USO DE HERRAMIENTAS DE VIRTUALIZACIÓN Y CONTENEDORES DOCKER, PARA LA CREACIÓN DE ENTORNOS DE DESARROLLO PORTABLES, CONSISTENTES Y REPRODUCIBLES

JUAN CARLOS LEÓN DIAZ

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA - UNAD
ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA - ECBTI
TECNOLOGÍA EN GESTIÓN DE REDES DE ACCESO DE TELECOMUNICACIONES
MARIQUITA – TOLIMA
NOVIEMBRE 2023

USO DE HERRAMIENTAS DE VIRTUALIZACIÓN Y CONTENEDORES DOCKER, PARA LA CREACIÓN DE ENTORNOS DE DESARROLLO PORTABLES, CONSISTENTES Y REPRODUCIBLES

JUAN CARLOS LEÓN DIAZ

PROYECTO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR AL TÍTULO DE:
TECNÓLOGO EN GESTIÓN DE REDES DE ACCESO DE TELECOMUNICACIONES

ASESOR
MARITZA FARLEY MONDRAGON GUZMAN

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA - UNAD
ESCUELA DE CIENCIAS BÁSICAS, TECNOLOGÍA E INGENIERÍA - ECBTI
TECNOLOGÍA EN GESTIÓN DE REDES DE ACCESO DE TELECOMUNICACIONES
MARIQUITA – TOLIMA
NOVIEMBRE – 2023

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Agradecimientos

“Gracias a Dios a mi familia y a Luz, por apoyarme siempre y tenerme tanta paciencia.”

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

GLOSARIO

CONTENEDOR DOCKER: Son una forma de virtualización del sistema operativo, se puede usar para ejecutar cualquier cosa, desde un micro servicio o un proceso de software a una aplicación de mayor tamaño.

DEMONIO: Es un programa que se ejecuta en segundo plano sin necesidad de interacción por parte del usuario.

ENTORNO O AMBIENTE: En el ámbito de la computación, un "entorno" o "ambiente" se refiere a un contexto o conjunto de condiciones en el que opera un programa, sistema o proceso.

ENTORNO DE PRODUCCIÓN: Este es el sitio en vivo, es un término utilizado principalmente para describir el entorno en el que el hardware y software y otros productos se ponen realmente en funcionamiento para los usos previstos por los usuarios finales. Se puede pensar en un entorno de producción como un entorno en tiempo real donde se ejecutan programas y se instalan configuraciones de hardware y se confía en ellas para las operaciones comerciales diarias o de la organización.

ENTORNO DE TRABAJO Y/O DESARROLLO: Un entorno de desarrollo o de trabajo es un área que incluye hardware y software que permite que una aplicación o un grupo de aplicaciones se ejecuten juntas. Se llama entorno porque hay muchos componentes que se utilizan como:
Sistema Operativo, Servidores, bases de datos, Código de programa, Ajustes de configuración e Interfaces a otro sistema.

GIT: Es una herramienta que realiza una función de control de versiones de código de forma distribuida, es el sistema de control de versiones líder, fue diseñado por Linus Torvalds el creador del kernel Linux y permite realizar seguimiento y monitoreo del código fuente y trabajar junto con los miembros de un equipo en el mismo espacio de trabajo.

HIPERVISOR: Es el que se encarga de crear una capa de virtualización, asigna dinámicamente a cada máquina virtual los recursos que necesita para que independientemente del sistema operativo que utiliza, crea que el hardware físico del servidor principal está a su disposición.

LINUX DISTRO: Una distribución de Linux, abreviada como distro es un sistema operativo creado a partir de una colección de software que incluye el kernel de Linux. Una distribución típica de Linux comprende un kernel de Linux, herramientas y bibliotecas GNU, software adicional, documentación, un administrador de ventanas y un

entorno de escritorio, algunas de las distros más populares son *debian, ubuntu, fedora, CentOS y opensuse.*

PILA LAMP: Es un conjunto de cuatro tecnologías de software diferentes que los administradores de sistemas y desarrolladores, utilizan para crear sitios web y aplicaciones web. LAMP es un acrónimo del sistema operativo Linux, el servidor web Apache, el servidor de base de datos MySQL y el lenguaje de programación PHP.

REPOSITORIO: Es el lugar en el que se almacena y se puede realizar la distribución del código de una aplicación o un programa. Este debe ser un servidor seguro que utiliza sistemas de control de versiones, debe contener las diferentes versiones de la aplicación o programa, disponiendo de un historial con los cambios realizados sobre el original y sobre cada nueva versión. Además, debe permitir poder revertir esos cambios. Y permitir que la aplicación o programa pueda ser utilizado en paralelo por diferentes usuarios al mismo tiempo, en la misma o en sus diferentes versiones.

VAGRANT: Es una herramienta de código abierto que nos ayuda a automatizar la creación y gestión de Máquinas Virtuales, proporciona un cliente de línea de comandos simple y fácil de usar para administrar estos entornos y un intérprete para las definiciones basadas en texto de cómo configurar cada entorno, llamado Vagrantfile.

VIRTUALBOX: Es una aplicación que sirve para crear máquinas virtuales con instalaciones de sistemas operativos. Esto quiere decir que, si se tiene un ordenador con Windows, GNU/Linux o incluso macOS, se puede crear una máquina virtual con cualquier otro sistema operativo para utilizarlo dentro del que se está usando.

VIRTUALIZACIÓN: Es una tecnología que permite la ejecución de varias máquinas virtuales sobre una máquina física con el objetivo de aprovechar al máximo los recursos de un sistema y que su rendimiento sea mayor.

RESUMEN

La complejidad y la falta de estandarización en la creación y configuración de entornos de desarrollo y pruebas llevan a tiempos de espera prolongados y resultados inconsistentes en el desarrollo de software. La solución propuesta permitirá a los equipos de desarrollo crear y configurar fácilmente entornos de desarrollo y pruebas, reducir el consumo de recursos, mejorar la eficiencia del desarrollo de software y mejorar la colaboración y el intercambio de información entre diferentes equipos, solucionando el típico problema "pero en mi máquina si funciona". it works on my machine.

El presente proyecto de grado tiene como objetivo la creación de ambientes de desarrollo y pruebas mediante el uso de máquinas virtuales Vagrant y contenedores Docker, con el fin de reducir el tiempo de configuración y despliegue, disminuir los errores y conflictos, automatizar los procesos y reducir el consumo de recursos.

El despliegue de esta solución estará apoyado por tecnologías de desarrollo libre y los laboratorios se encuentran publicados en un repositorio público en GitHub, para facilitar su despliegue. https://github.com/nullx5/proyecto

Palabras Clave: Entorno de Trabajo, Entorno de desarrollo, Virtualización, Vagrant, Contenedores Docker, docker-compose, Repositorio, Git, GitHub, Linux Distro.

ABSTRACT

The complexity and lack of standardization in creating and configuring development and test environments lead to long lead times and inconsistent results in software development. The proposed solution will allow development teams to easily create and configure development and test environments, reduce resource consumption, improve software development efficiency, and improve collaboration and information sharing between different teams, solving the typical problem ' but on my machine it works'. it works on my machine.

This degree project aims to create development and test environments through the use of Vagrant virtual machines and Docker containers, in order to reduce configuration and deployment time, reduce errors and conflicts, automate processes and reduce resource consumption.

The deployment of this solution will be supported by free development technologies and the laboratories are published in a public repository on GitHub, to facilitate its deployment. https://github.com/nullx5/proyecto

Keywords: Work Environment, Development Environment, Virtualization, Vagrant, Docker Containers, docker-compose, Repository, Git, GitHub, Linux Distro.

1. INTRODUCCIÓN

En el ámbito de la computación, un "entorno" o "ambiente" se refiere a un contexto o conjunto de condiciones en el que opera un programa, sistema o proceso. Un entorno puede ejecutarse correctamente en el entorno en el que fue diseñado, pero cuando se mueve a otra máquina, es muy probable que falle debido a conflictos o diferencias entre versiones de software, sistema operativo, aplicaciones instaladas, librerías, datos disponibles, etc.

Además la configuración y el despliegue de entornos, se vuelve una tarea monótona y repetitiva y se carece de un sistema que gestione el despliegue de los diferentes entornos de forma eficaz y automatizada, que sea configurado una única vez y solo baste con que el administrador de la infraestructura corra una “receta” o archivo de configuración y el sistema de gestión despliegue todo el entorno, con sus componentes, como bases de datos, servidores, lenguajes de programación, frameworks, librerías, etc.

Cuando se comienza a investigar sobre la tecnología de virtualización y de contenedores Docker lo primero que llama la atención son las expectativas de futuro que ha levantado por todo el mundo, posicionándose como tecnologías que lideran la industria. Las tecnologías de virtualización y contenedores Docker han demostrado ser una tecnología eficiente para el despliegue de entornos, ofreciendo atención a desafíos de seguridad, portabilidad y escalabilidad.

El presente proyecto aborda la creación y gestión eficiente de diferentes de entornos o ambientes, se analizará el uso de contenedores Docker y máquinas virtuales vagrant y se discutirán las ventajas de estas tecnologías sobre los entornos tradicionales, y se proporcionarán ejemplos de cómo crear entornos o ambientes para diferentes aplicaciones o servicios.

2. ANTECEDENTES DEL PROBLEMA

El desarrollo de software contemporáneo se enfrenta a desafíos significativos en la creación y configuración de entornos de desarrollo. La complejidad inherente y la falta de estandarización en este proceso conllevan a la aparición de errores, conflictos y dificultades en la colaboración interequipos.

Actualmente, los equipos de desarrollo se ven obligados a atravesar una serie de pasos manuales y configuraciones para establecer entornos de desarrollo y pruebas. Esta metodología conlleva a una pérdida considerable de tiempo y da pie a inconsistencias en los resultados, contribuyendo al común problema de "funciona en mi máquina".

En este contexto, se han desarrollado diversas herramientas que abordan aspectos clave de esta problemática. Las tecnologías de virtualización, como Vagrant, han facilitado la creación rápida y la gestión de máquinas virtuales. Asimismo, los contenedores Docker han emergido como una solución eficiente para empaquetar aplicaciones y sus dependencias, garantizando portabilidad y consistencia.

A pesar de estos avances, persisten desafíos en la estandarización y la reproducción fiel de entornos de desarrollo. La falta de una solución integral que combine la virtualización y los contenedores ha llevado a la necesidad de una herramienta que permita la creación de entornos portables, consistentes y reproducibles.

El proyecto propone explorar la implementación de una solución basada en herramientas de virtualización y contenedores Docker. El objetivo principal es lograr la automatización, estandarización y reproducibilidad en la creación de entornos de desarrollo, reduciendo así el tiempo de configuración, minimizando errores y facilitando la colaboración entre equipos de desarrollo.

Este proyecto se fundamenta en investigaciones previas sobre automatización de configuración, orquestación de contenedores, prácticas DevOps y optimización de recursos en entornos virtualizados. La sinergia de estas áreas será crucial para diseñar una solución integral y escalable.

La revisión de antecedentes demuestra la relevancia y la necesidad de una solución que integre de manera efectiva herramientas de virtualización y contenedores Docker. La implementación exitosa de esta propuesta no solo mejorará la eficiencia en el desarrollo de software, sino que también establecerá estándares que promoverán la colaboración y la consistencia en entornos de desarrollo.

3. PLANTEAMIENTO DEL PROBLEMA

2.1 Formulación del problema

¿De qué forma se puede implementar una solución que permita la configuración automatizada, la eliminación de errores, conflictos y disminución de consumo de recursos (RAM, CPU, DISCO) en el despliegue de entornos de trabajo o desarrollo?

2.2 Descripción del problema

El problema a abordar en este proyecto de grado es la complejidad y la falta de estandarización en la creación y configuración de entornos de desarrollo y pruebas. Actualmente, los equipos de desarrollo de software tienen que pasar por múltiples pasos manuales y configuraciones para crear y configurar sus entornos, lo que puede llevar mucho tiempo y puede resultar en errores y conflictos en el despliegue y las pruebas. Además, la falta de estandarización en la creación y configuración de entornos puede llevar a resultados inconsistentes y puede dificultar la colaboración y el intercambio de información entre diferentes equipos, presentando el típico problema "pero en mi máquina si funciona". it works on my machine.

Por lo tanto, se requiere una solución que permita crear y configurar fácilmente entornos de desarrollo y pruebas utilizando máquinas virtuales Vagrant y contenedores Docker. La solución debe permitir la estandarización y automatización de la creación y configuración de entornos, lo que reducirá el tiempo de configuración y despliegue y disminuirá los errores y conflictos. Además, la solución también debe reducir el consumo de recursos y mejorar la eficiencia del desarrollo de software. Esta solución también debe ser escalable y fácilmente replicable en diferentes entornos y equipos de desarrollo.

4. JUSTIFICACIÓN

La virtualización y los contenedores se están adoptando en muchos entornos y situaciones diferentes, esta solución tecnológica permite reducir el tiempo de configuración y despliegue, los errores y los conflictos entre dependencias. Sí es un desarrollador puede proporcionarle un entorno contenido en el que puede realizar casi cualquier tipo de desarrollo de forma segura y sin estropear su entorno de trabajo principal, si es un administrador de sistemas, puede separar más fácilmente sus servicios y moverlos según la demanda.

En este contexto, el proyecto de grado que se propone hace uso de contenedores Docker y máquinas virtuales Vagrant para la creación de ambientes o entornos seguros, fácilmente replicables y consistentes. Este proyecto permitirá a los desarrolladores de software y/o administradores de sistemas crear ambientes de desarrollo y pruebas con una configuración idéntica a la de producción, lo que reduce los errores y aumenta la calidad del software. Además, el uso de contenedores y máquinas virtuales ayuda a proteger la infraestructura y los datos sensibles del sistema de posibles amenazas, ya que estos componentes proporcionan una capa adicional de aislamiento.

En resumen, el uso de contenedores Docker y máquinas virtuales Vagrant es una práctica cada vez más común en el desarrollo de software. Su uso proporciona múltiples beneficios, como la creación de ambientes seguros, fácilmente replicables y consistentes, y una mayor calidad del software. Por lo tanto, el proyecto de grado propuesto tiene una justificación sólida y puede tener un impacto significativo.

5. OBJETIVOS

4.1 Objetivo General

Implementar un conjunto de ambientes de desarrollo y pruebas fácilmente replicables y consistentes, utilizando contenedores Docker y máquinas virtuales Vagrant, que permitan disminuir errores y conflictos, automatizar el tiempo de configuración y despliegue y disminuir el consumo de recursos.

4.2 Objetivos Específicos

- Investigar el uso de contenedores Docker y máquinas virtuales Vagrant en el desarrollo y el despliegue de aplicaciones.
- Diseñar y desplegar cinco entornos de desarrollo y pruebas utilizando contenedores Docker y máquinas virtuales Vagrant para diferentes necesidades y aplicativos.
- Probar y validar la configuración y el despliegue de cada entorno.
- Comparar las ventajas y desventajas de contenedores Docker y máquinas virtuales Vagrant.

6. ALCANCE Y LIMITACIONES DEL PROYECTO

El proyecto no pretende ser una referencia exhaustiva de las herramientas propuestas, propone una introducción práctica, que demuestra las principales características y capacidades de dichas herramientas, el proyecto está planteado para un ambiente seguro y replicable que funciona en cualquier sistema operativo.

Alcances:

- Desarrollo y configuración de máquinas virtuales y contenedores para distintos tipos de aplicaciones.

- Comparación de los beneficios y limitaciones de Vagrant y Docker en términos de reducción de errores y conflictos, automatización de procesos, tiempo de configuración y uso de recursos.

Limitaciones:

- La limitación de recursos de hardware disponibles para el equipo de desarrollo podría limitar la cantidad de entornos de contenedores Docker y máquinas virtuales Vagrant que se pueden ejecutar simultáneamente.

- La falta de experiencia previa del equipo de proyecto en la utilización de herramientas de automatización y orquestación de contenedores Docker y máquinas virtuales Vagrant podría limitar la implementación de procesos de despliegue y pruebas automatizados.

6. MARCO DE REFERENCIA

6.1 Estado del Arte

Docker y Vagrant son dos de las herramientas más populares para desplegar aplicaciones en la nube. Ambas herramientas ofrecen una serie de ventajas sobre las soluciones tradicionales de despliegue, como la facilidad de uso, la escalabilidad y la flexibilidad.

Docker es una herramienta de implementación que permite crear contenedores, que son entornos de ejecución independientes que contienen todo lo necesario para ejecutar una aplicación, incluidos los archivos de código, el sistema operativo y las bibliotecas. Los contenedores se pueden ejecutar en cualquier máquina que tenga instalado el motor Docker, lo que los hace muy portables.

Vagrant es una herramienta de aprovisionamiento que permite configurar y desplegar máquinas virtuales en la nube. Las máquinas virtuales son instancias de un sistema operativo completo que se pueden ejecutar en una máquina física o en la nube. Vagrant facilita el despliegue de máquinas virtuales con la configuración y el software necesario para ejecutar una aplicación.

Docker y Vagrant se pueden utilizar juntos para crear una solución de despliegue completa que sea fácil de usar, escalable y flexible. Vagrant se puede utilizar para desplegar máquinas virtuales con el sistema operativo y el software necesario para ejecutar Docker. Luego, Docker se puede utilizar para crear contenedores que contengan las aplicaciones que se ejecutarán en las máquinas virtuales.

Esta combinación de herramientas ofrece una serie de ventajas sobre las soluciones tradicionales de despliegue. Docker y Vagrant hacen que sea fácil desplegar aplicaciones en la nube, incluso para los desarrolladores que no tienen experiencia en la gestión de servidores. También permiten que sea fácil escalar aplicaciones, ya que las aplicaciones se pueden ejecutar en múltiples contenedores o máquinas virtuales. Además, facilitan gestionar aplicaciones, ya que las aplicaciones se pueden administrar de forma independiente en contenedores o máquinas virtuales.

Docker y Vagrant son herramientas muy populares y están siendo utilizadas por una gran comunidad de desarrolladores. Estas herramientas ofrecen una serie de ventajas sobre las soluciones tradicionales de despliegue, lo que las convierte en una excelente opción para desplegar aplicaciones en la nube.

Literatura existente, principales estudios y artículos sobre contenedores docker y máquinas virtuales vagrant:

- Kleppmann, M. (2015). A survey of containerization technologies. ACM SIGOPS Operating Systems Review.

- Ellingwood, J. (2016). A comparative study of Docker and Vagrant. Journal of Cloud Computing.

- Amit M Potdara, Narayan D Gb, Shivaraj Kengondc, Mohammed Moin Mullad.- ScienceDirect. (2020). Performance Evaluation of Docker Container and Virtual Machine.

- Srinath Reddy Meadusani - Cloud State University. (2018). Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications.

- Stuart Clark - cisco DevNet (2018). NetDevOps Developer Environments with Vagrant.

Evaluación de los resultados de estos estudios:

- Kleppmann (2015) proporciona una visión general de las tecnologías de contenedores, incluyendo Docker y Vagrant. El artículo discute los beneficios y limitaciones de cada tecnología, y sugiere que Docker es una mejor opción para la mayoría de las aplicaciones.

- Ellingwood (2016) realiza un estudio comparativo de Docker y Vagrant. El artículo encuentra que Docker es más rápido y fácil de usar que Vagrant, pero que Vagrant ofrece más flexibilidad.

- Potdara et al. (2020) realizan una evaluación comparativa del rendimiento de Docker y las máquinas virtuales. El estudio encuentra que Docker es más eficiente que las máquinas virtuales en términos de uso de memoria y CPU. Sin embargo, las máquinas virtuales son más flexibles que Docker, ya que permiten ejecutar sistemas operativos diferentes en la misma máquina.

- Meadusani (2018) proporciona una descripción general de la virtualización utilizando contenedores Docker. El artículo explica cómo los contenedores Docker pueden crear entornos reproducibles para aplicaciones containerizadas.

- Clark (2018) describe cómo se pueden utilizar las máquinas virtuales Vagrant para crear entornos de desarrollo DevOps. El artículo explica cómo Vagrant puede automatizar la configuración de máquinas virtuales para el desarrollo y el despliegue de aplicaciones.

6.2 Marco Contextual

En la actualidad, los contenedores Docker y las máquinas virtuales Vagrant son dos de las tecnologías más populares para el desarrollo y el despliegue de aplicaciones. Las tecnologías de virtualización y contenerización han transformado radicalmente la forma en que se construyen, implementan y gestionan sistemas de software. Este marco contextual tiene como objetivo situar la investigación en el contexto más amplio de la virtualización y la contenerización, resaltando la relevancia y el impacto de estas tecnologías en la industria y en el ámbito académico.

En este contexto, esta investigación se enfoca en analizar y comparar críticamente el uso de contenedores Docker y máquinas virtuales Vagrant en términos de eficiencia, rendimiento y agilidad en el desarrollo y despliegue de aplicaciones. La comparación de estas tecnologías proporcionará información valiosa para los profesionales de TI y los desarrolladores que buscan adoptar la mejor solución para sus necesidades. Además, esta investigación contribuirá al conocimiento existente en el campo de la virtualización y la contenerización, destacando los beneficios y desafíos asociados con cada enfoque.

6.3 Marco Teórico

El marco teórico de esta investigación se basa en los conceptos fundamentales de virtualización y contenerización, centrándose en el uso de contenedores Docker y máquinas virtuales Vagrant como herramientas clave para el desarrollo y despliegue de aplicaciones en entornos de TI. Este marco teórico busca establecer una base conceptual sólida que permita comprender las implicaciones y ventajas de ambas tecnologías en términos de eficiencia, agilidad y administración de recursos.

Virtualización y Contenerización:

La virtualización es una tecnología que permite la creación de entornos aislados y autónomos en un único sistema físico. Las máquinas virtuales (VMs) son un enfoque tradicional de virtualización que emula hardware y sistemas operativos completos, permitiendo la ejecución de múltiples sistemas independientes en una sola máquina física. Por otro lado, la contenerización se centra en encapsular aplicaciones y sus dependencias en entornos aislados conocidos como contenedores, compartiendo el mismo kernel del sistema operativo subyacente.

Contenedores Docker:

Docker es una plataforma líder en el ámbito de la contenerización. Los contenedores Docker ofrecen ventajas en términos de eficiencia, portabilidad y escalabilidad. Al compartir el kernel del sistema operativo, los contenedores son más livianos y requieren menos recursos en comparación con las VMs. Docker también simplifica la gestión de aplicaciones y sus dependencias, garantizando la coherencia en diferentes entornos.

Máquinas Virtuales Vagrant:

Vagrant, por otro lado, se enfoca en la creación y gestión de entornos de desarrollo reproducibles. A través de definiciones en código, Vagrant permite a los desarrolladores configurar y provisionar máquinas virtuales de manera uniforme. Esto aborda problemas de inconsistencia en los entornos de desarrollo y facilita la colaboración entre equipos.

6.4 Marco Conceptual

Entorno o ambiente:
En el ámbito de la computación, un "entorno" o "ambiente" se refiere a un contexto o conjunto de condiciones en el que opera un programa, sistema o proceso.

Contenedor vs Máquina virtual:

Mucha gente asume que, dado que los contenedores aíslan las aplicaciones, son lo mismo que las máquinas virtuales. A primera vista lo parece, pero la diferencia fundamental es que los contenedores comparten el mismo kernel que el host.

Docker solo aísla un único proceso (o un grupo de procesos) y todos los contenedores se ejecutan en el mismo sistema anfitrión. Dado que el aislamiento se aplica a nivel de kernel, ejecutar los contenedores no impone una gran sobrecarga al host en comparación a las máquinas virtuales.

Cuando se activa un contenedor, el proceso o grupo de procesos seleccionado aún se ejecuta en el mismo host, sin necesidad de virtualizar o emular. Por el contrario, cuando se activa una máquina virtual, el hipervisor virtualiza un sistema completo, desde la CPU hasta la RAM y el almacenamiento.

En las máquinas virtuales es necesario instalar un sistema operativo completo. Para efectos prácticos, el sistema virtualizado es una computadora completa corriendo en una computadora.

Figura 1. ***Arquitectura de Máquinas Virtuales.***

Nombre de la Fuente: (docker-ee.blogspot.com, 2015)

En resumen, una máquina virtual es:

- Un software que permiten emular el funcionamiento de un ordenador dentro de otro ordenador.
- El usuario debe configurar el procesador, RAM, disco duro, entre otros.
- Emulan desde cero sistemas como Windows, MacOS, Linux o Android.
- Requieren de gran configuración y rendimiento.

Figura 2. ***Arquitectura de contenedores de Docker.***

Nombre de la Fuente: (docker-ee.blogspot.com, 2015)

En resumen, los contenedores:

- Empaquetan el software, incluyendo las librerías, que actúan como un entorno aislado del sistema operativo.
- Trabajan directamente sobre el Kernel.
- No es necesario emular el sistema operativo porque trae todo lo necesario para ser ejecutado.
- Requieren menor uso de recursos por parte de la máquina.

¿Qué es Vagrant?:

Es una herramienta open source de líneas de comandos, escrita en el lenguaje de programación Ruby y creado por hashicorp, utilizada en el sector IT, para gestionar máquinas virtuales, permitiendo crear entornos de desarrollo virtualizados que pueden ser reproducidos y compartidos de una forma fácil y sencilla.

Esta aplicación nos permitirá manejar y configurar máquinas virtuales manteniendo un mismo entorno de trabajo, debido a que evita los problemas de compatibilidad entre sistemas y brinda la posibilidad de compartir sus archivos o también llamados, archivos Vagrantfile, donde se centralizan las configuraciones creadas. Para su funcionamiento, Vagrant utiliza 3 componentes básicos, los *Vagrantfile, los Provider y los Provisioner*.

Historia de Vagrant:

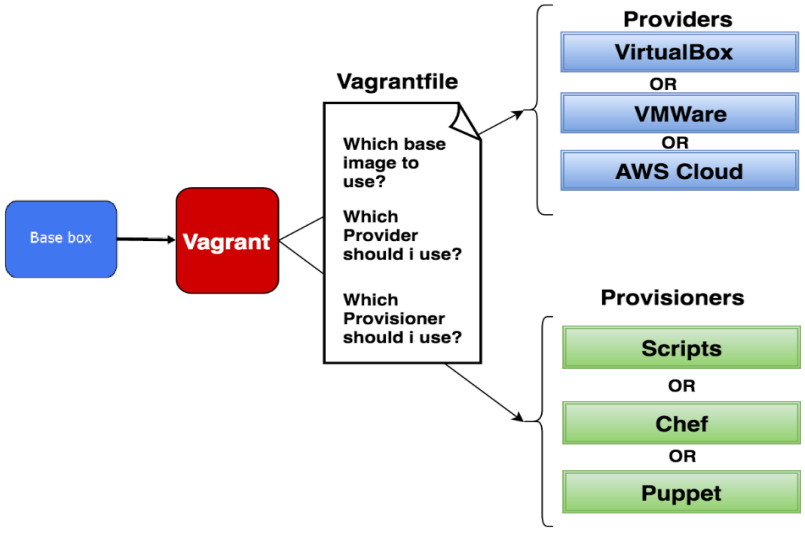
Vagrant se inició por primera vez como un proyecto paralelo personal de Mitchell Hashimoto en enero de 2010. La primera versión de Vagrant se lanzó en marzo de 2010. En octubre de 2010, Engine Yard declaró que iban a patrocinar el proyecto Vagrant. La primera versión estable, Vagrant 1.0, se lanzó en marzo de 2012, exactamente dos años después del lanzamiento de la versión original. En noviembre de 2012, Mitchell formó una organización llamada HashiCorp para apoyar el desarrollo a tiempo completo de Vagrant; Vagrant siguió siendo un software libre con licencia permisiva. HashiCorp ahora trabaja en la creación de ediciones comerciales y brinda soporte profesional y capacitación para Vagrant.

Originalmente, Vagrant estaba vinculado a VirtualBox, pero la versión 1.1 agregó soporte para otro software de virtualización como VMware y KVM, y para entornos de servidor como Amazon EC2. Vagrant está escrito en Ruby, pero puede usarse en proyectos escritos en otros lenguajes de programación como PHP, Python, Java, C# y JavaScript. Desde la versión 1.6, Vagrant es compatible de forma nativa con contenedores Docker, que en algunos casos pueden servir como sustituto de un sistema operativo totalmente virtualizado.

Arquitectura Vagrant:

Vagrant utiliza varios componentes, como lo son cajas o boxes, el archivo de configuración Vagrantfile, los proveedores o providers y los aprovisionadores o provisioners para administrar entornos de desarrollo.

Figura 3. ***Arquitectura de Vagrant.***



Nombre de la fuente: (onlineitguru.com, 2020).

Vagrant box**:**

Es la unidad básica de configuración de Vagrant, es una imagen independiente, más específicamente, es una Máquina Virtual empaquetada. En lugar de instalar un sistema operativo y todos los componentes de software dentro de una VM manualmente, la caja vagrant es una imagen base lista para usar de un entorno de máquina virtual.

Por ejemplo, si es necesario obtener alguna VM con todas las configuraciones predeterminadas del desarrollo de la aplicación Spring Boot, se puede obtener el mismo Vagrant Box. Todo lo que se requiere hacer es descargar Vagrant Box y ejecutarlo. Vagrant crea la máquina virtual y el trabajo de desarrollo puede comenzar de inmediato.

Una gran cantidad de Vagrant Box están presentes en el catálogo de cajas de *Vagrant Cloud*, que podemos usar como imagen base para nuestra propia máquina virtual.

Vagrant Cloud**:**

Sirve como un índice público de búsqueda de cajas de Vagrant. Es fácil encontrar cajas que puedas usar con Vagrant que contengan las tecnologías que necesitas, puedes encontrar cajas base de las principales distros Linux como son *debian, ubuntu, fedora, centOS y OpenSUSE* y otras tecnologías.

- https://app.vagrantup.com/debian
- https://app.vagrantup.com/ubuntu
- https://app.vagrantup.com/fedora
- https://app.vagrantup.com/centos
- https://app.vagrantup.com/opensuse
- https://app.vagrantup.com/kalilinux

Vagrantfile:

Vagrant mantiene un archivo de configuración, llamado Vagrantfile, donde se mencionan todas las configuraciones de una máquina virtual, como pueden ser cantidad de Memoria RAM, CPU, tipo de red, etc. y Vagrant crea la máquina virtual con la misma configuración mencionada en el archivo Vagrantfile. Incluso si es necesario instalar algún software en la máquina virtual, se puede especificar en el mismo Vagrantfile, y Vagrant lo descarga e instala por nosotros.

La sintaxis de Vagrantfiles es Ruby, pero no es necesario el conocimiento del lenguaje de programación Ruby para realizar modificaciones en el Vagrantfile, ya que en su mayoría es una simple asignación de variables.

La función principal de Vagrantfile es describir el tipo de máquina requerida para un proyecto y cómo configurar y aprovisionar estas máquinas. Los archivos Vagrantfile se denominan Vagrantfile porque el nombre de archivo literal real para el archivo es Vagrantfile.

Vagrant está diseñado para ejecutarse con un Vagrantfile por proyecto, y se supone que el archivo Vagrantfile está comprometido con el sistema de control de versiones, ya que esto permite que otros involucrados en el proyecto revisen el código, ejecuten vagrant y sigan su camino. Los Vagrantfiles son portátiles en todas las plataformas compatibles con Vagrant.

Estructura básica de un archivo Vagrantfile:

```
Vagrant.configure(2) do |config|
        config.vm.box = "centos/7"
        config.vm.hostname = "myhost"
        config.vm.network "private_network", ip: "192.168.50.10"
        config.vm.synced_folder ".", "/var/www/html"

config.vm.provider "virtualbox" do |vb|
        vb.memory = "1024"
        vb.cpus = "2"
end

config.vm.provision "shell", inline: <<-SHELL
        sudo apt update
        sudo apt upgrade -y
        sudo apt install nginx -y
SHELL
```

Provider:
Los proveedores son los servicios usados por vagrant, con el fin de crear los entornos virtuales, también llamados hipervisores, vagrant soporta varios, algunos de los más populares son Virtualbox, VMware, Hyper-V, QEMU, etc. Para los propósitos de este proyecto se utiliza Virtualbox.

Provisioner:
Los aprovisionadores son herramientas que permiten a los usuarios instalar software y personalizar la configuración de los entornos virtuales.
Para propósitos del proyecto y para simplificar su funcionamiento utilizaremos como aprovisionador solo scripts de Shell, sin embargo, existen herramientas más especializadas como Ansible, Puppet y Chef que son los proveedores más utilizados en el ecosistema de Vagrant.

Comandos Básicos Vagrant**:**

```
vagrant --version
vagrant init ubuntu/focal64
vagrant status
vagrant up
vagrant ssh
vagrant halt
vagrant reload
vagrant destroy
vagrant box list
vagrant box add USER/BOX
vagrant box remove USER/BOX
vagrant plugin list
```

vagrant plugin install **PLUGIN**
vagrant plugin uninstall **PLUGIN**

vagrant --help

¿Qué es Docker?:

Docker es una plataforma de código abierto diseñada para que administradores de sistemas y desarrolladores desarrollen, envíen y ejecuten aplicaciones distribuidas en contenedores que corren en cualquier entorno como desarrollo, prueba o producción, los contenedores empaquetan todo lo necesario para que uno o más procesos funcionen con todas sus dependencias, es decir, con todo lo que necesita para poder ejecutarse, como el código, herramientas del sistema y librerías del sistema, etc. Esto garantiza que siempre se podrá ejecutar, independientemente del entorno en el que se quiera desplegar, bastando únicamente con tener instalado Docker dentro de la otra máquina, sin tener problemas con el entorno de ejecución.

Docker permite que las aplicaciones usen el mismo kernel de Linux como sistema en la computadora host, en lugar de crear un sistema operativo virtual completo, a primera vista se piensa en Docker como una especie de máquina virtual “liviana”, pero la verdad no lo es. En Docker no existe un hipervisor que virtualice hardware sobre el cual corra un sistema operativo completo. En Docker lo que se hace es usar las funcionalidades del Kernel para encapsular un sistema, de esta forma el proyecto que corre dentro de el no tendrá conocimiento que está en un contenedor. Los contenedores se encuentran aislados entre sí y se comportaran como máquinas independientes.

Historia de Docker:

Docker Inc. es la empresa detrás de Docker. Docker Inc. fue fundada como dotCloud Inc. en 2010 por Solomon Hykes. Los ingenieros de dotCloud crearon abstracción y herramientas para los contenedores de Linux y utilizaron las funciones del kernel de Linux *cgroups* y *Namespaces* con la intención de reducir la complejidad en torno al uso de contenedores de Linux. dotCloud hizo que sus herramientas fueran de código abierto y cambió el enfoque del negocio de PaaS para centrarse en la contenedorización.

Contenedorización a lo largo de los años**:**

Si bien la contenedorización se ha acelerado y su popularidad se ha disparado en los últimos años, el concepto de contenedorización se remonta a la década de 1970.

1979: chroot**:**

La llamada al sistema chroot se introdujo en la versión 7 de UNIX en 1979. La premisa de chroot era que cambiaba el directorio raíz aparente para el proceso en ejecución actual y sus hijos. Un proceso iniciado dentro de un chroot no puede acceder a archivos fuera del árbol de directorios asignado. Este entorno se conocía como la cárcel chroot.

2000: FreeBSD**:**

Jails Ampliando el concepto chroot, FreeBSD agregó soporte para una función que permitía particionar el sistema FreeBSD en varios sistemas independientes y aislados llamados jails. Cada cárcel es un entorno virtual en el sistema host con su propio conjunto de archivos, procesos y cuentas de usuario. Mientras que chroot solo restringía los procesos a una vista del sistema de archivos, FreeBSD encarcela las actividades restringidas del proceso encarcelado a todo el sistema, incluidas las direcciones IP que estaban vinculadas a él. Esto hizo que las cárceles de FreeBSD fueran la forma ideal de probar nuevas configuraciones de software conectado a Internet, lo que facilita experimentar con diferentes configuraciones sin permitir que los cambios de la cárcel afecten al sistema principal externo.

2005: OpenVZ**:**

OpenVZ fue bastante popular en la provisión de virtualización del sistema operativo para proveedores de servidor privado virtual (VPS) de gama baja. OpenVZ permitió un servidor físico para ejecutar varias instancias aisladas del sistema operativo, conocidas como contenedores. OpenVZ usa un kernel Linux parcheado, compartiéndolo con todos los contenedores Cada contenedor actúa como una entidad separada y tiene su propio virtualizado conjunto de archivos, usuarios, grupos, árboles de proceso y dispositivos de red virtual.

2006: cgroups**:**

Originalmente conocidos como contenedores de procesos, los ingenieros de Google iniciaron cgroups (abreviatura de grupos de control). cgroups es una característica del kernel de Linux que limita y aísla el uso de recursos (como CPU, memoria, E/S de disco y red) a una colección de procesos. cgroups se ha rediseñado varias veces, cada rediseño representa su creciente número de casos de uso y características requeridas.

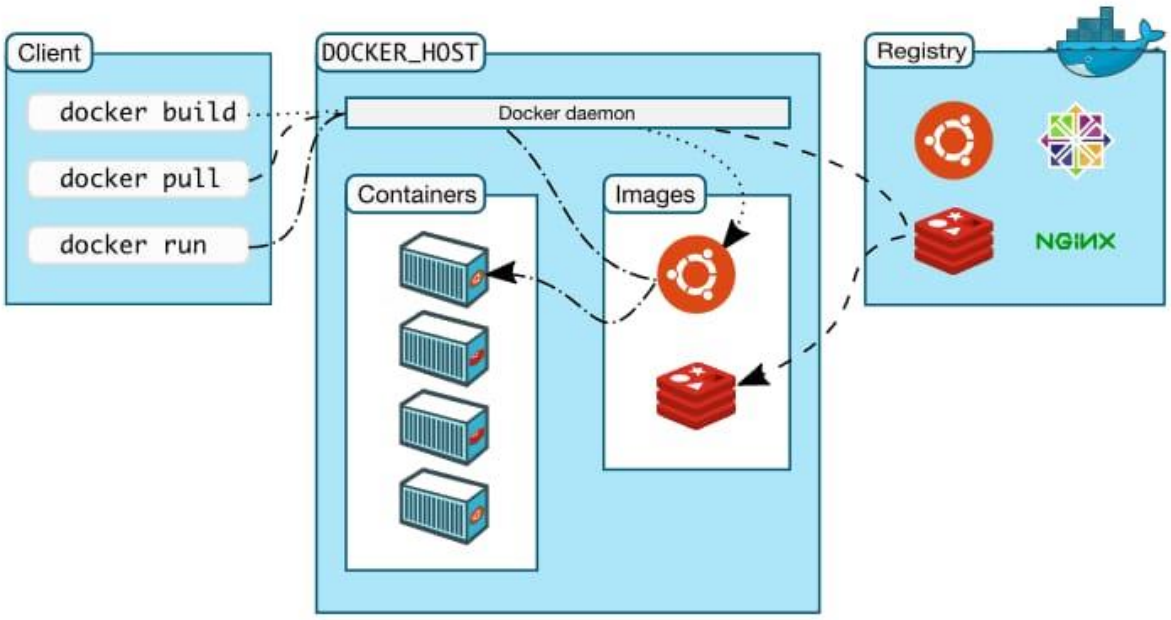
2008: LXC**:**

LXC proporciona virtualización a nivel de sistema operativo mediante la combinación de cgroups del kernel de Linux y la compatibilidad con espacios de nombres aislados para proporcionar un entorno aislado para las aplicaciones. Docker utilizó inicialmente LXC para proporcionar las funciones de aislamiento, pero luego cambió a su propia biblioteca.

Arquitectura Docker:

Los componentes básicos incluyen *Docker Daemon, Docker Client, Docker Image, Docker Registry, Docker Container y Docker Networking.*

Figura 4. *Arquitectura de Docker.*



Nombre de la fuente: (docs.docker.com, 2020).

Imágenes Docker:

Una imagen de Docker es una plantilla de solo lectura que define su contenedor. La imagen contiene el código que se ejecutará, incluida cualquier definición para cualquier biblioteca o dependencia que el código necesite. Un contenedor de Docker es una imagen de Docker instanciada (en ejecución), la imagen se crea a partir del Dockerfile y a partir de la imagen se construye un contenedor en Docker.

DockerHub:

Docker Hub es un servicio de registro en la nube, para imágenes de Docker que le permite alojar y descargar imágenes de Docker. Cuenta con una gran biblioteca de imágenes confiables y certificadas por Docker y por editores, alojando más de 100.000 imágenes, incluidas imágenes oficiales para MongoDB, nginx, Apache, Ubuntu y MySQL, que se han descargado más de mil millones de veces cada una.

- https://hub.docker.com/_/mongo
- https://hub.docker.com/_/nginx
- https://hub.docker.com/_/httpd
- https://hub.docker.com/_/python
- https://hub.docker.com/_/node
- https://hub.docker.com/_/php
- https://hub.docker.com/_/postgres
- https://hub.docker.com/_/mariadb
- https://hub.docker.com/_/mysql
- https://hub.docker.com/r/ubuntu/bind9
- https://hub.docker.com/r/ubuntu/squid
- https://hub.docker.com/_/wordpress
- https://hub.docker.com/_/drupal
- https://hub.docker.com/_/joomla

Además de los repositorios públicos de los que cualquiera puede extraer, Docker Hub ofrece repositorios privados donde las personas o los equipos pueden alojar imágenes a las que desean restringir el acceso.

Contenedor Docker:

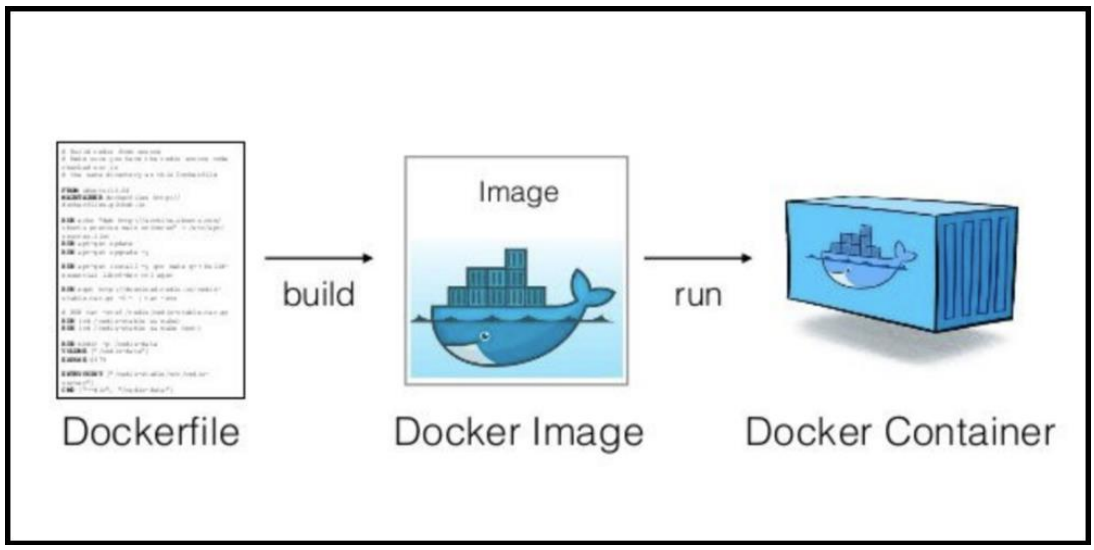
Los contenedores son instancias de imágenes de Docker, es decir son creados a partir de las imágenes de Docker y se pueden ejecutar o iniciar con el comando docker run. El propósito básico de Docker es ejecutar contenedores.

Son una abstracción en la capa de la aplicación que empaqueta el código y las dependencias juntos. Varios contenedores pueden ejecutarse en la misma máquina y compartir el kernel del sistema operativo con otros contenedores, cada uno ejecutándose como procesos aislados en el espacio del usuario.

Los contenedores ocupan menos espacio que las máquinas virtuales (las imágenes de contenedores suelen tener un tamaño de decenas de MB), pueden manejar más aplicaciones y requieren menos máquinas virtuales y sistemas operativos.

Por ejemplo, supongamos que una aplicación que incluye el sistema operativo Ubuntu y el servidor Nginx debe agregarse al archivo docker. Usando el comando 'docker run', se crea un contenedor con la imagen del sistema operativo Ubuntu que consiste en el servidor Nginx.

Figura 5. ***Flujo de trabajo contenedor Docker.***



Nombre de la fuente: (docs.docker.com, 2020).

Dockerfile:

Un Dockerfile es un archivo de texto base que describe los pasos que debe seguir Docker para construir una imagen, incluida la instalación de paquetes, la creación de directorios y la definición de variables de entorno, entre otras cosas, las instrucciones van de forma secuencial. Dockerfile se describe mejor como infraestructura como código.

Las instrucciones más frecuentes que se utilizan para los Dockerfiles:

FROM imagen

Permite coger como base una imagen del repositorio de Docker Hub.

WORKDIR directorio

Directorio de trabajo dentro del contenedor.

ENV variable=valor

Permite pasar variables ambientales al contenedor.

COPY directorio-host directorio-imagen

Permite copiar el contenido de un directorio o de un fichero del equipo Host al directorio que se especifique dentro de la imagen base.

VOLUME ruta-directorio

Establece los puntos de montaje externalizando un determinado directorio y proporcionando persistencia (las imágenes de Docker son de solo lectura y no almacenan datos entre diferentes ejecuciones).

RUN comando

Ejecuta los comandos que se lanzan dentro de nuestra imagen. Debe de ser acorde al sistema operativo que use la imagen base.

EXPOSE puerto

Indica a Docker que el servicio del contenedor se puede conectar a través del puerto que se pasa como argumento.

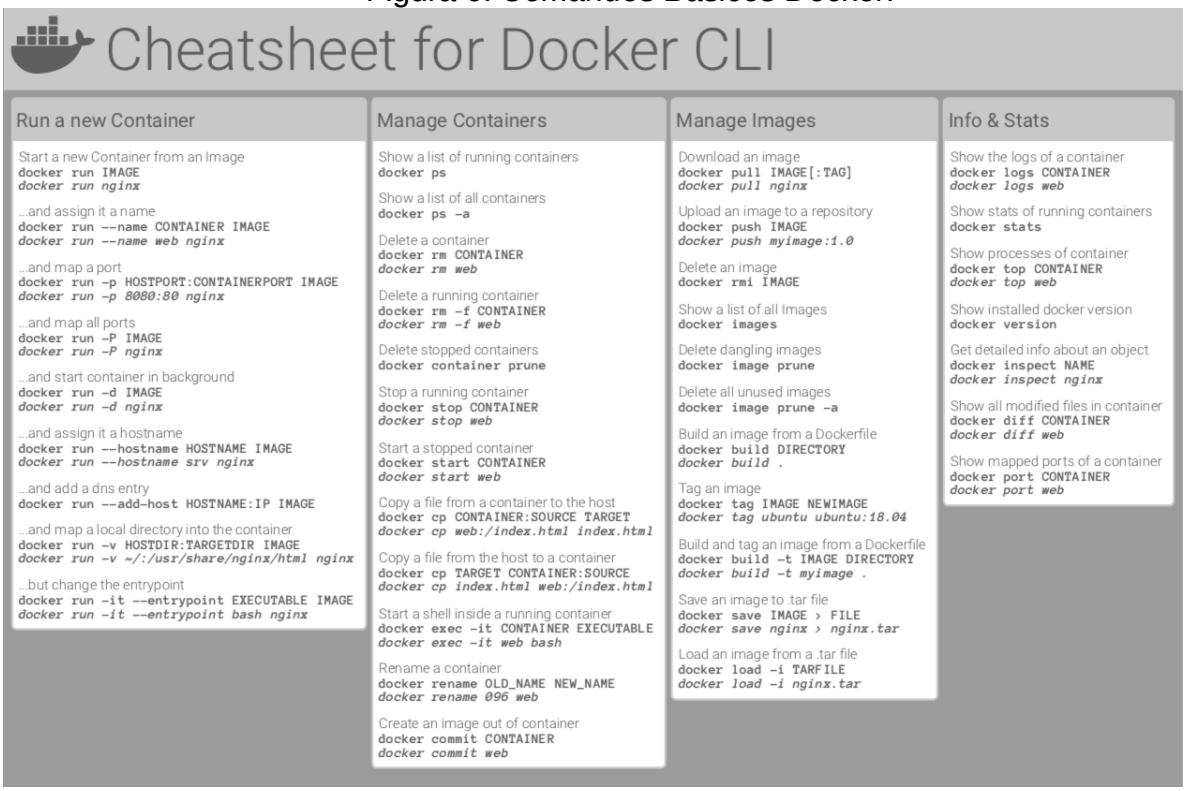
CMD ejecutable parametro1 parametro2

Ejecuta un comando o ejecutable una vez que el contenedor se haya inicializado.

ENTRYPOINT
Se utiliza para definir el comando y los argumentos predeterminados que se ejecutarán cuando inicies un contenedor.

Comandos Básicos Docker**:**

Figura 6. *Comandos Básicos Docker.*



Nombre de la fuente: (dockerlabs.collabnix.com, 2019)

Docker-compose:

docker-compose es una herramienta que sirve para definir y ejecutar aplicaciones Docker de múltiples contenedores, permitiendo crear redes de contenedores enlazados. Con docker-compose, utiliza un archivo YAML para configurar los servicios de su aplicación, luego, con un solo comando, crea e inicia todos los servicios desde su archivo de configuración docker-compose.yml.

Usar docker-compose es básicamente un proceso de tres pasos:

- Defina el entorno de su aplicación con un Dockerfile para que pueda reproducirse en cualquier lugar.
- Defina los servicios que componen su aplicación docker-compose.yml para que puedan ejecutarse juntos en un entorno o red aislada.
- Ejecutar docker-compose up, inicia y ejecuta toda su aplicación.

Docker-compose.yml:

Estructura básica de un archivo docker-compose.yml.

```
version: "3.9"
services:
  db:
    image: mariadb:10.6.4-focal
    volumes:
      - db_data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=somewordpress
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=wordpress
    expose:
      - 3306

  wordpress:
    image: wordpress:latest
    volumes:
      - wp_data:/var/www/html
    ports:
      - 80:80
    restart: always
    environment:
      - WORDPRESS_DB_HOST=db
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=wordpress
      - WORDPRESS_DB_NAME=wordpress
```

Comandos Básicos Docker-compose**:**

Commands:

build	**Build or** rebuild services
bundle	**Generate** a **Docker** bundle from the **Compose** file
config	**Validate and** view the **Compose** file
create	**Create** services
down	**Stop and** remove containers, networks, images, **and** volumes
events	**Receive** real time events from containers
exec	**Execute** a command **in** a running container
help	**Get** help on a command
images	**List** images
kill	**Kill** containers
logs	**View** output from containers
pause	**Pause** services
port	**Print** the **public** port **for** a port binding
ps	**List** containers
pull	**Pull** service images
push	**Push** service images
restart	**Restart** services
rm	**Remove** stopped containers
run	**Run** a one-off command
scale	**Set** number of containers **for** a service
start	**Start** services
stop	**Stop** services
top	**Display** the running processes
unpause	**Unpause** services
up	**Create and** start containers
version	**Show** the **Docker**-**Compose** version information

6.5 Marco Normativo

Las regulaciones y normativas pueden variar según la industria, el país y el contexto. sin embargo, estos conceptos pueden encajar en los marcos normativos de seguridad de datos y regulaciones de privacidad, como el Reglamento General de Protección de Datos **(GDPR)** de la Unión Europea o las regulaciones de privacidad en los Estados Unidos, o en el caso de Colombia, **la ley 1273 de 2009** *“Por medio de la cual se modifica el Código Penal, se crea un nuevo bien jurídico tutelado - denominado "de la protección de la información y de los datos" y se preservan integralmente los sistemas que utilicen las tecnologías de la información y las comunicaciones, entre otras disposiciones"* y la **ley estatutaria 1581 de 2012** *"Por la cual se dictan disposiciones generales para la protección de datos personales."*

*E*sto puede afectar cómo manejas los datos dentro de los contenedores y las máquinas virtuales. Es importante asegurarse que los datos sensibles se manejen de manera adecuada y segura.

7. METODOLOGIA

7.1 Método

Esta investigación utilizará una combinación de métodos cualitativos y cuantitativos. Los métodos cualitativos se utilizarán para investigar las experiencias de los desarrolladores que utilizan contenedores Docker y máquinas virtuales Vagrant, según la encuesta de stackoverflow 2023. Los métodos cuantitativos se utilizarán para comparar las ventajas y desventajas de contenedores Docker y máquinas virtuales Vagrant.

7.2 Tipo de Estudio

La metodología de esta propuesta es el método aplicativo, que consiste en dar solución a una necesidad práctica. Se debe conocer el problema observar las causas y consecuencias del mismo de manera que permita plantear soluciones adecuadas, en este caso una solución que permita la gestión de entornos de trabajo o desarrollo, al final de esta fase el lector obtendrá una idea de alto nivel sobre cómo gestionar entornos, crear contenedores usando Docker y cómo implementarlos en un entorno de trabajo o desarrollo.

El tipo de investigación que se va a desarrollar es la investigación aplicada tecnológica, de corte cualitativa, que busca a partir de la teoría establecer estrategias para dar solución a un determinado problema.

Para llevar a cabo la implementación, se detalla la configuración de varias simulaciones que van a representar los diferentes entornos o ambientes. Para realizar todas las pruebas con Docker se ha utilizado vagrant como gestor de máquina virtual a través del hipervisor Virtualbox con las distribuciones Linux *debian y ubuntu*.

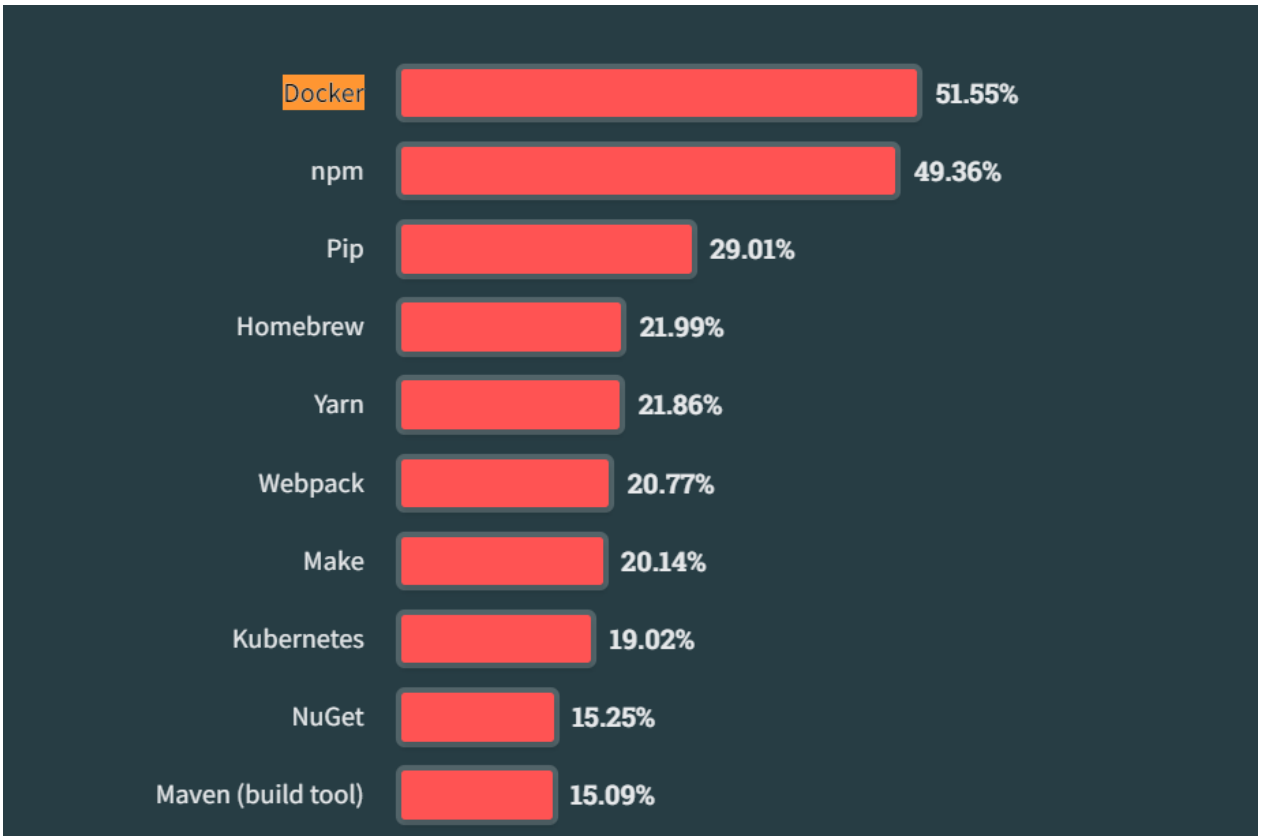
7.3 Recolección de Datos

La población está representada en la encuesta anual de la comunidad de usuarios de StackOverflow 2023, los usuarios administradores de sistemas sysadmin, ingenieros de redes, desarrolladores de software, personal de departamentos de Tecnologías de la Información e infraestructura.

Más de 90.000 administradores de sistemas y desarrolladores de más 180 países de todo el mundo, le dijeron a StackOverflow, cómo aprenden y qué herramientas usan.

Los encuestados fueron reclutados principalmente a través de canales propiedad de StackOverflow. Las principales fuentes de encuestados fueron mensajes en el sitio, publicaciones en blogs, listas de correo electrónico, anuncios publicitarios y publicaciones en redes sociales.

Figura 7. *Tecnologías más populares según StackOverflow.*



Nombre de la fuente: (survey.stackoverflow.co, 2023).

La encuesta encontró que el 93% de los desarrolladores utilizan contenedores, y que Docker es la plataforma de contenedores más popular, utilizada por el 84% de los desarrolladores.

La encuesta también encontró que los desarrolladores utilizan contenedores por una variedad de razones, incluyendo la facilidad de despliegue, la escalabilidad y la portabilidad.

Los desarrolladores también encontraron que los contenedores han tenido un impacto positivo en su trabajo, haciéndoles más eficientes y productivos.

Resultados y porcentajes sobre el uso de contenedores Docker en la encuesta de Stack Overflow 2023:

¿Utilizas contenedores?
Sí: 93%
No: 7%
¿Qué plataforma de contenedores utilizas?
Docker: 84%
Kubernetes: 10%
Otros: 6%
¿Por qué utilizas contenedores?
Facilidad de despliegue: 55%
Escalabilidad: 45%
Portabilidad: 40%
Eficiencia: 35%
Productividad: 30%
¿Qué impacto han tenido los contenedores en tu trabajo?
Positivo: 85%
Negativo: 15%

Tabla 1. ***Comparativa Máquina Virtual y Docker.***

Máquina virtual	Contenedor Docker
Aislamiento de procesos a nivel de hardware	Aislamiento de procesos a nivel de sistema operativo
Cada máquina virtual tiene un sistema operativo independiente	Cada contenedor puede compartir el sistema operativo
bootea en minutos	bootea en segundos
Las máquinas virtuales son de pocos GB	Los contenedores son ligeros (KB/MB)
Las máquinas virtuales listas para usar son difíciles de encontrar	Los contenedores Docker pre construidos están fácilmente disponibles
Las máquinas virtuales pueden moverse fácilmente a un nuevo host	Los contenedores se destruyen y recrean en lugar de moverse
La creación de VM lleva un tiempo relativamente más largo	Los contenedores se pueden crear en segundos.
Más uso de recursos	Menos uso de recursos

8. IMPLEMENTACION DE LABORATORIOS

Los laboratorios tendrán la siguiente estructura:

- Primero se presenta una breve descripción de los componentes o tecnologías que contendrá el entorno.
- Segundo se muestra la descripción de los archivos de configuración Vagrantfile, Dockerfile o docker-compose.yml, según corresponda.
- Tercero se muestran los comandos para iniciar el entorno.
- Cuarto se muestran los comandos para verificar que el entorno se configuro correctamente.
- Quinto se muestra los servicios corriendo correctamente.

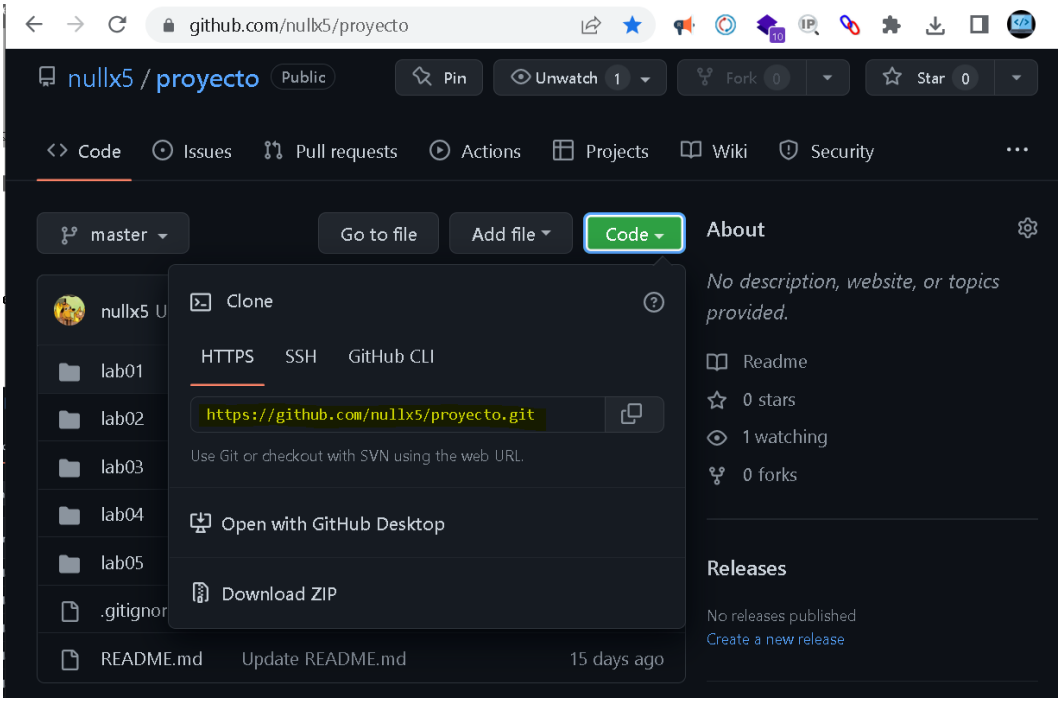
Figura 8. Ciclo de despliegue

Nombre de la fuente: (Elaboración propia).

Para iniciar el despliegue de los laboratorios se debe tener instalado el programa **git**, el hipervisor **virtualbox** y el cliente de **vagrant**, luego se puede clonar el repositorio con los laboratorios en la maquina local, con el siguiente comando:

```
git clone https://github.com/nullx5/proyecto.git
```

Figura 9. *Repositorio Github con los laboratorios.*



Nombre de la fuente: (Elaboración propia).

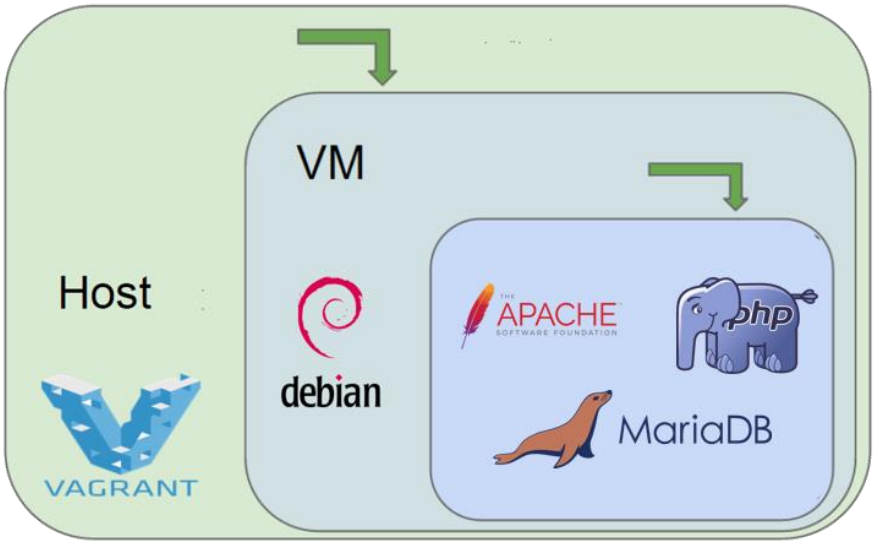
8.1 LABORATORIO 1:

8.1.1 Descripción Componentes del Entorno:

Este laboratorio configura y despliega un entorno de desarrollo con la pila LAMP, usando una máquina virtual, administrada por *vagrant* a través de su archivo de configuración Vagrantfile, como se muestra en la **Figura 10.**

Una pila *LAMP* es un conjunto de cuatro tecnologías de software diferentes que los administradores de sistemas y desarrolladores, utilizan para crear sitios web y aplicaciones web. LAMP es un acrónimo del sistema operativo Linux, el servidor web Apache, el servidor de base de datos MySQL y el lenguaje de programación PHP.

Figura 10. *Laboratorio 1.*



Nombre de la fuente: (Elaboración propia).

8.1.2 Descripción archivos de configuración:

A continuación, se detallan las configuraciones del archivo **Vagrantfile** laboratorio 1:

```
config.vm.box = "debian/bullseye64"
```

Con esta declaración se define el sistema operativo base, que usara la máquina virtual, en este caso se usa *Debian 11 bullseye,* se pueden encontrar otras distribuciones Linux en *vagrant cloud*.

```
config.vm.hostname = "debian1"
```

Con esta declaración se define el nombre del Host de la máquina.

```
config.vm.network "private_network", ip: "192.168.33.31"
```

Con esta declaración se define el adaptador de RED o NIC de la máquina en modo NAT, con la dirección IP “**192.168.33.31",** también se puede establecer el adaptador de RED en modo BRIDGE, con la opción "public_network".

```
config.vm.synced_folder ".", "/var/www/html/php"
```

Con esta declaración se define las carpetas sincronizadas, entre la máquina anfitrión y la máquina invitada, el primer parámetro es una ruta a un directorio en la máquina host, el segundo parámetro debe ser una ruta absoluta de dónde compartir la carpeta dentro de la máquina invitada.

```
config.vm.provider "virtualbox" do |vb|

vb.name = "VM-debian1"
vb.memory = "1024"
vb.cpus = 2

end
```

Con esta declaración se define los proveedores, también llamados hipervisores, vagrant soporta varios, algunos de los más populares son Virtualbox, VMware, Hyper-V, QEMU, etc. Para los propósitos de este proyecto se utiliza Virtualbox.

También se definen parámetros extra, como nombre de la máquina virtual y uso de memoria RAM y CPU.

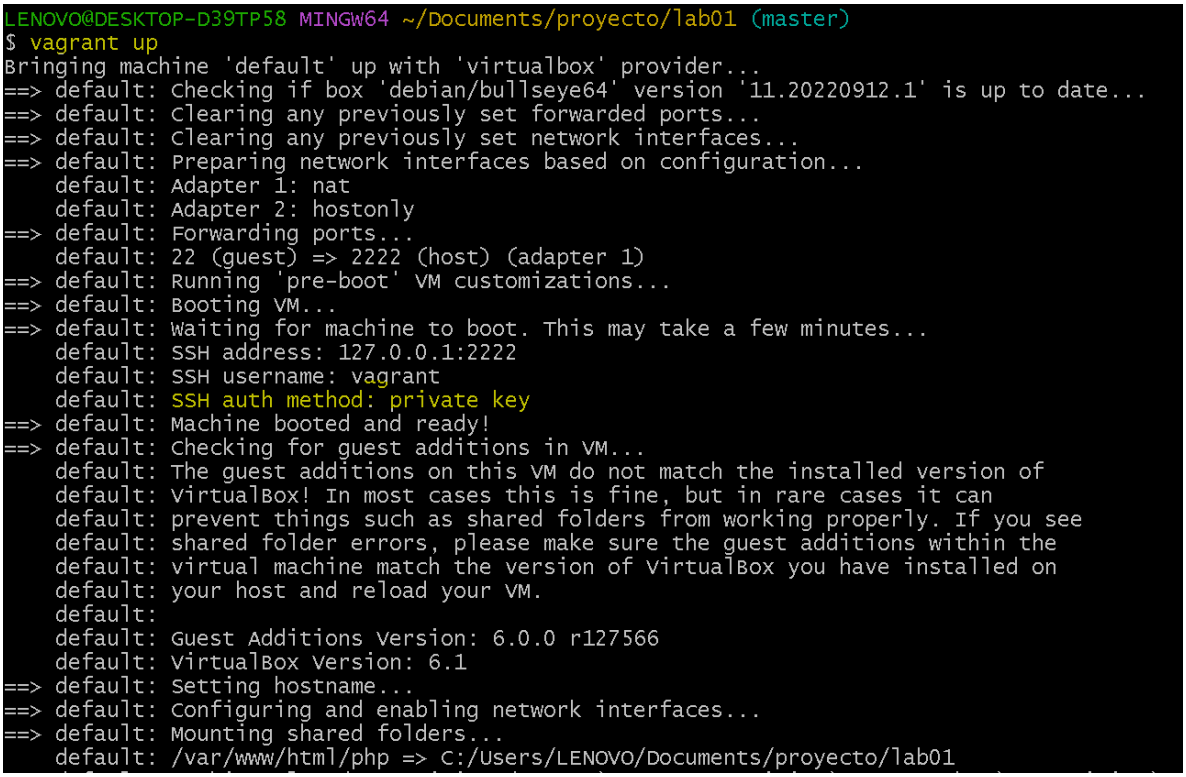
```
config.vm.provision "shell", inline: <<-SHELL
  # Instalar Stack LAMP
    sudo apt update
    sudo apt upgrade -y
    sudo apt install apache2 mariadb-server php7.4 libapache2-mod-php7.4 php-mysql php7.4-cli -y
SHELL
```

Con esta declaración se define el aprovisionador que permite instalar software y personalizar la configuración del entorno virtual. Para propósitos del proyecto y para simplificar su funcionamiento utilizaremos como aprovisionador scripts de Shell, en este caso instalamos solo los paquetes necesarios para la pila LAMP.

8.1.3 Comandos para iniciar el entorno:

Para iniciar el entorno, es necesario, ubicarse en el directorio del laboratorio 1, en el cual se encuentra el archivo Vagrantfile y ejecutar el comando ***vagrant up***.

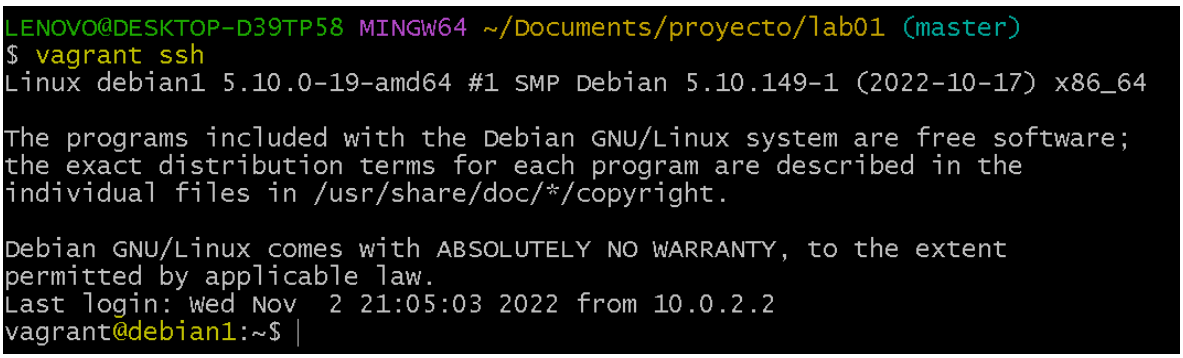
Figura 11. *Iniciar entorno laboratorio 1.*



Nombre de la fuente: (Elaboración propia).

Como se observa Vagrant administra la conexión a la máquina a través de llaves SSH automáticamente, para ingresar a la máquina usamos el comando ***vagrant ssh***.

Figura 12. *Ingresar entorno laboratorio 1 con SSH.*

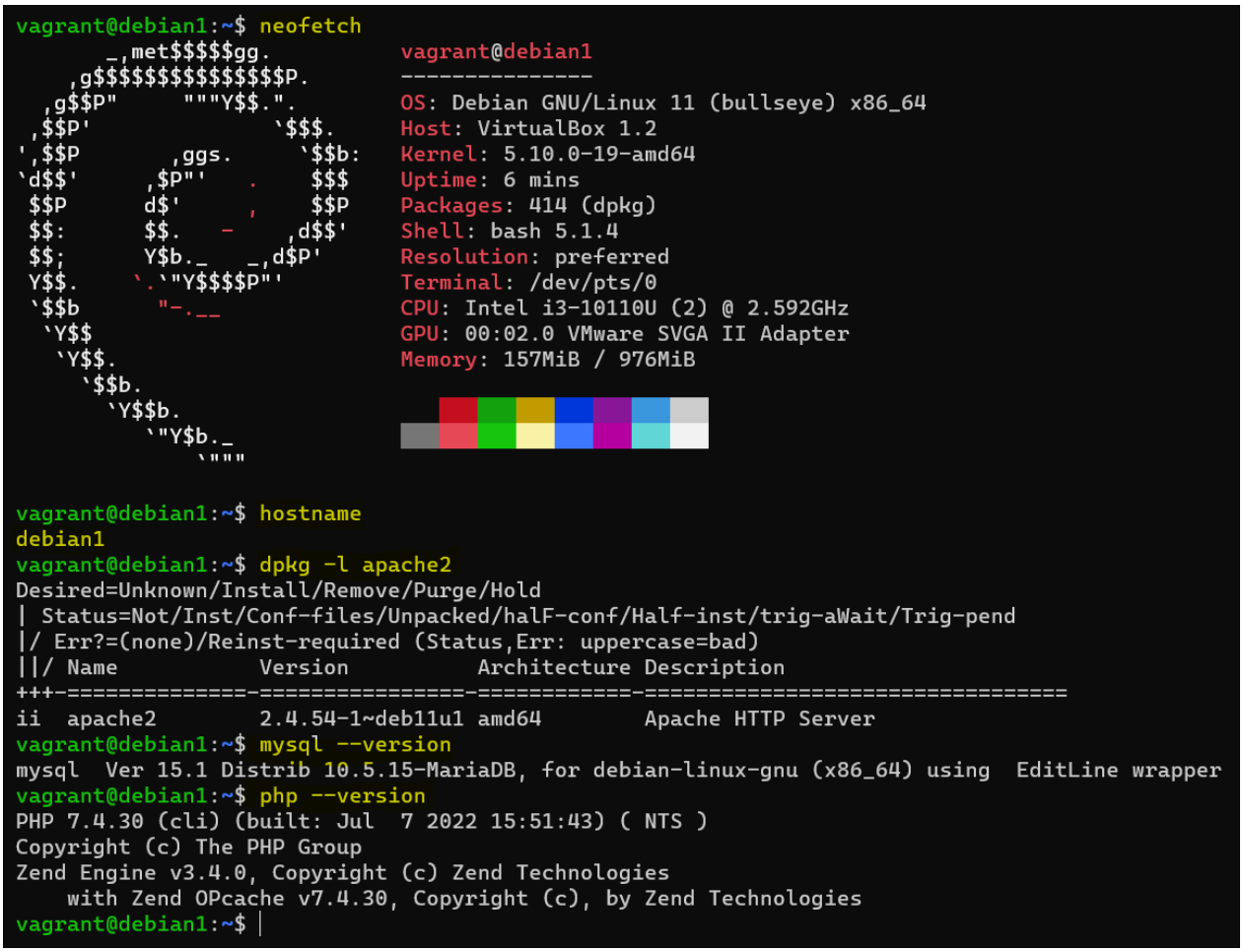


Nombre de la fuente: (Elaboración propia).

8.1.4 Comandos para verificar la configuración del entorno:

Comprobar que la pila LAMP se instaló correctamente.

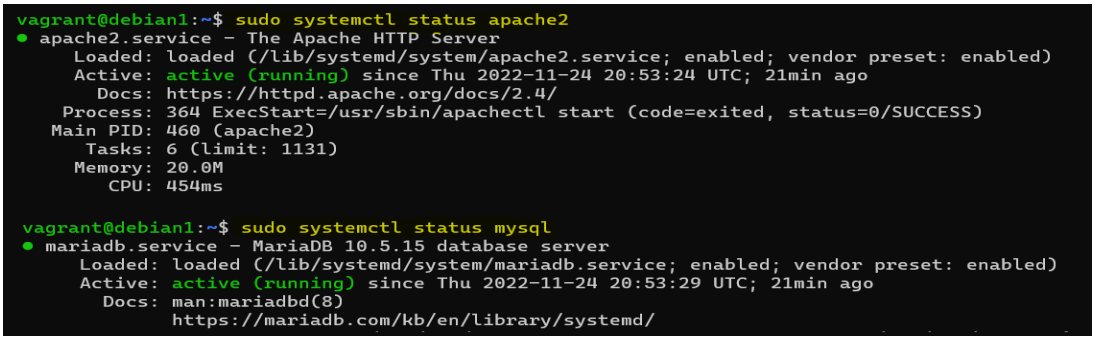
Figura 13. *Verificación Configuración del Entorno y Pila LAMP instalada.*



Nombre de la fuente: (Elaboración propia).

Comprobar que los demonios de apache y MySQL se están ejecutando.

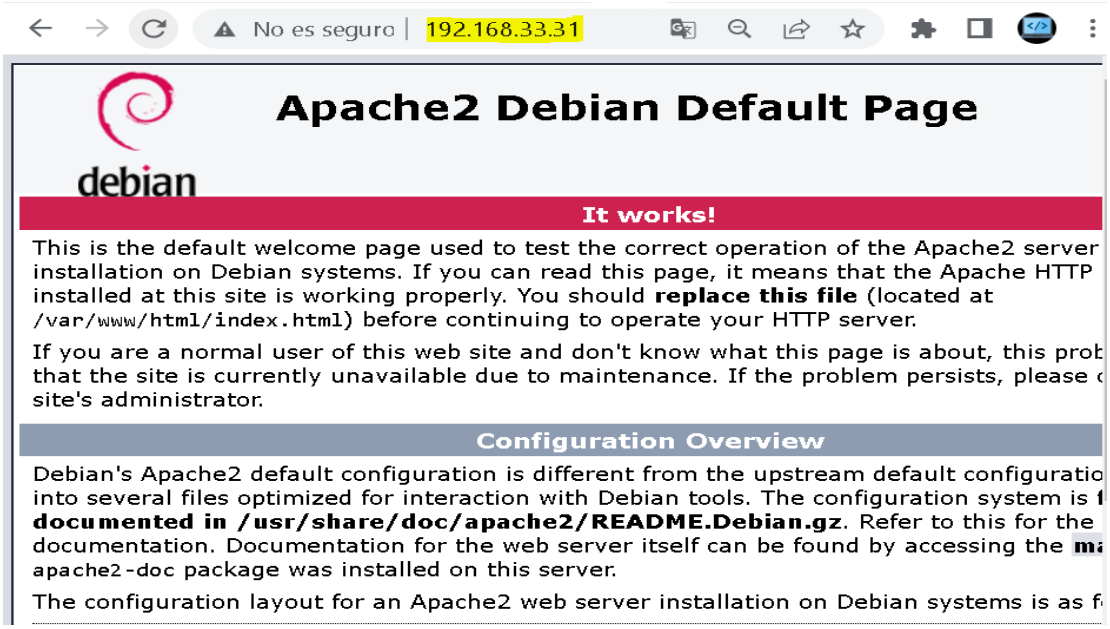
Figura 14. *Demonios apache y MySQL corriendo.*



Nombre de la fuente: (Elaboración propia).

8.1.5 Servicios corriendo correctamente:

Figura 15. *Servidor apache ejecutándose.*



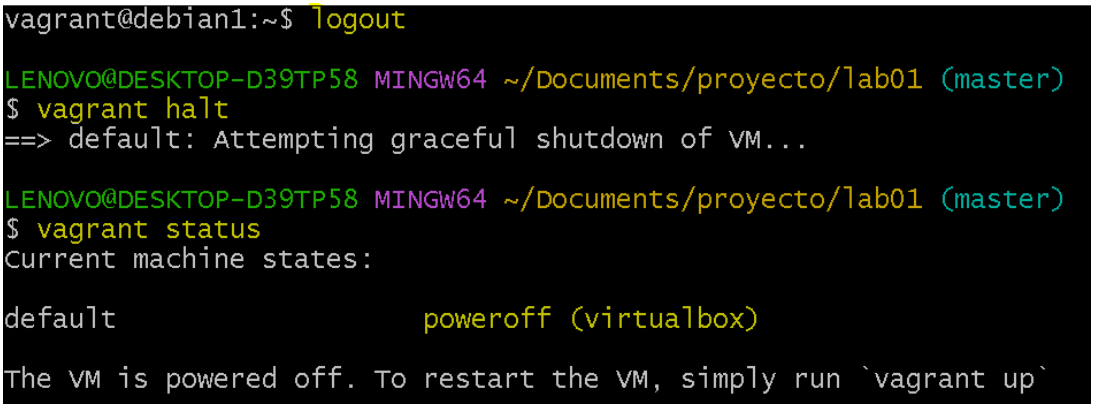
Nombre de la fuente: (Elaboración propia).

Figura 16. *Módulos PHP ejecutándose.*

Nombre de la fuente: (Elaboración propia).

Finalmente se puede cerrar sesión, apagar la máquina y tenerla lista para cuando se necesite de nuevo. Sí se realiza alguna modificación en el archivo *Vagrantfile* se puede usar el comando ***vagrant reload***, para que tome los nuevos cambios sin crear de nuevo la máquina.

Figura 17. *Deteniendo el entorno y apagando la máquina Laboratorio 1.*



Nombre de la fuente: (Elaboración propia).

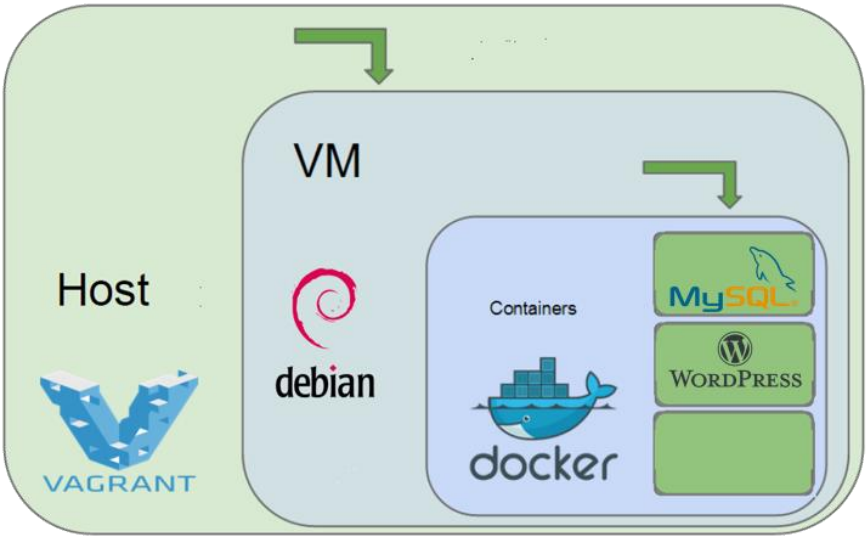
8.2 LABORATORIO 2:

8.2.1 Descripción Componentes del Entorno:

Este laboratorio configura y despliega un entorno de desarrollo con el *CMS WordPress* y la base de datos *MySQL*, a diferencia del laboratorio 1, no se instalarán los servicios a nivel de sistema, si no que se desplegaran sobre una red de contenedores Docker, como se muestra en la **Figura 18.**

WordPress es un CMS, Sistema de Gestión de Contenidos más popular del mundo, utilizado para administrar sitios web, blogs, tiendas online, portales de noticias, áreas de miembros y otros tipos de páginas web.

Figura 18. *Laboratorio 2.*



Nombre de la fuente: (Elaboración propia).

8.2.2 Descripción Archivos de Configuración:

A continuación, se detallan las configuraciones del archivo **Vagrantfile** laboratorio 2:

```
config.vm.box = "debian/bullseye64"
```

Con esta declaración se define el sistema base *Debian 11 bullseye.*

```
config.vm.hostname = "debian2"
```

Con esta declaración se define el nombre del Host de la máquina.

```
config.vm.network "private_network", ip: "192.168.33.32"
```

Con esta declaración se define el adaptador de RED o NIC de la máquina en modo NAT, con la dirección IP “**192.168.33.32".**

```
config.vm.synced_folder ".", "/home/vagrant/docker"
```

Con esta declaración se define las carpetas sincronizadas, entre la máquina anfitrión y la máquina invitada.

```
config.vm.provider "virtualbox" do |vb|

  vb.name = "VM-debian2"
  vb.memory = "1024"
  vb.cpus = 2

end
```

Con esta declaración se define memoria RAM y CPU y el proveedor en este caso Virtualbox.

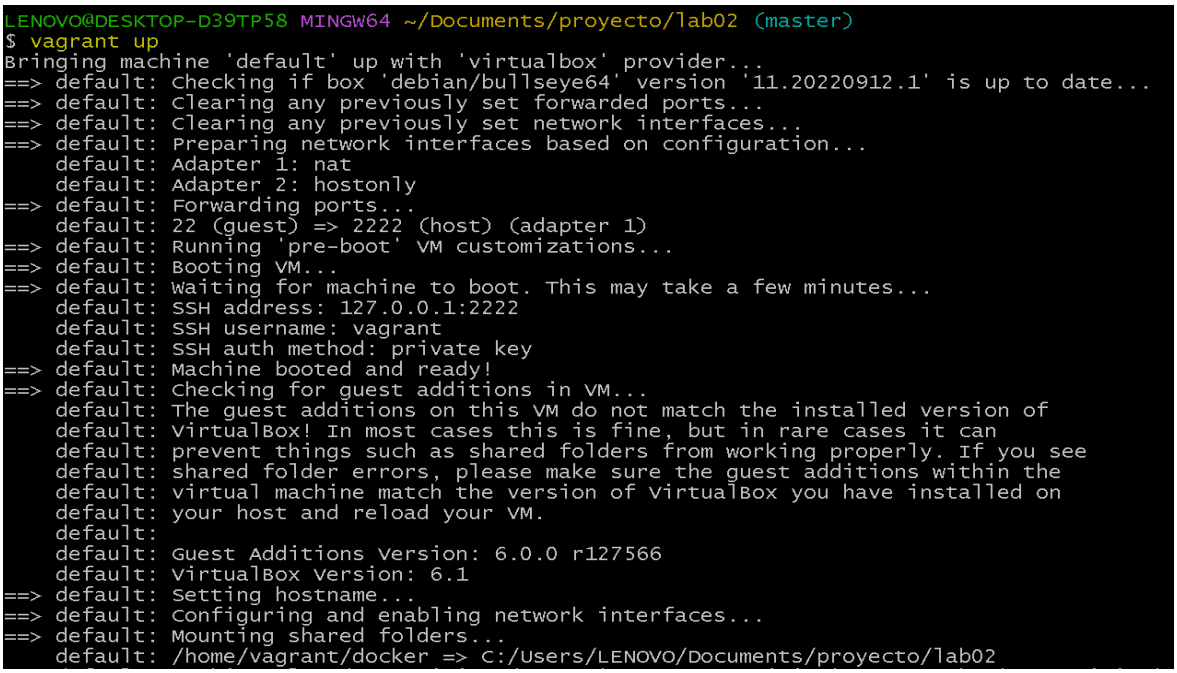
```
config.vm.provision "shell", inline: <<-SHELL
  sudo apt update
  sudo apt upgrade -y
SHELL

config.vm.provision :docker
```

Con esta declaración se instala Docker.

8.2.3 Comandos para iniciar el Entorno:

Para iniciar el entorno, es necesario, ubicarse en el directorio del laboratorio 2, en el cual se encuentra el archivo Vagrantfile y ejecutar el comando ***vagrant up***.

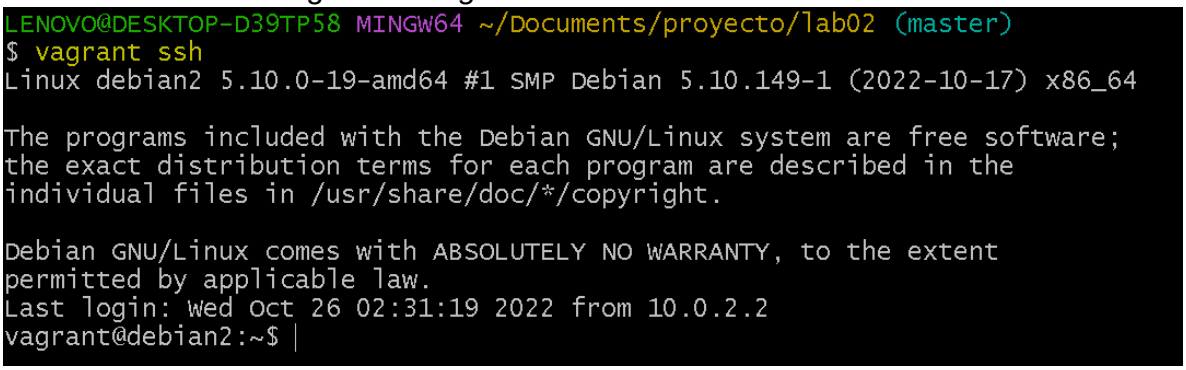
Figura 19. *Iniciar entorno laboratorio 2.*



Nombre de la fuente: (Elaboración propia).

Para ingresar a la máquina se usa el comando **vagrant ssh**.

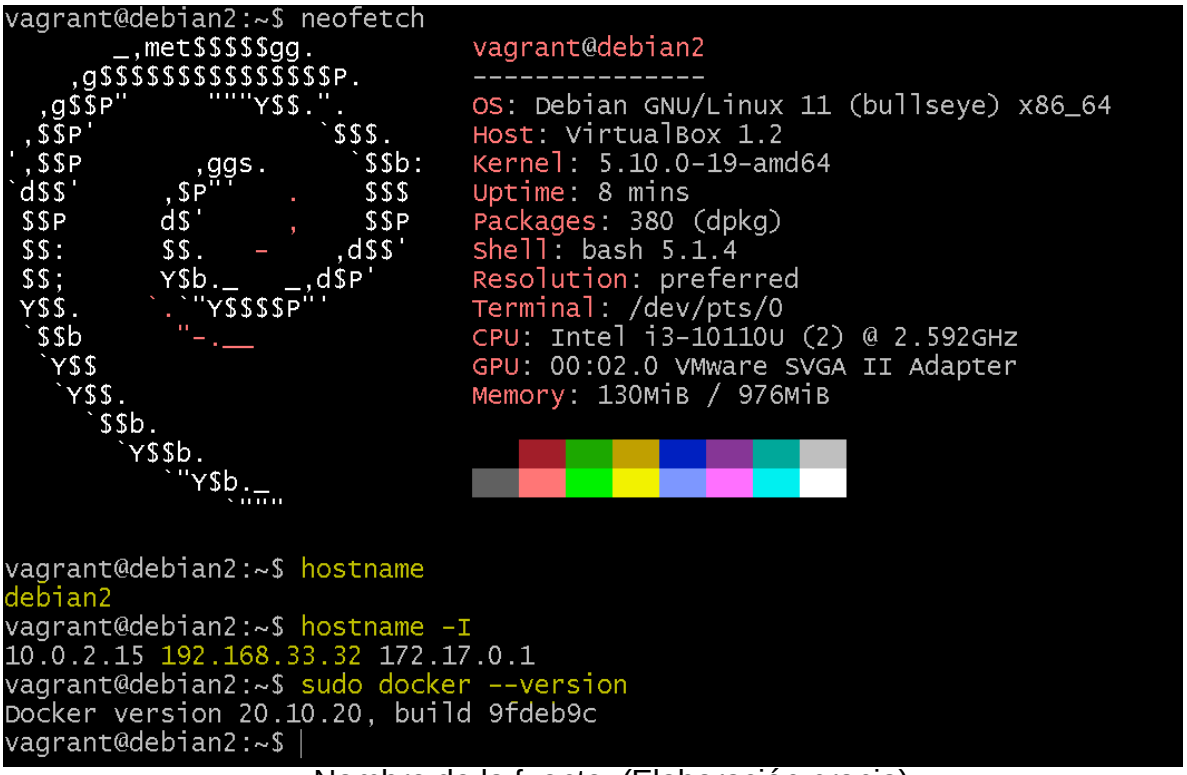
Figura 20. *Ingresar entorno laboratorio 2 con SSH.*



Nombre de la fuente: (Elaboración propia).

8.2.4 Comandos para verificar la configuración del entorno:

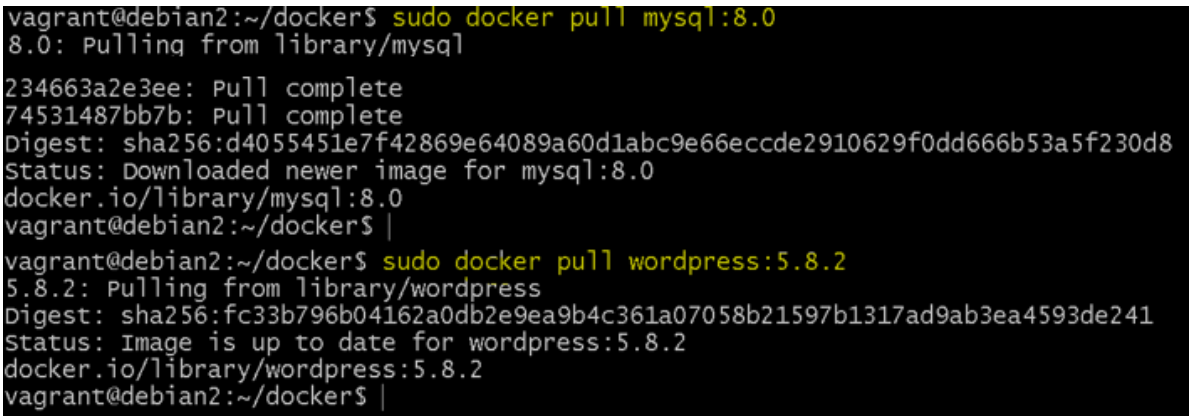
Figura 21. *Verificación Configuración del Entorno e instalación Docker.*



Nombre de la fuente: (Elaboración propia).

Como se puede ver, lo único que se instaló en la máquina fue Docker, ahora se puede descargar las imágenes de MySQL y WordPress, con el comando ***sudo docker pull****, desde dockerhub.*

Figura 22. *Descargando Imágenes Docker de MYSQL y WordPress.*



Nombre de la fuente: (Elaboración propia)

Para iniciar los contenedores se ejecutan los siguientes comandos, primero se inicia el contenedor con MySQL y luego el de WordPress ya que este depende del primero.

COMANDO 1:

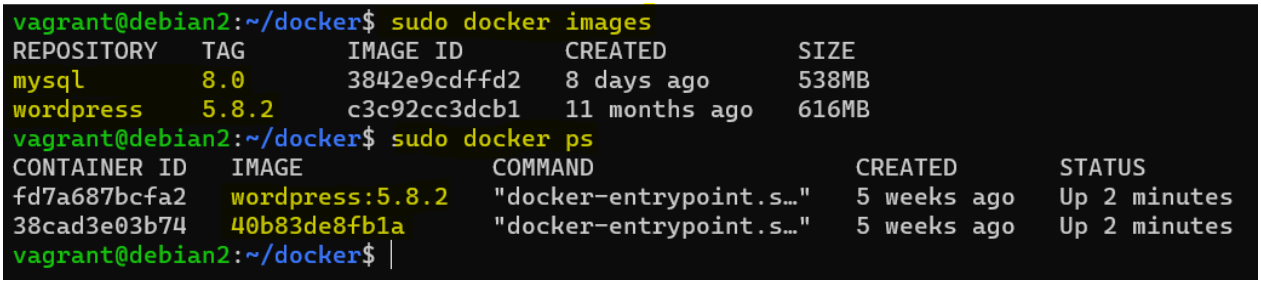
```
docker run -d --name mysqlwp -e MYSQL_ROOT_PASSWORD=wordpresspwd -e MYSQL_DATABASE=wordpress_db -e MYSQL_USER=wordpress_user -e MYSQL_PASSWORD=wordpress123 mysql:8.0
```

COMANDO 2:

```
docker run -d --name wordpress --link mysqlwp:mysql -p 80:80 -e WORDPRESS_DB_NAME=wordpress_db -e WORDPRESS_DB_USER=wordpress_user -e WORDPRESS_DB_PASSWORD=wordpress123 wordpress:5.8.2
```

Se verifica que la red con los dos contenedores está corriendo.

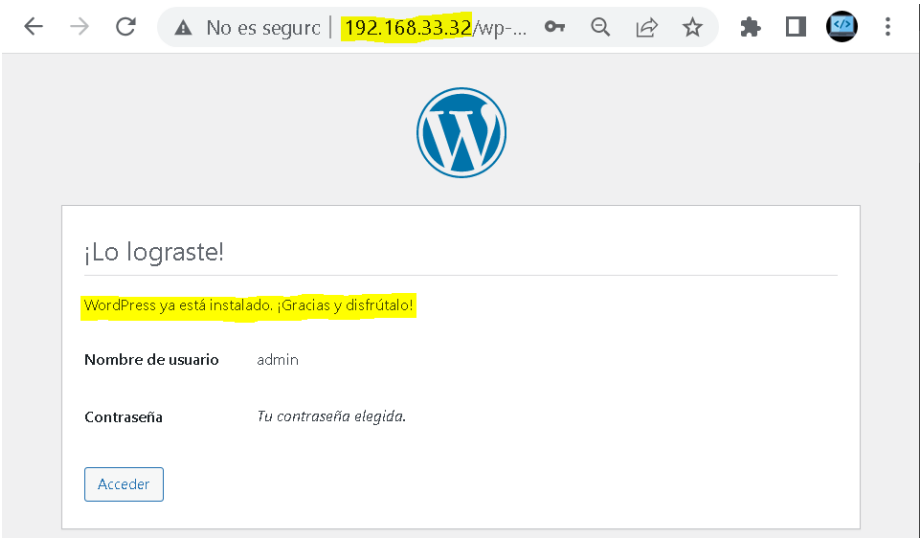
Figura 23. *Verificación red de contenedores corriendo MySQL y WordPress.*



Nombre de la fuente: (Elaboración propia).

8.2.5 Servicios corriendo correctamente:

Figura 24. *WordPress Ejecutándose Correctamente.*



Nombre de la fuente: (Elaboración propia).

Se pueden detener los contenedores, cerrar sesión y apagar la máquina.

Figura 25. *Deteniendo el entorno y apagando la máquina Laboratorio 2.*

```
vagrant@debian2:~/docker$ sudo docker stop 38cad3e03b74 fd7a687bcfa2
38cad3e03b74
fd7a687bcfa2
vagrant@debian2:~/docker$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
vagrant@debian2:~/docker$ logout

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab02 (master)
$ vagrant halt
==> default: Attempting graceful shutdown of VM...
tcsetattr: Input/output error
Connection to 127.0.0.1 closed by remote host.

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab02 (master)
$ vagrant status
Current machine states:

default                   poweroff (virtualbox)

The VM is powered off. To restart the VM, simply run `vagrant up`
```

Nombre de la fuente: (Elaboración propia).

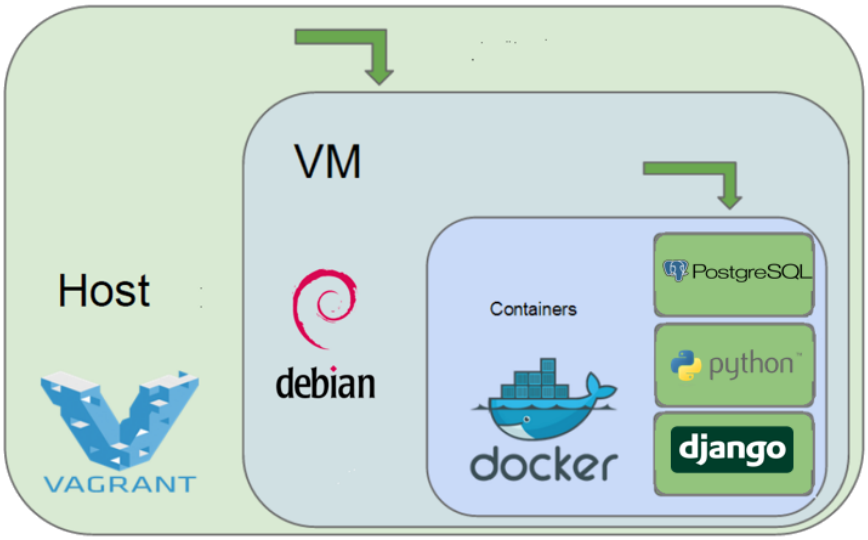
8.3 LABORATORIO 3:

8.3.1 Descripción Componentes del Entorno:

Este laboratorio configura y despliega un entorno de desarrollo con la base de datos PostgreSQL, el lenguaje de programación Python y el framework Django, a diferencia de los laboratorios anteriores, no se instalarán los servicios a nivel de sistema, si no que se desplegaran sobre una red de contenedores usando **docker-compose**, como se muestra en la **Figura 26.**

Django es un framework para desarrollo web gratuito y de código libre, escrito en el lenguaje de programación Python. Se basa en el patrón de diseño MVT.
PostgreSQL es uno de los sistemas de bases de datos relacionales, más usados del mundo, es totalmente gratuito y de código abierto, es considerado el motor de base de datos más avanzado en la actualidad.

Figura 26. *Laboratorio 3.*



Nombre de la fuente: (Elaboración propia).

8.3.2 Descripción Archivos de Configuración:

A continuación, se detallan las configuraciones del archivo **Vagrantfile** laboratorio 3:

```
config.vm.box = "debian/bullseye64"
```

Con esta declaración se define el sistema base Debian 11 bullseye.

```
config.vm.hostname = "debian3"
```

Con esta declaración se define el nombre del Host de la máquina.

```
config.vm.network "private_network", ip: "192.168.33.33"
```

Con esta declaración se define el adaptador de RED o NIC de la máquina en modo NAT, con la dirección IP “192.168.33.33".

```
config.vm.synced_folder ".", "/home/vagrant/docker"
```

Con esta declaración se define las carpetas sincronizadas, entre la máquina anfitrión y la máquina invitada.

```
config.vm.provider "virtualbox" do |vb|

  vb.name = "VM-debian3"
  vb.memory = "1024"
  vb.cpus = 2

end
```

Con esta declaración se configura la memoria RAM y CPU, definimos el proveedor Virtualbox.

```
config.vm.provision "shell", inline: <<-SHELL
    sudo apt update
    sudo apt upgrade -y
    sudo apt install docker-compose -y
SHELL
```

```
config.vm.provision :docker
```

Con esta declaración se instala Docker y docker-compose.

A continuación, se detallan las configuraciones del archivo **Dockerfile** del laboratorio 3.

```
FROM python:3.6
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
WORKDIR /code
COPY requirements.txt /code/
RUN pip install -r requirements.txt
COPY . /code/
```

A continuación, se detallan las configuraciones del archivo **docker-compose.yml**

```
version: "3.7"

services:
  db:
    image: postgres
    volumes:
      - ./data/db:/home/vagrant/docker
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/code
    ports:
      - "80:8000"
    environment:
      - POSTGRES_NAME=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
    depends_on:
      - db
```

Básicamente se definen 2 servicios:

- ***Db:***
  - Usa una imagen base de PostgreSQL.
  - Se define un volumen para persistencia de datos, similar a los directorios sincronizados de vagrant.
  - Se establecen unas variables ambientales, para configurar la base de datos.

- ***Web:***
  - Construye una imagen base a partir del archivo Dockerfile.
  - Ejecuta el servidor de prueba de Django.

- Se define un volumen.
- Se exponen los puertos, entre el host y el contenedor para que este sea visible.
- Se establecen las variables ambientales que configuran la base de datos, previamente se debió haber configurado los parámetros de PostgreSQL en el archivo de configuración de Django *settings.py*.

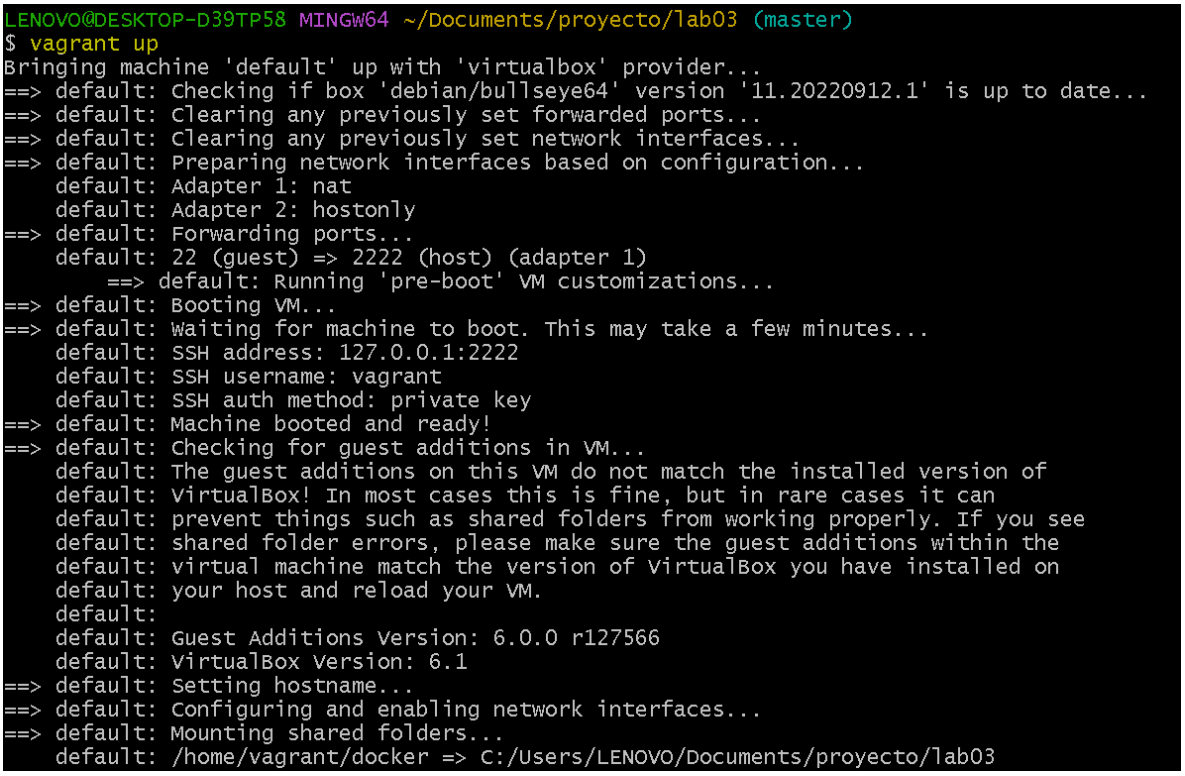
```
ALLOWED_HOSTS = ["*"]

DATABASES = {
        'default': {
'ENGINE': 'django.db.backends.postgresql',
'NAME': os.environ.get('POSTGRES_NAME'),
'USER': os.environ.get('POSTGRES_USER'),
'PASSWORD': os.environ.get('POSTGRES_PASSWORD'),
'HOST': 'db',
'PORT': 5432,
        }
}
```

- Docker-compose crea automáticamente la conexión entre los dos contenedores.

8.3.3 Comandos para iniciar el entorno:

Para iniciar el entorno, es necesario, ubicarse en el directorio del laboratorio 3, en el cual se encuentra el archivo Vagrantfile y ejecutar el comando **vagrant up.**

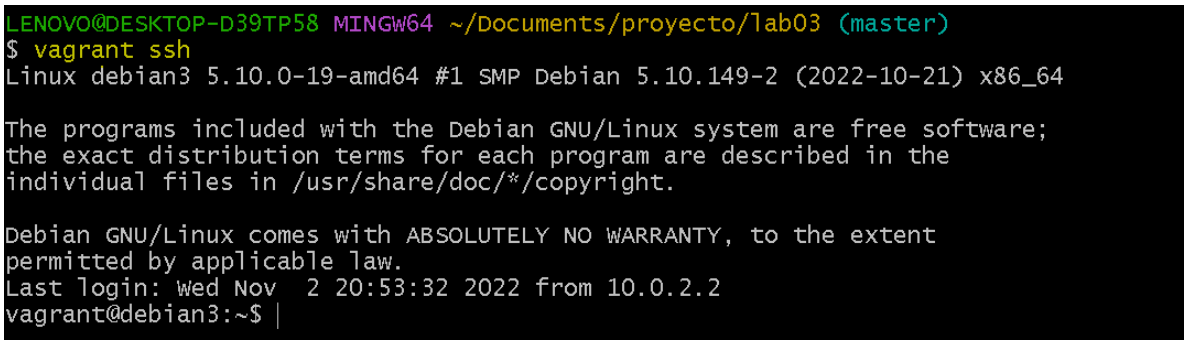
Figura 27. *Iniciar entorno Laboratorio 3.*



Nombre de la fuente: (Elaboración propia).

Para ingresar a la máquina se usa el comando **vagrant ssh.**

Figura 28. *Ingresar entorno laboratorio 3 con SSH.*



Nombre de la fuente: (Elaboración propia).

8.3.4 Comandos para verificar la configuración del entorno:

Figura 29. *Verificación Configuración del Entorno e instalación Docker y docker-compose.*

Nombre de la fuente: (Elaboración propia).

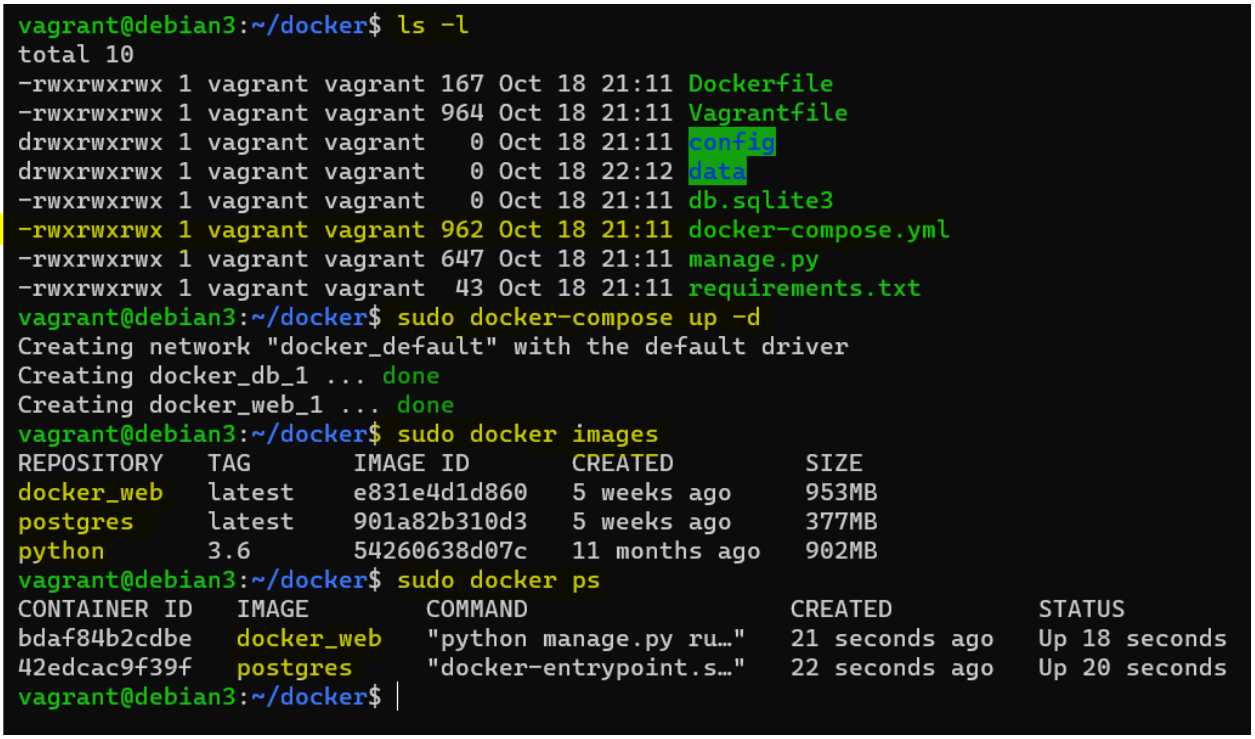
Como se puede ver lo único que se instaló en la máquina a nivel de sistema fue *Docker y docker-compose*, ahora podemos descargar y desplegar los contenedores usando el archivo de configuración de docker-compose el ***docker-compose.yml***

Para desplegar la red con los contenedores, se debe ubicar en el directorio que contiene el archivo ***docker-compose.yml*** y ejecutar el siguiente comando:

sudo docker-compose up -d

Se verifica que la red con los contenedores está corriendo, el framework Django y la base de datos PostgreSQL.

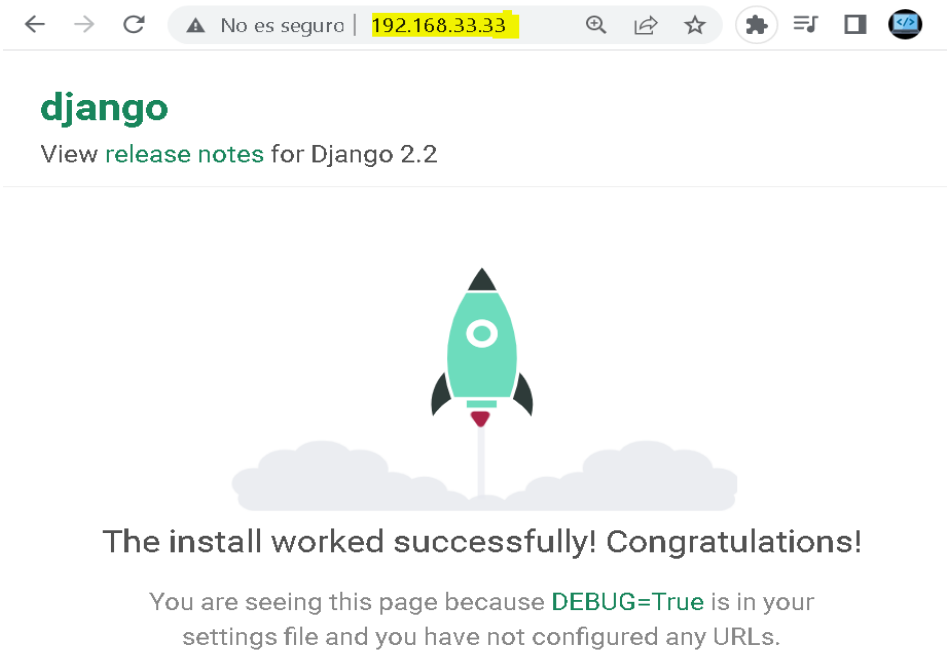
Figura 30. *Verificación red de contenedores corriendo Django y PostgreSQL.*



Nombre de la fuente: (Elaboración propia).

8.3.5 Servicios corriendo correctamente:

Figura 31. *Django y PostgreSQL Ejecutándose Correctamente.*



Nombre de la fuente: (Elaboración propia).

Se pueden detener los contenedores, cerrar sesión y apagar la máquina.

Figura 32. *Deteniendo el entorno y apagando la máquina Laboratorio 3*.

```
vagrant@debian3:~/docker$ sudo docker-compose down
Stopping docker_web_1 ... done
Stopping docker_db_1  ... done
Removing docker_web_1 ... done
Removing docker_db_1  ... done
Removing network docker_default
vagrant@debian3:~/docker$
vagrant@debian3:~/docker$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
vagrant@debian3:~/docker$
vagrant@debian3:~/docker$ logout

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab03 (master)
$ vagrant halt
==> default: Attempting graceful shutdown of VM...

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab03 (master)
$ vagrant status
Current machine states:

default                   poweroff (virtualbox)

The VM is powered off. To restart the VM, simply run `vagrant up`
```

Nombre de la fuente: (Elaboración propia)

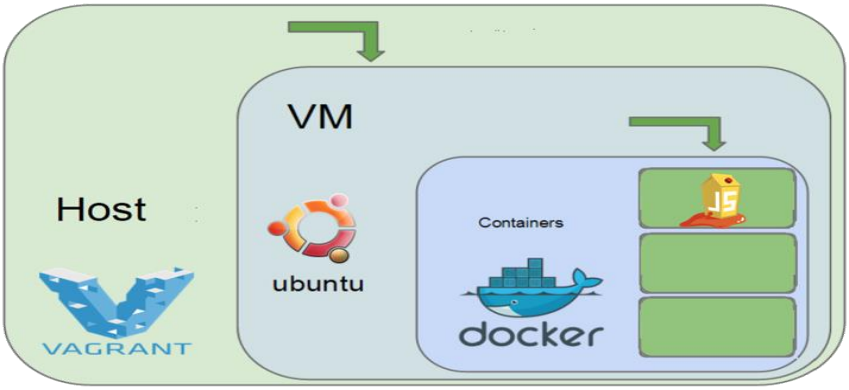
8.4 LABORATORIO 4:

8.4.1 Descripción Componentes del Entorno:

Este laboratorio despliega, un entorno de desarrollo, para la aplicación de tienda de jugos OWASP, sobre una máquina vagrant con ubuntu y un contenedor Docker, como se muestra en la **Figura 33.**

La tienda de jugos OWASP es un proyecto de código abierto, escrita en nodejs y organizado por el Proyecto de Seguridad de Aplicaciones Web Abiertas (OWASP) sin fines de lucro y es desarrollado y mantenido por voluntarios, es una herramienta de concientización, capacitación, demostración y ejercicio para los riesgos de seguridad en los entornos modernos de aplicaciones web.

Figura 33. *Laboratorio 4.*



Nombre de la fuente: (Elaboración propia).

8.4.2 Descripción Archivos de Configuración:

A continuación, se detallan las configuraciones del archivo **Vagrantfile** laboratorio 4:

```
config.vm.box = "generic/ubuntu2110"
```

Con esta declaración se define el sistema base ubuntu.

```
config.vm.hostname = "ubuntu"
```

Con esta declaración se define el nombre del Host de la máquina.

```
config.vm.provider "virtualbox" do |vb|

  vb.name = "VM-ubuntu"
  vb.memory = "1024"
  vb.cpus = 2

end
```

Con esta declaración se configura el Provider, la RAM y CPU.

```
config.vm.network "private_network", ip: "192.168.33.34"
```

Con esta declaración se configura el adaptador de RED o NIC de la máquina en modo NAT, con la dirección IP “192.168.33.34".

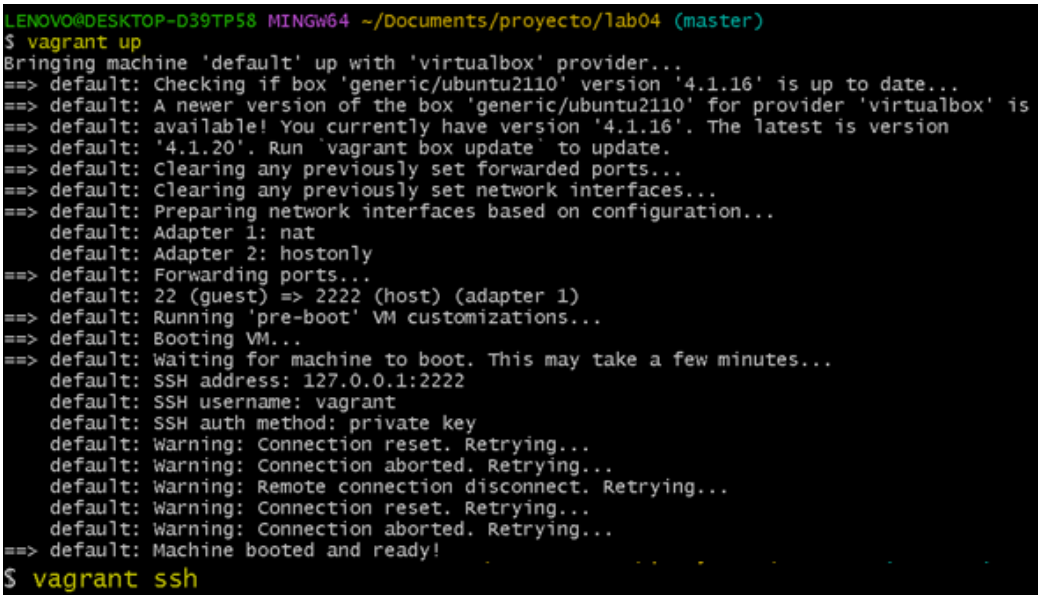
```
config.vm.provision "file", source: "./default.conf", destination: "/tmp/juice-shop/default.conf"
config.vm.provision :shell, path: "bootstrap.sh"
```

Con esta declaración se configura un script de Shell y VirtualHost.

8.4.3 Comandos para iniciar el Entorno:

Para iniciar el entorno, es necesario, ubicarse en el directorio del laboratorio 4, en el cual se encuentra el archivo Vagrantfile y ejecutar el comando **vagrant up** y luego **vagrant ssh** para ingresar.

Figura 34. *Iniciar entorno laboratorio 4.*

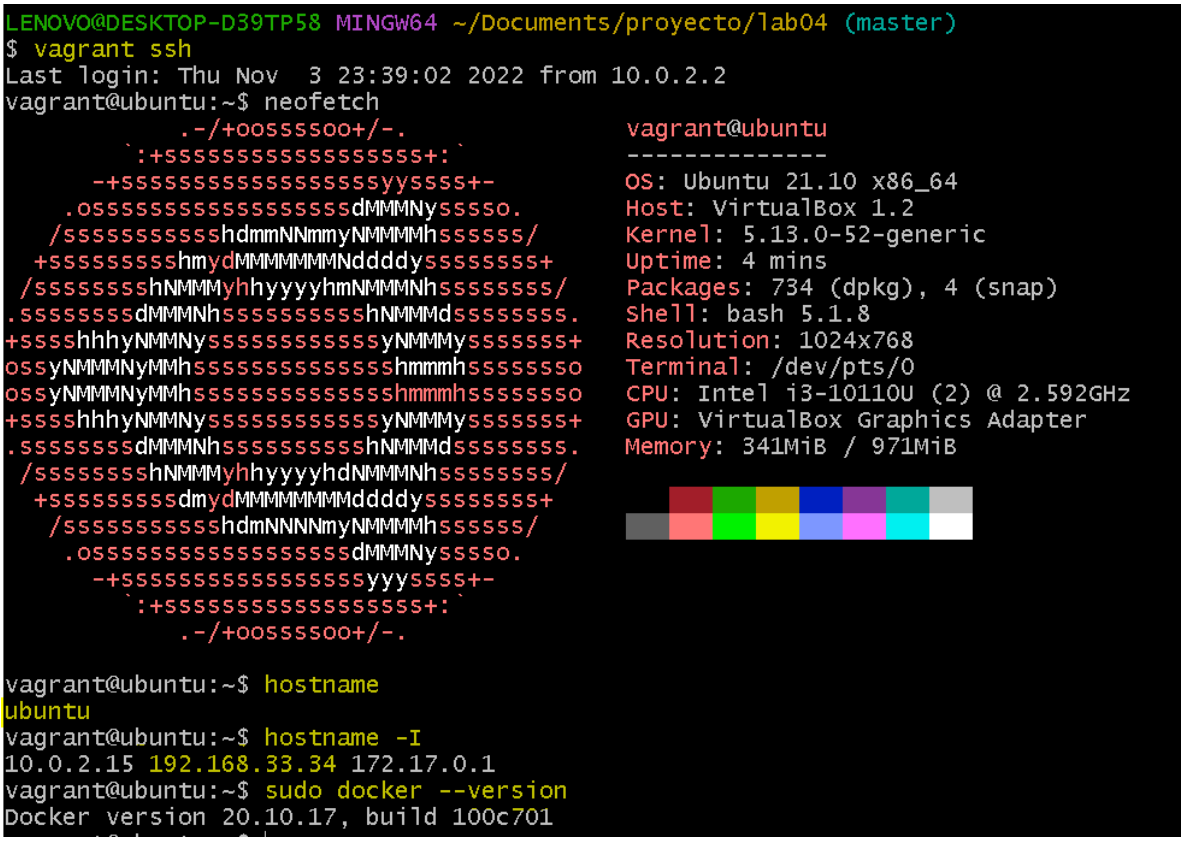


Nombre de la fuente: (Elaboración propia).

8.4.4 Comandos para verificar la configuración del entorno:

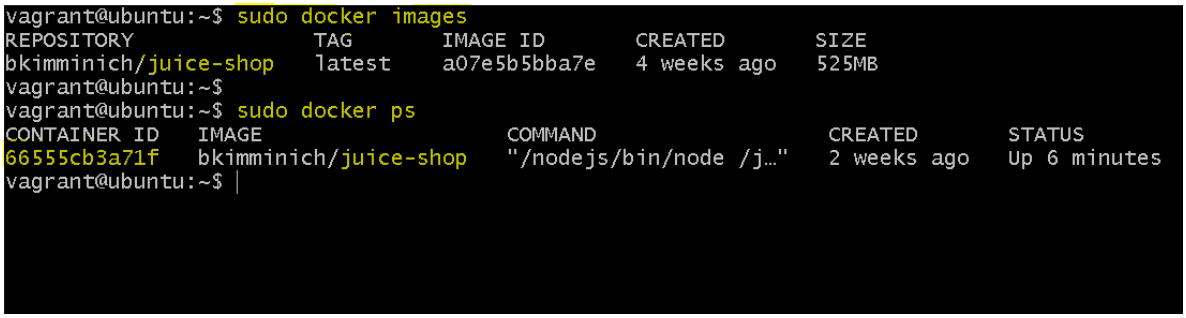
Se ingresa a la máquina y se comprueba que se instaló todo correctamente y que los servicios están en ejecución.

Figura 35. *Verificación Configuración del Entorno e instalación Docker.*



Nombre de la fuente: (Elaboración propia).

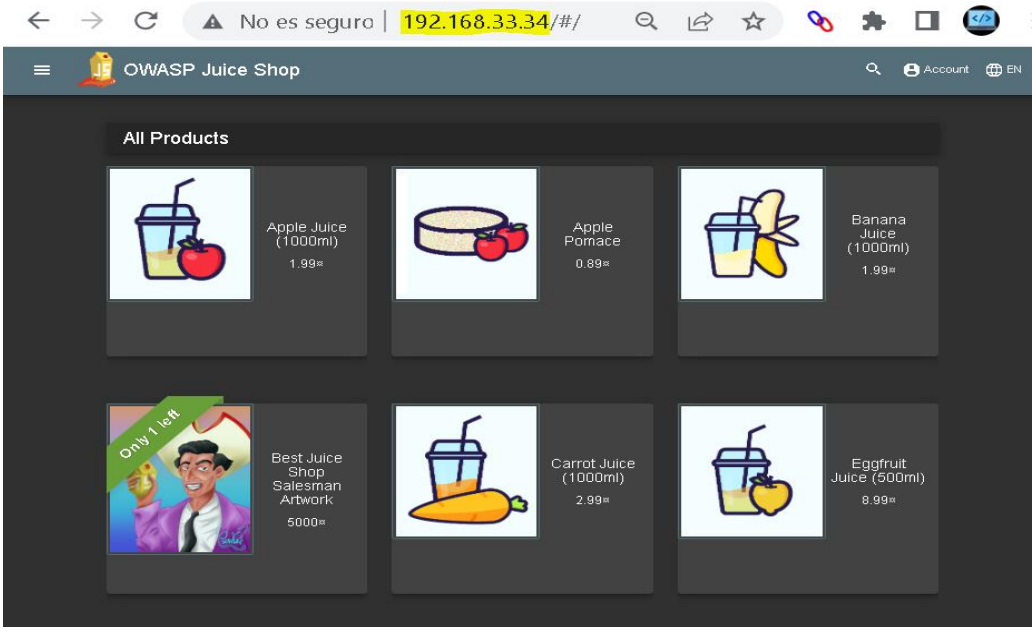
Figura 36. Verificación de contenedores corriendo



Nombre de la fuente: (Elaboración propia).

8.4.5 Servicios corriendo correctamente:

Figura 37. *OWASPJuice Shop Ejecutándose Correctamente.*



Nombre de la fuente: (Elaboración propia).

Se pueden detener los contenedores, cerrar sesión y apagar la máquina.

Figura 38. *Deteniendo el entorno y apagando la máquina Laboratorio 4.*

```
vagrant@ubuntu:~$ sudo docker stop 66555cb3a71f
66555cb3a71f
vagrant@ubuntu:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
vagrant@ubuntu:~$
vagrant@ubuntu:~$ logout

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab04 (master)
$ vagrant halt
```

Nombre de la fuente: (Elaboración propia).

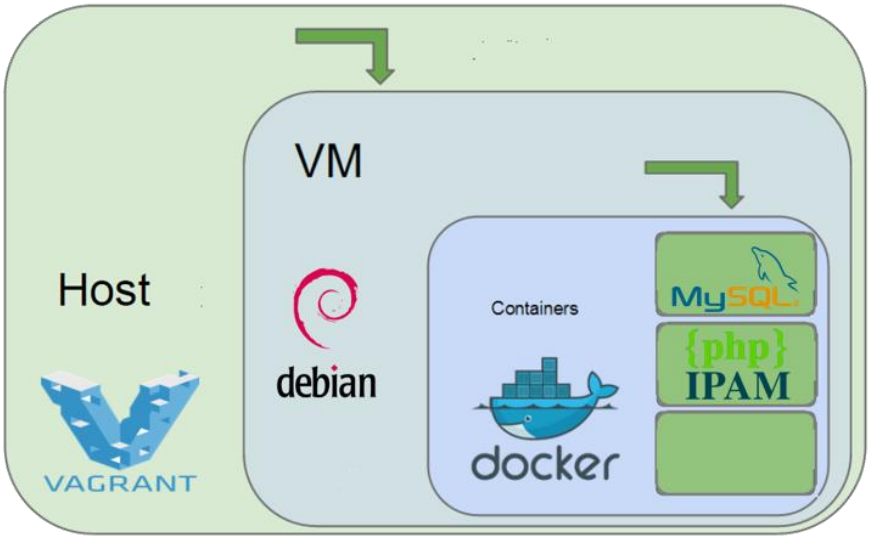
8.5 LABORATORIO 5:

8.5.1 Descripción Componentes del Entorno:

Este laboratorio configura y despliega un entorno de trabajo con el Gestor de direcciones IP PHPipam y la base de datos MySQL, como se muestra en la **Figura 39.**

PHPipam es una aplicación de gestión de direcciones para IPv4 como para IPv6 web (IPAM) de código abierto. Su objetivo es proporcionar una gestión de direcciones IP ligera, moderna y útil. Es una aplicación basada en php con backend de base de datos MySQL.

Figura 39. *Laboratorio 5.*



Nombre de la fuente: (Elaboración propia).

8.5.2 Descripción Archivo de Configuración:

A continuación, se detallan las configuraciones del archivo **Vagrantfile** laboratorio 5:

```
config.vm.box = "debian/bullseye64"
```

Con esta declaración se define el sistema base Debian 11.

```
Config.vm.hostname = “debian4”
```

Se establece un nombre del Host de la máquina.

```
Config.vm.network “private_network”, ip: “192.168.33.35”
```

Configuramos el adaptador de RED o NIC de la máquina en modo NAT, con la dirección IP “192.168.33.35".

```
config.vm.synced_folder ".", "/home/vagrant/docker"
```

Con esta declaración se sincronizan las carpetas, entre la máquina anfitrión y la máquina invitada.

```
config.vm.provider "virtualbox" do |vb|

  vb.name = "VM-debian4"
  vb.memory = "1024"
  vb.cpus = 2

end
```

Con esta declaración se configura la memoria RAM y CPU y se define el proveedor Virtualbox.

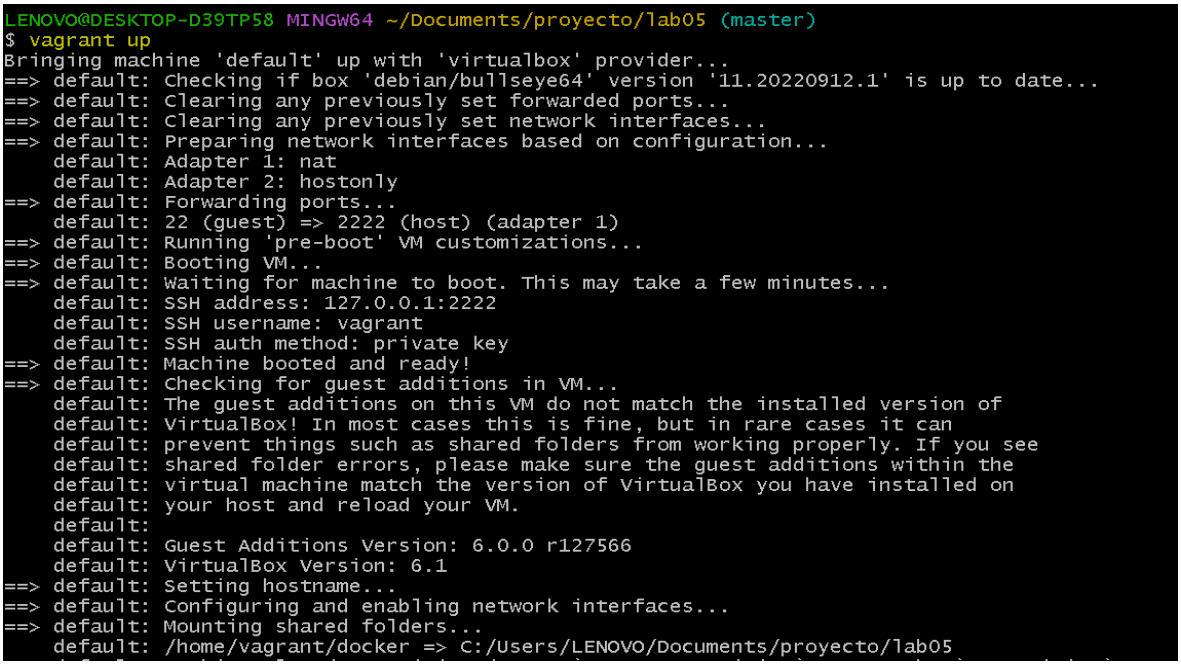
```
config.vm.provision "shell", inline: <<-SHELL
  sudo apt update
  sudo apt upgrade -y
SHELL

config.vm.provision :docker
```

Con esta declaración se Instala Docker.

8.5.3 Comandos para iniciar el Entorno:

Para levantar la máquina, es necesario, ubicarse en el directorio del laboratorio 5, en el cual se encuentra el archivo *Vagrantfile* y ejecutar el comando ***vagrant up***.

Figura 40. *Iniciar entorno Laboratorio 5.*



Nombre de la fuente: (Elaboración propia).

8.5.4 Comandos para verificar la configuración del entorno:

Se ingresa a la máquina y se comprueba que se instaló todo correctamente y que los servicios están en ejecución.

Figura 41. *Verificación Configuración del Entorno e instalación Docker*

Nombre de la fuente: (Elaboración propia).

Se despliega la red con los contenedores, Primero se despliega el contenedor con MySQL y luego el de PHPipam, con los siguientes comandos.

COMANDO 1:

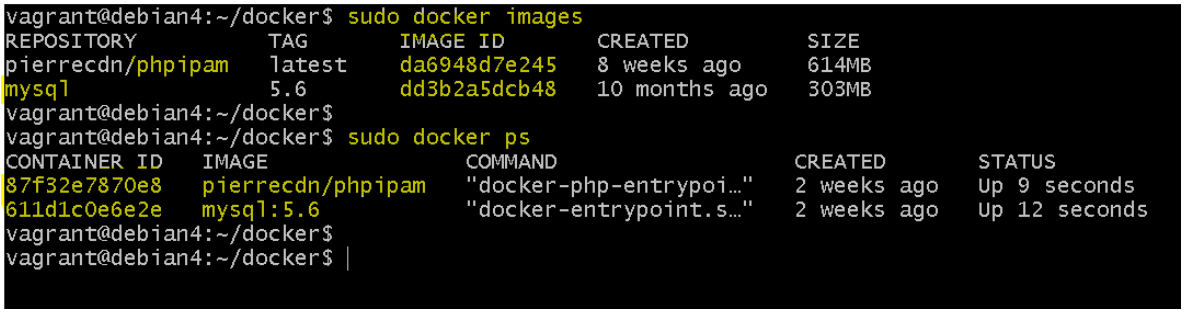
docker run --name phpipam-mysql -e **MYSQL_ROOT_PASSWORD**=my-secret-pw -v /my_dir/phpipam:/var/lib/mysql -d mysql:**5.6**

COMANDO 2:

docker run -d -p **80**:**80** -e **MYSQL_ENV_MYSQL_ROOT_PASSWORD**=my-secret-pw --name ipam --link phpipam-mysql:mysql pierrecdn/phpipam

Se verifica que la red con los dos contenedores está corriendo.

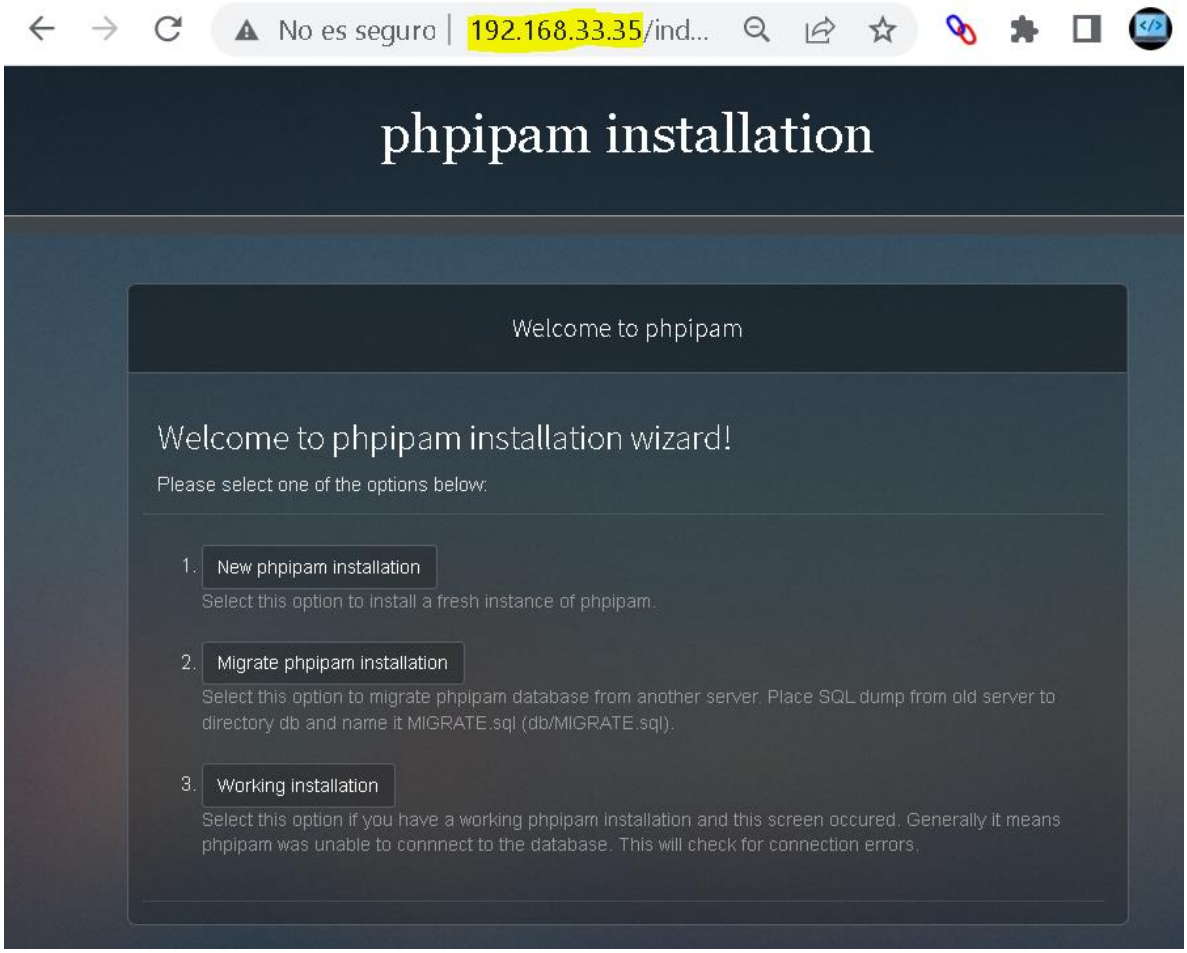
Figura 42. *Verificación Contenedores Corriendo.*



Nombre de la fuente: (Elaboración propia).

8.5.5 Servicios corriendo correctamente:

Figura 43. *PhpIPAM Ejecutándose Correctamente.*



Nombre de la fuente: (Elaboración propia).

Se pueden detener los contenedores, cerrar sesión y apagar la máquina.

Figura 44. *Deteniendo el entorno y apagando la máquina Laboratorio 5.*

```
vagrant@debian4:~/docker$ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED       STATUS
87f32e7870e8   pierrecdn/phpipam  "docker-php-entrypoi…"   2 weeks ago   Up 4 minutes
611d1c0e6e2e   mysql:5.6          "docker-entrypoint.s…"   2 weeks ago   Up 4 minutes
vagrant@debian4:~/docker$ sudo docker stop 611d1c0e6e2e 87f32e7870e8
611d1c0e6e2e
87f32e7870e8
vagrant@debian4:~/docker$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
vagrant@debian4:~/docker$
vagrant@debian4:~/docker$ logout

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab05 (master)
$ vagrant halt
==> default: Attempting graceful shutdown of VM...

LENOVO@DESKTOP-D39TP58 MINGW64 ~/Documents/proyecto/lab05 (master)
$ vagrant status
Current machine states:

default                   poweroff (virtualbox)
```

Nombre de la fuente: (Elaboración propia).

9. RESULTADOS

9.1 Resultado 1

Disminución de consumo de recursos: El uso de contenedores y máquinas virtuales Vagrant puede reducir significativamente el consumo de recursos, como RAM, CPU y disco, en comparación con el uso de máquinas virtuales tradicionales.

El uso eficiente de recursos por parte de Docker y vagrant, permite una mayor densidad de desarrollo en un solo servidor, lo que lleva a una utilización más eficiente de la infraestructura, esta optimización de recursos puede traducirse en ahorros significativos de costos y una mayor escalabilidad, ya que se pueden ejecutar más entornos de desarrollo en el mismo hardware.

Contenedores Docker

sudo docker images

```
ubuntu      18.04     f9a80a55f492   2 months ago   63.2MB
```

sudo docker stats 4671771c5a17

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT    MEM %     NET I/O
4671771c5a17   cool_lumiere  0.00%     5.074MiB / 957.3MiB  0.53%     1.09kB /0B
```

sudo docker inspect 4671771c5a17

Máquina virtual Vagrant

sudo free –h

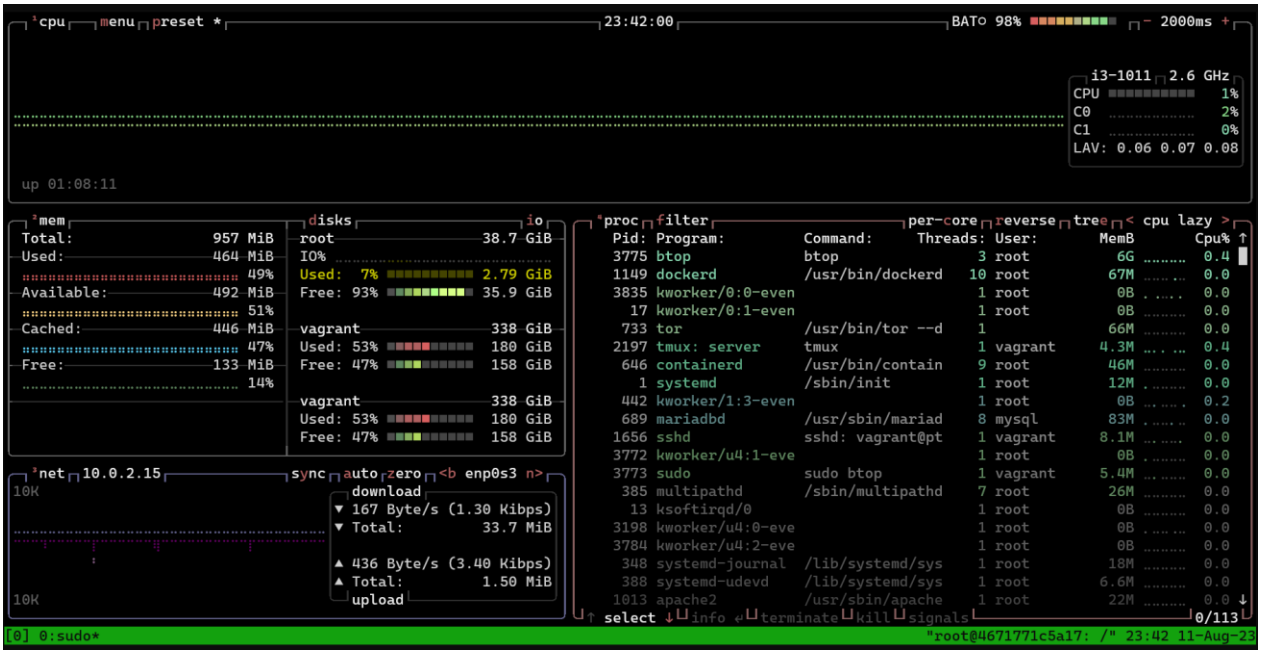
```
               total        used        free      shared  buff/cache   available
Mem:           957Mi       322Mi       635Mi       3.0Mi       497Mi       478Mi
```

sudo lsblk –f

```
NAME   FSTYPE   FSAVAIL FSUSE% MOUNTPOINTS
sda
└─sda1 ext4      35.9G     7% 2.79GB   /
```

sudo btop

Figura 45. Estadísticas consumo de recursos máquina virtual vagrant.BTOP



Nombre de la fuente: (Elaboración propia)

9.2 Resultado 2

Ventajas de estas tecnologías sobre los entornos tradicionales

Ventajas de Docker

Portabilidad: Las imágenes de Docker contienen todo lo necesario para ejecutar una aplicación, incluyendo el código, las bibliotecas y las dependencias. Esto hace que las aplicaciones sean fácilmente portables entre diferentes entornos.

Aislamiento: Docker utiliza contenedores para encapsular aplicaciones y sus dependencias. Esto proporciona un alto grado de aislamiento, lo que significa que las aplicaciones no afectan el entorno subyacente ni entre sí.

Rápida implementación: Docker permite una implementación rápida y consistente de aplicaciones en diferentes entornos. Puedes replicar el mismo entorno de desarrollo, prueba y producción utilizando las mismas imágenes.

Eficiencia de recursos: Los contenedores comparten el kernel del sistema operativo subyacente, lo que hace que sean más livianos en comparación con las máquinas virtuales tradicionales. Esto lleva a un uso más eficiente de los recursos del sistema.

Gestión de versiones y rollbacks: Docker facilita la gestión de versiones de aplicaciones. Puedes versionar imágenes y revertir a versiones anteriores si es necesario, facilitando el mantenimiento y la resolución de problemas.

Escalabilidad: Docker facilita la escalabilidad mediante la creación y ejecución de múltiples contenedores. Puedes escalar horizontalmente agregando instancias de contenedores según sea necesario.

Ventajas de Vagrant

Configuración fácil y reproducible: Vagrant permite la creación de entornos de desarrollo reproducibles mediante la definición de configuraciones en un archivo. Esto facilita la creación de entornos de desarrollo consistentes en diferentes máquinas.

Entornos de desarrollo idénticos: Vagrant ayuda a garantizar que todos los miembros del equipo tengan entornos de desarrollo idénticos, evitando problemas de "funciona en mi máquina" y mejorando la colaboración.

Aislamiento de máquinas virtuales: Vagrant utiliza máquinas virtuales para crear entornos aislados. Cada proyecto puede tener su propia máquina virtual con configuraciones y dependencias específicas.

Integración con múltiples proveedores: Vagrant es compatible con varios proveedores de virtualización, como VirtualBox, VMware, y otros. Esto brinda flexibilidad para elegir el entorno de virtualización que mejor se adapte a tus necesidades.

Gestión de recursos: Aunque Vagrant utiliza máquinas virtuales, proporciona un equilibrio entre la virtualización completa y la eficiencia de recursos, permitiendo una mayor flexibilidad en la asignación de recursos.

Reversión rápida: Vagrant facilita la reversión rápida a un estado anterior del entorno, lo que es útil para probar diferentes configuraciones o para deshacer cambios problemáticos.

9.3 Resultado 3

Reducción del tiempo necesario para configuraciones: Docker y vagrant facilitan la reproducibilidad de entornos de desarrollo, permitiendo a los desarrolladores compartir configuraciones específicas de manera fácil y garantizar que otros puedan replicar exactamente el mismo entorno, permitiendo a los desarrolladores ejecutar sus aplicaciones en diversos sistemas operativos sin preocuparse por las diferencias de configuración.

9.4 Resultado 4

Reducción de errores y conflictos: Vagrant y docker contribuyen a la reducción de los errores pero "en mi máquina si funciona" y facilitan la identificación y corrección de problemas, mejorando la calidad del desarrollo de software, evitando problemas asociados con las diferencias de configuración entre entornos.

10. CONCLUSIONES

Docker y Vagrant son herramientas que permiten automatizar la gestión de configuración y despliegue de entornos de trabajo o desarrollo, permiten crear aplicaciones en contenedores que son ligeros, portátiles y autosuficientes, que permiten agilizar una serie de procesos operativos, enfocando el tiempo en el desarrollo de soluciones de forma eficiente y a menor costo.

La comparación de consumo de recursos entre el uso de contenedores Docker y máquinas virtuales puede variar dependiendo del proyecto y la infraestructura específicos. Sin embargo el consumo de recursos de un contenedor puedo usar cerca de 60% menos de RAM, y pueden reducir los requisitos de almacenamiento en un 50% y los requisitos de red en un 10%.

Los contenedores introducen una sobrecarga insignificante para el rendimiento del CPU, ideal para equipos más pequeños, al utilizar Docker se ahorran recursos a la hora de levantar un entorno de desarrollo, el entorno corre en estos contenedores totalmente aislado de otros contenedores, comparten los recursos de la máquina sin la sobrecarga que aporta la capa del hipervisor.

La industria ahora se beneficia de un flujo de trabajo de contenedores estandarizado y un amplio ecosistema de herramientas y servicios que se puede ejecutar en cualquier máquina, independientemente del sistema operativo.

11. BIBLIOGRAFIA

AMIT POTDAR, M.; NARAYAN, D. G.; KENGONDC, S. y MOIN MULLAD, M. Performance Evaluation of Docker Container and Virtual Machine. [En Línea]. Recuperado 13 de Agosto de 2020. Disponible en: https://www.sciencedirect.com/science/article/pii/S1877050920311315

DOCKER. Docker customer survey. [Sitio Web]. Recuperado 10 de Julio de 2020. Disponible en:https://www.docker.com/sites/default/files/What%20we%20Learned%20from%20Docker%20Customer%20Survey%202016.pdf

KEITH PEACOCK, M. Packt Publishing. Creating Development Environments with Vagrant. [En Línea]. Recuperado 12 de Marzo 2015. Disponible en: https://www.packtpub.com/product/creating-development-environments-with-vagrant-second-edition/9781784397029

SATHYAJITH, B. Practical Docker with Python. [En Línea]. Recuperado 26 de Julio de 2018. Disponible en: https://books.google.com.co/books/about/Practical_Docker_with_Python.html?id=jpdmDwAAQBAJ

SÉBASTIEN, G. O'Reilly. Docker Cookbook, Solutions and examples for building distributed applications. [En Línea]. Recuperado 4 de Noviembre de 2015. Disponible en: https://books.google.com.co/books/about/Docker_Cookbook.html?id=5sTeCgAAQBAJ

SRINATH REDDY, M. Cloud State University. Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications. [En Línea]. Recuperado 1 de Mayo de 2018. Disponible en: https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1091&context=msia_etds

STUART, C. cisco DevNet. NetDevOps Developer Environments with Vagrant. [En Línea]. Recuperado 18 de Junio de 2018. Disponible en: https://drive.google.com/file/d/1H9JvZE28MNjnW_8ppwQ70D1WiabxXD6c/view

VAGRANT. Vagrant by HashiCorp. [Sitio Web]. Recuperado 20 de Marzo 2021. Disponible en: https://www.vagrantup.com/docs/about