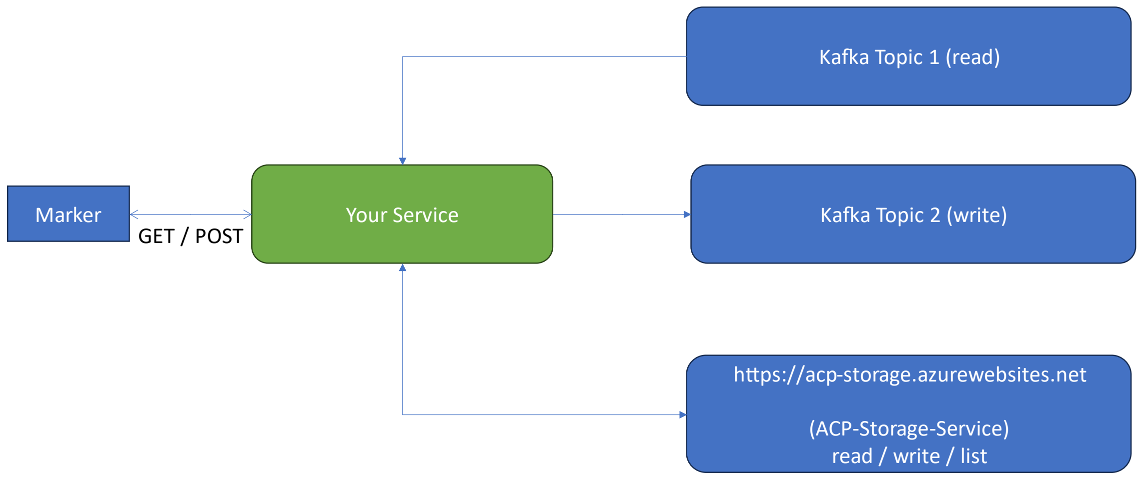
ACP Assignment 2 Specifications (Programming Task)

18.02.2024

In this assignment you are supposed to implement a service which provides several endpoints to communicate with other services / Kafka.

The auto-marker will call your endpoints using POST.

Scenario:



Your main tasks can be summarized as follows (no changes from assignment 1):

1. Create a Java-REST-Service
   - Preferred with Spring Boot, though other frameworks can be used as well
   - Port 8080 is consumed
   - Implement the endpoints
   - Proper parameter handling
   - Proper return code handling
   - JSON handling

2. Place the service in a docker image
   - **amd64** as target architecture – **not arm64** (this is relevant for the Mac users!)

3. save the docker image in a file called **acp_submission_image.tar**
   (it is in TAR format anyhow)

4. **place** the file **acp_submission_image.tar into your root directory of your solution**

   Your directory would look something like this:

```
acp_submission_2
    acp_submission_image.tar
    src (the Java sources…)
        main
                …
        …
```

5. Create a ZIP file of your solution directory
   - Image
   - Sources
   - IntelliJ (or whatever IDE you are using) project files

6. upload the ZIP as your submission in Learn

General information:

- All connection information to connect to Kafka will be passed into the endpoints using a list of key-values as JSON in the BODY of the request. The topics are passed in the URL

  Info: Due to this passing as JSON in the body, we have no classical GET methods, only POST, as GET is not intended to carry any payload in the body of the request

  - The properties used in all Kafka-examples we did are of type Properties (https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html)

  - A property is in principle only a collection of key – value pairs and this is what you will receive int the request. An array of key – value pairs

```
[
   { “key” : “value” },
   { “key” : “value” }
]
```

    As an example this could look like:

```
[
   { “bootstrap.servers”: “…server address…” },
   { “sasl.jaas.config” : “…config / connect string…” },
   { “security.protocol” : “SASL_SSL” },
   { “sasl.mechanism” : “PLAIN” },
   { “group.id” : “…group identifier…” },
    { “storage.server” : “…base url of storage server…” }

]
```

The last entry is new, only needed for the storage service and not present in every request.
Only requests where the storage service is to be used, will have this entry.

- So, you must read your standard properties and apply the passed in ones (overwrite) your loaded ones to be ready for the access
  This way, you can always test in your local environment with just your defaults (and pass in an empty array – so no overrides happen) and during testing, you will receive the necessary information

  For the storage services you can assume http://acp-storage.azurewebsites.net/ as a base address. This is hosted on a small instance, so sometimes the requests can be slow.

  You can test using e.g. curl https://acp-storage.azurewebsites.net/list/blob to list all existing BLOB records

- All JSON passed in will be always in the syntactical correct format, so you can ignore error handling there. What you still have to check is that the data gives you a proper connection and no wrong address or invalid user / password is passed (so classical flow errors in an application).

- Parameters given as {…} in the task will be replaced at runtime with the corresponding value. So, for the first task readTopic/{topicName} could become readTopic/topic123

- **Every writing to a Kafka topic should use your student id as key**

- The points after a task are the maximum achievable points for the individual task

- 200 shall be returned for all OK operations, 400 for problems and 500 only in case you did not catch an exception – which will cause a penalty in points, as exceptions are to be caught by you and only 400 returned.

  The auto-marker will not force to produce 500 codes, yet should a 500 arise this is a clear indicator that you didn’t catch an exception…

The provided and used storage service has additional features:

Write data to a BLOB
`https://acp-storage.azurewebsites.net/write/blob`

List all BLOBs
https://acp-storage.azurewebsites.net/list/blob

Read the data structure from the BLOB
https://acp-storage.azurewebsites.net/read/blob/ba20ea3b-aa49-439f-bea6-cee55be6bc7b

The REST-Service has to provide the following endpoints (all POST):

- **`(4) readTopic/{topicName}`**

  Return the data read from the topic {topicName} as a String from the service

- **`(4) writeTopic/{topicName}/{data}`**

  Write the data {data} to topic {topicName}

- **(7) transformMessage/{readTopic}/{writeTopic}**

  Read String data from {readTopic}, convert to uppercase and write to {writeTopic}

- **`(9) store/{readTopic}/{writeTopic}`**

  - Read String data from {readTopic}
  - Write it to a BLOB using the storage service (passed in in the properties) where the actual URL to use is composed of: baseUrl + **/write/blob**

    For this to succeed the BODY of the request has to have the following JSON object

    ```
    {
       "uid": "your student id",
       "datasetName": "whatever you want",
       "data": "the data read from {readTopic}"
    }
    ```

  - The storage service returns a UUID (not a string!) as result, which you have to
  - write to {writeTopic}

- **(11) retrieve/{writeTopic}/{uuid}**

  Read the BLOB for the passed UUID and write the entire JSON structure into the topic using your student id as key

  The last request is the most complex one. On a logical level this would work out like:

  - Read data from topic ABC -> "XXX"
  - Write the data as a BLOB and get the UUID for the BLOB
  - Write the UUID to a topic using your student id as key and the UUID as value

The following should be considered when implementing the REST-service:

- Do proper checking for URLs, data, etc. Don't handle anything not accurate (you will receive error data and requests!)
- Your endpoint names have to match the specification
- Storing data in a REST service either has to be done on a per session, or (as not differently specified) could be done on a global basis. Up to you.
- Test your endpoints using a tool like Postman or curl. Plain Chrome / Firefox, etc. will do equally for the GET operations
- The filename for the docker image file has to be exactly as defined as well as the location of it in the ZIP-file. Should you be in doubt, use copy & paste to get the name right

Should you need help:

- See the literature links in Week 3, 4 and 5. You should find most information there
- If you cannot find an answer to your question, please post it on Piazza, though try finding it yourself first, please (as we have only limited capacity)

Disclaimer: We will not able to answer last minute questions right before the deadline, so please make sure you start the assignment in good time

Marking:

This programming task has a maximum mark of 35 / 100 points in relation to the entire ACP mark for the course (essay task will be 30 / 100 in relation to the entire ACP mark for assignment 2).

The marks will be allocated purely on auto-tests based on the following criteria:

- Proper runnable docker image
- Proper behavior (functionality)
- Proper error handling
- Proper status codes

Should you fail to provide a runnable docker image according to the specification or provide no source code in the submission, no marking will be possible, and you will receive 0 points.

Additional information sources:

- https://www.codejava.net/frameworks/spring-boot/file-download-upload-rest-api-examples

- https://dzone.com/articles/java-springboot-rest-api-to-uploaddownload-file-on
- https://medium.com/techinpieces/how-to-upload-files-using-rest-service-with-java-jersey-jax-rs-on-tomcat-847dc0e6a179