FAASIN: Bringing UNIX pipes to serverless cloud function pipelines

Mathieu Bacou (mathieu.bacou@telecom-sudparis.eu), ACMES
Keywords: cloud, Function-as-a-Service, systems programming, containers, performance

Context

Cloud computing, or in other words, renting remote computing resources, is the mainstream way of deploying any application. This deployment model had a feedback effect on the applications' *architectures*: they are more and more "cloud native". The newest paradigm in cloud applications is Function-as-a-Service (FaaS): an application's features are served by chaining and replicating elementary functions. However, *chaining function executions in a pipeline is not efficient*, because passing output data is extremely slow. Indeed, the Serverless nature implies that all data, in order to persist beyond the execution of one function, must be copied out of the function that produced it, and written to an external service (be it block or object storage, message queue, etc.). Data movement between the FaaS platform and this remote service is slow by nature, and constitutes a big drawback to porting many applications to the FaaS model.

Goals

The goal of the internship is to explore the following novel way of passing data to the next function in line: to t*ransfer data between functions by streaming through a local in-memory pipe*.

This is very similar to UNIX pipelines of commands: when a command line such as `grep pattern | wc -l` is executed, the standard output stream (`stdout`) of the command `grep` is piped (i.e., sent through a pipe located in local memory) to the standard input stream (`stdin`) of the command `wc`. The implementation will provide functions with new file descriptors that correspond to the input and the output of their step in the pipeline. The input stream from the function chain is called Function-as-a-Service input (`faasin`), and the output stream is called Function-as-a-Service output (`faasout`); thus the name of the project: FAASIN (pronounce "phase in").

The advantage of this method are:

- actual pipelining of functions: all steps of the pipeline can run in parallel, without waiting for the output of the previous one;
- huge performance boost: data is no longer marshaled, and remains in-memory;
- easy for the developer: pipes are a well-known and well-integrated way of passing data.

For now, the steps and goals of the internship to implement FAASIN are:

- implement the general mechanism of opening `faasin` and `faasout` at a low level (in Docker containers, as commonly used by FaaS platforms);
- implement it in a FaaS platform: integrate FaasIn in the pipelining of functions;
- implement it at the function runtime and at the function programming language level (i.e., at the actual programming level of FaaS);
- demonstrate the performance and programmability gains on real-world applications.