CHAPTER 10

SportsStore: Deployment

In this chapter, I prepare the SportsStore application for deployment by incorporating all of the application content and the Angular framework into a single file and "containerizing" the application using Docker.

Preparing the Application for Deployment

In the original version of this chapter, I demonstrated how to prepare the application for deployment manually, which was a complicated process that required careful configuration. The `angular-cli` package simplifies the preparation process to a single command.

Make sure that you don't have `ng serve` or the RESTful web service running from previous chapters and run the following command in the SportsStore folder:

```
ng build --target=production
```

This command tells `angular-cli` to compile the code in the application and generate a set of optimized JavaScript files that contain the application functionality and all of the Angular features it depends on. The build process can take a few moments and once it is complete, you will see that a `SportsStore/dist` folder has been created. In addition to the JavaScript files, there is an `index.html` file that has been copied from the `SportsStore/src` folder and modified to use the newly-build files.

Containerizing the SportsStore Application

To complete this chapter, I am going to create a container for the SportsStore application that replaces the development tools I have been using in previous chapters and shows the

transition from development to deployment. In the sections that follow, I will create a Docker container. At the time of writing, Docker is the most popular way to create containers, which is a pared-down version of Linux with just enough functionality to run the application. Most cloud platforms or hosting engines have support for Docker, and its tools run on the most popular operating systems.

Installing Docker

The first step is to download and install the Docker tools on your development machine, which is available from `www.docker.com/products/docker`. There are versions for macOS, Windows, and Linux, and there are some specialized versions to work with the Amazon and Microsoft cloud platforms. The free Community edition is sufficient for this chapter.

Caution One drawback of using Docker is that the company that produces the software has gained a reputation for making breaking changes. This may mean that the example that follows may not work as intended with later versions. The version I used for the examples in this chapter is 17.03. If you have problems, check the repository for this book for updates (`https://github.com/Apress/pro-angular-2ed`) or contact me at `adam@adam-freeman.com`.

Preparing the Application

The first step is to create a configuration file for NPM that will be used to download the additional packages required by the application for use in the container. I created a file called `deploy-package.json` in the `SportsStore` folder with the content shown in Listing 10-1.

Listing 10-1. The Contents of the deploy-package.json File in the SportsStore Folder

```
{
  "dependencies": {
    "bootstrap": "4.0.0-alpha.4",
    "font-awesome": "4.7.0"
  },

  "devDependencies": {
    "express": "4.14.0",
    "concurrently": "2.2.0",
    "json-server": "0.8.21",
    "jsonwebtoken": "7.1.9"
  },
```

```
  "scripts": {
    "start": "concurrently \"npm run express\" \"npm run json\" ",
    "express": "node server.js",
    "json": "json-server data.js -m authMiddleware.js -p 3500"
  }
}
```

The `dependencies` section omits Angular and all of the other runtime packages that were added to the `package.json` file by `angular-cli` when the project is created. This is because the build process incorporates all of the JavaScript code required by the application into the files in the `dist` folder, which means only the add-ons that I am using need to be listed. The `devDependencies` section includes two new tools that were not used in development, as described in Table 10-1.

Table 10-1. *The New NPM Packages Used in the deploy-package.json File*

Name	Description
express	This is a popular HTTP server that is widely used both in its own right and by many other development tools.
concurrently	This is a simple package that allows multiple NPM packages to be started from a single command.

The `scripts` section of the `deploy-package.json` file is set up so that the `npm start` command will start both the RESTful web service and the HTTP server that will deliver the HTML, CSS and JavaScript content to the browser.

Configuring the HTTP Server

One of the packages in Listing 10-1 is `express`, which is a commonly-used HTTP server and which will be used to handle HTTP requested by the application when it is deployed.

Note The express package is an excellent server (and provides many more features than I am using here), but there are plenty of other choices, and any web server will do. I chose express because it is popular and because it is easy to use in a Docker container.

To configure express to run the application, I created a file called `deploy-server.js` in the `SportsStore` folder and added the code shown in Listing 10-2.

Listing 10-2. The Contents of the deploy-server.js File in the SportsStore Folder

```
var express = require("express");

var app = express();

app.use("/node_modules",
    express.static("/usr/src/sportsstore/node_modules"));
app.use("/", express.static("/usr/src/sportsstore/app"));

app.listen(3000, function () {
    console.log("HTTP Server running on port 3000");
});
```

The statements in this file create an HTTP server on port 3000, serving the files in `/usr/src/sportsstore/app` and `/usr/src/sportsstore/node_modules` folders.

Creating the Docker Container

To define the container, I added a file called `Dockerfile` (with no extension) to the `SportsStore` folder and added the content shown in Listing 10-3.

Listing 10-3. The Contents of the Dockerfile File in the SportsStore Folder

```
FROM node:6.9.1

RUN mkdir -p /usr/src/sportsstore

COPY dist /usr/src/sportsstore/app

COPY authMiddleware.js /usr/src/sportsstore/
COPY data.js /usr/src/sportsstore/
COPY deploy-server.js /usr/src/sportsstore/server.js
COPY deploy-package.json /usr/src/sportsstore/package.json

WORKDIR /usr/src/sportsstore

RUN npm install

EXPOSE 3000
EXPOSE 3500

CMD ["npm", "start"]
```

The contents of the `Dockerfile` use a base image that has been configured with Node.js and copies the files required to run the application, including the bundle file containing the

application and the `package.json` file that will be used to install the packages required to run the application in deployment.

Run the following command in the `SportsStore` folder to create an image that will contain the SportsStore application, along with all of the tools and packages it requires:

```
docker build . -t sportsstore  -f  Dockerfile
```

An image is a template for containers. As Docker processes the instructions in the Docker file, the NPM packages will be downloaded and installed and the configuration and code files will be copied into the image.

Running the Application

Once the image has been created, create and start a new container using the following command:

```
docker run -p 3000:3000 -p3500:3500 sportsstore
```

You can test the application by opening `http://localhost:3000` in the browser, which will display the response provided by the web server running in the container, as shown in Figure 10-1.

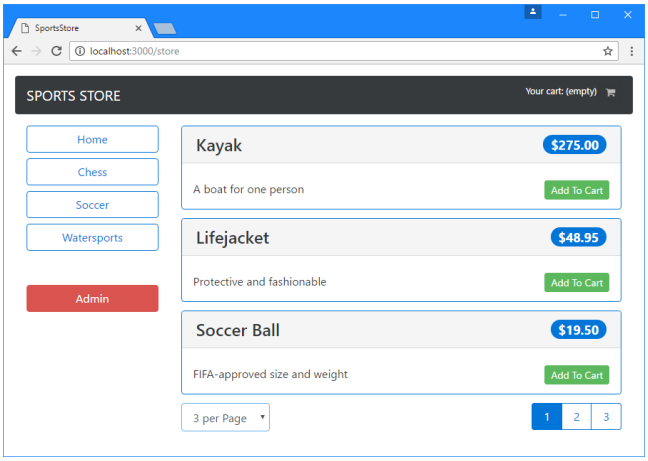

Figure 10-1. Running the containerized SportsStore application

To stop the container, run the following command:

```
docker ps
```

You will see a list of running containers, like this (I have omitted some fields for brevity):

```
CONTAINER ID        IMAGE               COMMAND             CREATED
ecc84f7245d6        sportsstore         "npm start"         33 seconds ago
```

Using the value in the Container ID column, run the following command:

```
docker stop ecc84f7245d6
```

Summary

This chapter completes the SportsStore application, showing how an Angular application can be prepared for deployment and how easy it is to put an Angular application into a container such as Docker.

That's the end of this part of the book. In Part 2, I begin the process of digging into the details and show you how the features I used to create the SportsStore application work in depth.