International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 04 | April - 2022 Impact Factor: 7.185 ISSN: 2582-3930

FaaS (Function as a Service)

Injamam Ansari, Nihal Sangole, Areeb Ahmad

Dept. of Computer Science Engineering

Anjuman College of Engineering & Technology

Nagpur, Maharashtra, India

Prof. Abdul Razzaque, Dept. of Computer Science Engineering ,Anjuman College of Engineering and Technology ,Nagpur,Maharshtra,India

Abstract - In this paper, we provide an analysis of Function as a Service (FaaS) infrastructures. In the past few years, FaaS has gained significant popularity and became a go-to choice for deploying cloud applications and micro-services. FaaS with its unique 'pay as you go' pricing model and key performance benefits over other cloud services, offers an easy and intuitive programming model to build cloud applications. , a developer focuses on writing the code of the application while infrastructure management is left to the cloud provider who is responsible for the underlying resources, security, isolation, and scaling of the application. FaaS is an important, emerging category of cloud computing, which requires that software applications are designed and deployed using distributed, highly-decoupled service-based architectures, one example of which is the microservices architecture paradigm. FaaS is associated with on-demand functionality and allows developers to build applications without the overhead associated with server management. As such, FaaS is a type of serverless provisioning model wherein a provider dynamically manages and allocates machine resources, with the developers deploying source code into a production environment. This research provides an analysis of scalability, cost, execution times, integration support, and the constraints associated with FaaS services provided by several vendors: AWS Lambda, Google Cloud Functions, and Azure Functions. We discuss the implications of the findings for software developers.

Keywords- Online Compiler, Cloud Computing, Load Balancing Functions-as-a-Service, Infrastructures, Server less, Cloud Computing, Scalability, Constraints, AWS Lambda, Microsoft Azure, Google Cloud Functions.

Introduction

Cloud Computing is computing that involves a large number of computers connected through a communication network such as the internet, similar to utility computing. [1] The International Telecommunication Union (ITU) defines 'cloud service' as 'a service that is delivered and consumed on demand at any time, through any access network, using any connected devices using cloud computing

technologies.' Cloud Service is further classified into Cloud Software as a Service (SaaS), Communications as a Service (CaaS), Cloud Platform as a Service (PaaS), Cloud infrastructure as a service (IaaS) and Network as a service (NaaS). In this paper, we propose Online Compiler as a Software as a Service (SaaS). A compiler transforms source code from a higher level language to a lower, machine level language. This is mainly done in order to create executable files which can then be run in order to execute the program and its instructions.

Function-as-a-Service (FaaS) has emerged as a new paradigm that makes the cloud-based application development model simple and hassle-free. In the FaaS model, an application developer focuses on writing code and producing new features without worrying about infrastructure management, which is left to the cloud provider.

Many cloud-computing vendors, such as Google, AWS, and Azure, among others, offer FaaS services. AWS Lambda, Google Cloud Functions, and Azure Functions are among the most commonly used FaaS services in industry today [10]. Each vendor offers a different set of capabilities with their FaaS infrastructure implementations, from language runtime support and memory usage to the ability to execute functions at regional edge cache locations in response to events generated by content delivery networks [11]. When considering FaaS as part of a systems architecture, it is vital to choose the solution that works best for the system under consideration. For this reason, it is vital that factors surrounding FaaS infrastructures which influence this decision are discussed and investigated.

As developers, we often agonize over the amount of time spent procuring resources, setting up environments, and performing all the other tasks that prevent us from doing what we love most: developing! While cloud-computing technologies have helped to address this problem by making it easy to acquire resources such as servers, computing power, and storage, the problem of setting up these complex application hosts still plagues us. To further compound the issue, maintaining these servers can be quite costly in terms of time and money. Fortunately, technology often rises to meet the needs of its users, and so we have our featured serverless architecture.

In this paper, factors which will be analyzed include execution times, memory configurations, abilities to scale, pricing and cost of FaaS services, the constraints of FaaS infrastructures, and how well integrated vendor FaaS infrastructures are, not only with their own platforms, but how they can be integrated with third party services. This is an important factor as vendor lock-in is a major barrier to the adoption of cloud computing due to the lack of standardization [2].

Literature Review

FaaS is a new and emerging technology. Whilst it first began to be discussed around 2010 [8], it is only in recent years that the paradigm itself has witnessed sustained interest and is now being offered by the largest cloud providers (AWS, Azure, Google Cloud) as a service. As the team conducting this basic review did not have much experience with research, there was some difficulty in finding informative, trusted sources on the subject and a degree of uncertainty when judging the reliability of some grey sources included in this review. However, advice and training was given each week throughout the course of the six week research period on how to correctly conduct research, and in particular, multifocal literature reviews.

Kuhlenkamp and Werner [2] introduce a methodology for a collaborative SLR on FaaS benchmarking and report on preliminary result of 9 studies. They capture more fine-grained experiments within each paper and extract data regarding workload generator, function implementation, platform configuration, and whether external services are used. A completeness score of these categories represents the reproducibility of FaaS experiments and indicates insufficient experimental description. Somu et al. [4] summarize the capabilities of 7 FaaS benchmarking studies along 34 characteristics for parameters, benchmarks, and metrics. Their results indicate a strong focus on the AWS Lambda platform and identify a lack of support for function chaining, especially in combination with different trigger types.
Taibi et al. [5] conduct an MLR on server less cloud computing patterns to catalogue 32 patterns originating from 24 sources. Their MLR has a strong practitioner perspective but is limited to 7 peer-reviewed sources. Our work focuses on performance whereas their pattern catalogue only occasionally mentions performance as part of discussing a pattern.
Yussupov et al. [3] conduct a systematic mapping study on FaaS platforms and tools to identify overall research trends and underlying main challenges and drivers in this field across 62 selected publications. Their work covers a broader range of FaaS research and explicitly excludes FaaS benchmarking studies "without proposing any modifications" [3] through their exclusion criteria. Nevertheless, they identify 5 benchmarking studies and 26 function execution studies on performance optimization. Al-Ameen and Spillner [7] introduced a curated "Serverless Literature Dataset" that initially covered 60 scientific publications and preprints related to FaaS and Serverless computing in general, but in its latest Version 0.4 (2019-10- 23) [6] the dataset has been extended to 188 articles. The authors classify their work as no survey itself, but rather envision its potential as input for future surveys such as ours. We demonstrate this potential in the manual search process for academic literature where the serverless literature dataset covers 34 out of 35 relevant studies. These two general studies identify publication trends, common technologies, and categories of research but do not extract and synthesize more specific data on FaaS benchmarking aspects we cover in our work.

Disadvantages of Existing System:

There is a potential downside to using Function as a Service. Some of the things you need to consider before you go all-in with FaaS include:
Vendor lock-in: Building your application on a FaaS platform may make you reliant on that vendor and make it difficult to switch.
Testing hurdles: Depending on the vendor, you may have challenges when creating a test environment for your application.
Cold starts: There's sometimes a delay in the execution of a function, as much as 3 seconds, which can adversely impact some types of applications.
Security: You are at the vendor's mercy when it comes to security and may not have the visibility you need to ensure the vendor complies with regulations governing your use or storage of certain types of data.
Cost: This item is repeated from the benefits list, but in some cases, FaaS can actually cost more than using dedicated servers, depending on the processes you are running.

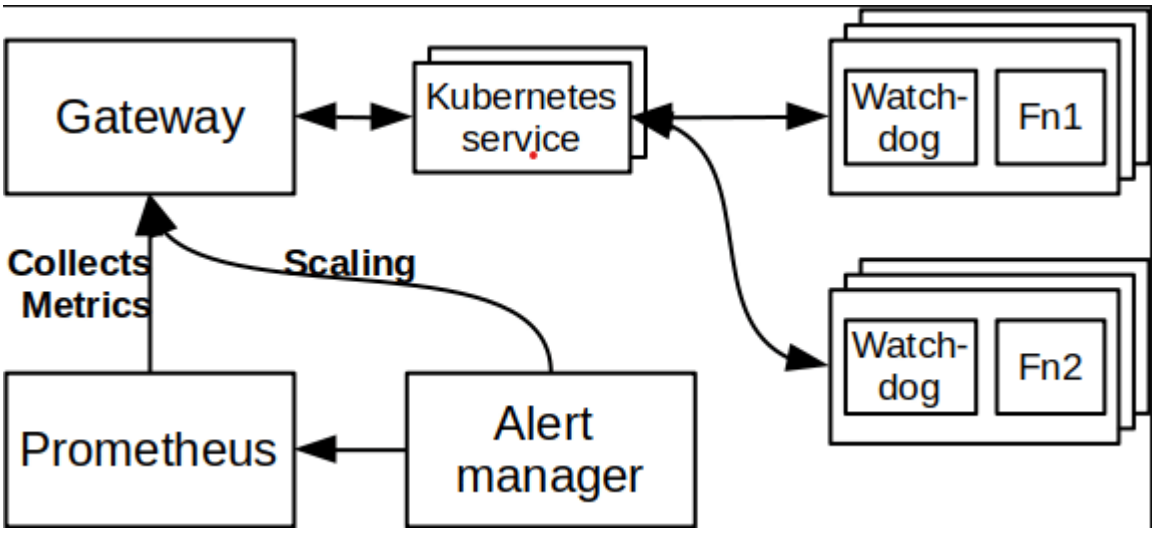
The limitation on application state isn't the only constraint on serverless computing. There are a few more, and they may prevent your application from running as one or more functions.
Or, they might mean you need to rethink your design.
The platform loads functions on demand. They should start up quickly, usually in milliseconds.
Then the platform immediately gives them a request. When processing completes, it terminates them.
The platform may reuse an instance with a "warm start" to save time, but the function cannot rely on this.
This is where the constraint on state comes from. But it also means that an application that performs a lot of initialization will not work well with FaaS.
AWS limits Lambdas to 15 minutes of execution time. Azure limits its Functions to 10 minutes.
This is plenty of time for an API call, but it might not be for a scheduled job. Unfortunately, functions have hard limits on execution time.
FaaS might also be a bad fit if you're concerned about vendor lock-in and can't figure out how to code around it. If you're going to have someone else run your code on their platform, you need to write to their API.
Depending on how you structure your code, you may be able to avoid lock-in. But if you do, serverless might not be the right solution. Or you might not care.

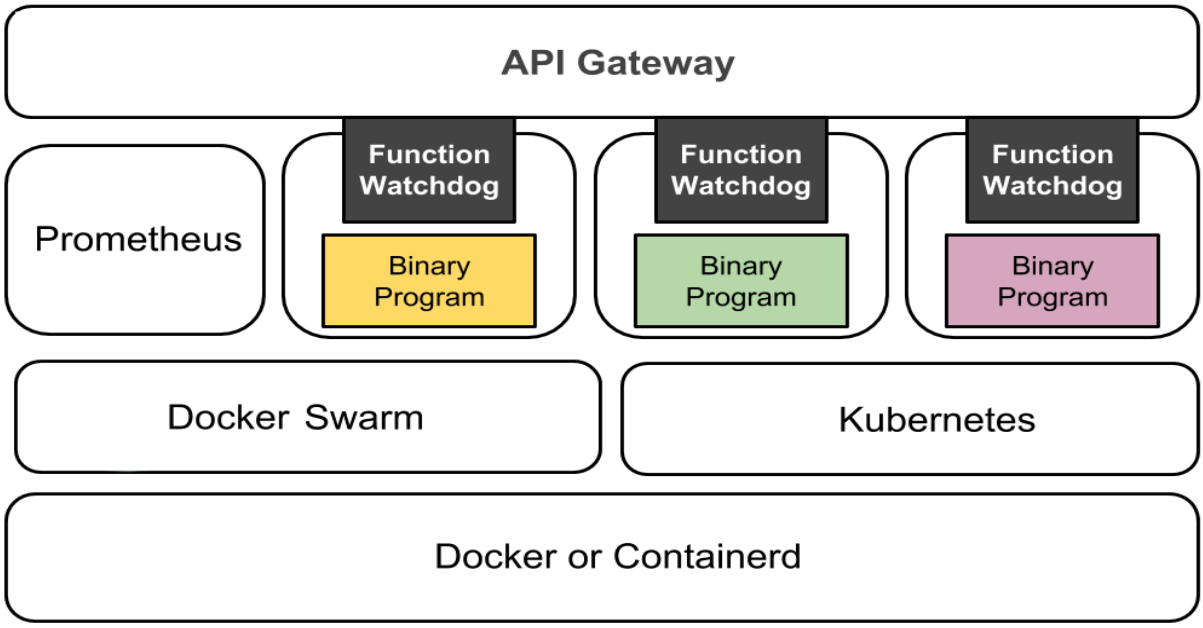
Proposed Approach:

Cloud computing model is for enabling convenient as well as a network access to a shared pool of configurable computing resources. In this internet world all the things are online. Here we use an online compiler. This project's main aim is, we can easily write program, compile and debug it in online. In this project, we have three online compilers namely, Online C/C++, JAVA and Perl compiler. Different programming languages are being compiled using cloud computing, which is portable and reduces the storage space, online java, c/c++, Perl compiler using cloud computing, which provides most convenient tool to compile code and remove the errors. These three compilers provide online compiler service, so no need to install separate compiler on each PC. By using all these application we can conduct online practical examination.

Architecture:

FaaS and serverless are generally synonymous with one another due to its ability to listen and act upon the events of other serverless services .However FaaS is a subset of serverless [10], which resides within the compute category. By understanding the variety of categories and the services that lie within we can better ascertain the best provider for the users architectural needs. The core categories that relate to architecture that are consistent across all three providers are the following, compute, storage, data stores and integration. The main purpose of this research is to see how FaaS integrates with other serverless services. For the sake of brevity, other compute services are omitted as the core focus of this research is FaaS.
Storage provides consistent object-level data storage across all three providers [14, 13, 15]. However AWS also provides EFS; an NFS file system service that can be easily integrated with on-premises or cloud resources [14].





System Requirement

Hardware Requirements-

32-bit Intel® Pentium® 4 or compatible processor running at 2 GHz or greater

512 MB RAM

Disk space: 30 GB at least

Software Requirements-

Any Linux based Operating system

Arkede

Docker,

Kubernetes

OpenFaaS

FaaS-cli

Java, C#, Python, Node.js, and Go, installed in system.

FaaS Templates Requirements-

1) Csharp 2) dockerfile 3) go 4) java11-vertx, java11

5) Node, node12-debain, node12, node14, node16 6) php

7) Python, python3-debain, python3 8) ruby

Setup Procedure

Step1- Install openfaas in system

The arkade install command installs OpenFaaS using its official helm chart. Arkade can also install other important software for OpenFaaS users such as cert-manager and nginx-ingress. It's the easiest and quickest way to get up and running.

You can use arkade to install OpenFaaS to a regular cloud cluster, your laptop, a VM, a Raspberry Pi, or a 64-bit ARM machine.

Get arkade

For MacOS / Linux:

curl -SLsf https://get.arkade.dev/ | sudo sh

For Windows (using Git Bash)

curl -SLsf https://get.arkade.dev/ | sh

Step 2- Create Dictionary function

We will use arkade to install and deploy apps and services to Kubernetes.

Download only, install yourself with sudo

$ curl -sLS https://dl.get-arkade.dev | sh

Download and install

$ curl -sLS https://dl.get-arkade.dev | sudo sh

Step3- Setup Minifest

Step4- FaaS cli secret congratulation

The OpenFaaS CLI allows you to create, update, list and delete secrets using faas-cli instead of Docker or Kubernetes command line tools.

The reason behind this is to give you simplicity when you need to use secrets for your functions as well as to provide a layer of abstraction, as it will work for both Kubernetes and faas.

Create

To create a secret from stdin, you can run:

faas-cli secret create secret-name

To create it from file use:

faas-cli secret create secret-name \

--from-file=~/Downloads/derek.pem

Target a specific namespace:

faas-cli secret create \

--namespace staging-fn \

```
secret-name \
 --from-literal="04385e5c413c10ed68afb010ebe8c5dd706aa20a"
```

Step5- Deploy Dictionary function

Step6- Testing Dictionary function

Reference:

[1] Mariana Carroll, Paula Kotz':, Alta van der Merwe (2012). "Securing Virtual and Cloud Environments". In I. Ivanovet aI. Cloud Computing and Services Science, Service Science: Research and Innovations in the Service Economy. Springer Science+Business Media.

[2]J. Kuhlenkamp, S. Werner, Benchmarking FaaS platforms: Call for community participation, in: 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 2018, pp. 189–194. doi:10.1109/UCCCompanion.2018.0005

[3]V. Yussupov, U. Breitenbücher, F. Leymann, M. Wurster, A systematic mapping study on engineering function-as-a-service platforms and tools, in: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, 2019, pp. 229–240. doi:10.1145/3344341.3368803

[4]N. Somme, N. Daw, U. Bellur, P. Kulkarni, Panopticon: A comprehensive benchmarking tool for serverless applications, in: 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), 2020, pp. 144–151. doi:10.1109/ COMSNETS48256.2020.9027346.

[5] D. Taibi, N. El Ioini, C. Pahl, J. R. S. Niederkofler, Serverless cloud computing (function-as-a-service) patterns: A multivocal literature review, Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER'20) (2020). URL: https://www.researchgate.net/publication/ 340121613_Patterns_for_Serverless_Functions_Functionas-a-Service_A_Multivocal_Literature_Review, in press.

[6] J. Spillner, M. Al-Ameen, Serverless literature dataset, 2019. doi:10.5281/zenodo.2649001.

[7] M. Al-Ameen, J. Spillner, Systematic and open exploration of faas and serverless computing research, in: Proceedings of the European Symposium on Serverless Computing and Applications (ESSCA), 2018, pp. 30–35. URL: http://ceur-ws.org/ Vol-2330/short2.pdf.

[8] G. C. Fox, et al., "Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research," arXiv:1708.08028 [cs], 2017, doi: 10.13140/RG.2.2.15007.87206.

[9] J. Opara-Martins, et al., "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," Journal of Cloud Computing, vol. 5, no. 1, p. 4, Apr. 2016, doi: 10.1186/s13677-016-0054-z

[10] Nemanja Novkovic, "Top Function As A Service (Faas) Providers," Dashbird, 14-May-2018. [Online]. Available: https://dashbird.io/blog/top-function-as-a-service-faas-providers/. [Accessed: 11-Mar-2020].

[11] "Edge Computing | CDN, Global Serverless Code, Distribution | AWS Lambda@Edge," Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/lambda/edge/. [Accessed: 11-Mar-2020].

[12] M. Sewak and S. Singh, "Winning in the Era of Serverless Computing and Function as a Service," in 2018 3rd International Conference for Convergence in Technology (I2CT), 2018, pp. 1–5, doi: 10.1109/I2CT.2018.8529465.

[13] "Serverless Architecture | Google Cloud" [Online]. Available: https://cloud.google.com/serverless/whitepaper/. [Accessed: 26-Feb-2020]

[14] "Serverless Computing - Amazon Web Services" [Online]. Available: https://aws.amazon.com/serverless/ [Accessed: 28-Feb-2020]

[15] "Azure Serverless | Microsoft Azure" [Online]. Available: https://azure.microsoft.com/enus/solutions/serverless/#solutions [Accessed: 28-Feb-2020]