ISSN 2347 - 3983
Volume 10. No.2, February 2022
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter071022022.pdf
https://doi.org/10.30534/ijeter/2022/071022022

Implementation of Monoliths within Enterprise Architectures using Disruptive Technologies in Docker Containers

José W. Cifuentes S.[1], Julián R. Camargo L.[2], César A. Perdomo Ch.[3]
[1] Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
[2] Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia
[3] Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Received Date : December 31, 2021 Accepted Date : January 27, 2022 Published Date : February 07, 2022

ABSTRACT

This document shows a solution to deploy and centralize inside an enterprise architecture a monolith (a specific development that doesn't use any integration with the elements of the enterprise architecture ecosystem such as LDAP or ESB or legacy systems inside a company). Usually, the monoliths have been rejected because they don't conserve the cross applications' structure, but what if the company needs this software for missional goals? This paper will show how to implement and manage these elements using a tool designed in new technologies and open source languages.

Key words: Deploy, Enterprise Architecture, Manage Software, Monolith, Open Source

1. INTRODUCTION

Within the Enterprise Software Architecture, the development of systems outside the proposed guidelines or having a structure that is not easily embeddable with the ecosystem of applications defined by the entities' architecture area is avoided as much as possible. In response to this position, which are tailor-made developments that generally have specific functionalities but are not easily integrated within an ecosystem of applications.

Thus, the industry has defined application integration elements within more robust solutions called portals, but these portals have different limitations; one of them is that all deployed applications must share the same dependencies and in some cases, the same language. In this regard, the challenge arises when integrating custom applications that do not contain the dependencies or the capacities to be absorbed by a portal [1].

These applications usually belong to systems that preserve the main characteristics of the high cohesion and low coupling software architecture, where cohesion is understood as "The measure of how strongly the elements are functionally related within a module … and the elements within a module can be instructions, groups of instructions, data definition, callings from another module, etc., and the goal is always for functions that are strongly related.

Therefore, the expectation is that everything within the module is connected, where the focus is on the task and these strong relationships reduce modules and minimize coupling" [2].

On the other hand, coupling within a software development environment is defined as follows:

> "Coupling is the degree to which the modules of a program depend on each other.
> If it is necessary to make changes in a different module to adjust a module of the program, there is a coupling between both modules.
> In Object-Oriented Programming, if class X uses a class Y, it is said that X depends on Y. That is, X cannot do its work without Y, therefore there is a coupling between classes X and Y.
> As it can be seen, the coupling is directional, there may be a coupling of class X with class Y, but this does not imply that it happens in the opposite direction" [3].

2. MONOLITHIC ARCHITECTURE IN ENTERPRISE ARCHITECTURE

2.1 Monolithic architecture

Monolithic architecture is a style of software architecture. "In summary, it can be said that in monolithic architecture, the software is structured so that all its functional aspects are coupled and subject to the same program. From this type of construction, characteristics such as the maximum degree of coupling of all the necessary information for the performance of any program or the creation of highly layered work environments that do not present too high a flexibility are derived.

All in all, the required information to work with this type of system is stably hosted on a single server. Due to this, there is no division between modules of any kind, so the different strata of a program depend entirely on the rest of the set" [4].

Therefore, since monoliths are a series of applications that do not depend on another system and are self-sufficient, they are considered stand-alone applications, which means isolated environments that function without external integrations, which is not necessarily bad within a series of missionary functionalities of a given entity.
In other words, they fulfill unique and isolated functions that are part of a great "whole". that is, the missionary duty of an entity or company is an adequate description of synergy, where there are independent components but that advance towards the same end without necessarily being dependent on each other.

The use of this style of architecture has been challenging due to the multiple disadvantages that this approach presents:

- It isn't easy to work in parallel in the same code base.
- No matter how small it is, any change requires the implementation of a new version of the entire application.
- Refactoring potentially affects the whole application.
- The only solution to scale is usually to create multiple copies of the monolith, which are resource-intensive.
- Integration can be complex when systems are expanded or acquired by other methods.
- It may not be easy to test due to the need to configure the entire monolith.
- Code reuse is challenging, and other applications often have their code copies.

2.2 Enterprise Software Architecture

Nowadays, businesses are increasingly complex and require flexible business processes, and information technology (IT) systems must effectively support these processes. To capture the complete view of all dimensions and complexity of the business system, the concept of enterprise architecture emerged, it identifies the main components of the organization and the relationships between them to achieve business objectives. Likewise, it integrates between the business plan, commercial operations, and technical aspects.

New technologies that change in response to the needs of the company. Enterprise architecture implementation is based on establishing a set of architectural guidelines that can ensure the coordinated development between the model and the company's requirements and between business processes and information technology.

These IT strategy guidelines should start from the mission of the company and the understanding of the business strategies and activities that support it, followed by obtaining the necessary information related to the organization's operations, the necessary technology to support the organization, and the implementation process.

The success in the missionary work of a company or entity and the costs related to the fulfillment of said purposes are increasingly dependent on the information systems implemented. These systems require a framework in which they can be managed. Within the context of business architecture, a reference environment or framework refers to specialized components that act based on the structuring and assembly of more significant or more complex components or buildings; that is, a business architecture framework determines in what terms the architecture is defined and documented.

Table 1 below lists the most widespread enterprise architecture frameworks used in the industry.

Table 1: Enterprise architecture frameworks [5]

Framework	Description/Information
Zachman	Zachman Framework for Enterprise Architecture (http://www.zifa.com)
E2AF	Extended Enterprise Architecture Framework (http://www.enterprice-architecture.info)
TOGAF	The Open Group Architecture Framework (http://www.opengroup.org/togaf)
GEAF	Gartner Enterprise Architecture Framework (http://www.gartner.com)
FEAF	Federal Enterprise Architecture Framework US (http://www.cio.gov)
BTEP	GC Enterprise Architecture and Standards. CANADA (http://www.tbs-sct.ca/inf-inf/index_e.asp)

2.3 What is enterprise architecture?

"Enterprise architecture in an organization corresponds to the way of representing the company integrally, allowing to cover and consider each one of the elements that are part of it.

This leads to a clear vision of the objectives, goals and lines of business in the company being established, starting from the strategic perspective (mission, vision, guidelines and strategic indicators), until reaching a current and future structure for the organization processes; which incorporates some of the components that are considered critical for its operation:

- The processes: business models and processes.
- The organizational structure: people and administrative structures.
- Information technologies: applications, information, technological infrastructure, and computer security.

As a result, the necessary tools and mechanisms will be available for the proper operation and functioning of the company, and therefore, support the fulfillment of its strategic objectives" [6].

3. GUIDELINES FOR MONOLITH INTEGRATION

Given the contextualization above, it can be concluded that within enterprise architecture, monoliths are not correctly a good practice for the reasons previously stated. That said, it is necessary to generate some guidelines to integrate monoliths within enterprise architecture; in other words, a way must be found to comply with the fundamental guidelines of enterprise architecture (systems that can be managed under a discipline or defined framework).

The implementation was carried out within the LASER research group (Automation, Embedded Systems, Robotics and Intelligent Systems Laboratory) of the Universidad Distrital Francisco José de Caldas. This is a group in which, given its investigative orientation, many technological developments and implementations are generated (usually independent from other components) to make them available to users via the Internet.

In this context, there is a need for creating a tool that complies with enterprise architecture standards and at the same time allows the coexistence of monoliths within a corporate environment.

For presenting these developments or monoliths, the following guidelines have been developed:

- The applications must be data processing.
- The applications must be delivered in docker-type containers.
- The applications must not generate dependency on transactional databases.
- The calls to the applications are request-response types. Fire and Go applications must be carefully validated so that their tasks are part of a superior system but not of other monoliths.
- The applications should not be placed on database containers; this is not a good practice since data changes over time and a container is designed to maintain a static image of development.

As for the languages that can be used to generate the different developments, a solution is conceived to generate no type of barrier. On the other hand, working with rest API (Application Programming Interface) is proposed to evolve the project, although the industry has business solutions such as API manager. For this case, as the implementation environment in which the developments are to be centralized is merely educational, it is expected that these integrations can be executed in the second phase of evolution.

3.1 Docker as a deployment component

Docker is an Open Source project defined as a container creation technology that allows the creation and use of Linux containers; it is promoted by the company Docker Inc. which "develops the work of the Docker community, making it more secure and sharing these advances with the rest of the community. It also supports enhanced and hardened technologies for business customers" [7].

Due to the arrival of disruptive technologies that within business environments presuppose an improvement of existing resources, large software providers and architectural parts manufacturers have designed a whole line of products under the Docker EE standard, in such a way that it is already possible to find in the Docker hub cloud images of versions of enterprise servers such as WebLogic, WebSphere and JBoos, which are commonly for business use.

Deployment on containers has two benefits: on the one hand, reproducibility since it is possible to generate instances of the same application and, on the other hand, scalability, since a container can be thought of as a software package that contains its libraries, services and the application itself [8]-[11].

Having the applications and all their dependencies packaged in the same container allows to execute and deploy in any system that uses the container engine or container specifications, and thus reproducibility is achieved.

On the other hand, scalability is achieved to the extent that it can be deployed in multiple systems when network traffic is high or provide redundancy when the system requires updating. All you have to do is configure it correctly to run on various systems and adjust the load balancer to redirect traffic. Tools like Docker swarm make it easy to scale containers across the host [12].

Therefore, a container can be defined as "a standard unit of software that packages the code and all its dependencies so that the application runs quickly and reliably from one computing environment to another" [13].

An image of a container is a lightweight, independent and executable package that contains enough dependencies for executing an application: binaries (obfuscated or compiled code), execution environment, system tools, additional libraries and required configurations.

Images are stored in a Docker registry as registry.hub.docker.com. Because images can be pretty significant, they are designed to be composed of layers from other images, which allows an infinite amount of data to be sent when images are transferred over the network" [14].

On the other hand, container images become containers at runtime; in the case of Docker, images become containers when they are run on the Docker Engine. This engine is available for applications based on Linux and Windows and runs in the same way, independent of its environment or infrastructure [16], [17]. An essential characteristic of containers is that they can isolate their environment to guarantee homogeneous execution regardless of the differences, such as development and organization [18].

Figure 1 shows a Docker container containing a basic web runtime environment, based on stacked images of an operating system and a web application server and defined by a name and a label with the specific version of the image.

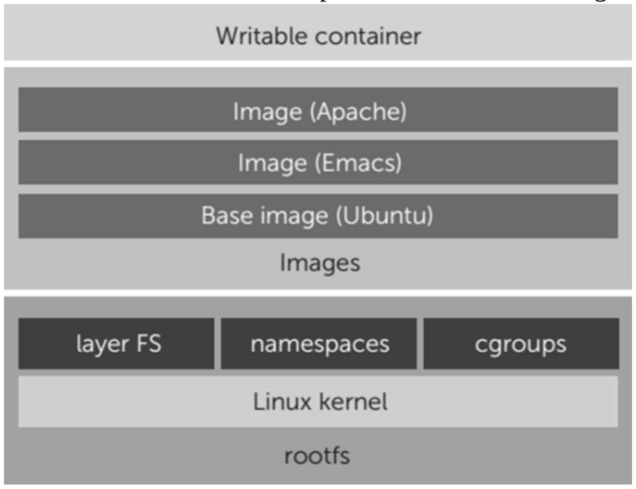

Figure 1: Architecture of a Docker image [15]

In this way, an image is defined as follows: "An image is an inert, immutable file, which is essentially a snapshot of a container. Images are created with the "comstackción" command and will produce a container when the execution starts.

3.2 Docker architecture

Within the Docker architecture (Figure 2), the following definitions can be found:

DOCKER HOST: The virtual or physical machine in which both Docker services and containers are deployed is called the "Docker Host". The Docker service is in charge of creating, running, and monitoring containers and storing the images. The execution of the Docker service is a task commonly assigned to the operating system of the machine or host system.

DOCKER CLIENT: The Docker client communicates with the service using a RESTfull API to write user commands to control the "Host", create images, publish, control, execute and in general manage containers, which are instances of the images inside the machine. The client communicates using bi-directionally HTTP services to allow and facilitate remote connections to the Docker service. Both the "Docker Client" and the "Docker Host" make up what is known as the "Docker engine" [15].

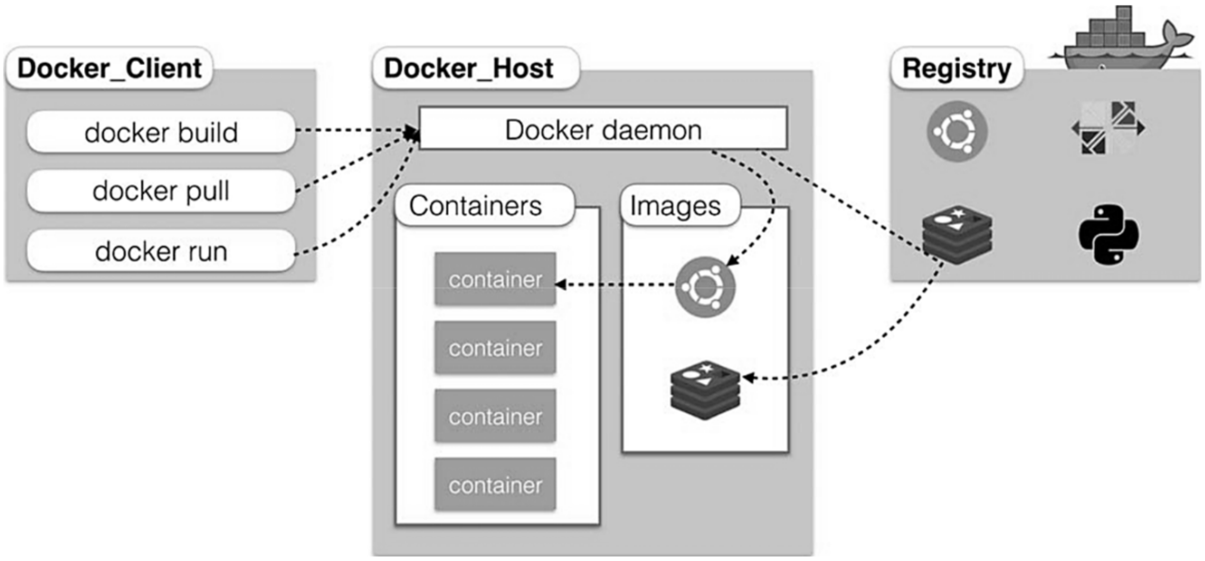

Figure 2: Docker architecture [15]

Features of Docker containers running on Docker Engine:

- Standard: Docker defined and introduced the standard type for containers to the industry, which means they can be portable anywhere.
- Lightweight: Containers share the kernel of the machine's operating system and therefore do not require an operating system per application; this increases server efficiency and reduces both physical

infrastructure and licensing costs.

- Security: Applications are more secure when containerized, as Docker provides the most robust isolation capabilities in the industry [19].

3.3 Advantages of using Docker containers

Since the project is aimed at monoliths and in the industry, until the arrival of containers, a solution has not been provided to the unknown of how to integrate monoliths within enterprise architecture, the use of Docker containers offers the following advantages:

- Modularity: In a contextualized approach by technologies, Full Stack Docker focuses on taking small parts of an extensive application to update or repair it without taking the entire application. In the proposed approach, this modularity is helpful because the monoliths can be repaired and/or updated without impacting an entire application system.
- Image and layer version control: Each Docker image file is made up of a series of layers. These layers are combined into a single image. A layer is created when the image changes. Every time a user specifies a command, such as run or copy, a new layer is created. Docker reuses these layers to build new containers, which makes the build process much faster. Intermediate changes are shared between images, thus improving speed, size and efficiency. Version control is inherent in creating layers. Every time a new change occurs, you have a built-in changelog - complete control of your container images.
- Restoration: With Docker, you can restore an image or any of its layers, this is possible because as mentioned above each change to an image generates a version of the container, and if at any time an unwanted event occurs on the container, it can be restored to its desired version.
- Rapid implementation: Because the proposal is based on monolithic architecture, the execution is done under the free selection of both the implementation technologies and the languages. Taking into account the nature of the monoliths that will be deployed, the deployment technology must facilitate publication tasks and be transparent to the selected language, to reduce implementation times and coupling between tools, which at this point would be non-existent.

4. PYTHON AND ITS API DOCKER

One of the great challenges that the planning and execution of an administrative tool for containers lay on the table is precisely accessing, managing and manipulating Docker objects such as containers, images, clusters, swarms, etc.

The Dicker Inc. development group has published a library for Python language which allows executing different actions on these objects [20].

To use the API library for Docker it is necessary to install the docker-py package, this action can be performed using the pip command or for current versions of Python using the pip3 command.

```
pip3 install docker−py
```

to execute docker commands within a Python script, it is necessary to connect to the docker "daemon" (docker Daemon), this can be done with the following commands:

```
#import client
from docker import client
#create a client object to connect to the daemon
myClient =
client.Client(base_url='unix://var/run/docker.sock')
```

The previous command will give us a list of the existing containers in the entire local machine in which the script is executed, with this data it is possible to obtain information on the ID, associated image and image ID, labels, ports, status, etc.

The creation of a new container can be executed with the following command:

```
myContainer=myClient.create_container(image='ubuntu:la
test',command='/bin/bash')
```

Using the previous example, you can create a container for the ubuntu image and provide a command to open bash or any other command you want.

Additionally, to validate the created container, the following command can be executed

```
print(myContainer['Id'])
```

In this sense, if you want to create a volume, you must specify its name, as well as the unit and its options if desired.

```
volume=myClient.create_volume(name='myVolume1',
driver='local', driver_opts={})
```

to validate if the volume was created correctly the following option can be executed:

```
print(volume)
```

to perform inspections on the volume, from the client object you can run

```
myClient.inspect_volume('myVolume1')
```

The following example shows how you can create a container with a volume.

```
mounted_container = myClient.create_container(
'ubuntu', 'ls',
volumes=['/var/lib/docker/volumes/myVolume1'],
host_config=myClient.create_host_config(binds=[
'/var/lib/docker/volumes/myVolume1:/usr/src/app/myVolum
e1'
, ])
)
```

The previous command creates a container based on the Ubuntu image and identifying the entry point as "ls", and mounting a volume located in the path of the local machine "/var/lib/docker/volumes/myVolume1" for "/usr/src/app/myVolume1" in the docker container [20]

5. PROPOSED ARCHITECTURE

5.1 Frontend Tool

After validating the available tools, an application that has a Frontend structure based on TypeScript language (javascript) is implemented using the react framework, which was developed by Facebook. The most important characteristics of reacting are listed below:

A. Component composition

As in functional programming, functions can be sent as parameters to solve complex problems, ReacJS applies this pattern by composing components.
"The applications are made with the composition of several components. These components encapsulate a behavior, a view, and a state. They can be very complex, but it is something that we do not need to worry about when we are developing the application because the behavior remains within the component and we do not need to complicate it once it has been done " [21].

B. Declarative Vs Imperative Development

Traditional forms of frontend web programming often use libraries like jQuery or "Vanilla JavaScript" with an imperative programming style, that is, scripts that report step-by-step on the changes that need to be executed in the DOM are made. On the other hand, the React style is declarative, that is, in this style, there is an application state and the components react to its change. "Components have a given functionality and when one of their properties changes they produce a change" [21].

C. One-way data flow

This feature, despite not being exclusive to React, makes it a very versatile tool when programming, since this model focuses on communication between higher-order and lower-order components, propagating data to them. Subsequently, the lower-order components work with such data and as a result, they disseminate events to the higher-order components or states.

D. Performance through Virtual DOM

React architecture operates on a virtual DOM that it owns. This does not mean that it does not use the browser's own DOM, what it means is that by using a virtual component to execute operations before reaching the real DOM, it makes its performance optimal, so that the virtual DOM takes care of memory and thanks to the differentiation tools between virtual and real, the browser updates more quickly, achieving updates of up to 60 frames per second, which as a result produces more fluid applications with smoother movements.

E. Isomorphism

This quality is the framework's ability to execute the code of both the client and the server. In this case, only the client-side will be used without forgetting that the applications can be extended.

F. Elements and JSX

React does not properly return HTML code, its code is a form of JavaScript with its expressions implemented by the manufacturer which are called JSX and produce elements in memory but not in the DOM in a traditional way, allowing heavy browser objects to be produced in less time.

G. Stateful and stateless components

React allows you to create stateless and stateful objects, the difference is described below:

- Stateless: these are the components that have no state, that is, they do not store data in their memory. That doesn't mean they can't receive property values, but those properties will always bring them into views without producing a state inside the component.
- Stateful: They are slightly more complex components because they are capable of saving a state and maintaining the usual business logic. Their main difference is that they are written in the code in a more complex way, generally using an ES6 class (JavaScript with ECMAScript 2015), which means that you can have attributes and methods to perform all kinds of operations.

H. Component life cycle

React implements a life cycle in the form of methods that are executed when common things happen to the component, which allows subscribing to actions when an initialization occurs, a promising return is received, etc.

5.2 Backend layer

For the Backend application, it was thought of using an object-oriented language that would allow easy and structured use of the information that comes from the Front layer. For this purpose, it was considered to use Java exposing microservices that controlled access to databases, but due to the nature of the Docker control and management API, it was decided to use Python as the Backend language.
Not only should this layer carry out the management of the containers, but it also has the responsibility of managing the access to the information in the database and exposing as microservices each of the methods of the tables that are part of

the database layer.

5.3 Database layer

In this case, a MySQL database was used because it is robust and open-source, so for the educational purpose of the project, it does not generate additional costs. This database stores the information of the applications (monoliths) and users that are part of the research group's developments [22].

Figure 3 shows the final architecture implemented for the solution to the proposed problem for educational purposes.

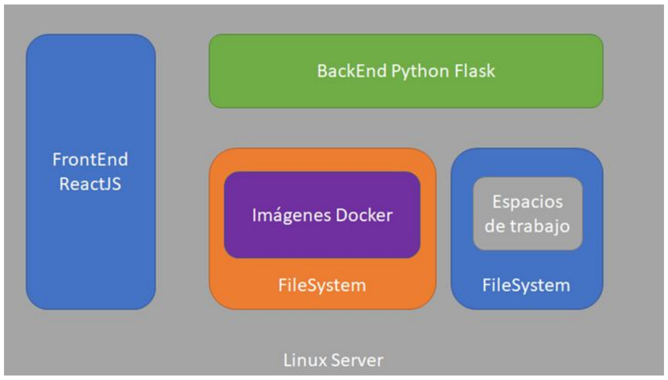

Figure 3: Solution architecture

6. CONCLUSIONS

Enterprise architecture can accommodate monoliths in its design without them becoming isolated applications and without constant monitoring, it is only enough to have correct use of tools that allow their orchestration and administration.

Containers represent a modern era of IT transformation, similar to the era of virtualization. However, unlike virtualization, containers exceed IT operations and require detailed input from multiple organizations. As a result, companies must change the way these organizations work together. Successful container adoption is as dependent on people and processes as it is on technology.

A good container performance strategy starts with its application specifications and user SLA guarantees, which should include processes such as container and code definition reviews, container-specific and stress testing, user testing and loading, in addition to the operating system, network and hardware metrics. Future performance expansion strategies should also include hardware support, continuous integration, and testing, as well as release automation and container runtime modeling to help predict performance issues before they are detected by customers. This must be supported by the correct tools to aid monitoring.

The use of containers in monoliths is a strategy that reduces administrative, operational and infrastructure costs because with the use of containers they can be made scalable so that resources are not underused and can be correctly assigned to different applications.

In terms of infrastructure, having containers facilitates administration, since they can be stored on a single machine (as proposed in this paper) and resources are shared optimally.
Containers ensure business success because the rate of return on the software investment and the investment itself is done accurately and quickly. Additionally, it detaches itself from the platform dependencies that usually bring higher costs to the day-to-day operation.

ACKNOWLEDGEMENT

The authors would like to thank the Universidad Distrital Francisco José de Caldas and the LASER research group that supported the development and testing of the project.

REFERENCES

1. E. Srbinovska and P. Mitrevski. **Web Services Deployment in Microsoft Azure Cloud Computing Platform**, *Conference: XLIX International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST 2014)*, 2014.
2. J. Likness. **Aplicaciones sin servidor: Arquitectura, patrones e implementación de Azure**. [Online]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/serverless/
3. J. R. Pascual. **Acoplamiento y Cohesión**, [Online]. Available: https://www.disrupciontecnologica.com/acoplamiento-y-cohesion/
4. Viewnext. **Arquitectura de microservicios vs arquitectura monolítica**, [Online]. Available: https://www.viewnext.com/arquitectura-de-microservicios-vs-arquitectura-monolitica/
5. E. M. Cordova and Y. J. Maldonado. **Propuesta de arquitectura empresarial para un centro de servicios compartidos dentro de un grupo empresarial privado**, Bachelor thesis. Universidad Peruana de Ciencias Aplicadas. [Online]. Available: https://repositorioacademico.upc.edu.pe/bitstream/handle/10757/624305/CORDOVA_ME.pdf?cv=1
6. Evaluando Software. **Arquitectura empresarial ¿qué es y para qué sirve?**, [Online]: Available: https://www.evaluandosoftware.com/arquitectura-empresarial/
7. RedHat. **Contenedores de Software: ¿Qué es Docker?**, [Online]: Available: https://www.redhat.com/es/topics/containers/what-is-docker
8. J. Cito, V. Ferme and H. Gall. **Using Docker Containers to Improve Reproducibility in Software and Web**

Engineering Research, *International Conference on Web Engineering*, 2016.
9. V. Nedu and A. R. Megalapete. **A survey on Docker and its significance in cloud**, 2016.
10. B. B. Rad, H. J. Bhatti and M. Ahmad. **An Introduction to Docker and Analysis of its Performance**, *International Journal of Computer Science and Network Security*, vol. 17, no.3, pp. 228-235, 2017.
11. C. D. Boettige**. An introduction to Docker for reproducible research, with examples from the R environment**, *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, 2014.
12. D. Bartoletti and C. Dai. **The Forrester New Wave™: Enterprise Container Platform Software Suites, Q4 2018**, [Online]: Available: https://www.redhat.com/rhdc/managed-files/cm-forrester-new-wave-enterprise-container-platform-software-suites-q42018-analyst-paper-f14768-201810-en.pdf
13. C. Fetzer. **Building Critical Applications Using Microservices**, *IEEE Security & Privacy*, vol. 14, no. 6, pp. 86-89, 2016.
14. Dorky Desarrollo. **¿Cuál es la diferencia entre una imagen Docker y un contenedor?**, [Online]: Available: https://www.dokry.com/31180
15. L. A. Iñiguez. **Arquitectura tecnológica para la entrega continua de software con despliegue en contenedores**, M.S. thesis, Universidad de Cuenca. [Online]: Available: http://dspace.ucuenca.edu.ec/handle/123456789/28529 2017
16. S. Singh and N. Singh. **Containers & Docker: Emerging roles & future of Cloud technology**, *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, 2016.
17. K. Kharbanda and K. Kaur. **Performance Study of Applications Using Dockers Container**, *Proceedings of the International Conference on Advances in Electronics, Electrical & Computational Intelligence (ICAEEC)*, 2019
18. Docker. **What is a Container?**, [Online]: Available: https://www.docker.com/resources/what-container
19. J. Stubbs, W. Moreira and R. Dooley. **Distributed Systems of Microservices Using Docker and Serfnode**, *2015 7th International Workshop on Science Gateways*, 2015.
20. R. Jain. **Python Library API for Docker**, [Online]: Available: https://www.tutorialspoint.com/python-library-api-for-docker
21. D. García. **Características de React**, [Online]: Available: https://desarrolloweb.com/articulos/caracteristicas-react.html
22. E. Bacis S. Mutti, S. Capelli and S. Paraboschi. **DockerPolicyModules: Mandatory Access Control for Docker Containers**, *Conference: IEEE Conference on Communications and Network Security (CNS)*, 2015.