CSE 138: Distributed Systems

Fall 2019 - Assignment #2

Assigned: Friday, 10/11/19
Due: Wednesday, 10/23/19

Instructions

General

- You must do your work as part of a team. Your team may not use an existing key-value store (e.g. Redis, MongoDB, etc.)
- Your team will build a **RESTful multi-site key-value store**.
  - The key-value store must be able to:
    1. Insert a new key-value pair.
    2. Update the value of an existing key.
    3. Get the value of an existing key.
    4. Delete an existing key from the store.
  - The key-value store must run as a collection of communicating instances:
    1. One **main** instance directly responds client and forwarded requests.
    2. Many **follower** instances:
       1. forward requests from a client (the **originating client**) to the main instance.
       2. forward responses from the main instance to the **originating client**.
- You will use **Docker** to create a container that runs the RESTful multi-site key-value store at port 13800.
- Your key-value store does not need to persist the data (in-memory only is acceptable).

Building and testing your container

- We provide a test script, `test_assignment2.py`, that you **should** use to test your work.
- The provided tests are similar to the tests we will use to evaluate your submitted assignment.

Submission workflow

- Create a private repository for the team.
  - For simplicity, the repository may be named `cse138_assignment2`.
  - If you wish to reuse this new team repository, or your assignment1 repository, for future assignments, we recommend the use of tags for marking the final version of each assignment and branches for active development. Some helpful links:

        * For tagging repositories: https://git-scm.com/book/en/v2/Git-Basics-Tagging
        * For branch fundamentals: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
        * A "simple" strategy for branch management: https://nvie.com/posts/a-successful-git-branching-model/#the-main-branches

- The Github accounts of the other members of the team as well as `ucsc-cse138-staff` should be added as collaborators to the repository.
- The repository should contain:
    - The `Dockerfile` defining how to build your Docker image.
    - The project file(s) implementing the key-value store.
    - Team members and their contributions listed in the file, `member-contributions.tsv`. This file should be in 2-column, tab-delimited format:
        * The 1st column contains the UCSC ID of a team member.
        * The 2nd column contains a description of the team member's contributions.
- Submit your team name, repository URL, and the commit ID (aka commit hash) to be evaluated here: https://forms.gle/BpAUpfW5xNQGp8iG8
    - The commit timestamp **must be no later than 10/23/2019 11:59 PM PDT**
    - The google form must be submitted within a reasonable time of the due date (preferably 10 minutes).
    - Late submissions are accepted, with a 10% penalty for every 24 hours after the above deadline.

Evaluation and grading

- To evaluate the assignment, the course staff will run your team's project using Docker
- We will check that your team's key-value store will send the correct response and status codes in response to GET, PUT, and DELETE requests.

Key-Value Store REST API

Endpoints

- `/kv-store/<key>` will accept GET, PUT, and DELETE requests with JSON content type. The response will be in JSON format and return a status code as appropriate.

PUT Requests

Insert new key

- To insert a key named `sampleKey`, send a PUT request to /kv-store/sampleKey. The key-value store should respond with status code 201 and JSON: `{"message":"Added successfully","replaced":false}`.

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                  \
       --data      '{"value": "sampleValue"}'        \
       http://127.0.0.1:13800/kv-store/sampleKey

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

- If no value is provided for the new key, the key-value store should respond with status code 400 and JSON: `{"error":"Value is missing","message":"Error in PUT"}`.

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                   \
       --data      '{}'                               \
       http://127.0.0.1:13800/kv-store/sampleKey

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

- If the value provided for the new key has length greater than 50, the key-value store should respond with status code 400 and JSON: `{"error":"Key is too long","message":"Error in PUT"}`.

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                   \
       --data      '{"value": "sampleValue"}'         \
       http://127.0.0.1:13800/kv-store/6TLxbmwMTN4hX7L0QX5NflWH0QKfrTlzcuM5PUQHS52lCizKbEM

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

Update existing key

- To update an existing key named `sampleKey`, send a PUT request to `/kv-store/sampleKey`. The key-value store should respond with status code 200 and JSON: `{"message":"Updated successfully","replaced":true}`

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                   \
       --data      '{"value": "updatedValue"}'        \
       http://127.0.0.1:13800/kv-store/sampleKey

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

- If no updated value is provided for the key, the key-value store should respond with status code 400 and JSON: `{"error":"Value is missing","message":"Error in PUT"}`

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                   \
       --data      '{}'                               \
       http://127.0.0.1:13800/kv-store/sampleKey

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

GET

Read an existing key

- To get an existing key named `sampleKey`, send a GET request to `/kv-store/sampleKey`.

    - Assuming the current value of `sampleKey` is `sampleValue`, the key-value store should respond with status code 200 and JSON: `{"doesExist":true,"message":"Retrieved successfully","value":"sampleValue"}`
    - If the value of `sampleKey` has been updated to `updatedValue` (as in the "Update existing key" example above), then the key-value store should respond with status code 200 and JSON: `{"doesExist":true,"message":"Retrieved successfully","value":"updatedValue"}`

```
$ curl --request   PUT                                \
       --header    "Content-Type: application/json"  \
       --write-out "%{http_code}\n"                  \
       --data      '{"value": "sampleValue"}'        \
       http://127.0.0.1:13800/kv-store/sampleKey

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

$ curl --request GET                                \
       --header "Content-Type: application/json" \
       --write-out "%{http_code}\n"                 \
       http://127.0.0.1:13800/kv-store/sampleKey

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

$ curl --request   PUT                               \
       --header    "Content-Type: application/json" \
       --write-out "%{http_code}\n"                 \
       --data      '{"value": "updatedValue"}'      \
       http://127.0.0.1:13800/kv-store/sampleKey

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

$ curl --request GET                                \
       --header "Content-Type: application/json" \
       --write-out "%{http_code}\n"                 \
       http://127.0.0.1:13800/kv-store/sampleKey

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

- If the key, `sampleKey`, does not exist, the key-value store should respond with status code 404 and the JSON: `{"doesExist":false,"error":"Key does not exist","message":"Error in GET"}`

```
$ curl --request GET                                \
       --header "Content-Type: application/json" \
       --write-out "%{http_code}\n"                 \
       http://127.0.0.1:13800/kv-store/sampleKey

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

DELETE

Remove an existing key

- To delete an existing key named `sampleKey`, send a DELETE request to `/kv-store/sampleKey`. The key-value store should respond with status code 200 and JSON: `{"doesExist":true,"message":"Deleted successfully"}`

```
$ curl --request DELETE                        \
       --header "Content-Type: application/json" \
       --write-out "%{http_code}\n"            \
       http://127.0.0.1:13800/kv-store/sampleKey

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

- If the key, `sampleKey`, does not exist, the key-value store should respond with status code 404 and JSON: `{"doesExist":false,"error":"Key does not exist","message":"Error in DELETE"}`

```
$ curl --request DELETE                        \
       --header "Content-Type: application/json" \
       --write-out "%{http_code}\n"            \
       http://127.0.0.1:13800/kv-store/sampleKey

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

Main and Forwarding Roles for Multi-site coordination

Description

```
+---------+                         +------+
| client |                          | main |
+---------+                         +------+
    |                                   |
    | ---------------request-----------> |
    | <--------------response----------- |
    |                                   |
    |                                   |
    |             +----------+          |
    |             | follower |          |
    |             +----------+          |
    |                  |                |
    | ----request--->  |                |
    |                  | ---request---> |
    |                  | <---response--- |
    | <---response---  |                |
    |                  |                |
    v                  v                v
```

You will implement a key-value store that may be started as either: a **main** instance or a **follower** instance. Your key-value store will check the value of the environment variable, `FORWARDING_ADDRESS` to determine its role:

- If `FORWARDING_ADDRESS` is empty, the instance is the **main** instance.
- Otherwise, the instance is a **follower** instance.
- The main instance should always respond directly to requests.
- The follower instance should forward requests to the main instance, then forward the response to the client. If the follower instance does not receive a response from the main instance the response to the

client must have status code 503 and one of the following JSON:

– {"error":"Main instance is down","message":"Error in PUT"}
– {"error":"Main instance is down","message":"Error in GET"}
– {"error":"Main instance is down","message":"Error in DELETE"}

Docker Network Management

In the following, we explain a scenario where we have one main instance, one follower instance, and a Docker subnet named **kv_subnet**.

- Create subnet, `kv_subnet`, with IP range `10.10.0.0/16`:

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

- Build Docker image containing the key-value store implementation:

```
$ docker build -t kv-store:2.0 <path-to-Dockerfile-directory>
```

- Run the main instance at `10.10.0.2:13800` in a Docker container named `main-instance`:

```
$ docker run -p 13800:13800 --net=kv_subnet --ip=10.10.0.2 --name="main-instance" kv-store:2.0
```

- Run the follower instance at `10.10.0.3:13800` in a Docker container named `follower-instance`, which will forward requests to the main instance:

```
$ docker run -p 13801:13800                          \
             --net=kv_subnet                         \
             --ip=10.10.0.3                          \
             --name="follower-instance"              \
             -e FORWARDING_ADDRESS="10.10.0.2:13800" \
             kv-store:2.0
```

- Stop and remove containers:

```
$ docker stop main-instance
$ docker stop follower-instance
$ docker rm main-instance
$ docker rm follower-instance
```