="jenkins-tomcat-deploy"
      roles="manager-script" />
```

After you edit the `tomcat-users.xml` file, it's a good idea to bounce the Tomcat server to confirm the changes have taken effect.

Step 2: Add the 'Deploy WAR/EAR to a container Jenkins' plugin

Out of the box, there are no built-in features that perform a Jenkins WAR file deployment to Tomcat. That means a Jenkins Tomcat deploy plugin must be installed in the CI tool to make a deployment happen.

The most popular Jenkins Tomcat deployment plugin is named *Deploy to container*, which can be installed through the Plugin Manager tab under the "Manage Jenkins" section of the tool.



Install the Deploy to container Jenkins Tomcat deploy plugin.

Step 3: The Jenkins build job

With the Jenkins Tomcat deployment plugin installed, it's time to create a new Jenkins build job that can build an application and deploy a packaged WAR file to Tomcat.

Step 3A: Create a Jenkins freestyle project

The Jenkins build job we need to create will be named `deploy-war-from-jenkins-to-tomcat`, and it will be a [freestyle project type](#).



This freestyle Jenkins job will build a WAR and deploy it to Tomcat.

Step 3B: Configure standard build job properties

The Jenkins build job will be configured with the following properties:

JDK: java8

Git Repository URL: <https://github.com/cameronmcnz/rock-paper-scissors.git>

Git branch specifier: */patch-1

Maven Goals: clean install

The screenshot shows the Jenkins configuration interface. The 'Source Code Management' section has 'Git' selected. The 'Repository URL' is set to 'https://github.com/cameronmcnz/rock-paper-scissors.git'. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' set to '*/patch-1'. The 'Build' section has 'Invoke top-level Maven targets' selected. The 'Maven Version' is set to 'maven' and the 'Goals' are set to 'clean install'.

Basic

Jenkins build job settings for a Tomcat WAR deployment.

Step 3C: The Jenkins Tomcat deploy plugin post-build action

The screenshot shows the 'Build' section of the Jenkins configuration. A list of post-build actions is displayed, including 'Invoke top-level Maven targets', 'Aggregate downstream test results', 'Archive the artifacts', 'Build other projects', 'Publish JUnit test result report', 'Publish Javadoc', 'Record fingerprints of files to track usage', 'Use publishers from another project', 'Git Publisher', 'Deploy war/ear to a container', 'Email Notification', and 'Trigger parameterized build on other projects'. The 'Deploy war/ear to a container' action is highlighted with a red box. Below this list, there is a button labeled 'Add post-build action' with a dropdown arrow, which is also highlighted with a blue box.

The Jenkins deploy war/ear to a container post-build action.

After a build, the final step of a Jenkins pipeline deploy to Tomcat is to use the *Deploy to container* plugin in a post-build action.

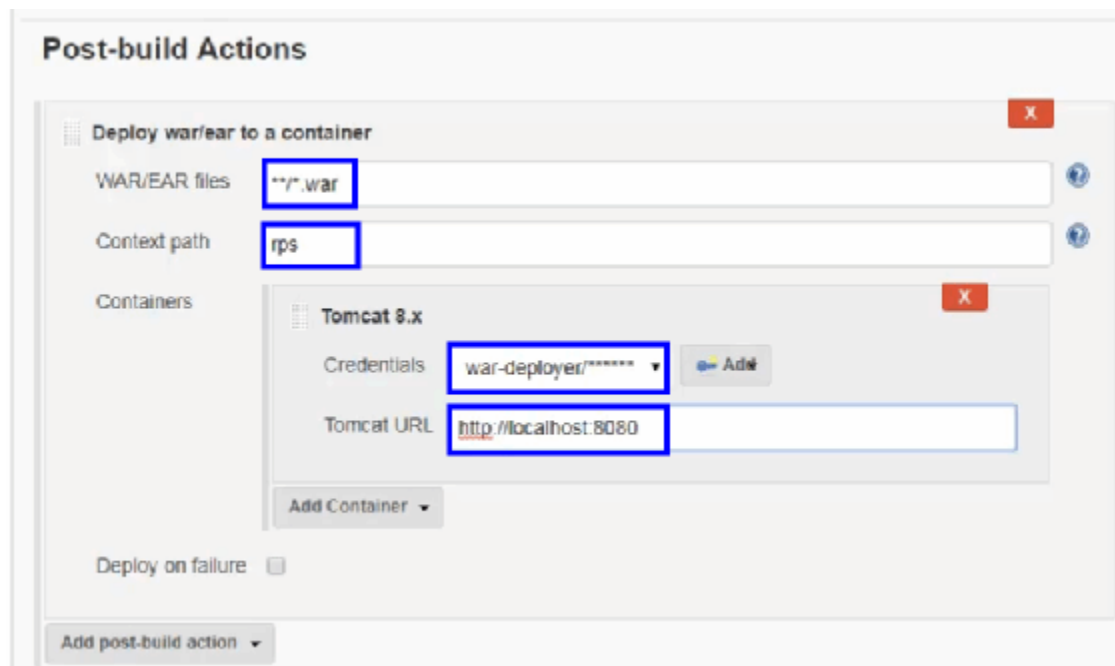
Three of the four settings used by the *Deploy WAR/EAR to a container* plugin can be typed in directly:

WAR/EAR files: `**/*.war`

Context path: `rps`

Containers: Tomcat 8.x

Tomcat URL: `http://localhost:8080`



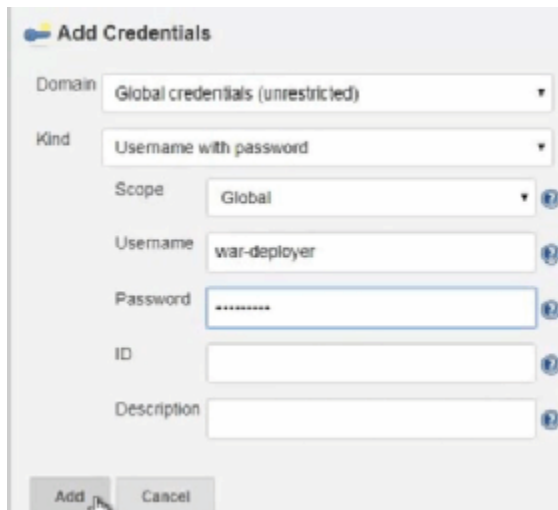
The screenshot shows the 'Post-build Actions' configuration in Jenkins. The 'Deploy war/ear to a container' plugin is selected. The configuration fields are as follows:

- WAR/EAR files:** `**/*.war`
- Context path:** `rps`
- Containers:** A list containing 'Tomcat 8.x'. Below this list, the 'Tomcat 8.x' container is expanded, showing:
 - Credentials:** `war-deployer/*****`
 - Tomcat URL:** `http://localhost:8080`

At the bottom, there is a checkbox for 'Deploy on failure' which is unchecked, and a button for 'Add post-build action'.

Add in the Jenkins post-build action for Tomcat.

To configure the credentials, you must click the Add button next to the empty entry field and create a new Jenkins credentials object:

The image shows the 'Add Credentials' dialog box in Jenkins. It has a title bar with a key icon and the text 'Add Credentials'. Below the title bar, there are several fields: 'Domain' is a dropdown menu set to 'Global credentials (unrestricted)'; 'Kind' is a dropdown menu set to 'Username with password'; 'Scope' is a dropdown menu set to 'Global'; 'Username' is a text input field containing 'war-deployer'; 'Password' is a password input field with masked characters; 'ID' is an empty text input field; and 'Description' is an empty text input field. Each of the last four fields has a small blue help icon to its right. At the bottom left, there are two buttons: 'Add' and 'Cancel'. A mouse cursor is pointing at the 'Add' button.

Jenkins credential used by the Deploy war/ear to a container plugin.

The username and password need to match what was entered into the tomcat-users.xml file in an earlier step:

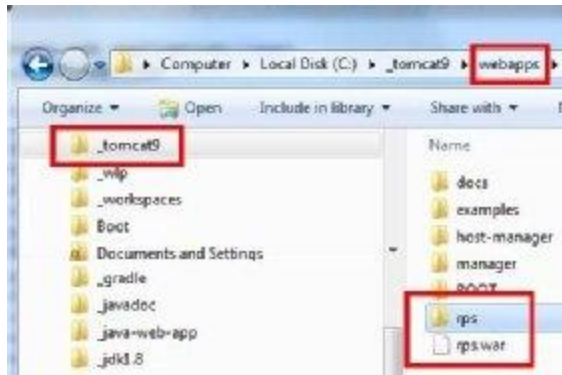
```
Username: war-deployer
```

```
Password: jenkins-tomcat-deploy
```

Step 3D: Run the Jenkins build job

Now that you have specified all of the configurations, the Jenkins build job can be saved and run.

When the build job finishes, the Jenkins Tomcat deploy of a WAR file will have also completed, and a file named `rps.war` will be visible in the `webapps` directory of Tomcat.



A WAR file deployed to Tomcat through Jenkins.

Step 4: Test the deployed WAR file

With the WAR file deployed, test the application by running Tomcat and pointing your browser to the following URL:

<http://localhost:8080/rps/index.html>

Jenkins Tomcat WAR deployment summary

To recap, here is a summary of the steps required to perform a Jenkins Tomcat WAR file deployment:

1. Add a user with WAR deployment rights to the `tomcat-users.xml`.
2. Add the *Deploy to container* Jenkins Tomcat plugin.
3. Create a Jenkins build job with a *Deploy to container* post-build action.
4. Run the Jenkins build job.
5. Test to ensure the Jenkins deployment of the WAR file to Tomcat was successful.

And that's how easy it is to get Jenkins to deploy WAR files to Tomcat.