SQUARE

User documentation

I. Some information

It's a project written in JAVA using Quarkus framework . Due to portability of JAVA, it will work on all OS but we have only use linux for development and testing. Also it's required a POSTGRESQL database.
This project was structured as following folders:

- apps: contains all jar which can be specify in deploy request
- docker-images: contains dockerfile build dynamically by square
- lib-client: contain executable of square-client project
- square: contains sources of Square REST API
- square-client: contains sources of a lib which are embedded in all of deployed container
- docs: contains some documentation's files like javadoc, swagger which describe REST endpoint.

This repository contains also some files like schema or conception's diagrams.

II. Install dependencies

To work correctly, some packages are need like docker-cli, maven and JDK 11 or later.
First, pull project repository from gitlab with following url and go to the folder, this is url: ***https://gitlab.com/esipe-info2019/dacosta-koffi.git***
We provide a shell script to make this job. To run script named *script.sh* , grant execution right on this file using ***chmod u+x script.sh*** and run with ***./script.sh***.
This script will install all needed dependencies and set up your ***JAVA_HOME***.

III. Build, package and run project

- Square

Folder named ***square*** contains sources of a REST API which use by user to make some action like run docker, get a list of current running docker, get logs of x last minutes, etc...

First go to ***square*** folder, open the project with your favorite editor and change database credential with your database credentials in file ***src/main/resources/application.properties***.

```
quarkus.datasource.url=jdbc:postgresql://151.80.57.175/square_logs
quarkus.datasource.driver=org.postgresql.Driver
quarkus.datasource.username=square
quarkus.datasource.password=kawaii
quarkus.datasource.max-size=8
quarkus.datasource.min-size=2
```

After that, change in same file change host value `quarkus.http.host=127.0.0.1`, replace ***127.0.0.1*** by your IP.

Now run ***./mvnw package***, it will create an executable of square projet ***target*** folder named ***square-1.0-SNAPSHOT-runner.jar*** .

Copy ***target/square-1.0-SNAPSHOT-runner.jar*** to root folder of repository as square.jar and go to root folder.

Now check if postgres service is started and run ***java -jar square.jar*** to run REST API.

- Square-client

Square-client is a small project which will add in each docker container. It's monitoring app running inside docker to send logs produced by app, and notify Square API if app is kill.

Go to ***square-client*** folder and run ***mvn package*** to build, and package jar of square-client. After that, copy executable named ***square-client-1.0-SNAPSHOT-jar-with-dependencies.jar*** from ***target*** folder of current folder into folder named ***lib-client*** which is located at the root of repository.

IV. Usage

Now, you can use your favorite http request client's tool (ex: Postman, curl ...) to interact with Square REST API.
It's will details into this part all of endpoints and requests format to interact with API.
All endpoints consumes and produces JSON only, so please configure your client tool to use ***"application/json"*** as ***Content-Type***.
All endpoint in blue are GET method, and GREEN are POST method.
All POST request which not respect request sample will return a 400 http status to indicate it's a bad request. In case of code request http code will be 201.
All GET request return 200 as status code to indicate success.
Illegal argument status code (422) are return if your request have a right format but argument is not good. For example, you ask to stop a docker with id 1 but nobody don't have this id.
There are three main endpoints:

- /app/* : use to manage instance
    - /app/deploy: this endpoint is used to start a docker's container with a specified application which must present in the apps folder described in the first part.
      This is a valid request example:

```
{
    "app": "todomvc:8082"
}
```

      Return either an empty json if some error occurred during deploy or if app specified is not present in the ***apps*** folder but a json contains all information concerned deploy application like id, docker-instance, service-port, app name.

```
{
    "id": 201,
    "app": "todomvc:8082",
    "port": 8082,
    "service-port": 15201,
    "docker-instance": "todomvc-12"
}
```

- /app/list: get list of current running application
- /app/stop: use to stop a specified application by using id received after a deploy request. Example of valid request:

```
{
    "id": 201
}
```

Return either an empty json if some error occurred during stop process or a json contains some information concerned stopped application like elapsed time.

example:

```
{
    "id": 201,
    "app": "todomvc:8082",
    "port": 8082,
    "service-port": 15201,
    "docker-instance": "todomvc-12",
    "elapsed-time": "4m37s"
}
```

- /logs/*: use get logs of deployed apps
  - /logs/:time: get all logs between now and *time* last minutes. *time* must be an integer between 0 and INTEGER_MAX value.
  - /logs/:time/:filter: get all logs between now and *time* last minutes for specified *filter*.
    filter can be an id, docker-instance or app field value received after a deploy request

An example of response is:

```
[
    {
        "id": 201,
        "app": "todomvc:8082",
        "port": 8082,
        "service-port": 15201,
        "docker-instance": "todomvc-12",
        "message": "ceci est un message de log",
        "timestamp": "2019-10-15T23:58:00.000Z"
    }
]
```

- /auto-scale/*: use manage auto scaling service
  - /auto-scale/update: endpoint use to set up an auto-scale configuration.

An example of valid request:

```
{
    "todomvc:8082": 2,
    "demo:8083": 1
}
```

This request set the following configuration, for app name *todomvc:8082,* 2 instances must be running is the same time and for *demo:8083*, only 1 instance is needed.
Return a json contains information about action need for applications which have a scaling constraint, for example:

```
{
    "todomvc:8082": "no action",
    "demo:8083": "need to stop 1 instance(s)"
}
```

- /auto-scale/status: use to get action which will perform by auto scaling service for current running instances.
  An example of response is:

```
{
    "todomvc:8082": "no action",
    "demo:8083": "need to stop 1 instance(s)"
}
```

- /auto-scale/stop: use to stop auto scaling service, it return a scaling config performed by scaling service before stop.
  An example of response is:

```
{
    "todomvc:8082": 2,
    "demo:8083": 1
}
```

In addition to previous listed endpoints, there are two hidden endpoints use to receive some information from lib-client embedded inside of each docker container.
- /container-log/send-log: normally you doesn't need to trigger it manually, it only trigger dynamically by lib-client to send logs of a running application.

- /container-log/status: it use by *lib-client* to notify if an application running inside docker is killed.
At bonus, you can use /healthcheck endpoint to check is square REST API is current alive or not.