Containers in development

Containers are not only great in production. They can be used in development environments as well and offer several benefits. The same *works-on-my-machine* problem is faced often when a new developer joins the team. Not to mention the headache of switching runtime versions or a local database!

For example, a software development team at the University of Helsinki has a fully containerized development environment. The principle in all development projects is to have a setup so that a new developer only needs to install Docker and clone the project code from GitHub to get started. Not a single dependency is ever installed on to host machine, Git, Docker and the text editor of choice are the only things that are needed.

Even if your application is not completely containerized during development, containers can be very helpful. For example, say you need MongoDB version 4.0.22 installed in port 5656. It's now an oneliner: "docker run -p 5656:27017 mongo:4.0.22" (MongoDB uses 27017 as the default port).

Let's containerize a NodeJS development environment. As you perhaps know NodeJS is a cross-platform JavaScript runtime that makes it possible to run JavaScript in your machine, servers and embedded devices, among many other platforms

The setup requires some expertise in the way how NodeJS works. Here is a simplified explanation if you're not familiar: libraries are defined in `package.json` and `package-lock.json` and installed with `npm install`. npm is the Node package manager.

To run the application with the packages we have a script defined in package.json that instructs Node to execute index.js, the main/entry file in this case the script is executed with `npm start`. The application already includes code to watch for changes in the filesystem and restart the application if any changes are detected.

The project "node-dev-env" is here https://github.com/docker-hy/material-applications/tree/main/node-dev-env. We have already included a development Dockerfile and a helpful docker-compose.yml.

Dockerfile

```
FROM node:16

WORKDIR /usr/src/app

COPY package* ./

RUN npm install
```

docker-compose.yml

```yaml
version: '3.8'

services:
  node-dev-env:
    build: . # Build with the Dockerfile here
    command: npm start # Run npm start as the command
    ports:
      - 3000:3000 # The app uses port 3000 by default, publish it as 3000
    volumes:
      - ./:/usr/src/app # Let us modify the contents of the container locally
      - node_modules:/usr/src/app/node_modules # A bit of node magic, this ensures the dependencies bui
    container_name: node-dev-env # Container name for convenience

volumes: # This is required for the node_modules named volume
  node_modules:
```

And that's it. We'll use volume to copy all source code inside the volume so CMD will run the application we're developing. Let's try it!

```
$ docker compose up

...

Attaching to node-dev-env
node-dev-env  |
node-dev-env  | > dev-env@1.0.0 start
node-dev-env  | > nodemon index.js
node-dev-env  |
node-dev-env  | [nodemon] 2.0.7
node-dev-env  | [nodemon] to restart at any time, enter `rs`
node-dev-env  | [nodemon] watching path(s): *.*
node-dev-env  | [nodemon] watching extensions: js,mjs,json
node-dev-env  | [nodemon] starting `node index.js`
node-dev-env  | App listening in port 3000
```

Great! The initial start-up is a bit slow. It is a lot faster now that the image is already built. We can rebuild the whole environment whenever we want with `docker compose up --build`.

Let's see if the application works. Use the browser to access http://localhost:3000, it should do a simple plus calculation with the query params.

However, the calculation doesn't make sense! Let's fix the bug. I bet it's this line right here https://github.com/docker-hy/material-applications/blob/main/node-dev-env/index.js#L5

When I change the line, on my host machine the application instantly notices that files have changed:

```
$ docker compose up

...

Attaching to node-dev-env
node-dev-env  |
node-dev-env  | > dev-env@1.0.0 start
node-dev-env  | > nodemon index.js
node-dev-env  |
node-dev-env  | [nodemon] 2.0.7
node-dev-env  | [nodemon] to restart at any time, enter `rs`
node-dev-env  | [nodemon] watching path(s): *.*
node-dev-env  | [nodemon] watching extensions: js,mjs,json
node-dev-env  | [nodemon] starting `node index.js`
node-dev-env  | App listening in port 3000
node-dev-env  | [nodemon] restarting due to changes...
node-dev-env  | [nodemon] starting `node index.js`
node-dev-env  | App listening in port 3000
```

And now a page refresh shows that our code change fixed the issue. The development environment works.

The next exercise can be extremely easy or extremely hard. Feel free to have fun with it.

Exercise 2.11

> **EXERCISE 2.11**
>
> Select some of your own development projects and start utilizing containers in the development environment.
>
> Explain what you have done. It can be anything, e.g., support for docker-compose.yml to have services (such as databases) containerized or even a fully blown containerized development environment.

If you are interested in how to build a containerized development environment for a React/Node Single page web app, please have a look at the course Full stack open which has one chapter devoted to the topic.

Edit this page