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

Updated: 2019-10-01.

Introduction to the lab

Leaning 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 section (🗁ies2.1, 🗁ies2.2,...) 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**; 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 🗐.

2.1 Server-side programming with servlets

A Servlet is a class that runs at the server, handles requests, processes them and reply with a response. A servlet must be deployed into a (Java) **Servlet Container** in order to become usable. *Servlet* is a generic interface and the *HttpServlet* is an extension of that interface (the most used type of Sevlets).

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 services that we can find in Java EE Application containers.

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 (v8.5 zip) and expand to a local folder.

   Run the application server (see scripts inside bin folder) and access it from your browser:

   ```
   http://localhost:8080
   ```

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

   You may need to register appropriate roles in conf/tomcat-users.xml (see step #6 here).

   From the Manager environment, explore the Examples application already installed in Tomcat (in particular Examples --> Servlet --> Request Parameters).

c) Create a maven-based web application project. Use the following Maven archetype:

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

(You may need to add it 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).

You should get a .war file in the ./target folder. This is your application packaged as a Web ARchive; you may check its contents in a regular Zip tool.

e) Deploy the packed application into the application server.

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

You should be able to confirm the page with "Hello World!" is displayed in the browser (e.g.: http://localhost:8080/yourwebapp/)

f) Deploying using the Manager page has some disadvantages: it is not practical to deploy from within the IDE; it is specific to Tomcat and won't work for other application server.

The productive alternative is to use the **IDE integrated deployment support**.

Be sure to configure the Tomcat server in your IDE, e.g., for Eclipse, for IntelliJ, for NetBeans (just add the server with the Tools/Servers menu); for VS Code.

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).

Note 1: For Java EE, the proper edition of Eclipse is "Eclipse IDE for Enterprise Java Developers"
Note 2: It is also possible to automatize the deployment with Maven, using the Cargo plugin to copy the artifact to the server (see step#5 here).

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

h) 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.

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).

i) Be sure to include a deliberate runtime error in your app (e.g.: a NullPointerException) caused when the servlet methods are used. You should be able to find the exception in the application container log. (Then, correct the error.)

- The project with the servlet example (after a "clean").
- An excerpt from the application server, highlighting the successful deployment of the application.
- Elaborate on: what are the responsibilities/services of a "servlet container"?

j) 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: the container word is being used with different meanings. 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.

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

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

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.

l) Implement the embedded server example using the Jetty server. (Step #3.4 in the example not required; be sure to complete step #3.5).

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.

Note: building Spring Boot applications relies on a lot of dependencies and plugins. You may consider **optimize the Maven downloads** by specifying a mirror of the Maven central repository.

m) Use the Spring Initializr 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.

n) 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).

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

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

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