Function-as-a-Service Orchestration in Fog Computing Environments

Gaetano Francesco Pittalà, Gianluca Davoli,
Davide Borsatti, Walter Cerroni, Carla Raffaelli
University of Bologna, Italy
Email: gaetano.pittala2@studio.unibo.it,
{gianluca.davoli, davide.borsatti, walter.cerroni, carla.raffaelli}@unibo.it

***Abstract*—With the establishment of the Everything-as-a-Service (XaaS) paradigm for service provisioning, coupled with the increasingly-demanding requirements imposed by modern network services, the need for a XaaS-aware orchestration system able to cope with a heterogeneous infrastructure, such as the one of Fog Computing environments, is evident. In this work, we describe the working principles and implementation aspects that allow the orchestration of services offered according to the Function-as-a-Service (FaaS) model. The live demonstration will showcase the ability of the system to deploy this kind of services on a suitable test bed, with comments on the procedure and the performance.**

I. Introduction

The advent of revolutionary communication and service provisioning models, such as the Internet of Things, brought about a massive number of connected devices, along with an ever-increasing necessity for real-time applications. Now more than ever before, network infrastructures need to manage resources in an efficient, scalable, and smart way.

In order to cope with such specifications, in modern telecommunication systems resources are often displaced and physically located in proximity to the users, causing a shift from the widely-adopted Cloud Computing (CC) paradigm [1] to the rising Fog Computing (FC) [2] and Edge Computing (EC) [3] ones. Specifically, FC mainly aims at reducing service fruition latency by making use of computing power residing in portions of the infrastructure which are closer to the user. In such scenarios, where computation may be distributed over a large number of heterogeneous interconnected nodes, an orchestration system able to manage the activation of services on a dynamic and heterogeneous infrastructure is clearly needed [4]. The trend of providing "everything as a service" (XaaS), borrowed from the CC paradigm, depicts a promising scenario where the development and deployment of software applications as services is supported by suitable architectures [5].

These scenarios also caught the attention of standardization bodies, such as European Telecommunications Standards Institute (ETSI), that is standardizing Multi-access Edge Computing (MEC), an emerging system architecture where CC applications are extended to the edge of networks leveraging the computing power of mobile base stations and edge data centers.

Several architectures are proposed for orchestrating services in FC environments. For instance, [6] presents a FC orchestration architecture adopting a two-stage multi-objective optimization method to select the best fitting node to deploy the service. Similarly, [7] presents a system that lays its foundations on MEC, and it is able to merge various "simple" services into composite ones. However, these works are not concerned with details such as the ones we cover here, where we present our vision of activation and fruition of lightweight services offered through the network infrastructure , along with implementation details.

Along to the mentioned solutions, FORCH (short for Fog ORCHestrator) [8] stands out as a XaaS-aware service-based orchestration system specifically designed for FC scenarios, supporting the Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) models.

Along with the mentioned declinations of the XaaS paradigm, the FaaS model is presented as a means to introduce further cost-efficiencies, reduce configuration and management overheads, and rapidly increase the ability of an application to achieve better performance on FC/EC/CC scenarios [9]. FaaS is an event-driven computing execution model in which application logic is deployed in a virtualized entity (e.g., a container) that is fully managed by the platform and executed as needed. Moreover, the FaaS model is mainly based on serverless computing, that is a partial realization of an event-driven ideal in which applications are defined by actions and the events that trigger them [10].

In this work, we show how we can exploit, adapt and enhance the functionalities of FORCH to support the orchestration of FaaS elements in a FC environment. After implementing the required software components and devising a proper test bed for a practical validation, we prepared a live demonstration of the activation of a sample FaaS element, as well as of the fruition of the service it offers. Section II gives an overview of the reference system architecture, while Section III describes the steps taken for the activation of a service in the reference context, and Section IV presents the test bed and the planned demonstration.

II. System Architecture

To understand how FORCH was employed to orchestrate FaaS components, a brief overview of the architecture adopted

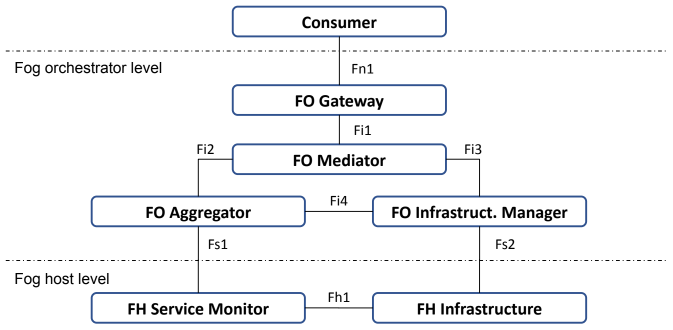

Fig. 1. FORCH reference architecture.

in [8] is necessary. The overall system architecture is shown in Fig. 1, and it can be decomposed into four main functional entities, namely Gateway (GW), Mediator (ME), Aggregator (AG), and Infrastructure Manager (IM), in order of logical proximity to the user.

The GW is the entity in charge of handling the communication between the orchestrator and the users, exposing a suitable REST API. The ME hosts the core functionality of the orchestrator. It is the entity that processes service activation requests, taking decisions based on current service status and resource availability information gathered from the nodes. The scope of the AG is to collect information regarding the status of services activated on the nodes and provide them to the ME. The IM manages the resource monitoring on the underlying infrastructure (i.e., the available nodes), as well as the interactions needed to activate services on it.

These functional entities were exploited to add new capabilities to FORCH, mainly concerning the support of the FaaS paradigm. The fundamental working principles and mechanisms of the orchestration system were left untouched, but some of its components were enhanced by acting on its source code[1]. Firstly, the system needed an abstraction to provide continuous service availability to the user. To do that, we included a new REST API endpoint in the GW, specific for FaaS-related requests. Secondly, we needed a way to treat FaaS elements differently from the other services the system supported before. To this aim, we acted on the ME, by replicating part of the code for service activation and using it to enable the system to activate services without an explicit service activation request from the user. The working principles and the technical aspects are discussed in greater detail in Section III.

III. Mechanisms and implementation

Before moving on, it is useful to introduce some terminology. In the context of this work, a service *activation* may happen in one of two ways. In the former, called *allocation*, a node already offers the service and the orchestration system makes it available for the user requesting it. While in the latter, called *deployment*, the service is not already offered by a node, so the orchestration system needs to use a virtualization engine to instantiate it before giving access to the user.

With the service models previously supported by FORCH (i.e., S/P/IaaS), users needed to submit a specific service activation request to the orchestrator, which would attempt to satisfy it and then inform the user on how to reach the activated service. This was carried out as a sequence of operations, briefly recalled here. The first interaction a user is expected to have with the orchestration system is the one to discover the list of currently available services, with a `GET` request to the `/services` endpoint on the GW. Then, the user would submit a `POST` request to the same endpoint, specifying the desired service to be activated. Then, the request would be processed by the ME, which would use the information on current service status and resource availability to decide where to activate the service, if possible, before informing the user about the outcome of the process.

This procedure must be adjusted for services activated according to the FaaS model. In fact, FaaS elements should not be activated explicitly by the user, who should be kept unaware of the service activation operations, in line with the serverless principle. In the FaaS case, the user should be able to directly make use of the service, with the orchestration system handling the activation process, according to the event-driven rationale. The system should also consider that a service is not assigned to a particular user, as it is with the other service models. This way, a user that wishes to access a service does not need to formulate an explicit service activation request, but rather they can immediately interact with the service itself through the endpoint provided by the orchestrator. This is the point where the enhancements presented in this work play a major role. The endpoint provided by the orchestrator for every FaaS element, in fact, might be the one residing on the node actually offering the service, or it might be just a symbolic one, hosted by the orchestrator. This depends on the current status of the service to be offered according to the FaaS model. If such service is already active on a node, the orchestrator will provide the user with a way to reach the node directly. Conversely, if the service is defined but not currently active on any node, the orchestrator needs to step in and provide the user with a symbolic endpoint (actually hosted by the orchestrator itself) to make use of the service. In the latter case, the user will send its service fruition request to the symbolic endpoint, triggering the service activation operations, transparently handled by the orchestrator, which will then reply to the user with a `Redirect` message including the endpoint of the node actually offering the service. The information a user can retrieve with a `GET` request to the enhanced GW is shown in Fig. 2.

To sum up, in order to make use of a service deployed as FaaS, the user just needs to discover the list of services offered by the orchestrator. Then, every FaaS element is either already active, and the user can access its service right away, or it will be activated the first time someone tries to use the service, remaining available for following requests. This fits in the serverless and event-driven paradigms associated with

[1]Available at: https://github.com/giditre/unibo_gaucho

```
Request method: GET
Request URL: http://192.168.64.117:6000/services
Response code: 200
Response JSON: {
  "message": "Found 3 service(s).",
  "services": [ "APP992", "FVE001" ],
  "faasservices": [
    "LAF001 192.168.64.117:6000/faasservices/LAF001",
    "LAF002 192.168.64.121:5000/transcode",
  ]
}
```

Fig. 2. The user requests the list of available services, and the orchestration system replies with separate lists of S/P/IaaS and FaaS elements. APPxxx and FVExxx refer to SaaS and IaaS elements, while LAFxxx refers to FaaS elements, respectively.

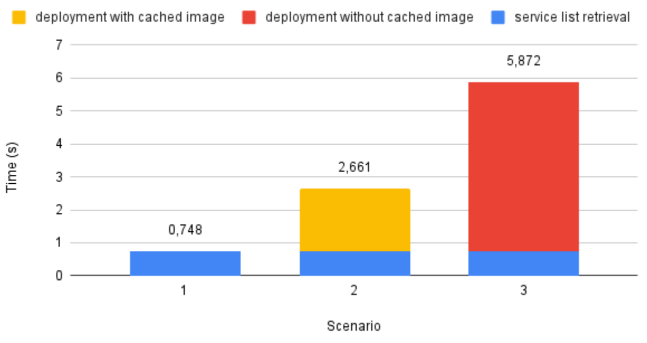

Fig. 3. Time needed for the user to begin service fruition of a FaaS element.

the FaaS model.

As a side note, it is worth specifying that we are not considering security aspects here (including authentication issues) as they fall outside the scope of this work. For the sake of simplicity, we assume that anyone who can interact with the system has already gone through an authentication process, handled by an external entity.

IV. Demonstrator Description

We prepared a test bed that is logically coherent with the architecture depicted in Figure 1. A Virtual Machine (VM) with 4 vCPUs and 4 GB of RAM hosts all FORCH software components, which were developed as separate Python programs meant to be run independently and capable of communicating with each other via REST APIs. The Fog nodes are deployed on different hardware platforms, including two additional VMs, an Intel NUC MiniPC equipped with a 4-core 8th-gen Intel i7 processor and 16 GB of RAM, and a RaspberryPi Single Board Computer, model 3B+, equipped with a 4-core ARMv8 processor and 1 GB of RAM. We use Docker as our virtualization engine of choice, and Prometheus as the monitoring system.

We chose to employ a simple image transcoder as the service to be activated according to the FaaS model. We built a suitable Docker image and uploaded it to the Docker Hub[2]. We performed some experimental evaluations to get an idea of the amount of time needed by the user to retrieve the endpoint to be used for service fruition. The results are shown in Fig. 3, differentiated into three scenarios. In the first one, the service is already active on a node, therefore all the user needs to do is to retrieve the list of active services. In the second case, the service is not active on a node yet, but the node already holds the image to use to activate it. The third case is similar to the second one, but the node needs to download the image from the remote repository, making the required time very much dependent on the connection quality. Keep in mind that, although these values were obtained as the mean over a large number of samples, they are only meant to be indicative of the order of magnitude of the times at play here.

[2]https://hub.docker.com/r/giditre/gaucho-transcode

The demonstration will showcase the ability of the enhanced orchestration system to manage services activated according to the FaaS model, in line with the procedures described in the previous sections. We will act as a user interacting with the orchestrator through its REST API, and then make use of the service on the provided endpoint, demonstrating the efficacy of the approach.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility,” *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X08001957

[2] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, “Fog computing: Survey of trends, architectures, requirements, and research directions,” *IEEE Access*, vol. 6, pp. 47 980–48 009, 2018.

[3] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, “Edge computing: A survey,” *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18319903

[4] N. F. Saraiva de Sousa, D. A. Lachos Perez, R. V. Rosa, M. A. Santos, and C. Esteve Rothenberg, “Network service orchestration: A survey,” *Computer Communications*, vol. 142-143, pp. 69–94, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366418309502

[5] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, “Everything as a service (xaas) on the cloud: Origins, current and future trends,” in *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 621–628.

[6] N. Morkevicius, A. Venčkauskas, N. Šatkauskas, and J. Toldinas, “Method for dynamic service orchestration in fog computing,” *Electronics*, vol. 10, no. 15, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/15/1796

[7] D. Borsatti, G. Davoli, W. Cerroni, and C. Raffaelli, “Enabling Industrial IoT as a Service with Multi-Access Edge Computing,” *IEEE Communications Magazine*, vol. 59, no. 8, pp. 21–27, 2021.

[8] G. Davoli, W. Cerroni, D. Borsatti, M. Valieri, D. Tarchi, and C. Raffaelli, “A Fog Computing Orchestrator Architecture with Service Model Awareness,” *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[9] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, “A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms,” in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 162–169.

[10] G. McGrath and P. R. Brenner, “Serverless computing: Design, implementation, and performance,” in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017, pp. 405–410.