

Universidad de Pamplona
Facultad de Ingenierías y Arquitectura
Programa de Ingeniería de Sistemas

Tema:

**DISEÑO DEL PROCESO OPERACIONAL PARA LA IMPLEMENTACIÓN DE
CONTENEDORES EN EL DESPLIEGUE DE APLICACIONES SOBRE LA
INFRAESTRUCTURA TECNOLÓGICA DEL ÁREA DE DESARROLLO DE LA
UNIVERSIDAD DE PAMPLONA**

Autor:

Juan Sebastián Sánchez Parada

Pamplona, Norte De Santander

Diciembre 2020

Universidad De Pamplona
Facultad De Ingenierías y Arquitectura
Programa De Ingeniería De Sistemas

Trabajo de grado presentado para optar al título de Ingeniero de Sistemas.

Tema:

**DISEÑO DEL PROCESO OPERACIONAL PARA LA IMPLEMENTACIÓN DE
CONTENEDORES EN EL DESPLIEGUE DE APLICACIONES SOBRE LA
INFRAESTRUCTURA TECNOLÓGICA DEL ÁREA DE DESARROLLO DE LA
UNIVERSIDAD DE PAMPLONA.**

Autor:

Juan Sebastián Sánchez Parada

Director:

Elvis Navarro Vega

Especialista en Gestión de Proyectos Informáticos.

Pamplona, Norte de Santander.

Diciembre 2020.

Resumen

El despliegue de software es un proceso, que, en algunos casos, puede ser muy repetitivo y con altas probabilidades de cometer errores por intervención humana, debido a que en la mayoría de los casos se realizan de forma manual. Al momento de realizar un despliegue de aplicación se deben ejecutar todos los elementos involucrados para que este sea satisfactorio. La automatización establece la operación controlada automáticamente de un aparato, proceso o sistemas mediante un dispositivo que toma el lugar del ser humano (Yeja & Rubier, 2016), disminuyendo las probabilidades de cometer errores producto de las operaciones sobre los sistemas, produciendo un ahorro de tiempo en el despliegue de procesos. Este trabajo consiste en analizar y determinar los distintos tipos de contenedores y de plataformas de orquestación de código abierto más utilizadas en la actualidad, para diseñar un documento que indique cuál de estos es el más adecuado para la automatización en los procesos de despliegue de aplicaciones sobre la infraestructura tecnológica del área de desarrollo de la Universidad de Pamplona.

Abstract

Software deployment is a process that, in some cases, can be very repetitive and with a high probability of making mistakes due to human intervention, since in most cases they are done manually. When carrying out an application deployment, all the elements involved must be executed for it to be satisfactory. Automation establishes the automatically controlled operation of a device, process or systems by means of a device that takes the place of the human being (Yeja & Rubier, 2016), reducing the probability of making errors as a result of operations on the systems, producing savings of time in the deployment of processes. This work consists of analyzing and determining the different types of containers and open source orchestration platforms most used today, to design a document that indicates which of these is the most appropriate for automation in the deployment of applications on the technological infrastructure of the development area of the University of Pamplona.

Tabla de contenidos

1	Descripción del proyecto	13
1.1	Planteamiento del problema.	13
1.2	Justificación	14
1.3	Delimitación	15
	• Objetivo General:	15
	• Objetivos Específicos	15
1.4	Acotaciones	16
1.5	Metodología	17
2	Marco teórico y estado del arte	18
2.1	Marco conceptual	18
2.4	Estado del arte	26
2.4.1	Internacional:	26
2.4.2	Nacional:	26
2.4.3	Regional:	28
3	Análisis preliminar.	29
3.1	Contenedores de software.	29
3.1.1	Docker.	29

3.1.2	Rkt.	36
3.1.3	LXC.	38
3.2	Motores de orquestación.	41
3.2.1	Kubernetes.	43
3.2.2	Docker Swarm.	48
3.2.3	Nomad.	50
4	Definición base.	52
4.1	Selección de herramienta de empaquetado de software.	52
4.1.1	Identificar las ventajas y desventajas de cada tipo.	53
4.2	Selección de herramienta de orquestación de contenedores.	55
5	Diseño	59
5.1	Términos para tener en cuenta.	59
5.1.1	Imagen.	59
5.1.2	Dockerfile.	59
5.1.2.1	Estructura de un Dockerfile.	60
5.2	Proceso	61
5.2.3	Adquisición de imagen.	62
5.2.4	Ejecución.	62

5.2.5	Conexión.	63
5.2.6	Modificación de los contenedores.	65
5.2.7	Conexión entre contenedores por red.	65
5.2.8	Clonar un contenedor.	66
5.2.9	Portabilidad.	67
5.2.10	Orquestación.	68
6	Validación.	72
6.1	Adquisición de imagen.	72
6.2	Ejecución.	74
6.3	Conexión.	74
6.4	Modificación de los contenedores.	75
6.5	Verificación de funcionamiento.	77
6.6	Conexión entre contenedores por red.	79
6.7	Clonar un contenedor.	81
6.8	Portabilidad.	83
6.9	Orquestación.	84
7	Conclusiones	91
8	Recomendaciones y trabajos futuros	92

9 Bibliografía.

93

Tabla de Figuras

Ilustración 1. Arquitectura máquinas virtuales (Dockercon17, 2017).	19
Ilustración 2. Arquitectura contenedores (Dockercon17, 2017).	20
Ilustración 3. Motor de Docker. (Docker docs, 2020).	31
Ilustración 4. Arquitectura de Docker (Docker docs, 2020).	32
Ilustración 5. Arquitectura Kubernetes (RedHat, 2020).	44
Ilustración 6. Arquitectura Docker Swarm (Hernández A. , 2019).	49
Ilustración 7. Arquitectura Nomad (Nomad, 2017).	51
Ilustración 8. Estructura Dockerfile (elaboración propia).	60
Ilustración 9. Flujo de construcción de un contenedor (Goig, 2016).	61
Ilustración 10. Docker images (elaboración propia).	62
Ilustración 11. Contenedores en ejecución (elaboración propia).	64
Ilustración 12. Redes creadas por Docker(elaboración propia).	66
Ilustración 13. Estructura de pods (Melendez, 2019).	69
Ilustración 14. Estructura Deployments (Melendez, 2019).	70
Ilustración 15. Estructura Service (Melendez, 2019).	71
Ilustración 16 Representación gráfica de despliegue de contenedores (elaboración propia).	71
Ilustración 17. Docker Hub (DockerHub, 2020).	72
Ilustración 18. Imagen oficial de MySQL de Docker Hub (DockerHub, 2020).	73
Ilustración 19. Imagen oficial de PHP (DockerHub, 2020).	73

Ilustración 20. Docker images (elaboración propia).	74
Ilustración 21. Verificación del funcionamiento de los contenedores (elaboración propia).	75
Ilustración 22. Conexión base de datos (elaboración propia).	76
Ilustración 23. Verificación de funcionamiento de los contenedores (elaboración propia).	77
Ilustración 24. Contenido control de usuarios (elaboración propia)	77
Ilustración 25. Registro de usuarios (elaboración propia).	78
Ilustración 26. Estructura de desarrollo (elaboración propia)	78
Ilustración 27. Nueva network (elaboración propia).	79
Ilustración 28. Inspección de red (elaboración propia).	79
Ilustración 29. Inspección network conectado a contenedores (elaboración propia).	80
Ilustración 30. Verificación de conexión de red (elaboración propia)	81
Ilustración 31. Estructura de desarrollo masivo (elaboración propia)	82
Ilustración 32. Repositorio de Docker Hub (DockerHub, 2020).	83
Ilustración 33. Lista de servicios (elaboración propia).	84
Ilustración 34. Dashboard kubernetes (elaboración propia).	85
Ilustración 35. Archivo YAML (elaboración propia).	86
Ilustración 36. Pod creado (elaboración propia).	87
Ilustración 37. Estructura deployment (elaboración propia)	87
Ilustración 38. Deployment run (elaboración propia).	88

Ilustración 39. Sustitución de un pod (elaboración propia).	89
Ilustración 40. Dashboard kubernetes (elaboración propia).	89
Ilustración 41. Estructura servicio (elaboración propia).	90

Tablas

Tabla 1. Ventajas y desventajas de Docker (elaboración propia).	53
Tabla 2. Ventajas y desventajas de rkt (elaboración propia).	54
Tabla 3. Ventajas y desventajas de LXC (elaboración propia).	54
Tabla 4. Ventajas y desventajas de Kubernetes (elaboración propia).	56
Tabla 5. Ventajas y desventajas de Docker Swarm (elaboración propia).	56
Tabla 6. Ventajas y desventajas de Nomad (elaboración propia).	57

1 Descripción del proyecto

1.1 Planteamiento del problema.

La Universidad de Pamplona ofrece servicios de desarrollo de software, “despliegue de software suele ser un proceso profundo, en algunos casos, reiterativo y con cuantiosas posibilidades de cometer errores, debido a que mayormente, estos casos se realizan de forma manual, ya que si no se practican todos los elementos involucrados en el despliegue de forma precisa la aplicación no se desempeñará de manera satisfactoria” (Yeja & Rubier, 2016).

Una problemática que tiene en estos momentos el CIADTI, es la falta de procedimientos automatizados en las actividades de despliegue de aplicaciones, lo cual consume tiempo y puede provocar errores humanos. Una de las mejores prácticas de desarrollo de software lo constituye la Integración Continua, la cual permite la reducción de riesgos y tareas repetitivas, ejecución de pruebas, generación de software listo para desplegar e incrementar la calidad, lo que permite el monitoreo de sistemas de control de versiones.

De esta manera, con este trabajo se pretende resolver la siguiente pregunta: ¿De qué forma se puede automatizar el despliegue de aplicaciones sobre la infraestructura tecnológica del área de desarrollo de la Universidad de Pamplona?

1.2 Justificación

Al realizar el proceso de automatización este proporciona unos beneficios, uno de estos es rendimiento, ya que se reducen los tiempos de ejecución en el despliegue de aplicativos en el departamento de producción. Para realizar el despliegue de una aplicación, en ocasiones se necesita ejecutar máquinas virtuales tradicionales, las cuales consumen demasiados recursos, solo para que levanten ciertos servicios que el aplicativo necesita para que esta se ejecute de manera satisfactoria, al automatizar procesos, se optimiza el número de actividades que se realizan, evitando errores humanos y reduciendo los costos operativos, esto por medio de contenedores, los cuales son entendidos como sistemas empaquetados orientados a micro servicios y gestionados dinámicamente. La gran ventaja de los Contenedores es que permiten que una aplicación y sus dependencias sean empaquetadas y operadas con menos recursos que las instancias de máquinas virtuales, además que puede ejecutarse en cualquier servidor. Asimismo, permite la flexibilidad y portabilidad en dónde la aplicación se puede ejecutar, ya sea en las instalaciones físicas, en la nube pública o en una nube privada.

1.3 Delimitación

- **Objetivo General:**
 - Diseñar un proceso operacional para la implementación de contenedores en el despliegue de aplicaciones sobre la infraestructura tecnológica del área de desarrollo de la Universidad de Pamplona.
- **Objetivos Específicos**
 - Elaborar marco de referencia de contenedores para el despliegue de aplicaciones y de motores de orquestación.
 - Definir y determinar un contenedor para el proceso de despliegue de aplicaciones creadas por área de desarrollo de la Universidad de Pamplona.
 - Validar el contenedor seleccionado a través de un prototipo.

1.4 Acotaciones

El presente trabajo empleará solamente herramientas de software que sean de código libre (Open Source), el análisis comparativo de los diferentes tipos de contenedores y motores de orquestación se realizarán teóricamente, es decir, basado en investigaciones y artículos, también se limitará la implementación del prototipo a un recurso y un servicio de la Universidad de Pamplona.

1.5 Metodología

Para el desarrollo de este proyecto se empleó una metodología con un enfoque de investigación aplicada, dado que busca resolver problemas prácticos donde se exploraron y realizaron análisis sobre los diferentes tipos de contenedores y orquestadores de código libre más comunes en la actualidad para lograr el cumplimiento de los objetivos de este proyecto.

El proyecto se segmentó en tres fases: La primera fase se basó en explorar, investigar y dar una apertura teórica al tema, se exploraron antecedentes de los diferentes tipos de contenedores y motores de orquestación de código libre más comunes, además, una comparativa entre estos diferentes tipos de contenedores y orquestadores. Posteriormente, en la segunda fase con la información recopilada se realizó una toma de decisiones para la cual consistió en la selección de un tipo de contenedor y un tipo de orquestador, además, de diseñar un proceso operacional para el despliegue de aplicaciones de la universidad de Pamplona y continuar con la tercera, básicamente esta última fase consiste en la construcción de un proceso operacional y de un prototipo con base a los elementos teóricos y herramientas seleccionadas en las fases anteriores.

También se establece que la recopilación de información para este proyecto es basada en fuentes de información como artículos, investigaciones, libros e internet.

2 Marco teórico y estado del arte

2.1 Marco conceptual

2.1.1 Contenedores:

Los contenedores son una unidad de software estándar que empaqueta código para que la aplicación se desarrolle y ejecute de una forma rápida y segura. También permite su paso de un entorno informático a otro, con gran agilidad. Esto hace que los contenedores sean la solución ideal para solventar el problema que hay a la hora de hacer que el software funcione de manera segura cuando se mueve de un entorno a otro. A la hora de desplegar aplicaciones, el desarrollador puede mover ese software desde su propio ordenador hasta un entorno de prueba, o uno de producción, incluso más allá, hasta un entorno de nube pública o privada (MuyCanal, 2020).

2.1.2 Ventajas del uso de contenedores:

Al utilizar contenedores se tienen grandes beneficios, ya que estos consumen menos recursos que las máquinas virtuales tradicionales. Al momento de realizar despliegues de aplicaciones, el servidor ejecuta estas en contenedores, ejecuta un solo sistema operativo y cada contenedor comparte el núcleo del sistema operativo con los otros contenedores. Las partes compartidas del sistema operativo son solo de lectura, mientras que cada contenedor tiene su propio soporte para escribir. Esto significa que usan menos recursos que las máquinas virtuales.

2.1.2.1 Contenedor Vs Máquina virtual tradicional:

La arquitectura de Docker es diferente al de las máquinas virtuales tradicionales, la portabilidad es el punto más importante de su uso ya que son más fáciles de transportar. Como se puede observar en la ilustración 1 (Docker, 2017), está la arquitectura de una máquina virtual tradicional donde se aloja toda la virtualización en el sistema operativo invitado donde se alojan las diferentes librerías y aplicaciones, con lo cual se tendría que disponer de un gran tamaño de almacenamiento en la máquina anfitriona.

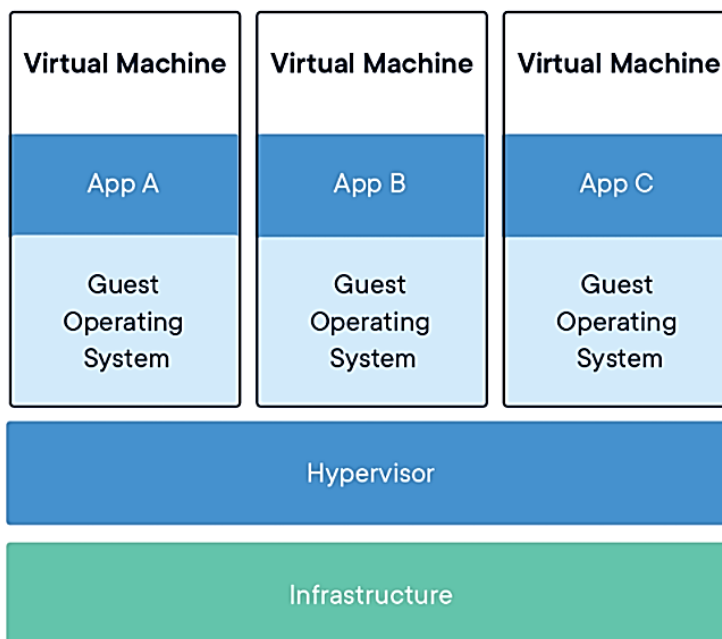


Ilustración 1. Arquitectura máquinas virtuales (Docker, 2017).

A diferencia de las máquinas virtuales, los contenedores de Docker pueden ser vistos como herramientas más flexibles para el empaquetamiento, entrega y despliegue del software y de sus aplicaciones. La arquitectura de un contenedor de Docker, como se puede observar

en la ilustración 2 (Docker, 2017), está conformado por las librerías y la aplicación con su respectiva dependencia, pero a diferencia de la máquina virtual, todos los contenedores comparten el mismo Kernel de la máquina anfitriona. Cada uno de los contenedores posee su propio espacio de usuario en el sistema operativo anfitrión y se ejecutan en cualquier ordenador, en cualquier infraestructura y en cualquier nube que tenga desplegado el motor de Docker Engine (Docker, 2017)

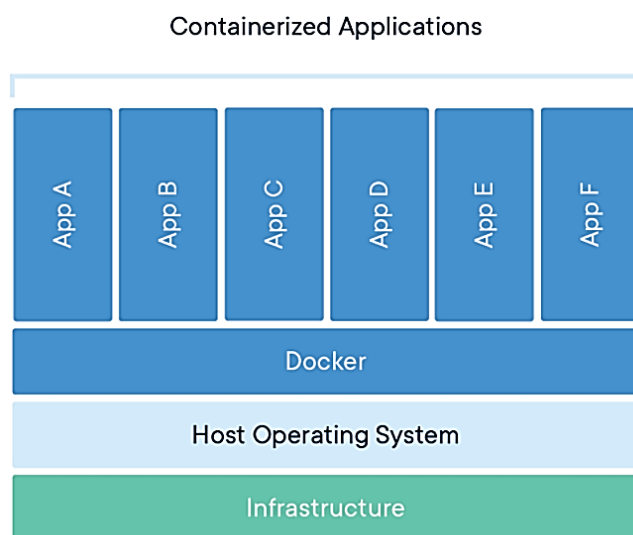


Ilustración 2. Arquitectura contenedores (Docker, 2017).

2.1.3 Orquestadores:

Los orquestadores o motor de orquestación de contenedores son herramientas o sistemas que automatizan el despliegue, la gestión, la interconexión, la disponibilidad y agrupan a los contenedores que componen una aplicación.

2.1.4 Despliegue de software:

El despliegue de software consiste en varias actividades que permiten que un software o

aplicación este listo para su uso. Los ambientes involucrados en el ciclo de vida de desarrollo de software: Antes de llegar a producción una aplicación debe pasar primero por distintos ambientes que certifiquen la calidad de software, los ambientes mas típicos por los que pasa una aplicación son el ambiente de desarrollo, el ambiente de test, el ambiente de aceptación por parte de los usuarios, el ambiente de pre producción y el ambiente de producción. Según las necesidades de cada organización se dará el caso en el que algunos de estos ambientes no sea necesario (Alex, 2015).

2.1.4.1 Ambiente de desarrollo:

El ambiente de desarrollo es donde se crea el software y generalmente está conformado por los ordenadores de los desarrolladores más los servidores de desarrollo como por ejemplo los de base de datos o cualquier otro servicio (Alex, 2015).

2.1.4.2 Ambiente de test:

En el ambiente de test es donde se prueban las aplicaciones por parte del equipo, es donde se deben detectar todos los bugs o incidencias funcionales y no funcionales, de manera que el software pueda llegar a producción libre de bugs y cumpliendo con todos los requerimientos funcionales (Alex, 2015).

2.1.4.3 Ambiente de aceptación:

En este ambiente un grupo de usuarios prueba el software y se asegura de que funcionalmente cumple con los requisitos y con la lógica de negocio de la empresa. Además, se comprueba que software sea usable y robusto (Alex, 2015).

2.1.4.4 Ambiente de pre producción:

El ambiente de pre producción es el ambiente final antes de poner el software en producción, aquí las aplicaciones son probadas en condiciones de hardware muy parecidas a las de producción (Alex, 2015).

2.1.4.5 Ambiente de producción:

En el ambiente de producción es donde finalmente se despliegan las aplicaciones.

2.1.5 Código abierto:

También conocido como Open Source, “diseñado de manera que sea accesible al público: todos pueden ver, modificar y distribuir el código de la forma que consideren conveniente. El software open source se desarrolla de manera descentralizada y colaborativa, así que depende de la revisión entre compañeros y la producción de la comunidad. Además, suele ser más económico, flexible y duradero que sus alternativas propietarias, ya que las encargadas de su desarrollo son las comunidades y no un solo autor o una sola empresa.” (RedHat, 2017) . Existe una diferencia entre código libre y software libre, suelen ser lo mismo en la mayor parte de los casos con excepciones. En algunos casos se basan principalmente en el acuerdo de licencias de uso y distribución.

2.1.6 Software Libre:

“El movimiento del software libre defiende la libertad de los usuarios de ordenadores, en un movimiento en pro de la libertad y la justicia” (GNU, 2008). En sus principios está el poder copiar, ejecutar y distribuir programas informáticos bajo un acuerdo que de libertad y sin asumir ningún recargo monetario por este.

2.1.7 Arquitectura de software:

Dentro de un proyecto de desarrollo, e independientemente de la metodología que se utilice, se puede hablar de “desarrollo de la arquitectura de software”. Este desarrollo, que precede a la construcción del sistema, está dividido en las siguientes etapas: requerimientos, diseño, documentación y evaluación. Cabe señalar que las actividades relacionadas con el desarrollo de la arquitectura de software generalmente forman parte de las actividades definidas dentro de las metodologías de desarrollo (Cervantes, 2019). A continuación, se describen dichas etapas.

- **Requerimientos:** En esta etapa básicamente se procede a definir los requerimientos funcionales del sistema, esto con el fin de poder definir el manejo de la información, importante para optar por una arquitectura fácilmente escalable (Cervantes, 2019).
- **Diseño:** En esta etapa se buscan las tecnologías más adecuadas según los requerimientos definidos en la primera etapa, se busca que la arquitectura sea fácilmente adaptable al modelo del sistema o software a construir y se definen las funciones y responsabilidades de los componentes con los que va contar este (Cervantes, 2019).
- **Documentación:** La documentación de una arquitectura involucra la representación de varias de sus estructuras que son representadas a través de distintas vistas. Una vista generalmente contiene un diagrama, además de información adicional, que apoya en la comprensión de dicho diagrama

(Cervantes, 2019).

- Evaluación: Dado que la arquitectura de software juega un papel crítico en el desarrollo, es conveniente evaluar el diseño una vez que este ha sido documentado con el fin de identificar posibles problemas y riesgos (Cervantes, 2019).

2.2 Contenedores más comunes:

2.2.1 Docker:

Docker es un proyecto de Open Source, que permite que varias aplicaciones se ejecuten de forma paralela en contenedores, los cuales empaquetan todo lo necesario para que una aplicación funcione de manera satisfactoria como el código, herramientas del sistema y librerías del sistema. Esto garantiza que se puedan ejecutar en cualquier entorno de despliegue sin necesidad de gastar recursos de la máquina.

2.2.2 rkt (rocket):

rkt es un sistema de contenedores desarrollado por CoreOS. Está construido en base a un estándar de contenedor abierto conocido como "App Container" o especificación "appc". Esto permite que las imágenes rkt sean portátiles en muchos sistemas de contenedores que siguen el formato abierto "appc" (IONOS, 2019).

2.2.3 LXC (Linux Containers):

LXC es un conjunto de herramientas, plantillas, bibliotecas y códigos que constituye una *User Space Interface* para las funciones nativas de virtualización con contenedores del

núcleo de Linux. LXC facilita la creación y administración de contenedores para aplicaciones.

2.3 Motores de orquestadores más comunes:

2.3.1 Kubernetes:

Kubernetes es el motor de orquestación de contenedores creado por Google y es el más popular que existe en el mercado. Miles de equipos de desarrolladores usan kubernetes para desplegar contenedores en producción, ejecutando miles de millones de contenedores cada semana. La herramienta funciona agrupando contenedores que componen una aplicación en unidades lógicas para una fácil gestión y descubrimiento.

2.3.2 Docker Swarm:

Docker Swarm es la solución que propone Docker ante los problemas de los desarrolladores a la hora de orquestar y planificar contenedores a través de muchos servidores. Swarm viene incluido junto al motor de Docker, desarrollado para el despliegue de contenedores Docker ofreciendo muchas funciones avanzadas integradas como el descubrimiento de servicios, balanceo de carga, escalado y seguridad.

2.3.3 HashiCorp Nomad:

Nomad es una herramienta binaria única capaz de planificar todas las aplicaciones virtualizadas, en contenedores o independientes. Nomad te da la capacidad de ejecutar, si quisieras, 1 millón de contenedores a través de 5.000 hosts en cuestión de minutos. Nomad

ayuda a mejorar la densidad a la vez que reduce costes, ya que es capaz de distribuir de manera eficiente más aplicaciones en menos servidores.

2.4 Estado del arte

2.4.1 Internacional:

Diseño e implementación de un sistema de entrega continua para aplicaciones web sobre contenedores Docker. “El objetivo principal de este proyecto de fin de grado es reducir el tiempo necesario para desplegar cualquier aplicación desarrollada en BiiT Sourcing Solutions en un entorno de producción. Todo ello con total garantía de que el software generado cumplirá con los estándares mínimos de calidad. Para lograrlo, hemos implementado un sistema de entrega continua, donde una vez que el equipo de desarrollo ha finalizado una nueva versión de la aplicación, se realizan automáticamente todas las pruebas necesarias asegurando el correcto funcionamiento del software. Además de esto, las soluciones propuestas se basan en la virtualización utilizando contenedores Docker para asegurar la portabilidad y la optimización de los recursos de la empresa.” (Morales, 2015). En este informe el autor del libro recopiló información sobre la portabilidad y optimización que aportan los contenedores a la hora de realizar despliegue de aplicaciones.

2.4.2 Nacional:

Diseño e implementación del sistema de gestión de entornos para la oficina asesora de sistemas de la Universidad Distrital. “Para poder construir soluciones de software se necesita un equipo de desarrollo, una metodología, una serie de recursos en

hardware como servidores, ordenadores, dispositivos de almacenamiento, dispositivos incorporados en red. Y una serie de recursos a nivel de software como las bases de datos, DEs, Frameworks, librerías, Pods y NuGets, los cuales gracias a los nuevos avances tecnológicos hacen más sencilla la forma de integrar herramientas en la nube, introduciendo conceptos como PaaS y IaaS los cuales brindan métodos más versátiles y fáciles de afrontar los nuevos retos a solucionar. Todo esto hace que las soluciones estén cada vez más cargadas de componentes de diversas tecnologías aumentando así la cantidad de requisitos para la construcción de las mismas lo que lleva al departamento de infraestructura de las compañías a tener que instalar y configurar mayor cantidad de softwares dentro de los entornos de desarrollo. La Oficina Asesora de Sistemas, carece de un sistema que gestione el despliegue de los diferentes entornos de desarrollo de forma eficaz y automatizada, que sea configurada una única vez y solo baste con que el administrador de la infraestructura corra una “receta” y el sistema de gestión despliegue las bases de datos, IDEs, Frameworks, librerías, Pods y NuGets, necesarios para dicho entorno de trabajo. Esta solución tecnológica permite reducir el tiempo invertido si se efectuara de forma individual permitiendo ejecutar una serie de comandos secuenciales en los dispositivos pertenecientes a un entorno de trabajo en específico. Proporcionando características relevantes como lo es la habilitación de los puertos de comunicación de cada una de las aplicaciones o la versión de la aplicación que se va a manejar dentro del proyecto, generando un estándar para todo el proyecto. Con lo cual se reducen los errores humanos teniendo en cuenta que el administrador de la infraestructura, al interactuar solo con una configuración inicial, tanto el esfuerzo como el tiempo de

concentración se reduce radicalmente, sin permitir que se vuelva una tarea monótona y repetitiva, causal de disminución del índice de atención.” (Menseses, 2016). Este informe presenta información importante en la cual el autor del libro tomo como referencia para realizar un primer acercamiento a la arquitectura física y al despliegue de aplicaciones con contenedores.

2.4.3 Regional:

Migración del software kactus-hcm de una arquitectura Cliente-servidor a una arquitectura cliente-contenedor. “El presente proyecto se ha llevado a cabo en la empresa Digital Ware con la colaboración del área de tecnología y del Ingeniero Edwin Barreto, este consiste en realizar la migración de uno de los productos de la compañía, dicho producto cuenta actualmente con una arquitectura de cliente-servidor, y la misión es cambiar el servidor físico o virtualizado, por un contenedor de software que contenga todo lo necesario para realizar el despliegue de las aplicaciones. Lo cual acarrea una investigación, puesto que, es una tecnología emergente, por tanto, se deberá comprender su funcionamiento y conocer las oportunidades que esta brinda. El desarrollo de este se ejecutará en un ambiente de pruebas, además, con este trabajo se pretende brindar a la empresa una herramienta que le permita disminuir los costos en la infraestructura y disminuir los tiempos de despliegue de las aplicaciones, ya sea para ambientes productivos o ambientes de pruebas” (HOYOS, 2019). Este informe presenta información importante en la cual el autor del libro tomo como referencia al enfoque en el despliegue de aplicaciones con contenedores.

3 Análisis preliminar.

El siguiente capítulo al cual el escritor del libro lo segmentó describiendo los principales aspectos que hacen referencia a las herramientas de empaquetado de aplicaciones o contenedores y de herramientas de orquestación. Se enfatizó con aspectos generales de estas, con el fin de contar con una base sólida que aporte a la toma de decisiones.

3.1 Contenedores de software.

Los contenedores de software son ambientes de ejecución de aplicaciones con archivos, variables de entorno y librerías que necesita la aplicación para que se muestre de forma satisfactoria, incluso, cuando esta es cambiada de entorno, reduciendo al mínimo las posibles fallas y maximizando su portabilidad. La virtualización mediante contenedores no inicia un sistema operativo adicional, por el contrario, las máquinas virtuales convencionales si lo hacen, de esta forma, la virtualización mediante contenedores requiera de recursos computacionales mínimos y son rápidos y fáciles de instalar, lo que permite desplegar un número mucho mayor de aplicaciones en los servidores.

El análisis preliminar se centra en un análisis a las herramientas a tratar, buscando puntos de referencia más importantes de cada una de ellas con esto facilitando la toma de decisiones.

3.1.1 Docker.

Esta herramienta es una de las más comunes, Docker es una plataforma abierta para desarrollar, enviar y ejecutar aplicaciones. Docker le permite separar sus aplicaciones de su infraestructura para que pueda entregar software rápidamente. Con Docker, puede

administrar su infraestructura de la misma manera que administra sus aplicaciones. Al aprovechar las metodologías de Docker para enviar, probar e implementar código rápidamente, puede reducir significativamente el retraso entre la escritura del código y su ejecución en producción.

Docker brinda la capacidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado contenedor. El aislamiento y la seguridad le permiten ejecutar muchos contenedores simultáneamente en un host determinado. Los contenedores son livianos porque no necesitan la carga adicional de un hipervisor, sino que se ejecutan directamente dentro del kernel de la máquina host. Esto significa que puede ejecutar más contenedores en una combinación de hardware determinada que si estuviera utilizando máquinas virtuales (Docker docs, 2020).

El funcionamiento de esta herramienta proviene del motor de Docker o también llamado Docker Engine, que es una aplicación cliente-servidor con estos componentes principales:

- Un servidor que es un tipo de programa de larga duración llamado proceso demonio, también llamado “the dockerd command”.
- Una API REST que especifica interfaces que los programas pueden usar para hablar con el demonio e indicarle que hacer.
- Un cliente de interfaz de línea de comandos (CLI)(the docker command).

Como se evidencia en la ilustración 3 (Docker docs, 2020), el CLI utiliza la API REST de Docker para controlar o interactuar con el demonio de Docker a través de secuencias de comandos o comandos directos de la CLI. Muchas otras aplicaciones de Docker utilizan la API y la CLI subyacentes. El Daemon crea y administra objetos Docker, como imágenes, contenedores, redes y volúmenes.

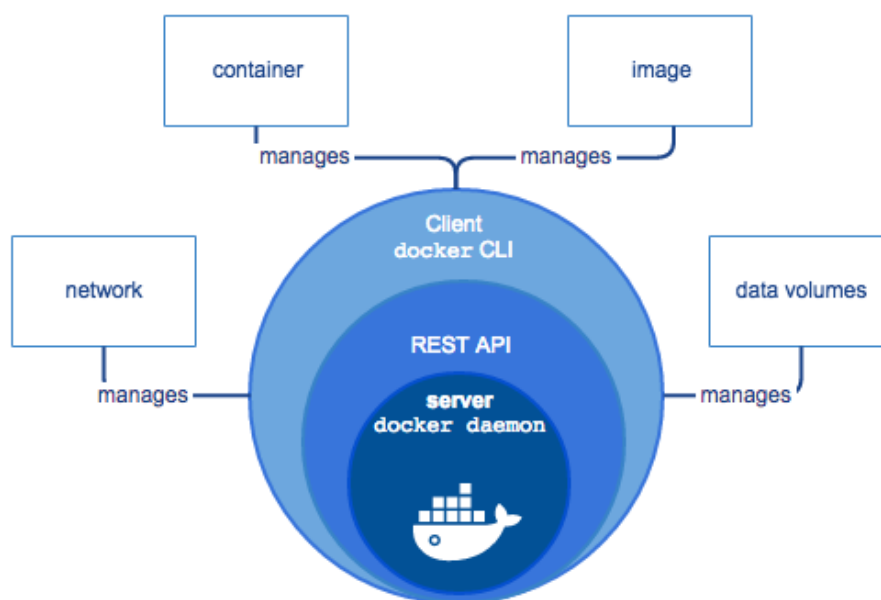


Ilustración 3. Motor de Docker. (Docker docs, 2020).

3.1.1.1 Arquitectura:

La arquitectura que utiliza Docker es una arquitectura cliente-servidor, como se ve en la ilustración 4 (Docker docs, 2020). El cliente de Docker habla con el demonio de Docker, que hace el trabajo pesado de compilar, ejecutar y distribuir sus contenedores de Docker. El

cliente y el demonio de Docker pueden ejecutarse en el mismo sistema, o puede conectar un cliente de Docker a un demonio de Docker remoto. El cliente y el demonio de Docker se comunican mediante una API REST, a través de sockets UNIX o una interfaz de red.

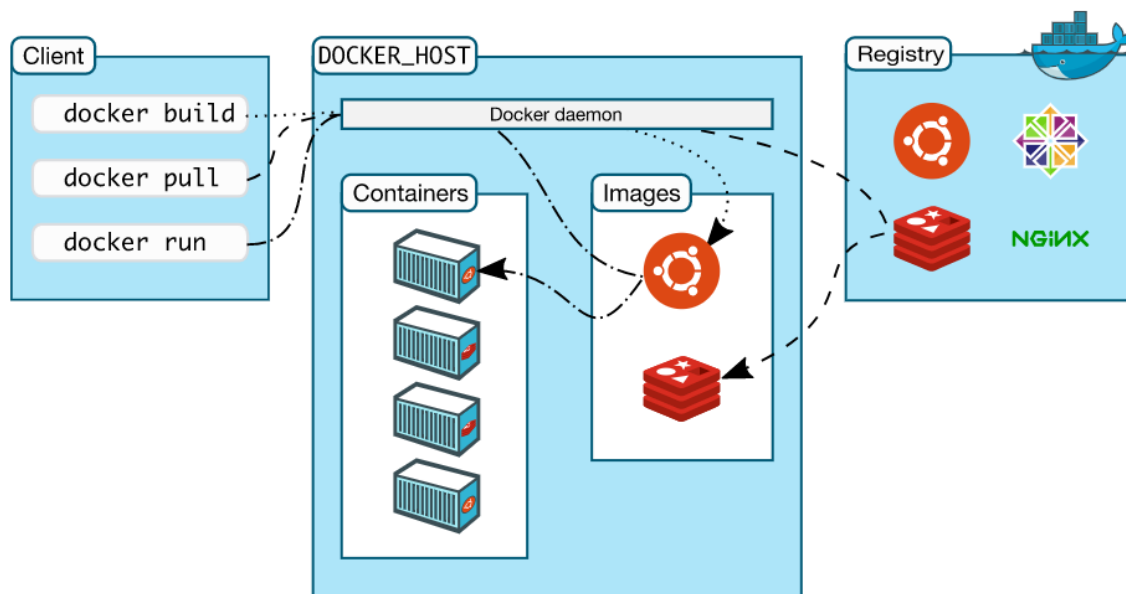


Ilustración 4. Arquitectura de Docker (Docker docs, 2020).

- **El demonio de Docker.**

El demonio de Docker (`dockerd`) escucha las solicitudes de la API de Docker y administra los objetos de Docker, como imágenes, contenedores, redes y volúmenes. Un daemon también puede comunicarse con otros daemons para administrar los servicios de Docker.

- **El cliente de Docker.**

El cliente de Docker (`docker`) es la forma principal en que muchos usuarios de Docker interactúan con Docker. Cuando utiliza comandos como `docker run`, el cliente envía estos

comandos a `dockerd`, que los ejecuta. El `docker command` usa la API de Docker. El cliente de Docker puede comunicarse con más de un demonio.

- **Registros de Docker.**

Un registro de Docker almacena imágenes de Docker. Docker Hub es un registro público que cualquiera puede usar y Docker está configurado para buscar imágenes en Docker Hub de forma predeterminada. Incluso puede ejecutar su propio registro privado. Cuando usa los comandos `docker pull`, `docker run`, las imágenes requeridas se extraen de su registro configurado. Cuando usa el comando `docker push`, su imagen se envía a su registro configurado.

- **Objetos Docker.**

Cuando usa Docker, está creando y usando imágenes, contenedores, redes, volúmenes, complementos y otros objetos. Esta sección es una breve descripción de algunos de esos objetos.

- **Imágenes Docker.**

Una imagen es una plantilla de solo lectura con instrucciones para crear un contenedor Docker. A menudo, una imagen se basa en otra imagen, con alguna personalización adicional. Por ejemplo, puede crear una imagen basada en la Ubuntu imagen, pero instala el servidor web Apache y su aplicación, así como los detalles de configuración necesarios para ejecutar su aplicación. Puede crear sus propias imágenes o puede usar solo las creadas por otros y publicadas en un registro.

- **Contenedores.**

Un contenedor es una instancia ejecutable de una imagen. Puede crear, iniciar, detener, mover o eliminar un contenedor mediante la API o la CLI de Docker. Puede conectar un contenedor a una o más redes, adjuntar almacenamiento a él o incluso crear una nueva imagen basada en su estado actual.

- **Servicios.**

Los servicios le permiten escalar contenedores en varios demonios de Docker, que funcionan juntos como un enjambre con varios administradores y trabajadores. Cada miembro de un enjambre es un demonio de Docker, y todos los demonios se comunican mediante la API de Docker. Un servicio le permite definir el estado deseado, como la cantidad de réplicas del servicio que deben estar disponibles en un momento dado. De forma predeterminada, el servicio tiene un equilibrio de carga en todos los nodos trabajadores. Para el consumidor, el servicio Docker parece ser una sola aplicación (Docker docs, 2020).

- **La tecnología subyacente.**

Docker está escrito en el lenguaje de programación Go y aprovecha varias características del kernel de Linux para ofrecer su funcionalidad.

- **Espacios de nombres.**

Docker utiliza una tecnología llamada namespaces para proporcionar el espacio de trabajo aislado llamado contenedor. Cuando ejecuta un contenedor, Docker crea un conjunto de espacios de nombres para ese contenedor. Estos espacios de nombres proporcionan una capa de aislamiento. Cada aspecto de un contenedor se ejecuta en un espacio de nombres separado

y su acceso está limitado a ese espacio de nombres. Docker Engine usa espacios de nombres como los siguientes en Linux:

- El pid espacio de nombres: Aislamiento del proceso (PID: ID del proceso).
- El net espacio de nombres: Gestión de interfaces de red (NET: Networking).
- El ipc espacio de nombres: gestión del acceso a los recursos de IPC (IPC: comunicación entre procesos).
- El mnt espacio de nombres: Gestión de puntos de montaje del sistema de archivos (MNT: Mount).
- El uts espacio de nombres: aislar identificadores de kernel y versión. (UTS: Sistema de tiempo compartido Unix).

- **Grupos de control.**

Docker Engine en Linux también se basa en otra tecnología llamada grupos de control (cgroups). Un cgroup limita una aplicación a un conjunto específico de recursos. Los grupos de control permiten a Docker Engine compartir los recursos de hardware disponibles con los contenedores y, opcionalmente, hacer cumplir los límites y restricciones. Por ejemplo, puede limitar la memoria disponible a un contenedor específico.

- **Sistemas de archivos de unión.**

Los sistemas de archivos Union, o UnionFS, son sistemas de archivos que operan creando capas, lo que los hace muy ligeros y rápidos. Docker Engine utiliza UnionFS para proporcionar los componentes básicos de los contenedores.

- **Formato de contenedor.**

Docker Engine combina los espacios de nombres, los grupos de control y UnionFS en un contenedor llamado formato contenedor. El formato de contenedor predeterminado es libcontainer. En el futuro, Docker puede admitir otros formatos de contenedor mediante la integración con tecnologías como BSD Jails o Solaris Zones.

3.1.2 Rkt.

Rkt también conocido como rocket es una CLI para ejecutar contenedores de aplicaciones en Linux. rkt está diseñado para ser seguro, compatible y basado en estándares. rkt es un motor de contenedores de aplicaciones desarrollado para entornos nativos de la nube de producción moderna. Cuenta con un enfoque nativo de pod, un entorno de ejecución conectable y un área de superficie bien definida que lo hace ideal para la integración con otros sistemas.

La unidad de ejecución central de rkt es el pod, una colección de una o más aplicaciones que se ejecutan en un contexto compartido. rkt permite a los usuarios aplicar diferentes configuraciones (como parámetros de aislamiento) tanto a nivel de pod como a nivel más granular por aplicación. La arquitectura de rkt significa que cada pod se ejecuta directamente en el modelo de proceso clásico de Unix (es decir, no hay un demonio central), en un entorno autónomo y aislado. rkt implementa un formato de contenedor estándar, abierto y moderno, la especificación App Container (appc), pero también puede ejecutar otras imágenes de contenedor, como las creadas con Docker (CoreOS, 2020).

Desde su introducción por CoreOS en diciembre de 2014, el proyecto rkt ha madurado mucho y se usa ampliamente. Está disponible para la mayoría de las principales distribuciones de Linux y cada versión de rkt crea paquetes rpm / deb independientes que los usuarios pueden instalar. Algunas de las características y objetivos clave de rkt incluyen:

- **Pod-native.**

la unidad básica de ejecución de rkt es un pod, que vincula recursos y aplicaciones de usuario en un entorno autónomo.

- **Seguridad.**

rkt está desarrollado con un principio de "seguridad por defecto" e incluye una serie de características de seguridad importantes como soporte para SELinux, medición de TPM y ejecución de contenedores de aplicaciones en máquinas virtuales aisladas por hardware.

- **Compuestabilidad.**

rkt está diseñado para la integración de primera clase con los sistemas de init (como systemd, advenedizos) y herramientas de orquestación de clúster (como Kubernetes y Nomad), y soportes de motores de ejecución intercambiables.

- **Estándares abiertos y compatibilidad.**

rkt implementa la especificación appc, admite la especificación Container Networking Interface y puede ejecutar imágenes de Docker e imágenes OCI. Más amplio soporte nativo para imágenes y tiempos de ejecución OCI está en desarrollo.

3.1.3 LXC.

LXC es una interfaz de espacio de usuario para las funciones de contención del kernel de Linux. A través de una poderosa API y herramientas simples, permite a los usuarios de Linux crear y administrar fácilmente contenedores de aplicaciones o sistemas.

LXC también conocidos como Contenedor de Linux se refiere a las aplicaciones virtualizadas sobre la base de Linux, así como a la plataforma y la tecnología de contenedores subyacente. Esto debe tenerse en cuenta sobre todo cuando se habla de otras plataformas de contenedores que también utilizan la tecnología de Linux Containers. Esta herramienta está diseñada para la integración de primera clase con los sistemas de init (como systemd, advenedizos) y herramientas de orquestación de clúster (como Kubernetes y Nomad), y soportes de motores de ejecución intercambiables.

El concepto de la tecnología de contenedores de Linux se remonta a 2001. Ese año, por primera vez, se implementó un entorno aislado en el marco del proyecto VServer. Esa fue la base para crear diversos espacios de nombres (namespaces) controlados en Linux y los ahora denominados contenedores de Linux. Les siguieron otras tecnologías, como los cgroups (grupos de control), que permiten controlar y restringir el uso de recursos para un proceso o conjunto de procesos, o systemd, un sistema de inicialización para administrar los espacios de nombres y sus procesos. En la práctica, LXC agiliza el desarrollo de las aplicaciones, ya que la tecnología de contenedores contribuye a la portabilidad, la configuración y el aislamiento, entre otras cosas. Los contenedores también son útiles a la hora de transmitir datos en tiempo real, porque proporcionan la escalabilidad necesaria a las aplicaciones.

Asimismo, los contenedores de Linux se adaptan a cada infraestructura, lo que los hace muy independientes, de modo que pueden implementarse tanto localmente como en la nube o en un entorno híbrido.

De forma predeterminada, los contenedores de Linux consumen menos recursos que una máquina virtual y tienen una interfaz de usuario estándar, lo que facilita y simplifica la administración de varios contenedores al mismo tiempo. Incluso es posible gestionar una plataforma de LXC en varias nubes: esto asegura la portabilidad y garantiza que las aplicaciones que se ejecutan correctamente en el sistema del desarrollador también funcionen en cualquier otro sistema. A través de la interfaz del contenedor de Linux, incluso las aplicaciones más grandes pueden iniciarse, detenerse o cambiar sus variables de entorno. En resumen, el objetivo de LXC es crear un entorno que se parezca lo más posible a una instalación estándar de Linux sin la necesidad de un kernel independiente (Containers, 2017).

La plataforma de contenedores de Linux actual utiliza las siguientes características del kernel para incorporar aplicaciones y procesos en los contenedores:

- Espacios de nombres del núcleo (ipc, uts, mount, pid, red y usuario).
- Perfiles Apparmor y SELinux
- Directrices de Seccomp
- Chroots (utilizando pivot_root)
- Capacidades del kernel
- cgroups (grupos de control)

Los contenedores de Linux deben permanecer compactos. Por lo tanto, solo constan de unos pocos componentes:

- Biblioteca liblxc
- Vinculación de varios lenguajes para la API:
 - Python 3 (soporte a largo plazo en 2.0.x)
 - Lua (soporte a largo plazo en 2.0.x)
 - Go
 - Ruby
 - Python
 - Haskell
- Varias herramientas estándar para administrar los contenedores
- Plantillas para distribuciones

El funcionamiento de LXC o Contenedor de Linux, radica en la importancia del aislamiento y la virtualización que ayudan a administrar los recursos y las medidas de seguridad de la manera más eficiente posible. Además, facilitan la supervisión, permitiendo, por ejemplo, detectar errores en el sistema que a menudo no tienen nada que ver con las aplicaciones en desarrollo. La forma más fácil y razonable de utilizar los contenedores de Linux es vincular cada contenedor a un proceso, de manera que el usuario mantenga totalmente el control. Para cada proceso, son especialmente importantes los espacios de nombres, que hacen que los recursos estén disponibles para uno o más procesos que utilicen

el mismo espacio de nombres. Los procesos también permiten controlar el acceso para garantizar la seguridad de los contenedores.

Para implementar un entorno LXC, las características y sus funciones deben definirse claramente. Los cgroups (grupos de control del kernel) limitan y aíslan los recursos del proceso, como CPU, E/S, memoria RAM y recursos de red. Además, el contenido de un grupo de control se puede administrar, monitorear, priorizar y editar.

LXC se desarrolló para ejecutar diferentes contenedores de sistema (full system containers) en un sistema anfitrión. Un contenedor Linux suele iniciar una distribución completa en un entorno virtual partiendo de una imagen del sistema operativo y los usuarios interactúan con ella de forma parecida a como harían con una máquina virtual (Containers, 2017).

3.2 Motores de orquestación.

La orquestación de contenedores hace todas estas cosas por medio de automatización de infraestructura. El gran beneficio de la orquestación es aprovechar al máximo tareas automatizadas y procesos ya definidos. La orquestación permite gestionar infraestructura muy compleja de manera simple (aplyca, 2018). Una herramienta de orquestación de contenedores tiene como propósito el manejo del ciclo de vida de los contenedores, principalmente en ambientes de producción dinámicos, son utilizados para el manejo de diferentes tareas. Por ejemplo:

- Monitorear la salud de los contenedores activos.

- Provisionar redundancia.
- Provisionar alta disponibilidad.
- Balanceo de carga y descubrimiento de servicios.
- Repartición de recursos y otras tareas.

El utilizar una herramienta de orquestación implica que la configuración del ambiente de despliegue se describe en uno o varios archivos de configuración, estas configuraciones incluyen las imágenes de los contenedores a utilizar, volúmenes de almacenamiento, conexión de red entre los diferentes servicios entre otras configuraciones. El poder utilizar este tipo de archivos (YML o JSON) permite que se pueda llevar un control de las versiones de las configuraciones y controlar el despliegue en diferentes ambientes, puede ser un ambiente de pruebas o un ambiente de producción, esto debido a que todo se maneja como código. Las herramientas de orquestación de contenedores permiten desplegar los contenedores en clústers, además, permiten administrar los recursos necesarios para cada uno de los servicios funcione de manera satisfactoria y agregar o quitar contenedores de ser necesario (Hernández R. A., 2020).

El análisis preliminar se centra en un análisis a las herramientas a tratar, buscando puntos de referencia más importantes de cada una de ellas con esto facilitando la toma de decisiones.

Actualmente existen diferentes herramientas disponibles para lograr todo lo anteriormente mencionado, a continuación, se mencionan algunas de ellas.

3.2.1 Kubernetes.

Kubernetes es un motor de orquestación de contenedores de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores.

Kubernetes se encarga del trabajo duro en el escalamiento, recuperación automática, balanceo de cargas, despliegues y mucho más. Agrupa los contenedores que componen una aplicación en unidades lógicas para gestionarlas y darles visibilidad.

En el año 2014, Google liberó Kubernetes, su propia herramienta open source de orquestación de contenedores. Kubernetes se desarrolló originalmente para ejecutar las numerosas aplicaciones globales de Google para modificar sus productos de manera fácil y eficaz, así como hacer correcciones de software a la mayor escala posible. Se rumora que Google ejecuta alrededor de dos mil millones de contenedores por semana en su versión interna de Kubernetes.

El proyecto recoge 15 años de experiencia de Google usando contenedores y los enriquece con las mejores ideas y prácticas de una comunidad inmensa. La herramienta funciona agrupando contenedores que componen una aplicación en unidades lógicas para una fácil gestión y descubrimiento (Kubernetes, 2020).

Kubernetes es un sistema orquestador, pero no solo de contenedores, si no de todas sus características como dónde recopilar las imágenes de los contenedores, por ejemplo, de Docker Hub, cómo establecer la red, cómo montar volúmenes de almacenamiento, dónde almacenar los logs. Es claramente un sistema operativo para gestionar cada uno de tus

equipos, en la ilustración 5 (RedHat, 2020), se puede evidenciar la arquitectura de Kubernetes.

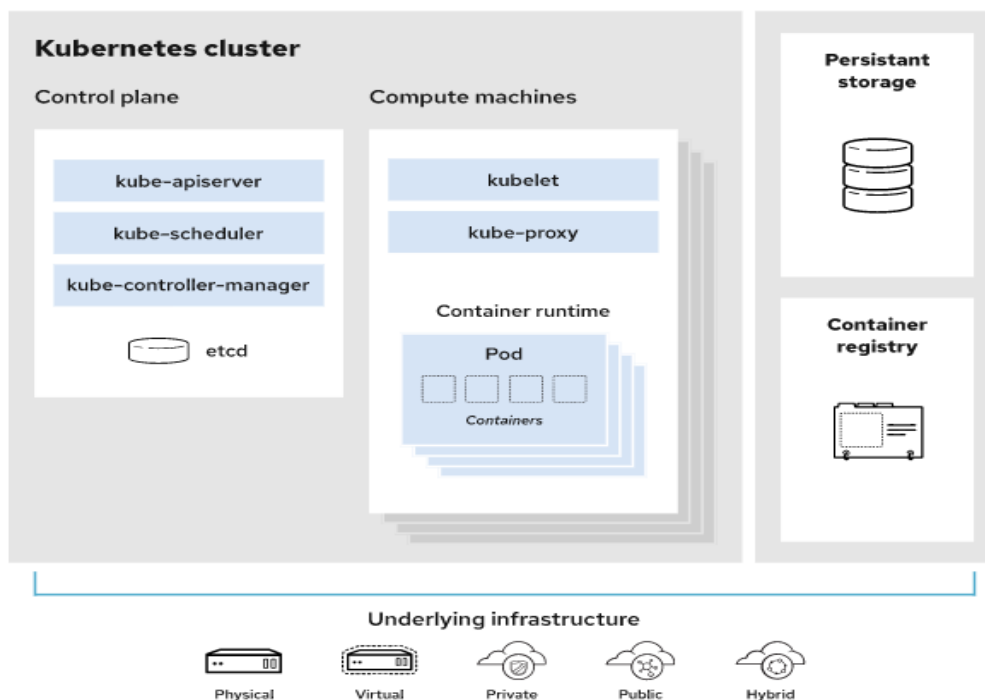


Ilustración 5. Arquitectura Kubernetes (RedHat, 2020).

- **Clúster.**

Grupo de máquinas virtuales o físicas que alojan Kubernetes.

- **Pod.**

Un pod es un grupo de uno o más contenedores, con almacenamiento compartido, recursos de red y una especificación sobre como ejecutar los contenedores.

- **Labels y selectors.**

Es un sistema Key: Value asociado a pods, services, replication controllers que te permitirá identificarlos para luego tener la capacidad de gestionarlos.

- **Nodo.**

Servidor que aloja el sistema de Kubernetes y donde se desplegarán todos los pods ‘contenedores’.

- **Nodo maestro.**

Servidor que aloja el sistema de Kubernetes y controla los nodos. Esta máquina es la encargada de asignar tareas.

- **Nodo de trabajo.**

Servidor que realiza las tareas solicitadas y asignadas.

- **Controlador de replicación.**

Sistema responsable de gestionar la vida, estado y características de los pods, permitiéndote poder escalar de forma sencilla.

- **Deployments.**

Número de réplicas de pods que tendrás en el sistema.

- **Namespaces.**

Espacios de trabajo para diferentes escenarios. Por ejemplo, podrías realizar un Namespace para producción y otro para desarrollo y cada Namespace tendría sus propios pods, replication controllers.

- **Volumen.**

Sistema de almacenamiento, al que pueden acceder los contenedores de un pod.

- **Secrets.**

Donde se añaden las credenciales de configuración para poder acceder a los recursos.

- **Plano de control.**

EL plano de control es la pieza central del clúster de Kubernetes. Aquí se encuentran los elementos de que controlan el clúster, junto con los datos sobre su estado y configuración. Los elementos principales de Kubernetes tienen la importante tarea de garantizar que los contenedores se ejecuten en cantidades suficientes y con los recursos necesarios.

El plano de control está en contacto permanente con las máquinas informáticas. Se encarga de garantizar que el clúster se ejecute de acuerdo con la manera en que usted lo haya configurado (RedHat, 2020).

- **kube-apiserver.**

Para interactuar con el clúster de Kubernetes se debe emplear la API de Kubernetes, que es el frontend del plano de control de este sistema y se encarga de gestionar las solicitudes internas y externas. El servidor de API determina si una solicitud es válida, y en ese caso la procesa. Puede acceder a la API con llamadas de REST, con la interfaz de la línea de comandos `kubectl` o con otras herramientas de la línea de comandos, como `kubeadm`.

- **kube-scheduler.**

¿Su clúster está en buen estado? Si se necesitaran clústeres nuevos, ¿dónde sería más adecuado colocarlos? De esto se encarga el programador de Kubernetes.

Para ello, debe tener en cuenta los recursos que necesita un pod, como CPU o memoria, junto con el estado del clúster. Luego, programa el pod en un nodo de trabajo adecuado.

- **kube-controller-manager.**

Los controladores son los que realmente ejecutan el clúster. El controlador y administrador de Kubernetes contiene varias funciones de este tipo en una. El controlador realiza una consulta al programador y se asegura de que se esté ejecutando la cantidad correcta de pods. Si uno de ellos se cae, otro controlador lo percibe y responde al problema. Un controlador conecta los servicios a los pods, de manera que las solicitudes van a los extremos correctos. Además, hay algunos que permiten crear cuentas y tokens de acceso a las API.

- **Etd.**

Los datos de configuración y la información sobre el estado del clúster se alojan en la etcd, una base de datos de almacenamiento de valor clave. Esta base de datos distribuida y con tolerancia a los fallos está diseñada para ser la principal fuente de información del clúster.

- **Kubelet.**

Cada nodo de trabajo contiene una kubelet: una aplicación pequeña que se comunica con el nodo maestro y garantiza que los contenedores se ejecuten en un pod. Cuando el plano de control necesita que algo suceda en un nodo, la kubelet ejecuta la acción.

- **kube-proxy**

Los nodos de trabajo también contienen un kube-proxy, es decir, un proxy de red para facilitar los servicios de red de Kubernetes. El kube-proxy administra las comunicaciones de red dentro y fuera del clúster. Para ello, confía en la capa de filtrado de paquetes de su sistema operativo o reenvía el tráfico por cuenta propia (RedHat, 2020).

3.2.2 Docker Swarm.

Docker Swarm es una herramienta de orquestación de contenedores, lo que significa que permite al usuario administrar múltiples contenedores implementados en múltiples máquinas host. Un Docker Swarm es un grupo de máquinas físicas o virtuales que ejecutan la aplicación Docker y que se han configurado para unirse en un clúster. Una vez que un grupo de máquinas se ha agrupado, aún puede ejecutar los comandos de Docker a los que está acostumbrado, pero ahora los ejecutarán las máquinas de su clúster. Las actividades del clúster están controladas por un administrador de enjambres y las máquinas que se han unido al clúster se denominan nodos. Docker Swarm tiene dos tipos de servicios: replicados y globales.

- **Servicios replicados.**

Los servicios replicados en el modo Swarm funcionan al especificar una serie de tareas de réplica para que el administrador del enjambre las asigne a los nodos disponibles.

- **Servicios globales.**

Los servicios globales funcionan mediante el uso del administrador de nodo para programar una tarea para cada nodo disponible que cumpla con las restricciones de servicios y los requisitos de recursos.

Docker Engine, como se ve en la ilustración 6 (RedHat, 2020), introduce el modo enjambre que le permite crear un clúster de uno o más motores Docker llamado enjambre.

Un enjambre consta de uno o más nodos: máquinas físicas o virtuales que ejecutan Docker Engine 1.12 o posterior en modo enjambre.

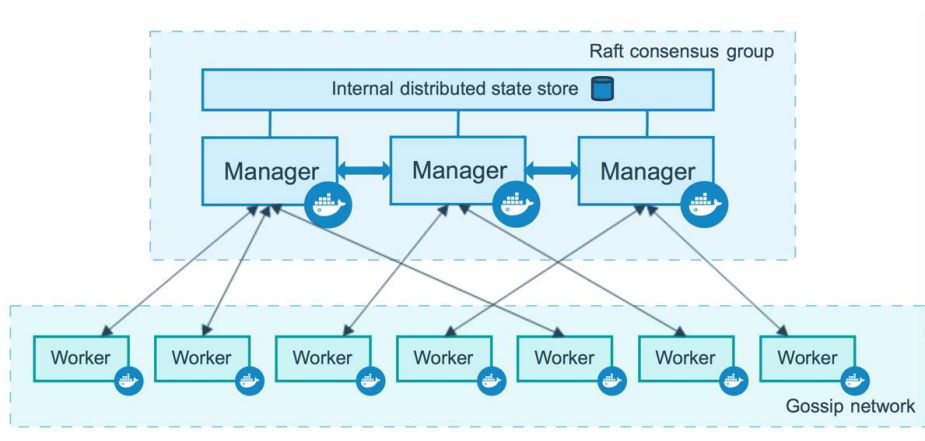


Ilustración 6. Arquitectura Docker Swarm (Hernández A. , 2019).

Docker Swarm está compuesto por un grupo de máquinas físicas o virtuales que operan en un clúster. Cuando una máquina se une al clúster, se convierte en un nodo en ese enjambre. La función Docker Swarm reconoce tres tipos diferentes de nodos, cada uno con un rol diferente dentro del ecosistema Docker Swarm:

- **Nodo administrador.**

La función principal de los nodos administradores es asignar tareas a los nodos trabajadores en el enjambre. Los nodos de administrador también ayudan a llevar a cabo algunas de las tareas administrativas necesarias para operar el enjambre. Docker recomienda un máximo de siete nodos de administrador para un enjambre.

- **Nodo líder.**

Cuando se establece un clúster, se utiliza el algoritmo de consenso de Raft para asignar uno de ellos como "nodo líder". El nodo líder toma todas las decisiones de gestión del enjambre y orquestación de tareas para el enjambre. Si el nodo líder deja de estar disponible debido a una interrupción o falla, se puede elegir un nuevo nodo líder utilizando el algoritmo de consenso de Raft.

- **Nodo trabajador.**

En un enjambre de ventanas acoplables con numerosos hosts, cada nodo trabajador funciona recibiendo y ejecutando las tareas que le asignan los nodos administradores. De forma predeterminada, todos los modos de administrador también son nodos de trabajo y son capaces de ejecutar tareas cuando tienen los recursos disponibles para hacerlo.

3.2.3 Nomad.

Nomad es un orquestador de cargas de trabajo simple, flexible y fácil de usar para implementar y administrar contenedores y aplicaciones no contenedores en las instalaciones y en la nube a escala. Nomad se ejecuta como un único binario con una pequeña huella de recursos (35 MB) y es compatible con macOS, Windows, Linux.

Los desarrolladores utilizan infraestructura declarativa, como se evidencia en la ilustración 7 (Nomad, 2017), para implementar sus aplicaciones y definir cómo se debe ejecutar una aplicación. Nomad recupera automáticamente las aplicaciones de las fallas.

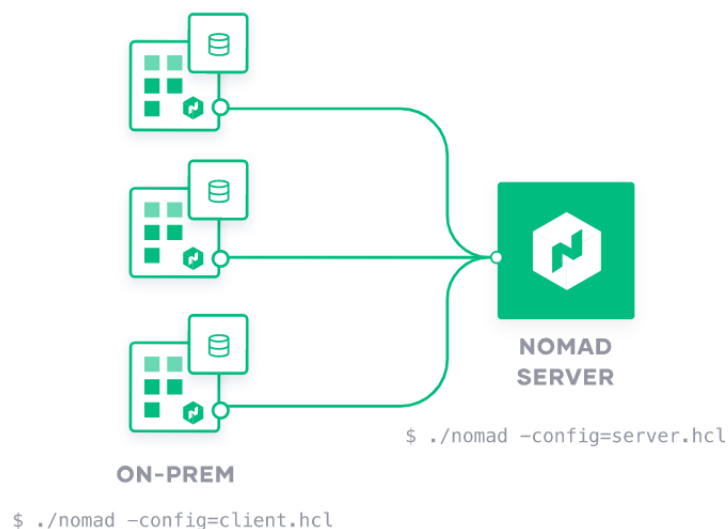


Ilustración 7. Arquitectura Nomad (Nomad, 2017).

Nomad Orquesta aplicaciones de cualquier tipo, no solo contenedores. Proporciona soporte de primera clase para Docker, Windows, Java, VM. Es un planificador de despliegues que se encarga de ejecutar de manera distribuida los procesos que se le indiquen, escalarlos y supervisar su ejecución para lanzar nuevas instancias en caso de caídas.

Así mismo, es una pequeña pieza de software que se instala como un simple binario. El clúster debe contar con 3 servidores (para evitar pérdida de información) que se encargarán de la gestión de los despliegues y la monitorización y de tantos clientes como se requiera para alojar estos despliegues (Nomad, 2017).

4 Definición base.

Como resultado del capítulo 3, se obtiene las bases fundamentales para realizar la toma de decisiones sobre qué herramienta de empaquetado de software y que herramienta orquestación de contenedores se debe seleccionar para llevar a cabo el proceso operacional para posteriormente realizar el diseño del prototipo. Este capítulo describe como el autor seleccionó la herramienta de empaquetado de software y herramienta de orquestación de software realizando una comparativa de las herramientas, identificando las ventajas y desventajas que cada herramienta mencionada en el capítulo anterior.

4.1 Selección de herramienta de empaquetado de software.

Para la selección de esta, inicialmente se realizó una exploración e investigación de las herramientas más comunes que se adaptaban a la intención del prototipo, la cual es que sea fácil de implementar y de usar en cuestión de practicidad, optimizar el despliegue de aplicaciones, además, la decisión se basó en la herramienta que cumpliera con los siguientes parámetros: herramienta que soporte diversas plataformas, Facilidad de implementación, recomendaciones de otros autores y completitud de la herramienta. Las herramientas seleccionadas son open source. Un factor importante en la toma de decisión, será el apoyo que tienen estas herramientas de empresas externas y el apoyo de la comunidad. Dando como resultado la herramienta de empaquetado de software. Como primera instancia se descartaron

algunas herramientas de empaquetado de software, que, si bien son muy comunes, no son de open source.

El autor seleccionó para realizar la comparativa tres herramientas de las más comunes y utilizadas por la comunidad: Docker, rkt también conocido como Rocket y LXC también conocido como Linux Container.

4.1.1 Identificar las ventajas y desventajas de cada tipo.

Para realizar la comparativa se partió de escritores que ya hubieran realizado estudios a estas técnicas, bajo estos el autor decide plasmar las ventajas, desventajas y otros criterios; dando como resultado las siguientes tablas 1, 2 y 3 que representan las ventajas y desventajas de Docker, rkt y LXC respectivamente:

- **Docker.**

Tabla 1. Ventajas y desventajas de Docker (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • Docker es compatible con diversos sistemas operativos y plataformas como: Linux, Windows, MacOS, Microsoft Azure, Amazon Web Services (AWS). • Con las herramientas Docker Swarm y Docker Compose, la plataforma Docker ya ofrece herramientas de gestión de clústers. • Los usuarios cuentan con un amplio repositorio de imágenes, ya sea imágenes oficiales o no oficiales, gracias al apoyo de empresas externas y la comunidad. • Docker está en constante crecimiento, brinda una gran documentación y pone a disposición de sus usuarios diversas herramientas, plugins y componentes de infraestructura. 	<ul style="list-style-type: none"> • El motor de Docker solo es compatible con su propio formato de contenedor. • Los contenedores Docker están enfocados a aislar procesos entre sí, pero no virtualizan sistemas operativos completos.

- **rkt (rocket).**

Tabla 2. Ventajas y desventajas de rkt (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • rocket soporta diferentes formatos de imágenes, su propio formato ACI e imágenes docker. • Rocket facilita separar contenedores de software entre sí de forma segura. 	<ul style="list-style-type: none"> • El apoyo que tiene esta herramienta es menor a otras herramientas del mercado, lo cual brinda menos apoyo al obtener errores. • Rkt no virtualiza sistemas operativos completos.

- **LXC (Linux Containers).**

Tabla 3. Ventajas y desventajas de LXC (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • LXC o <i>Linux Containers</i> si está optimizado para virtualizar sistemas operativos completos. 	<ul style="list-style-type: none"> • Ya que esta herramienta, no brinda virtualización de aplicaciones, no es tan común que se utiliza de forma estándar. • Esta herramienta solo brinda implementación para distribuciones Linux, no cuenta con ninguna implementación nativa para otros sistemas operativos.

Estas ventajas y desventajas son tomadas de referencia de la documentación oficial de Docker (Docker docs, 2020), la documentación oficial de Rkt (CoreOS, 2020), la documentación oficial de LXC (Containers, 2017), artículos relacionados como Alternativas a los contenedores Docker (IONOS S. , 2020) y de algunos sitios web únicamente como criterios.

Con las ventajas y desventajas definidas, el autor del libro seleccionó como herramienta de empaquetado de software a Docker y hará uso de esta para el proceso operacional. La decisión para seleccionar esta herramienta se debe al crecimiento y aceptación que ha tenido Docker en los últimos años, además, tiene un gran apoyo por parte de grandes compañías y de la comunidad. Actualmente Docker cuenta con un repositorio oficial y público, donde hay multitud de imágenes gratuitas, las cuales se utilizan como plantillas para la creación de contenedores, llamado Docker Hub.

Docker al ser la herramienta de empaquetado de software más común en la actualidad, cuenta con una gran bibliográfica y documentación sobre su uso e implementación en múltiples plataformas. Rocket es la herramienta que más compete contra Docker, actualmente no tiene el mismo apoyo por la comunidad, ni la cantidad de imágenes disponibles para la creación de contenedores.

4.2 Selección de herramienta de orquestación de contenedores.

Para la selección de esta, inicialmente se realizó una exploración e investigación de las herramientas más comunes que se adaptaban a la herramienta seleccionada anteriormente, la decisión se basó en la herramienta que cumpliera con los siguientes parámetros: herramienta que soporte diversas plataformas, recomendaciones de otros autores y completitud de la herramienta. Un factor importante en la toma de decisión será el apoyo que tiene esta herramienta de empresas externas o el apoyo de la comunidad. Dando como resultado la herramienta de orquestación de contenedores.

El autor seleccionó para realizar la comparativa tres herramientas de las más comunes y utilizadas por la comunidad, como resultado en las tablas 4, 5 y 6 son presentadas las ventajas y desventajas de los contenedores Kubernetes, Docker Swarm y Nomad.

- **Kubernetes.**

Tabla 4. Ventajas y desventajas de Kubernetes (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • Kubernetes tiene un enorme apoyo de la comunidad, convirtiéndose en la más popular para orquestar contenedores. • Kubernetes es una herramienta de código abierto y es compatible con diversos sistemas operativos. • Kubernetes ofrece una organización de servicio fácil con pods. 	<ul style="list-style-type: none"> • La instalación de Kubernetes puede ser un poco compleja. • En Kubernetes, se requiere tener un conjunto separado de herramientas para la administración, incluida la CLI de kubectl. • Es incompatible con las herramientas existentes de Docker CLI y Docker Compose.

- **Docker Swarm.**

Tabla 5. Ventajas y desventajas de Docker Swarm (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • Docker Swarm es una herramienta fácil de instalar con una configuración rápida. • Docker Swarm se integra sin problemas con Docker Compose y Docker CLI. Eso es porque estas son herramientas nativas de Docker. La mayoría de los comandos de Docker CLI funcionarán con Docker Swarm. 	<ul style="list-style-type: none"> • Docker Swarm ofrece una funcionalidad limitada en comparación a otras herramientas competidoras. • Docker Swarm no tiene tanto apoyo por parte de la comunidad, en comparación con la comunidad de Kubernetes. • Docker Swarm es poco permisivo a las fallas, lo que conlleva a posibles problemas durante el despliegue de aplicaciones.

- **Nomad.**

Tabla 6. Ventajas y desventajas de Nomad (elaboración propia).

Ventajas	Desventajas
<ul style="list-style-type: none"> • Nomad permite el despliegue tanto de contenedores docker, como aplicaciones java y VMS. • Nomad es una herramienta de código abierto y es compatible con diversos sistemas operativos. 	<ul style="list-style-type: none"> • No tiene la capacidad para poder funcionar como balanceador de carga, para las aplicaciones desplegadas en un clúster de contenedores. • No está capacitado para poder funcionar como proxy, para poder realizar la redirección de peticiones, en una aplicación desplegada en un clúster de contenedores. • No tiene la disposición para la gestión de secretos de una orquestación de contenedores Docker.

Estas ventajas y desventajas son tomadas de referencia de la documentación oficial de Docker Swarm (Docker docs, 2020), la documentación oficial de kubernetes (Kubernetes, 2020), la documentación oficial de Nomad (Nomad, 2017), artículos relacionados como la creación de un Cluster con Docker Swarm (Hernández A. , 2019), ¿Qué es un cluster de kubernetes? (RedHat, 2020) y de algunos sitios web únicamente como criterios.

Para realizar la selección de que herramienta se va a emplear en el proceso operacional se debe tener en cuenta las necesidades de la organización.

Docker Swarm es una herramienta muy buena si desea una configuración rápida, además, no tiene requisitos de configuración exhaustivos, con una instalación bastante simplificada y

una curva de aprendizaje gradual, aunque le faltan algunas características que limitan su funcionalidad y robustez.

Con las ventajas y desventajas definidas, el autor del libro seleccionó como herramienta de orquestación de contenedores a Kubernetes y hará uso de esta para el proceso operacional. La decisión para seleccionar esta herramienta se debe al crecimiento y aceptación que ha tenido, además, la comunidad ha crecido de manera considerable. Google y Red Hat son quienes más contribuyen, pero también están Meteor, CoreOS y Huawei.

5 Diseño

En el capítulo 4 se consolidó la herramienta de empaquetado de software Docker y motor de orquestación Kubernetes. En este capítulo se detalla la forma en que el autor del libro realizó el proceso operacional utilizando estas herramientas.

Para este diseño se llevará a cabo la creación y el despliegue de contenedores de manera satisfactoria. Antes de iniciar con la creación de contenedores es importante tener en consideración los siguientes términos.

5.1 Términos para tener en cuenta.

5.1.1 Imagen.

Una imagen se puede considerar como una plantilla de solo lectura, la cual contiene ciertas instrucciones que permiten la creación de un contenedor Docker.

Para la construcción de una imagen Docker propia, se crea un Dockerfile con una sintaxis simple para definir los pasos necesarios para crear la imagen y ejecutarla. Cada instrucción en un Dockerfile crea una capa en la imagen. Cuando cambia el Dockerfile y reconstruye la imagen, solo se reconstruyen las capas que han cambiado. Esto es parte de lo que hace que las imágenes sean tan ligeras, pequeñas y rápidas en comparación con otras tecnologías de virtualización (Docker docs, 2020).

5.1.2 Dockerfile.

Un Dockerfile es un archivo de configuración, como se puede evidenciar en la ilustración 8 (elaboración propia), el cual contiene comandos para montar una imagen.

5.1.2.1 Estructura de un Dockerfile.

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get -y install apache2
EXPOSE 81
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Ilustración 8. Estructura Dockerfile (elaboración propia).

- **FROM.**

Definir una imagen base para crear nuestra imagen con Dockerfile.

- **RUN.**

Permite la ejecución de comandos en la imagen base antes de ser creada.

- **ADD/COPY.**

Permite agregar o copiar archivos en el equipo local a la imagen.

- **EXPOSE.**

Permite exponer por defecto un puerto para el contenedor.

- **CMD.**

Permite ejecutar comando por defecto al crear el contenedor, ejecutando la finalidad de este.

Uno de problemas más comunes que hay a la hora de desarrollar, es utilizar las mismas versiones de las herramientas para el desarrollo de aplicaciones, para esto los contenedores permiten manejar versiones más específicas con ayuda de un TAG, los TAG también

llamadas etiquetas se pueden utilizar para obtener versiones de imágenes Docker más específicas. Como se ve en la ilustración 8, se define como imagen base la imagen de Ubuntu, utilizando un TAG, afirmando que se desea utilizar la versión de Ubuntu 16.04. El comando

```
docker build -t <Nombre-Imagen>:<Tag>
```

La bandera *-t* permite etiquetar la imagen con un nombre y una versión específica.

La ilustración 9 (Goig, 2016), muestra el flujo de construcción de un contenedor docker.



Ilustración 9. Flujo de construcción de un contenedor (Goig, 2016).

5.2 Proceso

Para el diseño se plantea realizar la creación de contenedores a partir de imágenes oficiales, adquiridas desde el repositorio de imágenes, Docker Hub, que consta de dos contenedores conectados entre sí, permitiendo la obtención de información contenida en ellos.

La herramienta Docker permite su ejecución en diversas plataformas, como Windows, MacOS o distribuciones de GNU/Linux, la cual es de libre elección. Para realizar dicha

instalación, el autor del libro recomienda la guía de instalación de la documentación oficial de Docker (Docker docs, 2020).

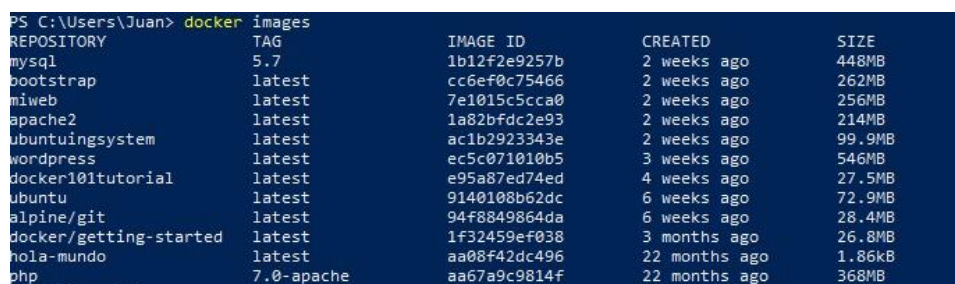
5.2.3 Adquisición de imagen.

Una de las grandes características con la que cuenta Docker es su comunidad, la cual está en constante crecimiento, Docker cuenta con un repositorio con miles de imágenes que pueden ser utilizadas libremente. Las imágenes que se utilizarán, serán imágenes oficiales desde el repositorio de imágenes Docker Hub (DockerHub, 2020).

El comando para obtener imágenes oficiales desde Docker Hub, es el siguiente.

```
docker pull <Nombre Imagen>:<Tag>
```

Con el comando *docker images*, como se evidencia en la ilustración 10 (elaboración propia), permite ver las imágenes guardadas en el host, para ser utilizadas.



REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql	5.7	1b12f2e9257b	2 weeks ago	448MB
bootstrap	latest	cc6ef0c75466	2 weeks ago	262MB
miweb	latest	7e1015c5cca0	2 weeks ago	256MB
apache2	latest	1a82bfdc2e93	2 weeks ago	214MB
ubuntuingsystem	latest	ac1b2923343e	2 weeks ago	99.9MB
wordpress	latest	ec5c071010b5	3 weeks ago	546MB
docker101tutorial1	latest	e95a87ed74ed	4 weeks ago	27.5MB
ubuntu	latest	9140108b62dc	6 weeks ago	72.9MB
alpine/git	latest	94f8849864da	6 weeks ago	28.4MB
docker/getting-started	latest	1f32459ef038	3 months ago	26.8MB
hola-mundo	latest	aa08f42dc496	22 months ago	1.86kB
php	7.0-apache	aa67a9c9814f	22 months ago	368MB

Ilustración 10. Docker images (elaboración propia).

5.2.4 Ejecución.

Al momento de la creación de un contenedor, se debe tener en cuenta los volúmenes, los volúmenes permiten tener datos persistentes, al tener un contenedor con un servicio específico, por ejemplo, MySQL y por algún motivo este contenedor falla, es fundamental

que la información en este contenedor no se pierda. Aquí es donde entran los volúmenes, que permiten asociar información del contenedor en la maquina host.

Para la ejecución de un contenedor asignado un volumen, se debe ejecutar el siguiente comando.

```
docker run -p <Puerto-host>:<Puerto-Container> --name <Nombre Container> -v  
<host-directorio>:<Container-directorio> -d <Imagen-Base>
```

- **-p.**

da acceso al puerto del contenedor, la primera sección *<Puerto-host>*, es el puerto por el cual se tendrá acceso desde la maquina host, la segunda sección *<Puerto-Container>*, es el puerto al que estará conectado el contenedor.

- **--name.**

Permite dar un nombre al contenedor, se recomienda dar este nombre con relación al servicio por el que es creado.

- **-v.**

Permite creación de volúmenes para tener persistencia de datos.

- **-d.**

Permite que la imagen base se ejecute en segundo plano.

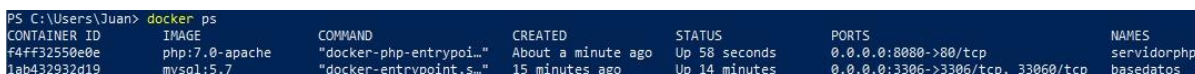
5.2.5 Conexión.

La conexión entre dos contenedores es muy útil ya que en ocasiones se requiere información de un contenedor a otro. Para esto se requiere de una bandera llamada link, la

cual permite conectar dos contenedores. Al momento de crear un contenedor que requiera conexión con otro contenedor, se recomienda emplear la bandera `link`, de la siguiente manera:

```
docker run -p <Puerto-host>:<Puerto-Container> --name <Nombre-Container> -v
<host-directorio>:<Container-directorio> -d --link <Container-a-Conectar> <Imagen-
Base>
```

El comando `docker ps`, permite ver los contenedores que están en ejecución. Si a este comando se le añade la bandera `-a`, que en definición es: `docker ps -a`, como se evidencia en la ilustración 11 (elaboración propia), permite ver todos los contenedores, incluyendo los contenedores detenidos.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
f4ff32550a0e	php:7.0-apache	"docker-php-entrypoi..."	About a minute ago	Up 58 seconds	0.0.0.0:8080->80/tcp	servidorphp
1ab432932d19	mysql:5.7	"docker-entrypoint.s..."	15 minutes ago	Up 14 minutes	0.0.0.0:3306->3306/tcp, 33060/tcp	basedatos

Ilustración 11. Contenedores en ejecución (elaboración propia).

Para detener un contenedor en ejecución el comando `docker stop <Container-ID>`, permite hacerlo.

El comando `docker start <Container-ID>`, permite iniciar un contenedor detenido.

El comando `docker inspect <Container-ID>`, permite inspeccionar un contenedor mostrando toda su configuración.

El comando `docker rm <Container-ID>`, permite eliminar un contenedor detenido.

El comando *docker rmi <Imagen>*, permite eliminar una imagen, siempre y cuando un contenedor, no dependa de esta.

5.2.6 Modificación de los contenedores.

Para tener acceso a los contenedores se debe ejecutar el comando *exec* el cual permite ejecutar comando en contenedores ya iniciados.

```
docker exec -i -t <Container> /bin/bash
```

- **-i.**

Mantiene STDIN (Standar Input) abierto, lo que permite ingresar contenido dentro del contenedor.

- **-t.**

Asigna un pseudo-TTY. Permite tener acceso a una terminal.

Este comando permite tener acceso al contenedor en una terminal bash; al tener este acceso, se puede modificar, crear, eliminar archivos e instalar dependencias que se requieran al momento de desarrollar aplicaciones.

El comando *docker restart <Container>*, el cual permite reiniciar un contenedor.

5.2.7 Conexión entre contenedores por red.

Docker permite conectar contenedores mediante redes. Por defecto docker crea redes para conectar estos como se puede evidenciar en la ilustración 12(elaboración propia), el comando *docker network ls*, permite ver las redes existentes.

NETWORK ID	NAME	DRIVER	SCOPE
6e763d91ff6d	bridge	bridge	local
3292e4aaf222	host	host	local
0fbe4b9074ce	none	null	local

Ilustración 12. Redes creadas por Docker (elaboración propia).

- **bridge.**

Esta es una red que está en desuso, por lo cual no será utilizada.

- **host.**

Esta es la red de la máquina host, con lo cual es peligroso utilizarla, ya que el contenedor queda expuesto a toda la red local.

- **none.**

Esta es una red que hace que los contenedores conectados a ella estén totalmente aislados se utiliza en contenedores que manejan procesos sensibles, para que no puedan ni comunicarse ni recibir datos con el exterior.

El comando `docker network create <Nombre de la red>` permite la creación de una nueva red. Docker permite inspeccionar las redes creadas, el comando `docker network inspect <Nombre de la red>`, mostrará información relevante de esta red.

Para conectar contenedores por medio de red, se ejecuta el siguiente comando.

```
docker network connect <Nombre de la red> <Nombre del contenedor>
```

5.2.8 Clonar un contenedor.

En muchas ocasiones es necesario modificar información que contiene un contenedor, pero esta tarea puede ser algo dispendiosa y con altas posibilidades de cometer errores, puede

llevar a que un contenedor termine no funcionando de manera satisfactoria. Docker permite crear una imagen a partir de un contenedor, conteniendo las mismas modificaciones y características que el contenedor original, el siguiente comando:

```
docker commit -m <Comentario> -a <Autor> <ID-contenedor-a-clonar> <Nombre-que-tendrá-la-imagen>
```

Permite crear una imagen a partir de un contenedor, con el cual puede clonar este contenedor con toda su configuración almacenada. El comando *Docker images* permite visualizar las imágenes creadas; con la obtención de esta segunda imagen, es punto de partida para la creación de nuevos contenedores con las mismas configuraciones.

5.2.9 Portabilidad.

Una de las ventajas del uso de los contenedores es su portabilidad, al no requerir muchos recursos de su máquina anfitriona, puede ser trasladado de una máquina a otra. Para esto, se emplea el repositorio de Docker Hub, el cual permite subir imágenes personalizadas, ya sea, un repositorio público o un repositorio privado.

Para subir archivos a este gestor de repositorios, se deben llevar a cabo los siguientes pasos.

- Se debe crear un repositorio, en la página oficial de Docker Hub (DockerHub, 2020).
- Haber iniciado sesión desde consola en la máquina host, con el siguiente comando.

```
Docker login -u <usuario>.
```

- Las imágenes docker deben tener cierta estructura en el espacio de nombres para ser compartida correctamente en Docker Hub. Se deben nombrar de la siguiente manera:
<Docker-ID>/<Nombre-Repository>:<Tag>.
- El comando *docker push <Imagen>*, permite la publicación de la imagen en Docker Hub. Los repositorios son públicos de forma predeterminada.

5.2.10 Orquestación.

Uno de los problemas más grandes que hay a la hora de desplegar contenedores, es cuando se empieza a correr contenedores a gran escala, el desplegar un gran número de contenedores de manera manual, puede llegar a hacer algo dispendioso. Aquí es donde entra Kubernetes, un orquestador de contenedores, encargado de gestionar el ciclo de vida de los contenedores de una aplicación.

Para orquestar contenedores, kubernetes lo realiza por medio de un clúster. Un clúster de kubernetes es un conjunto de máquinas, de nodos que ejecutan aplicaciones en contenedores. Un clúster posee un nodo de trabajo y un nodo maestro como mínimo. El nodo maestro es el encargado de mantener y controlar las aplicaciones, las imágenes y los contenedores que se ejecutan. Los nodos de trabajo son los que ejecutan las aplicaciones y las tareas.

Un clúster de Kubernetes tiene un estado deseado, que define las aplicaciones o cargas de trabajo que deben ejecutarse, las imágenes que se utilizan, los recursos que deben estar disponibles y otros detalles de configuración.

El estado deseado se define a través de archivos de configuración compuestos por manifiestos. Se trata de archivos JSON o YAML que indican el tipo de aplicación que se

ejecutará y la cantidad de réplicas necesarias para que un sistema funcione en buenas condiciones.

El estado deseado del clúster se define con la API de Kubernetes. Esto puede llevarse a cabo desde la línea de comandos (con kubectl) o usando la API para interactuar con el clúster y establecer o modificar el estado deseado.

Kubernetes gestionará el clúster de forma automática para que coincida con dicho estado. Por ejemplo, al implementar una aplicación con un estado deseado de "tres". Esto quiere decir que deberían ejecutarse tres réplicas de la aplicación. Si uno de los contenedores falla, Kubernetes notará que solo hay dos réplicas en ejecución y agregará una para lograr el estado deseado (RedHat, 2020).

- **Pod.**

Un pod es la unidad de cómputo más pequeña de kubernetes, como se evidencia en la ilustración 13 (Melendez, 2019), agrupa uno más contenedores, compartiendo la red.

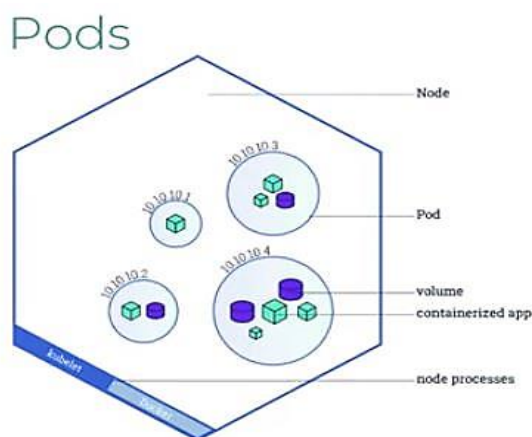


Ilustración 13. Estructura de pods (Melendez, 2019).

- **Deployments.**

Un deployment describe un estado deseado en una implementación. La ilustración 14 (Melendez, 2019), muestra la estructura que contiene un Deployment, el cual puede definir implementaciones para crear nuevos ReplicaSets, o para eliminar implementaciones existentes y adoptar todos sus recursos con nuevas implementaciones (Kubernetes, 2020).

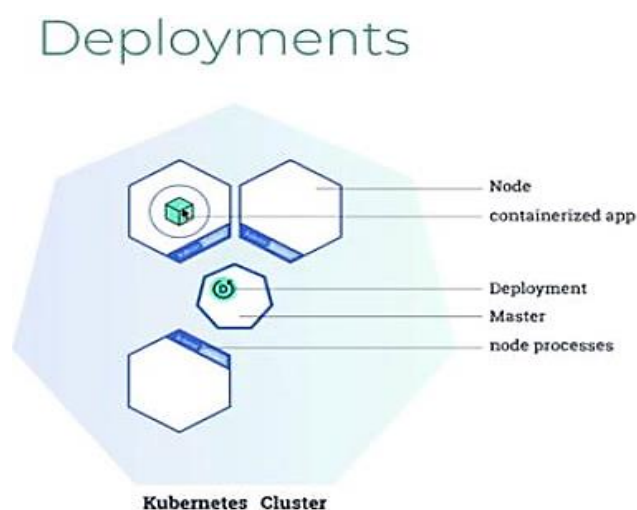


Ilustración 14. Estructura Deployments (Melendez, 2019).

- **Service.**

Una forma abstracta de exponer una aplicación que se ejecuta en un conjunto de Vainas (envoltura) como un servicio de red, como se evidencia en la ilustración 15 (Melendez, 2019). Con Kubernetes, no necesita modificar su aplicación para usar un mecanismo de descubrimiento de servicios desconocido. Kubernetes les da a los pods sus propias direcciones IP y un solo nombre DNS para un conjunto de pods, y puede equilibrar la carga entre ellos.

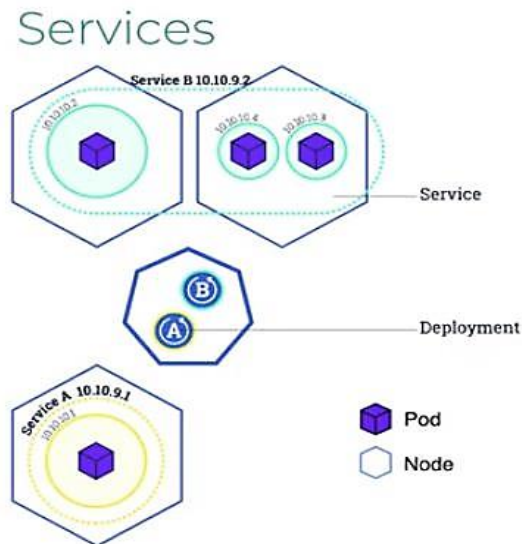


Ilustración 15. Estructura Service (Melendez, 2019).

En la ilustración 16 hace representación a un proceso operacional para el despliegue de aplicaciones basada en contenedores Docker.

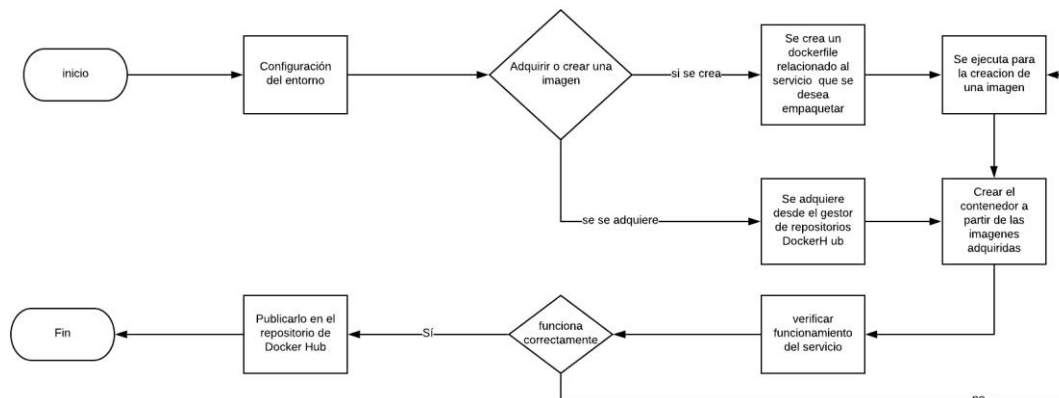


Ilustración 16 Representación gráfica de despliegue de contenedores (elaboración propia).

6 Validación.

Para la validación se planea realizar la creación de contenedores a partir de imágenes, que consta de un contenedor con una base de datos en MySQL, otro contenedor con PHP/Apache, conectar los contenedores entre sí con una propiedad llamada link y realizar un despliegue satisfactorio.

El sistema operativo que se utilizó, para poder realizar la creación y despliegue de estos contenedores, es el sistema operativo Windows 10, en el que previamente debe tener instalado Docker (Docker docs, 2020). También se puede utilizar otro sistema operativo como MacOS o una distribución GNU/Linux.

En el siguiente apartado, se explicará la creación del escenario descrito.

6.1 Adquisición de imagen.

Docker Hub cuenta con miles de imágenes para ser utilizadas libremente, la Ilustración 17. Docker Hub . (DockerHub, 2020), muestra el portal de bienvenida del gestor de repositorios de Docker Hub. Para este escenario, se adquieren imágenes oficiales de MySQL y PHP.



Ilustración 17. Docker Hub (DockerHub, 2020).

Como se evidencia en la Ilustración 18. Imagen oficial de MySQL de Docker Hub. (DockerHub, 2020), y Ilustración 19. Imagen oficial de PHP. (DockerHub, 2020), Docker Hub brinda imágenes oficiales para obtener las siguientes imágenes. La imagen de MySQL con el comando:

```
docker pull mysql:5.7
```

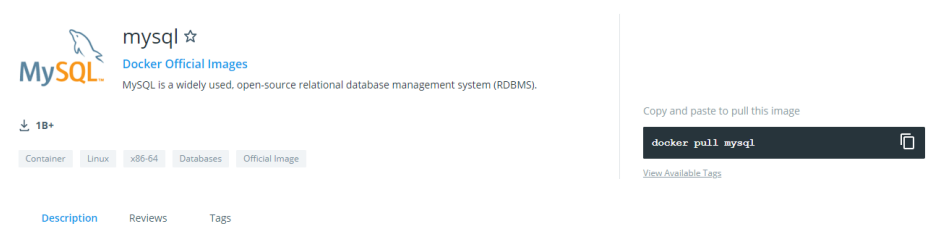


Ilustración 18. Imagen oficial de MySQL de Docker Hub (DockerHub, 2020).

Se debe utilizar una TAG para obtener la versión de MySQL 5.7

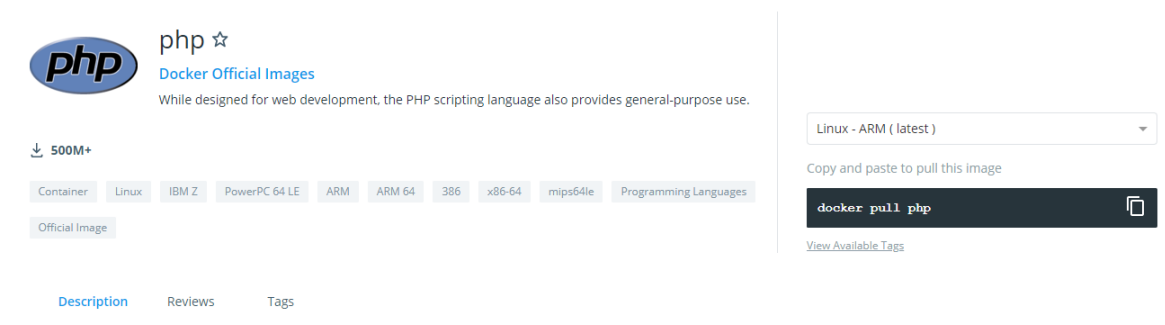


Ilustración 19. Imagen oficial de PHP (DockerHub, 2020).

El comando permite obtener la imagen oficial de MySQL.

```
docker pull php:7.0-apache
```

Se debe utilizar una TAG para obtener la versión de PHP 7.0 con apache. Con el comando `docker images`, como se evidencia en la Ilustración 20. Docker images (elaboración propia).(elaboración propia), verá las imágenes guardadas en el host, para ser utilizadas.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mysql	5.7	1b12f2e9257b	2 weeks ago	448MB
bootstrap	latest	cc6ef0c75466	2 weeks ago	262MB
miweb	latest	7e1015c5cca0	2 weeks ago	256MB
apache2	latest	1a82bfdc2e93	2 weeks ago	214MB
ubuntuingsystem	latest	ac1b2923343e	2 weeks ago	99.9MB
wordpress	latest	ec5c071010b5	3 weeks ago	546MB
docker101tutorial	latest	e95a87ed74ed	4 weeks ago	27.5MB
ubuntu	latest	9140108b62dc	6 weeks ago	72.9MB
alpine/git	latest	94f8849864da	6 weeks ago	28.4MB
docker/getting-started	latest	1f32459ef038	3 months ago	26.8MB
hola-mundo	latest	aa08f42dc496	22 months ago	1.86kB
php	7.0-apache	aa67a9c9814f	22 months ago	368MB

Ilustración 20. Docker images (elaboración propia).

6.2 Ejecución.

Para la ejecución de un contenedor con MySQL, se debe ejecutar el siguiente comando.

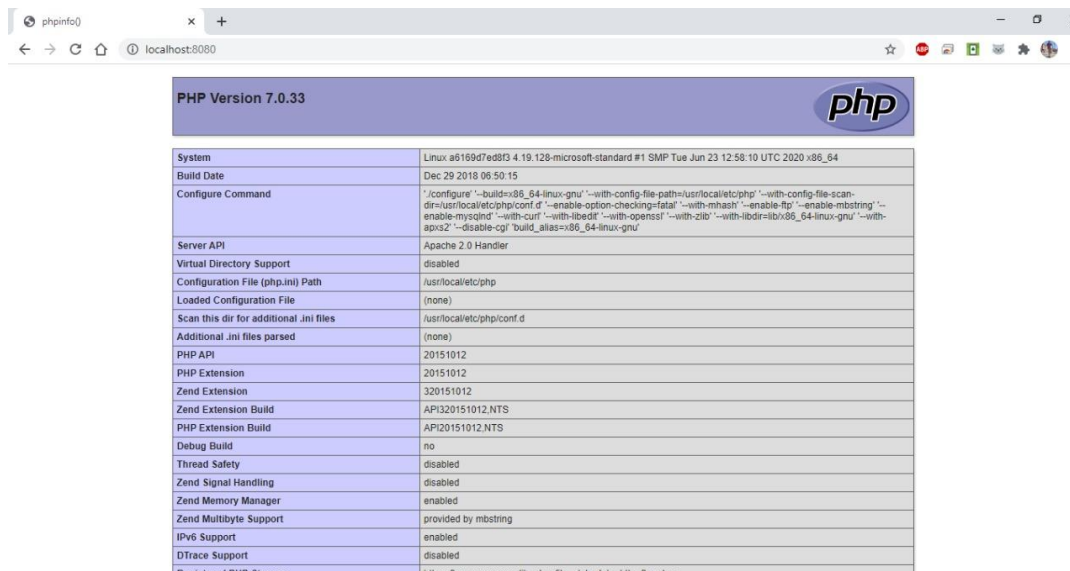
```
docker run -p 3306:3306 --name basedatos -v C:\Users\Juan\dockerServicio\database:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=mipassword -d mysql:5.7
```

6.3 Conexión.

Para este escenario se dispone a la creación de un contenedor levantando el servicio PHP/Apache, el cual se conectará con el contenedor de MySQL creado anteriormente, utilizando el siguiente comando.

```
docker run -p 8080:80 -v C:\Users\Juan\dockerServicio:/var/www/html --name servidorphp -d --link basedatos php:7.0-apache
```

Para comprobar si esta funcionando de manera satisfactoria, como se evidencia en la, se realiza por medio del navegador en la siguiente ruta: localhost:8080.



PHP Version 7.0.33	
System	Linux a6169d7ed8f3 4.19.128-microsoft-standard #1 SMP Tue Jun 23 12:58:10 UTC 2020 x86_64
Build Date	Dec 29 2018 06:50:15
Configure Command	'configure' '--build=x86_64-linux-gnu' '--with-config-file-path=/usr/local/etc/php' '--with-config-file-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-checking=fatal' '--with-mhash' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-curl' '--with-libedit' '--with-openssl' '--with-zlib' '--with-libdir=libx86_64-linux-gnu' '--with-apxs2' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/etc/php
Loaded Configuration File	(none)
Scan this dir for additional .ini files	/usr/local/etc/php/conf.d
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012.NTS
PHP Extension Build	API20151012.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	https, ftps, compress.zlib, odbc, file, dlob, data, http, ftp, ohar

Ilustración 21. Verificación del funcionamiento de los contenedores (elaboración propia).

6.4 Modificación de los contenedores.

Para tener acceso a los contenedores se debe ejecutar el comando `exec` el cual permite ejecutar comando en contenedores ya iniciados.

```
docker exec -i -t basedatos /bin/bash
```

El cual permite tener acceso al contenedor en una terminal `bash`. Al obtener acceso al contenedor de MySQL, se debe crear una base de datos para el manejo de usuarios desde la página web, con el servicio de apache. Se accede al contenedor de PHP/Apache para agregar librerías que permitan tener acceso a la base de datos, el comando:

```
docker exec -i -t servidorphp /bin/bash
```

Permite tener acceso al contenedor, para instalar la extensión de MySQL con el siguiente comando.

```
Docker-php-ext-install mysqli
```

Una vez dentro del contenedor, se agregará la siguiente extensión a los archivos *php.ini-development* y *php.ini-production* ubicados en la siguiente ruta */usr/local/etc/php*. Dicha extensión es:

```
extension=/usr/local/lib/php/extensions/no-debug-non-zts-20151012/mysqli.so
```

La cual brindará conexión a la base de datos, con un archivo conexión, como se evidencia en la Ilustración 22. Conexión base de datos (elaboración propia). Posteriormente se requiere el reinicio del contenedor PHP/Apache, para ello se debe ejecutar el siguiente comando.

```
docker restart servidorphp
```

```
<?php
//create connection aborted
$conn=mysqli_connect('basedatos:3306','root','sistemas','usuarios');

//check connection
if (!$conn){
    die("connection failed: " .mysqli_connect_error());
}
echo "Connected successfull"
?>
```

Ilustración 22. Conexión base de datos (elaboración propia).

6.5 Verificación de funcionamiento.

Para realizar la verificación del funcionamiento de los contenedores, se puede hacer desde el navegador, accediendo al *localhost* en el puerto 8080, como se evidencia en la Ilustración 23. Verificación de funcionamiento de los contenedores (elaboración propia).

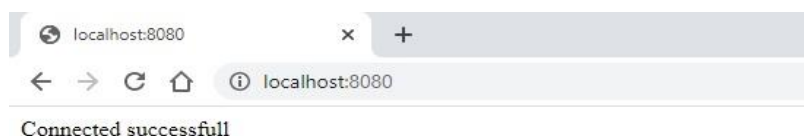


Ilustración 23. Verificación de funcionamiento de los contenedores (elaboración propia).

Al agregar contenido php a la carpeta asociada al contenedor, se puede visualizar en el navegador, como se evidencia en la Ilustración 24. Contenido control de usuarios (elaboración propia), para este escenario se plasmará un sistema de control de usuario, el cual estará conectado al contenedor con la base de datos.

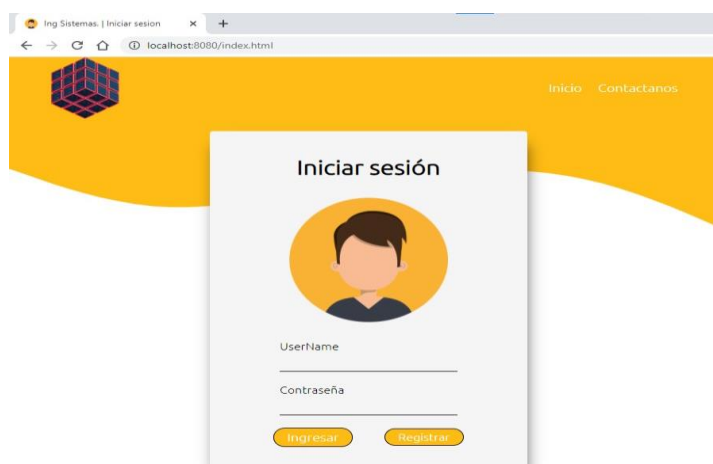


Ilustración 24. Contenido control de usuarios (elaboración propia)

Al registrar un usuario desde el entorno gráfico, se puede ver reflejada la inserción de usuarios a la base de datos, como se muestra en la Ilustración 25. Registro de usuarios (elaboración propia).

```
mysql> select * from clientes;
+-----+-----+-----+-----+
| username | nombre | correo | contra |
+-----+-----+-----+-----+
| IngSistemas | juan | juan@gmail.com | juan |
| sebastian | Sebastian | sebastian@gamil.com | sebastian |
+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

Ilustración 25. Registro de usuarios (elaboración propia).

Como se ve en la Ilustración 26. Estructura de desarrollo (elaboración propia), se evidencia la estructura de los contenedores que se ha creado anteriormente. En el primer contenedor empaqueta el servicio de Apache/PHP, el cual se conecta a un segundo contenedor que contiene el servicio de MySQL.

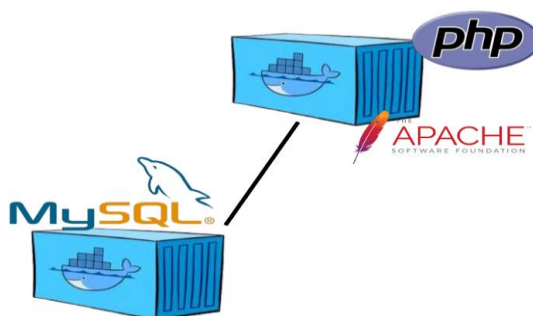


Ilustración 26. Estructura de desarrollo (elaboración propia)

6.6 Conexión entre contenedores por red.

El comando `docker network create networkservices`, permite la creación de una nueva red. Para este escenario se crea una red que permita la conexión entre dos contenedores, como se puede ver en la Ilustración 27. Nueva network (elaboración propia).

NETWORK ID	NAME	DRIVER	SCOPE
38bdfdb521e9	bridge	bridge	local
b3429559844b	dockerwordpress_default	bridge	local
d9ccddc2e058	host	host	local
b134d70aefb5	networkservice	bridge	local
4cc8aa6f0410	none	null	local

Ilustración 27. Nueva network (elaboración propia).

Docker permite inspeccionar las redes creadas, el comando `docker network inspect <Nombre de la red>`, mostrará información relevante de esta red, como se evidencia en la Ilustración 28. Inspección de red (elaboración propia), de la siguiente manera `docker network inspect networkservice`.

```
[
  {
    "Name": "networkservice",
    "Id": "b134d70aefb5057b319233d1f83644e0510e66fb36f1b47e4353419f96f8d3fd",
    "Created": "2020-11-10T15:19:00.6460526Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {},
    "Labels": {}
  }
]
```

Ilustración 28. Inspección de red (elaboración propia).

Para conectar contenedores por medio de esta network, se puede realizar gracias al siguiente comando.

```
docker network connect network basedatos
```

Conectando el contenedor *basedatos* a esta red, para conectar en contenedor *servidorphp*, se hace de la misma manera.

```
docker network connect network servidorphp
```

Al realizar de nuevo la inspección de la red creada, se puede evidenciar, en la *key containers*, cómo se evidencia en la Ilustración 29. Inspección network conectado a contenedores (elaboración propia), la conexión con estos contenedores.

```
[
  {
    "Name": "networkservice",
    "Id": "b134d70aefb5057b319233d1f83644e0510e66fb36f1b47e4353419f96f8d3fd",
    "Created": "2020-11-10T15:19:00.6460526Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": {},
      "Config": [
        {
          "Subnet": "172.19.0.0/16",
          "Gateway": "172.19.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "687bdcca027660c0112ad37619cadb3b0d71c-fbf21e71f6f58a6af097eaba6e9": {
        "Name": "basedatos",
        "EndpointID": "866c1cc7eaf97c3f446860604f314228f2d0ab2ac3f38407a46f5d8765b5521e",
        "MacAddress": "02:42:ac:13:00:02",
        "IPv4Address": "172.19.0.2/16",
        "IPv6Address": ""
      },
      "a6169d7ed8f306ab3e213645fb772aee5385de03ad1857aa7aec02aed6492e68": {
        "Name": "servidorphp",
        "EndpointID": "fa828cc718dc39f33e5a2340673afb1311cbc425149770142c4699b6feabc735",
        "MacAddress": "02:42:ac:13:00:03",
        "IPv4Address": "172.19.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
```

Ilustración 29. Inspección network conectado a contenedores (elaboración propia).

Para evidenciar el funcionamiento de estos contenedores, conectados a esta red, se hace por medio del Gateway de esta network, como se evidencia en la Ilustración 30. Verificación de conexión de red (elaboración propia), verificado desde el navegador.

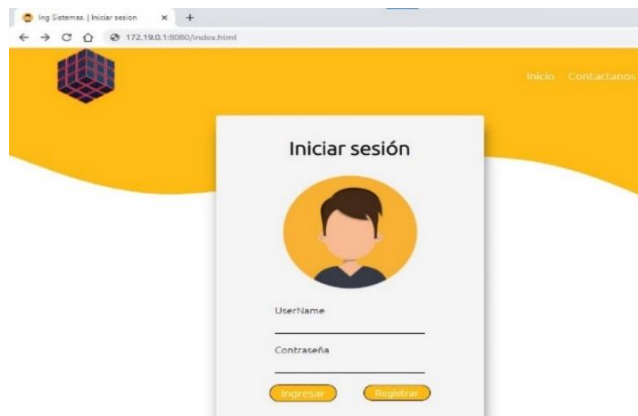


Ilustración 30. Verificación de conexión de red (elaboración propia)

6.7 Clonar un contenedor.

Docker permite crear una imagen a partir de un contenedor, como se ha visto en los escenarios anteriores, se ha modificado ciertos archivos del contenedor PHP/Apache, si se desea crear otro contenedor con las mismas modificaciones se puede realizar de la siguiente manera.

```
docker commit -m 'PHP/Apache con extensión MySQL' -a 'Juan' a6169d7
phpachemysql
```

Permitiendo clonar el contenedor con toda su configuración almacenada. Para este escenario se creará una imagen a partir del contenedor de PHP/Apache creado anteriormente, se puede revisar con el comando *docker images*.

Partiendo de esta imagen se crea un nuevo contenedor para este servicio, con un nuevo puerto de escucha, ya que el contenedor original ya lo tiene en uso; este nuevo servicio tendrá otro volumen asociado a este contenedor y se conectará al contenedor que tiene el servicio de MySQL:

```
docker run -p 8081:80 -v C:\Users\Juan\dockerServicioClon:/var/www/html --name
servidorphpclone -d --link basedatos phppachemysql
```

Si la creación del contenedor es satisfactoria, se puede copiar la página web del contenedor original, verificando en el navegador con el puerto que se le asignó al nuevo contenedor; si es correcto su funcionamiento, en el navegador dará como resultado la misma página que el escenario anterior, que tiene acceso al contenedor con la base de datos, cómo se evidencia en la Ilustración 31. Estructura de desarrollo masivo (elaboración propia), esta sería la nueva estructura de los contenedores creados.

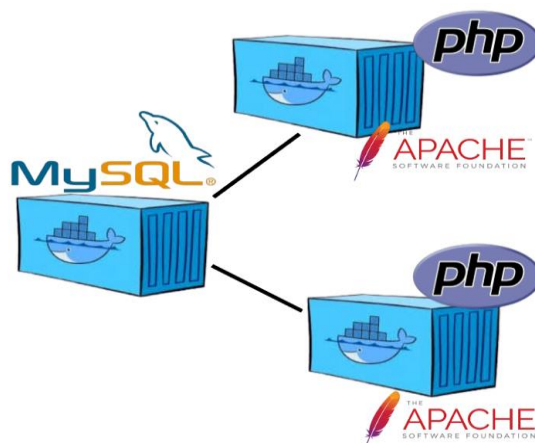


Ilustración 31. Estructura de desarrollo masivo (elaboración propia)

6.8 Portabilidad.

Para compartir una imagen con otros usuarios, se realiza a través de Docker Hub, el cual permite subir imágenes personalizadas, ya sea, un repositorio público o un repositorio privado. Antes de compartir una imagen, se requiere iniciar sesión, desde la consola con el comando `docker login -u <usuario>`, posteriormente pide la contraseña de Docker Hub.

Para este escenario, se crea una imagen, a partir de un contenedor, estructurando el espacio de nombres, de la siguiente manera:

```
docker commit -m 'PHP/Apache adding MySQL extension'-a 'Juan' a6169d7
usuario/repositorio
```

Como se evidencia en la Ilustración 32. Repositorio de Docker Hub., este es la etiqueta creada en el repositorio, después de publicarla.


Tags and Scans		VULNERABILITY SCANNING - DISABLED Enable	
This repository contains 1 tag(s).			
TAG	OS	PULLED	PUSHED
■ latest		a minute ago	a minute ago
See all			

Ilustración 32. Repositorio de Docker Hub (DockerHub, 2020).

6.9 Orquestación.

Para este escenario, se emplea el despliegue de una aplicación basada en microservicios, empleando kubernetes. El cual se encarga del escalamiento, recuperación automática, balanceo de cargas y despliegue de contenedores. Para este escenario se va a realizar un despliegue de aplicaciones sobre un clúster de Kubernetes de manera local, esto gracias a Minikube.

Minikube es una herramienta que despliega un clúster de kubernetes con un único nodo en una máquina virtual. Esta herramienta es compatible con diferentes tipos de sistemas operativos. Así mismo se requiere la instalación de *kubectctl*, que permite la comunicación con el servidor de kubernetes, el autor del libro recomienda la documentación oficial (Kubernetes, 2020) , para la instalación de dichas herramientas, para el funcionamiento correcto de Minikube, requiere la instalación de VirtualBox o Hyper-v.

Una vez instalado las herramientas ya mencionadas, se inicia la herramienta Minikube con el siguiente comando: *minukube start*, para comprobar que estas herramientas se han instalado correctamente, se utiliza el comando *kubectl get services*, que permite listar los servicios en ejecución, como se evidencia en la Ilustración 33. Lista de servicios (elaboración propia).

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	4m14s

Ilustración 33. Lista de servicios (elaboración propia).

El manejo de Kubernetes normalmente es por comandos, aunque incluye un dashboard por defecto para manejarlo, cómo se evidencia en la Ilustración 34. Dashboard kubernetes

(elaboración propia)., que se puede ejecutar con el siguiente comando: *minikube*

dashboard

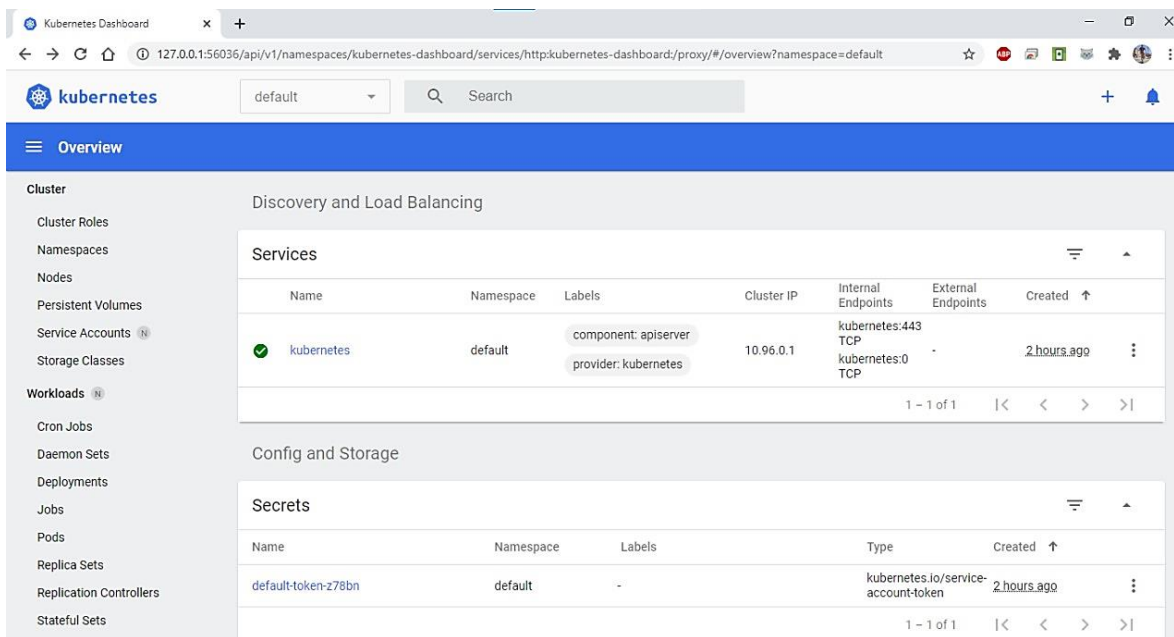


Ilustración 34. Dashboard kubernetes (elaboración propia).

Kubernetes permite la creación de Pods, deployment y servicios en un archivo YAML de la siguiente forma.

Para la creación de un Pod, cómo se evidencia en la Ilustración 35. Archivo YAML (elaboración propia)., es la estructura que debe tener un archivo YAML, para la creación de este.

```

apiVersion: v1
kind: Pod
metadata:
  name: api-hello-pod
  labels:
    app: api-hello
spec:
  containers:
  - name: api-hello
    image: christianhxc/api-hello:latest
    ports:
    - containerPort: 8082

```

Ilustración 35. Archivo YAML (elaboración propia).

- **apiVersion.**

Número de versión del api que se va a emplear.

- **kind.**

Tipo de fichero que se va a crear.

- **metadata.**

Contiene los datos propios del pod, como el nombre y labels que tienen asociados para seleccionarlo.

- **labels.**

Se especifica que el pod tenga un label con clave y valor.

- **spec.**

Contiene la especificación del pod.

- **containers.**

Nombra los contenedores que forman parte de este pod.

- **restartPolicy.**

Define la política de restauración en caso de que el pod se detenga o deje de ejecutarse debido a un fallo interno.

El comando `kubectl apply -f k8s/pod.yaml`, permite la creación de este pod. El comando `kubectl get pods`, cómo se evidencia en la Ilustración 36. Pod creado (elaboración propia), permite ver los pods creados y su respectivo estado.

NAME	READY	STATUS	RESTARTS	AGE
api-hello-pod	1/1	Running	0	74s

Ilustración 36. Pod creado (elaboración propia).

Para eliminar este pod creado, se debe ejecutar el siguiente comando. `Kubectl delete pod api-hello-pod`.

Para la creación de un deployment, cómo se evidencia en la Ilustración 37. Estructura deployment (elaboración propia), es la estructura que debe tener un archivo YAML, para la creación de este.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-hello-deployment
  labels:
    app: api-hello
spec:
  replicas: 10
  selector:
    matchLabels:
      app: api-hello
  template:
    metadata:
      labels:
        app: api-hello
    spec:
      containers:
        - name: api-hello
          image: christianhxc/api-hello:1.0
          ports:
            - containerPort: 8080
```

Ilustración 37. Estructura deployment (elaboración propia)

- **replicas.**

Numero de réplicas del pod.

- **selector.**

Todo pod que contenga esta etiqueta y este value, lo va a incluir dentro del comportamiento del deployment.

Para la ejecución del deployment, se realiza con el mismo comando *kubectl apply -f k8s/deployment.yaml*, para verificar si su ejecución fue correcta, el comando *kubectl get all*, como se evidencia en la Ilustración 38. Deployment run (elaboración propia)., permite evidenciar los pod en ejecución, junto con las réplicas contenidas en su configuración.

```

NAME                                     READY   STATUS    RESTARTS   AGE
pod/api-hello-deployment-58966758f4-85g57 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-8nnbz 1/1     Running   0           51s
pod/api-hello-deployment-58966758f4-91pdp 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-c8zbv 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-fzpjg 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-h1qwn 1/1     Running   0           51s
pod/api-hello-deployment-58966758f4-1xzc8 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-nlbpm 1/1     Running   0           52s
pod/api-hello-deployment-58966758f4-v5t6c 1/1     Running   0           51s
pod/api-hello-deployment-58966758f4-wpxx4 1/1     Running   0           52s
pod/api-hello-pod                          1/1     Running   0           101m

NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP    10.96.0.1    <none>        443/TCP    5h41m

NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api-hello-deployment  10/10   10            10          53s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/api-hello-deployment-58966758f4  10        10        10     53s

```

Ilustración 38. Deployment run (elaboración propia).

Al eliminar un pod, cómo se evidencia en la Ilustración 39. Sustitución de un pod (elaboración propia)., con el comando *kubectl delete pod <pod-name>*, y listar de nuevo los pod en ejecución, se evidencia que hay un nuevo pod, que reemplaza al pod eliminado.


```

NAME                                READY   STATUS    RESTARTS   AGE
pod/api-hello-deployment-58966758f4-6t5kq  1/1     Running   0           20s
pod/api-hello-deployment-58966758f4-8nnbz  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-91pdp  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-c8zbv  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-fzpjg  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-h1qwn  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-lxzc8  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-nlbpm  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-v5t6c  1/1     Running   0           15m
pod/api-hello-deployment-58966758f4-wpxx4  1/1     Running   0           15m
pod/api-hello-pod                        1/1     Running   0           116m

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                   ClusterIP     10.96.0.1    <none>        443/TCP    5h56m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api-hello-deployment  10/10   10           10          15m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/api-hello-deployment-58966758f4  10       10       10     15m

```

Ilustración 39. Sustitución de un pod (elaboración propia).

La dashboard de kubernetes, cómo se evidencia en la Ilustración 40. Dashboard kubernetes (elaboración propia)., evidencia la creación de estos pods, con las réplicas contenidas en la configuración.

Name	Namespace	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
api-hello-deployment-58966758f4-6t5kq	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:
api-hello-deployment-58966758f4-8nnbz	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:
api-hello-deployment-58966758f4-v5t6c	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:
api-hello-deployment-58966758f4-h1qwn	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:
api-hello-deployment-58966758f4-c8zbv	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:
api-hello-deployment-58966758f4-fzpjg	default	app: api-hello pod-template-hash: 58966758f4	minikube	Running	0	-	-	:

Ilustración 40. Dashboard kubernetes (elaboración propia).

Al momento de crear un servicio, este debe tener cierta estructura para que se ejecute de manera satisfactoria, como se evidencia en la Ilustración 41. Estructura servicio (elaboración propia).

```
kind: Service
apiVersion: v1
metadata:
  name: api-hello-service
spec:
  type: LoadBalancer
  selector:
    app: api-hello
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Ilustración 41. Estructura servicio (elaboración propia).

Para la ejecución del servicio, se realiza con el comando `kubectl apply -f k8s/service.yaml`, para verificar si su ejecución fue correcta, el comando `kubectl get svc`, permite evidenciar los servicios en ejecución. Con ello se puede evidenciar como exponer servicios desde kubernetes.

7 Conclusiones

- Con base en los aspectos referentes al despliegue de aplicaciones, plasmados en este documento, se pudo establecer un diseño operacional para el despliegue de aplicaciones.
- Se implementan contenedores para facilitar la vida del desarrollador ya que, gracias a su simplicidad, ahorran mucho más tiempo y esfuerzo que al implementar máquinas virtuales.
- El estudio de las herramientas de empaquetado de software y herramientas de orquestación, determinó la selección de las herramientas Docker y Kubernetes, para el proceso de despliegue de aplicaciones, debido al gran apoyo que les brindan, tanto empresas como de la comunidad.
- Se diseñó un prototipo capaz de desplegar de forma sencilla contenedores. Al separar cada servicio de una aplicación en contenedores diferentes, proporciona una mejor administración de cada uno de ellos, permitiendo modificar recursos, sin perjudicar otros servicios, alojados en diversos contenedores.

8 Recomendaciones y trabajos futuros

- Como trabajo futuro se pretende implementar el despliegue de aplicaciones por medio de contenedores de software.
- Emplear la integración continua, con herramientas como Docker, Jenkins y Git, para lograr agilidad a la hora de desarrollar.
- El estudio de las opciones que da actualmente la herramienta Kubernetes, como *kubernetes en OpenStack*, que permite gestionar un híbrido de nube pública y privada.

9 Bibliografía.

Docker. (2020a, Agosto 10). Why? <https://www.docker.com/why-docker>

Docker. (2020, 2 septiembre). Docker overview. Docker Documentation.

<https://docs.docker.com/get-started/overview/>

Cynixit, S. (2020, 6 febrero). Kubernetes components - FAUN. Medium.

<https://medium.com/faun/kubernetes-components-ca583ddedeb6>

Hernández, A. (2019, 25 diciembre). Crear un cluster de docker con docker swarm.

LinkedIn. https://es.linkedin.com/pulse/crear-un-cluster-de-docker-con-swarm-adri%C3%A1n-hern%C3%A1ndez-r%C3%ADos?trk=read_related_article-card_title

Nomad, N. (2017, 10 junio). Documentation. Nomad by HashiCorp.

<https://www.nomadproject.io/docs>

Docker Hub. (2020). Docker Hub. <https://hub.docker.com/>

Goig, M. (2016, 15 abril). *Arquitectura de Docker*. DOCS.

<https://docs.mikelgoig.com/docker/arquitectura-de-docker.html>

RedHat. (2020). ¿Qué es un clúster de Kubernetes?

<https://www.redhat.com/es/topics/containers/what-is-a-kubernetes-cluster>

Kubernetes. (2020). Production-Grade Container Orchestration. <https://kubernetes.io/>

Christian M. (2019, 30 enero). Introducción a kubernetes [Archivo de vídeo]. Recuperado

de <https://www.youtube.com/watch?v=6jeCUFNv0XI&t=2120s>

Carvajal, J. (2018, 19 octubre). Primeros pasos con Kubernetes. Adictos al trabajo.

[https://www.adictosaltrabajo.com/2016/04/25/primeros-pasos-con-](https://www.adictosaltrabajo.com/2016/04/25/primeros-pasos-con-kubernetes/#:%7E:text=Los%20pods%20se%20pueden%20crear,introducir%20al%20final%20del%20fichero%20)

[kubernetes/#:%7E:text=Los%20pods%20se%20pueden%20crear,introducir%20al%20final%20del%20fichero%20.](https://www.adictosaltrabajo.com/2016/04/25/primeros-pasos-con-kubernetes/#:%7E:text=Los%20pods%20se%20pueden%20crear,introducir%20al%20final%20del%20fichero%20)

Ecys, S. (2020, 2 mayo). Docker y Herramientas de Orquestación. *Revista digital Git*, 16.

https://revistaecys.github.io/16Edicion/08_rcutz.html

Díaz, A. (2010, 25 septiembre). Desplegando software de manera segura. highscalability.

<https://highscalability.wordpress.com/2010/09/27/desplegando-software-de-manera-segura/>

IONOS S.L.U. (2020, 2 noviembre). Alternativas a los contenedores en Docker.

Recuperado de <https://www.ionos.es/digitalguide/servidores/know-how/alternativas-a-los-contenedores-en-docker/>