Benchmarking Serverless Computing: Performance and Usability

Mubashra Sadaqat, Østfold University College, Halden, Norway

Mary Sánchez-Gordón, Østfold University College, Halden, Norway

Ricardo Colomo-Palacios, Østfold University College, Halden, Norway

1 Introduction

Serverless computing is part of the cloud computing paradigm, which is based on the microservice architecture. It focuses on providing resources in a fine granular and flexible manner in order to further utilize the benefits of cloud computing (Völker, 2018). Serverless computing aims to save companies significant amounts of time, money and resources by hosting, running, and managing the applications and core functions of a business. Using this model, a company is freed from the efforts and processes of running and maintaining server applications (Hardin, 2018). As an instance of such processes, traditional server architectures involve a server process, typically listening to a TCP socket, waiting for clients to connect and send requests (Sadaqat, Colomo-Palacios, & Emil Skrimstad Knudsen, 2018).

The serverless environments are delegating the job of running service functions to cloud providers, thereby allowing them to decide how to manage execution. The cloud vendors such as Amazon Web Services (AWS), Microsoft Azure, Google, IBM, and Iron.io are all incarnations of this idea in various stages of development and adoption where application logic is split into functions and executed in response to an event (McGrath & Brenner, 2017). With services such as AWS Lambda, Microsoft Azure Functions, Google Cloud Functions, and IBM OpenWhisk, users have the

ability to write desired applications. These applications are a collections of stateless functions that are event-triggered, short-lived, and fully managed by the cloud provider and function deploy directly to a serverless framework instead of running tasks on traditional virtual machines with pre-allocated resources (Klimovic et al., 2018). Moreover, serverless computing as normal cloud computing effectively eliminates the time consuming and expensive traditional approach of purchasing hardware/software, installing servers, configuring, and troubleshooting. This idea makes serverless services more attractive for IT and business industry (Sadaqat et al., 2018).

There are several reasons for the rapid emergence of serverless computing mentioned in the literature (Erwin van Eyk, Iosup, Abad, Grohmann, & Eismann, 2018). For example, the cloud users are relieved of the need to manage resources, designing auto scaling procedures, and other operational logic. This allows them to focus on their business logic, reducing the costs of development, removing the need for huge distributed systems expertise and improving the time-to-market for applications.

Despite the above-mentioned facts, it is quite tricky and challenging for companies to select the most suitable cloud vendor, related to their requirement of application and demand of customers (Vázquez, Ramki Krishnan, & John, 2014). One way to develop and deploy services in a profitable manner and resources, companies look at the variety of options provided by cloud vendors in the form of pricing, performance, load-balancing, programming language and other sets of features. Therefore, it becomes increasingly important for potential cloud users to have deep and clear knowledge to work in an efficient way (Qu, Wang, & Orgun, 2013).

In this scenario, benchmarking is widely used for the evaluation of computer systems. The benchmarks exist for a variety of levels of abstraction, from the CPU and the

database software, to complete enterprise systems (Cooper, Silberstein, Tam, Ramakrishnan, & Sears, 2010). However, there is a cloud user need for serverless benchmarks in a technical and suitability perspective. In this study, we focus on Amazon and Microsoft Azure platforms because these two platforms are very close competent players of cloud according to (Scale, 2018). The benchmark of serverless would help the developer and cloud users to differentiate the services provided by the cloud vendors, Amazon and Microsoft Azure. We believe that having a serverless service benchmark would also provide a unique opportunity for software engineering researchers and IT industry. The comparison would be helpful for not only the IT industry but also for the new users of serverless computing to understand the underlying features and important aspects of the serverless platform including developer experience.

2 Background

2.1 Platform of Serverless Computing

There are some well-known serverless cloud vendors such as Amazon, Microsoft, Google, IBM and, iron.io. Apart from these cloud providers, other open source providers of serverless platforms also exist. However, Amazon cloud vendor is on top of the other providers with its solution named “Lambda”. It is the most fully featured and provides the first widely used commercial serverless solutions (Eivy, 2017). It is worth noting that other cloud providers also offer their own variation of serverless services that share the main characteristic of Lambda. On the other hand, Microsoft Azure platform is the second largest cloud platform (Scale, 2018). IBM cloud functions and Google cloud functions are also commercial offerings of serverless services (Baldini et al., 2017). The platform launched by IBM is the open source implementation of serverless service called OpenWhisk, which was released by IBM

as an apache project and OpenLambda. The cloud service provided by Google deserves to be mentioned because it is a unique platform on its own way (E. van Eyk et al., 2018; Baldini et al., 2017). In the end, Iron Function is an open source serverless computing framework made by iron.io (Kanso & Youssef, 2017). Figure 1 shows the evolution of serverless computing platform.

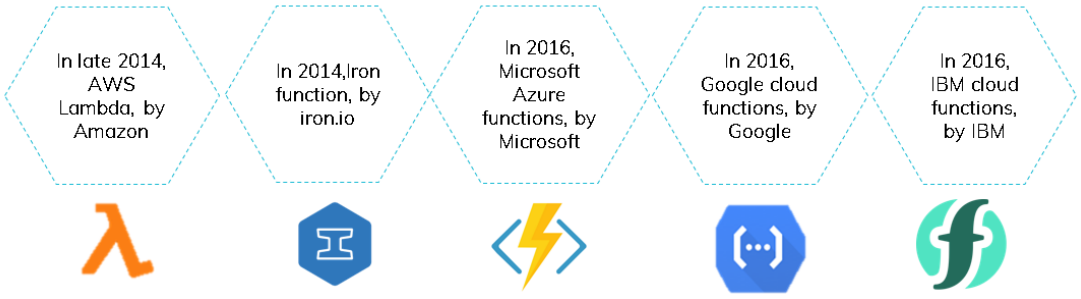

Figure 1. Evolution of serverless computing platform

2.2 Related Work

Lee et al. (Lee, Satyam, & Fox, 2018) evaluated four serverless computing environments in production related to the CPU performance, network bandwidth, a file I/O throughput, and concurrent invocations on serverless computing including Amazon Lambda, Microsoft Azure Functions, Google Cloud Functions, and IBM Cloud Functions.

Feng et al. (Feng, Kudva, Silva, & Hu, 2018) investigated the use of serverless runtimes while leveraging data parallelism for large models, showed the challenges and limitations, proposing modifications to the underlying runtime implementations. They pointed out that serverless runtimes can provide significant benefits. In addition, Ishakian et al. (Ishakian, Muthusamy, & Slominski, 2018) evaluated the suitability of serverless computing to measure the performance by the cost of running three different MXNet trained deep learning models on the AWS Lambda serverless computing platform. They studied the suitability of using a serverless platform for AI workloads. Kim and Lin (Kim & Lin, 2018) examined the feasibility of analytical

processing on big data using a serverless architecture. It describes the design, implementation, and performance of Flint, along with the challenges associated with serverless analytics. As a result, they show that big data analytics using a serverless architecture is in fact feasible.

Saha and Jindal (Saha & Jindal, 2018) presented an analysis of resource allocation carried out in serverless platforms. They introduce EMARS, an efficient resource management system for serverless cloud computing frameworks with the goal to enhance resource allocation (with focus on memory) among containers, building a prototype using an open-source serverless platform OpenLambda. They conducted experiments in AWS and OpenLambda to motivate the need for EMARS in resource management solution. For the analysis of memory requirements, authors wrote two functions, one, more memory intensive and the other, for the response time with different memory limits imposed on them.

Al-Ali et al. (Al-Ali et al., 2018) propose a new architecture called ServerlessOS. They present a serverless abstraction that enables the seamless and scale-out features provided by current serverless architectures while supporting a wide variety of application types in a model that programmers are familiar with the same abstraction provided by an operating system today. Perera and Perera (Perera & Perera, 2018) presented TheArchitect model, which focuses on generating best fitted microservices and serverless based high-level architecture. They demonstrated a performance evaluation of TheArchitect in terms of the processing time. They attempt to generate high-level architecture for few real-world applications containing different number of system requirements as well as summarized statistics on a conducted user study to evaluate the worth of TheArchitect as a supportive tool for both software architects and developers.

Jun et al. (Jun, Kang, Kim, & Kim, 2018) proposed a GPU-supported serverless computing framework that can deploy services faster than existing serverless computing framework using CPU. They designed a new system architecture based on Iron Functions and NVIDIA-Docker that offers the commands to allow framework users to use the GPU for the following two purposes. The first one is to use PyCUDA to deploy Python-based high-performance services. The second one is to run the enrichment code using the remote computer's GPU on a computer without a local GPU. Authors concluded that developers who want to run deep learning programs without a GPU environment can run code on remote GPUs with little performance degradation.

To the best of our knowledge, no academic literature is yet studied the usability test on the two trendiest platforms of serverless services. This led us to include the study of usability testing on AWS Lambda and Azure Function.

3 Research Approach

The research method used for this study is a quasi-experiment based on performance test and usability test (Wohlin, 2007). Therefore, we collected quantitative data and qualitative data, respectively. In this way, execution and measurement were in control. The design also included the possibility to replicate this study which is another important aspect. Despite the fact that there are some serverless computing platforms available both in commercial and open source in the market today, this study is focused on AWS and Azure as mentioned above in section 2.1.

3.1 Research Questions

The aim of this study is to better understand serverless services and help practitioners to select between two major competitors of cloud vendor by conducting a benchmark. For this, we formulated two research questions which are as follows:

RQ1: Which platform performs better, Amazon or Azure?

RQ2: Which is more user friendly, Amazon or Azure?

3.2 Usability Test

According to ISO 9241-11, usability is defined as “the extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (Bevan, Carter, Earthy, Geis, & Harker, 2016). The System Usability Scale (SUS) method is a well-known usability test. In this case, a modified SUS was used in order to evaluate the usability of the Amazon Lambda vs Microsoft Azure. It assesses whether a user can freely navigate through the screens without any difficulties or not. The key strength of the method is the possibility of evaluating application to any participant sample size, even a small one, without affecting the results (Bangor et al., 2008; Sauro & Lewis, 2011). Figure 2 shows an overview of the Usability Test Design of AWS lambda and Azure function.

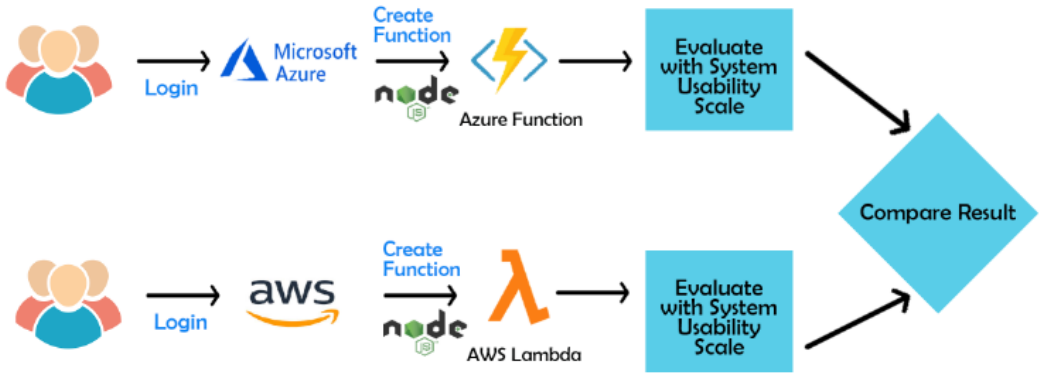

Figure 2. Usability Test Design of AWS lambda vs Azure function.

3.2.1 Test session procedure

The testing sessions were conducted on 27th March 2019 and it took 1 hour to complete the set of tasks using Azure and AWS, including the questionnaire. At the

start of the session, the participants had been shown a quick demo video in order to form a better understanding of what is expected from them. The participants were ensured that the purpose of the session is to investigate which platform is closer to the user-centered design (UCD) rules. After the video presentation, users were encouraged to take a look on their own laptop at the Amazon Lambda and Microsoft Azure Function pages. They were asked to follow a task-based scenario. After completing the set of tasks, the participants were asked to fill in a form with ten system usability scale questionnaires. The questionnaires mainly aim to understand the background of the participants and to view feedback about their overall experience.

3.2.2 Participants

The participants were a group, composed of 11 students of Høgskolen I Østfold, Halden, Norway, from the Advanced Topics of Information Systems course. The group was primarily represented by 1st year students. All of the participants were carrying out the Master in Applied Computer Science degree program. Therefore, all the participants had a lot in common regarding the study arrangements. However, their experience with Amazon Lambda and Microsoft Azure Function, in general, varied. Therefore, we defined the user classification according to two levels of knowledge of the participants. Level 1 is a user with less than one year of experience or no previous experience in the platforms and services, while Level 2 is a user who has more than one year of previous experience with these technologies.

3.3.3 Tool

The SUS statements will give a numerical score, which is used as a benchmark for estimating the user's experience. It gives ten inquiries, having a 5-point Likert scale from 1 to 5, respectively: (1) strongly disagree, (2) disagree, (3) not sure, (4) agree,

and (5) strongly agree. It is worth noting that the ISO 9241-11 standard is used to evaluate system usability by adopting a SUS modified statement. Table 1 shows modified version of SUS statements for Azure and AWS usability task test.

ID	Modified SUS Statements for Amazon web service	Modified SUS Statements for Microsoft Azure
1	I would like to use AWS service to develop applications.	I would like to use Microsoft Azure service to develop applications.
2	I found development of service to be quite complex.	I found development of service to be quite complex.
3	I thought creating applications on AWS is simple and easy to go.	I thought creating applications on Microsoft Azure is simple and easy to go.
4	I thought building the application required technical support from service provider or technical person.	I thought building the application required technical support from service provider or technical person.
5	I found the integration of services are well included.	I found the integration of services are well included.
6	I think that using AWS service contains too much inconsistency.	I think that using Microsoft Azure service contains too much inconsistency.
7	I found that using AWS service is very easy to use and learn.	I found that using Microsoft Azure service is very easy to use and learn.
8	I found the development of application on AWS unwieldy to use.	I found the development of application on Microsoft Azure unwieldy to use.
9	I felt very confident using AWS platform.	I felt very confident using Microsoft Azure platform.
10	I needed technical background before starting work on AWS platform.	I needed technical background before starting work on Microsoft Azure platform.

Table 1. Modified SUS statements

To calculate the SUS questionnaire score (Jordan, Thomas, McClelland, & Weerdmeester, 2014), for the odd numbered questions i.e. 1,3,5,7 and 9, the score input is the scale position minus 1. For questions 2,4,6,8 and 10 i.e. even numbered questions, the input is 5 minus the scale position. At the end, multiply the sum of the scores by 2.5 to obtain the overall value of the system usability. SUS scores have a range of 0 to 100, where the usability level increments respectively with the number.

The range from 0 to 25 represents worst imaginable result. Greater than 25 and lesser than 39 shows poor result range. The range from above or equal to 39 to 51 represents an average usability level, the range from 52 to 73 shows good usability of system. A result between 73 to 85 represents an excellent result and onward shows the best imaginable system usability.

3.3.4 Test Scenario

There is a different option to deploy functions on each serverless platform. In what follows, we present the test scenario used in each platform.

Microsoft Azure Function test steps

Prerequisites

1. Account on Microsoft Azure portal.
2. NPM installation.

Create Azure function

1. Login to the Azure portal.
2. Create a resource.
3. In the search plane, search for Function app.
4. Create a new function.
5. Fill in the function details with windows OS and JavaScript runtime environment.
6. Wait for the deployment to complete.

Create an HTTP triggered function

1. Click on new function.
2. Select In portal option.
3. Choose WebHook + API and create.
4. The file index.js will appear.

5. In the body variable, type Hello world.
6. Save and run the function.
7. Click on get function URL and copy it.
8. Paste the function URL into browser address bar.
9. See the response returned by the function displayed in the browser.

Amazon Web Services Lambda test steps

Prerequisites

1. Account on AWS account.
2. Installed NPM dependencies.

Create a Lambda Function with the Console

1. Sign in to the AWS Management Console.
2. In the search panel, search for Lambda.
3. Create a function.
4. Fill in the basic formation about the function such as name, runtime, execution role.
5. Name the function and set the runtime to Node.js 8.10 version.
6. Click on create function.

Create an API Gateway, which will trigger Lambda function.

1. Click on services, type API Gateway.
2. Create an API and chose REST API option.
3. Click on Create API.
4. From the menu Actions, chose create method of GET.
5. Select Integration type as the Lambda function.
6. Back to the Lambda function page.

7. Go to the Basic Settings and increase the timeout to 2 min. This is to let request/response complete before Lambda terminates function execution.
8. In the body variable, type Hello world.
9. Save and run the function.

To send the request to the Lambda function, we will need URL.

1. Click on API Gateway, you will see Invoke URL.
2. Paste the function URL into browser address bar.
3. See the response returned by the function displayed in the browser.

3.3 Performance Test

Performance testing is to determine how the system behaves and performs. A desktop environment was set and simulation was used to generate data randomly, then the outcomes were compared. In this case, the threads hit the server with various number of users, iterations, and number of queries. It will help us to understand server performance which in turn allows conclusions based on metrics. By reviewing the literature, the metrics chosen with best measurement of the results are as follows:

1. Bytes throughput over time
2. Hits per seconds
3. Latencies over time
4. Response codes per second
5. Response time distribution
6. Response time over time
7. Throughput vs. Threads
8. Time versus Thread
9. Transaction Per Second
10. Response time

3.3.1 Test Environment

We hosted a data set for Twitter tweets on both Azure and AWS database and created Azure Function and AWS Lambda APIs to call the queries. This allowed us to analyze both Azure and AWS for the aforementioned metrics. Figures 3 and 4 depict an overview of the scenario to access MSSQL database using Node.js runtime in Azure function and AWS Lambda, respectively. Moreover, Table 2 shows the Amazon and Azure setup.

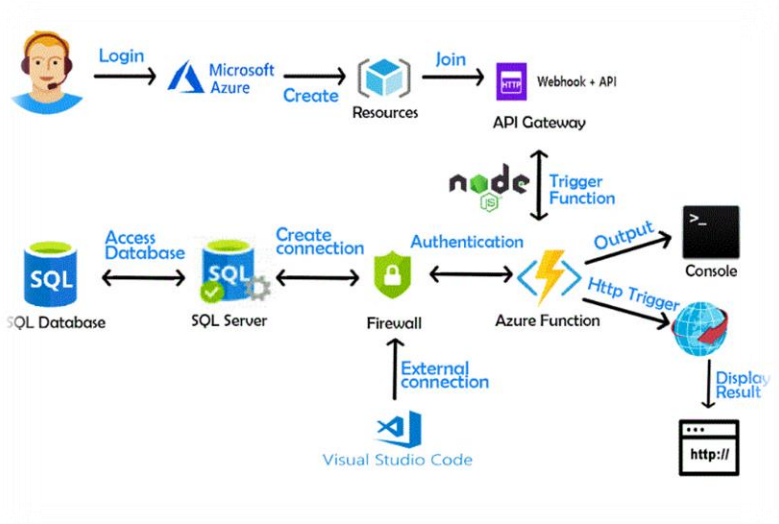

Figure 3. Scenario to access MSSQL database using Node.js runtime in Azure function.

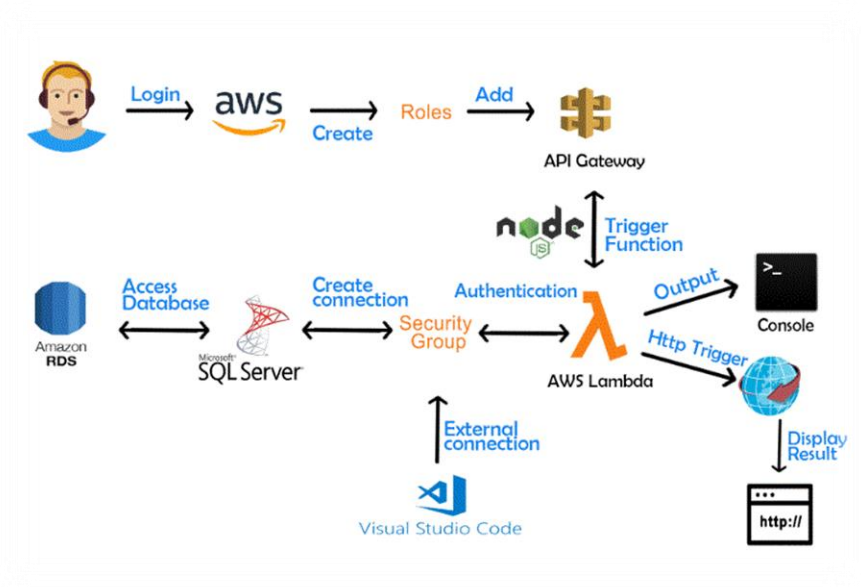

Figure 4. Access MSSQL database using Node.js runtime in AWS Lambda.

Setup	Amazon	Azure
Trigger	HTTP	HTTP
HTTP	version 2.0	version 2.0
Platform	64 bits	64 bits
Hosting plan	Pay as you go	Pay as you go
Runtime	JavaScript	Node.js 8.10
Operating system	Windows	Linux

Table 2. Setup

To access the MSSQL Database from an Azure Function using node.js:

1. Create a function in the Azure Portal using the "Webhook + API" and choose JavaScript as the runtime language.
2. While creating the function, select the option of consumption plan as a hosting plan, which defines how resources are allocated to the function app.
3. Once the MSSQL database has been created, configure firewall settings by adding an IP address to access the database remotely.
4. After that, create a Node.js application and add dependencies package such as Tedious and MSSQL to run function successfully.
5. On the other hand, to manage the database, download SQL Server Management Studio 2018.
6. Then, create a “table” named “twcs” with an identifier field and a name field.
7. In addition, upload the CSV file. To connect to the database, use the server name (e.g. server.database.windows.net) with SQL Server Authentication using the username and password to access remotely productively.

To access MSSQL Database from an AWS, using node.js function:

1. Use the Amazon Relational Database Service (RDS) to create a Microsoft SQL Server DB Instance in AWS console.
2. For the Amazon Lambda, specify the code source in a ZIP file and upload it in the Lambda function.

3. When creating the function, the function needs minimal IAM roles to operate as it is not calling any AWS Services directly.
4. In case of the node.js application, the dependencies are managed with NPM and MSSQL. Using NPM, the required libraries for interacting with the services in Amazon are included in each function while uploading the zip file into function.

In this experiment, the performance tests were run with 15Mbps internet connection on JMeter version 5.1.1. The basic setup for using Apache JMeter in our scenario is MSSQL database and two Java drivers. The Microsoft JDBC Driver 7.2 for SQL Server and JTDS Driver for SQL Server 1.3.1. They required Java version 8.

Note: Given that an IP address was setting up in this experiment, sometimes the connection to the database was temporarily unavailable. In this case, the system automatically changed IP address after few days, which will not allow to access platform database. Therefore, the IP address needs to be assigned again.

3.3.2 Dataset

Performance test of AWS and Microsoft Azure was carried out based on a Twitter dataset as mentioned before. Such a dataset is the CSV file that we want to upload into “table” named “twcs”.The dataset chosen for it was taken from the Kaggle website (kaggle.com, 2017). According to such a website, the file was updated one year ago and the file size is 167MB. The dataset consists of over 3 million tweets and replies from the biggest brands as well as a total of 93 instances and 7 attributes related to customers. In this scenario, a simple select statement was formulated to test the performance of AWS and Microsoft Azure: “'select top 5 * FROM twitter.dbo.twcs'”.

3.3.3 Tool

Apache JMeter is an open source Java-based application. It helps to measure the respond time at every load level (Stevens, 2019). JMeter simulates the client to send a request to the web server or database server (Wang & Du, 2012). JMeter tool enables to create and execute test cases on cloud and makes it possible to run the tests with the help of thousands of virtual users without requiring any setups on machines (Kılınç, Sezer, & Mishra, 2018).

3.3.4 Test Scenario

For our analysis, we conducted many tests with 5, 50 and 100 users. We used 500, 1000, 5000, 10000 and 50000 queries combination with different number of users. Furthermore, we used 1, 5 and 10 loop count for each user and query combination. In total 45 treatments were performed. Therefore, all combinations of these values are used.

4 Results

4.1 Usability Test Results

The SUS measures perceived usability of the evaluated system as well as user satisfaction. The appendix A shows the system usability test score calculation.
In the testing sessions, 63,63% of the participants were male and 90,90% (10/11) of the participants were classified in Level 1 while just one participant was in Level 2 because he has 2 years of experience with both platforms. Table 3 shows the SUS score. In summary, the SUS result of AWS lies in the category of 'Average' whereas the result of Azure lies in the category of 'Good'. This finding is supported by one of the participants who pointed out that *"By comparison of these two platforms, I should say both are quite the same but as I become a bit more precise, I realized that Microsoft Azure is easier to work with than AWS".*

Gender	Age	Participants	AWS SUS Score	Azure SUS Score
Male	26	1	50,00	40,00
Male	33	2	45,00	45,00
Female	36	3	55,00	57,50
Male	27	4	47,50	52,50
Male	36	5	42,50	50,00
Female	33	6	62,50	62,50
Male	29	7	60,00	60,00
Male	Not mention	8	55,00	45,00
Male	24	9	35,00	60,00
Female	23	10	52,50	60,00
Female	28	11	50,00	60,00
		Average	**50,45**	**53,86**

Table 3. Summary of SUS scores

After the usability testing, we came to the conclusion that both Azure and AWS are for professionals with technical expertise and background along with basic knowledge about the domain of computer science. The learning curve for both the platforms is steep and complex for new users. For a non-technical person, navigating through both Azure and AWS is like walking through a maze even with the platform's official guidelines and instructions. From a practitioner perspective, the demand for these platforms is high and users look in the market for the features provided by these platforms. However, most of the non-specialist customers are unaware of these platforms' full potential due to their complex interface. They only use these platforms to achieve one or two specific functionalities such as database hosting and are unaware of the rest.

4.2 Performance Test Results

The performance test results in detail are available on a Git repository (Sadaqat, 2019). In order to keep document compact, a summary of the results of the ten metrics is as follows.

First, *bytes throughput over time* has received nearly similar bytes per second and delay after that the data transfer ends in both the platforms, although AWS comes

ahead in this metric (see details in (Sadaqat, 2019)). The AWS overall experiment average elapsed start time is 29:08.3, which shows the time just before sending the request to just after the last response has been received, while elapsed end time is 26:33.3, whereas, Azure overall average elapsed start time is 29:20.6 with an elapsed end time of 26:19. The bytes received in a sec were 1369395.473 for AWS and 1362652.705 for Azure.

Second, *hits per second* is a metric where the total treatments average elapsed time is 28:24.2 and server hits is 5.469506 hits per second for AWS compared to 29:30.3 elapsed time and 5.415002 hits per second for Azure. Initially, AWS shows more diverse values but as the number of users increases, the number of hits for both Azure and AWS section is similar. However, one interesting results was given by the scenario of 5 users each, one iteration and combination of queries. Here, the number of hits per second for both Azure and AWS was the same, 2.5 hits per second.

Third, *latencies over time* shows that the AWS overall average elapsed time is 30:00.6 with response latencies of about 10071.86ms. In comparison, for Azure the overall average elapsed time 30:01.5 with 17190.45ms average response latencies. Based on this, we see that AWS outperforms Azure as increased response is better for the end user.

Fourth, *response code per second* is based on minimum 5 users and maximum 100 users. We receive an average elapsed time of 30:00.6 for AWS, whereas Azure has a delay of 30:01.5 before sending the request to just after the last response has been received. Here, we get 2.365768154/sec success error with 11.659346/sec error status from AWS, while on other hand, we receive a 2.724609769 per second success rate and 8.394029 per second error status from Azure.

Fifth, *response time distribution* is based on 45 treatments. Here, each treatment was repeated 10 times in order to improve the accuracy of the results for each user. The overall average shows that Azure has 37766.67615ms response times, whereas AWS has 33452.22769ms. Based on this Azure overtakes AWS in these metrics.

Sixth, *response time over time* shows the change in response time for requests over the period or duration of test. Here, AWS shows an average elapsed time of 30:00.6 per second and a response time of 30431.35ms. Whereas Azure shows the elapsed time is of 30:01.5 per second and response time of 31721.69ms. Based on these results, we note that AWS outperforms Azure as less response time and less elapsed time in results is more desirable.

Seventh, *throughput over thread* shows us the amount of throughput each thread receives in a request over the progress of the test. Here, AWS shows an average around 25 active threads receive 3.829578 transactions per second, whereas Azure — same number of active threads on average— shows an average of 3.677647615 transactions per second.

Eighth, *time versus thread* gives us information about the response time each thread receives throughout our test. As a result, we note that the approximate number of active threads of both Azure and AWS is 25. However, average of response time in AWS is less than in Azure, 24831.45ms and 30504.93ms, respectively.

Ninth, *transaction per second* is the duplicate of response code per second which includes elapsed time, and the success and failure of response per second. The aggregate report of the 45 tests shows an *average error rate* around 37,62% for AWS and 36,88% for Azure while the error rate is slightly similar.

Tenth, *response time* shows that the average response time for AWS is 24830.92ms, whereas for Azure is 30504.53846ms. For Azure, minimum response time is

22503.31ms while maximum is 58578ms delay before the request is started. For, AWS minimum time is 15827.08 and maximum is 51618.38.Therefore, even with fewer throughputs because of less delay we get less response time the standard deviation of AWS 8108.013ms is less than Azure 8964.175ms. It means that AWS has more consistency with respect to response time. As response time of AWS is less, we can see that average median, 90th percentile, 95th percentile, and 99th percentile of AWS compared to Azure is less (see Table 4).

Response time	AWS	Azure
Median	22470.08	26136.08
90^{th}	35551.31	42912.54
95^{th}	40873.92	48326.62
99^{th}	46243.15	52954.23

Table 4. Response time result

4.3 Limitation of Research

The main limitation of this performance test study was the budget. Initially the free tier subscription service was used for serverless service on both AWS and Azure platforms. But later on, after the credit became zero, the subscription was upgraded to “pay as you go”, which adds the factor of increasing and managing the budget. Furthermore, the dataset was another major limitation so that further studies should replicate this experiment using other datasets and different test scenarios (see section 3.4.4).

For the usability test, a major limitation was the small sample size. There was only one participant in Level 2, whereas all others were in Level 1. If the ratio of Level 1 and Level 2 were the same, the result would have been significantly more balanced. Moreover, the test scenario is limited due to the scope of this exploratory study (see section 3.3.4) but, nevertheless, it provides a preliminary overview of usability. Therefore, further research is also needed.

4.4 Replication package

Given that the replication is one of the fundamentals of the experimental methods, a replication package is available on a Git repository (Sadaqat, 2019) in order to increase the validity and reliability of the result. By providing a replication package, it will facilitate others to replicate or to reproduce this experiment. The replication package includes the file of data which has the dataset URL address and SUS questionnaire. The second file called by name procedure presents the experiment overview which explains test environment, the tool used for the experiment, and the test code. This folder also includes subfolders by test scenario title including performance test and usability test. The third folder and fourth folder contain performance results and SUS calculation result score, respectively.

5. Conclusion

The main findings of this study revealed that AWS and Microsoft Azure platforms provide similar core capabilities you would expect but there are considerable differences in terms of structure, creations of services, configuration of function and so on. We came to the conclusion that both Azure and AWS perform quite similar to some extent. Azure performs better marginally in a certain scenario, whereas AWS performs better in some situations than Azure. The real difference between the two platforms lies in usability of these services. In other words, the same functionality is provided in two very different interfaces. AWS Lambda requires complex network and database configuration including IAM (identity access management), NAT (Network Address Translation), inbound, outbound rules for network traffic, and various policies roles. Whereas Azure only requires user IP address to be assigned to Azure firewall for access, including database and HTTP trigger for function.

According to the usability test session, most participants suggested that they found Azure to be more user friendly and easy to use. They were able to work on Azure and understand their interface quicker than AWS. However, both Azure and AWS are professional level platforms aimed towards users with stronger technical background along with knowledge about the domain of computer science. The learning curve for both of the platforms is steep and complex for new users. For a non-experience user, navigating through both Azure and AWS is like walking through a maze even with the platform's guidelines and instructions. From a technical perspective, even for a developer from a strong technical background, configuring everything, from complex network settings to compound database options, it is sometimes confusing and timely to understand the procedures. However, it is worth noting that Azure interface and its documentation were easier to understand and navigate through.

We concluded that Azure and AWS can extend their market and capture more customers by making their interfaces simpler and more user friendly, according to the tests conducted. At the moment, both platforms' interfaces seem to be targeted towards technical users, but they should be made more user friendly for non-experience users as well, by developing a more selection-oriented interface instead of the current configuration oriented interfaces. There is also a need for automation in most aspects of the interface such as in network configuration and database setting such as IAM (identity and access management) in AWS. They should be replaced with pop-up dialogs with functional and procedural description instead of technical configuration. Currently, this is not something unheard of as we have some examples such as Wordpress and Wix. However, it is an exploratory study about usability and more research is needed to understand how usability could be implemented and its

real benefits. Therefore, as a future work would be interesting to perform a focus group in order to gain more insights of users about the usability.

Given that it is important to facilitate replication of experiments, a replication package is available on a Git repository (Sadaqat, 2019). It may be used for more exploratory studies and more knowledge may be gained. For further research, it would be interesting to perform a load and stress test on AWS Lambda and Azure function. An additional performance of benchmarking based on financial perspective could be useful. Finally, in any future research, the implementation of more services such as those for DevOps and internet of things, with the combination of current serverless services should be conducted.

REFERENCES

Al-Ali, Z., Goodarzy, S., Hunter, E., Ha, S., Han, R., Keller, E., & Rozner, E. (2018). Making Serverless Computing More Serverless. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 456–459. https://doi.org/10.1109/CLOUD.2018.00064

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., … Suter, P. (2017). Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing* (pp. 1–20). https://doi.org/10.1007/978-981-10-5026-8_1

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction*, *24*(6), 574–594. https://doi.org/10.1080/10447310802205776

Bevan, N., Carter, J., Earthy, J., Geis, T., & Harker, S. (2016). New ISO Standards for Usability, Usability Reports and Usability Measures. *Proceedings, Part I, of the 18th International Conference on Human-Computer Interaction. Theory, Design, Development and Practice - Volume 9731*, 268–278. https://doi.org/10.1007/978-3-319-39510-4_25

Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking Cloud Serving Systems with YCSB. *Proceedings of the 1st ACM Symposium on Cloud Computing*, 143–154. https://doi.org/10.1145/1807128.1807152

Eivy, A. (2017). Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing*, *4*(2), 6–12. https://doi.org/10.1109/MCC.2017.32

Eyk, E. van, Toader, L., Talluri, S., Versluis, L., Uță, A., & Iosup, A. (2018). Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing*, *22*(5), 8–17. https://doi.org/10.1109/MIC.2018.053681358

Feng, L., Kudva, P., Silva, D. D., & Hu, J. (2018). Exploring Serverless Computing for Neural Network Training. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 334–341. https://doi.org/10.1109/CLOUD.2018.00049

Hardin, T. (2018). 2018 Digital Trend: Serverless Technology [Educational]. Retrieved July 24, 2018, from G2 Crowd website: https://blog.g2crowd.com/blog/trends/digital-platforms/2018-dp/serverless-computing/

Ishakian, V., Muthusamy, V., & Slominski, A. (2018). Serving Deep Learning Models in a Serverless Platform. *2018 IEEE International Conference on Cloud Engineering (IC2E)*, 257–262. https://doi.org/10.1109/IC2E.2018.00052

Jordan, P. W., Thomas, B., McClelland, I. L., & Weerdmeester, B. (2014). *Usability Evaluation In Industry*. CRC Press.

Jun, T. J., Kang, D., Kim, D., & Kim, D. (2018). GPU Enabled Serverless Computing Framework. *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 533–540. https://doi.org/10.1109/PDP2018.2018.00090

kaggle.com. (2017). Customer Support on Twitter [Dataset]. Retrieved May 8, 2019, from https://kaggle.com/thoughtvector/customer-support-on-twitter

Kanso, A., & Youssef, A. (2017). Serverless: Beyond the Cloud. *Proceedings of the 2Nd International Workshop on Serverless Computing*, 6–10. https://doi.org/10.1145/3154847.3154854

Kim, Y., & Lin, J. (2018). Serverless Data Analytics with Flint. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 451–455. https://doi.org/10.1109/CLOUD.2018.00063

Kılınç, N., Sezer, L., & Mishra, L. (2018). Cloud-Based Test Tools: A Brief Comparative View. *Cybernetics and Information Technologies*, *18*(4), 3–14. https://doi.org/10.2478/cait-2018-0044

Klimovic, A., Wang, Y., Kozyrakis, C., Stuedi, P., Pfefferle, J., & Trivedi, A. (2018). Understanding Ephemeral Storage for Serverless Analytics. *2018 USENIX Annual Technical Conference (USENIX ATC '18)*. Presented at the 2018 {USENIX} Annual Technical Conference ({USENIX} {ATC} 18), Boston, MA. Retrieved from https://www.usenix.org/system/files/conference/atc18/atc18-klimovic-serverless.pdf

Lee, H., Satyam, K., & Fox, G. (2018). Evaluation of Production Serverless Computing Environments. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 442–450. https://doi.org/10.1109/CLOUD.2018.00062

McGrath, G., & Brenner, P. R. (2017). Serverless Computing: Design, Implementation, and Performance. *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 405–410. https://doi.org/10.1109/ICDCSW.2017.36

Perera, K. J. P. G., & Perera, I. (2018). TheArchitect: A Serverless-Microservices Based High-level Architecture Generation Tool. *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, 204–210. https://doi.org/10.1109/ICIS.2018.8466390

Qu, L., Wang, Y., & Orgun, M. A. (2013). Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment.

2013 IEEE International Conference on Services Computing, 152–159. https://doi.org/10.1109/SCC.2013.92

Sadaqat, M. (2019, September 30). Performance test results. Retrieved from Severless Service Performance Testing website: https://github.com/mubasha/Serverless-service-performace-testing.git

Sadaqat, M., Colomo-Palacios, R., & Emil Skrimstad Knudsen, L. (2018). SERVERLESS COMPUTING A MULTIVOCAL LITERATURE REVIEW. *Proceedings from the Annual NOKOBIT Conference Held at Svalbard the 18th-20th of September 2018*, *26*. Retrieved from http://ojs.bibsys.no/index.php/Nokobit/article/view/544

Saha, A., & Jindal, S. (2018). EMARS: Efficient Management and Allocation of Resources in Serverless. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 827–830. https://doi.org/10.1109/CLOUD.2018.00113

Sauro, J., & Lewis, J. R. (2011). When Designing Usability Questionnaires, Does It Hurt to Be Positive? *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2215–2224. https://doi.org/10.1145/1978942.1979266

Scale, R. (2018). RightScale 2018 State of the Cloud Report [Report]. Retrieved August 18, 2018, from https://www.rightscale.com/lp/state-of-the-cloud

Stevens, S. (2019, August 5). JMeter—Open Source Functional and Load Testing. Retrieved May 8, 2019, from http://www.methodsandtools.com/tools/tools.php?jmeter

van Eyk, Erwin, Iosup, A., Abad, C. L., Grohmann, J., & Eismann, S. (2018). A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures. *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 21–24. https://doi.org/10.1145/3185768.3186308

Vázquez, C., Ramki Krishnan, R., & John, E. (2014). *Cloud Computing Benchmarking: A Survey*. Retrieved from /paper/Cloud-Computing-Benchmarking-%3A-A-Survey-V%C3%A1zquez-Krishnan/22bcd82109ac22d9f379fbb55ba2d8912b2efb7f

Völker, C. (2018). *Suitability of serverless computing approaches*. http://dx.doi.org/10.18419/opus-9711

Wang, F., & Du, W. (2012). A Test Automation Framework Based on WEB. *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 683–687. https://doi.org/10.1109/ICIS.2012.21

Wohlin, C. (2007). Empirical Software Engineering: Teaching Methods and Conducting Studies. In V. R. Basili, D. Rombach, K. Schneider, B. Kitchenham, D. Pfahl, & R. W. Selby (Eds.), *Empirical Software Engineering Issues. Critical Assessment and Future Directions: International Workshop, Dagstuhl Castle, Germany, June 26-30, 2006. Revised Papers* (pp. 135–142). https://doi.org/10.1007/978-3-540-71301-2_42

APPENDIX A

Table 5 shows the system usability score calculation.

SUS Calculation														
	Experience year(s) and month(s)	**Gender**	**Age**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Score**
AWS	0	Male	26	4	3	4	5	4	2	5	4	2	5	50,00
	0	Male	33	3	4	3	2	4	2	2	4	3	5	45,00
	0	Female	36	3	2	4	3	3	3	3	3	3	3	55,00
	0	Male	27	4	3	3	3	3	3	4	4	3	5	47,50
	0	Male	36	4	3	2	4	3	3	2	3	3	4	42,50
	0	Female	33	3	2	4	4	5	3	5	3	4	4	62,50
	2 year	Male	29	4	3	4	4	4	2	4	3	4	4	60,00
	Not mention	Male	Not mention	4	3	3	2	4	2	4	4	3	5	55,00
	Not mention	Male	24	3	4	2	3	2	3	2	4	3	4	35,00
	0	Female	23	4	3	3	4	4	1	3	3	2	4	52,50
	0	Female	28	3	3	2	4	4	2	3	2	3	4	50,00
														50,45
Azure	**Experience** year(s) and month(s)	**Gender**	**Age**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Score**
	0	Male	26	4	3	4	4	3	4	2	4	3	5	40,00
	3 month	Male	33	3	3	4	4	3	2	2	3	2	4	45,00
	Not mention	Female	36	3	3	4	3	3	2	4	2	2	3	57,50
	0	Male	27	4	3	4	3	3	3	4	3	3	5	52,50
	0	Male	36	4	3	3	4	4	3	2	3	4	4	50,00
	0	Female	33	4	2	4	5	4	3	5	3	4	3	62,50
	2 year	Male	29	4	3	4	4	4	2	4	3	4	4	60,00
	Not mention	Male	Not mention	3	4	3	3	3	2	4	4	2	4	45,00
	6 month	Male	24	4	3	4	3	3	3	3	2	4	3	60,00
	0	Female	23	4	3	4	3	4	2	4	3	3	4	60,00
	0	Female	28	4	2	4	4	4	1	3	3	3	4	60,00
														53,86

Table 5. SUS Calculation