deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

Lab 2 Java at the server-side and the role of application containers

Updated: 2020-10-20.

Introduction to the lab

Learning outcomes

— Deploy a java web application into an application container (servlets).

— Deploy web applications using embedded containers.

— Start a simple web application with Spring Boot and Spring Initializr.

Submission

Each student should submit in Moodle evidence of the work in class. A (single) zip file with one folder per each section (📁lab2.1, 📁lab2.2,...) should be prepared; in the root, there should be a file "readme.md" with the author ID.

You should use the **Readme.md as a notepad/logbook**; use this file both to take notes that will help you study later, and to provide a log of the work done.

The items required to appear in the delivery are marked with 📑 (but you can add more).

2.1 Server-side programming with servlets

Java Servlet is the foundation web specification in the Java Enterprise environment. A Servlet is a Java class that runs at the server, handles (client) requests, processes them, and reply with a response. A servlet must be deployed into a (multithreaded) **Servlet Container** in order to become usable. Containers[1] handle multiple requests concurrently. *Servlet* is a generic interface and the *HttpServlet* is an extension of that interface (the most used type of Servlets).

When an application running in a web server receives a request, the Server hands the request to the Servlet Container which in turn passes it to the target Servlet.

Servlet Container is a part of the usual set of services that we can find in Java Application Server.

a) Prepare an application server to run your web projects.

There are several production-ready application servers you can choose from (e.g.: Apache Tomcat, RedHat WildFly, Payara, IBM Websphere,...).

For this exercise, consider using the Apache Tomcat, which is open source. Just download (Tomcat v9, core zip) and expand to a local folder[2].

Run the application server (use the startup script inside <path to Tomcat>/bin folder).

Confirm that Tomcat server is running. Here are 3 alternatives to do it:

```
i)      $ curl -I 127.0.0.1:8080
ii)     Try in the browser: http://localhost:8080
iii)    Observe the server log: $ tail logs/catalina.out
```

---

[1] Servlet Containers and Docker Containers are very different concepts. You use Docker Containers to deploy virtualized runtimes, for any kind of services; Servlet Containers provide a runtime to execute server-side, web-related Java code.

[2] For a production environment, you would install Tomcat as a system service/daemon or, even better, as a "dockerized" service. In this case, we will just start and stop from the command line, in the installation folder.

b) Tomcat includes a management environment ("Manager app") in which you can control the server, including deploying and un-deploying the applications you develop (http://localhost:8080/manager).

First, you need to register appropriate roles in conf/tomcat-users.xml (see users configuration sample in step #6 here to create a user "admin"). Restart tomcat.

From the web Manager environment, explore the Examples application already installed in Tomcat (in particular: Examples --> Servlet --> **Request Parameters**). Check the related source code; note the superclass in "extends".

c) Create a maven-based **web application** project. For that, you need to use a suitable Maven archetype. You may use the default Apache's archetype or the suggestedcodehaus':

    archetypeGroupId=org.codehaus.mojo.archetypes
    archetypeArtifactId=webapp-javaee7
    archetypeVersion=1.1

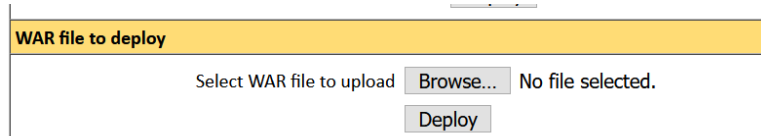(You may need to add this additional archetype to your IDE.)

d) This base project already includes an index page so you can test the web project.

Build the project and ensure there are no errors (mvn install).

Confirm that you have a **.war** file in <project folder>/target. This is your application packaged as a Web ARchive; you may check its contents in a regular Zip tool.

e) Deploy the packed application (.war) into the application server.

Use the Tomcat management interface (Tomcat manager application) to deploy a .war file (http://localhost:8080/manager).

f) Confirm that your application was successfully deployer in Tomcat: you should get the page with "Hello World!" displayed in the browser (e.g.: http://localhost:8080/*your-web-app-name*/)

g) Also inspect the application **server log** (the Tomcat log) and look for evidence that the deployment was successful.

h) Deploying using the Tomcat Manager page has some disadvantages: it is not "connected" with the IDE and is specific to Tomcat. The productive alternative is to use the **IDE integrated deployment support**.

Be sure to configure the Tomcat server integration in your IDE, e.g., for IntelliJ[3], for VS Code, for Eclipse[4].

In this example, the artifact to deploy is the <your project>.war file.

Important: when deploying from IDE for development purposes, Tomcat should not be started from outside the IDE (it will be started/stopped by the IDE as needed).

i) Add a basic servlet to your project that takes the name of the user, passed as a parameter in the HTTP request and prints a customized message.

---

[3] If you are using the IntelliJ Ultimate edition, the plugin "Tomcat and TomEE" is already installed.
[4] For Java Enterprise, the proper edition of Eclipse is "Eclipse IDE for Enterprise Java Developers".

You may adapt these instructions (from section "Develop Servlet with @WebServlet Annotation" to section "Handling Servlet Request and Response").

Note: in older versions of the Servlet Container specifications, the developer was expected to write a **web.xml** file with the configuration descriptors, including the mapping of Servlet classes and URL paths. The current implementations, however, use annotations (check for the presence of @WebServlet annotation), which makes the use of web.xml not required (in most of the cases).

j) Be sure to include a deliberate runtime error in your app (e.g.: a NullPointerException) caused when the servlet methods are used.

Look for the exception (ant the stack trace) in the application container log. (Then, correct the error.)

📑 The project with the servlet example (after a "clean").

📑 An excerpt from the application server log, highlighting the successful deployment of the application.

📑 Elaborate on: what are the responsibilities/services of a "servlet container"?

k) Assume your app is ready for production. In this case, you may choose to bundle your app and the application server in a Docker container.

Deploy your web app into Tomcat running on port 8088 in a docker container.

Note 1: not to be confused: "servlet/web containers" and "Docker/linux containers"

Note 2: IntelliJ will provide support for deploying a web application into a Tomcat server running in a Docker container.

l) [**Optional**] Given that Java application containers are based on open specifications, you may run your project using a different server, without code modifications. The deployment methods, however, will be different.

You may try other popular servers, e.g.: Wildfly or Payara, instead of Tomcat.

2.2 Server-side programming with embedded servers

For some use cases, you may prefer to run the web container from within your app. In this case, you will be using an "embedded server", since its lifecycle (start, stop) and the deployment of the artifacts is controlled by your application code.

m) Adapting from the sample in embedded server example using the Jetty server (Step #3.4 in the example not required; be sure to complete step #3.5): implement the example from i) but with an embedded server approach.

📑 Project with the embedded container application (after a "clean").

2.3 Introduction to web apps with Spring Boot

Spring Boot is a rapid application development platform built on top of the popular Spring Framework.

By assuming opinionated choices by default (e.g.: assume common configuration options without the need to specify everything), Spring Boot is a convention-over-configuration addition to the Spring platform, very useful to get started with minimum effort and create stand-alone, production-grade applications.

n) Use the Spring Initializr[5] to create the base maven project for your app.

Be sure to add the "**Spring web**" dependency. Spring Initializr templates contain a collection of all the relevant transitive dependencies that are needed to start a particular functionality and will simplify the setup of the POM.

Download the "starter kit" generated by Spring Initializr.

You should be able to build your application using the regular Maven commands. The generated seed project also includes the Maven wrapper script (mvnw).

```
$ mvn install -DskipTests && java -jar target\webapp1-0.0.1-SNAPSHOT.jar
or
$ ./mvnw spring-boot:run
```

Access your browser at http://localhost:8080/; you should retrieve a page produced by Spring Boot (white label error).

Note: mind the **used ports**! You will get an error if 8080 is already in use (maybe there is running Tomcat?...).

o) Build a simple application to serve web content as detailed in this Getting Started article. Start the project from the scratch (instead of downloading the available source code). Try to type the code (so you learn the annotations).

Change the default port (look in application.properties) of the embedded server to something different from 8080 (default).

You may need to include the Thymeleaf starter in the dependencies.

Note: The implementation of Spring MVC relies on the Servlets engine, however, you do not need to "see" them. The abstraction layers available will provide the developer with more convenient interfaces.

p) In the previous example, you created a web application that intercepts the HTTP request and redirects to a page, with a custom message (be sure to change the optional parameter "name").

Extend you project to create a REST endpoint, which listen to the HTTP request, and answer with a JSON result. You may find this guide helpful.

Instead a @Controller, you will use a **@RestController**.

Be sure to access the method from the command line, using the curl utility (or use the Postman tool for API development).

Note: mind the URL path; it should be different from the previous task.

▤ Project with the Spring Boot application (after a "clean").

▤ Web applications in Java can be deployed to stand-alone applications servers or embedded servers. Elaborate on when to choose one over the other (and relate with Spring Boot).

---

[5] Spring Initializr is integrated with intelliJ and can also be used with VS Code.

- Give specific examples of annotations in Spring Boot that implement the principle of convention-over-configuration.

- Which annotations are transitively included in the @SpringBootApplication?