Distributed Systems (2020W) - Proseminar

Homework 05 - Introduction to Abstract Function Choreography Language - Building Scalable FCs

Sashko Ristov

November 2020

DEADLINE: Monday, 23.11.2020, 08 am.

In the previous Homework 04, you were able to distribute the work (calculating Fibonacci numbers) by running multiple concurrent functions. For this specific problem only, you needed to develop an application-specific invoker. With this approach, you need to develop a single Java application (invoker) for each application. However, as we mentioned during lectures, the goal of distributed systems is to build an application-independent middleware.

The aim of this homework is to learn how to build scalable serverless workflow applications (Fucntion Choreographies - FCs) in our AFCL[1] (Abstract Function Choreography Language) using our FC Editor. After building the FC, you will be able to run the whole FC using our *xAFCL* enactment engine.

1 Given codes / tools

All codes are given on the following link[2].

1.1 Input JSON object

Input to the FC (for Section 2.2) is a JSON (`FibonacciFCParInput.json`) which comprises two parameters:

- N - how many Fibonacci numbers need to be calculated with the FC
- f - the distributor which determines how many functions to run in parallel to calculate the Fibonacci numbers.

[1]S. Ristov, S. Pedratscher, and T. Fahringer, "AFCL: An Abstract Function Choreography Language for serverless workflow specification," Future Generation Compputer Systems, vol. 114, pp. 368 – 382, 2021. https://doi.org/10.1016/j.future.2020.08.012

[2]https://github.com/sashkoristov/PSDS2020W/tree/main/H05/Resources

1.2 *xAFCL* Enactment Engine

Provided is our *xAFCL* tool, which allows you to invoke a whole FC on multiple FaaS systems (AWS Lambda, IBM Cloud Functions, Azure Functions, Alibaba Function Compute and Google Cloud Functions) and their regions, following the control and data flow of the FC. *xAFCL* parses the FC and uses jFaaS to invoke functions on specified locations (`ARN`).

Create the `credentials.properties` file with the following structure:

```
aws_access_key=
aws_secret_key=
aws_session_token=
```

1.3 Test FC and test input

Before developing the tasks, you can test the given jar (Section 1.2) with the following test files:

- `helloWorld.yaml`, which consists of a single base function `helloWorld`.
- `helloWorldInput.json`, which should be given as the input to the FC `helloWorld.yaml`
- `helloWorld.py`, a code of the serverless function that reads `name` and returns `Hello <name>`.

You can place all three files (the enactment engine jar file, `helloWorld.yaml`, and `helloWorldInput.json`) in the same folder and use the following command to test the engine:

```
java -jar enactment-engine-all.jar helloWorld.yaml helloWorldInput.json
```

Prerequisites:

- deploy the function `helloWorld.py` on AWS Lambda
- update the file `helloWorld.yaml` (the `value` of the `resource key`) with the location of the deployed function `helloWorld.py`
- `credentials.properties` must be in the same folder as the `xAFCL`.

1.4 Other ways to build an FC

Two other ways exist to develop an FC in AFCL. The first one is a naive approach by developing the FC in a text editor. The other approach is to build your FC with our AFCLCore Java API, presented in the AFCL paper.

2 Development tasks - build FCs from Homework 04

2.1 Development task 1 - Develop a Fibonacci FC for Task 1 of Section 2.1 in Homework 04

You need to develop a simple FC (very similar to helloWorld), which receives the same `input.json` from Homework 04, passes it to the same serverless function `LambdaFibonacci128MB()` of Homework 04. Finally, the output of the serverless function `LambdaFibonacci128MB()` should be passed as an output of the FC.

2.2 Deployment task 2 - Develop a Fibonacci FC for the development task of Section 3.1 in Homework 04

You need to develop a simple FC, which does the same work as Homework 04 (Section 3.1), i.e., distribute Fibonacci functions. For this purpose, you need to develop a simple serverless function `workers`, which gets the given input JSON (`FibonacciFCParInput.json`) from the FC input and generates f arrays with length N/f. Afterwards, you need to run f instances of the function `LambdaFibonacci128MB()` and distribute the output of the function `workers` to each function. For this purpose, use the `distribution` key with value `BLOCK(1)`.

2.3 Evaluation task

Run FCs from sections 2.1 and 2.2 and observe whether the outputs are the same as the ones from your Homework 04.

Note: You may need to adapt (cast numbers) your function because Java differentiates numbers, while for JSON, anything is a number (1, 1.0).

Optional: You can use the following constraint for `dataOuts` of `parallelFor` to aggregate the outputs as a single array:

- name: "aggregation"
value: "+"

3 Upload

In order to pass the task, you have to upload all .java, .JSON, .yaml files used in your solution, as well as a text file containing a description and discussion of the development and evaluation tasks.