ACP Assignment 1 Specifications (Programming Task)

24.01.2026

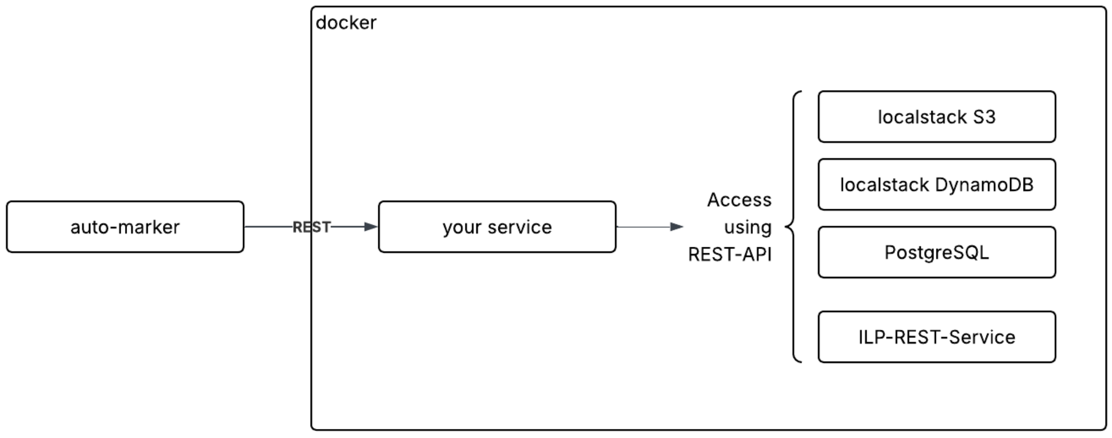
The first assignment has been designed as a preliminary to tackling larger and more complex situations in the second assignment.

Please make sure you read this and the ***accompanying ACP Submission Check document*** as it contains additional information.

Scenario:

For CW1 a somehow constructed scenario has been defined, which allows several common services in cloud programming to be utilized: S3, DynamoDB and Postgres as well as direct integration of a 3rd party service. S3 and DynamoDB will be made available via localstack, Postgres as independent container as well as the 3rd party ILP service.

The schematic overview looks like this:



Your service will run as a dedicated container inside docker – alongside the other services. As you won't know the necessary addresses, access points and credentials it is mandatory that you can accept them using the defined environment variable passing system (Spring Boot Configuration).

These variables will be:

- ACP_POSTGRES for the Connection string to the Postgres Service
- ACP_S3 for the S3 endpoint
- ACP_DYNAMODB for the DynamoDB endpoint

Your main tasks can be summarized as follows:

1. Create a Java-REST-Service according to the specification below
    - Preferred with Spring Boot, though other frameworks can be used as well – if Java is the underlying programming language
    - Port 8080 is consumed
    - Implement one endpoint each for POST and GET
    - Proper parameter handling
    - Proper return code handling
    - JSON handling

2. Place the service in a docker image

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_1
    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:

- Whenever you have a successful operation, you return 200 as http-code. If something is not found (and should be there), 404. ***Any other code is not to be used.***

- Code 404 needs no BODY-data as it is elaborate enough on its own.

- All API-calls and endpoints will start with /api/v1/acp -> so, `http://localhost:8080/api/v1/acp/uuid` , etc.

- Please pay explicit attention to endpoint names, data structures / contracts and optional / mandatory fields as these are non-negotiable and if not applied correctly will result in failure / deductions

System information:

- You are supposed to use the aws SDK (some documentation can be found here: https://docs.localstack.cloud/aws/integrations/aws-sdks/java/) with V2 wherever possible

- Postgres
    - user / password will be postgres / postgres
    - database name will be acp
    - you are (by means of the connect string) using a dedicated schema for your user (SID) inside the database

- aws
    - ACCESS_KEY and SECRET for aws will be "test"
    - Aws-Region will be US_EAST_1 - `Region.US_EAST_1` - (makes it easier with some tasks as otherwise much more must be defined)
    - In DynamoDB the table will be your SID
    - In S3 the bucket will be your SID

The REST-Service must provide the following endpoints:

{key}, {bucket}, {table}, etc. will be replaced by real values at runtime

VERB	Endpoint	Data in
GET	**all/s3/{bucket}**	-
Read all objects in {bucket} and return the content of each verbatim in a large JSON Array (each object being one array element)		
GET	**single/s3/{bucket}/{key}**	-
Read the object specified by {key} in {bucket} and return the content verbatim as JSON		
GET	**all/dynamo/{table}**	
Read all objects in {table} and return the content of each verbatim in a large JSON Array (each element being one object)		
GET	**single/dynamo/{table}/{key}**	
Read the object specified by {key} in {table} and return the content verbatim as JSON		
GET	**all/postgres/{table}**	
Read all rows in {table} and return the content as a JSON array. Each element of the array shall be the row where each column is an element in the JSON object. As example: If your table has 3 columns with ID (int), Name (Varchar) and Age (int) your JSON would look like: { "ID": 1234, "Name": "Michael", "Age": -1 }		
POST	**process/dump**	{ "urlPath": "https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/drones" }
Read the URL defined by the ACP_URL_ENDPOINT + {urlPath} and serialize the returned objects like: { "name": "Drone1", "id": "1", "capability": { "cooling": true, "heating": true,		

`"capacity": 4,`
`"maxMoves": 2000,`
`"costPerMove": 0.01,`
`"costInitial": 4.3,`
`"costFinal": 6.5`
`}`
`}`
 	
As result return the above data structure enriched with a field "costPer100Moves" on the topmost level which is calculated like:
costInitial + costFinal + costPerMove * 100
 	
Should a value be NaN assume 0 for it
 	
Therefore, you are supposed to read the data, de-serialize it, analyze + process and then write it back as response.		

POST	**process/dynamo**	{
		"urlPath": "https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/drones"
		}

As /process/dump just that no return to the caller is done (only 200), yet the processed records are written to DynamoDB (table = SID) and the key is the drone's name. All drone data as JSON is the attribute		

POST	**process/s3**	{
		"urlPath": "https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/drones"
		}

As /process/dump just that no return to the caller is done (only 200), yet the processed records are written to a S3 bucket (bucket = SID) and each object's key is the drone's name. All drone data as JSON is the content		

POST	**process/postgres/{table}**	{
		"urlPath": "https://ilp-rest-2025-bvh6e9hschfagrgy.ukwest-01.azurewebsites.net/drones"
		}

As /process/dump just that no return to the caller is done (only 200), yet the processed records are written one row per object to the {table} in Postgres. The table is existing before the call occurs and has 1 column for each attribute of the target JSON object in the corresponding data type. So, the "name" attribute would be the column name with a VARCHAR datatype and as primary key. You must therefore transform the JSON into INSERT statements and execute these against the database.		

POST	**copy-content/dynamo/{table}**	--
Take the Postgres {table} and read all lines like above in “/all/postgres/{table} just not returning the data to the caller but writing one entry into DynamoDB for each object. The key shall be a dynamically generated UUID So, if there are 7 rows in Postgres you are supposed to have 7 rows in DynamoDB		
POST	**copy-content/S3/{table}**	--
Take the Postgres {table} and read all lines like above in “/all/postgres/{table} just not returning the data to the caller but writing one object into the bucket (SID) for each object. The object key shall be a dynamically generated UUID So, if there are 7 rows in Postgres you are supposed to have 7 objects in the bucket		

Should you need help:

- See the literature links in Week 2 and 3. 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 be able to answer last minute questions right before the deadline, so please make sure you start the assignment in good time. Piazza will be closed for questions 3 days prior to the submission deadline

Marking:

This programming task has a maximum mark of 25 / 100 points in relation to the entire ACP course.

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

- Proper runnable docker image
- Proper behavior (functionality)

Distribution of marks will be:

Task	**Points max**
Read all from S3 objects and return to caller	3
READ from S3 object and return JSON to caller	2
READ all from DynamoDB table and return JSON	3
READ from DynamoDB by key and return JSON to caller	2
READ all records from PostgreSQL table and return result as JSON records to caller	3
READ data from External Service plus processing of data and return to caller (JSON)	2
Write the result as x entries to DynamoDB	2
Write the result as x objects inside a S3 bucket	2
Write the result as x rows inside a Postgres table	2
Copy the content of a Postgres table line by line to S3 objects (one per line)	2
Copy to DynamoDB	2