CSE138 (Distributed Systems) Assignment 1

Fall 2023

HTTP Service

- Create an **HTTP web service** that differentiates between requests with different HTTP verbs (`GET` and `POST`) and URI paths (`/hello`, `/hello/<name>`, and `/test`). Implement the interface described in the **HTTP Interface** section. Feel free to use any programming languages you find yourself fimiliar with.
- Package the HTTP Service in a container image (specified as a Docker `Dockerfile` or Podman `Containerfile`) which listens on port 8090.

General instructions

- 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.
- Due **Wednesday, October 10th, 2023 by 11:59:59 PM**. No late submissions accepted.

Submission workflow

1. Create a GitHub account if you don't have one already: https://github.com/join
2. Create a **private** repository named `CSE138_Assignment1`. **Make sure it is private and that only you have access to it.**
3. Invite the `ucsc-cse138-fall2023-staff` GitHub account as a collaborator to the repository.
4. Clone the repository to your local machine.
5. At the top level of your repository, create a Docker `Dockerfile` or Podman `Containerfile` to describe how to create your container image.
6. 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.**
7. Include a `README.md` file at the top level of your project directory containing **Acknowledgements** and **Citations** sections. Please refer to the course overview website to learn what needs to go in the below Academic integrity on assignments section.
8. 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/6e6T5Qkw3Drni1QM7

Academic integrity on assignments

You're expected and encouraged to discuss your work on assignments with others. That said, **all the work you turn in for this course must be your own, independent work (for assignment 1) or the independent work of your team (for subsequent assignments).** Students who do otherwise risk failing the course.

You can ask the TAs, the tutors, and classmates for advice, but you cannot copy from anyone else: once you understand the concepts, you must write your own code. While you work on your own homework solution, you can:

- Discuss with others the general techniques involved, *without sharing your code with them.*

- Use publicly available resources such as online documentation.

In the `README.md` file you include with each assignment, you are **required** to include the following sections:

- *Team Contributions* lists each member of the team and what they contributed to the assignment. (There's no need to include this for assignment 1, since assignment 1 is done independently.)
- *Acknowledgments* lists people you discussed the assignment with and got help from. List each person you talked to and the concept that they helped with. If you didn't get help from anyone, you should explicitly say so by writing "N/A" or "We didn't consult anyone."
- *Citations* is for citing sources you used. For anything you needed to look up, document where you looked it up.

Thorough citation is the way to avoid running afoul of accusations of misconduct.

About Container Images

If you are unfamiliar with the syntax of `Dockerfile`/`Containerfile` images, see the examples and documentation at https://docs.docker.com/develop/develop-images/dockerfile_best-practices/. It is recommended to start with a `FROM` specifying an official image for your project's programming language (for example, Python has `FROM python:3` at https://hub.docker.com/_/python and NodeJS has `FROM node:14` at https://hub.docker.com/_/node).

Building and testing

- 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 --rm -p 8090:8090 your-project
```

- We will test your project by sending HTTP requests to the container port 8090 and checking that the correct responses and status codes are sent back from your HTTP service.
- We have provided a short test script `test_assignment1.py` that you can use to test your work. To run the test script you must install the `requests` library for the version of python on your computer (eg. if you are using Python 3.8 do `pip38 install requests`). 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.

HTTP Interface

The HTTP web service you create will have the following endpoints: `/hello`, `/hello/<name>`, and `/test`.

`/hello`

- The endpoint at `/hello` accepts a `GET` request (with no parameter) and returns the JSON response body `{"message":"world"}` 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:8090/hello

{"message":"world"}
200
```

- If the `/hello` endpoint receives a `POST` request, the response must have status code 405. Any response body, or no response body, is acceptable. It should look something like this:

```
$ curl --request POST --write-out "\n%{http_code}\n" http://localhost:8090/hello

Method Not Allowed
405
```

/hello/<name>

- The endpoint at `/hello/<name>` accepts a `POST` request with the path-parameter "name". The response should have the JSON response body `{"message":"Hi, <name>."}` and status code 200 like this:

```
$ curl --request POST --write-out "\n%{http_code}\n" http://localhost:8090/hello/slug

{"message":"Hi, slug."}
200
```

- If the `/hello/<name>` endpoint receives a `GET` request, the response must have status code 405. Any response body, or no response body, is acceptable. It should look something like this:

```
$ curl --request GET --write-out "\n%{http_code}\n" http://localhost:8090/hello/slug

Method Not Allowed
405
```

/test

- The endpoint at `/test` accepts a `GET` request with no query parameters. The response should have the JSON body `{"message":"test is successful"}` and status code 200:

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://localhost:8090/test

{"message":"test is successful"}
200
```

- The `/test` endpoint also accepts a `POST` request with a `msg` query parameter. The response should have the JSON 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:8090/test?msg=foo"

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

- If the `/test` endpoint receives a POST request with no `msg` query parameter, the response must have status code 400. Any response body, or no response body, is acceptable. It should look something like this:

```
$ curl --request POST --write-out "\n%{http_code}\n"  http://localhost:8090/test

Bad Request
400
```

- If the `/test` endpoint receives an unknown query parameter, you can either emit an error response or ignore the unknown query parameter and behave as though it weren't there.

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 written by the CSE138 Fall 2021 course staff, based on Peter Alvaro's course design and with input from the staff from past instances of the course, and was later modified by CSE138 Fall 2023 course staff.

Copyright