2016 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing

Infrastructure Cost Comparison of Running Web Applications in the Cloud using AWS Lambda and Monolithic and Microservice Architectures

Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro
COMIT Research Group, Systems and Computing Engineering Department
Universidad de los Andes
Bogotá D.C., Colombia
Email: {mj.villamizar24, ok.garces10, lm.ochoa750, hcastro}@uniandes.edu.co

Lorena Salamanca, Mauricio Verano, Rubby Casallas
TICSw Research Group, Systems and Computing Engineering Department
Universidad de los Andes
Bogotá D.C., Colombia
Email: {l.salamanca10, m.verano239, rcasalla}@uniandes.edu.co

Santiago Gil, Carlos Valencia, Angee Zambrano, Mery Lang
Project Management Department
Mapeo
Bogotá D.C., Colombia
Email: {sgil, cvalencia, azambrano, mlang}@heinsohn.com.co

***Abstract*—Large Internet companies like Amazon, Netflix, and LinkedIn are using the microservice architecture pattern to deploy large applications in the cloud as a set of small services that can be developed, tested, deployed, scaled, operated and upgraded independently. However, aside from gaining agility, independent development, and scalability, infrastructure costs are a major concern for companies adopting this pattern. This paper presents a cost comparison of a web application developed and deployed using the same scalable scenarios with three different approaches: 1) a monolithic architecture, 2) a microservice architecture operated by the cloud customer, and 3) a microservice architecture operated by the cloud provider. Test results show that microservices can help reduce infrastructure costs in comparison to standard monolithic architectures. Moreover, the use of services specifically designed to deploy and scale microservices reduces infrastructure costs by 70% or more. Lastly, we also describe the challenges we faced while implementing and deploying microservice applications.** [1]

***Keywords*—cloud computing, microservices, service oriented architectures, scalable applications, software engineering, software architecture, microservice architecture, AWS Lambda, Amazon Web Services**

I. Introduction

Cloud computing [1] is a model that allows companies to deploy enterprise applications that, if properly designed, can scale their computing resources on demand. While companies try to avoid the hassle and the development costs of migrating to the cloud, most of them start by deploying traditional and monolithic applications that use IaaS/PaaS solutions. In this context a *monolithic application* refers to an application with a single large codebase/repository that offers tens or hundreds of services using different interfaces such as HTML pages, Web services, or/and REST services.

Microservice architectures propose a solution to scale computing resources efficiently and solve many other issues presented in monolithic architectures [2]. However, microservice architectures face other challenges such as the effort required to deploy each microservice, and to scale and operate them in cloud infrastructures. To address these concerns, services like AWS Lambda [3] allow implementing microservice architectures without the need of managing servers. Thus, it facilitates the creation of functions (i.e. microservices) that can be easily deployed and automatically scaled, and it also helps reduce infrastructure and operation costs.

The remainder of this paper is organized as follows: Section II presents different efforts related to the microservices architecture pattern. The description of the case study that was developed and tested is presented in Section III. Section IV describes the application deployments on the AWS infrastructure. Section V shows the results of performance tests executed for each architecture in order to compare their infrastructure costs. Section VI concludes and presents several research lines that can be addressed.

II. Related work

To avoid the problems of monolithic applications and take advantage of some of the SOA architecture benefits, the microservice architecture pattern has emerged as a lightweight subset of the SOA architecture pattern. This pattern is being used by companies like Amazon [4], Netflix [5] and LinkedIn [6] to support and scale their applications and products. The microservice pattern [7] proposes to divide an application A into a set of small business services μS (μS1, μS2, μSn), each one of them offering a subset of the services S (S1, S2, Sx) provided by application A. Each microservice is developed using independent codebases and the team μTi is also in charge of deploying, scaling and operating the microservice in a cloud computing IaaS/PaaS solution.

One of the concerns of implementing microservices is related to the efforts required to deploy and scale each microservice/gateway in the cloud. Although companies implementing microservices can use different DevOps [8] automation tools such as Docker, Chef, Puppet, auto scaling, among others, the implementation of those tools consumes time and resources. To

[1]This work is the result of a research project partially funded by Colciencias under contract 0569-2013 "Construcción de una línea de productos de aplicaciones de crédito/cartera que serán ofrecidas a través de un Marketplace en forma SaaS para el sector no bancarizado (PYMES) de Colombia" COD. 1204-562-37152.

978-1-5090-2453-7/16 $31.00 © 2016 IEEE
DOI 10.1109/CCGrid.2016.37

address this concern, cloud providers like AWS have recently launched services such as AWS Lambda, which allows the deployment of microservices without the need of managing servers. This service is designed to offer a per request cost structure, meaning that developers only have to worry about writing individual functions to implement each microservice and then deploying them on AWS Lambda.

III. Case Study

In order to evaluate the implications of using microservices operated by the cloud customer and the cloud provider in a real scenario versus using a monolithic architecture, we worked with a software company in the development of an application that uses the three aforementioned architectures. The application was designed to support the business process of generating and querying payment plans for loans of money delivered by an institution to its customers. In order to provide a simple case study we considered two services from the complete set of services offered by the original application.

The first service, called S1, was in charge of generating a payment plan with the set of payments (from 1 to 180 months). This service implemented CPU intensive algorithms to generate payment plans, did not store information in the database. The second service, called S2, was responsible for returning an existing payment plan previously stored in the database with its respective set of payments. The typical response times of S1 and S2 were around 3000 and 300 milliseconds respectively. Below, we describe the three architectures that were defined to develop the application:

A. Monolithic architecture

In order to develop the application using a typical monolithic approach, we kept in mind that it should have a single codebase and it should be developed using an MVC web application framework. In this architecture, the web application publishes the two services as REST services over the Internet, and they are consumed by the MVC front-end application executed in the browser. This approach can be deployed in a single-server environment where the scalability is limited to the computing resources of one server, or it can be deployed in a multi-server environment.

B. Microservice architecture operated by the cloud customer

For simplicity, in this case study we selected two micro services (μS1 and μS2), one for each service of the monolithic application. Each microservice may be developed as an independent three-tier application using different technological stacks. Given that μS1 only generates new payment plans without storing information, it does not need the persistence layer. In contrast, μS2 returns the complete information of a payment plan saved in the relational database, therefore, it requires persistence. The gateway does not store any information, so, it does not need a persistence layer. In this architecture, the gateway and each microservice can be scaled independently.

C. Microservice architecture operated by AWS Lambda

Functions are the unit of execution/development in AWS Lambda and they can be developed in Java 8, Python, or Node.js. In contrast with the microservice architecture, where both gateway services were implemented in the same web application, in AWS Lambda the two gateway services must be implemented as two independent functions that receive requests from end-users (browsers) through the Internet, consume microservice functions through REST, get the results from microservice functions, and return the results to end-users. Each gateway function uses the Internet to publish a REST service, which is consumed by the MVC front-end application.

IV. Deployment in a Cloud Computing Infrastructure

In order to compare the infrastructure costs required to run each architecture, the three architecture implementations were deployed in AWS. We started the comparison by defining a baseline architecture for the monolithic architecture. Then, we executed the performance tests described in Section V, in order to calculate the number of requests per minute that the monolithic architecture supported, in both stacks (Play and Jax-RS). Based on that performance, we defined a similar infrastructure for the deployment of the microservice architecture with the goal of supporting a similar number of requests per minute. Finally, we executed individual tests to identify the best configuration to execute each function in AWS Lambda. We describe the deployment and infrastructure services used for each architecture below.

A. Monolithic architecture deployment

The monolithic architecture, developed in Play and Jax-RS, was deployed on AWS as shown in Figure 1.

B. Microservice architecture deployment

The microservice architecture operated by the cloud customer, whose applications were developed in Play, was deployed as illustrated in Figure 2.

C. Deployment of the AWS Lambda architecture

The microservice architecture operated by AWS Lambda, whose services were developed in Node.js, was deployed as shown in Figure 3. The microservice/gateway functions were implemented a four independent functions of type microservice-http-endpoint.

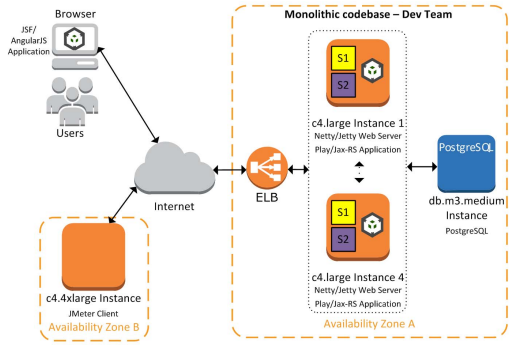

Fig. 1. Deployment of the monolithic architecture on Amazon Web Services

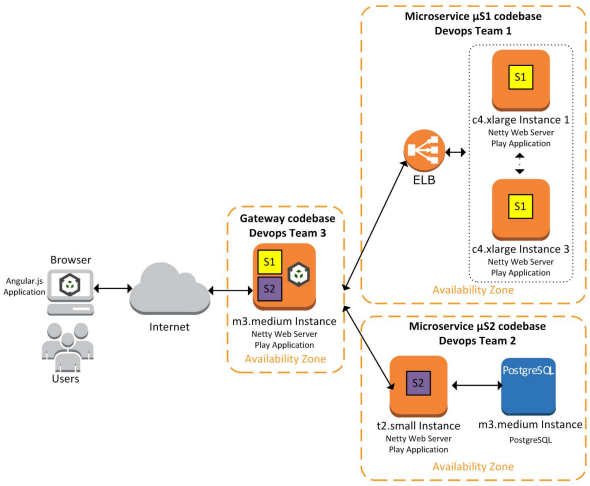

Fig. 2. Deployment of the microservice architecture on Amazon Web Services

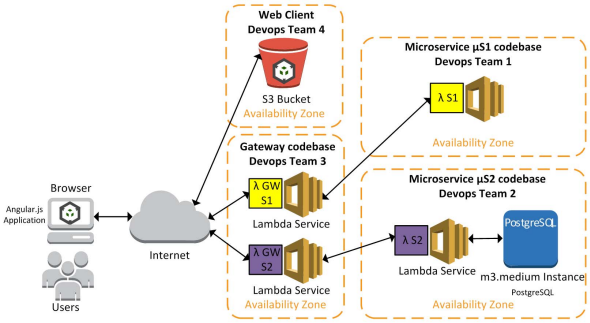

Fig. 3. Deployment of the Amazon Web Services Lambda architecture

V. Test and Results

A. Performance tests

In order to test and compare the performance and infrastructure costs of the three architectures, we defined three business scenarios. The first scenario defines that 20% of the requests made to the web application consumed service S1 and 80% consumed service S2. We called this the 20/80 scenario. In the second scenario, each service receives 50% of the requests and it is called 50/50. In the third scenario, called 80/20, 80% of the requests consumed service S1 and 20% consumed service S2.

To execute the stress tests of each architecture, we configured JMeter [9] 2.13 in an AWS c4.large EC2 instance, and we defined the maximum response time of S1 and S2 in 20.000 and 3.000 milliseconds, respectively (their typical response times are 3.000 and 300 ms, respectively).

The stress tests were executed with the goal of identifying the maximum number of requests supported by the monolithic architecture in Play and Jax-RS. This number was calculated by increasing the number of requests for each scenario until the application began to generate errors or the response time defined for S1 and S2 was not met. The results of the performance tests executed to the monolithic architecture are shown in Figure 4. These results show that, when considering the

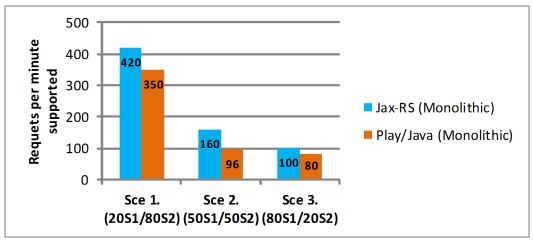

Fig. 4. Performance results of the Monolithic Architecture in Play and Jax-RS on AWS

monolithic architecture, Jax-RS provides a better performance than Play.

Based on the number or requests per minute supported by the monolithic architecture, we executed performance tests for the microservice architecture operated by the cloud customer, in order to identify the minimum infrastructure needed for supporting the same number of requests. The performance tests results for the microservice architecture are illustrated in Figure 5. These results show that, in our case study, microservices can provide an even better performance than Jax-RS in the monolithic architecture, at a lower cost. The monthly cost of the microservice architecture is $390,96 USD in contrast to the $403,20 USD related to the monolithic architecture; furthermore, the former supported more requests per minute in the three defined scenarios.

Similarly, costs in AWS Lambda are based on the number and the configuration of requests per executed microservice/gateway function. Therefore, the performance tests and cost estimation performed for the deployment shown in Figure 3 were defined to support the same number of requests per minute as the microservice architecture.

B. Cost comparison

Given that each architecture was deployed in different infrastructures, we defined and calculated the metric *Cost per Million of Requests (CMR)* for each architecture in the three scenarios, in order to easily compare their execution costs. For each scenario and architecture, this metric was calculated by dividing the monthly infrastructure costs by the number of requests supported per month, which is calculated by multiplying the number of requests supported per minute by 43,200 —the number of minutes per month (60*24*30)—. We assumed a constant throughput per minute during a month. The CMR metric for each architecture is shown in Figure 6.

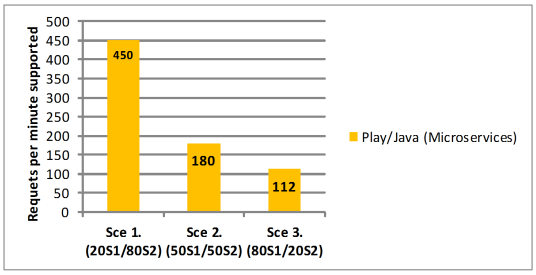

Fig. 5. Performance results of the Microservice Architecture in Play on AWS

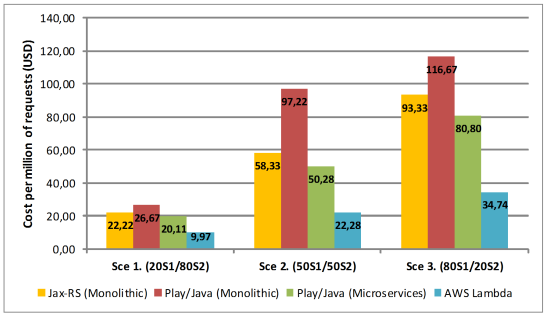

Fig. 6. Cost Comparison of The Three Architectures per Million of Requests

These results show that the microservice architecture implemented with Play can help reduce up to 9.50%, 13.81% and 13.42% of the costs per scenario in contrast to the monolithic architecture implemented with Jax-RS. However, the use of services exclusively designed to implement microservices, such as the third architecture implemented in AWS Lambda, can help reduce up to 50.43%, 55.69% and 57.01% of the costs per scenario in contrast to the microservice architecture; up to 55.14%, 61.81% and 62.78% of the costs per scenario in contrast to the monolithic architecture implemented in Jax-RS; and up to 62.61%, 77.08% and 70.23% of the costs per scenario in contrast to the Play monolithic architecture.

C. Response time

To identify how each architecture affects the response time to requests during peak periods, we measured the Average Response Time (ART) during the performance tests. The ART for S1 and S2 are shown in Figure 7 and Figure 8, respectively. The ART for S1 and S2 in the monolithic architectures with Play and Jax-RS are similar; however, when considering the microservice architecture, the ART for both services increases because each request must pass between the gateway and each microservice. For the microservice architecture operated by AWS Lambda, the ART remains very similar for both services in the three scenarios, but it varied between scenarios for the other architectures. This result was obtained because each function/request in AWS Lambda is executed in an isolated computing environment dedicated to a single request, while in the other architectures each web server executes several concurrent requests.

VI. Conclusion and Future Work

Based on the performance tests executed for a monolithic architecture, a microservice architecture operated by the cloud

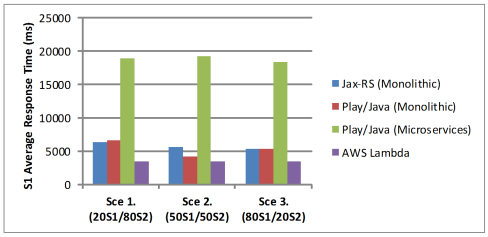

Fig. 7. Average Response Time for S1 During Peak Periods

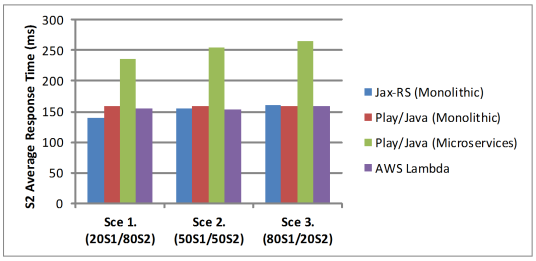

Fig. 8. Average Response Time for S2 During Peak Periods

customer, and a microservice architecture operated by AWS Lambda, we can conclude that the use of microservices can help reduce up to 13.42% of infrastructure costs. However, the use of emerging cloud services such as AWS Lambda, exclusively designed to deploy microservices at a more granular level (per HTTP request/function), allows companies to reduce their infrastructure costs in up to 77.08%. Microservices also enable large applications to be developed as a set of small applications that can be independently implemented and operated, thus managing large codebases by using a more practical methodology, where incremental improvements are executed by small teams on independent codebases. Agility, cost reduction and granular scalability must be balanced with the development efforts, technical challenges, and costs incurred by companies resulting from microservices requiring the adoption of new practices, processes, and methodologies.

The work in this paper was focused in comparing the impact of using microservices in the infrastructure costs, as future work we plan to evaluate the costs incurred by companies during the process of implementing microservices. The process of defining the number of microservices to implement, and the tools required to automate the deployment of microservices in general purpose IaaS solutions and services such as AWS Lambda, will also be evaluated. Tests to evaluate other technical concerns regarding microservices such as failure tolerance, distributed transactions, data distribution, service versioning, and microservice granularity, are also interesting research areas.

References

[1] R. Buyya, "Cloud computing: The next revolution in information technology," in *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, pp. 2–3, Oct 2010.

[2] J. Lewis and M. Fowler, "Microservices," 2014.

[3] A. W. Services, "Aws lambda," 2015.

[4] GIGAOM, "The biggest thing amazon got right: The platform," 2011.

[5] Nginx, "Adopting microservices at netflix: Lessons for architectural design," 2015.

[6] InfoQ, "From a monolith to microservices + rest: the evolution of linkedin's service architecture," 2015.

[7] Microservices, "Pattern: Microservices architecture," 2014.

[8] F. Nemeth, R. Steinert, P. Kreuger, and P. Skoldstrom, "Roles of devops tools in an automated, dynamic service creation architecture," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 1153–1154, May 2015.

[9] D. Rahmel, "Testing a site with apachebench, jmeter, and selenium," in *Advanced Joomla!*, pp. 211–247, Apress, 2013.