SCAD P03 – Workflows with Cloud Functions

This is a one-week lab, i.e. SW 4, submission before SW 5 lecture.

1. Function-based Chaining of Functions

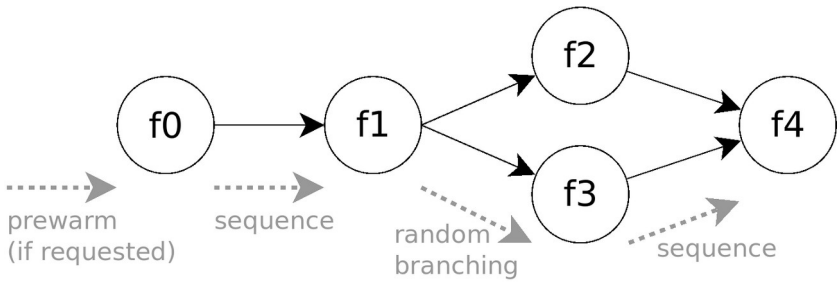
1.1 Preparation

The task is can be realised on top of any FaaS platform as long as you can demonstrate the solution to work. Keep in mind the trade-off between simplicity and security; for this lab, unauthenticated requests are permitted.

Note: This is a pure on-site team task. You will need to finish with a basic solution after 90 minutes by submitting all code into your team's Git repository. A final solution can be submitted within a week after the lab.

1.2 Objective

Imagine that there are no cloud function workflow services. Build a simple self-made "workflow manager" in the form of a cloud function (i.e. "microservice controller" in the Control Flow Methods terminology) which centrally orchestrates invocations of other deployed functions. The orchestration is simplified to the point of only chaining other functions into sequences, transferring the output of one to another, and random branching, along with the ability to benefit from prewarming all candidate functions. The figure below summarises the intended functionality.



1.3 Implementation

Your workflow manager function takes three inputs as "event":

- a declarative textual description of the abstract chain in the form of a list including random branching, e.g. f0-f1-f2|f3-f4 for the exemplary flow above
- a mapping of abstract functions to concrete implementations, e.g. f1:http://...
- a boolean pre-warming flag that, if true, causes all functions to be pinged first

The workflow manager function should parse the input, orchestrate the invocations, and return the output of the last function, along with statistical information: How much time was spent on each individual function? If an error occurs, which function failed and why?

If the pre-warming flag is set, the manager function should “ping” each listed function with an empty message. The functions should in this case return an empty message.

For the individual functions, implementing dummy code that takes a few seconds to execute on non-empty messages (but immediately returns on empty ones) is sufficient for the task. Think about the need for idempotent function invocation semantics as condition for the pre-warming.

Finally, observe the timing behaviour and invocation count. Can you provoke a cold-start situation where the pre-warming shows actual effects? Are conditionally branched functions invoked less? Is prewarming more helpful with longer sequences? Did your workflow manager function ever time out? Gather the necessary statistics and describe your findings along with the code in a simple observation text file.

1.4 Advanced implementation

You can optionally figure out how to do a parallel invocation of two functions if time permits using a modified branching syntax (f1-f2&f3-f4). In this case, f4 would need the merged output of f2 and f3 as its input, or can work on one of the outputs. Furthermore, you can think of a catalogue of re-usable transformation functions or internal filters, to e.g. a helper function for data conversion that allows to pass on a single string to f2 out of a dictionary returned from f1.