Workshop 9 – Serverless (Function as a Service (FaaS))

Luca Morandini
Cloud Architect – Melbourne eResearch Group
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:** Fn Workshop
  - Introduction to Fn
  - Installation
  - 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): the service provider does it for you
- 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 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 merged 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

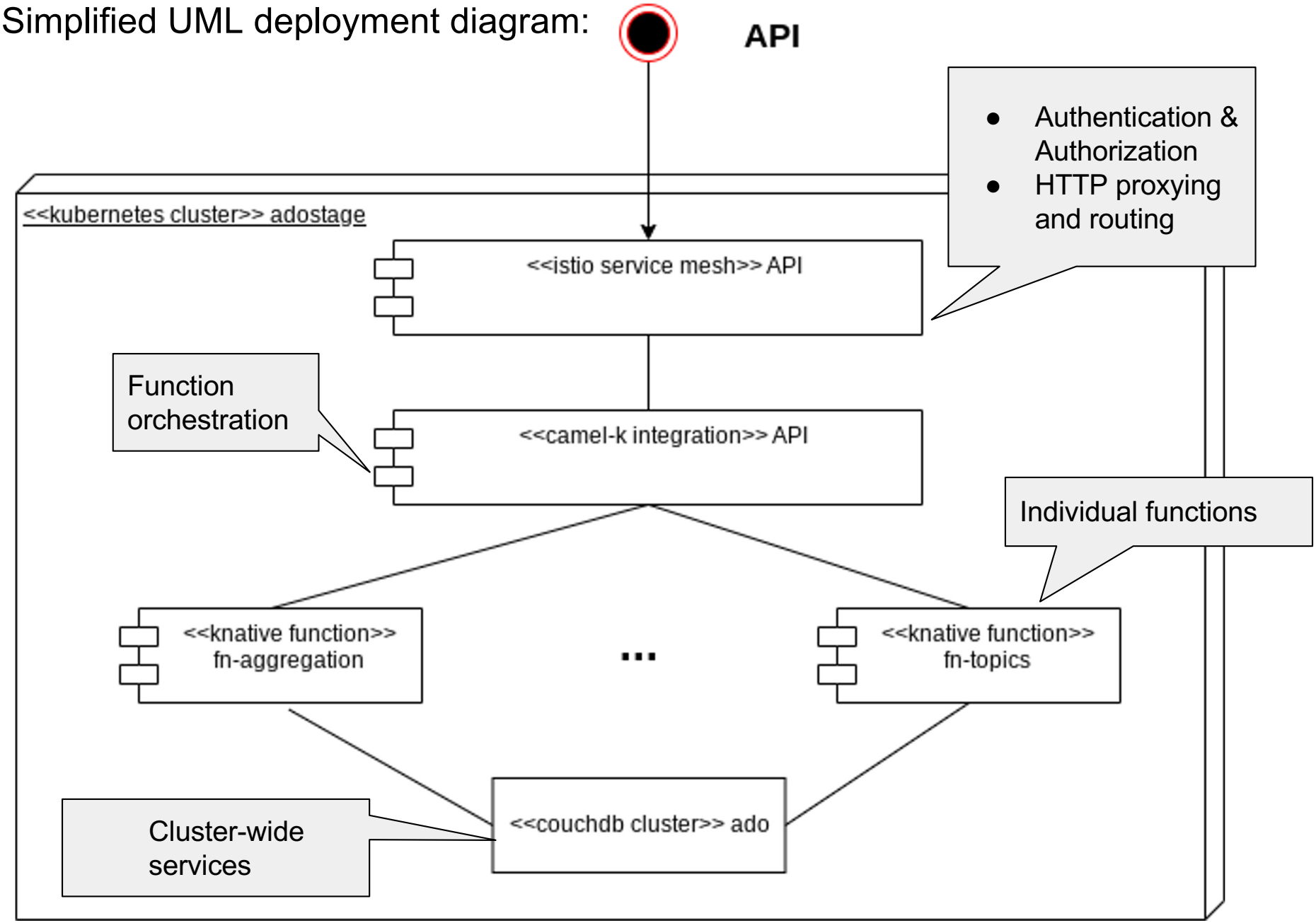
Serverless Application: ADO #1

- The ADO backend is build on a Kubernetes cluster. Kubernetes is a container orchestration platform that allows to distribute containers across a number of VMs

- ADO use a few frameworks:
  - Knative: serverless framework for deploying and managing the life-cycle of functions including auto-scaling
  - Camel-K: orchestration of Knative functions, such as function composition (flow), caching of function results, message queues
  - Istio: a service mesh that takes care of authentication and authorization, HTTP header management, routing

Serverless Application: ADO #2

- Simplified UML deployment diagram:



Serverless Application: ADO #3

- The API is defined as an Open API specification:

```
/data/collections/{collection}/summary:
   get:
     summary: Return information about a collection
     operationId: analysisAggregateSummary
     parameters:
       - name: collection
         in: path
         description: collection name
         required: true
```

- A Camel-K integration routes the `operationID` to actual Knative functions:

```
from("direct:analysisAggregateSummary")
   .to("knative:endpoint/fn-aggregation")
```

- Functions are Kubernetes pods (sets of related containers):

```
fn-aggregation-00001-deployment-cdff847d-mnj6n                    2/2
Running   0          20d
…
fn-textsearch-00001-deployment-6c6c589f69-hjqxl                   2/2
Running   0          20d
fn-topics-00001-deployment-6fdbc7b556-9wkpp                       2/2
Running   0          20d
```

- Functions are independent of each other, the API specification is written in YAML, and the Camel-K integration routing could be written in XML/Java/JavaScript/etc. Therefore a developer team can mix and match different languages, libraries, and even operating systems without worrying about the deployment on the actual computing resources

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*, Fn, and *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)

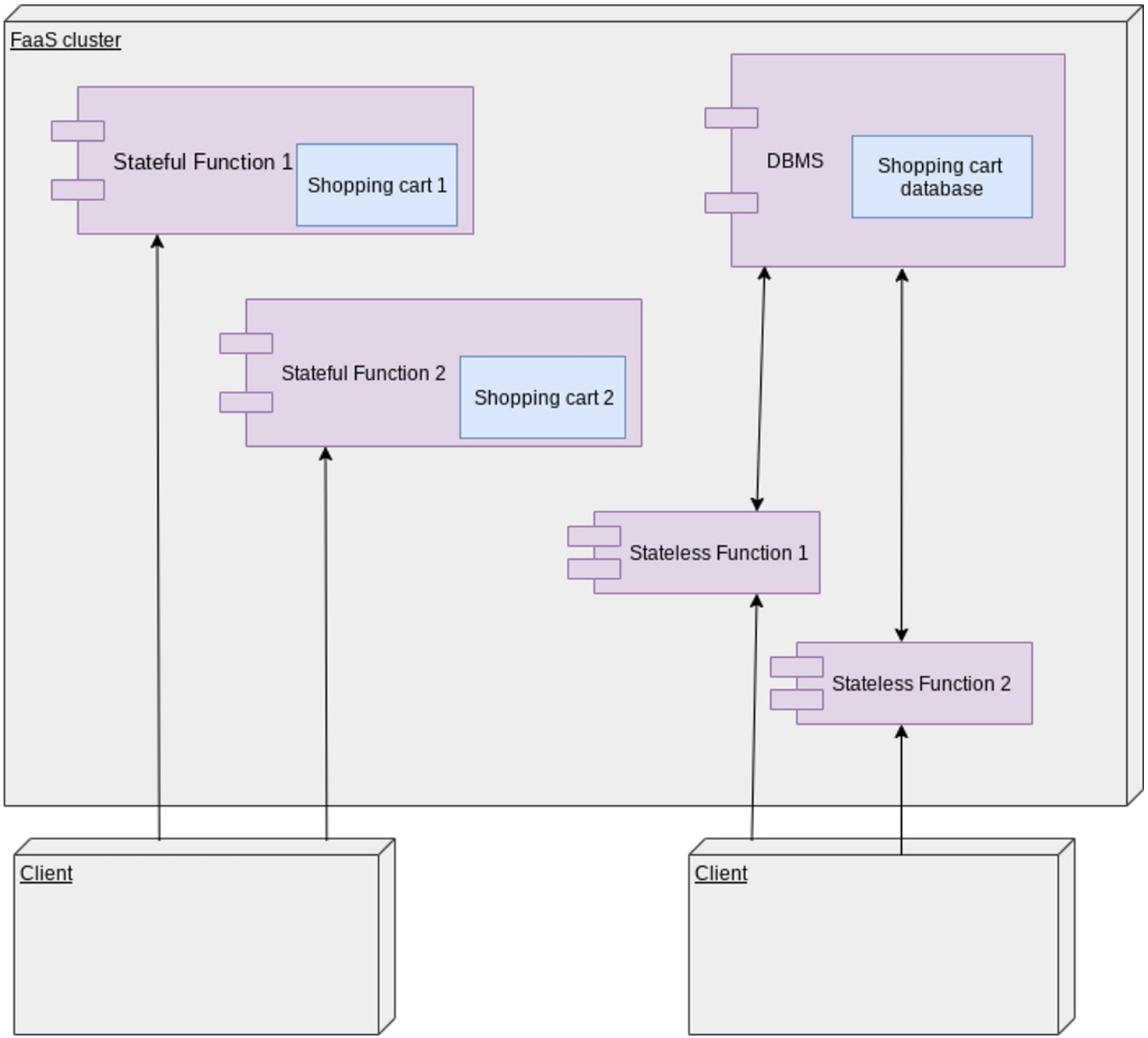
- Side-effect free functions can be run in parallel, and are guaranteed to return the same output given the same input

- 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: Fn Workshop

Introduction to Fn

- Fn is an open-source framework that uses Docker containers to deliver FaaS functionality

- Every function in Fn is a Docker container, ensuring loose coupling between functions (functions can be written in different languages and mixed freely)

- By using Docker containers as functions, Fn 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`)

...More on Fn

- Fn is the technology behind Oracle Functions (the serverless service of Oracle Cloud)
- Fn could be deployed on Kubernetes to manage cluster of nodes on which functions are run
- Fn allows both synchronous and asynchronous functions
- With Fn Flow, functions can be composed efficiently
- Fn can add more Docker containers when a function is called more often, and remove containers when the function is called less often

Calling Functions Defined in Fn

- Calling a function in Fn can be done via POST HTTP request, as in:

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

- Functions can be grouped into an “App”, so that they are arranged into hierarchical URLs (i.e. `/myapp/myfunction`). By default the paths follow the directory structure of the source code

- Function can be triggered by HTTP requests, but also by other classes of events, such as new messages in message queues (publish/subscribe)

Node.js Function in Fn

```
const fdk = require ('@fnproject/fdk');

fdk.handle ( (input) => {
 let counts = {};
 input.toLowerCase ()
   .split(/\W+/)
   .filter ((w) => {
     return w.length > 1;
   })
   .forEach ((w) => {
     counts[w] = (counts[w] ? counts[w] + 1 : 1);
   });
 return counts;
})
```

Python Function in OpenFaaS

```
import io
import json
import logging
from fdk import response

def handler(ctx, data: io.BytesIO = None):
    try:
        counts = dict()
        logger = logging.getLogger()
        for word in data.getvalue().split():
            counts[word.decode('utf-8')]=
                    (counts.get(word.decode('utf-8')) or 0) + 1
    except (Exception, ValueError) as ex:
        logging.getLogger().info('error: ' + str(ex))
    return response.Response(
        ctx, response_data=json.dumps(counts),
                headers={"Content-Type": "application/json"}
    )
```

- "ctx" is a Context object than contains metadata about the function and its invocation: additional configuration parameters, the HTTP method, HTTP headers, etc.

Workshop repository

- Use git to clone the repository at: `https://gitlab.unimelb.edu.au/feit-comp90024/comp90024`
- Go to the `fn` directory
- Follow the instructions in the `README.md`