DistributedFaaS: Execution of Containerized Serverless Applications in Multi-Cloud Infrastructures

Adbys Vasconcelos, Lucas Vieira, Ítalo Batista, Rodolfo Silva and Francisco Brasileiro
Departamento de Sistemas e Computação, Universidade Federal de Campina Grande, Brazil
{adbys, lucas.vieira, italobatista, rodolfoams, fubica}@lsd.ufcg.edu.br

Keywords: Distributed Computing, Cloud Federation, Function-as-a-Service.

Abstract: The adoption of cloud computing is continuously increasing due to the attractiveness of low costs of infrastructure acquisition and maintenance, as well as having virtually infinite resources available for scaling applications based on demand. Due to the increasing interest in this topic, there is a continuos search for better, more cost-effective ways to manage such infrastructures. One of the most recent steps was taken by the definition and development of Serverless computing, a.k.a. Function-as-a-Service (FaaS). FaaS is a cloud computing service model where developers can deploy functions to a cloud platform and have them executed based either on the triggering of events by other services, or by making requests directly to an HTTP(S) gateway, without having to worry about setting up the underlying infrastructure. In this paper, we propose an architecture for deploying FaaS platforms in hybrid clouds that can be composed by multiple cloud providers. This architecture aims at enabling privately deployed FaaS platforms to perform auto-scaling of resources (virtual machines) in a distributed infrastructure, while considering the scenario where the users of such platform are scattered around the globe. This allows the execution of requests in servers geographically located as close as possible from the client, with benefits to both the clients and the service providers.

1 INTRODUCTION

Cloud Computing has become one of the preferred means of hosting Information Technology services (RightScale, 2018). In the last decade, multiple deployment models have been proposed for such infrastructures. Initially, cloud computing was only possible via the use of services offered by big cloud providers, namely *public clouds*. Then, the rise of new technologies, similar to the ones used by big cloud providers, enabled the deployment of *private clouds*, using companies' own infrastructures. More recently, an increasing demand for hybrid cloud deployments has emerged. Such kind of cloud deployments integrates a number of different public and/or private providers in a single environment, creating multi-cloud infrastructures. A recent study conducted by RightScale indicates that 81% of the respondents used a multi-cloud approach (10% using multiple private clouds, 21% using multiple public clouds, and 51% using a hybrid cloud formed by public and private cloud providers).

Likewise, different service models for deployment in cloud computing infrastructures have been proposed. Initially those models were restricted to SaaS – Software as a Service, PaaS – Platform as a Service, and IaaS – Infrastructure as a Service, but nowadays almost anything can be provided as a service and offered by cloud computing providers. One of those service models particularly has drawn interest from a large number of researchers and cloud computing users, FaaS – Function as a Service.

The main goal of a FaaS provider is to allow developers to deploy **functions** in highly scalable and highly available infrastructures, without having to deal directly with the underlying physical and virtual infrastructure that will host and execute such functions. Developers who want to use the FaaS service can then focus on writing highly-specialized code that performs a single task. The result is a much shorter development cycle, due to the fact that the code base will be very small and much easier and faster to be tested in such environments. Additionally, this model also enables the application to adopt the widely used pay-per-use billing model, using a very fine-grain, based on the number of individual requests made.

The FaaS service model has been made available by the big cloud providers since 2014, but it has gained a lot of interest also in the private clouds scenario. Because of that, multiple initiatives to provide

FaaS support were created, making it possible to deploy FaaS-based services on private and public cloud computing infrastructures.

FaaS architectures usually rely on the existence of a main component – API Gateway – that works as an entry point for the entire platform, *i.e.*, it receives requests to deploy, update or remove functions and executes them when an event is triggered or an execution request is made to it. The API Gateway is also responsible for managing the computational infrastructure where the functions will be executed and to increase or decrease the allocated resources for executing the functions, depending on the demand.

Considering the possibility that the user base of a FaaS-based service is spread around the globe, it makes sense that the infrastructure that is used for such service is also distributed around the globe, allowing requests made by users to be handled by instances of the service hosted closer to them.

In this work we analyse the feasibility of implementing distributed platforms that support FaaS, considering both multi-cloud and federated cloud scenarios, that can effectively balance the load between multiple instances of the service, as well as automatically manage the computational infrastructure capacity available for executing each registered function (auto-scaling), considering both virtual machines and containers.

The rest of the paper is structured as follows: in Section 2, we present work related to this research. Then, in Section 3 we propose an architecture for deploying FaaS-based services in geographically distributed multi-clouds. Thereafter, in Section 4 we provide a reference implementation of the proposed architecture. Finally, in Section 6 we present our insights when developing this work.

2 RELATED WORK

Over the course of years, several researches about cloud computing deployment and service models have been developed. Such researches aim at providing cloud-based services that have, among other characteristics, high availability and scalibility, while maintaining low operation costs.

Considering the deployment and management of hybrid multi-clouds, Brasileiro *et al.* proposed Fogbow – a middleware to federate IaaS clouds (Brasileiro et al., 2016). This middleware eases the interoperability of hybrid multi-clouds and the management of federated cloud resources.

Multiple open source FaaS platforms have been developed, but they are usually only able to perform auto-scaling of function replicas (containers) and not the auto-scaling of the underlying infrastructure (Spillner, 2017).

To the best of our knowledge, this is the first time that the aforementioned technologies or any of their kind have been combined together to enable the deployment of a FaaS platform in a multi-cloud environment with auto-scaling of nodes (virtual machines).

3 ARCHITECTURE

In this paper we propose an architecture for deploying and executing Serverless applications in federated clouds. This architecture comprises three main components: (*i*) **FaaS Cluster**, (*ii*) **FaaS Proxy**, and (*iii*) **Multi-Cloud Resource Allocator** (MCRA). Such architecture is depicted in Figure 1.

The **FaaS Cluster**, as its name suggests, is a cluster where the actual FaaS platform is deployed. Multiple instances of the FaaS Cluster are deployed on geographically distributed cloud infrastructures. This component receives and processes requests coming from the FaaS Proxy. It is also responsible for monitoring the resources used in each individual cloud, and request the MCRA to provide more resources when needed, or release resources when they are no longer necessary.

The **FaaS Proxy** is responsible for providing the same interface that a regular FaaS gateway would. However, instead of processing the requests directly, it must choose between either forwarding the request to a load balancing service or forwarding it to a multicasting service. This choice depends on the type of the request received. Requests related to registering or removing functions available should be sent to a multicaster service. This service will make sure the received requests are replicated in all FaaS Clusters deployed. On the other hand, requests for the execution of a function are forwarded to a load balancing service. This service will chose where to execute the function, based on the geographic location of the user who made the request and that of the available FaaS Clusters.

Last but not least, the **Multi-Cloud Resource Allocator** (MCRA) is a component able to allocate or remove resources in any of the clouds that provide the infrastructure to the FaaS Clusters. These clouds may be operated by different providers, and run different orchestrator middleware. The MCRA implements auto-scaling at the virtual machine level.

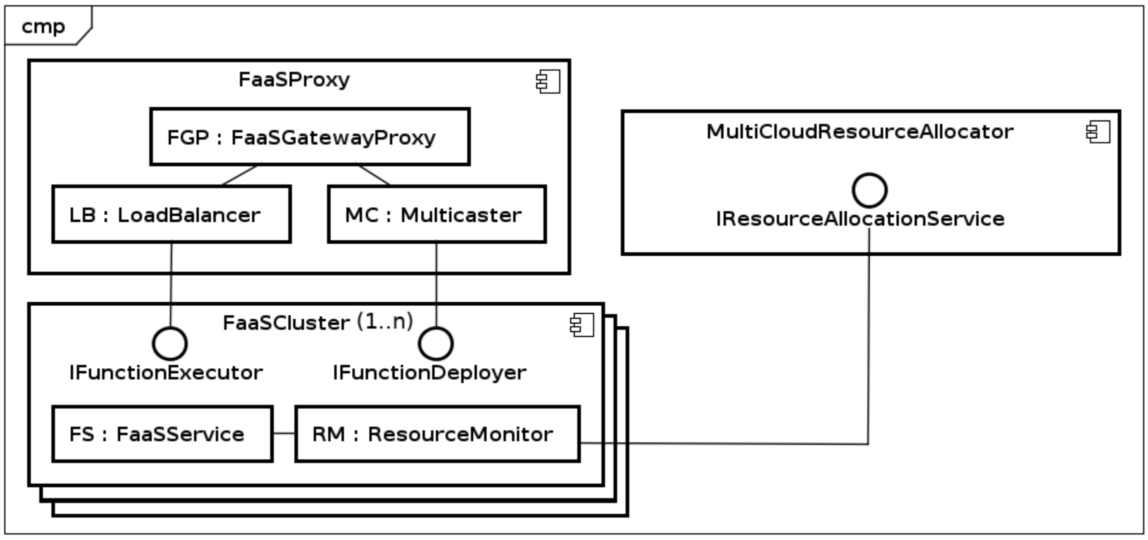


Figure 1: DistributedFaaS architecture.

4 IMPLEMENTATION

As a proof of concept, we have implemented the architecture proposed in this paper based on some of the most prominent solutions in cloud computing at this time: Fogbow[1], OpenFaaS[2], Kubernetes[3], HAProxy[4] and Asperathos[5]. Fogbow is used for the provisioning of virtual machines in a heterogeneous multi-cloud setting, where the FaaS Clusters will be deployed. OpenFaaS is used as the FaaS platform of the individual FaaS Clusters. It is used for deploying and executing functions, as well as performing intra-cluster auto-scaling of resources (containers). Kubernetes is the backend technology used by OpenFaaS to deploy its infrastrucure. HAProxy, on its turn, is used to implement load balancing at the FaaS Proxy component. Finally, Asperathos is used for monitoring the usage of physical resources (memory, CPU, etc.) at each FaaS Cluster, and perform the auto-scaling of virtual machines using Fogbow. The implementation of the FaaS Proxy is complemented with a simple multicaster explained next.

4.1 FaaS Proxy

The FaaS proxy is composed by three subcomponents: (*i*) an OpenFaaS Gateway Proxy, a (*ii*) Load Balancer, and a (*iii*) Multicaster.

[1]http://www.fogbowcloud.org/
[2]https://www.openfaas.com/
[3]https://kubernetes.io/
[4]http://www.haproxy.org/
[5]https://github.com/ufcg-lsd/asperathos

The OpenFaaS Gateway Proxy is an HTTP(S) server written in Golang that exposes the same API as a regular OpenFaaS API Gateway. That is, the server is able to take requests to create, update, and delete functions, as well as requests for deleting functions that are no longer needed. When a request for creating, updating or deleting a function is received, it is forwarded to the FaaS Proxy Multicaster, so that the message can then be sent to all FaaS Clusters belonging to the same cloud federation. If a request to execute a function is received, it is forwarded to the FaaS Proxy Load Balancer, so that it can choose to which cloud provider should that request be forwarded.

The FaaS Proxy Multicaster, also developed in Golang, receives requests from the FaaS Gateway Proxy and replicates such requests across all FaaS Clusters currently deployed. If a request fails to be executed in any of the FaaS Clusters, the Multicaster is responsible for retrying it in background, until it succeeds. The Multicaster also keeps a history of all actions performed in time, so that if a new FaaS Cluster joins the cloud federation, it can replicate the state of the other FaaS Clusters (deploying the currently available functions).

The FaaS Proxy Load Balancer, implemented using HAProxy, receives requests from the FaaS Gateway Proxy to execute functions. It then uses a set of rules configured as Access Control Lists (ACLs) and the information about the client geographic location (based on the IP of the sender) to determine to which FaaS Cluster should it forward the request for executing a given function.

Table 1: Delay added to messages when sending messages from clients depending on the vLAN used.

Network	Source	Delay (in ms)
Distributed-faas-main-subnet	Client 1	40 ± 5
Distributed-faas-main-subnet	Client 2	40 ± 5
Distributed-faas-site-1-subnet	Client 1	40 ± 5
Distributed-faas-site-1-subnet	Client 2	110 ± 10
Distributed-faas-site-2-subnet	Client 1	110 ± 10
Distributed-faas-site-2-subnet	Client 2	40 ± 5

4.2 FaaS Cluster

The FaaS Cluster is based on the OpenFaaS platform. It can be subdivided in two main subcomponents: (*i*) FaaS Service, and (*ii*) Resource Monitor, further described in the following.

The FaaS Service is an instance of the OpenFaaS platform. We have chosen to use Kubernetes as the underlying technology for deploying it, as it eases the task of auto-deploying and auto-scaling the containers that compose a given application or service. The OpenFaaS platform comprises an API Gateway, responsible for receiving requests to create, update execute and remove functions, as well as containers of the functions themselves. Additionally, we have included a deployment of a Kubernetes Metrics API, that enables us to collect metrics about the resources (CPU, memory, etc.) used by the OpenFaaS cluster.

The FaaS Cluster Resource Monitor was implemented in Python using Asperathos, which is a platform to facilitate the deployment and control of applications running in cloud environments. For the purposes of this work, we have developed three *Asperathos plugins* that are responsible for: (*i*) collecting metrics from the Kubernetes Metrics API, (*ii*) analysing the need for new resources (virtual machines) or releasing resources that are no longer needed, and (*iii*) making the actual requests to the MCRA to allocate or release resources.

4.3 Multi-Cloud Resource Allocator

The MCRA is based on the Fogbow middleware. It is responsible for receiving requests from the FaaS Cluster Resource Monitor to allocate and release resources. Upon receiving a request, the MCRA contacts the Fogbow Resource Allocation Service (RAS), and asks it to make the necessary adjustments (adding or removing virtual machines) to the appropriate cloud where the request came from. This component, together with the FaaS Cluster enables our solution to achieve the aimed auto-scaling of resources.

5 EVALUATION

To assess the performance of serverless applications deployed in a DistributedFaaS environment, we performed a series of experiments that compare it to a regular OpenFaaS deployment model. In this section we describe the experiments performed, and present a discussion based on the results obtained.

5.1 Experiments Setup

The experiments were conducted using 5 virtual machines running Ubuntu Linux 16.04, each with two virtual CPUs @2.4GHz and 4 *GB* of RAM. We used each of the virtual machines to host two clients, two OpenFaaS sites and an OFGP as depicted in Figure 2.

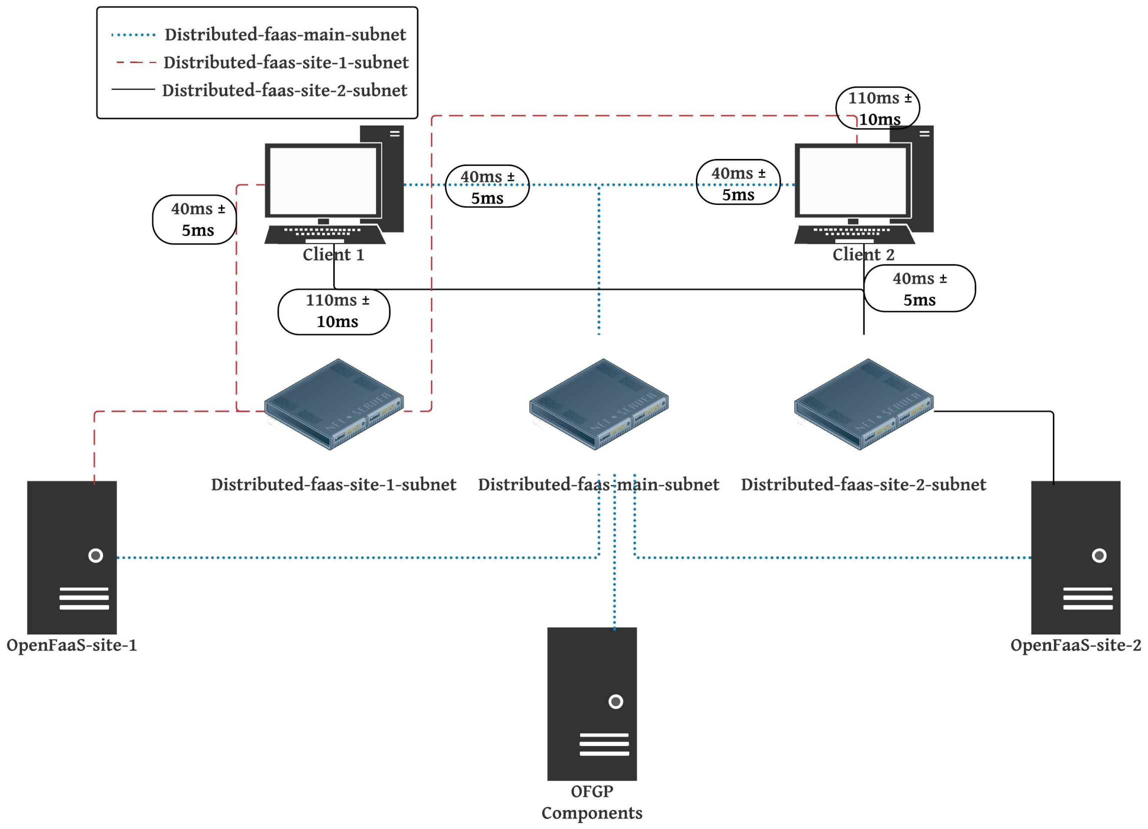
To emulate different scenarios in our architecture, we have also created three vLANs, *i.e.*, Distributed-faas-main-subnet, Distributed-faas-site-1-subnet and Distributed-faas-site-2-subnet. Such vLANs were used to simulate multi-cloud platforms, where the proximity of the clients to the OpenFaaS sites was emulated by adding a delay to messages sent from the clients depending on the vLAN it sends messages to. Table 1 describes the delay added when sending messages depending on the vLAN used. The posible communication paths considering all participants and vLANs is shown in Figure 2.

In our experiments, we measured the latency that represents the total time taken from the client requesting the server to process a function until the client receives the response from the server. As the sample function requested to be executed by our clients, we used the *nodeinfo* function[6], published by OpenFaaS itself, which retrieves CPU/network information on the container where the function is executed.

To benchmark the performance of both deployments we used the widespread Apache Benchmark tool[7], which provides, among other information, a

[6]https://github.com/openfaas/faas/tree/master/sample-functions

[7]https://httpd.apache.org/docs/2.4/programs/ab.html



Distributed-faas-main-subnet
Distributed-faas-site-1-subnet
Distributed-faas-site-2-subnet
110ms ± 10ms
40ms ± 5ms
40ms ± 5ms
40ms ± 5ms
Client 1
Client 2
110ms ± 10ms
40ms ± 5ms
Distributed-faas-site-1-subnet
Distributed-faas-main-subnet
Distributed-faas-site-2-subnet
OpenFaaS-site-1
OpenFaaS-site-2
OFGP Components

Figure 2: Experiments Network Diagram.

summary of the latency obtained from performing requests to a given server.

5.2 The Experiments

The experiments consisted of three scenarios. In our first scenario, which is our baseline evaluation, both clients performed requests directly to the OpenFaaS sites closest to them. That is, **Client 1** sent the requests directly to **OpenFaaS-site-1** while **Client 2** sent the requests directly to **OpenFaaS-site-2**, both using the **Distributed-faas-main-subnet**.

In the second scenario, used to evaluate the overhead of adding the OFGP components, to a regular OpenFaaS deployment, both **Client 1** and **Client 2** sent their requests via the OFGP components also using the **Distributed-faas-main-subnet**.

In the third scenario, we simulated the overhead of having geographically distributed multi-clouds by considering two situations: (*i*) both **Client 1** and **Client 2** sending requests directly to **OpenFaaS-site-1** using the **Distributed-faas-site-1-subnet**, and (*ii*) both **Client 1** and **Client 2** sending requests directly to **OpenFaaS-site-2** using the **Distributed-faas-site-2-subnet**.

In all scenarios of the experiment each client sent one request to execute the *nodeinfo* function in the designated OpenFaaS site and using the described vLAN. In total, 100 runs of each scenario were executed.

5.3 Results Obtained

Table 2 shows the results obtained from our experiments. From the results obtained, we can show that the addition of the OFGP components only generates an overhead of 2.2% when compared with our baseline deployment model, which uses the OpenFaaS reference implementation. That overhead is negligible when compared to the overhead resulting from the use of geographically distributed multi-clouds which is of 27.1%.

With that being said, we demonstrated that the architecture proposed in this paper can be used to enable the deployment of serverless applications in distributed multi-clouds without adding much overhead to it.

Table 2: Delay added to messages when sending messages from clients depending on the vLAN used.

	Latency (in ms)	**Requests/sec**	**Relative overhead**
Scenario 1 (baseline)	290.910	3.44	-
Scenario 2 (OFGP overhead)	297.303	3.36	2.2%
Scenario 3 (Distributed multi-cloud overhead)	369.683	2.81	27.1%

6 CONCLUSIONS

In the first implementation of the proposed architecture we were able to deploy and execute a FaaS platform distributed across federated clouds. This was done by implementing a new component (FaaS Gateway Proxy) that acts as an interface between the users of the distributed FaaS platform and the actual FaaS Clusters that are deployed in the federated clouds.

The availability and scalability of functions was achieved by the use of OpenFaaS and Kubernetes clusters. By using such technologies, replicas of the functions are automatically deployed and removed in virtual machines belonging to a same FaaS Cluster. We were also able to achieve auto-scaling of virtual machines in a distributed multi-cloud environment, by using the Asperathos framework in conjunction with the Fogbow middleware. This allowed our system to allocate and release both containers and virtual machines seamlessly.

The third goal of this work, distributing the load across the multiple FaaS Clusters based on the geographic location of users who make the requests was achieved by writing ACLs to an HAProxy server proxy.

We have also shown that the use of the OFGP components only contribute to a small overhead of 2.2% in latency when compared to a regular OpenFaaS deployment.

As a future work, we intend to better evaluate the performance of FaaS-based applications deployed in such platform, as well as to assess the overall benefits regarding resources utilisation.

ACKNOWLEDGEMENTS

This work was partially funded by CAPES and Dell EMC.

REFERENCES

Brasileiro, F., Silva, G., Araújo, F., Nóbrega, M., Silva, I., and Rocha, G. (2016). Fogbow: A middleware for the federation of iaas clouds. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 531–534. IEEE.

RightScale (2018). Rightscale 2018 state of the cloud report. https://assets.rightscale.com/uploads/pdfs/RightScale-2018-State-of-the-Cloud-Report.pdf. Last access 12/06/2018.

Spillner, J. (2017). Practical tooling for serverless computing. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 185–186. ACM.