CSE138 (Distributed Systems) Assignment 2

Spring 2021

Instructions

General

In this assignment, you will build a **HTTP single-site key-value store** that will support adding a new key-value pair to the store, retrieving and updating the value of an existing key, and deleting an existing key from the store. Furthermore, your key-value store will run as a collection of communicating instances, in which exactly one instance has a **main** role and one or more instances have a **forwarding** role. The main instance responds to a client's or forwarding instance's request directly, while a forwarding instance forwards each request to the main instance, gets a response from it, and returns the response to the client.

- You must do your work as part of a team.
- You will use Docker to create an image that exposes a key-value store at port 8085 implementing the HTTP interface described in the next section.
- The key-value store does not need to persist the data, i.e., it can store data in memory only. That is, if the key-value store goes down and then gets started again, it does not need to contain the data it had before the crash.
- You need to implement your own key-value store, and not use an existing key-value store such as Redis, CouchDB, MongoDB, etc.
- We will only grade the most recently submitted commit ID for each repository, so it is OK to submit more than once.
- The assignment is due **April 23, 2021 (Friday) 11:59:59 PM**. Late submissions are accepted, with a 10% penalty per day of lateness. Submitting during the first 24 hours after the deadline counts as one day late; 24-48 hours after the deadline counts as two days late; and so on.
- You may consider the *order* of name/value pairs in JSON objects to be irrelevant. For example, `{"foo":"bar","baz":"quux"}` is equivalent to `{"baz":"quux","foo":"bar"}`.

Building and testing your container

We have provided a test script `test_assignment2.py` that you can use to test your work. It is critical that you run the test script before submitting your assignment. The tests provided are similar to the ones we will run on our side during grading, and we may also run additional tests consistent with the assignment specification.

The test script contains two *test suites*:

1. `python test_assignment2.py Part1` expects you to run your web service bound to `http://localhost:8085`, similar to the test script for the previous assignment. This suite runs quickly.
2. `python test_assignment2.py Part2` will run several containers in a small virtual network and test communication among the instances. Since the containers have your web service running inside, you don't need to run it manually (and indeed, shouldn't run it manually because it could cause a port collision). This suite takes a couple of minutes to run.

Submission workflow

- One of the members of your team should create a **private** GitHub repository named `CSE138_Assignment2`. Add all the members of the team as collaborators to the repository.
- Invite the `ucsc-cse138-staff` GitHub account as a collaborator to the repository.
- The repository should contain the project files implementing the key-value store, the `Dockerfile` describing how to create your container image, and a `README.md` file at the top level. The readme should have sections for **Acknowledgements**, **Citations**, and **Team Contributions**. 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 *This is a team assignment, and contributions from all the team members are required.*
- Submit your team name, your repository URL, and the commit ID that you would like to be used for grading to the following Google form: https://forms.gle/VYHefrMGEtUKhNJb6
- Only one of the team members needs to submit the form.

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 assignment2-img .
  docker run -p 8085:8085 assignment2-img
  ```

- We will test Part 1 of your project by sending GET, PUT, and DELETE requests to the key-value store running inside your container and listening on port 8085. We will be checking that the correct responses and status codes are sent back from your key-value store.
- For Part 2 of your project we will launch multiple containers with **main** and **forwarding** roles and create a subnet for them to communicate. With that setup we will test by sending GET, PUT, and DELETE requests to particular containers and checking the resultant responses and status codes.

Part 1: Single-Site Key-Value Store

The HTTP single-site key-value store you create has `/key-value-store` as the main endpoint. To do GET, PUT, and DELETE operations on a particular key, the request should be sent to `/key-value-store/<key>`. For example, to get the value of key `myKey`, the GET request is sent to `/key-value-store/myKey` endpoint.

The endpoint at `/key-value-store/<key>` accepts GET, PUT, and DELETE requests with JSON content type. It returns a response in JSON format as well as the appropriate HTTP status code. In the following, we provide examples of interaction with the key-value store using PUT, GET, and DELETE requests.

PUT

If a client sends a PUT request to the `/key-value-store/<key>` endpoint for non-existent key `<key>` with value `<value>`, the new key-value pair is added to the store. Moreover, the store returns the JSON response `{"message":"Added successfully", "replaced":false}` and status code 201 (Created). For example, adding the non-existent key `course1` with value `Distributed Systems` produces the following response:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Distributed Systems"}' http://127.0.0.1:8085/key-value-store/course1

  {"message":"Added successfully","replaced":false}
  201
```

If a client sends a PUT request to the `/key-value-store/<key>` endpoint for existent key `<key>` with value `<value>`, the store updates the value of `<key>` to the value `<value>` and returns the JSON response `{"message":"Updated successfully", "replaced":true}` and status code 200 (OK). For example, updating the value of the existent key `course1` to `Data Structures` produces the following response:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Data Structures"}' http://127.0.0.1:8085/key-value-store/course1

  {"message":"Updated successfully","replaced":true}
  200
```

If a client sends a PUT request to `/key-value-store/<key>` endpoint for (existent or non-existent) key `<key>` **without providing a value**, the store returns the JSON response `{"error":"Value is missing", "message":"Error in PUT"}` and status code 400 (Bad Request). For example, attempting to update the value of key `course1` without providing a value produces the following response:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{}' http://127.0.0.1:8085/key-value-store/course1

  {"error":"Value is missing","message":"Error in PUT"}
  400
```

If a client sends a PUT request to `/key-value-store/<key>` endpoint for non-existent key `<key>` whose length is greater than 50 characters, the store returns the JSON response `{"error":"Key is too long","message":"Error in PUT"}` and status code 400 (Bad Request). For example, attempting to add the key `6TLxbmwMTN4hX7LOQX5NflWHOQKfrTlzcuM5PUQHS52lCizKbEM` with value `Haha` produces the following response:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Haha"}'
    http://127.0.0.1:8085/key-value-store/6TLxbmwMTN4hX7LOQX5NflWHOQKfrTlzcuM5PUQHS52lCizKbEM

  {"error":"Key is too long","message":"Error in PUT"}
  400
```

GET

If a client sends a GET request to the `/key-value-store/<key>` endpoint for existent key `<key>`, the store returns the JSON response `{"doesExist":true,"message":"Retrieved successfully","value":"<value>"}` and status code 200 (OK). For example, retrieving the value of existent key `course1` (with value `Data Structures`) produces the following response:

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8085/key-value-store/course1

  {"doesExist":true,"message":"Retrieved successfully","value":"Data Structures"}
  200
```

If a client sends a GET request to the `/key-value-store/<key>` endpoint for non-existent key `<key>`, the store returns the JSON response `{"doesExist":false,"error":"Key does not exist","message":"Error in GET"}` and status code 404 (Not Found). For example, retrieving non-existent key `course1` produces the following response:

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8085/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in GET"}
  404
```

DELETE

If a client sends a DELETE request to the `/key-value-store/<key>` endpoint for existent key `<key>`, the store deletes the key-value pair and returns the JSON response `{"doesExist":true,"message":"Deleted successfully"}` and status code 200 (OK). For example, to deleting the existent key `course1` produces the following response:

```
$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8085/key-value-store/course1

  {"doesExist":true,"message":"Deleted successfully"}
  200
```

If a client sends a DELETE request to `/key-value-store/<key>` endpoint for non-existent key `<key>`, the store returns the JSON response `{"doesExist":false,"error":"Key does not exist","message":"Error in DELETE"}` and status code 404 (Not Found). For example, attempting to delete the non-existent key `course1` produces the following response:

```
$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8085/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in DELETE"}
  404
```

Part 2: Main and Forwarding Roles

You need to implement both main and forwarding roles in your key-value store. Your implementation will check the `FORWARDING_ADDRESS` environment variable to determine its role. If the value of `FORWARDING_ADDRESS` is empty, the instance is the main instance. Otherwise, it is a forwarding instance.

Assume a scenario in which we have a main instance and a forwarding instance, each running inside Docker containers connected to the same subnet named `mynet` with IP address range `10.10.0.0/16`. The IP addresses of the main and forwarding instances are `10.10.0.2` and `10.10.0.3`, respectively. Both instances are exposed at port number 8085. A client sends a GET, PUT, or DELETE request to one of the instances. If the client sends the request to the main instance, it will get the response directly from the main instance. In the case that the client sends the request to the forwarding instance, the forwarding instance forwards the request to the main instance, gets the response from the main instance, and forwards that response to the client.

In the following, we explain how to implement the above scenario using Docker.

Create subnet

To create the subnet `mynet` with IP range `10.10.0.0/16`, execute

```
$ docker network create --subnet=10.10.0.0/16 mynet
```

Build Docker image containing the key-value store implementation

Execute the following command to build your Docker image:

```
$ docker build -t assignment2-img .
```

Run the main instance

To run the main instance at IP address `10.10.0.2` and port `8085` (mapped from port `8086` of the host machine) in a Docker container named `main-container`, execute the following command:

```
$ docker run -p 8086:8085 --net=mynet --ip=10.10.0.2 --name="main-container" assignment2-img
```

Run the forwarding instance

Execute the following command to run the forwarding instance at IP address `10.10.0.3` and port `8085` (mapped from port `8087` of the host machine) in a container named `forwarding-container`, which will forward requests to the main instance:

```
$ docker run -p 8087:8085 --net=mynet --ip=10.10.0.3 --name="forwarding-container"
  -e FORWARDING_ADDRESS=10.10.0.2:8085 assignment2-img
```

Send requests from client to the main instance

If the client sends a PUT, GET, or DELETE request to the `/key-value-store/<key>` endpoint for (existent or non-existent) key `<key>` at the main instance, the main instance should respond directly to the client with the appropriate response and status code. Some examples are shown below.

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8086/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in GET"}
  404

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Distributed Systems"}' http://127.0.0.1:8086/key-value-store/course1

  {"message":"Added successfully","replaced":false}
  201

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Data Structures"}' http://127.0.0.1:8086/key-value-store/course1

  {"message":"Updated successfully","replaced":true}
  200

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{}'  http://127.0.0.1:8086/key-value-store/course1

  {"error":"Value is missing","message":"Error in PUT"}
  400

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Haha"}'
    http://127.0.0.1:8086/key-value-store/6TLxbmwMTN4hX7LOQX5NflWHOQKfrTlzcuM5PUQHS52lCizKbEM

  {"error":"Key is too long","message":"Error in PUT"}
  400

$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8086/key-value-store/course1

  {"doesExist":true,"message":"Retrieved successfully","value":"Data Structures"}
  200

$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8086/key-value-store/course1

  {"doesExist":true,"message":"Deleted successfully"}
  200
```

```
$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8086/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in DELETE"}
  404
```

Send requests from client to the forwarding instance

If the client sends a PUT, GET, or DELETE request to the forwarding instance, it should get a proper response from the forwarding instance (the forwarding instance forwards the request to the main instance, gets the response, and sends the response to the client).

First stop and remove both the main and forwarding instances:

```
$ docker stop main-container

$ docker stop forwarding-container

$ docker rm main-container

$ docker rm forwarding-container
```

Next, run both main and forwarding instances:

```
$ docker run -p 8086:8085 --net=mynet --ip=10.10.0.2 --name="main-container" assignment2-img

$ docker run -p 8087:8085 --net=mynet --ip=10.10.0.3 --name="forwarding-container"
  -e FORWARDING_ADDRESS=10.10.0.2:8085 assignment2-img
```

Finally, send the following requests to the forwarding instance:

```
$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in GET"}
  404

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Distributed Systems"}' http://127.0.0.1:8087/key-value-store/course1

  {"message":"Added successfully","replaced":false}
  201

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Data Structures"}' http://127.0.0.1:8087/key-value-store/course1

  {"message":"Updated successfully","replaced":true}
  200

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{}'  http://127.0.0.1:8087/key-value-store/course1

  {"error":"Value is missing","message":"Error in PUT"}
  400

$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Haha"}'
    http://127.0.0.1:8087/key-value-store/6TLxbmwMTN4hX7LOQX5NflWHOQKfrTlzcuM5PUQHS52lCizKbEM

  {"error":"Key is too long","message":"Error in PUT"}
```

```
  400

$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"doesExist":true,"message":"Retrieved successfully","value":"Data Structures"}
  200

$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"doesExist":true,"message":"Deleted successfully"}
  200

$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"doesExist":false,"error":"Key does not exist","message":"Error in DELETE"}
  404
```

Send requests from the client to the main instance while the forwarding instance is stopped

Stop and remove both instances:

```
$ docker stop main-container

$ docker stop forwarding-container

$ docker rm main-container

$ docker rm forwarding-container
```

Next, run only the main instance:

```
$ docker run -p 8086:8085 --net=mynet --ip=10.10.0.2 --name="main-container" assignment2-img
```

Now, if the client sends a request to the main instance, it will get the same response as it receives in the case that the forwarding instance is running, e.g., for a PUT request, we have:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Distributed Systems"}' http://127.0.0.1:8086/key-value-store/course2

  {"message":"Added successfully","replaced":false}
  201
```

That is, the main instance can respond to the client's request independently.

Send requests from the client to the forwarding instance while the main instance is stopped

Stop and remove both instances:

```
$ docker stop main-container

$ docker stop forwarding-container

$ docker rm main-container

$ docker rm forwarding-container
```

Next, run only the forwarding instance:

```
$ docker run -p 8087:8085 --net=mynet --ip=10.10.0.3 --name="forwarding-container"
  -e FORWARDING_ADDRESS=10.10.0.2:8085 assignment2-img
```

Now, if the client sends a request to the forwarding instance, the response must have status code 503 (Service Unavailable) and message body as shown below:

```
$ curl --request PUT --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    --data '{"value": "Distributed Systems"}' http://127.0.0.1:8087/key-value-store/course1

  {"error":"Main instance is down","message":"Error in PUT"}
  503

$ curl --request GET --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"error":"Main instance is down","message":"Error in GET"}
  503

$ curl --request DELETE --header "Content-Type: application/json" --write-out "\n%{http_code}\n"
    http://127.0.0.1:8087/key-value-store/course1

  {"error":"Main instance is down","message":"Error in DELETE"}
  503
```

This is because the forwarding instance will forward the request to the main instance, which is not running anymore, at which point the forwarding instance will get an error and relay the error to the client.

Acknowledgment

This assignment was written by 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