CSE 138: Distributed Systems

Winter 2023 - Assignment #2

Assigned: Saturday, 01/21/2023 — Due: Saturday, 02/04/2023

General Instructions

- You must do your own individual work and submit this assignment as an individual.
- You should get started to form a group for future assignments.
- You may not use an existing key-value store (e.g. Redis, MongoDB, etc.)
- You 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 to clients and forwarded requests from follower instances.
    2. Many **follower** instances:
       1. forward requests from a client (the **originating client**) to their upstream (either the main instance or another follower instance).
       2. forward responses from their respective upstream 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 (only in-memory).

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 simplicity, the repository may be named `cse138_assignment2`.
  - If you wish to reuse 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 `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.
- Submit your CruzID, repository URL, and the commit ID (aka commit hash) to be evaluated here: https://forms.gle/ohMPKjU4EmLiQfFw6
  - **For full credit**, you will also need to submit a list of CruzIDs of your teammates.
  - The commit timestamp **must be no later than 02/04/2023 11:59 PM PT**
  - 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 project using Docker.
- We will check that your 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

- `/kvs` 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.

API Specification

PUT A PUT request to endpoint `/kvs` should add a value for a key or update it if it already exists.

- To set the value of an key named `sampleKey` to `superValue`, send a PUT request to `/kvs` with JSON body `{"key": "sampleKey", "val": "superValue"}`.
  - If the key is new and the key-value store previously contained no value for it, the key-value store should respond with status code 201 and JSON body `{"replaced": false}`.
  - If the key already existed with previous value `sampleValue`, the key-value store should respond with status code 200 and JSON body `{"replaced": true, "prev": "sampleValue"}`.
- If the key or the value have length greater than 200, the key-value store should respond with status code 400 and JSON: `{"error": "key or val too long"}`.
- If a request has a malformed body (not a JSON with `key` and `val`), the key-value store should respond with status code 400 and JSON body `{"error": "bad PUT"}`

GET A GET request to endpoint `/kvs` should return the current value of the key, or an error if it does not exist.

- To get an existing key named `sampleKey`, send a GET request to `/kvs` with JSON body `{"key": "sampleKey"}`.
  - If the current value of `sampleKey` is `sampleValue`, the key-value store should respond with status code 200 and JSON body `{"val": "sampleValue"}`
  - If the key, does not exist, the key-value store should respond with status code 404 and JSON body: `{"error": "not found"}`
- If a request has a malformed body (not a JSON with `key`), the key-value store should respond with status code 400 and JSON body `{"error": "bad GET"}`

DELETE A DELETE request to endpoint `/kvs` should delete a value for a key if it already exists.

- To delete the value of a key named `sampleKey`, send a DELETE request to `/kvs` with JSON body `{"key": "sampleKey"}`.
  - If the key exists with value `sampleValue`, the key-value store should respond with status code 200 and JSON body `{"prev": "sampleValue"}`.
  - If the key does not exist, the key-value store should respond with status code 404 and JSON body `{"error": "not found"}`.
- If a request has a malformed body (not a JSON with `key`), the key-value store should respond with status code 400 and JSON body `{"error": "bad DELETE"}`

Example

1. GET a non-existent key `sampleKey`:

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

{"error": "not found"}
404
```

2. PUT a non-existent key `sampleKey` with value `sampleValue`:

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

{"replaced": false}
201
```

3. GET an existing key `sampleKey` with pre-existing value `sampleValue`:

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

```
{"val": "sampleValue"}
200
```

4. PUT an existing key `sampleKey` with value `superValue`:

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

{"replaced": true, "prev": "sampleValue"}
200
```

5. GET an existing key `sampleKey` with (updated) pre-existing value `superValue`:

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

{"val": "superValue"}
200
```

6. DELETE an existing key `sampleKey` with pre-existing value `superValue`:

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

{"prev": "superValue"}
200
```

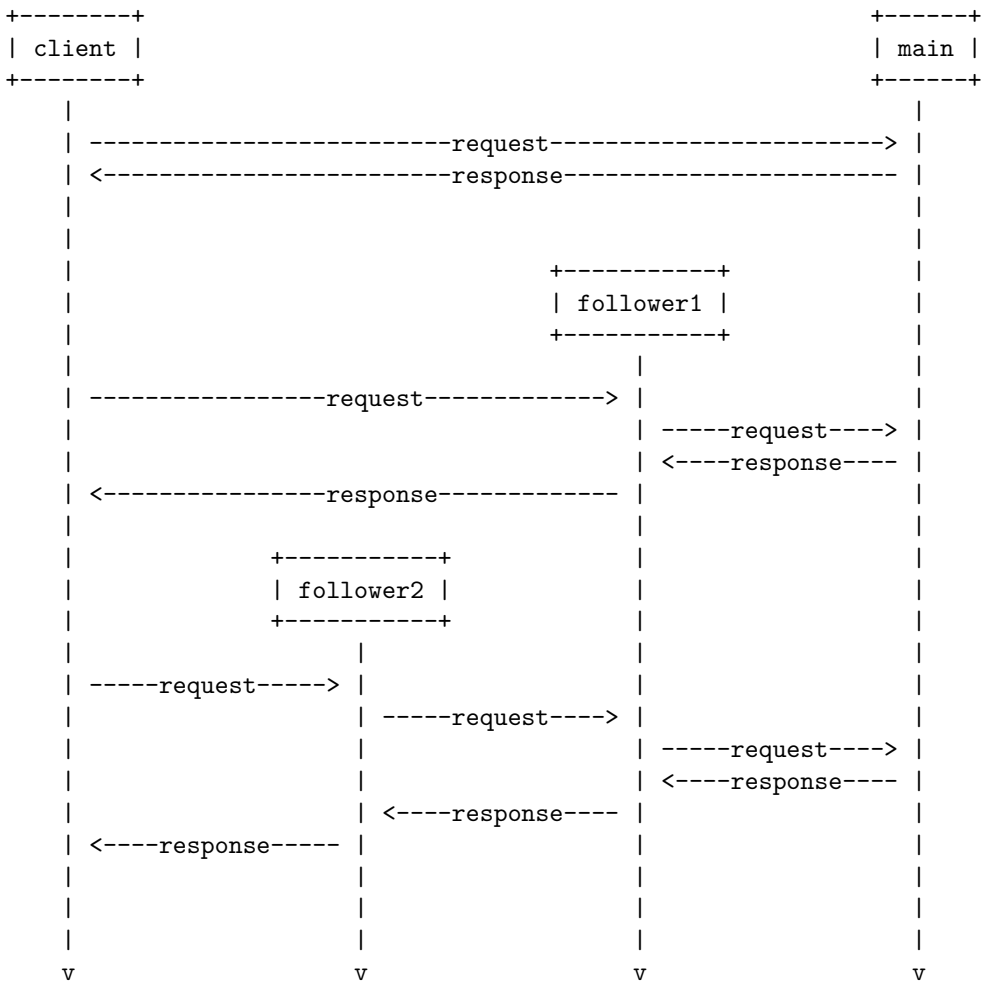
7. DELETE a non-existent (deleted) key `sampleKey`:

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

{"error": "not found"}
404
```

8. GET a non-existent (deleted) key `sampleKey`:

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

{"error": "not found"}
404
```

Main and forwarding roles for multi-site coordination



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 its upstream, then forward the response (including errors generated by a follower, if its upstream is a follower) to its downstream.
- If the follower instance has an upstream with address "10.10.0.42:13800" and does not receive a response from its upsteram in a reasonable amount of time (say, 10s), response to the downstream must have status code 503 and one of the following JSON: `{"error": "upstream down", "upstream": "10.10.0.42:13800"}`
- A follower instance should always launch, regardless of whether its upstream is alive during launch.

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 kvs:1.0 <path-to-Dockerfile-directory>
```

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

```
$ docker run --publish 13800:13800 \
             --net=kv_subnet \
             --ip=10.10.0.2 \
             --name="kvs-main" \
             kvs:1.0
```

- Run the follower instance at `10.10.0.3:13800` in a Docker container named `kvs-follower1`, which will forward requests to the main instance:

```
$ docker run --publish 13801:13800 \
             --net=kv_subnet \
             --ip=10.10.0.3 \
             --name="kvs-follower1" \
             --env FORWARDING_ADDRESS="10.10.0.2:13800" \
             kvs:1.0
```

- Stop and remove containers:

```
$ docker stop kvs-main
$ docker stop kvs-follower1
$ docker rm kvs-main
$ docker rm kvs-follower1
```