CSE138 (Distributed Systems) Assignment 1

Spring 2021

Instructions

General

- For this assignment, you will create an **HTTP web service** that differentiates between `GET` and `POST` requests and that responds to requests at the endpoints `/ping`, `/ping/<name>`, and `/echo`.
- You will use **Docker** to create a container image that exposes a web server at port 8085 implementing the **HTTP interface** described in the next section.
- You must do your own individual work and submit this assignment as an individual. **Do not work with a team.** Teams will be for assignments 2, 3, and 4.
- The assignment is due **04/09/2021 (Friday) 11:59:59 PM**. No late submissions accepted.

Aside: Why containers?

Distributed systems run on multiple computers. For our class, using multiple computers is hard to orchestrate. Virtual machines (VMs) and containers are two ways to simulate running multiple computers on one computer. Using multiple VMs would take up too much resources (memory and processor) for your computer though, so we use multiple containers to use less resources. Docker is a tool for configuring and running containers. Since Docker is standardized it helps course staff to grade everybody's assignments running on multiple containers (computers). Another tool called Podman also implements the container configuration standard which Docker created.

One more reason to use containers is that within a container you have an isolated linux environment. Within that environment you can freely install dependencies of your project. This makes it possible to develop your code on any system which runs Docker (Linux, Mac, Windows) without worrying about differences between the development and deployment environments. The `Dockerfile` describes how to build a container image, by listing the commands necessary to fetch dependencies and eventually run your project code.

Building and testing your container

We have provided a short test script `test_assignment1.py` that you can use to test your work. It is critical that you run the test script before submitting your assignment. The tests we provide are similar to the ones we will run while grading your submissions.

Submission workflow

- Create a GitHub account (https://github.com/join) using your UCSC email if you don't have one already.
- Create a **private** repository named `CSE138_Assignment1`. **Make sure it is private and that only you have access to it.**
- Invite the `ucsc-cse138-staff` GitHub account as a collaborator to the repository.

- Clone the repository to your local machine.
- Create a `Dockerfile`, which will describe how to create your container image, at the top level of your project directory. If you are unfamiliar with Docker, we suggest beginning with the documentation at https://docs.docker.com/get-started/. Parts 1, 2, 3 and 9 of the "Get started" section should help you with this assignment.
- Implement the HTTP interface described below, and commit and push your project files to the repository. **It is better to commit and push early and often.**
- Include a `README.md` file at the top level of your project directory containing the *Acknowledgements* and *Citations* sections. Please refer to the course overview website to learn what needs to go in these sections: http://composition.al/CSE138-2021-03/course-overview.html#academic-integrity-on-assignments
- Submit your repository URL and the commit ID that you would like to be used for grading to the following Google form: https://forms.gle/dvT8tqsoiDubXoJ16

Evaluation and grading

To evaluate the assignment, the course staff will create a container image using the Dockerfile in your project directory by running something like:

```
docker build -t your-project .
docker run -p 8085:8085 your-project
```

We will test your project by sending `GET` and `POST` requests to port 8085. We will be checking that the correct responses and status codes are sent back from your app.

HTTP Interface

The HTTP web service you create will have the following end points: `/ping`, `/ping/<name>`, and `/echo`.

- `/ping`
  - The endpoint at `/ping` accepts a `get` request (with no parameter) and returns the json message body `{"message":"I'm alive!!" }` and status code 200. Here is an example interaction with the service that shows both the response and the status code:

    ```
    $ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://localhost:8085/ping

    {"message":"I'm alive!!"}
    200
    ```

  - If the `/ping` endpoint receives a `POST` request, the response must have status code 405. (We only check the status code in this case; any message body (or no message body) is acceptable.) It should look something like this:

    ```
    $ curl --request POST --write-out "\n%{http_code}\n" http://localhost:8085/ping

    Method Not Allowed
    405
    ```

- `/ping/<name>`
  - The endpoint at `/ping/<name>` accepts a `POST` request (with the path parameter name) . The response should have the JSON message body `{"message":"I'm alive, <name>!!"}` and status code 200 like this:

```
$ curl --request POST --write-out "\n%{http_code}\n" http://localhost:8085/ping/slug

{"message":"I'm alive, slug!!"}
200
```

  - If the `/ping/<name>` endpoint receives a `GET` request, the response must have status code 405. (We only check the status code in this case; any message body (or no message body) is acceptable.) It should look something like this:

```
$ curl --request GET --write-out "\n%{http_code}\n" http://localhost:8085/ping/slug

Method Not Allowed
405
```

- `/echo`
  - The endpoint at `/echo` accepts a `GET` request (with no query parameters). The response should have the JSON message body `{"message":"Get Message Received"}` and status code 200:

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
 http://localhost:8085/echo

{"message":"Get Message Received"}
200
```

  - The `/echo` endpoint also accepts a `POST` request with a `msg` query parameter. The response should have the JSON message body `{"message":"<msg>"}` and status code 200, where `<msg>` is the string passed to the `msg` query parameter:

```
$ curl --request POST --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
http://localhost:8085/echo?msg=foo

{"message":"foo"}
200
```

  - If the `/echo` endpoint receives a POST request with no `msg` query parameter, the response must have status code 400. (We only check the status code in this case; any message body (or no message body) is acceptable.) It should look something like this:

```
$ curl --request POST --write-out "\n%{http_code}\n"  http://localhost:8085/echo

Bad Request
400
```

Further reading

These links aren't necessary to complete the assignment, but might provide useful and interesting context!

- *What even is a container: namespaces and cgroups* is a blog post by Julia Evans that gives a quick overview of what containers are.
- *RESTful Web Services* is a free online book that describes how to make well designed HTTP interfaces called *REST* APIs.

Acknowledgement

This assignment was originally written by Saheed Adepoju, Reza NasiriGerdeh, and Lindsey Kuper, based on Peter Alvaro's course design. The current version includes updates suggested by the CSE138 Spring 2021 course staff (http://composition.al/CSE138-2021-03/course-overview.html#course-staff).

Copyright