Lecture 8.3 – Serverless (Function as a Service (FaaS))

Luca Morandini
Data Architect – AURIN Project
University of Melbourne
luca.morandini@unimelb.edu.au

Outline of the Lecture

- **Part 1:** What is FaaS
  - Serverless / FaaS defined
  - Why functions?
  - FaaS services and frameworks

- **Part 2:** More About Functions
  - Functions with side-effects
  - Stateful/stateless functions
  - Synchronous/Asynchronous functions

- **Part 3:** OpenFaaS Workshop
  - OpenFaaS Architecture
  - Cluster Set-up
  - Function Deployment

Part 1: What is Function as a Service (FaaS) ?

Disambiguation

- FaaS is also know as *Serverless computing* (more catchy, but less precise)
- The idea behind Serverless/FaaS is to develop software applications without bothering with the infrastructure (especially scaling-up and down as load increases or decreases)
- Therefore, it is more *Server-unseen than Server-less*
- A FaaS service allows functions to be added, removed, updated, executed, and auto-scaled
- FaaS is, in a way, an extreme form of *microservice architecture*

Why Functions?

- A *function* in computer science is typically a piece of code that takes in parameters and returns a value

- Functions are the founding concept of *functional programming* - one of the oldest programming paradigms

- Functions are - or should be - free of *side-effects*, *ephemeral*, and *stateless*, which make them ideal for parallel execution and rapid scaling-up and -down, hence their use in FaaS (more on this topic later)

Why FaaS?

- Simpler deployment (the service provider takes care of the infrastructure)
- Reduced computing costs (only the time during which functions are executed is billed)
- Reduced application complexity due to loosely-coupled architecture

FaaS Applications?

- Functions are triggered by events
- Functions can call each other
- Functions and events can be combined to build software applications
- For instance: a function can be triggered every hour (say, to compress log files), or every time disk space on a volume is scarce (to remove old log files), or when a pull-request is closed in GitHub, or when a message is stored in a queue
- Combining event-driven scenarios and functions resembles how User Interface software is built: user actions trigger the execution of pieces of code

FaaS Services and Frameworks

- The first widely available FaaS service was Amazon's AWS Lambda. Since then Google Cloud Functions (part of Firebase) and Azure Functions by Microsoft

- All of the FaaS above allow functions to use the services of their respective platforms, thus providing a rich development environment

- There are several open-source frameworks (*funtainers* - or *fun*ctions conta*iners*) such as *Apache OpenWhisk*, *OpenFaas*, and *Kubernetes Knative*

- The main difference between proprietary FaaS services and open-source FaaS frameworks is that the latter can be deployed on your cluster, peered into, disassembled, and improved by you.

Part 2: More About Functions

Side-effect Free Functions

- A function that does not modify the state of the system is said to be *side-effect free* (for instance, a function that takes an image and returns a thumbnail of that image)

- A function that changes the system somehow is not side-effect free (for instance, a function that writes to the file system the thumbnail of an image)

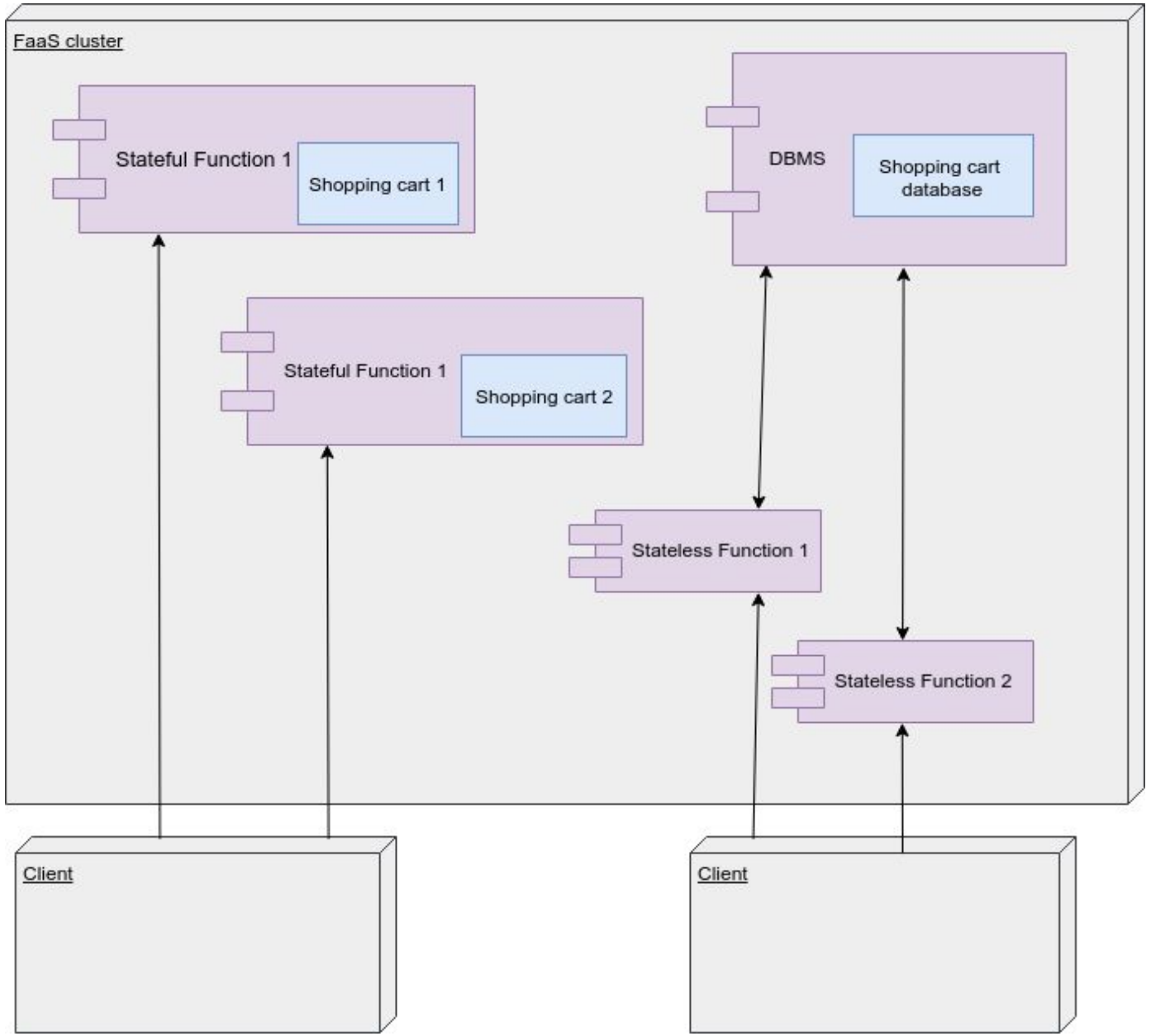
- Side-effect free functions can be run in parallel, and are guaranteed to return the same output given the same input

- Side-effects, however, are almost inevitable in a relatively complex system. Therefore consideration must be given on how to make functions with side effects run in parallel, as typically required in FaaS environments.

Stateful/Stateless Functions #1

- A subset of functions with side-effects is composed of *stateful functions*
- A stateful function is one whose output changes in relation to internally stored information (hence its input cannot entirely predict its output), e.g. a function that adds items to a “shopping cart” and retains that information internally
- Conversely, a *stateless function* is one that does not store information internally, e.g. adding an item to a “shopping cart” stored in a DBMS service and not internally would make the function above stateless, but not side-effect free.
- This is important in FaaS services since there are multiple instances of the same function, and there is no guarantee the same user would call the same function instance twice

Stateful/Stateless Functions #2



Synchronous/Asynchronous Functions

- By default functions in FaaS are *synchronous*, hence they return their result immediately (or almost so)

- However, there may be functions that take longer to return a result, hence they incur timeouts and lock connections with clients in the process, hence it is better to transform them into *asynchronous functions*

- Asynchronous functions return a code that informs the client that the execution has started, and then trigger an event when the execution completes

- In more complex cases a *publish/subscribe pattern* involving a queuing system can be used to deal with asynchronous functions

Part 3: OpenFaas Workshop

Introduction to OpenFaaS

- OpenFaaS is an open-source framework that uses Docker containers to deliver FaaS functionality
- Every function in OpenFaaS is a Docker container, ensuring loose coupling between functions (functions can be written in different languages and mixed freely)
- Functions are passed a request as an object in the language of choice and return a response as an object
- OpenFaaS can use either Docker Swarm or Kubernetes to manage cluster of nodes on which functions run
- By using Docker containers as functions, OpenFaaS allow to freely mix different languages and environments at the cost of decreased performance, as containers are inherently heavier than threads. However, by using a bit of finesse, a container with a single executable, can weight only a few MBs (check out `https://hub.docker.com/_/scratch`)

Calling Functions Defined in OpenFaaS

- Calling a function in OpenFaaS is done via POST HTTP request, as in:

```
curl -XPOST\
 "http://0.0.0.0:8080/function/wcmp"\
 --data "Lorem ipsum dixit"
```

(You can use either GET or POST methods, but POST allows data of arbitrary size to be passed in the body.)

- Functions are not grouped into namespaces (still in alpha). You can get around this limitation with a proxy server acting as a facade (i.e. converting the `/maps/select/byRectangle` path to request to a function named `maps__select__byRectangle`)

Node.js Function in OpenFaaS

```
module.exports = (context, callback) => {
  try {
   const result = {
         status: "You said: " +
            JSON.stringify(context)
    };
  } catch (err) {
    callback(err, null);
  }
  callback(null, result);
}
```

(Better access to the HTTP request can be had by using the `node10-express` template, which is still in beta.)

Python Function in OpenFaaS

```
import requests
import json

def handle(req):
    result = {"found": False}
    json_req = json.loads(req)
    r = requests.get(json_req["url"])
    if json_req["term"] in r.text:
        result = {"found": True}

    print json.dumps(result)
```

Auto-scalability and OpenFaaS

- OpenFaaS can add more Docker containers when a function is called more often, and remove containers when the function is called less often

- The scaling-up (and down) of functions can be tied to memory or CPU utilization as well (currently only on Kubernetes-managed clusters though)

Workshop repository

- Use git to clone the repository at:
  `https://github.com/AURIN/comp90024`

- Go to the `openfaas` directory

- Follow the instructions in the `README.md`

In addition, if you want to see an example of an application built on OpenFaaS, you may have a look at: `https://github.com/aurin/pgfaas`