UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniera en Ciencias Aplicadas

Carrera de Ingeniería en Sistemas Computacionales

Optimización de aplicaciones Java Enterprise monolíticas mediante el uso de contenedores Docker.

Trabajo de Grado previo a la obtención del título de Ingeniero en Sistemas Computacionales

Autor:

Vizcaíno Quiroz Yuliza Dayana

Director:

Msc. Rea Peñafiel Xavier Mauricio

Ibarra – Ecuador

2021

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004151435		
APELLIDOS Y NOMBRES:	Vizcaíno Quiroz Yuliza Dayana		
DIRECCIÓN:	Cotacachi		
EMAIL:	ydvizcainoq@utn.edu.ec		
TELÉFONO FIJO:	490-018	**TELÉFONO MÓVIL:**	0983974453

DATOS DE LA OBRA	
TÍTULO:	OPTIMIZACIÓN DE APLICACIONES JAVA ENTERPRISE MONOLÍTICAS MEDIANTE EL USO DE CONTENEDORES DOCKER.
AUTOR (ES):	Vizcaíno Quiroz Yuliza Dayana
FECHA: DD/MM/AAAA	10/03/2021
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniera en Sistemas Computacionales
ASESOR /DIRECTOR:	Msc. Xavier Mauricio Rea Peñafiel

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de esta y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 10 días del mes de marzo de 2021

EL AUTOR:

(Firma).....
Nombre: Yuliza Dayana Vizcaíno Quiroz

CERTIFICADO DEL DIRECTOR DE TRABAJO DE GRADO

UNIVERSIDAD TÉCNICA DEL NORTE
Facultad de Ingeniería en Ciencias Aplicadas

Certificación del Director de Trabajo de Grado

En mi calidad de Tutor del Trabajo de Grado presentado por la egresada, Vizcaíno Quiroz Yuliza Dayana para optar por el Título de Ingeniero en Sistemas Computacionales, cuyo tema es: OPTIMIZACIÓN DE APLICACIONES JAVA ENTERPRISE MONOLÍTICAS MEDIANTE EL USO DE CONTENEDORES DOCKER. Considero que el presente trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del tribunal examinador.

Es todo cuanto puedo certificar en honor a la verdad.

Atentamente,

firmado electrónicamente por:
XAVIER MAURICIO REA PEÑAFIEL

Msc. Mauricio Rea

Director de Tesis

Dedicatoria

“Ayer es historia, mañana es un misterio y hoy es un regalo de Dios, por eso se lo llama presente”.

Este trabajo de investigación va dedicado a mis padres Narcisa Quiroz y Miguel Vizcaíno, por haberme apoyado en el transcurso de mi carrera, por su sacrificio, por su lucha y amor.

A mis hermanos Paola y David que junto a mis padres son ese motor que me han dado la fuerza y valentía para culminar con esta etapa de mi vida, y quienes seguirán siendo ese motivo para cumplir muchos anhelos y sueños.

Yuli

Agradecimiento

Agradezco a mis padres por el sacrificio día a día y sus consejos; por ese apoyo incondicional, por el ejemplo y valores que me han inculcado, gracias a todo eso, hoy logró cumplir una meta más.

Pao mi negrita, gracias por la ayuda que me brindaste incondicionalmente en esta etapa de mi vida, gracias por ser esa hermana cariñosa y a la vez peleona y estresante, eres esa pieza fundamental que forma parte de mi ser.

Mil gracias, Javi por ser ese soporte y apoyo que nunca dejo que decayera, que siempre tuvo las palabras precisas y alentadoras para que en cada caída me levantara con mas fuerza y siguiera luchando por mis sueños.

Gisse gracias por ser mi amiga, mi gemela, por ser parte de este largo proceso, y por todos los momentos vividos, de ti me quedo con los mejores recuerdos.

Agradezco también a mi tutor Msc. Mauricio Rea, un docente de calidad y una gran ser humano. Gracias, infinitas Inge, por todo su apoyo, por sus conocimientos impartidos que sin duda alguna han sido la clave del éxito en este proceso de titulación.

A mis amigos y compañeros gracias por tantas risas, por todo este tiempo de amistad que hicieron que esta etapa sea divertida, de cada uno de ustedes me llevo lo mejor.

Sin Dios nada de esto sería posible, por eso mi agradecimiento entero es a él.

Yuli

Tabla De Contenido

Índice de Figuras

Índice de Cuadros

Resumen

Con la evolución de la tecnología, la informática a experimentado grandes cambios basados tanto en software como hardware, brindando al usuario mejor experiencia y usabilidad. Sin embargo, en varias ocasiones actualizar equipos representa un coste muy elevado volviéndolo inaccesible. A medida que se genera más requerimientos o necesidades incrementa el consumo de recursos tecnológicos, es ahí donde la informática entra a optimizar recursos a través del uso de nuevas tecnologías que permitan mayor eficiencia en tiempos y costos.

El presente trabajo permitió optimizar recursos en el despliegue de aplicaciones monolíticas creadas por la carrera de Ingeniería en Sistemas Computacionales, haciendo uso de contenedores Docker denominados máquinas virtuales ligeras que proporcionan portabilidad, ligereza y autosuficiencia a las aplicaciones. Se construyó un ambiente de virtualización con imágenes Docker de PostgreSQL y Wildfly que permitieron ejecutar una aplicación con el fin de demostrar su funcionalidad y viabilidad, para esto se realizó una validación de rendimiento con la ISO 25010 comparando la arquitectura tradicional con los contenedores. Los resultados se los obtuvo con una matriz de evaluación obteniendo calificaciones para cada arquitectura demostrando así que Docker es viable.

Abstract

With the evolution of technology, computing has undergone great changes based on both software and hardware, providing the user with a better experience and usability. However, on several occasions updating equipment represents a very high cost, making it inaccessible. As more requirements or needs are generated, the consumption of technological resources increases, that is where computing enters to optimize resources using new technologies that allow greater efficiency in time and costs.

The present work allowed optimizing resources in the deployment of monolithic applications created by the Computer Systems Engineering career, making use of Docker containers called light virtual machines that provide portability, lightness, and self-sufficiency to the applications. A virtualization environment was built with PostgreSQL and Wildfly Docker images that allowed an application to be executed to demonstrate its functionality and viability, for this a performance validation was carried out with ISO 25010, comparing the traditional architecture with containers. The results were obtained with an evaluation matrix obtaining qualifications for each architecture, thus demonstrating that Docker is viable.

INTRODUCCIÓN

Antecedentes

En los últimos años según (Maya & López, 2018), a nivel empresarial y tanto en el sector privado como público se ha hecho uso de arquitecturas monolíticas en la implantación de software, incidiendo en diferentes aspectos tecnológicos y administrativos, este modelo proporciona un esquema tradicional que tiene un front-end conectado a un back-end con los módulos o servicios que integran una aplicación o software, ejecutándose en una misma máquina virtual haciendo que la comunicación entre los módulos pase a memoria en cada llamada a métodos.

La arquitectura monolítica se aloja en servidores de nivel empresarial, que son puestos en producción una vez que cumplan los requerimientos de despliegue, a medida que se genera más requerimientos o necesidades incrementa el consumo de recursos tecnológicos que afecta en tiempo y recursos en el desarrollo e implementación de cualquier modificación (Chiza., 2018).

Se ha evidenciado mediante estudios tomados de dos Instituciones de Educación Superior (IES) del Ecuador, la Escuela Politécnica Nacional y la Universidad de Guayaquil, que toda la capacidad con la que cuentan los servidores llega a saturarse, por la gran cantidad de máquinas virtuales ejecutándose, haciendo que el servicio se torne lento y en ocasiones se genere fallos completos, generando un uso desmedido de los recursos informáticos en especial en la capacidad de almacenamiento (Arboleda Carlos, 2017).

Situación Actual

En la actualidad en Ecuador los modelos de negocio a nivel empresarial utilizan arquitecturas Java Enterprise, empleando mayor cantidad de recursos informáticos, con necesidades de actualización y mantenimiento para desarrollar nuevos procesos (Lopez, 2017).

Se evidencia que existe una gran utilización de estos recursos informáticos, lo que ocasiona bajo rendimiento en los servidores, agotamiento de almacenamiento, mayor tiempo de ejecución y baja capacidad de la máquina soportada.

En las IES del Ecuador, caso de estudio Universidad Técnica del Norte (UTN), como proceso de enseñanza aprendizaje en las asignaturas de aplicación, se desarrolla soluciones basadas en arquitectura Java Enterprise Edition (JEE) en casos han sido desplegadas como aplicaciones para la Carrera de Ingeniería en Sistemas Computacionales (CISIC) perteneciente a la Facultad de Ingeniería en Ciencias Aplicadas (FICA), ayudando a mejorar los procesos y contar con herramientas de apoyo (Vivas, 2019).

La CISIC dispone con aplicaciones desarrolladas e implementadas con la arquitectura monolítica de JEE en el Cloud FICA mediante Proxmox VE como gestor de virtualización de servidores de código abierto, donde la creación de muchas máquinas virtuales para alojar las aplicaciones y realizar las pertinentes pruebas de funcionamiento, hace que presente un uso desmedido de los recursos informáticos (Simbaña, 2016).

Planteamiento del Problema

La Facultad FICA UTN, usa muchos recursos informáticos de su servicio Cloud, al realizar copias de máquinas virtuales con todos los recursos ya instalados para el funcionamiento y despliegue de las aplicaciones, lo que ocasiona que la capacidad del rendimiento y almacenamiento del servidor genere inconsistencia y en ocasiones fallos irreversibles como: la pérdida de productividad que impide realizar un trabajo durante un tiempo, perdida de datos de los usuarios que estén usando el servicio en ese momento.

Al momento de llegan a sufrir fallos, ya sea por falta de una adecuada planta de energía eléctrica, o por una mala infraestructura, se produce indisponibilidad de los servicios y también pérdidas de información, impidiendo así el manejo inmediato de recuperación de los sistemas desplegados o puestos en producción.

En la Fig1. se establece la problemática del uso de la infraestructura tradicional.

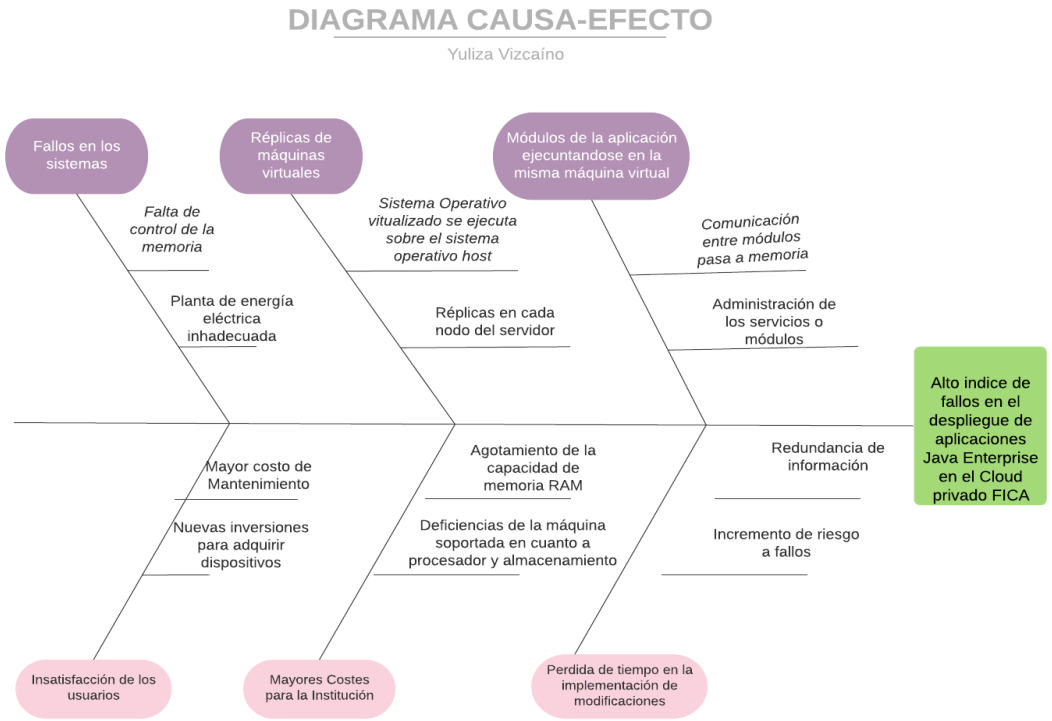


Fig. 1 Diagrama de Problemas

Elaboración - Propia

Objetivos

Objetivo General

Optimizar aplicaciones Java Enterprise (JEE) monolíticas mediante el uso de contenedores Docker

Objetivos Específicos

- Desarrollar y construir una base teórica de la propuesta de investigación con las características de Docker sus funcionalidades y limitaciones
- Construir un ambiente de virtualización basado en Docker desplegado en el Cloud privado FICA para el proceso de alojamiento de aplicaciones Java Enterprise (JEE).
- Validar los resultados de la investigación con la característica: Eficiencia de Desempeño de la norma ISO/IEC 25010

Alcance

Construir un ambiente de virtualización mediante contenedores Docker, para alojar aplicaciones Java Enterprise monolíticas desarrolladas en la Carrera de Ingeniería en Sistemas Computacionales (CISIC).

Para ello se ha divido el proyecto en tres partes:

1. Desarrollar una base teórica de los contenedores Docker

Investigación bibliográfica de los microservicios basados en Docker y arquitecturas monolíticas, sus aportes, beneficios y aplicaciones; que permita tener conocimientos base para el desarrollo del proyecto con la documentación relevante del tema.

2. Construir un ambiente de virtualización

Creación de imágenes Docker de PostgreSQL y Wildfly para el despliegue y ejecución de aplicaciones desarrolladas por la Carrera de Ingeniería en Sistemas Computacionales (CISIC), dentro de contenedores Docker.

3. Validación de resultados

Se validarán los resultados mediante la característica: Eficiencia de Desempeño de la norma ISO/IEC 25010, para obtener resultados reales, en cuanto a tiempos y uso de recursos, haciendo uso de la virtualización local y de Cloud Computing con contenedores Docker.

Arquitectura de Funcionamiento

El presente proyecto permitirá a la Facultad de Ingeniería en Ciencias Aplicadas (FICA) ahorrar costos de infraestructura en cuanto tiempo y recursos, haciendo más efectiva la ejecución y despliegue de las aplicaciones, debido a que Docker permite disminuir fallos en el sistema a través del mantenimiento y configuración rápida, generando satisfacción al usuario sin necesidad de adquirir tecnología constantemente.

Docker es un proyecto de código abierto que admite la creación de contenedores, definidos como máquinas virtuales ligeras en Linux, que alojan aplicaciones con su propio lenguaje. Son sistemas basados en microservicios y permite su gestión dinámicamente, de esta manera se puede obtener ventajas al empaquetar aplicaciones y sus dependencias para su operación y despliegue con menos recursos, a diferencia de la creación de máquinas virtuales.

Brinda la facilidad de crear repositorios en la nube con los contenedores creados y a su vez descargarlos desde otro host o plataforma Cloud que tenga instalado Docker para desplegarlo sin ningún problema. Además, cuenta con un repositorio público (hub) de imágenes disponibles del mismo, muchos de estos configurados por los mismos creadores de los sistemas operativos.

Los contenedores son una pieza fundamental del software donde contiene un sistema de archivos necesarios para la ejecución de la aplicación, es decir bibliotecas de sistemas, herramientas del sistema, código, el tiempo de ejecución y lo que se podría instalar en el servidor.

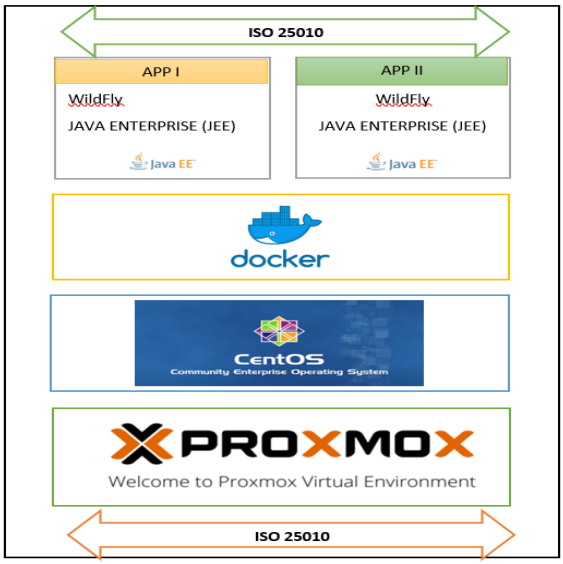


Fig. 2. Arquitectura de Funcionamiento

Fuente: Elaboración Propia

Justificación

El presente proyecto tiene un enfoque en el Objetivo de Desarrollo Sostenible (ODS) Nro. 9: “Industria, innovación e infraestructura”, planteado por la ONU y UNESCO (Naciones Unidas, 2016).

Según (Naciones Unidas, 2016) este objetivo busca: “Construir infraestructuras resilientes, promover la industrialización inclusiva, sostenible y fomentar la innovación.

Las metas del objetivo que se cumplen son:

9.4 De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas.

9.5 Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, 26 entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo.

9.b Apoyar el desarrollo de tecnologías, la investigación y la innovación nacionales en los países en desarrollo, incluso garantizando un entorno normativo propicio a la diversificación industrial y la adición de valor a los productos básicos, entre otras cosas.

9.c Aumentar significativamente el acceso a la tecnología de la información y las comunicaciones y esforzarse por proporcionar acceso universal y asequible a Internet en los países menos adelantados de aquí a 2020.

El tema de Cloud Computing actualmente está en auge y se sigue implementando cosas a futuro para mejorar este servicio, minimiza tiempos y costos de instalación, es un servicio que se puede usar desde cualquier parte donde se encuentre y permite la optimización de recursos a nivel general.

Ambiental

Esto implica evitar gastos en nueva infraestructura que representa costos muy elevados de adquisición y contamina al medio ambiente con los componentes que se encuentran elaborados.

Tecnológico

Es importante utilizar tecnologías actuales que permitan desarrollar un sistema óptimo, para hacer frente a la competencia existente en esta era tecnológica siendo el aprendizaje continuo necesario.

Metodológica

En el presente proyecto se desarrollará una investigación aplicada, tecnológica, descriptiva y de campo con la finalidad de contribuir a la construcción de un ambiente de virtualización robusto y eficiente en el despliegue de servicios para el desarrollo de aplicaciones empresariales.

CAPÍTULO 1

1 MARCO TEÓRICO

1.1 Arquitecturas Monolíticas.

Es un modelo ejecutable con una única base de código, donde el sistema operativo y los servicios fundamentales son encapsulados en un mismo repositorio. Para (Al-Debagy & Martinek, 2018) un arquitectura monolítica es una aplicación que contiene múltiples servicios, comunicándose con sistemas externos o consumidores a través de interfaces como servicios web, Páginas HTML, etc.

Según (Martínez, 2018) son estructuras rígidas que carecen de flexibilidad y a medida que la complejidad de la lógica de negocio aumenta, el número de funcionalidades que la aplicación debe proporcionar, comienza a ponerse de manifiesto algunos problemas serios.

Una arquitectura monolítica es autónoma, encapsula todos los servicios en un archivo .war, obedeciendo solo a dichos servicios que en este caso serian un front-end, back-end y base de datos que permiten su despliegue y ejecución, la comunicación pasa a memoria a cada llamada a métodos, esto puede presentar inconvenientes al realizar actualizaciones, se tendrá que subir los cambios a producción y si en ocasiones contiene algún error, se afectará a toda la aplicación y se obtendrá perdidas para la empresa.

Se dividen en capas según la necesidad del negocio como se muestra en la Fig.3, la más común es de 3 capas: Capa cliente, Capa de aplicación de negocio y Capa de repositorio de datos, por eso son llamadas monolíticas al poseer un único ejecutable lógico, ejecutándose en un mismo contexto (Martínez, 2018).

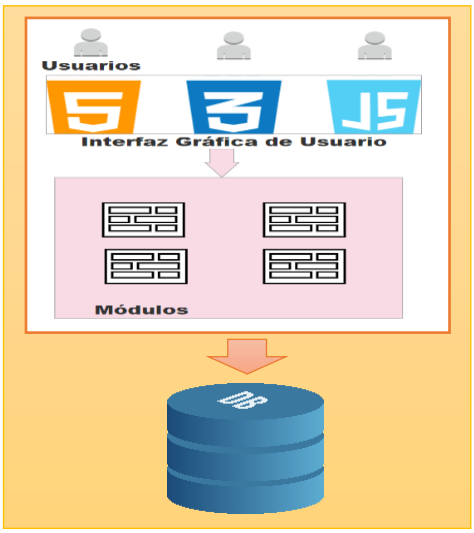


Fig. 3. Diseño Arquitectura Monolítica

Fuente: Elaboración Propia

1.1.1 Arquitectura Java Enterprise Edition

Java Enterprise Edition (Java EE) es una tecnología web, que proporciona herramientas necesarias para el desarrollo de aplicaciones web grandes y robustas; debido a que no obliga a usar el patrón arquitectónico MVC, permite que las solicitudes entrantes puedan ser procesadas por servicios web como RESTful JAX-RS, ejecutando algunas funcionalidades para producir resultados deseados (Schutt & Balci, 2016).

La arquitectura Java EE se basa en un modelo de aplicación distribuida multicapa para aplicaciones empresariales, se divide a la lógica de negocio en componentes según la función que realiza, los componentes son instalados en diferentes máquinas dependiendo de la capa del entorno de Java EE multicapa a la que pertenece el componente (Sierra, 2015).

En la Fig.4, se muestran las capas que conforma la arquitectura Java EE:

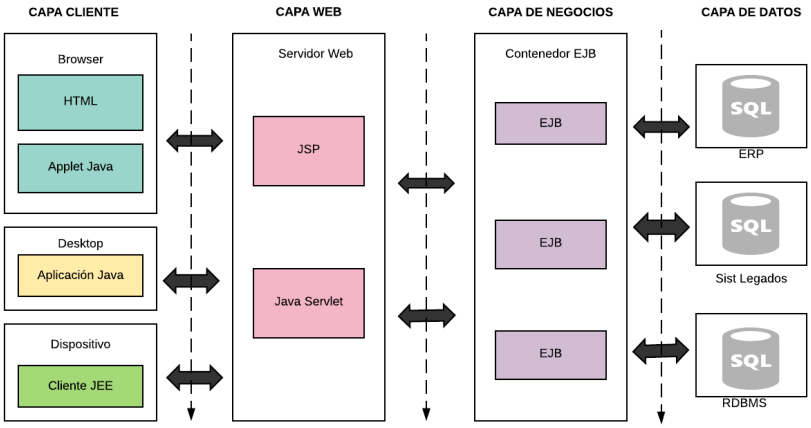


Fig. 4. Arquitectura Java EE

Fuente: Elaboración Propia

Una aplicación Java EE se conforma de componentes que se ejecutan en distintos contenedores lógicos, y es entregada en ficheros con extensión .war (Web ARchive), .ear (Entreprise ARchive) o .jar (Java ARchive), que contienen módulos Java EE con uno o más componentes y un documento XML con detalles para ejecución y localización.

1.1.2 Principales ventajas de las Arquitecturas Monolíticas

La estructura de este tipo de aplicaciones como ya se lo mencionó anteriormente se realiza a través de n capas que la vuelve más eficientes y rápidas. En la TABLA 1 se establecen las principales ventajas de usar una arquitectura monolítica.

TABLA 1 Ventajas de las Arquitecturas Monolíticas

VENTAJAS	
Facilidad de desarrollo	Gran cantidad de entornos de desarrollo
Simplicidad de pruebas	Pruebas sistemáticas en las distintas capas
Sencillez de despliegue	Único archivo que se copia en el servidor
Viabilidad del escalado	Fácilmente escalables a través de la duplicación de servidores.

Puntos de fallo	No dependen de otro factor para su funcionamiento, amenorando así errores de comunicación, red, etc.
Autónomas	Son totalmente autosuficientes
Performance	Son rápidas debido a su procesamiento local sin necesidad de consumir procesos distribuidos para cumplir con una tarea

1.1.3 Inconvenientes de las Arquitecturas Monolíticas

A pesar de que estas aplicaciones son hechas a medida, son arquitecturas rígidas que carecen de flexibilidad, mientras más crece la lógica del negocio, el número de funcionalidades que proporciona la aplicación y los componentes de soporte, comienzan a desplegarse serios problemas (Martínez, 2018).

Para (Arboleda Cola Carlos Augusto, 2017) una arquitectura monolítica no siempre es la mejor opción por los siguientes puntos:

a. Sistema demandado*:* requiere ser escalado afectando a todas las funcionalidades, generando uso desmedido de recursos.
b. Existe un solo repositorio de códigos*:* si no son tratados de manera adecuada, se vuelve un punto de fallo para el sistema
c. Cambio continuo de tecnología*:* el software se vuelve frágil y complicado de actualizar, ocasionando errores.
d. Implementar reutilización de código*:* limitaciones con otros equipos en cuanto acoplamiento.

1.2 Virtualización

Creación virtual de algún recurso tecnológico como hardware, software, dispositivo de almacenamiento o recursos de red. Para (Rubiano, 2015), es una abstracción de los recursos de una computadora, llamada hipervisor o Virtual Machine Monitor (VMM) que crea una capa de abstracción entre el hardware de la máquina física y el sistema operativo de la máquina virtual, dividiéndose el recurso en uno o más entornos de ejecución.

Es la simulación de hardware de una computadora para ejecutar un sistema operativo, juntamente con sus aplicaciones a través de un servidor con capacidades mayores a las de una máquina, permitiendo ser controlados a un nivel de velocidad y automatización que en máquinas físicas no es posible realizar (Llaven, 2015).

Para (Van De Belt, Ahmadi, & Doyle, 2017) los recursos virtuales son independientes asignándolos simultáneamente a múltiples usuarios, cada usuario adopta la ilusión de

propiedad de sus recursos, puesto que ellos no ven la diferencia entre recursos virtuales y recursos reales.

A través de la virtualización se puede compartir recursos de hardware como: CPU, RAM, ancho de banda, entre otros, este proceso se realiza por más de una máquina virtual, disminuyendo el consumo de energía al optimizar dichos recursos (Dörterler, Dörterler, & Ozdemir, 2017).

1.2.1 Máquina Virtual

Es un recurso tecnológico de software con las mismas funciones que una máquina real, tiene las mismas características de la máquina física que la soporta debido a que se alimenta de los recursos que esta le provee, permitiendo construir y ejecutar aplicaciones sin ningún problema (Dörterler et al., 2017).

Una máquina virtual contiene su propio hardware virtual que es asignado al hardware de la maquina real, ahorrando costos de adquisición y mantenimiento de equipos, además de reducir la demanda de energía y refrigeración (Azure, n.d.).

1.2.2 Evolución de la virtualización

En los años 60 IBM desarrolla sistemas de particiones lógicas, equivalentes a máquinas virtuales que corrían en mainframes de aquella época y solventaban la necesidad de tener varias máquinas independientes, después de un tiempo en los años 80, aparece la arquitectura x86 haciendo que empresas cambien su estructura de un mainframe único y muchos terminales a múltiples maquinas como servidores, perdiéndose de nuevo la necesidad de virtualizar. Es en los años 90 que empieza nuevamente a sentirse la necesidad de virtualización, resultaba rentable tener un sistema simplificado, con menor inversión de dinero y con facilidad de administración (Pérez & Pérez, 2014).

De los años 90 hasta la actualidad se hace uso de la virtualización a nivel empresarial. A esto se le suma también Cloud Computing que actualmente es una tecnología que está abarcando el mercado, debido a su rapidez de despliegue a través de una red.

El internet y las nuevas tecnologías evolucionan con un solo fin “Satisfacer los requisitos de los servicios actuales y futuros en desarrollo”. Por tal motivo la virtualización es pieza fundamental de la informática en la nube, permite implementar varios servidores virtuales en un servidor físico (Bhardwaj & Krishna, 2019).

1.2.3 Tipos de Virtualización

1.2.3.1 Virtualización a partir de un sistema operativo host

El sistema operativo se instala en una máquina física y a partir de este se crea la capa de virtualización como una funcionalidad del mismo sistema, a su vez se crean las Máquinas Virtuales (MV), y se encapsulan aplicaciones en altos niveles a través de sistemas operativos virtuales(Alemandi & Jara, 2014). Este tipo de virtualización presenta flexibilidad de ejecutar aplicaciones sin entornos de virtualización, al mismo tiempo que se ejecutan las MV como se muestra en la Fig.5. Sin embargo, al tener varias capas se produce una sobrecarga muy alta que afecta al rendimiento (Morán, Paul, Fernando, & Ángel, 2013).

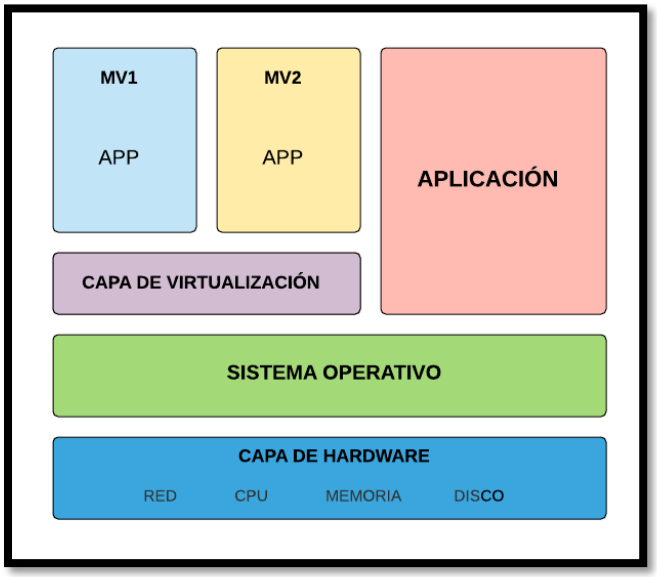


Fig. 5. Virtualización con Sistema Operativo Host
Fuente: Elaboración Propia

1.2.3.2 Virtualización a través de un hipervisor

En este tipo de virtualización no existe un sistema operativo, estableciendo un hipervisor como capa de virtualización adaptada al hardware que permita la comunicación entre las MV y la máquina física, ejecutar aplicaciones solo se permite dentro de las MV (Morán et al., 2013).

Ofrece mejor rendimiento a las MV, con eficiencia, robustez y escalabilidad. Debido a la estructura que tiene los recursos están totalmente disponibles paras las MV que se despliegan en él, el sistema es el encargado de repartir los recursos entre las distintas maquinas ejecutándose (Pérez & Pérez, 2014). Ver la Fig.6.

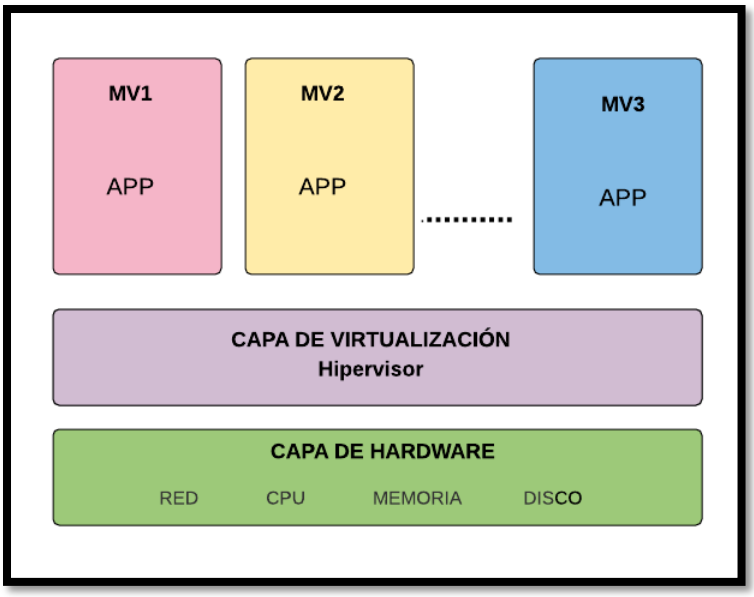


Fig. 6. Virtualización con Hipervisor

Fuente: Propia

1.2.4 Usos de la virtualización

Gracias a la virtualización, se logra la simulación de modelos útiles para industrias, trabajos de robótica, mercados competitivos, educación, entre otros, siendo una de las funciones más ricas que presenta la tecnología (Millán-Rojas, Gallego-Torres, & Chico-Vargas, 2016).

En las TABLA 2 y 3, se establecen los usos y beneficios que presenta la virtualización.

TABLA 2 Usos de la virtualización

Entornos de empleo de la virtualización	
Consolidación de servidores	Se reduce la cantidad de máquinas físicas, ejecutando varias MV en un solo servidor
Recuperación ante desastres	Permite realizar backup de la MV, copiando este fichero en otra maquina física
Alta disponibilidad	Si una MV se cae se puede levantar otra similar sin afectar al servicio
Desarrollo y testing	Cargar una MV con un SO diferente al que tiene la máquina fisca, así se puede probar los desarrollos realizados en otros SO.
Portabilidad de Aplicaciones	Se puede encapsular la aplicación son los archivos temporales que utiliza para su ejecución

Fuente: (Pérez & Pérez, 2014)

TABLA 3 Beneficios de la virtualización

Beneficios de la virtualización	
Disminución de costos	Entre más MV menor será el costo de adquisición de hardware
Reduce tiempos de despliegue	El tiempo de aprovisionar una aplicación se reduce considerablemente
Flexibilidad y escalabilidad	Mas flexible al poner servicios en producción, por los tiempos de despliegue
Gestión de las MV	Gestión centralizada de todas las MV implementadas en una compañía
Ahorro de energía	Se reduce el consumo eléctrico y la alimentación de estos como la refrigeración

Fuente: (Pérez & Pérez, 2014)

1.3 Contenedores Linux

1.3.1 Nueva era de la virtualización

Un nuevo paradigma para la gestión de software resalta en la actualidad, se trata de la contenerización que gracias a la capacidad que tienen los contendores se logra reducir esfuerzos de desarrollo e implementación de software personalizado siendo de alta gama en infraestructura informática (Younge, Pedretti, Grant, & Brightwell, 2017).

"Chroot" era una tecnología Linux que permitía aislar procesos individuales entre sí, sin emular hardware diferente, a partir de esta tecnología aparecen los contenedores como sistemas operativos livianos que se ejecutan dentro del sistemas operativo host que los contiene, usando instrucciones nativas sin necesitar un administrador de memoria virtual (Kovács, 2017). Son mucho más pequeños en tamaño que una máquina virtual, una imagen de contenedor es en megabytes, y un disco virtual con las mismas aplicaciones su tamaño llega a ser de gigabytes (Maenhaut, Volckaert, Ongenae, & De Turck, 2019).

"Los contenedores son una tecnología emergente que promete mejorar la productividad y el código" (Hale, Li, N., & Wells, 2017). Permiten especificar la construcción del entorno de software para códigos determinados, entre ellos se puede establecer la base de un SO, bibliotecas, variables de entorno, y la manera de compilar una aplicación. Tienen la capacidad de probar aplicaciones en un gran número de servidores.

1.3.2 Diferentes técnicas de virtualización con contenedores

1.3.2.1 Linux Containers (LXC)

Es una de las primeras técnicas de contenedor utilizadas; la virtualización es a nivel de sistema operativo ejecutando múltiples contenedores en un solo núcleo de Linux, funciona como parche en el Kernel de Linux, tiene su enfoque más cercano al de una máquina virtual, pero sin sobrecarga (Bachiega et al., 2018). El contenedor comparte el núcleo con el sistema operativo, permitiendo que los procesos en ejecución sean visibles y manejables desde el SO host (Kovács, 2017).

1.3.2.2 OpenVZ (Open Virtuozzo)

Es un contenedor Linux para crear varios contenedores aislados, ejecutándose como servidor independiente, es un software de código abierto disponible para GNU GPL. Con esta tecnología cada contenedor puede reinicializar y contiene direcciones IP, memoria, procesos, aplicaciones y bibliotecas (Bachiega et al., 2018).

1.3.2.3 Kubernetes

Originalmente fue diseñado por Google siendo uno de los primeros colaboradores de la nueva tecnología de contenedores, llevan años de experiencia trabajando con estas arquitecturas, sus servicios son implementados en contenedores por más de 2 000 millones por semana (Acuña, 2016).

Es un sistema de código abierto, permite la creación de un clúster de grupos de host ejecutados en contenedores Linux para la eliminación de procesos manuales que son parte de la implementación y escalabilidad de aplicaciones en contenedores, de igual forma permite su administración con eficacia y facilidad (Red Hat, 2020).

Kubernetes es un sistema encargado de gestionar el clúster de servidores, distribuye contenedores a través del sistema, teniendo en cuenta los recursos disponibles. Esta arquitectura también permite realizar una gestión automatizada del estado y los aspectos más relevantes de los contenedores (Alvaro, 2018).

La Fig.7, representa una de las arquitecturas usadas para Kubernetes

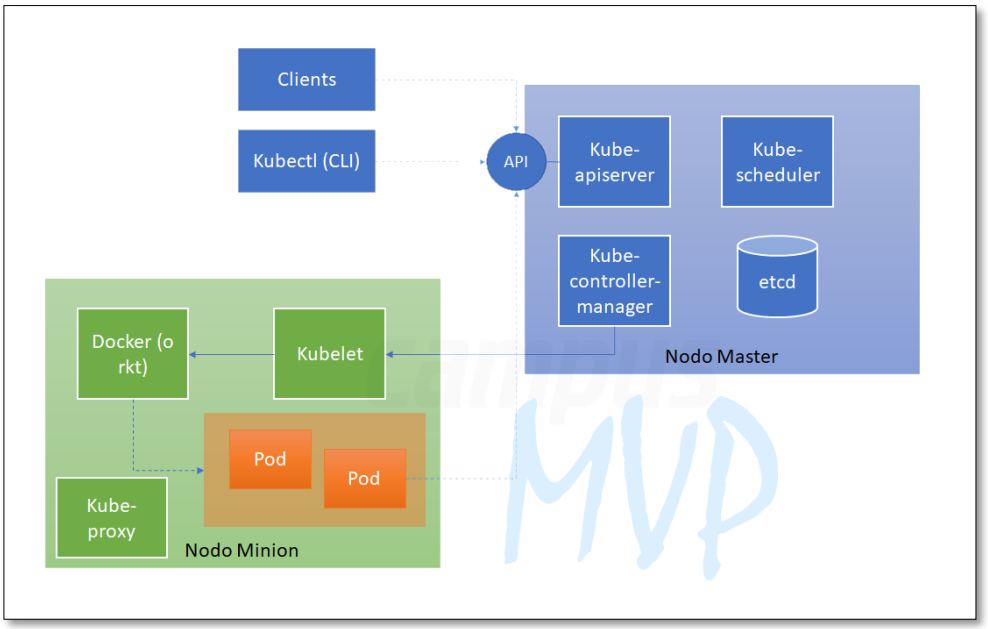


Fig. 7. Arquitectura Kubernetes

Fuente: (Tomás, 2019)

1.3.2.4 Docker

Es una de las primeras técnicas de virtualización con contenedores, llegó a remplazar LXC con su propia biblioteca, fue diseñado para desplegar microservicio en la nube y a través de esta técnica se reinventa la manera de desarrollar software con una virtualización ligera (Rudyy, 2019).

Emplea contenedores Linux, los procesos, las redes, los sistemas de datos y usuarios se encuentran separados entre sí permitiendo que su inicio sea rápido sin agregar mucha sobrecarga, utilizando una configuración personalizada de los recursos del sistema operativo (Dalgleish et al., 2016). Docker es como un multi inquilino donde los contenedores se comparten a través del mismo kernel de host, pero cada uno tiene su propio adaptador de red virtualizado y sistema de archivos como Cgroups y namespaces que segregan procesos para su independiente ejecución (Biradar, Shekhar, & Reddy, 2019).

Los contenedores Docker se publican en Docker Hub facilitando su implementación, este es un repositorio público para almacenar imágenes ya configuradas, también permite la creación de un repositorio de imágenes Docker privado para facilitar su administración en las organizaciones (Maenhaut et al., 2019).

La tecnología Docker usa el kernel de Linux y las funciones de este, ver la Fig.8, como Cgroups y namespaces, y se compone de las siguientes partes:

- Docker Daemon
- Docker Cliente CLI
- Docker Hub

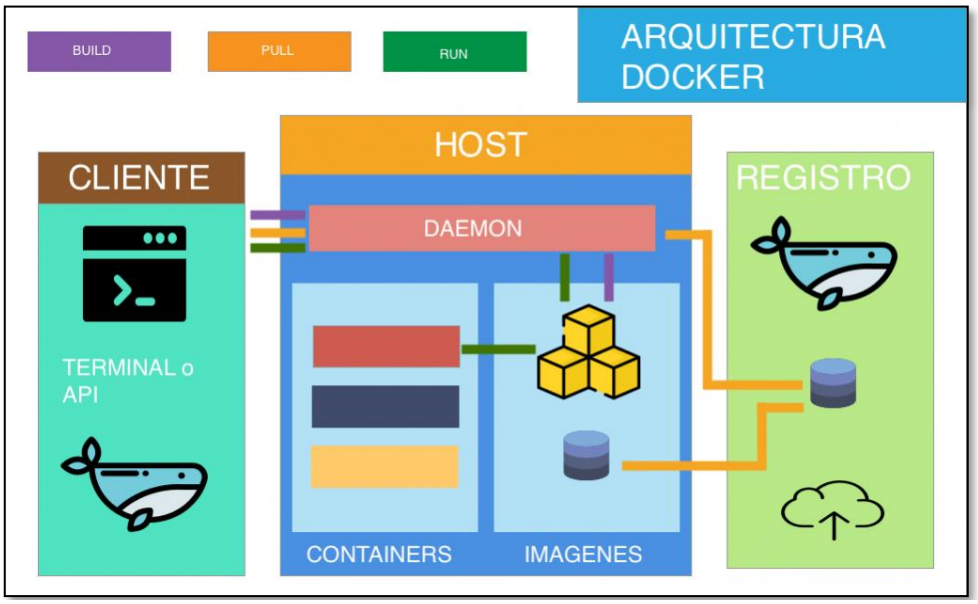
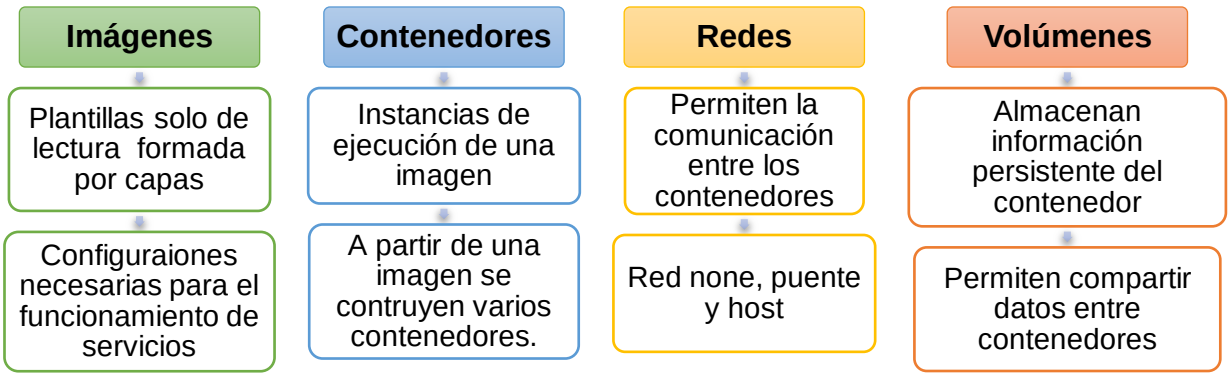


Fig. 8. Arquitectura Docker
Fuente: (Moya, 2018)

Las Imágenes, Contenedores, Redes y Volúmenes, que son conocidos como Objetos de Docker, indispensable conocerlos para trabajar.



1.3.3 Diferencias entre máquinas virtuales y contenedores

Los dos términos son considerados técnicas de virtualización, sin embargo, los contenedores surgieron con la finalidad de ahorrar recursos informáticos haciendo uso de un mismo núcleo kernel, que en lugar de virtualizar el hardware como lo hace la virtualización tradicional, se virtualice al propio sistema operativo (Hale et al., 2017). Ver la Fig.9.

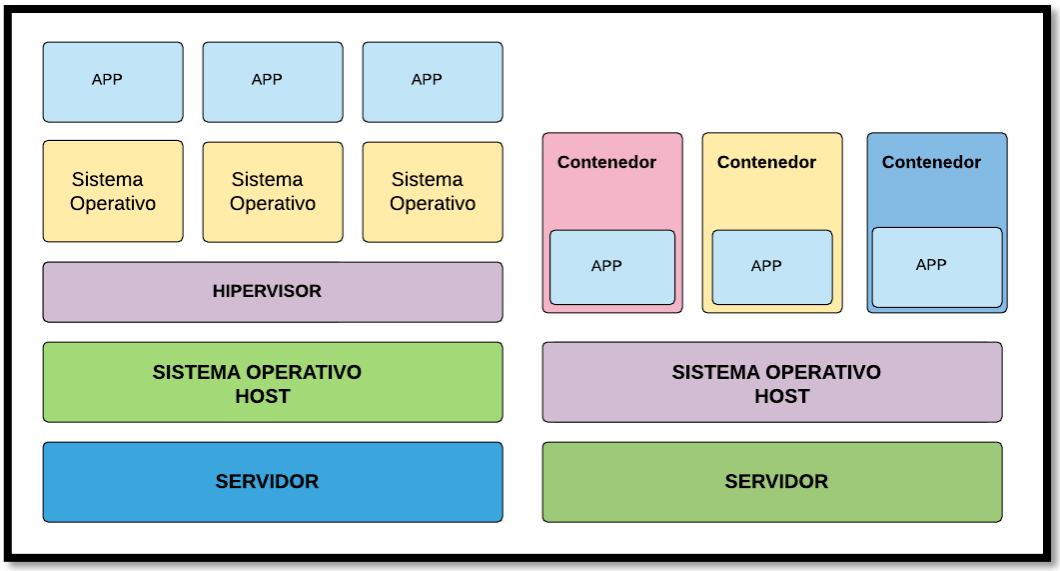


Fig. 9. Virtualización tradicional vs Contenedores

Fuente: Elaboración Propia

En la TBLA 4, se establecen las principales diferencia de las máquinas virtuales frente a los contenedores, estas características fueron tomadas de (Kovács, 2017) y (Bachiega et al., 2018).

TABLA 4 Diferencias entre Máquinas Virtuales y Contenedores

Máquinas Virtuales	Contenedores
El hipervisor que es una técnica de virtualización ejecuta al mismo tiempo múltiples sistemas operativos.	Método de virtualización a nivel de sistema operativo, se pueden crear múltiples sistemas aislados ejecutándose en un solo host y accediendo a un único kernel volviéndose eficientes por ejecutar varios casos bajo un único sistema operativo.
Actúan como IaaS (Infraestructura como servicio) con asignación y administración de hardware	Actúan como PaaS (Plataforma como servicio), proporcionando herramientas para soportar el software requerido
Cada SO utiliza diferentes kernels, incluso la misma distribución de Linux difiere en los núcleos, consumiendo así gran cantidad de recursos informáticos en su mayoría CPU y memoria	Tiene como objetivo proporcionar potencia y eficiencia informática a las aplicaciones, haciendo uso de un mismo núcleo reduciendo el consumo de recursos
La migración se realiza de un host a otro sin ningún problema	Carecen de la lista completa que ofrece la virtualización, es imposible realizar una migración en vivo, en este caso hay que detener el contenedor para moverlo a otro host
La máquina virtual arranca del propio kernel	El contendor lo hace desde el núcleo de trabajo del host
	Es utilizada para asignar recursos informáticos, utilizar aplicaciones interoperables y distribuir aplicaciones empaquetadas

1.4 Estándar ISO/IEC 25000

1.4.1 Definición de la ISO/IEC 25000

ISO/IEC 25000 es una familia de normas conocida como SQuaRE (System and Software Quality Requirements and Evaluation) cuyo objetivo es la creación de un marco de trabajo para la evaluación de la calidad de un producto software; es el resultado de la evolución de otras normas enfocadas en la descripción y evaluación de productos software de calidad. Está compuesta de 5 divisiones (ISO25000, 2019), como se observa en la Fig. 10.

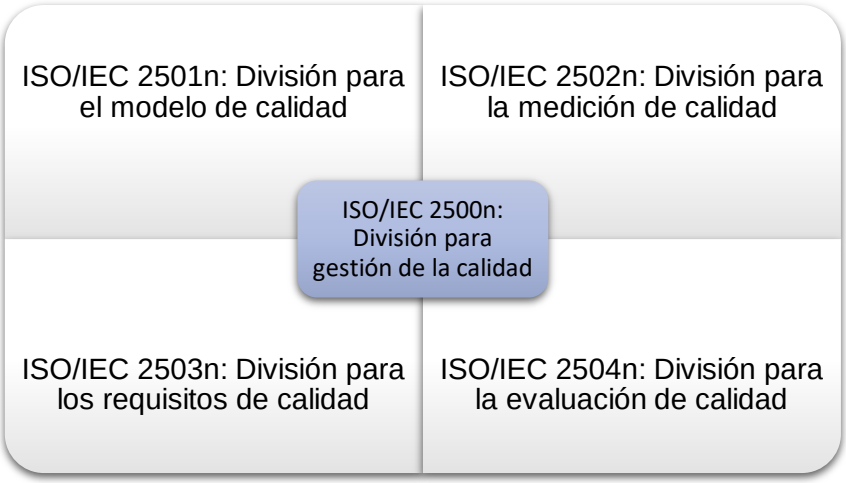


Fig. 10. Divisiones de la ISO/IEC 25000

Fuente: (ISO25000, 2019)

a. ISO/IEC 2500n - División de Gestión de Calidad: define todos los modelos, términos y definiciones de la familia SQuaRE (ISO25000, 2019).
b. ISO/IEC 2501n – División de Modelo de Calidad: establece definiciones, guía práctica y un modelo de referencia de la medición de la calidad de un producto (ISO25000, 2019).
c. ISO/IEC 2502n – División de Medición de Calidad: presenta un modelo de referencia de la medición de la calidad del producto, definiciones de medidas de calidad y guías prácticas para su aplicación (ISO25000, 2019).
d. ISO/IEC 2503n – División de Requisitos de Calidad: contribuye en la especificación de requisitos de calidad que pueden usarse en el proceso de elicitación de requisitos de calidad del producto software (ISO25000, 2019).
e. ISO/IEC 2504n – División de Evaluación de Calidad: dispone de normar que proporcionan requisitos, recomendaciones y guías a llevar a cabo en el proceso de evaluación del producto software (ISO25000, 2019).

Si se necesita medir y evaluar la calidad de un producto software se debe hacer uso de la norma ISO/IEC 25010 que es uno de los modelos que conforman la ISO/IEC 2501n para medir la calidad interna y externa del producto.

1.4.2 Modelo de Calidad ISO/IEC 25010

Este modelo de calidad establece las Características y Subcaracterística de calidad a tomarse en cuenta para evaluar las propiedades de un producto software, midiendo el grado de satisfacción de las necesidades declaradas por parte de sus interesados (ISO25000, 2019), la Fig. 11. Establece las ocho características de calidad que comprenden la ISO/IEC 25010.

Fig. 11 Divisiones de la ISO/IEC 25000

Fuente:(ISO25000, 2019)

De las ocho características que componen la ISO/IEC 25010, el presente trabajo se enfoca en la eficiencia de desempeño, la cual permite representar el rendimiento en base a los recursos utilizados en condiciones establecidas, con el fin de evaluar el rendimiento de las aplicaciones monolíticas Java EE desarrolladas por la CISIC, a través del uso de la virtualización tradicional y de los contenedores Docker.

Eficiencia en el Desempeño se compone de las siguientes Subcaracterística:

- **Comportamiento en el tiempo:** es el grado de satisfacción que presentan los requisitos de tiempos de respuesta, procesamiento y tasas de rendimiento de un sistema o aplicación en el momento que realiza sus funciones.
- **Utilización de recursos:** los requisitos de cantidad y tipos de recursos utilizados satisfacen las funciones realizadas por un sistema o aplicación
- **Capacidad:** grado de satisfacción de los requisitos de límites máximos de un sistema o aplicación.

1.4.3 Métricas de la característica eficiencia en el desempeño

La TABLA 5, muestra las métricas de calidad utilizadas para la validación de resultados.

TABLA 5. Métricas

Eficiencia en el desempeño	Comportamiento temporal	✓ Tiempo de respuesta ✓ Tiempo de espera ✓ Rendimiento
	Utilización de recursos	✓ Utilización de CPU ✓ Utilización de memoria
	Capacidad	✓ Número de peticiones online

A continuación, la TABLA 6, presenta la matriz con los datos y métodos a usarse en la evaluación de la eficiencia de Docker frente a la Tradicional.

TABLA 6. Matriz de Evaluación Eficiencia de Desempeño

Subcaracterística	Métrica	Propósito Métrica	Método de aplicación	Fórmula/ variables	Valor esperado	Tiempo de medida
Comportamiento del tiempo	Tiempo de respuesta	¿Cuál es el tiempo estimado para completar una tarea?	Tomar el tiempo desde que se envía la petición hasta obtener la respuesta	X = B - A A= Tiempo de envío de petición B = Tiempo en recibir la primera respuesta	0<=X<=1 El más cercano a 0 es el mejor. Donde el peor caso es >=15t.	X = Tiempo - Tiempo A = Tiempo B = Tiempo
	Tiempo de espera	¿Cuál es el tiempo desde que se envía una instrucción, para que inicie un trabajo, hasta que lo completa?	Tomar el tiempo cuando se inicia un trabajo y el tiempo en completar el trabajo	X = B - A A= Tiempo cuando se inicia un trabajo B = Tiempo en completar el trabajo	0<=X<=1 El más cercano a 0 es el mejor. Donde el peor caso es >=15t.	X = Tiempo - Tiempo A = Tiempo B = Tiempo
Utilización de recursos	Utilización de CPU	¿Cuánto tiempo de CPU es usado para realizar una tarea dada?	Tomar el tiempo de operación y la cantidad de tiempo de CPU que se usa para realizar una tarea	X = B-A A= La cantidad de tiempo de CPU que realmente es usado para realizar una tarea B = Tiempo de operación Dónde: B > 0	0<=X<=1 Cuanto más se acerque a 0 es lo mejor. Donde el peor caso es >=15t.	X = Tiempo - Tiempo A = Tiempo B = Tiempo
	Utilización de la memoria	¿Cuánto espacio de memoria es usado para realizar una tarea dada?	Medir la cantidad total de espacios de memoria y la cantidad de espacios de memoria que realmente es usado para realizar una tarea	X = B-A A = Cantidad de espacios de memoria que realmente es usado para realizar una tarea B = Cantidad total de espacios de memoria Dónde: B > 0	0<=X<=15 El más cercano a 0 es el mejor	X = Tamaño – Tamaño A = Tamaño B = Tamaño
Capacidad	Número de peticiones online	¿Cuántas peticiones online pueden ser procesadas por unidad de tiempo?	Contar el número máximo de peticiones online procesadas y tomar el tiempo de operación	X = A/T A= Número máximo de peticiones online procesada T = Tiempo de operación Dónde: T > 0	X = A/T El más lejano a 0/t es el mejor. Donde el mejor caso es >=10/t.	X = Contable / Tiempo A = Contable T = Tiempo

CAPITULO II

2 DESARROLLO

2.1 Materiales y métodos

2.1.1 Herramientas de trabajo

Proxmox

Plataforma de código abierto de nivel empresarial, similar a los productos de virtualización como VMware, vSphere, Hyper-V, Citrix, etc. Integra hipervisor KVM y contenedores LXC con almacenamiento definido por software y funcionalidad de red en una sola plataforma, permite la instalación en cualquier cantidad de “Servidores físicos”, sin límite con uso de Procesadores y Sockets. Facilidad de virtualizar cargas de trabajo exigentes, escalación dinámica y almacenamiento a medida que crecen las necesidades (Proxmox, 2020).

CentOS 7

Sistema operativo de código abierto y de clase empresarial, con distribución respaldada por Red Hat Enterprise Linux, no tiene costo y su redistribución es gratuita. Software robusto, estable, fácil de instalar y utilizar, cada versión de CentOS recibe soporte durante 10 años, siendo la versión actual la 7 lanzada en el 2014, misma que recibirá actualizaciones de seguridad hasta el 2024, usa yum como paquete de gestión de actualizaciones, es un Linux seguro de bajo mantenimiento, confiable, predecible y reproducible (CentOS, 2020).

CentOS es usado para la administración de sistemas, tiene gran popularidad como servidor web, por la reducción de problemas que presentan evitando caídas y errores al permitir solo versiones estables de software empaquetado. La versión de CentOS 7 permite la instalación del SO mínima y con interfaz gráfica dependiendo de las necesidades del usuario.

Putty

Cliente SSH y telnet de software libre que permite la comunicación con un servidor utilizando Windows, está disponible con código fuente, se encuentra desarrollado por un grupo de voluntarios (PuTTY).

PostgreSQL

Es un sistema de base de datos relacional de objetos de código abierto, que permite el envió y respuesta de datos en la aplicación (PostgreSQL, 2020).

Wildfly

Servidor de aplicaciones modular y ligero que permite la creación de aplicaciones, utiliza módulos JBoss para proporcionar un verdadero aislamiento de la aplicación (Red Hat).

Dockerfile

Archivo de configuración, que permite crear imágenes con instrucciones y comandos personalizados previamente establecidos, es aquí donde se generan las capas de una imagen, de tal manera que si se modifica el archivo Dockerfile al ejecutarlo se generan solo las capas modificadas, haciendo el proceso rápido y eficiente (Víctor Cuervo, 2019).

La TABLA 7, muestra las sentencias más usadas en un Dockerfile, para la creación de imágenes.

TABLA 7. Sentencias Dockerfile

FROM	Permite especificar una imagen desde la imagen oficial
RUN	Instrucciones que se pueden ejecutar en la terminal, ejecutando cualquier comando disponible en Linux
CMD	Comando que se ejecuta al hacer docker run
LABEL	Etiquetar la imagen
EXPOSE	Exponer puertos
ADD	Copiar archivos desde nuestra maquina hacia la imagen, permite poner URL's
COPY	Copiar archivos desde nuestra maquina hacia la imagen solo rutas locales
ENV	Variables de entorno a incluir en el contenedor

Con el comando de la Fig.12, la imagen Docker se construye ejecutando paso a paso las instrucciones establecidas dentro del Dockerfile.

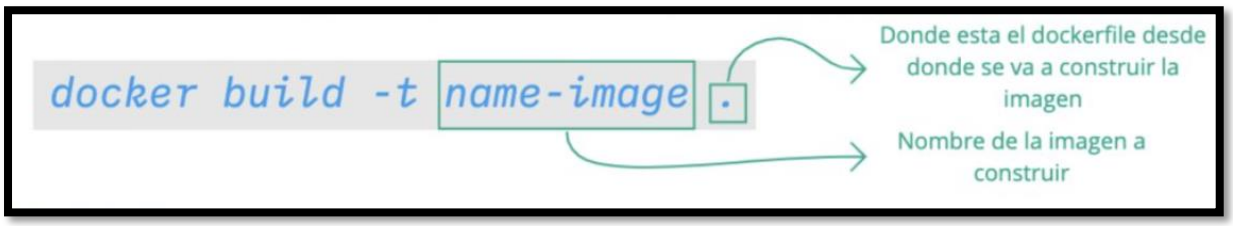


Fig. 12 Construir imagen

Fuente: Elaboración Propia

Docker-Compose

Herramienta que permite la ejecución de múltiples contendores, así como la configuración de servicios de aplicaciones, usando un solo comando que crea e inicia todos los servicios desde su configuración, a través de un archivo YAML. Es ideal para entornos de desarrollo, prueba y preparación, con el comando docker-compose up se inicia el contenedor con los parámetros establecidos (Docker Inc, 2020a).

Con docker-compose se logra simplificar el uso de Docker a través de las siguientes funcionalidades:

- Múltiples contenedores: permite la creación de contendores al mismo tiempo, con diversos entornos es de gran utilidad en la realización de pruebas en desarrollo y fundamental para la construcción de aplicaciones y microservicios.

- Conserva datos de los contenedores: se puede levantar y apagar contenedores sin riesgo a perdida de datos, esto se logra al mapeo de carpetas que realiza con la maquina host.

- Diferentes servicios: docker-compose permite crear servicios en cada contenedor, unirlos a un volumen, iniciarlos, apagarlos, habilitar puertos, conectarlos entre sí, etc.

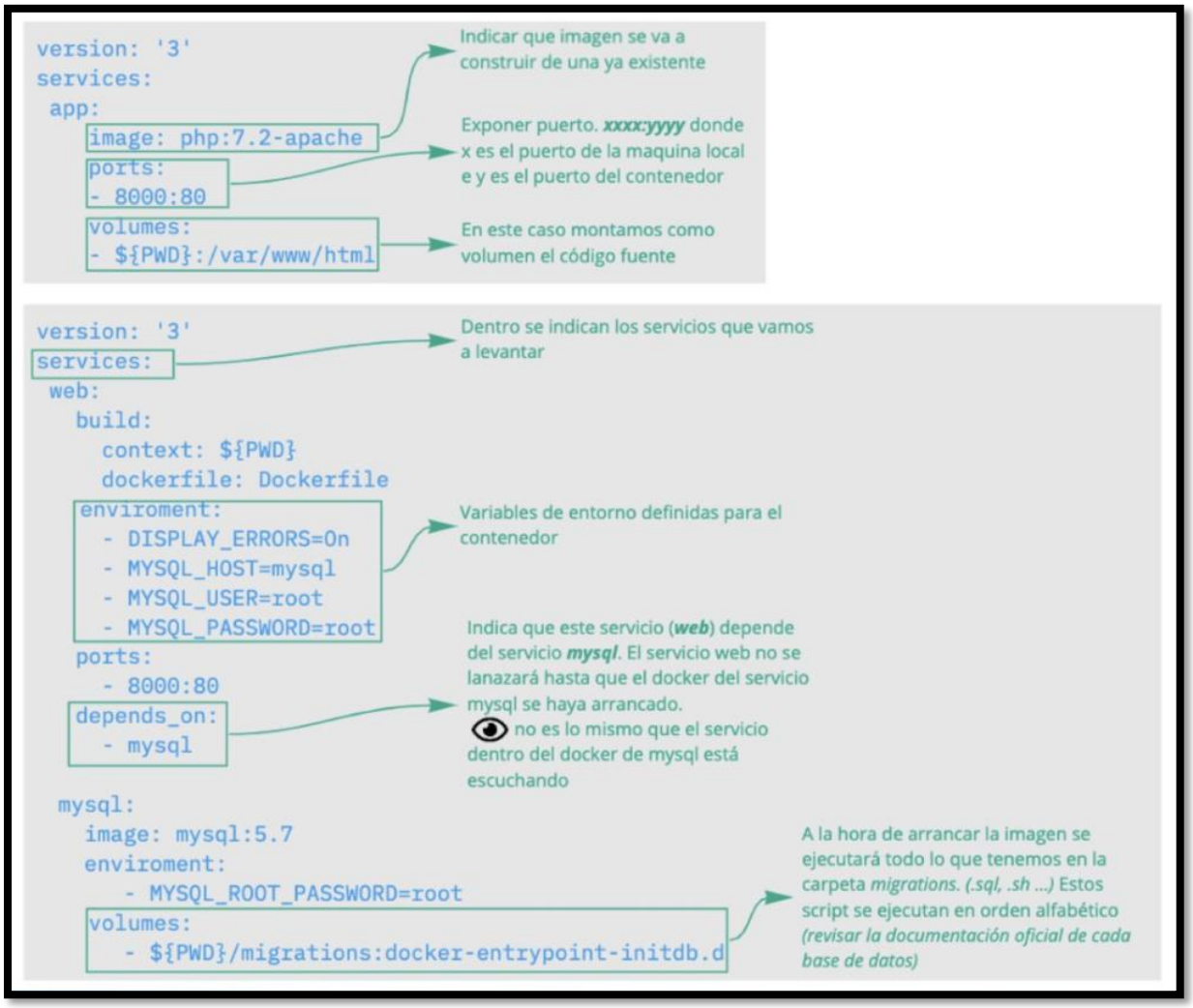


Fig. 13 Ejemplo Formato de archivo YML

Fuente: Elaboración Propia

Prometheus

Software de código abierto para monitoreo y alerta de sistemas; construido en SoundCloud en 2012, cuenta con una comunidad de usuarios y desarrolladores activa. Es un modelo de datos multidimensional, usa PromQL como lenguaje de consulta flexible, tiene múltiples soportes de gráficos y paneles, la recopilación de series de tiempo se da a través de un modelo de extracción por medio de HTTP (Prometheus, 2014).

2.1.2 Metodología de desarrollo Kanban

Kanban pertenece a las metodologías agiles, su principal objetivo es visualizar el trabajo y limitar la acumulación de tareas con eficiencia. “Nació para aplicarse a los procesos de

fabricación y con el tiempo se convirtió en un territorio reclamado por los desarrolladores de software"(Kanbanize).

Kanban tiene sus orígenes en Japón, por eso la palabra es compuesta por Kan y Ban que significa visual y tarjeta respectivamente, esta metodología gestiona las tareas a través de tarjetas visuales que organiza la realización de procesos y tareas, el tablero se compone por tres columnas: "Por Hacer", "En proceso" y "Hecho" que permiten ver el trabajo que se está realizando, evitando los cuellos de botella (Larriba, 2016).

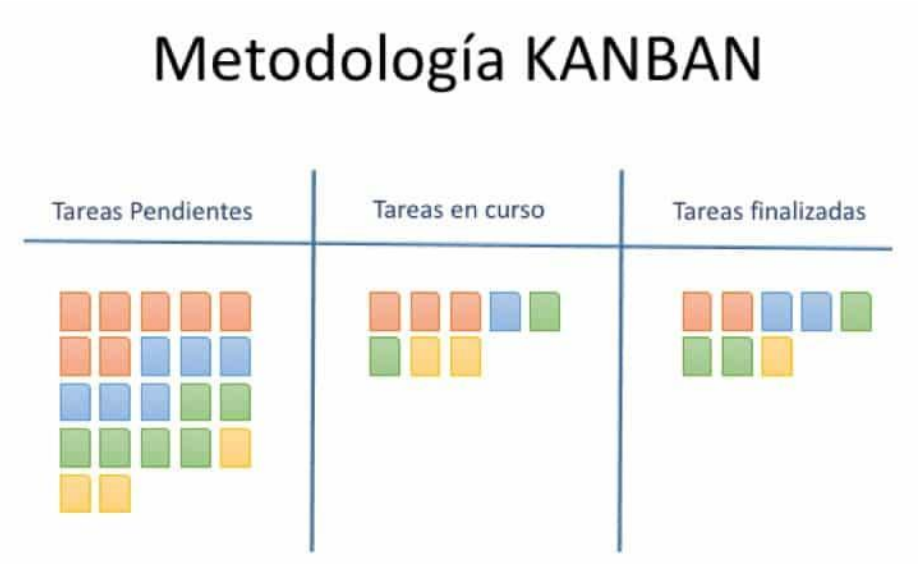


Fig. 14 Tablero Kanban

2.1.3 Método de evaluación

El método de evaluación se basa en el análisis de rendimiento de la máquina virtual con Docker instalado y ejecutándose las aplicaciones desarrollas con la arquitectura Java Enterprise Edition. Es decir, se realiza la comparativa entre maquinas con y sin implementación de Docker, obteniendo medidas de desempeño con las ISO 25010/IEC tomando en cuenta el uso de CPU, memoria RAM y tiempos de ejecución.

2.2 Comandos Básicos de Docker

Docker usa una combinación de comandos que permiten crear imágenes, ejecutar y administrar contenedores. En la TABLA 8 y 9, se presenta los comandos más usados por Docker y Docker Compose.

TABLA 8. Comandos ejecución y creación de contenedores

docker pull nginx:1.13-alpine	Descargar imagen ya construida
docker images | grep "nombre imagen"	Permite buscar imágenes especificas en Docker
docker images	Listar imágenes
docker build --tag "nombre imagen" .	Construir imagen a partir de Dockerfile.
docker run -d apache-centos	Crear contenedor a partir de la imagen.

docker history -H apache-centos:apache-cmd	Muestra historia del contenedor creado.
docker rm -fv "nombre contenedor"	Eliminar contenedor
docker run -d --name apache *-p 80:80* apache-centos:apache-cmd	Definir Puerto que utilizará el contenedor
docker rmi "nombre de images o id"	Eliminar imágenes
docker build -t test -f my-dockerfile .	Cambiar el nombre de un Dockerfile
docker images -f dangling=true	Ver imágenes huérfanas
docker images -f dangling=true -q | xargs docker rmi	Borrar grupo de imágenes huérfanas
docker ps - l	Ver último contenedor creado
docker rename wildfly wildfly-test	Cambiar nombre contenedor
docker exec -ti "nombre contenedor"	Entrar a la consola del contenedor
docker ps - q	Lista los id de los contenedores
docker ps -a	Listar contenedores en ejecución

TABLA 9. Comandos Docker Compose

	build	Construye las imágenes definidas en el archivo docker-compose
docker-compose	up	Levanta y construye los contenedores especificados en él
	up -d	Igual modo detach
	up - - build	Fuerza a construir la imagen
	ps	Lista de contenedores levantados
	log -f nombreservicio	Muestra los logs del servicio
	top	Muestra los procesos en ejecución
	stop	Detiene los contenedores especificados en el archivo
	down	Elimina los contenedores.

2.3 Contenerización de Aplicaciones

El uso de contenedores para construcción y despliegue de aplicaciones permite crear entornos de desarrollo, prueba y preparación, obteniendo beneficios de independencia de aplicación con el Sistema Operativo. Una sola máquina virtual puede ejecutar todas las aplicaciones en el mismo servidor, logrando disminuir el tiempo de despliegue y el uso de recursos del servidor.

Para empezar con la tecnología Docker, se realizó la contenerización de una aplicación web sencilla, creando contenedores que ejecutan Wildfly y PostgreSQL; esta aplicación permitió comprender el entorno y realizar pruebas de funcionamiento de Docker. La aplicación se containerizo haciendo uso de imágenes base alojadas en Docker Hub, el repositorio público en la nube mantenido por Docker, que permite almacenar y descargar imágenes facilitando su uso.

2.3.1 Tablero Kanban

Haciendo uso de los tableros Kanban se establecen las tareas a realizarse en la fase de desarrollo de contenedores Docker, detallando las actividades y priorizando cada una de ellas.

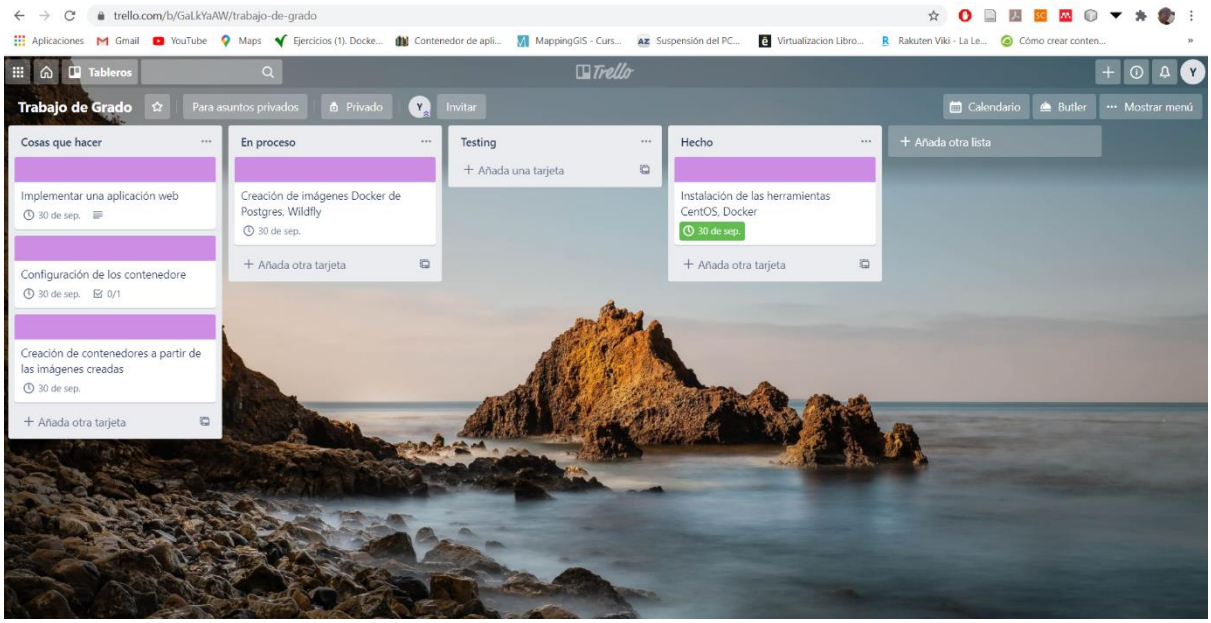


Fig. 15. Tablero Kanban

2.3.2 Instalación y configuración de herramientas

<u>Creación de máquina virtual</u>

Como punto de partida para empezar con el desarrollo de los contenedores, se configuraron tres máquinas virtuales, cada una con una instalación mínima de CentOS7, con las características de hardware requeridas para su funcionamiento.

De las tres máquinas creadas, dos se usaron como servidor de aplicaciones contendiendo la arquitectura Docker y la arquitectura tradicional respectivamente, en la tercera máquina se instaló un software para monitorear los servidores ya mencionados.

Instalación de CentOS 7

En el presente trabajo se usó la instalación mínima del CentOS 7 (véase anexo 1), que no ocupa mucho espacio en disco y es eficiente como servidor web. Los requisitos necesarios para el correcto funcionamiento del Sistema Operativo son: un procesador de 1ghz, RAM de 64 MB y 1 GB de espacio en disco.

Instalación de Docker

Para la instalación de Docker debemos tener en cuenta sobre qué Sistema Operativo se ejecutará, ya que se puede usar Windows o cualquier distribución de Linux, en este caso se usa CentOS 7 y el paquete yum.

Instalar las dependencias y los controladores de almacenamiento de Device Mapper para el funcionamiento Docker.

- yum install -y yum-utils device-mapper-persistent-data lvm2

Agregar el repositorio de Docker a CentOS 7, empezar la instalación.

- yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
- yum install docker-ce -y

Administrar el servicio de Docker con los comandos:

- systemctl start docker
- systemctl enable docker
- systemctl status docker

Instalación de Docker-compose

Se utiliza la herramienta docker-compose para ejecutar multi contenedores, en este caso PostgreSQL y Wildfly simultáneamente, estableciendo su configuración respectiva y la comunicación entre los contenedores.

En Linux se puede descargar docker-compose desde el repositorio de Compose en GitHub, ejecutando el comando curl se descargará la versión más actual.

- sudo curl -L "https://github.com/docker/compose/releases/download/1.27.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Aplicar los permisos de ejecución

- sudo chmod +x /usr/local/bin/docker-compose

Por último, se debe comprobar la instalación de docker-compose y su versión

- docker-compose –version

2.3.3 Creación de imágenes Docker

Docker tiene su repositorio donde se almacenan y descargan imágenes ya creadas, pero de igual manera permite la creación de nuevas imágenes con los servicios que se requiera. Para esto se hace uso de Dockerfile, un documento de texto con instrucciones y comando que permiten ensamblar una imagen.

Las imágenes son plantillas que contienen el software que una aplicación necesita para funcionar, se pueden crear a partir de otras en forma de capas personalizando las imágenes. En esta ocasión a través de este archivo Dockerfile se crearon las imágenes de PostgreSQL y Wildfly.

a. Imagen PostgreSQL

PostgreSQL, hoy en día uno de los motores de base de datos de mayor demanda debido a sus características y funcionalidades, cuenta con variedad de versiones y distintas formas de instalación, una de ellas mediante una imagen Docker que contiene una instalación ya configurada y lista para usarse.

Docker permita usar variables de entorno para personalizar el uso de la imagen PostgreSQL, una vez que se inicie el contenedor las variables empezaran cumplir su función, las más necesarias son:

- POSTGRES_PASSWORD
- POSTGRES_USER
- POSTGRES_DB

A través de Docker Hub se descarga la imagen de postgres con la última versión disponible, si se desea descargar una versión especifica se hace uso del tag que dispone la misma página.

- docker pull postgres
- docker pull postgres:9.6

Una vez descargada la imagen se usa el comando **docker images** verificando que la imagen se encuentre ya en el computador.

Ejecutar el contenedor Docker con la imagen de PostgreSQL descargada

- docker run -d --rm --name postgres -e POSTGRES_PASSWORD=mysecretpassword -p 5432:5432 postgres

El comando **docker ps,** permite verificar si el contenedor esta activo, en el caso de que el contenedor no aparezca, se debe a errores que se producen al crear el contenedor, en este caso el contenedor se crea, pero no se encuentra activo, el comando **docker ps -a** permite visualizar todos los contenedores que este inactivos o muertos.

b. Imagen Wildfly

Motor de aplicaciones e-business, es compatible con cualquier SO, por sus características y rendimiento, se adapta al trabajo en la nube aprovechando de las ventajas de Cloud, asegurando la accesibilidad desde cualquier lugar. Una aplicación que se desarrolla con Wildfly puede soportar cantidades de usuarios, tráficos o necesidades de procesamiento (Arsys, 2017).

Docker permite ejecutar un contenedor con una aplicación implementada en Wildfly, iniciando desde el directorio deployments, para lograr esto se necesita crear una nueva imagen de Wildfly a partir de la imagen base, añadiendo el directorio que puede ser **/opt/jboss/wildfly/standalone/deployments/** o **/opt/jboss/wildfly/domain/deployments/**, en este caso la creación de la imagen se utiliza Dockerfile donde se añaden algunas instrucciones para implementar la aplicación y configurar el datasource de Wildfly.

El Dockerfile tendrá el siguiente contenido como se puede ver en la Fig.16:

- Instrucciones para añadir el .jar de postgresql
- Configuración del datasource
- Añadir la aplicación para su despliegue
- Instalar el JDK
- Creación de usuario para acceder a la consola de Wildfly
- Iniciar Wildfly de modo independiente

```
FROM jboss/wildfly:19.0.0.Final

#despliegue de jar en postgres
ENV postgres_module_dir=/opt/jboss/wildfly/modules/system/layers/base/org/
postgresql/main/
RUN mkdir -p ${postgres_module_dir}
ADD module.xml ${postgres_module_dir}
ADD postgresql-42.2.5.jre6.jar ${postgres_module_dir}

#configuracion de datasource
ENV config_dir=/opt/jboss/wildfly/standalone/configuration
ADD standalone.xml ${config_dir}

# Deploy aplicación
ADD http://download1641.mediafire.com/8u3n6rqkg7ag/2q2krkidhnmnszi/academico.ear /
opt/jboss/wildfly/standalone/deployments/
USER root
RUN chown jboss:jboss /opt/jboss/wildfly/standalone/deployments/academico.ear
#Instalar el jdk
RUN yum -y install java-1.8.0-openjdk-devel && yum clean all
USER jboss
ENV JAVA_HOME /usr/lib/jvm/java
RUN mkdir /opt/jboss/wildfly/standalone/log
EXPOSE 8080

# Crear el usuario de Wildlfy
RUN /opt/jboss/wildfly/bin/add-user.sh admin admin --silent

# This will boot WildFly in the standalone mode and bind to all interface
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement",
"0.0.0.0"]
```

Fig. 16. Archivo de configuración Dockerfile

Fuente: Elaboración Propia

2.3.4 Creación y configuración de contenedores

El uso de contenedores Docker es común en el desarrollo de software, debido a la simplificación del proceso de despliegue y entrega de aplicaciones, permite la solución de problemas que pueden presentarse con la virtualización tradicional, montando el código en un entorno de pruebas consistente.

La ejecución de un contenedor se realiza mediante las imágenes descargadas en nuestro sistema, estas imágenes proporcionan plantillas con información requerida que permiten la creación y ejecución de los contenedores.

Primero se visualiza las imágenes que se encuentran en el sistema, con el comando **docker images** este listara todas las imágenes con su nombre y número de ID, una vez localizada la imagen se procede a ejecutarla creando contenedores a partir de esa imagen.

El comando que crea el contenedor establece **--name contenedor** para nombrar el proceso en ejecución y **apache-centos** es el nombre de la imagen base.

- docker run --name contenedor apache-centos

Las aplicaciones que se van a desplegar con Docker han sido desarrolladas con la arquitectura Java Enterprise Edition (JEE) y requieren de la creación de dos contenedores para el motor de base de datos PostgreSQL y servidor de aplicaciones Wildlfly que se conecten y permitan la comunicación entre sí. Existen tres maneras posibles de conectar contenedores:

- Enlace de Docker permite vincular uno o más contenedores Docker
- Red de Docker crea una red para conectar los contenedores a esa red
- Docker Compose crea una red compartida para todos los contenedores automáticamente

Cuando se necesita crear varios contenedores se usa docker compose que permite gestionarlos a través de un archivo YAML donde se definen los componentes de la aplicación, sus contenedores, configuración, enlaces, volúmenes, con una red compartida para su conexión de manera fácil y sencilla. Un solo comando logra ponerlo en marcha y al ejecutar dicho archivo que se lo llama docker-compose.yml los contenedores se enlazan, sin embargo, no comparten variables de entorno, por eso es necesario definirlas previamente.

El archivo docker-compose.yml usado para la construcción de los contenedores contiene las siguientes instrucciones y configuraciones:

- “version ‘3’”: Los archivos docker-compose.yml deben ser versionados, Docker evoluciona y aparecen nuevas versiones que tendrán compatibilidad con la anterior, por eso es importante indicar la versión lo que significa que es muy importante indicar la version de las instrucciones que queremos.

- “wildfly”: Es el nombre del servicio que se va a construir, aquí se indica el nombre que se desea haciendo referencia al tipo de servicio, en este caso son dos servicios “wildfly” y “postgresdb”.

- “build.”: Se utiliza para indicar que se construirá el contenedor a partir de un Dockerfile definiendo **“.”** automáticamente toma el Dockerfile que se encuentra en el directorio actual. Para ejecutar un Dockerfile de otro directorio se establece la ruta donde se encuentra, por ejemplo: **“./wildfly”**.

- “environment”: se establecen variables de entorno a los contenedores que usualmente pueden ser usuarios o contraseñas.

- “ports”: permite mapear los puertos que utilizan los servicios de Wildfly y PostgreSQL, con los puertos 8080 y 9990 se accede a la consola de wildfly y con el puerto 5432 la base de datos postgres se ejecutara, así se prueba el sitio que genera wildlfy.

- “volumes”: se puede mapear el directorio actual directamente con el directorio donde se crean los servicios, logrando una persistencia de los datos, cualquier cambio que se de en el directorio local se hará en el contenedor. De igual forma si elimina el

contenedor y se lo vuelve a crear, se demora menos en funcionar por los datos que se quedaron guardados

```yaml
version: '3'
services:

  wildfly:
    build: ./wildfly
    container_name: "wildfly"
    ports:
      - "8080:8080"
      - "9990:9990"
    links:
      - postgresdb:postgresql
    environment:
      - POSTGRES_PORT_5432_TCP_ADDR=postgresql
      - POSTGRES_PORT_5432_TCP_PORT=5432
      - POSTGRES_ENV_POSTGRES_USER=postgres
      - POSTGRES_ENV_POSTGRES_PASSWORD=postgres
      - POSTGRES_DATABASE_NAME=academico
  postgresdb:
    build: ./postgres
    container_name: "postgres"
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    volumes:
      - pg_data:/var/lib/postgresql/data
volumes:
  pg_data:
```

Fig. 17 Archivo de configuración YML

Fuente: Elaboración Propia

2.4 Implementación

Se implementa la aplicación web en los contenedores ejecutando Wildfly 19.0.0.1 y PostgreSQL 9.6, como ya se lo redacto en los puntos anteriores para la creación de los contenedores se usó como imagen base las imágenes que se encuentran alojadas en Docker Hub personalizándolas a través del archivo Dockerfile, aquí se añadieron las configuraciones necesarias para el funcionamiento de Wildfly e integración con la base de datos PostgreSQL.

Para el despliegue de la aplicación se requiere que el servidor esté previamente configurado e instalado CentOS 7, Docker y Docker Compose, una vez que toda la infraestructura docker se encuentre creada se procede a ejecutar el archivo docker-compose.yml que contiene las instrucciones para crear los contenedores junto con los Dockerfile personalizados con las herramientas que requiere la aplicación web.

En el archivo Dockerfile de postgres se establece la instrucción para ejecutar el script .sql que creara la base de datos junto con el contenedor. También se lo realiza ingresando a la consola del contenedor de postgres con el comando: **docker exec -ti postgres bash** y crear la base de dato manualmente, una vez realizado este procedimiento se reinicia el contenedor.

Es necesario crear el Datasource para que la aplicación puede mostrar los datos que se encuentran en postgres, este será el paso para comprobar que los dos contenedores se encuentran conectados y permiten su comunicación. Se agrega el archivo standalone.xml con

el nombre de la base de datos, el usuario y contraseña de postgres, a la ruta /opt/jboss/wildfly/standalone/configuration.

```
#configuracion de datasource
ENV config_dir=/opt/jboss/wildfly/standalone/configuration
ADD standalone.xml ${config_dir}
```

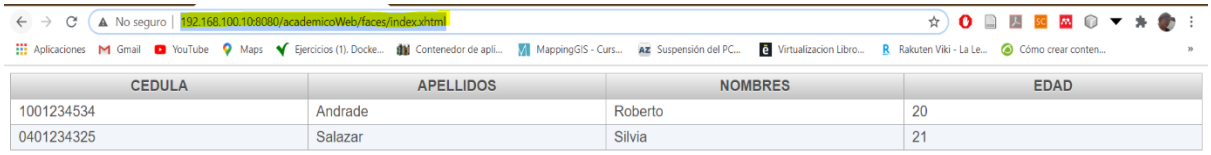
Fig. 18. Añadir archivo standalone.xml
Fuente: Elaboración Propia

La aplicación puede ser desplegada de dos formas, subiendo el archivo. ear desde la interfaz web de Wildlfy o agregando el archivo en la carpeta deployments dentro de wildlfy, en este caso se establece la instrucción ADD en del Dockerfile estableciendo la ruta de donde se encuentra el archivo y a donde debe ser copiado, como lo sugieren las instrucciones en Docker Hub.

```
# Deploy aplicación
ADD http://download1641.mediafire.com/8u3n6rqkg7ag/2q2krkidhnmnszi/
academico.ear /opt/jboss/wildfly/standalone/deployments/
```

Fig. 19. Añadir aplicación dentro del contenedor
Fuente: Elaboración Propia

Una vez que los contenedores estén configurados y ejecutándose se prueba la aplicación ingresando a la interfaz web de Wildfly con la IP del servidor y el puerto 8080, véase la Fig.20:



CEDULA	APELLIDOS	NOMBRES	EDAD
1001234534	Andrade	Roberto	20
0401234325	Salazar	Silvia	21

Fig. 20. Aplicación desplegada con Docker
Fuente: Elaboración Propia

CAPÍTULO III

3 RESULTADOS

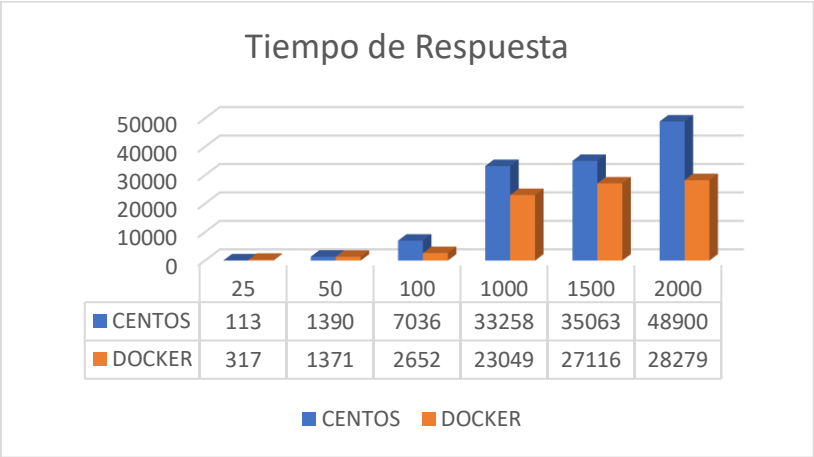
3.1 Pruebas

En las pruebas de funcionamiento de Docker, se utilizó el modelo de calidad de la ISO/IEC 25010, donde se establece el sistema para evaluar la calidad de un producto, seleccionando la característica Eficiencia de desempeño. Las Subcaracterística a medir fueron: Comportamiento temporal, Utilización de recursos y Capacidad; en base a ellas se muestra los resultados obtenidos, de las pruebas realizadas a la aplicación desplegada con la arquitectura Docker y la arquitectura tradicional respectivamente a través de las herramientas J-Meter y Prometheus.

3.1.1 Comportamiento temporal

En las pruebas realizadas sobre el comportamiento del tiempo se evaluaron los datos resultantes de la herramienta J-Meter en un formato de segundos (ss) y milisegundos(ms) para los Tiempos de Respuesta y Espera. En el caso del Rendimiento los datos son de tipo número, es decir el número de peticiones por segundo que acepta la aplicación.

a. Tiempos de respuesta



	25	50	100	1000	1500	2000
CENTOS	113	1390	7036	33258	35063	48900
DOCKER	317	1371	2652	23049	27116	28279

Fig. 21.Datos, Tiempo de respuesta
Fuente: Elaboración Propia

Como se observa en la Fig. 21, los datos resultantes sobre el tiempo de respuesta en las diferentes arquitecturas se ven afectados de acuerdo con el número de hilos usados para acceder a la aplicación. Es decir, en los 25 hilos Docker requiere de más tiempo de respuesta sin embargo en los 2000 hilos el tiempo de respuesta de la arquitectura tradicional duplica a la de Docker.

b. Tiempos de espera

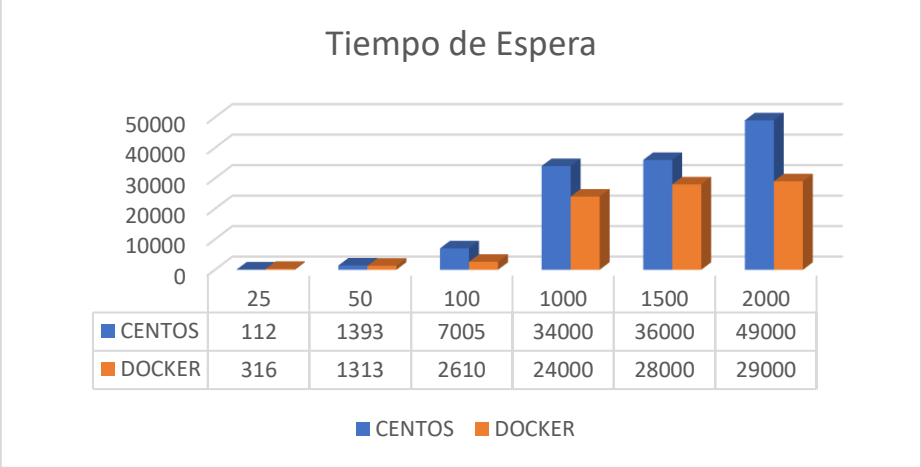


Fig. 22.Datos, Tiempo de Espera

Fuente: Elaboración Propia

Como se observa en el Fig. 22, los datos resultantes sobre el tiempo de respuesta en las diferentes arquitecturas se ven afectados de acuerdo con el número de hilos usados para acceder a la aplicación. Es decir, en los 25 hilos Docker requiere de más tiempo de espera. Sin embargo, en los 2000 hilos el tiempo de espera de la arquitectura tradicional duplica a la de Docker.

c. Rendimiento

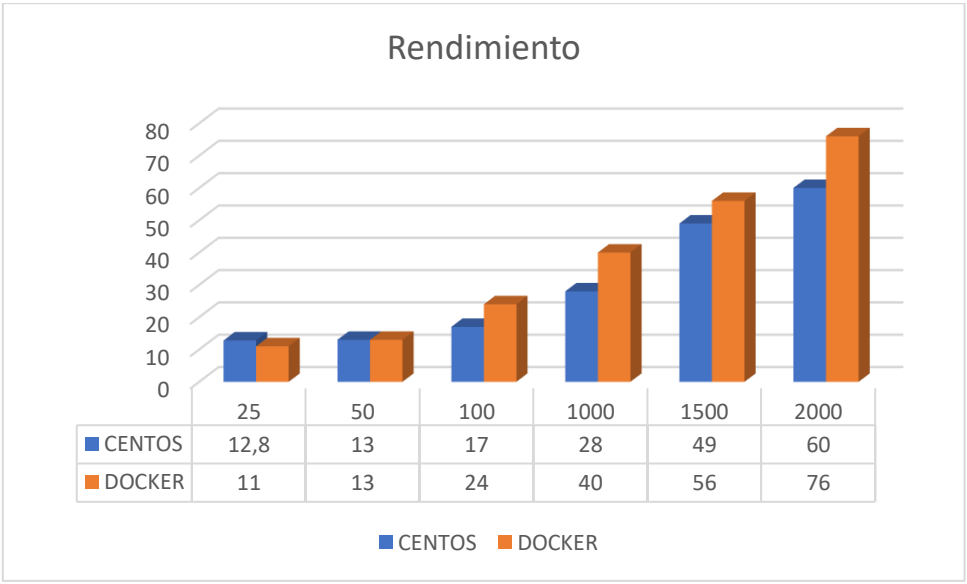


Fig. 23. Datos, Rendimiento

Fuente: Elaboración Propia

Como se observa en el Fig. 23, la arquitectura tradicional tiene mejor rendimiento cuando el grupo de hilos es bajo, en los 50 hilos las dos arquitecturas presentan el mismo rendimiento y cuando el número de hilos sube de 100 a 2000, Docker presente mejor rendimiento.

3.1.2 Utilización de Recursos

Los datos que se presentan en la *Utilización de CPU y Memoria* se los obtuvo a través de la herramienta Prometheus, con un formato de segundos (ss) y milisegundos(ms) en el caso de consumo de CPU. Para la medición del Uso de Memoria los datos son en unidades de información representándose en Megabyte.

a. Utilización de CPU

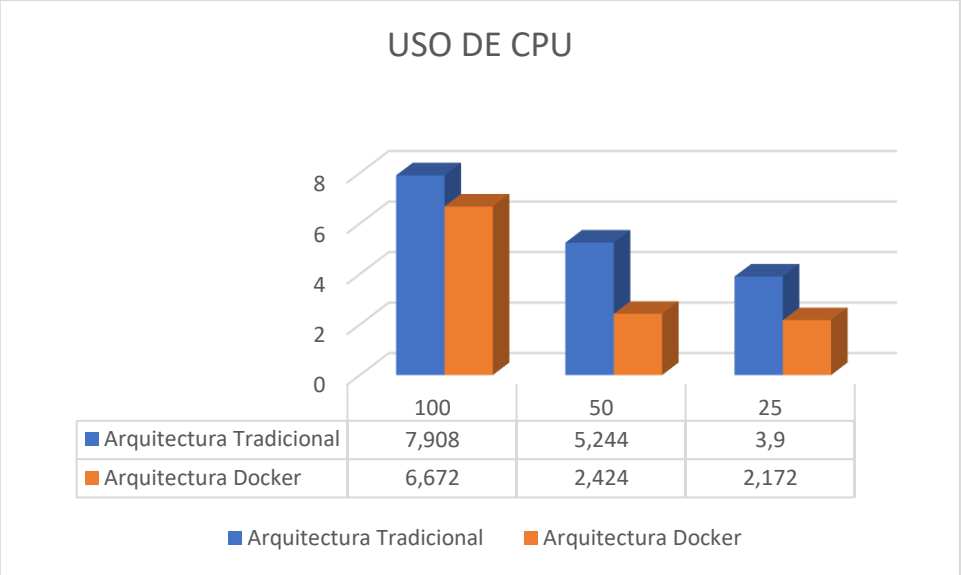


Fig. 24. Datos, Uso de CPU

Fuente: Elaboración Propia

Como se observa en el Fig. 24, en los tres diferentes grupos de hilos la arquitectura tradicional tiene más tiempos de utilización de CPU a diferencia de Docker que en los tres casos está presentando ahorro de CPU.

b. Utilización de memoria

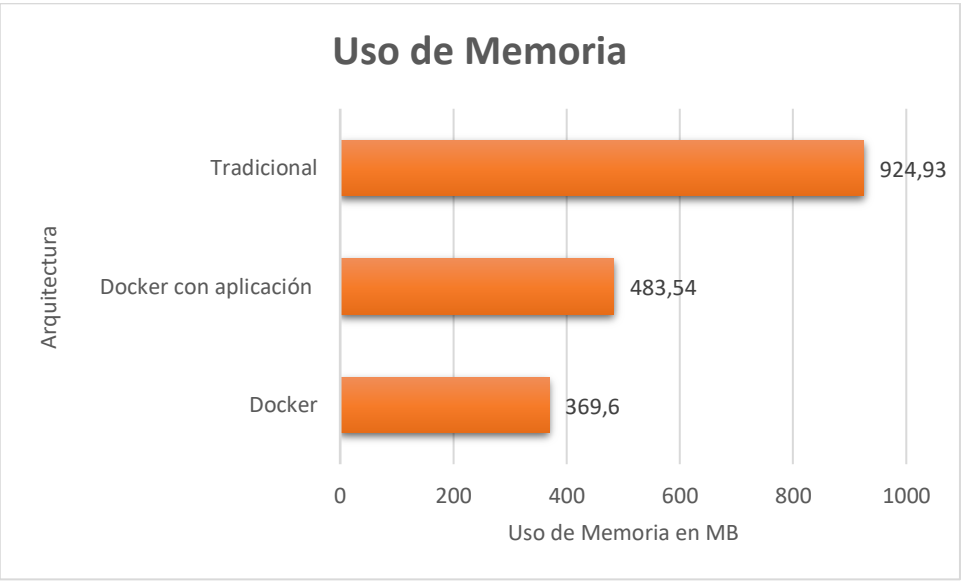


Fig. 25. Datos Uso de Memoria

Fuente: Elaboración Propia

Como se observa en el Fig. 25, el uso de memoria con Docker es bajo; una vez que se ejecuta la aplicación, la memoria sube 113 MB más a la inicial. Sin embargo, si se observa el uso de memoria con la arquitectura Tradicional, él valor se duplica.

3.1.3 Capacidad

En las pruebas de *Numero de Peticiones Online* los datos que se presentan son de tipo número.

a. Número de peticiones

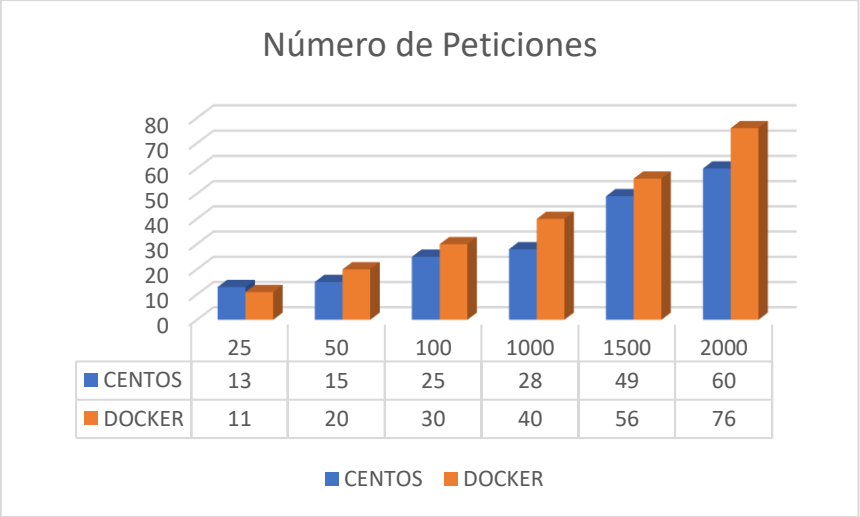


Fig. 26. Datos, Número de Peticiones Online

Fuente: Elaboración Propia

Como se observa en el Fig. 26, en los tres diferentes grupos de hilos a pesar de que Docker tiene más peticiones que la tradicional, entre las dos arquitecturas no se presentan grandes cambios en los hilos 25, 50 y 100, sin embargo, se puede ver un cambio significativo en los 2000 hilos.

3.2 Validación de Resultados

Para la aplicación del estándar ISO 25010 antes mencionado se utilizó las métricas especificadas en la ISO 25023, a través de una matriz en la que se establecen valores por cada Subcaracterística, que está evaluado sobre 10; tomando en cuenta el Peor caso y el Valor Deseado. Para ello se usaron las siguientes Subcaracterística: tiempo de respuesta, tiempos de espera, utilización del CPU, utilización de la memoria y número de peticiones online, con el propósito de validar el funcionamiento de cada arquitectura.

Para la obtención de datos y evaluación de las métricas se tomaron los grupos de hilos de: 100, 2000 respectivamente, . En el caso del Uso de memoria se evaluó con la arquitectura Docker, Docker ejecutando aplicación y Arquitectura tradicional.

a. Arquitectura Tradicional

Caracterísitca	Subcaracterística	Métrica	Peor caso	Valor Deseado	Variables			Valor Obtenido	Valor Métrica /10	Final subcaracterística	Total característica
					A	B	T	X			
Eficiencia de Desempeño	Comportamiento en el tiempo	Tiempo de respuesta	>=8 seg	<=4seg	0	7,036		7,036	7	3,2	7,96
		Tiempo de espera	>=7 seg	<=4seg	0,281	7,286		7,005	7		
		Rendimiento	0	>=5	68		4	17	10		
	Utilización de Recursos	Utilización de CPU	>=8seg	<=4seg	0,066	7,973		7,9071	6,5	2,76	
		Utilización de la memoria	>=1000mb	<=500mb	86,65	1011,58		924,93	7,3		
	Capacidad	Número de peticiones online	0	>=10	160		4	40	10	2	

Fig. 27. Evaluación Eficiencia de Desempeño concurrencia=100

Fuente: Elaboración Propia

Caracterísitca	Subcaracterística	Métrica	Peor caso	Valor Deseado	Variables			Valor Obtenido	Valor Métrica /10	Final subcaracterística	Total característica
					A	B	T	X			
Eficiencia de Desempeño	Comportamiento en el tiempo	Tiempo de respuesta	>=50,000 seg	<=25,00 seg	0	48,900		48,900	5	2	6,76
		Tiempo de espera	>=40,000 seg	<=20,000 seg	10,08	59,081		49,000	0		
		Rendimiento	0	>=5	2400		40	60	10		
	Utilización de Recursos	Utilización de CPU	>=8seg	<=4seg	0,066	7,973		7,9071	6,5	2,76	
		Utilización de la memoria	>=1000mb	<=500mb	86,65	1011,58		924,93	7,3		
	Capacidad	Número de peticiones online	0	>=10	2400		40	60	10	2	

Fig. 28. Evaluación Eficiencia de Desempeño concurrencia=2000

Fuente: Elaboración Propia

b. Arquitectura Docker

Caracterísitca	Subcaracterística	Métrica	Peor caso	Valor Deseado	Variables			Valor Obtenido	Valor Métrica /10	Final subcaracterística	Total característica
					A	B	T	X			
Eficiencia de Desempeño	Comportamiento en el tiempo	Tiempo de respuesta	>=8 seg	<=4seg	0	2,652		2,652	10	4	9,6
		Tiempo de espera	>=7 seg	<=4seg	0,62	3,236		2,616	10		
		Rendimiento	0	>=5	96		4	24	10		
	Utilización de Recursos	Utilización de CPU	>=8seg	<=4seg	0,0556	6,7276		6,672	8	3,6	
		Utilización de la memoria	>1000mb	<=500mb	330,67	814,21		483,54	10		
	Capacidad	Número de peticiones online	0	>=10	180		4	45	10	2	

Fig. 29. Evaluación Eficiencia de Desempeño concurrencia=100

Fuente: Elaboración Propia

Caracter ísitca	Subcaracterís tica	Métrica	Peor caso	Valor Deseado	Variables			Valor Obtenido	Valor Métrica /10	Final subcaracterís tica	Total característica
					A	B	T	X			
Eficiencia de Desempeño	Comportam iento en el tiempo	Tiempo de respuesta	>=50,000 seg	<=25,00 seg	0	28,279		28,279	9	3,733333333	9,19333
		Tiempo de espera	>=40,000 seg	<=20,000 seg	19,69	48,692		29,000	9		
		Rendimiento	0	>=5	3040		40	76	10		
	Utilización de Recursos	Utilización de CPU	>=40seg	<=20seg	0,066	16,48		16,4141	10	3,46	
		Utilización de la memoria	>=1000mb	<=500mb	86,65	1011,58		924,93	7,3		
	Capacidad	Número de peticiones online	0	>=10	3040		40	76	10	2	

Fig. 30.Evaluación Eficiencia de Desempeño concurrencia=2000

Fuente: Elaboración Propia

- **Análisis de los Resultados**

Para proceder con el análisis de la característica Eficiencia de Desempeño, en la TABLA 10, se estableció una escala de medición que aplica los niveles de puntuación y grados de satisfacción propuestas por la Norma ISO/IEC 25040. Con esto se determinó el grado de satisfacción de cada Arquitectura.

TABLA 10 Escala de Medición.

Escala de Medición	Niveles de Puntuación	Grado de Satisfacción
8,00- 10,00	Cumple con los requisitos	Muy Satisfactorio
5,00 – 7,95	Aceptable	Satisfactorio
1,00 - 4,95	Inaceptable	Insatisfactorio

Basándose en las evaluaciones realizadas se obtienen los siguientes puntajes:

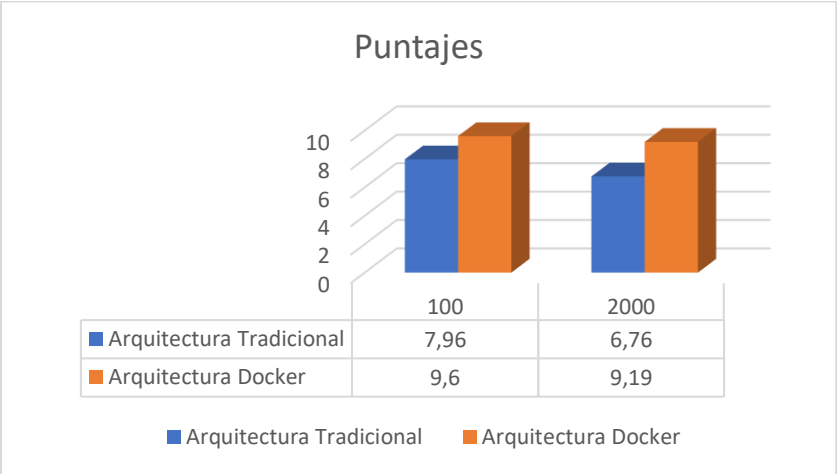


Fig. 31. Resultados de la evaluación de las Arquitecturas

Fuente: Elaboración Propia

Como se pude observar en la Fig. 33, y teniendo como base la escala de medición definida en la TABLA 10, La arquitectura Tradicional tiene un puntaje de 7,96/10, 6,76/10 en los diferentes hilos de concurrencia, a diferencia de la Arquitectura Docker que presenta un puntaje de 9,6/10, 9.19/10 respectivamente. Con esto se concluye que Docker es más eficiente para el despliegue de aplicaciones obteniendo un grado de satisfacción *"Muy Satisfactorio".*

3.3 Análisis de Impactos

Este trabajo promueve el uso y aprovechamiento de nuevas tecnologías que benefician a desarrolladores, tester y administradores de sistemas, en la automatización y monitoreo de tiempos, costos y recursos de infraestructura al implementar aplicaciones dentro de contenedores.

Para realizar el análisis se establecieron tres tipos de impactos que se los detalla a continuación:

- ✓ **Impacto Tecnológico**

Utilizar tecnologías actuales permiten mejorar habilidades y conocimientos para desarrollar, probar, corregir errores y lanzar a producción sistemas óptimos para los usuarios. Docker es una tecnología que permite crear contendores ligeros y portables, enfocándose en el uso mínimo de recursos y tiempos de despliegue de una aplicación. Su facilidad de iniciar, detener, borrar y migrar contenedores es lo que lo diferencia de las máquinas virtuales.

- ✓ **Impacto Ambiental**

Gracias a que Docker permite la creación de varios contenedores en una misma máquina, se logra aprovechar al máximo los recursos que esta provee, sin necesidad de adquirir nueva infraestructura, minimizando el uso de servidores para cada aplicación.

- ✓ **Impacto Económico**

El coste de inversión en infraestructura disminuye con la reducción de máquinas como servidor, debido a que no es necesario crear varias máquinas virtuales que alojan diferentes aplicaciones, sino que se usa los recursos de una sola maquina ejecutando múltiples contenedores, obteniendo los mismos resultados con mayor eficiencia, rendimiento y menores costes.

CONCLUSIONES

La construcción de la base teórica y el ambiente de virtualización con contenedores permitió una correcta ejecución y monitoreo de las aplicaciones desplegadas con Docker, facilitando así el aprendizaje de esta nueva tecnología.

La implementación de la aplicación con Docker en menor tiempo de ejecución y sin conflictos de versiones, demostró la eficiencia y facilidad de trabajo en la creación de diferentes contenedores que comparten los mismos recursos del sistema permitiendo el uso de diferentes aplicaciones.

La gestión y ejecución de los contenedores a través de sentencias establecidas en un mismo archivo de configuración YML, simplificó el uso de Docker al enlazar los contenedores de PostgreSQL y Wildfly con sus respectivas configuraciones, demostrando con ello que la aplicación funciona igual y con mejor rendimiento que la virtualización tradicional.

Para la evaluación de la Eficiencia de Desempeño establecida en la ISO/IEC 25010, se realizó una comparativa con la virtualización tradicional, analizando los tiempos de ejecución y el uso de recursos establecidas en los siguientes puntos:

- Se ve un ahorro de CPU y memoria
- Los tiempos de espera y respuesta se reducen considerablemente; la virtualización tradicional dobla a la de Docker.
- En cuanto a las peticiones se ve que Docker permite un mayor número.

Con esto resultados se demuestra que Docker presenta más eficiencia y requiere de menos recursos en el despliegue de aplicaciones.

La evaluación de las arquitecturas con la herramienta J-Meter, estableciendo diferentes grupos de hilos demostró que la arquitectura tradicional trabaja perfectamente hasta los 1500 hilos y comienza a presentar fallas de acceso y conexión a partir del número de hilos=1900, a diferencia de Docker que presenta eficiencia hasta los 2200 hilos.

RECOMENDACIONES

Hacer uso de la documentación establecida en Docker Hub, allí se encuentran las versiones que existen de cada imagen con las instrucciones necesarias para su ejecución, teniendo en cuenta las rutas para crear los volúmenes, las variables de entorno, los scripts de inicialización.

Conocer los complementos, librerías y otras dependencias requeridas al implementar una aplicación con Docker, con el fin de evitar posibles conflictos en la comunicación de contenedores, evitando el mal funcionamiento.

Una imagen debe ser lo más pequeña posible, para eso, se debe tener en cuenta al momento de crear el Dockerfile, que entre más sentencias deba ejecutar, más peso genera. Es necesario concatenar las sentencias que sean posibles, creando solo una capa de cache ocupando menos espacio.

Para trabajos a futuro se puede implementar Kubernetes como motor de orquestación, que permite gestionar una gran cantidad de contenedores Docker, generando facilidad y eficacia de implementación a escala, para grandes cagas de trabajo.

BIBLIOGRAFÍA

Acuña, P. (2016). Deploying Rails with Docker, Kubernetes and ECS. In *Deploying Rails with Docker, Kubernetes and ECS*. https://doi.org/10.1007/978-1-4842-2415-1

Al-Debagy, O., & Martinek, P. (2018). A Comparative Review of Microservices and Monolithic Architectures. *18th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2018 - Proceedings*, 149–154. https://doi.org/10.1109/CINTI.2018.8928192

Alemandi, M. I., & Jara, O. (2014). *Un Driver de Disco Virtual Tolerante a Fallos*. (0342), 94–104.

Alvaro. (2018). Kubernetes vs Docker: ¿En que se diferencian? - Guiadev. Retrieved March 8, 2020, from https://guiadev.com/kubernetes-vs-docker/

Andre, R. (2020). *Docker de Principiante a Experto*. Retrieved from https://www.udemy.com/course/docker-de-principiante-a-experto/

Arboleda Cola Carlos Augusto. (2017). *Propuesta Metodológica Para Migración De Sistemas Web Con Arquitectura Monolítica Hacia Una Arquitectura Basada En Microservicios* (Escuela Politécnica Nacional). Retrieved from file:///C:/Users/miguel/Downloads/CD-8386.pdf

Arsys. (2017, March 8). WildFly, el servidor de aplicaciones Java que multiplica su rendimiento en Cloud - Blog de arsys.es. Retrieved October 5, 2020, from Cloud website: https://www.arsys.es/blog/programacion/wildfly-cloud/

Azure, M. (n.d.). ¿Qué es una máquina virtual y cómo funciona? Retrieved March 3, 2020, from https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/

Bachiega, N. G., Souza, P. S. L., Bruschi, S. M., Souza, S. R. S. De, Trabalhador, S., & Trabalhador, S. (2018). Container-based Performance Evaluation : A Survey and Challenges. *2018 IEEE International Conference on Cloud Engineering*, 6. https://doi.org/10.1109/IC2E.2018.00075

Bhardwaj, A., & Krishna, C. R. (2019). A Container-Based Technique to Improve Virtual Machine Migration in Cloud Computing. *IETE Journal of Research*, *0*(0), 1–16. https://doi.org/10.1080/03772063.2019.1605848

Biradar, S. M., Shekhar, R., & Reddy, A. P. (2019). Build Minimal Docker Container Using Golang. *Proceedings of the 2nd International Conference on Intelligent Computing and*

Control Systems, ICICCS 2018, (Iciccs), 999–1002. https://doi.org/10.1109/ICCONS.2018.8663172

CentOS. (2020). CentOS. Retrieved July 6, 2020, from 2020 website: https://www.centos.org/about/#centos-linux

Chiza., A. F. S. (2018). *"Análisis De Rendimiento Entre Una Arquitectura Monolítica Y Una Arquitectura De Microservicios – Tecnología Basada En Contenedores"* (Vol. 10). UNIVERSIDAD TÉCNICA DEL NORTE.

Dalgleish, T., Williams, J. M. G. ., Golden, A.-M. J., Perkins, N., Barrett, L. F., Barnard, P. J., … Watkins, E. (2016). Microservices-Flexible-Software-Architecture. In *Journal of Experimental Psychology: General* (Vol. 136). Estados Unidos.

Docker Inc. (2020a). Comience con Docker. Retrieved July 7, 2020, from https://www.docker.com/

Docker Inc. (2020b). Descripción general de Docker | Documentación de Docker. Retrieved July 7, 2020, from https://docs.docker.com/get-started/overview/

Dörterler, S., Dörterler, M., & Ozdemir, S. (2017). *Multi-Objective Virtual Machine Placement Optimization for Cloud Computing*. 6.

Hale, J. S., Li, L., N., C., & Wells, R. and G. N. (2017). *Containers for Portable, Productive, and Performant Scientific Computing*. (December). https://doi.org/10.1109/MCSE.2017.2421459

Hasan, A. (2019). Docker: Componentes (Cliente, Host, Daemon, etc.) - Knoldus Blogs Docker. Retrieved July 7, 2020, from knóldus website: https://blog.knoldus.com/docker-components/

Hat, R. (2020). ¿Qué es Kubernetes? Retrieved March 8, 2020, from https://www.redhat.com/es/topics/containers/what-is-kubernetes

ISO25000. (2019). ISO 25000 Calidad de software. Retrieved March 7, 2020, from iso25000.com

Javier, G. (2015). Entendiendo Docker. Conceptos básicos: Imágenes, Contenedores, Links… - Javier Garzas. Retrieved July 7, 2020, from JavierGarzas.com website: https://www.javiergarzas.com/2015/07/entendiendo-docker.html

Kanbanize. (n.d.). Qué es Kanban: Definición, Características y Ventajas. Retrieved October 15, 2020, from https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban

Kovács, Á. (2017). *Comparison of Different Linux Containers*. 47–51.

Larriba, L. M. (2016). Metodología Kanban: ventajas y características. Retrieved October 15, 2020, from Metodología Kanban website: https://www.getbillage.com/es/blog/metodologia-kanban-ventajas-y-caracteristicas

Llaven, D. S. (2015). *Sistemas operativos Panorama para la Ingeniería en Computación e Informática* (J. E. Callejas, Ed.). México: Editorial Patria.

Lopez, J. (2017). *Universidad Técnica del Norte Instituto de Postgrado*. Universidad Técnica del Norte.

Maenhaut, P. J., Volckaert, B., Ongenae, V., & De Turck, F. (2019). Resource Management in a Containerized Cloud: Status and Challenges. In *Journal of Network and Systems Management*. https://doi.org/10.1007/s10922-019-09504-0

Martínez, D. R. P. V. V. T. B. (2018). *Microservicios un Enfoque Integrado* (RA-MA). Madrid.

Maya, E., & López, D. (2018). *Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web*. (November).

Mesh, J. (n.d.). Metodología Kanban: revoluciona tu manera de trabajar más ágil. Retrieved October 15, 2020, from https://blog.trello.com/es/metodologia-kanban

Millán-Rojas, E. E., Gallego-Torres, A. P., & Chico-Vargas, D. C. (2016). Simulación de una red Grid con máquinas virtuales para crear un entorno de aprendizaje de la computación de alto desempeño TT - Simulation of a Grid network with virtual machines to create a learning environment of high performance computing TT - Simula. *Revista Facultad de Ingeniería*, *25*(41), 85–92. Retrieved from http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0121-11292016000100009&lang=es%0Ahttp://www.scielo.org.co/pdf/rfing/v25n41/v25n41a09.pdf

Morán, G., Paul, J., Fernando, P., & Ángel, R. A. (2013). *Instala, administra, securiza y virtualiza entornos Linux* (2 edición). Madrid: RA-MA, S.A. Editorial y Publicaciones.

Moya, D. (2018). Contenedores docker la mejor guía. Retrieved March 7, 2020, from Diciembre 19 website: https://www.disenowebwordpress.com/contenedores-docker-la-mejor-guia/

Pérez, J. C. M., & Pérez, A. F. R. (2014). *Sistemas Operativos y Aplicaciones Informáticas*. Madrid: Editorial RA-MA.

PostgreSQL. (2020). PostgreSQL: la base de datos de código abierto más avanzada del mundo. Retrieved September 10, 2020, from https://www.postgresql.org/

Prometheus. (2014). Overview | Prometheus. Retrieved September 16, 2020, from 2014-2020 website: https://prometheus.io/docs/introduction/overview/

Proxmox. (2020). PROXMOX. Retrieved July 6, 2020, from 2004-2020 website: https://www.proxmox.com/en/

PuTTY. (n.d.). Descargue PuTTY, un cliente SSH y telnet gratuito para Windows. Retrieved September 10, 2020, from https://putty.org/

Raj, P., Chelladhurai, J. S., & Singh, V. (2015). Learning Docker. In *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Red Hat. (n.d.). WildFly. Retrieved September 10, 2020, from https://www.wildfly.org/

Rubiano, G. A. (2015). *Virtualización*.

Rudyy, O. (2019). *Viabilidad Del Uso De Contenedores En Entornos HPC 1*. 94.

Schutt, K., & Balci, O. (2016). Cloud software development platforms: A comparative overview. *2016 IEEE/ACIS 14th International Conference on Software Engineering Research, Management and Applications, SERA 2016*, 3–13. https://doi.org/10.1109/SERA.2016.7516122

Sierra, J. C. (2015). *Java Interfaces gráficas y aplicaciones para Internet 4. ° Edición* (RA-MA, S.A). Madrid: Editorial RA-MA.

SIMBAÑA, M. (2016). *ESTUDIO DEL CONTENEDOR CLOUD DOCKER Y PROPUESTA DE IMPLEMENTACIÓN PARA LA PLATAFORMA CLOUD FICA*. Universidad técnica del norte.

Tomás, E. (2019). ¿Qué es Kubernetes y cómo funciona? Retrieved from Campus MVP website: https://www.campusmvp.es/recursos/post/que-es-kubernetes-y-como-funciona.aspx

Unidas, N. (2016). *La Agenda 2030 y los Objetivos de Desarrollo Sostenible: una oportunidad para América Latina y el Caribe*. 93.

Van De Belt, J., Ahmadi, H., & Doyle, L. E. (2017). Defining and Surveying Wireless Link Virtualization and Wireless Network Virtualization. *IEEE Communications Surveys and Tutorials*, *19*(3), 1603–1627. https://doi.org/10.1109/COMST.2017.2704899

Víctor Cuervo. (2019, December 3). Objetos Docker – Arquitecto IT. Retrieved July 7, 2020,

from AiT website: http://www.arquitectoit.com/docker/objetos-docker/

Vivas, A. (2019). *Director :* Universidad Técnica del Norte.

Younge, A. J., Pedretti, K., Grant, R. E., & Brightwell, R. (2017). *A Tale of Two Systems : Using Containers to Deploy HPC Applications on Supercomputers and Clouds*. 74–81. https://doi.org/10.1109/CloudCom.2017.40

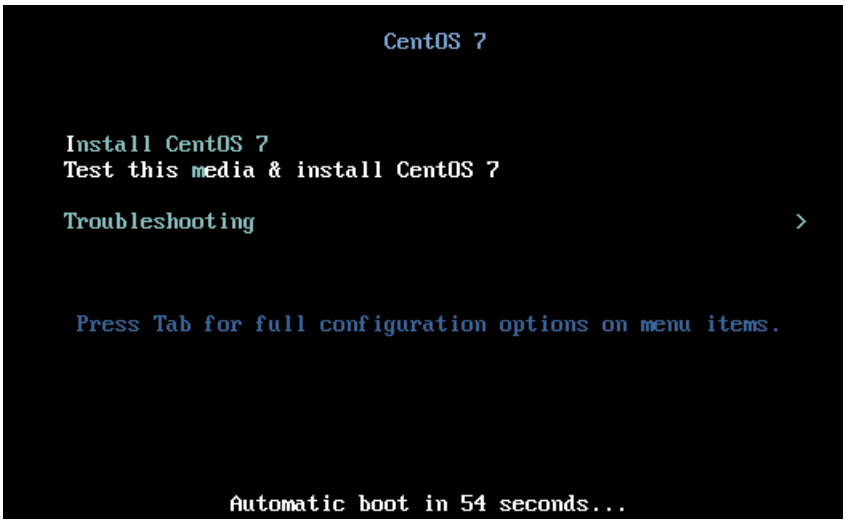
Zadka, M. (2019). DevOps in Python. In *DevOps in Python*. https://doi.org/10.1007/978-1-4842-4433-3

ANEXOS

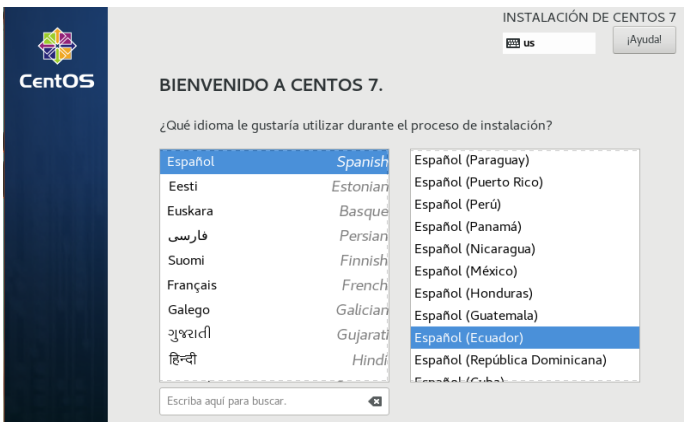
Anexo A: Instalación de Sistema Operativo CentOS7.

Descargar imagen ISO de la página oficial de CentOS**:** https://www.centos.org/download/

Cargar la ISO anteriormente descargada de CentOS 7

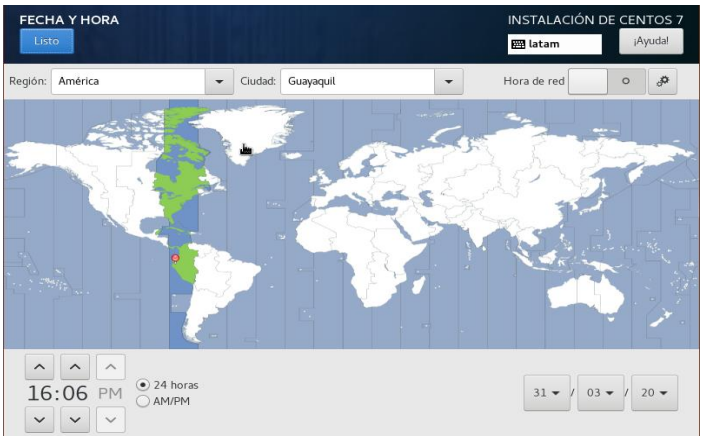


Al terminar la carga del sistema, se genera la pantalla de ayuda de instalación, aquí se define el idioma y la distribución de teclado.



Clic en continuar y la siguiente pantalla se muestra algunas configuraciones para la instalación del sistema operativo.

Definir la zona horaria en la opción de Fecha & Hora

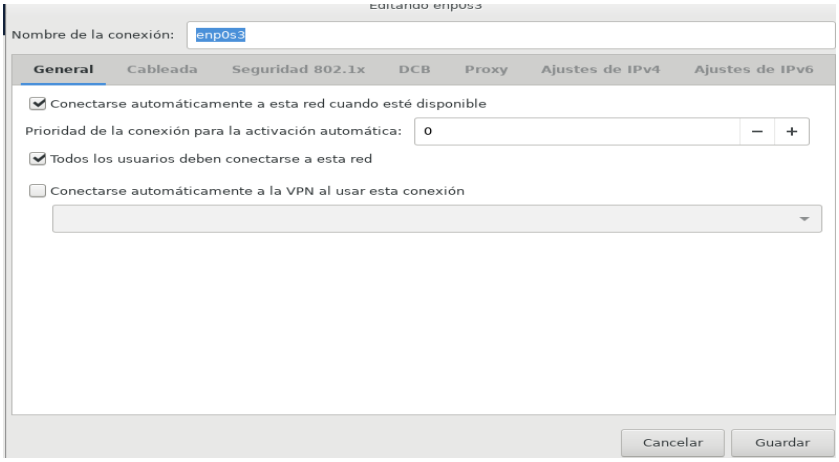


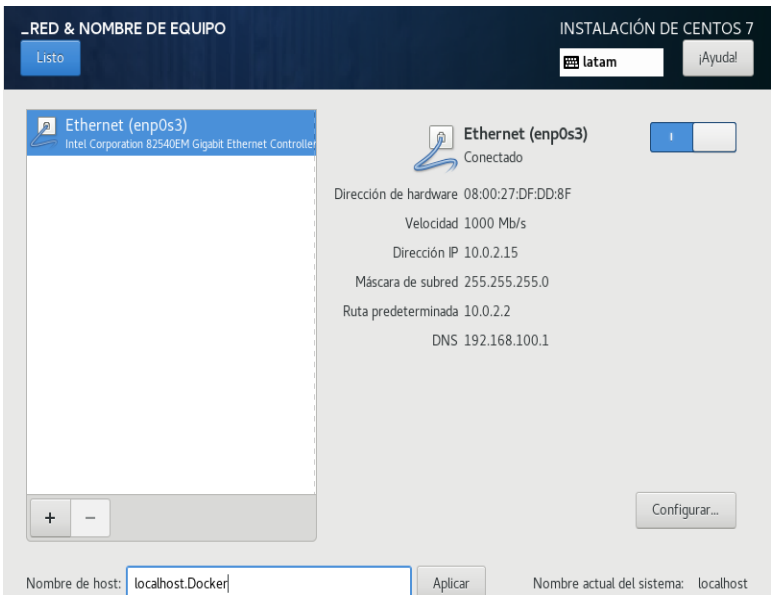
Debido a que el SO va a ser usado como servidor web, en la opción de ***Selección de Software*** se deja como instalación mínima.

En la opción ***Destino de la Instalación***, seleccionar donde se instalará el SO en el equipo.

Configurar la conexión de red en la opción ***Red y Nombre de Equipo***:

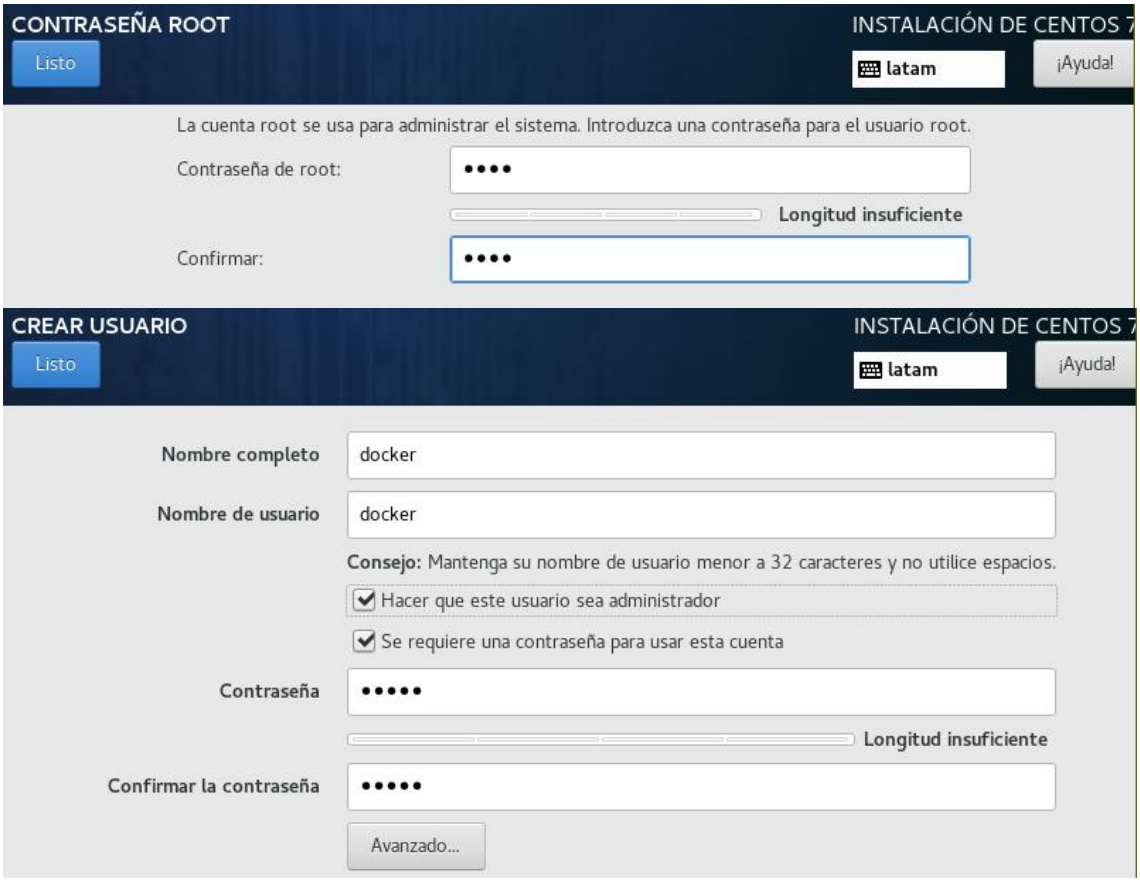
- Activar la opción de Ethernet
- En la pestaña General, activar la casilla Conectarse automáticamente
- Escribir el nombre del host y aplicar





Una vez finalizado el proceso de configuración con cada una de las opciones, dar clic en empezar instalación.

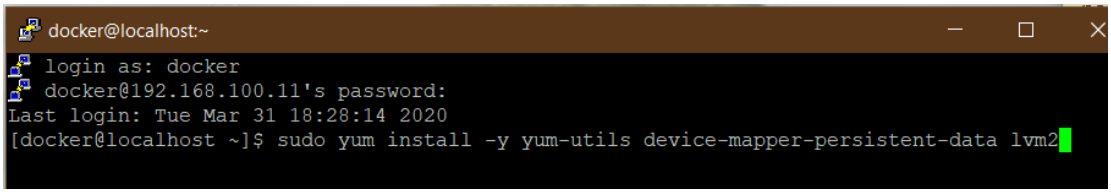
Para finalizar la instalación se debe definir la contraseña root y un usuario para el sistema.



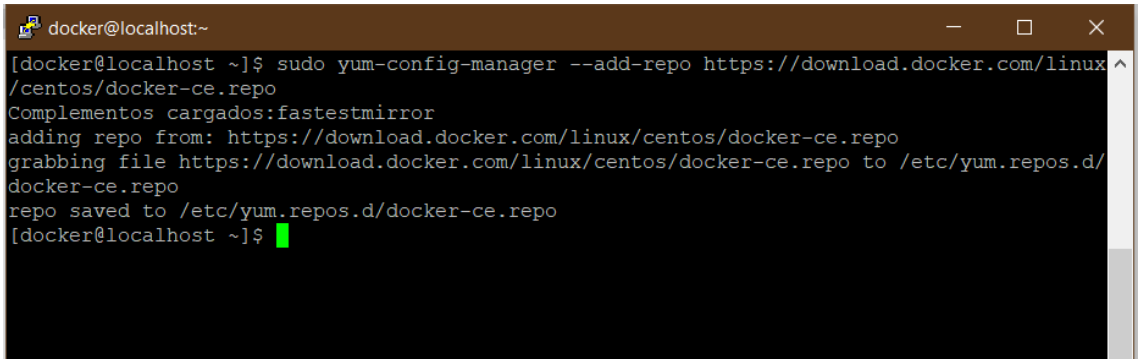
Esperar mientras termina de instalar todos los paquetes y configuraciones, como último paso reiniciar el equipo y empezar a usar CentOS 7.

Anexo B: Instalación Docker

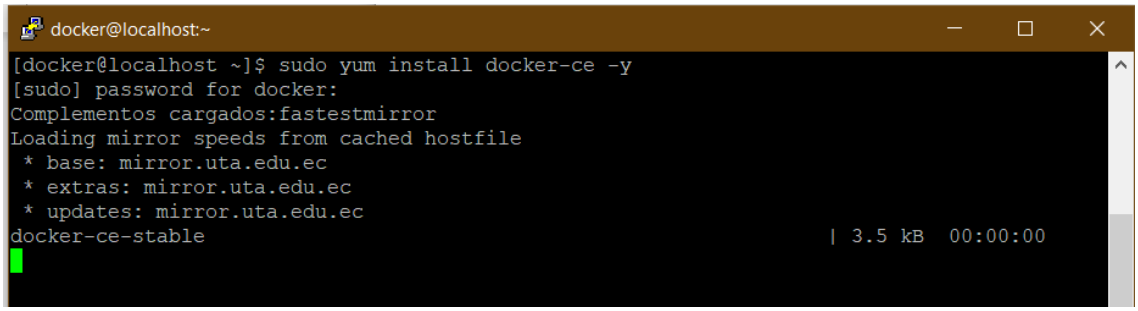
Instalar utilidades
sudo yum install -y yum-utils device-mapper-persistent-data lvm2



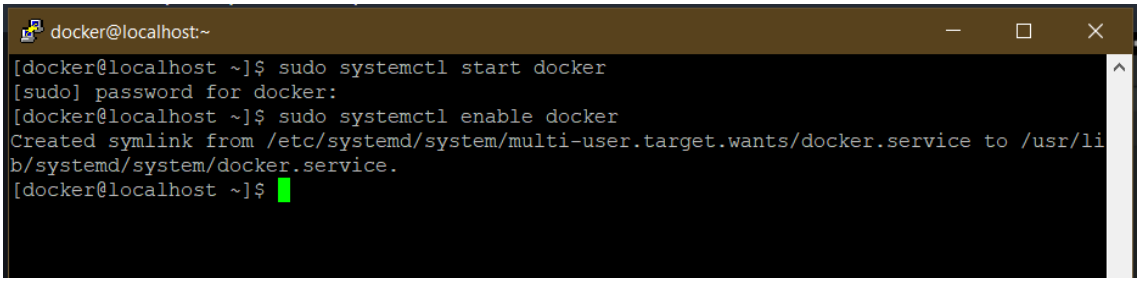
Agregar repositorio de Docker de donde se va a descargar el paquete
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo



Instalar el paquete Docker del repositorio agregado anteriormente
sudo yum install docker-ce -y

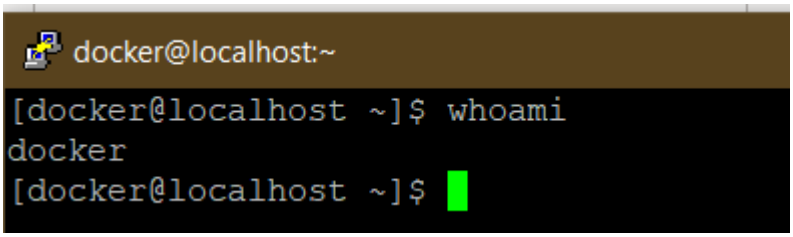


Iniciar y habilitar el servicio Docker

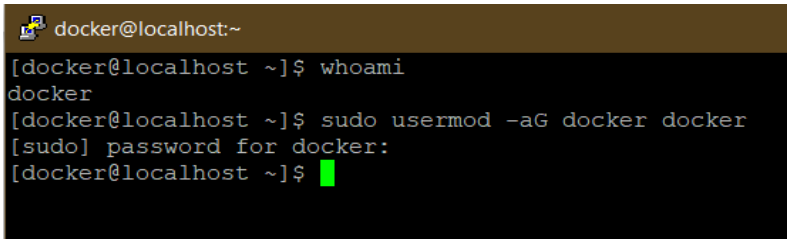


Agregar usuario al grupo Docker para que puede hacer uso de la herramienta

- Ver el usuario de la máquina

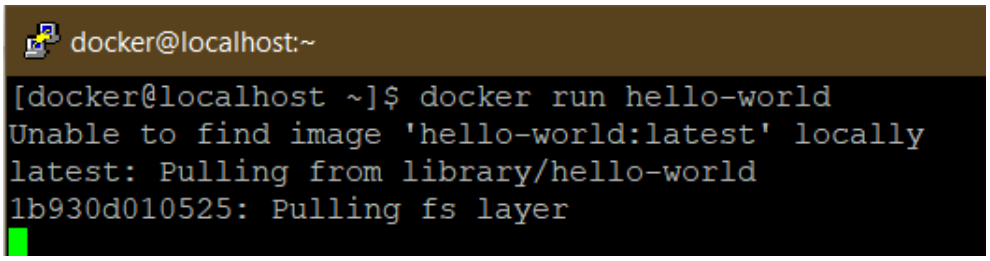


- sudo usermod -aG docker “docker” → nombre de la máquina

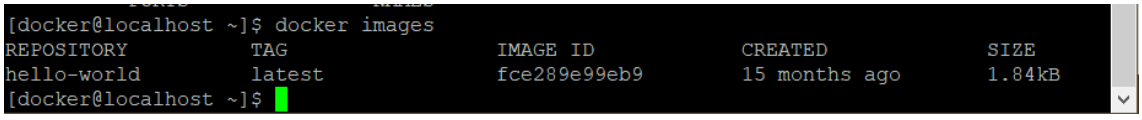


Una vez instalado Docker probar que funcione correctamente corriendo la imagen de prueba de Docker.

- docker run hello-world



- docker images, para observar que se descargó la imagen anterior



Por último, ver la ruta donde Docker tiene todo sus archivos, docker info