# Performance analysis of Monolithic and Micro service architectures – Containers technology

[1] Alexis Saransig, [2] Freddy Tapia

[1] Universidad Técnica del Norte,
Av. 17 de Julio, 5-21, 100105 Ibarra – Ecuador
[2] Universidad de las Fuerzas Armadas ESPE,
Av. General Rumiñahui s/n, 171-5-231B, Sangolquí - Ecuador
[1] afsaransigc@utn.edu.ec; [2] fmtapia@espe.edu.ec

**Abstract.** Comparative analysis of the performance of hardware resources, between Monolithic Architecture and Micro services Architecture, using virtualization technology based on development and production environments. Today, the new trend is the development and/or deployment of applications in the Cloud, in this aspect, monolithic applications have flexibility, scalability, maintainability and performance limitations. On the other hand, the focus of Microservices adapts to new trends and solves these limitations. Meanwhile, virtualization with virtual machines is currently not efficient enough with hardware resources. With the appearance of containers, this problem is solved due to its functioning characteristics as independent processes and resources optimization. Now, two scenarios are presented, the first consisting of a Web application based on a Monolithic Architecture that is executed in a Kernel based Virtual Machine - KVM and the second scenario shows the same Web application, this time, based on a Micro services Architecture and running in containers. Each scenario is subjected to the same stress tests; the generated data are recorded in "log" files for further analysis. The hardware resources are the same for both scenarios. The comparison of these scenarios helps to identify the efficiency of the Application and the hardware resources, as well as the development and/or deployment of Applications. This can be improved with the use of Microservices and Containers. In addition, the reduction of costs that would imply the optimization in the resources. For greater reliability in the interpretation of the data, two analysis tools were used: JMeter and NewRelic. Finally, the two resulting cases from the analysis are shown, each case being considered due to the feasibility of the same depending on the needs and availability of resources

**Keywords:** Monolithic Architecture, Micro services, Containers, Performance.

## 1 Introduction

Technological evolution aims to be more efficient in the use of resources. Software production must considerer different types of architectures, so that each product, would fulfill its objectives and be efficient in the use of resources. Monolithic Architecture is an example of this case and commonly used in Software production. Its fusion with Virtual Machines has turned it into a successful and effective formula for small and large-scale projects. In general, a monolithic application declines in its performance once it starts to grow more than expected. The solutions can be varied, for instance, migration to new technologies, management of independent services, and more powerful servers. In the long term, it can generate high costs due to their limited capacity of escalation and maintainability. Innovation has given rise to new architectures that propose optimal solutions to improve the Software production process [12].

Micro services Architecture arises to provide solutions and to be a deciding factor when an appropriate decision, as to DevOps concerns and future projects that relate, for benefits this presents. The Microservices Architecture replaces the Monolithic with a lightweight, narrowly focused distributed system and isolated service [13].

Container technology is still little known in our environment. However, trends and specialized reports show that there is more acceptance for this technology since it provides a more efficient resources management compared to Virtual Machines.

In this research, a comparative performance analysis is made between applications with Monolithic Architecture running on a Virtual Machine against the same application, but in a version based on a Micro services Architecture using Containers. Both scenarios will be running on a computer with the same characteristics, they were subjected to stress tests, in order to analyze the data stored in "logs" files.

With the obtained results, decision-making is facilitated in terms of the efficient management of resources and the production of Software. In addition, benefits related to new development methodologies, that are not detailed in depth in this investigation are set. However, they remain open for further investigation

## 2 Background

### 2.1 Monolith architecture

It usually uses a single technology development, which limits the availability of a suitable tool for each task to be executed by the system. A single logical executable [1], in which any change made in a part of this type of system involves the construction and deployment of a new version of the entire system. It involved mostly aspects or layers like presentation, processing and storage in a single software component that is supposed to run on a single server.

The advantages are few; one of them is the minimum requirement of changes that can be made in their context. Meanwhile, its disadvantages fall in different areas such as maintenance, debugging, scalability, distribution and deployment. The efficiency is quantified due to the minimal changes that are made in its context; meanwhile, its shortcomings lie in the difficulty of performing maintenance tasks, debugging, scaling, distribution and implementation.

Until now, the technological trend has been focused on the development of Software, offering solutions focused on a Monolithic Architecture. Habit or facility based on knowledge, leads to trust the same old solutions even knowing the short or long term risks that can be found.

Every computer system or technological implementation is bound to evolve, just as solid disks have plunged punch cards. The level of complexity of modern systems requires improvements in both the production and the performance of the Software. This means that in the Monolithic Architecture have been found inevitable defects [2], which over time will play against it. This leading to modern architectures, such as Micro services.

## 2.2 Microservices architecture

It is a relatively new architecture without formal definition yet. It is used to build large, complex and scalable applications. It is composed by small, independent and highly recoupable processes, communicating each one through API's [3].

With the features mentioned above, not only a solution is given to almost all the weak points of a Monolithic Architecture, but also it even proposes efficient solutions in the short and long term.

Functions composed of small and individual services, running in their own process and communicating with light application mechanisms. The independence of each one becomes fault tolerant and increases their availability. It is a new culture of automation, with decentralized processes, that allow independent deployments. Therefore, the whole is modeled around the core of the business [4] [5] [6].

Architecturally the structure of an application based on Micro services, differs in a great level from the Monolithic Architecture (Fig. 1 shows an example)

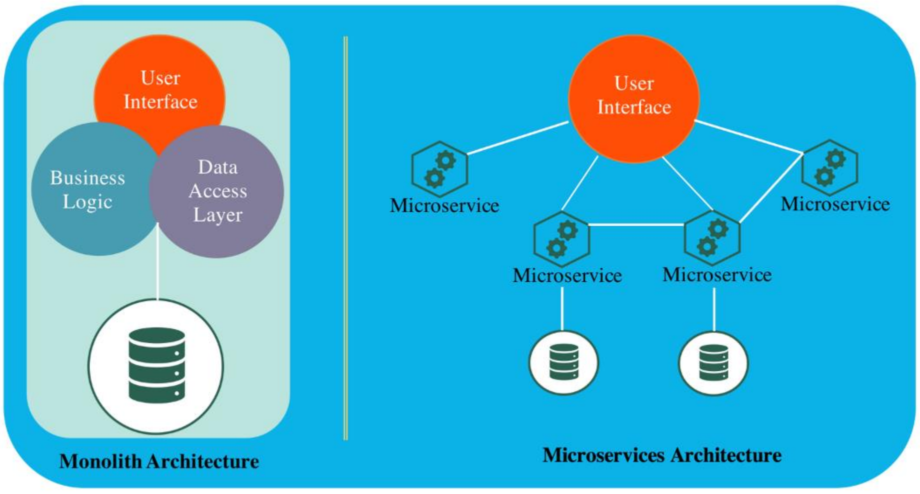


**Figure. 1.** Representation of the basic structure - Application with Monolithic Architecture and Micro services Architecture

## 2.3 KVM

The Kernel Virtual Machine – KVM, is an advanced virtualization technology that functions as a Linux process and is classified as a Hypervisor type I[1] [7].

Its administration is mainly done through the use of a Command Line Interface - CLI. Its allied application is QUEMU, also known as a generic and open source machine emulator and virtualize [8]. A KVM supports hot migration, which means that it allows physical servers or even Full Data Centers for maintenance without interrupting the Guest Operating System [9].

To facilitate the management of the KVM, the Virtual Machine Manager application brings a graphical interface to be used.

## 2.4 Containers

The containers are quite similar to virtual machines with the service these provide. The only exception is that they do not have an overhead that manages the execution of a separate kernel and the virtualization of all hardware components [9].

Containers enable each workload to have exclusive access to resources such as processor, memory, service account and libraries, which are essential for the development process [10]. In addition, they run as a group of isolated processes within an operating system. Like this, it optimizes startup times and easy maintenance. Each container is a package of libraries and dependencies necessary for its operation; therefore they area independent, hence the term of isolated processes.

One of the most critical risks in containers is the poor visibility of the processes that run inside a container with respect to the limit of resources which the host machine has [11].

As for safety, this is handled by management groups and namespaces permits. The users do not have the same treatment inside and outside the container.

Docker is broadly leading this technological segment as far as container handling is concerned. This Open Source technology has become the most commonly used tool by DevOps for container management in large-scale projects.

It has even been suggested as a solution for more interoperable cloud applications packaging [14] [15].

Docker initially using Linux Containers - LXC as default execution environment. However, changing distributions was a problem when trying to standardize a generic version. Docker, in version 0.9, includes its own runtime environment called LibContainer, which helped him become a standard and cross-platform technology Docker.

[1] Works at the hardware level without an intermediary Operating System.

## 3 System Approach

This research proposes the implementation of an application with modern technology, in addition, to adapting environments where it can be executed and subjected to stress tests to calculate the level of performance.

### 3.1 Application design

The application uses modern programming technology based on Node.js, SQL Databases (MySQL) and NoSQL (MongoDB). The application represents a basic design - in Backend - for the management of forums chats, comments or notifications. It consists of user, thread and publication.

This application is based on a model published by AMAZON WEB SERVICES – AWS, in its public repository of GitHub with the name Node.js Micro services Deployed on Elastic Compute Cloud - EC2 Container Service.

The evolution of an App based on a Monolithic Architecture is represented, and the result is the same App using a Micro services Architecture. Additionally, the adaptation of virtual environments where the services will be raised and the tests will be executed is increased.

The first version of the application uses the monolithic architecture, the structure of the directory (project) and the arrangement of the files with the code, it is understood that all functionalities are grouped into a single resulting component. (Fig. 2 shows the directory of the monolithic version)

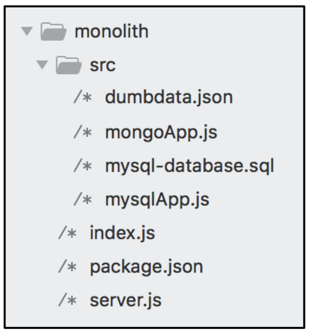
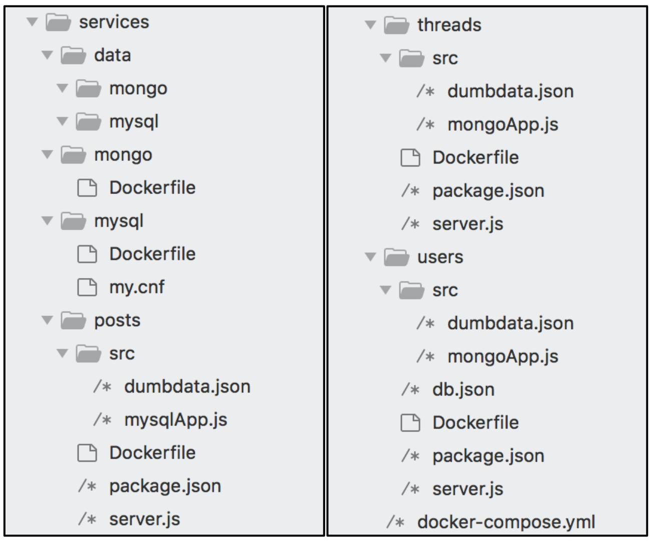

```
▼ monolith
  ▼ src
      /* dumbdata.json
      /* mongoApp.js
      /* mysql-database.sql
      /* mysqlApp.js
    /* index.js
    /* package.json
    /* server.js
```

**Figure. 2.** Structure of the Monolithic Application directory.

The second version is based on Micro services architecture. Both the structure of the directory and the distribution of the code in the files becomes different. Even giving place to its provision for its subsequent coupling with Docker containers (Fig. 3 shows the directory of the Micro services version).



```
services
  data
    mongo
    mysql
  mongo
    Dockerfile
  mysql
    Dockerfile
    my.cnf
  posts
    src
      dumbdata.json
      mysqlApp.js
    Dockerfile
    package.json
    server.js
  threads
    src
      dumbdata.json
      mongoApp.js
    Dockerfile
    package.json
    server.js
  users
    src
      dumbdata.json
      mongoApp.js
    db.json
    Dockerfile
    package.json
    server.js
  docker-compose.yml
```

**Figure. 3.** Structure of the application directory with Micro services.

### 3.2 Runtime Environment

Each version of the proposed application is executed in an environment that coincides with its trend. This is the case of the monolithic application that runs on a KVM and the application with the Micro services architecture opened on containers managed with Docker.

For both cases, hardware resources are of the same characteristics Intel Core i7 2.7 GHz CPU, 16 GB memory, HD disk and Virtual Network.

As per the Software, the base Operating System used for both, the host and the KVM, is Ubuntu 16.04, Long Term Support - LTS desktop and server respectively, QEMU 2.10, VMM 1.3.2, Docker 18.03.0-ce, Docker Compose 1.14, Java 1.8.0_162, NodeJS 6.14.1, MySQL 5.7.21, MongoDB 3.4.10, Apache JMeter 3.3, Server Agent 2.2 and NewRelic 2018.

With JMeter, scripts are generated with stress tests, allowing it to specify the conditions to which the application will be submitted in each environment. (Fig. 4 representation of the monolithic application on a KVM and Fig. 5 representation of the application with Micro services with containers).

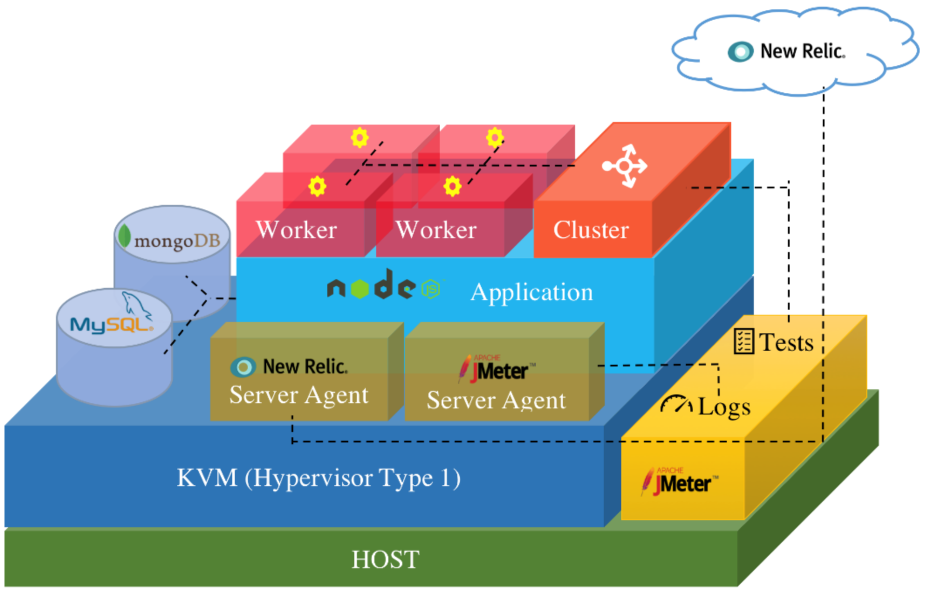


**Figure. 4.** First Scenario: Monolithic application on a KVM.

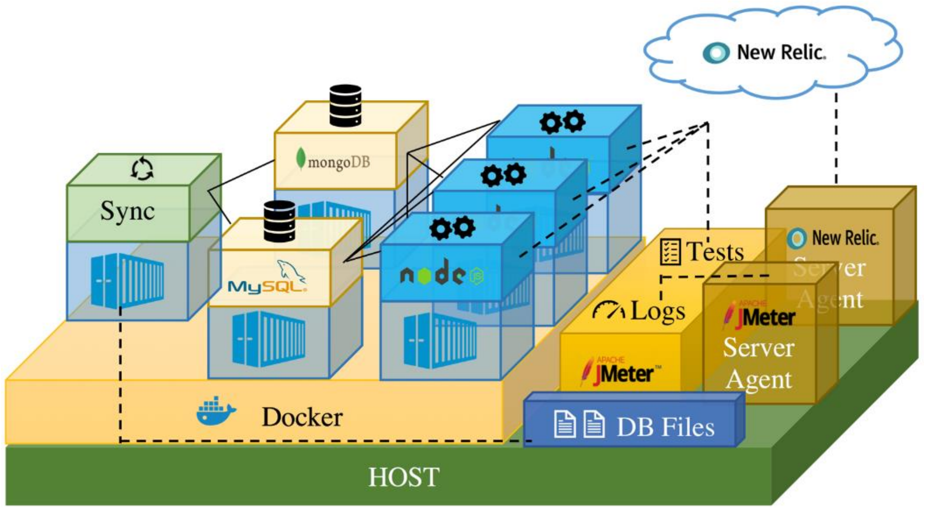


**Figure. 5.** Second Scenario: Application with Micro services on containers.

## 4 Assessments and Comparison

Stress tests are executed for each version of the application in its proper environment.

A series of tests are executed in a Script file generated with JMeter. It is configured to send asynchronous requests via HTTP to the application's Endpoints. Therefore, it simulates a number of connected users and defines a number of repetitions and threads. For the Monolithic Application, the KVM starts and then does the application. For the Application with Microservices, the containers are started using Docker that includes the start of the application. For stress tests, on one hand, there is the generation of data, and on the other hand the consumption (selection) of the same. The number of requests is limited to the processing capacity of the monolithic application because an increase in the number of requests generates errors of memory overflow or service drop (Application).

The tools (Software) JMeter, NewRelic and ServerAgent are used, which allow the collection of the "Logs", in files that are then processed and analyzed for their proper interpretation.

Two case of study are generated. The first with a reduced number of requests (273) to the application and the second case with a larger number of requests (1053). This number is a limit since increasing this amount generates memory errors and slows the response from the application.

## 4.1 CPU Consumption

Although, the performance of the application shows an advantage with the second scenario, it also evidences a greater consumption of the CPU resource. Indeed on average it consumes 67.73% versus 55.35% of the first scenario. Giving a difference of 12.38% in favor of the first scenario that generates less consumption of this resource.

## 4.2 Memory Consumption

Regarding this resource, it can be seen that on average there is a higher consumption. That is, 31.63% on the second scenario compared to 22.76% of the first scenario, with a difference of 8.87% in favor of this last scenario.

## 4.3 Network Bandwidth

In the case of the second scenario, the speed (kilobytes per second) is reduced compared to the first scenario. Because Docker generates a private network for the execution of its containers, the same one that serves as means of communication between them, thus reducing bandwidth capacity.

On average, the speed of the bandwidth in terms of receiving a KVM exceeds the containers by 85%. While the results improve in the speed of sending packages having a difference of 12.06%. For the purposes of the experiment, the average between these values is obtained, resulting in a 48.5% difference, where the first scenario exceeds the second scenario in terms of performance.

### 4.4 Disk write and read speed

In both cases, the bandwidth is greater in the second scenario, which implies greater efficiency. For this case there is a difference of 78.5% in disk reading and 79.2% in disk writing respectively for each environment. Overall and for research purposes, it gives a final result of 78.85% in favor of the second scenario.

To get a better idea in another unit of measure, on average the first scenario makes disk write at a speed of 788.11 kilobytes per second and in the second stage the amount of data is 3.76 Megabytes per second.

### 4.5 Application Performance

In both cases and scenarios the requests end successfully, however, in the second scenario case 1 generates 2 server response errors.

**Table 1.** Timesheets duration of the test from the first to the last request.

| | Number of Requests | Monolith Time | Microservices Time | Diff Time | Diff % |
|---|---|---|---|---|---|
| **Case 1** | 273 | 00:01:50 | 00:01:29 | 00:00:21 | 19.09% |
| **Case 2** | 1053 | 00:14:17 | 00:12:08 | 00:02:09 | 15.05% |

**Table 2.** Table of number of requests processed per second.

| | Number of Requests | Monolith | Microservices | Diff | % |
|---|---|---|---|---|---|
| **Case 1** | 273 | 2.5/s | 3.1/s | 0.6/s | 24% |
| **Case 2** | 1053 | 1.2/s | 1.4/s | 0.2/s | 16.67% |

On average, there is a reduction of 17.7% in total runtime using the scenario 2 with respect to containers Micro services and equivalent monolithic.

## 5 Conclusion and Future Work

The Monolithic Architecture, in conjunction with the KVM, show a favorable performance of CPU, RAM and Network. However, we must consider two new scenarios that a DevOps must take into account in the planning of a project. As per the Microservices, they prove to be efficient in the performance of resources, the Application and Disk. The two architecture are reasonably equal when the time is a relevant factor.

A shorter response time, means a higher consumption of resources, like in the specific case of Micro services with containers where usually large-scale computer systems, high flow and high performance information are driving. This scenario is convenient when the objective of the application is to process the greatest number of requirements in the shortest time possible, regardless of the limitation of resources and where scalability is a priority.

In longer response time there is a lower consumption of resources. This is a case specific to a monolithic application with KVM, where the computer system does not require a high level of resources and traffic is moderate or low.

Given the current trend, most computer systems look for ways to position themselves on top with the most modern technologies. The response time is almost imperceptible when accessing a technological service. The performance not only focuses on consuming a lower percentage of a resource, but also on optimizing so that consumption is adequate in shorter times.

Micro services and containers have been the subject of various investigations given their potential. This gives rise to new challenges, one of which is motivated to perform the same experiment proposed in this research but this time with services in the Cloud

## References

1. Nielsen, C.D.: Investigate availability and maintainability within a microservice architecture, http://cs.au.dk/fileadmin/site_files/cs/AA_pdf/ClausDNielsen_rapport.pdf, (2015).
2. Sun, L., Li, Y., Memon, R.A.: An open IoT framework based on microservices architecture. China Communications. 14, 154–162 (2017).
3. Kratzke, N.: About Microservices, Containers and their Underestimated Impact on Network Performance. In: ResearchGate. , Lubeck, Germany (2015).
4. Fowler, M., Lewis, J.: Microservices, https://martinfowler.com/articles/microservices.html.
5. Fussell, M.: Why a microservices approach to building applications?, https://docs.microsoft.com/es-es/azure/service-fabric/service-fabric-overview-microservices.
6. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, Inc. (2015).
7. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on. pp. 171–172. IEEE (2015).
8. Bartholomew, D.: Qemu: a multihost, multitarget emulator. Linux Journal. 2006, 3 (2006).
9. Prashant, D.: A Survey of Performance Comparison between Virtual Machines and Containers. 4, (2016).
10. Scott, J.: A practical Guide to Microservices and Containers: Mastering the Cloud, Data and Digital Transformation. (2017).
11. Vaughan-Nichols, S.J.: New Approach to Virtualization Is a Lightweight. Computer. 39, 12–14 (2006).
12. Khazaei, H., Barna, C., Beigi-Mohammadi, N., Litoiu, M.: Efficiency Analysis of Provisioning Microservices. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). pp. 261–268 (2016).

13 Stubbs, J., Moreira, W., Dooley, R.: Distributed Systems of Microservices Using Docker and Serfnode. In: 2015 7th International Workshop on Science Gateways. pp. 34–39 (2015).

14 Pahl, C.: Containerization and the PaaS Cloud. IEEE Cloud Computing. 2, 24–31 (2015).

15 Gerlach, W., Tang, W., Keegan, K., Harrison, T., Wilke, A., Bischof, J., D'Souza, M., Devoid, S., Murphy-Olson, D., Desai, N., others: Skyport: container-based execution environment management for multi-cloud scientific workflows. In: Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds. pp. 25–32. IEEE Press (2014).