Serverless MapReduce on OpenFaaS using MinIO

This project demonstrates the implementation of a serverless MapReduce system using OpenFaaS and MinIO. The project consists of three exercises, each implemented as a MapReduce job, comprising of a Map function and a Reduce function. We use OpenFaaS CLI to create, compile, and deploy serverless functions written in Python3-Debian base image. The communication between functions and data storage is facilitated by MinIO, where we have an initial Input bucket, a Mapper bucket for intermediate results, and a Reduce bucket for final results.

Preparation & Prerequisites

The project was developed in an Ubuntu 18.04 VM in which we had installed the following components:

1. Docker installed and running.
2. OpenFaaS CLI installed.
3. Python 3 and pip installed.
4. Access to MinIO server or a MinIO instance running locally. For the purpose of this project we crated a local instance by downloading the MC client

In MinIO we create an "*input*" bucket where we will upload our initial input (CSV file chunks), a "*mapper*" bucket, which will hold the intermediate results produced by our mapper and a "*reducer*" bucket which will store the final output of the reducer.

Implementation Details

The project has three exercises, and each exercise is implemented as a separate MapReduce job. The Python code for each function is written in the handler.py file, and its dependencies should be listed in the requirements.txt file.

For each exercise we need to:

1. Create a map and a reduce function using the command:

   faas-cli new --lang python3-debian <function name>

   e.g. "faas-cli new --lang python3-debian ex1-map" and
   "faas-cli new --lang python3-debian ex1-reduce"

   The command produces a <function name>.yml file and a <function name> folder which contains the handler.py file, in which we will put our code and the requirements.txt file in which we put the dependencies we need (e.g. minio).
2. Write the code for the map and the reduce function in the handle method of each handler.py file that corresponds to one of the aforementioned methods respectively.
3. Add any required dependencies to the requirements.txt file.

4. Compile the function using the command:

   faas-cli build -f <function name>.yml

5. Deploy the function to the OpenFaaS endpoint:

   faas-cli deploy -f <function name>.yml

5. Create a webhook to connect the Map function with the Input bucket (the following commands are run from inside the folder MC client is downloaded):

   ./mc admin config set minio notify_webhook:<ID> queue_limit="1000" endpoint="http://<VM IP>:8080/function/<function name>" queue_dir=""

   ./mc admin service restart minio

   Where endpoint=The function URL in OpenFaaS.

   Repeat the previous commands to create a webhook for the reduce function.

Workflow Description

1. We start by writing a python script that takes our initial CSV data file and separates it into N equal parts (where N is provided by the user, here 4).
2. For each exercise, we start by writing the Map and Reduce functions in the handler.py file and listing any dependencies in the requirements.txt file.
3. We then compile each function using the OpenFaaS CLI (faas-cli build) and deploy it to the local OpenFaaS endpoint (faas-cli deploy).
4. Next, we create a webhook to connect each Map function with the Input bucket and each Reduce function with the Mapper bucket. The webhook triggers the corresponding function when an object is uploaded to the specified bucket (PUT event).
5. When an object is uploaded to the Input bucket, MinIO sends a request in the form of a JSON object to the corresponding Map function. Essentially, the handle function is called and processes the input (i.e. the request). The intermediate results are stored in separate files in the Mapper bucket.
6. Similarly, when the intermediate results object is uploaded to the Mapper bucket, the corresponding Reduce function is triggered and processes the request. The final results are then uploaded to the Reduce bucket.

   It is worth noting that the JSON request (i.e. the PUT event) that is sent to the function is simply information about the object that was uploaded (filename, date etc) and the bucket in which it was uploaded. Therefore, in the function we only need to get the filename and bucket name in order to request the uploaded object and process it further.