Deploying Tomcat-based Java Web Applications with Webapp Runner

Last Updated: 28 October 2013

Table of Contents

Webapp Runner allows you to launch an application in a Tomcat container on any computer that has a JRE installed. No previous steps to install Tomcat are required when using Webapp Runner. It's just a jar file that can be executed and configured using the `java` command.

This article will walk you through building an application that launches with Webapp Runner and deploying that application to Heroku.

Follow each step to build an app from scratch, or skip to the end get the source for this article. You can also use almost any existing Maven webapp project.

If you have questions about Java on Heroku, consider discussing them in the Java on Heroku forums.

Prerequisites

- Basic Java knowledge, including an installed version of the JVM and Maven.
- Basic Git knowledge, including an installed version of Git.

How does Webapp Runner work?

When using Webapp Runner you'll launch your application locally and on Heroku with a command like this:

```
$ java -jar webapp-runner.jar application.war
deploying app from: /Users/johnsimone/dev/gitrepos/devcenter-webapp-runner/target/webappRunnerSample.war
Feb 14, 2012 5:21:44 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-bio-8080"]
Feb 14, 2012 5:21:44 PM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Tomcat
Feb 14, 2012 5:21:44 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/7.0.40
Feb 14, 2012 5:21:44 PM org.apache.catalina.startup.ContextConfig webConfig
INFO: No global web.xml found
Feb 14, 2012 5:21:44 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
```

Webapp Runner will then launch a Tomcat instance with the given war deployed to it. This takes advantage of Tomcat's embedded APIs and is similar to an option that Jetty offers: Jetty Runner. Webapp Runner is open source so you can view or contribute to the source code.

Create an application if you don't already have one

```
$ mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp
...
[INFO] Generating project in Interactive mode
Define value for property 'groupId': : com.example
Define value for property 'artifactId': : helloworld
```

(you can pick any groupId or artifactId). You now have a complete Java web app in the `helloworld` directory.

Configure Maven to download Webapp Runner

Although not necessary for using Webapp Runner it's a good idea to have your build tool download Webapp Runner for you since your application will need it to run. You could, of course, just download Webapp Runner and use it to launch your application without doing this. However having all of your dependencies defined in your build descriptor is important for application portability and repeatability of deployment. In this case we're using Maven so we'll use the dependency plugin to download the jar. Add the following plugin configuration to your pom.xml:

```
<build>
    ...
    <plugins>
        ...
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>2.3</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals><goal>copy</goal></goals>
                    <configuration>
                        <artifactItems>
                            <artifactItem>
                                <groupId>com.github.jsimone</groupId>
                                <artifactId>webapp-runner</artifactId>
                                <version>7.0.40.0</version>
```

```
                                <destFileName>webapp-runner.jar</destFileName>
                            </artifactItem>
                        </artifactItems>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Run your application

To build your application simply run:

```
$ mvn package
```

And then run your app using the java command:

```
$ java -jar target/dependency/webapp-runner.jar target/*.war
```

That's it. Your application should start up on port 8080.

Note: if you need your WAR file to be expanded before launching you can add the `--expand-war` option before `target/*.war`

Deploy your application to Heroku

Create a Procfile

You declare how you want your application executed in `Procfile` in the project root. Create this file with a single line:

```
web:    java $JAVA_OPTS -jar target/dependency/webapp-runner.jar --port $PORT target/*.war
```

Deploy to Heroku

Commit your changes to Git:

```
$ git init
$ git add .
$ git commit -m "Ready to deploy"
```

Create the app:

```
$ heroku create
Creating high-lightning-129... done, stack is cedar
http://high-lightning-129.herokuapp.com/ | git@heroku.com:high-lightning-129.git
Git remote heroku added
```

Deploy your code:

```
$ git push heroku master
Counting objects: 227, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (117/117), done.
Writing objects: 100% (227/227), 101.06 KiB, done.
Total 227 (delta 99), reused 220 (delta 98)

-----> Heroku receiving push
-----> Java app detected
-----> Installing Maven 3.0.3..... done
-----> Installing settings.xml..... done
-----> executing .maven/bin/mvn -B -Duser.home=/tmp/build_1jems2so86ck4 -s .m2/settings.xml -DskipTests=true clean install
       [INFO] Scanning for projects...
       [INFO]
       [INFO] ------------------------------------------------------------------------
       [INFO] Building webappRunnerSample Maven Webapp 1.0-SNAPSHOT
       [INFO] ------------------------------------------------------------------------
       ...
       [INFO] ------------------------------------------------------------------------
       [INFO] BUILD SUCCESS
       [INFO] ------------------------------------------------------------------------
       [INFO] Total time: 36.612s
       [INFO] Finished at: Tue Aug 30 04:03:02 UTC 2011
       [INFO] Final Memory: 19M/287M
       [INFO] ------------------------------------------------------------------------
-----> Discovering process types
       Procfile declares types -> web
-----> Compiled slug size is 4.5MB
-----> Launching... done, v5
       http://pure-window-800.herokuapp.com deployed to Heroku
```

Congratulations! Your web app should now be up and running on Heroku. Open it in your browser with:

```
$ heroku open
```

Use distributed HTTP sessions with Memcache

Explicitly storing session state in a database or other backend data store is a more scalable alternative to using distributed HTTP sessions. To find out if distributed HTTP sessions are the best design choice for your application see the article on distributed HTTP sessions.

Webapp runner supports the memcached-session-manager for Tomcat. In order to enable memcache backed sessions you need to make the configuration for

your memcache instance available through environment variables and then enable the sesssion manager.

Make memcache configuration information available

The Heroku Memcachier Add On will set the required environment variables for you. Once you have an existing app get the add on by running:

```
$ heroku addons:add memcachier:dev
```

Note: you may have to verify your account before you can add this add on.

When running locally you can either set up a local install of memcache or connect to the remote memcache service provisioned for you by the Heroku add on.

When used with webapp runner the memcache backed session manager looks for 3 environment variables: MEMCACHIER_SERVERS, MEMCACHIER_USERNAME, MEMCACHIER_PASSWORD. You can set these to point to a local memcache install or connect to the remote memcache service provisioned for you by the Heroku add on by running `heroku config` and copying the values into local environment variables.

Enable memcached-session-manager

To enable memcache backed sessions with webapp runner you include the following flag: `--session-store memcache`

So if launching locally your command would now look like:

```
$ java -jar target/dependency/webapp-runner.jar --session-store memcache target/*.war
```

Or your Procfile would look like:

```
web:    java $JAVA_OPTS -jar target/dependency/webapp-runner.jar --port $PORT --session-store memcache target/*.war
```

Clone the source

If you want to skip the creation steps you can clone the finished sample (without memcache backed session):

```
$ git clone git@github.com:heroku/devcenter-webapp-runner.git
```

Clone as a Heroku app

One of the tempalates available at java.heroku.com uses Webapp Runner with Spring MVC. You can clone this template into your Heroku account by going here.