ISSN (e): 2250 – 3005 || Volume, 11 || Issue, 1|| January – 2021 ||
International Journal of Computational Engineering Research (IJCER)

Serverless implementation of Data Wizard application using Azure Kubernetes Service

[1]Kishor Kumar Sethy
Technology architect, Infosys Ltd, Bhubaneswar, India

ABSTRACT
Cloud vendors offer a variety of server less technologies promising high availability and dynamic scaling while reducing operational and maintenance costs. One such technology, server less computing using Azure Kubernetes Service (AKS) and Dockers is advertised as a good candidate for Web API, data-processing, or backend services, where you only pay for usage. Unlike virtual machines (VMs), they come with automatic resource provisioning and allocation, providing elastic and automatic scaling. We present the results from our investigation of a specific server less candidate, Data Wizard application, deployed on AKS and as function(s)- as-a-service. We contrast these deployments by migrating data from Azure Gen1 to Gen2 for measuring response times and costs. We found no significant response time differences between deployments when VMs are configured for the expected load, and test scenarios are within the FaaS hardware limitations. Higher volumes of data migration are effortlessly handled by FaaS, whereas additional labor must be invested in VMs for equivalent results. We identified that the advantages of server less computing bringsclear choice between server less and virtual machines for a Data Wizard application.
KEYWORDS: *server less; Function-as-a-Service; response time; Web API*

Date of Submission: 16-02-2021 Date of acceptance: 02-03-2021

I. INTRODUCTION

Server less has developed into a very converting cloud solution nearly about any type of application, predicting to remove the overhead related with infrastructure maintenance. All major cloud suppliers are invested in this type of program [1]. The advertised benefits are the deficiency of server maintenance, auto scaling, pay-per-usage, and high availability [2–4]. Although server less covers resources like computing, database, storage, stream processing, message queuing, and more, our focus is on computing resources, specifically on function-as-eservice. This type of resource, also referred to as FaaS, is essentially a stateless computing container that is event-triggered and lasts for a single invocation. Unlike virtual machines, the provider is responsible for resource provisioning and allocation. FaaS is billed per invocation and per gigabyte of-memory used per second, measured during actual invocations [5,6]. FaaS can be used for data processing or as application backends, including web applications, or more generically, Web Application Programming Interfaces or Web APIs. Traditionally, the latter are deployed on virtual machines, with the additional cost of provisioning and configuring the underlying infrastructure. Features like load balancing and dynamic scaling fall under the customer's responsibility. This paper shows the results for the investigation on finding the best ways of deploying a Web API into the cloud, focusing on how end-users perceive performance, more precisely on response times. We look at three types of deployments in the cloud: monolithic on virtual machines, micro services on virtual machines, and function-as-a-service. The goal is to compare response times for each type of deployment, while also factoring-in costs, in order to help with better decision-making when it comes to designing and deploying a Web API into the cloud.

1.1. Research Questions FaaS is advertised as a good fit for Web API applications and especially micro services since they have similar granularity. We want to find out how the same application deployed on virtual machines compares to server less, by looking at monolithic, micro services, and function-as-a-service deployments. We start from the assumption that the existing resource constraints of FaaS are already considered, and the application is within these limits. We try to answer to the following research questions in order to support and structure the investigation:

1. How do different types of deployments in the cloud for Web API compare to one another in terms of response times with respect to load?
2. Are cost differences significant in order to offset the balance in favor of one type of deployment? Multiple cloud providers offer server less platforms, with many similarities between them in terms of hardware specification, configurability, and costs. We compare response times for our function-as-a-service application across different providers, using similar deployments in terms of hardware specification, in order to answer the following research question:
3. Is there any difference between FaaS providers when it comes to response times and costs?

1.2. Structure This paper contains an analysis of other work that intersect or help with the proposed investigation, followed by a summary of the server less offers from major cloud vendors. Section 4 describes the implementation setup and testing strategy in more detail. In the subsequent chapter, we discuss results while trying to answer Research Questions 1, 2, and 3. Finally, we summarize the results and draw the conclusions

II. RELATED WORK

2.1. Improving Web Application Deployment in the Cloud

Roberts and Chapin's work [7,8] is a good starting point to get an in-depth view of the Server less domain, illustrating areas where the Server less domain needs improvement, for example vendor lock-in, state management, the lack of any Server less architecture patterns, and more [9]. Lynn et al. [10] compare FaaS offerings using other criteria besides just performance and cost. They argue that the advertised advantages of Server less are based on few use-cases and research papers. However, Adzic and Chatley [11] studied two production applications that have successfully transitioned to Server less and found that the most compelling reasons for transitioning are hosting costs. They also concluded that high-throughput applications are a better choice than high availability ones when it comes to costs. Eivy [12] compared costs for running an application on a virtual machine versus running it on Server less and found that it is cheaper to deploy it on virtual machines if there is a constant and predictable load. He advises that rigorous testing and simulations should be performed anyway, before deciding to move to Server less. Trihinas et al. [13] make the case for micro services adoption, illustrating their drawbacks and presenting a solution [14] that promises to solve the existing challenges. Georgiou et al. [15] investigated a specific use-case, Internet-of-Things applications and edge processing, while showcasing their framework for streaming sensor analytics with the help of queries. Both references [13,15] point out the challenges of modern applications and successfully address these in one way or another, with the use of micro services and containers. This goes to show that although server less may address the same issues, there are solid alternatives, such as micro services and containers, that should be considered or even better, be used together with server less.

2.2. Serverless Computing Performance

Back and Andrikopoulos [16] as well as Lee et al. [17] did performance testing to compare FaaS offerings of different providers. They did this by gradually incrementing load on a single function while observing resource usage and comparing costs. The benchmark they defined can be used when individual raw power of a function is concerned. Lee et al. on the other hand, tracked more than just resource consumption and concluded that distributed data applications are a good candidate for FaaS, whereas applications that require high-end computing power are not a good fit. They found that the main reasons for this are the known execution time limit and fixed hardware resources. Lloyd et al. [18] deployed micro services as FaaS and looked at a variety of factors that affect their performance. They did this by changing the load in order to observe how the underlying infrastructure performs. Some papers [10–12] look at possible use-case for Server less by considering their advertised benefits and costs. Others [16–18] investigated FaaS performance in-depth and across multiple cloud providers. We see an opportunity here to investigate a specific and common use-case for FaaS, namely Web APIs. We plan to focus on the end-user's perspective by measuring and comparing a single metric, i.e., response times.

III. SERVERLESS COMPUTING PLATFORMS

We compared two commercial server less providers, Amazon Web Services or AWS Lambda and Microsoft Azure Functions, to help answer the proposed research questions. Google Cloud Functions are also considered for cost comparison.

3.1. Amazon Web Services Lambda

Amazon's FaaS offering has been around since 2014 and is one of the most popular implementations of FaaS. In 2019 it supports Node.js, Python, Java, GO, C#, and PowerShell. Amazon counts and bills each request that invokes your Lambda function, with $0.20 for each one million requests, across all your functions. Duration

is also billed depending on the amount of memory you allocate to your function, with $0.00001667 for every GB-second used [5]. Duration is calculated from the time your code begins executing until it returns or terminates, rounded up to the nearest 100 ms. Amazon also provides one million free requests per month and 400,000 GB-seconds of computing time per month. Each individual function can be configured to use any amount of memory between 128 MB and 3 GB in 64 MB increments and you will be billed based on your configured amount. Lambda allocates CPU power linearly in proportion to the amount of memory configured. At 1792 MB, a function has the equivalent of one full virtual CPU. Amazon limits the execution time of a function to 900 s. Total concurrent executions across all functions within a given region are also limited by default to 1000 [19].

3.2. Azure Functions

The FaaS alternative from Microsoft works with Node.js, Python, Java, C#, and F#. Like AWS Lambda, Azure Functions are billed based on executions and per-second resource consumption. Price for one million executions is $0.20 and price for consumption is $0.000016 per GB-second, identical to Amazon's pricing scheme. Free grants are also the same: one million executions and 400,000 GBseconds of computing time per month [6]. Individual functions are hosted in a Function app, so multiple functions can be implemented in a single host. Function apps are allocated 1536 MB and one CPU, but you are not billed on this amount, you are billed on actual memory consumption, rounded to the nearest 128 MB, with execution time calculated by rounding up to the nearest millisecond. The consumption plan is the default you can run your functions under. This plan has an execution time limit of 10 min and a scale limit of 200 instances [20]. A single instance may process more than one message or request at a time, so there is no limit on the number of concurrent executions. The App Service Plan is another plan you can choose, where your functions run on your dedicated virtual machines (VMs) and the executiontime is unlimited. There is also a Premium plan, still in public preview mode, that promises configurable instance sizes, unlimited execution duration, and always warm instances [21].

3.3. Cloud Functions

Google's implementation of FaaS only supports Node.js, Python, and GO. Pricing is $0.40 per million invocations and $0.009 per GB-hour rounded to the nearest 100 ms. Google also bills CPU usage per second at $0.036 per GHz-hour and in-out data traffic at $0.12 per GB. It also includes a free tier of two million executions, 400,000 GB-seconds, 200,000 GHz-seconds of computing time, and 5GB of internet traffic per month [22]. Although the previous providers do not mention data-traffic costs when describing FaaS, these costs exist and traffic coming in and out of their respective cloud networks is billed accordingly. Functions can be configured to use between 128 MB to 2 GB of memory and then CPU is allocated proportionally. A 2 GB memory will be allocated a 2.4 GHz CPU [23]. Cloud functions are limited to 540 s execution time and 1,000 concurrent invocations.

3.4. Use-Cases

FaaS-suggested usages are similar across providers and can be generalized into data-processing and back ends. The advertised solutions for AWS Lambda are data real-time file processing, real-time stream processing, data transformations, and backends for Internet-of-Things, mobile, or web applications [24]. Based on Microsoft's suggestions, Azure Functions can be used to implement web applications, APIs, micro services, machine learning workflows, data processing pipelines, and real-time processing applications [25]. Google Cloud Functions are suitable for implementing application back ends, real-time data processing systems, and intelligent applications [26]. This chapter was a walkthrough of the FaaS offers from Amazon, Microsoft, and Google. We learned what the limitations, costs, and use-cases are for function-as-a-service implementations. Web applications are advertised as good candidates for this serverless technology, so the proposed investigation targets Web APIs built within the hardware limitations of FaaS.

IV. EXPERIMENTAL SETUP

"Data Wizard" (DW) was implemented using Azure virtual machine (VM). All the required software installed and upgraded in VM. The application runs on VM's infrastructure. When setting up the application in other client environment the following activities need to be performedInfrastructure setup, Installation/upgrade of all required software, Code Migration, Configuration DW components and Integration of DW components

All the above activities involve time & cost. To reduce or optimize the process, Serverless implementation of DW is required. Dockers and Azure Kubernetes service used for serverless implementation of DW.

Data Wizard application built using open source and azure components.

Open Source

Component	Usage
Python 3.6.9	Backend scripting
Angular9/HTML/CSS3	Front-end User Interface
Node JS	UI deployment
Spark	2.4.4

Azure Resources

Component	Usage
Azure SQL DB	Metadata database
Azure Data Factory	Pipeline Creation Scheduling
Azure Kubernetes Service(AKS)	Azure Kubernetes is a platform that manages container-based applications and their associated networking and storage components.
Azure Databricks	For executing python scripts
Azure Container Registry (ACR)	allows you to build, store, and manage container images and artifacts in a private registry for all types of container deployments. ACR integrate with Kubernates to run the containers

Figure-1

Implementation Architecture:

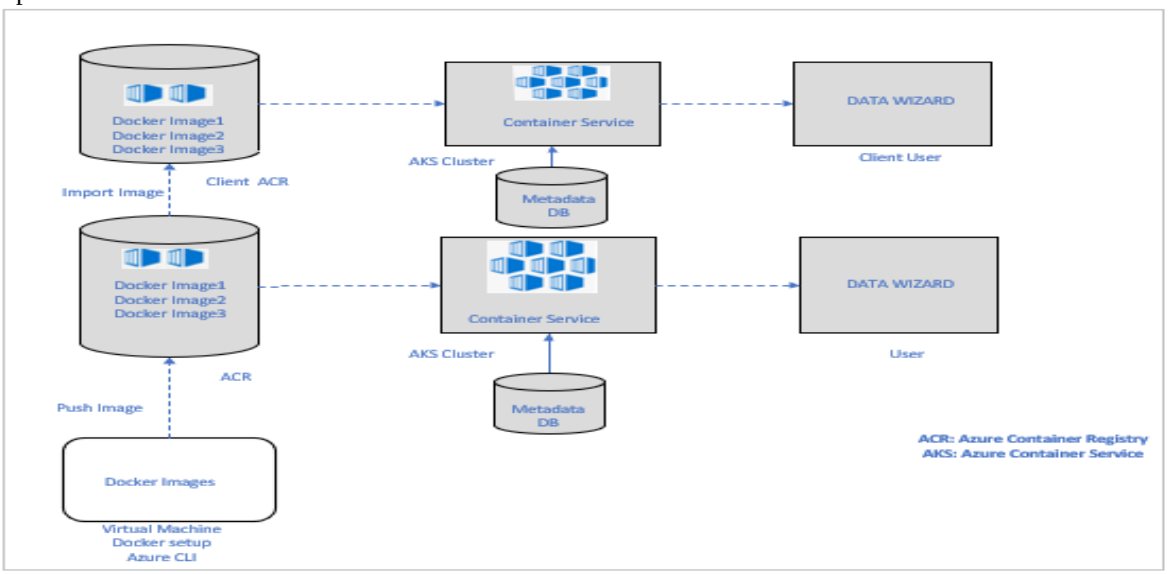

Figure-2

Implementation Details:

1. Azure Kubernetes Service Setup with Service Principle.
2. Azure Container Registry Setup
3. Azure CLI Installation
4. Authorization & Access
5. Docker Installation
6. Docker Image creation & migration to ACR
7. Configuration of container service from AKS

Docker implementation flow:

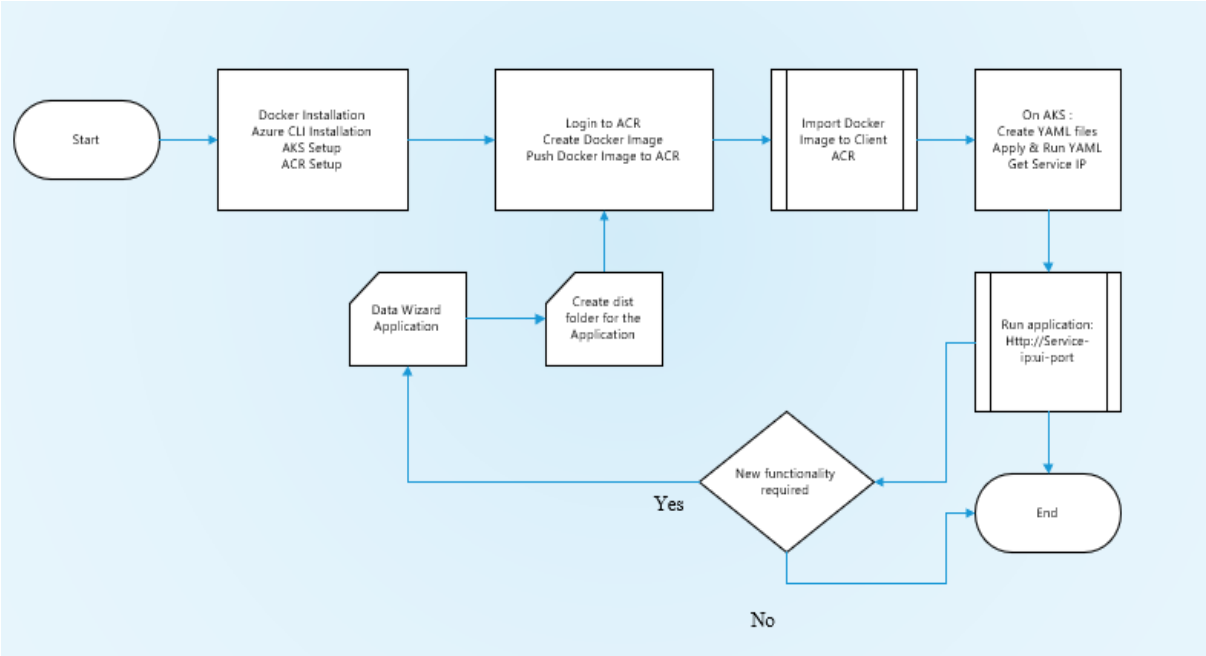

Figure-3

Azure Kubernetes Services(AKS):
Kubernetes is a rapidly evolving platform that manages container-based applications and their associated networking and storage components. The focus is on the application workloads, not the underlying infrastructure components. Kubernetes provides a declarative approach to deployments, backed by a robust set of APIs for management operations.
Azure Kubernetes Service (AKS) is a managed Kubernetes offering that further simplifies container-based application deployment and management.

Azure Container Registry (ACR):
Azure Container Registry is a managed Docker registry service based on the open-source Docker Registry 2.0. and it allows to store and manage images for all types of container deployments.

Dockers:
Docker provides the ability to package and run an application in a loosely isolated environment called a container. Docker enables us to separate our applications from infrastructure so we can deliver software quickly. The isolation and security allow to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so it does not need to rely on what is currently installed on the host.

Docker Images:
An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. To create a docker Image a docker file need be created, where all the instructions are provided for installation & configuration of the application.

V. EXPERIMENTAL RESULTS AND DISCUSSION

After the docker image hosted in AKS cluster, the application used to migrate data from Azure Gen1 to Gen2 and Gen2 to Gen2.

Figure-4 shows Data migration statistics: VIM vs AKS Cluster

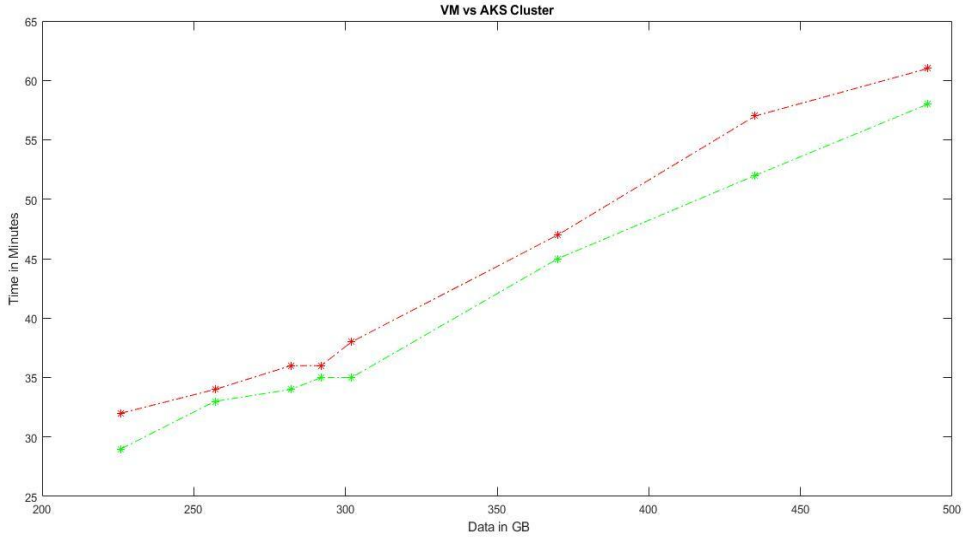

Figure -4

Implementation Time: With Docker & Without Docker

	No of PDs	
Activity	**Without Docker**	**With Docker**
Requirements Analysis	5	5
Detailed Design	3	3
Build & Unit Testing	10	10
SIT Deployment	3	0.5
UAT Deployment	3	0.5
Production Deployment	3	0.5
Documentation	5	3
Total PDs	**32**	**22.5**

Fgure-5 PD: Effort in Person days

The goal of the document is to help with better decision making when deploying Data Wizard application into cloud with serverless implementation. Additionally, we looked at costs to find out if there are significant differences in order to shift the decision in one direction or the other. We started from the assumption that the application's hardware requirements are within the limit of FaaS. Infrastructure cost is extra in case of using VM or Server. Implementation cost is more in case of without docker.

The following functional testing are yet to be done.

1. Running application using number of concurrent users.
2. Performance testing of the application and tuning the AKS components
3. Cost benefit analysis
4. Implementation of the same using other cloud.
5. Accessing the application using public/private DNS.

VI. CONCLUSIONS

The goal of this paper is to help with better decision-making when deploying a Web API into the cloud, with a focus on the end-user's perception of performance. Specifically, we compared response times for the same application, deployed as function-as-a-service, and as monolith and microservices on virtual machines.

Additionally, we looked at costs to find out if there are significant differences in order to shift the decision in one direction or the other. We started from the assumption that the application's hardware requirements are within the limits of FaaS and established some test phases where we vary the number of concurrent users. We found that there are no considerable differences in response times between deployments, when VMs are properly configured with respect to load. This also means that additional effort and money need to be invested in VM deployments in order to match the native capabilities of FaaS. Usage predictability is an important factor when deciding between deployments, as it can help reduce costs significantly. The presented results show that VM deployments are several times cheaper when properly configured for the expected load. However, our investigation does not account for the maintenance and configuration costs associated with VMs. More to this, FaaS successfully handles unexpected user growth with its built-in features, whereas additional effort would be required to setup VMs to match FaaS' auto-scaling capabilities. On the other hand, when payload increases, scaling out does not help, and VMs can provide more unused raw power as opposed to FaaS's fixed memory and CPU configuration. When it comes to cloud providers, response times seem to be better for AKS. However, in a real situation, multiple other infrastructure components are used, which affect response times and costs. To sum up, one needs to do a thorough analysis before choosing a cloud provider or a type of deployment, as there are many variables to consider affecting performance and costs.

REFERENCES

[1]. P.Arul Paul Sudhahar,V.K.KalaiVani, The Total Edge Monophonic Number of a Graph, International Journal of Applied Mathematics.ISSN 1819-4966 Volume 11,Number 2 (2016) pp.159-166.
[2]. F.Buckley and F. Harary, *Distance in Graphs*, Addison-Wesley, Redwood City, CA, (1990).
[3]. F. Buckley, F. Harary, and L. V. *Quintas*, Extremal Results on the Geodetic Number of a Graph, Scientia A2 (1988) 17-26.
[4]. G. Chartrand, F. Harary, and P. Zhang, On the Geodetic Number of a Graph, *Networks*. 39 (2002) 1-6.
[5]. F. Harary, *Graph Theory*, Addison- Wesley, 1969.
[6]. J. John, P. Arul Paul Sudhahar, On the edge monophonic number of a graph, Filomat 26:6 (2012), 1081 - 1089.
[7]. J.Jhon, P.Arul Paul Sudhahar,The Upper Edge Monophonic Number of a graph, IJMCAR, ISSN 2249-6955, vol 3,(2013) 291-296.
[8]. J. John, S. Panchali, The Upper Monophonic Number of a Graph, International J. Math, Combin. Vol. 4 (2010), 46-52.
[9]. A.P. Santhakumaran, P. Titus, K. Ganesamoorthy, On the Monophonic Number of a Graph, *J. Appl. Math. & Informatics* Vol. 32 (2014), No. 1 – 2, pp. 255 – 266.

Kishor Kumar Sethy, et. al. "Serverless implementation of Data Wizard application using Azure Kubernetes Service." *International Journal of Computational Engineering Research (IJCER*), vol. 11, no.1, 2021, pp 09-15.