Zurich University
of Applied Sciences

School of
Engineering

SCAD P04 – Building Complex Applications with Cloud Functions

This is a two-weeks lab, i.e. SW 5, submission before SW 7 lecture.

1. Application Planning

1.1 Preparation

After building only individual functions with limited holistic functionality, this lab combines many of the previous lecture and lab topics into a usable microservice-based serverless application. The application needs to fulfil a certain number of requirements, with some flexibility depending on your team's preferences, in addition to interactive usefulness.

The functionality/use case of the application is entirely chosen by the team, but needs to be input-dependent, i.e. the behaviour depends on small/large input. The advice is to start with minimalistic functionality in a single domain, and to re-use previous work. Similarly, programming language(s) and FaaS provider(s) are chosen freely by the team, i.e. whether a polyglot and/or cross-cloud implementation is provided.

Note: This is a two-weeks task, encompassing two on-site labs. The final solution must be available through Git before the next lab in semester week 7 is going to start. Nevertheless, it is advised to commit the emerging solution regularly.

1.2 Requirements selection

Several individual features are offered, of which a subset are required to pass this lab. Thus, choose any out of the following depending on the team size – one per team member:

R1: Deployment framework. Package your application into the Serverless Framework, Vercel, Claudia.js, SAM or a similar suitable deployment mechanism. {See V03}
R2: Workflow. Orchestrate at least three distinct functions with a production-grade workflow engine, using Step Functions, Cloud Composer, Fission Workflows or an alternative language/system dedicated to programming workflows. (You may have to arrange a custom FaaS provider account, or use the ZHAW-provided Fission instance.) {See V04}
R3: Advanced FaaS. Use layers, global static variables, dependency injection or other provider-specific features to demonstrate optimisation with and without that feature. {See V02}

R4: Inter-process communication. Set up a long-running rendezvous service (e.g. web framework running on PaaS) to convey the status of function instances to other instances and to synchronise functionality.
R5: FaaSification. Generate a subset of your functions from monolithic code with annotations, using a research or production-level FaaSification tool. {See V03 and faasification.com}
R6: Self-optimisation. The application contains calibration logic to reduce economic losses (temporal, spatial) or to achieve higher performance. This encompasses «warmup pinging» or self-awareness through the context object if supported by the provider. {See V02 and V04}
R7: Content trigger. A function should be triggered by a native event within the cloud platform. For instance, by an upload event of a file, a function returns the modified file, or by a cron event, a regular invocation is provided.
R8: Private FaaS. Instead of using commercially offered services, run your own (e.g. within a container on APPUiO, on a ZHAW Cloudlab VM or even in raw form on your notebook). Candidates are e.g. Fission, Kubeless, OpenFaaS and OpenWhisk.
R9: Adaptive scaling. Deploy two flavours of your function that perform heavy processing and route input to one of the two depending on the estimated complexity of the input (file size, or other indicators). Check if adaptive scaling helps performance-wise and what the cost implications are.

1.3 Deliverables

The solution consists of the code, a deployed demonstration instance as well as an overview documentation with a detailed architecture diagram which references concrete implementation artefacts and FaaS interfaces/platform services. The covered requirements (R1..R9) also need to be marked in the appropriate places of the diagram. A few measurements should be provided, especially for R3/R6/R9 (comparison) and for data input-dependent behaviour.

The application functionality needs to be demonstrated interactively, using a command-line interface or a web interface, including the ability to set parameters which influence the application execution.