UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA

UNAN – León

Facultad de Ciencias y Tecnología

Departamento de Computación

CREACIÓN DE AMBIENTES AISLADOS DE PRUEBAS PARA LA EJECUCIÓN DE APLICACIONES EN PHPV8, LARAVELV8, .NETV5 Y RUBY ON RAILSV6 EMPLEANDO DOCKER-COMPOSE VERSIÓN 2

Monografía para optar al título de

INGENIERO EN TELEMÁTICA

Autores:

Br. Ana Gabriela Rivera González

Br. Miguel Ángel Osejo Sandoval.

Br. Víctor Eliezer González Reyes.

Tutor:

MSc. Denis Leopoldo Espinoza Hernández

León, Noviembre de 2022

“A LA LIBERTAD POR LA UNIVERSIDAD”

UNIVERSIDAD NACIONAL AUTÓNOMA DE NICARAGUA

UNAN – León

Facultad de Ciencias y Tecnología

Departamento de Computación

CREACIÓN DE AMBIENTES AISLADOS DE PRUEBAS PARA LA EJECUCIÓN DE APLICACIONES EN PHPV8, LARAVELV8, .NETV5 Y RUBY ON RAILSV6 EMPLEANDO DOCKER-COMPOSE VERSIÓN 2

Monografía para optar al título de

INGENIERO EN TELEMÁTICA

Autores:

Br. Ana Gabriela Rivera González

Br. Miguel Ángel Osejo Sandoval.

Br. Víctor Eliezer González Reyes.

Tutor:

MSc. Denis Leopoldo Espinoza Hernández

León, Noviembre de 2022

“A LA LIBERTAD POR LA UNIVERSIDAD”

Resumen

Algunos de los problemas que los estudiantes del Dpto. de Computación de la UNAN-León han enfrentado a lo largo de la carrera es no contar con computadoras con grandes características, y al momento de elaborar algunas tareas, se necesita utilizar máquinas virtuales para montar sus escenarios de desarrollo encontrando muchas dificultades por la falta de recursos en sus ordenadores.

Con la iniciativa de ayudar con esta problemática surge la idea de implementar contenedores Docker en lugar de máquinas virtuales, para elaborar sus tareas asignadas, ya que estos no utilizan muchos recursos de cómputo, usando solo lo mínimo para correr cualquier aplicación y montar un entorno más fluido y fácil.

Siendo el resultado de este trabajo la elaboración de prácticas guiadas para que los estudiantes puedan familiarizarse con esta tecnología, y puedan probar sus aplicaciones de manera local, así como también archivos yml para el manejo de los múltiples contenedores que conforma el entorno de desarrollo.

Dedicatoria

A Dios por ser el inspirador y darnos fuerza para continuar en este proceso y lograr obtener uno de nuestros anhelos más deseados.

A nuestros padres por su amor, su comprensión, paciencia y sacrificios en todos estos años de estudios, gracias a ustedes hemos logrado llegar hasta aquí y convertirnos en lo que somos, dedicamos a ustedes este logro amado padres, como una meta más conquistada.

A nuestros hermanos por estar siempre presentes acompañándonos y por el apoyo moral, que nos brindaron a lo largo de esta etapa de nuestra vida.

Agradecimiento

A Dios sobre todas las cosas porque él ha estado con nosotros en todo momento llenándonos de sabiduría para que lográramos culminar nuestra monografía.

A nuestros padres porque ellos han sido siempre el motor que impulsa nuestros sueños y esperanzas, quienes estuvieron siempre a nuestro lado en los días y noches más difíciles durante horas de estudio, brindándonos su apoyo incondicional y por motivarnos a seguir siempre adelante, por confiar y tener fe en nosotros y por ser luz y guía en nuestro camino.

A nuestro tutor Msc Denis Espinoza quien nos compartió sus conocimientos, asesoró y dirigió en la realización de nuestra monografía.

Índice

Indice de Ilustraciones

1 Introducción

En la actualidad Docker es una herramienta que está obteniendo mucha popularidad en el área de la programación, ya que es una tecnología que facilita el desarrollo de aplicaciones mediante el uso de contenedores, lo que ha venido a transformar la manera en que los desarrolladores, gestionan, ejecutan y comparten sus proyectos en un archivo empaquetado con todas sus dependencias, por ende, esto hace un entorno de desarrollo más productivo y ágil.

El propósito de este trabajo es crear entornos donde pueda correr aplicaciones web independientemente del sistema operativo, implementando Docker y Docker Compose, aprovechando las posibilidades que tiene como herramienta y el aporte a la enseñanza de los estudiantes de las carreras del Dpto. de Computación de la UNAN-León.

Se textualizará conceptos básicos, elaboración de guías con ejemplos y explicaciones en cada paso para poder entender de la mejor manera el uso de los contenedores y la creación de los ambientes aislados de desarrollo.

1.1 Antecedentes

Erdwan Mazariegos y Luis Mora elaboraron una tesis nombrada: **"Implementación de contenedores Docker como herramienta de virtualización liviana para el apoyo del proceso enseñanza-aprendizaje en la carrera de Ingeniería en Telemática, Dpto. Computación, UNAN-León"**, cuyo trabajo tiene como objetivo apoyar al aprendizaje de los estudiantes de la carrera de ingeniería en telemática de la universidad UNAN-León, en este se enfocan en los componentes: Software como un Servicio (SAAS), Comercio Electrónico, Gestión de Red, implementando el uso de contenedores Docker para optimizar recursos de software y hardware utilizando la tecnología de contenedores Docker y Docker-compose creando archivos yaml para facilitar la configuración de dependencias y entorno de desarrollo de las practicas asignadas en los componente curriculares. (Erdwan Mazariego, 2022)

En 2017 Génesis Delgado y Harvin Mendoza desarrollaron el siguiente trabajo: **"Distribución Linux Ubuntu 16.04 que incluya paquetes de software preinstalados utilizados en las asignaturas de las áreas de Redes y Programación de las carreras que ofrece el Departamento de Computación de la UNAN-León"** (Nicaragua), se desarrolló esta tesis con el fin de crear un sistema Linux con todos los programas utilizados en los componentes de redes y programación impartida en las carreras del Departamento de Computación de la UNAN-León. (Genesis Delgado, 2017)

En 2020 Edwin Lacayo, Juan Mejía y Felipe Mejía realizaron el trabajo: **"Desarrollo de una aplicación web para la representación gráfica de topologías de red creadas entre contenedores Docker"** su objetivo es crear una herramienta web que permitiera ver de manera gráfica y poder interactuar de forma sencilla e indirecta con los contenedores a través de una terminal proporcionada por la herramienta creada. Ya que la tecnología Docker permite optimizar recursos, pero nos limita un entorno grafico para visualizar las interconexiones de red que se establecen entre los contenedores, obteniendo como resultado una aplicación web

capaz de exportar la estructura de la red interna que tiene dichos contenedores. (Edwin Lacayo, 2020)

Lilia María Tirado Zatarin, elaboró un trabajo de tesis titulado**: "Implementación de Docker en la gestión del entorno de desarrollo"**, en este trabajo se habla sobre el uso de Vagrant y Docker, haciendo énfasis en este último para el uso de aplicaciones de desarrollo de forma más ligeras.

Su principal objetivo es proponer una estandarización en la implementación de los entornos de desarrollo utilizando la tecnología de Docker para poder aumentar la productividad del desarrollo y reducir los requerimientos de hardware de las máquinas.

Obteniendo como resultado final, que al utilizar Docker se ahorran recursos al momento de levantar un entorno de desarrollo, y que esta, además, es una mejor opción a la hora de desarrollar en máquinas no tan potentes, pues utiliza menos recursos que una máquina virtual (Tirado Zatarain, 2017).

En la Universidad Nacional del Altiplano (Perú), Fredy Huanca (2017) también realizó una tesis llamada: "**Arquitectura para el desarrollo e implementación servicios web**", En este trabajo se identificaron necesidades en el proceso de desarrollo de software e implementación de servicios web, usando los contenedores Docker

El objetivo general fue construir un ambiente de desarrollo para aplicaciones web con estilo arquitectónico basado en Microservicios, para ello implementaremos prototipos de entornos aislados, bajo las tecnologías que proporcionan los contenedores, de esta manera las aplicaciones se ejecutarán de forma local. (Abel, 2017)

1.2 Planteamiento del Problema

En la carrera de Ingeniería en Telemática e Ingeniería en Sistemas de Información del Departamento de Computación UNAN-León los estudiantes suelen desarrollar en múltiples lenguajes y proyectos en diversas versiones esto obliga a tener que instalar distintos paquetes o software en la maquina lo que ralentizan el equipo.

Siendo el verdadero problema no contar con equipos de cómputo capaz de realizar estas tareas sin generarle limitaciones, a los que muchos utilizan las máquinas de los laboratorios o comparten equipos con sus compañeros lo que genera más complicaciones ya que al trabajar un proyecto en diferentes equipos no es tan simple como trasladar el código, pasa que las versiones no concuerdan o no tiene instaladas las dependencias necesarias. por lo que se necesita un mecanismo que instale o ejecute por un momento determinado un entorno de desarrollo y optimice los recursos del computador, como solución tenemos el uso de contenedores lo que va a permitir crear entornos completos de desarrollo y ejecución con todos los requerimientos de software y con la aplicación que el desarrollador este haciendo, utilizando la tecnología Docker para la creación de ambientes de desarrollo aislados del sistema, pero óptimos para desarrollar.

Pregunta General:

- ¿Cuál es la finalidad de crear ambientes aislados para la ejecución y pruebas de aplicaciones web utilizando Docker?

Preguntas Específicas:

- ¿Qué aspectos tomar en cuenta para facilitar la percepción de utilizar ambientes aislados?
- ¿Qué secuencia debe tener el documento de manera que permita a los estudiantes la comprensión y desarrollo de las guías propuestas?
- ¿En qué componentes se podrían implementar el uso de contenedores Docker?

1.3 Justificación

La presente investigación tiene como principal objetivo la elaboración de ambientes aislados para entornos de desarrollo y ejecución usando contenedores Docker, ya que los estudiantes se ven en la necesidad de usar diferentes maquinas parar realizar sus prácticas de ciertos componentes curriculares, esto se hace tomando en cuenta las necesidades de los usuarios que cuentan con computadoras con pocos recursos de hardware y software. En proyectos anteriores se han elaborado estudios sobre el uso de Docker para virtualizaciones y manejos de redes de comunicación, obteniendo un gran desempeño y buena aceptación por docentes y estudiantado.

Se propone crear posibles soluciones a los problemas que se presentan al momento de hacer uso de contenedores, tanto a la hora de instalar las dependencias y hacer uso de las mismas, así como también la implementación en el desarrollo de aplicaciones aisladas.

Este trabajo servirá como material de apoyo para facilitar el entorno de desarrollo de las aplicaciones creadas mediante los lenguajes de programación o Framework (Laravel, PHP, ASP.NET y Ruby on Rails), presentando la solución de problemas concretos, en un área de actividad específica. Esto permitiría no solo aumentar la eficiencia en relación a los proyectos de desarrollo, si no también perfeccionar los planes docentes de enseñanza de la programación y la informática.

2 Objetivos

2.1 Objetivo General

Crear ambientes aislados de pruebas haciendo uso de contenedores Docker para la ejecución de aplicaciones Web desarrolladas en múltiples lenguajes siendo esta una herramienta de apoyo para estudiantes del Dpto. Computación, UNAN-León.

2.2 Objetivos Específicos

- Documentar conceptos relevantes sobre el uso de contenedores Docker.
- Elaborar guías con ejercicios guiados que permitan la comprensión y utilización de la tecnología Docker.
- Desarrollar ficheros yaml de Docker Compose que sirvan como herramienta para la realización de prácticas de desarrollo en los distintos componentes curriculares.

3 Marco Teórico

3.1 Entorno de desarrollo

3.1.1 Definición

Un entorno de desarrollo en el mundo del software y la tecnología es un conjunto de procedimientos y herramientas utilizadas por los desarrolladores para codificar, generar, depurar, actualizar, integrar, testear, validar y ejecutar programas.

Funciona como un espacio de trabajo en el que los cambios se implementan en diferentes entornos hasta que se ponen en marcha en la versión real (versión del usuario). Nos referimos a un entorno de desarrollo al proceso integral de gestión del desarrollo de software. (Equipo Ekon, 2020)

Donde se debe tener en cuenta los siguientes criterios:

- Se debe disponer de herramientas de desarrollo.
- Administrador de código fuente.
- Administración de información entregada para uso de servidores con el formato Acuerdo de Confidencialidad Uso de Herramientas Tecnológicas.
- Puede residir en el computador personal del desarrollador (cuando el proyecto cuente con un solo desarrollador), cuando exista más de un desarrollador debe ser un servidor compartido.
- Utilizar nombres de dominio diferente para los ambientes de producción, pruebas y desarrollo a efectos de evitar confusión durante la ejecución de las pruebas. (INSOR, 2017)

El ambiente de prueba se usa para llevar a cabo pruebas de aceptación de un conjunto cualquiera de aplicaciones construidas en el ambiente de desarrollo.

El propósito principal del entorno de pruebas es simular al entorno de producción con el fin de testear las actualizaciones para asegurar que las mismas no corrompen

la aplicación existente en los servidores en producción. De esta forma se minimizan las caídas de la aplicación en producción. Además, este entorno puede funcionar tanto como demo como para entrenamiento y capacitación de los usuarios.

El término "ambiente" se utiliza en este contexto, en lugar de "máquina de prueba", porque la fase de ambiente de prueba se llevará a cabo en uno o más servidores. Un ambiente de prueba a veces es el mismo que el de desarrollo e integración y se hace para comparar tantos entornos de producción como sean posibles para la exactitud en las pruebas. (Leaf Group, 2018)

3.2 Docker

3.2.1 Definición

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará. (AWS Editorial Team, 2021)

3.2.2 Arquitectura Docker

La arquitectura Docker es una arquitectura cliente-servidor, dónde el cliente habla con el servidor (que es un proceso daemon) mediante un API para poder gestionar el ciclo de vida de los contenedores y así poder construir, ejecutar y distribuir los contenedores.

El hecho de que el cliente se comunique con el servidor mediante el API hace que el cliente y servidor puedan estar en la misma máquina comunicándose mediante sockets de UNIX o bien en máquinas diferentes comunicándose mediante un end-point en la red.

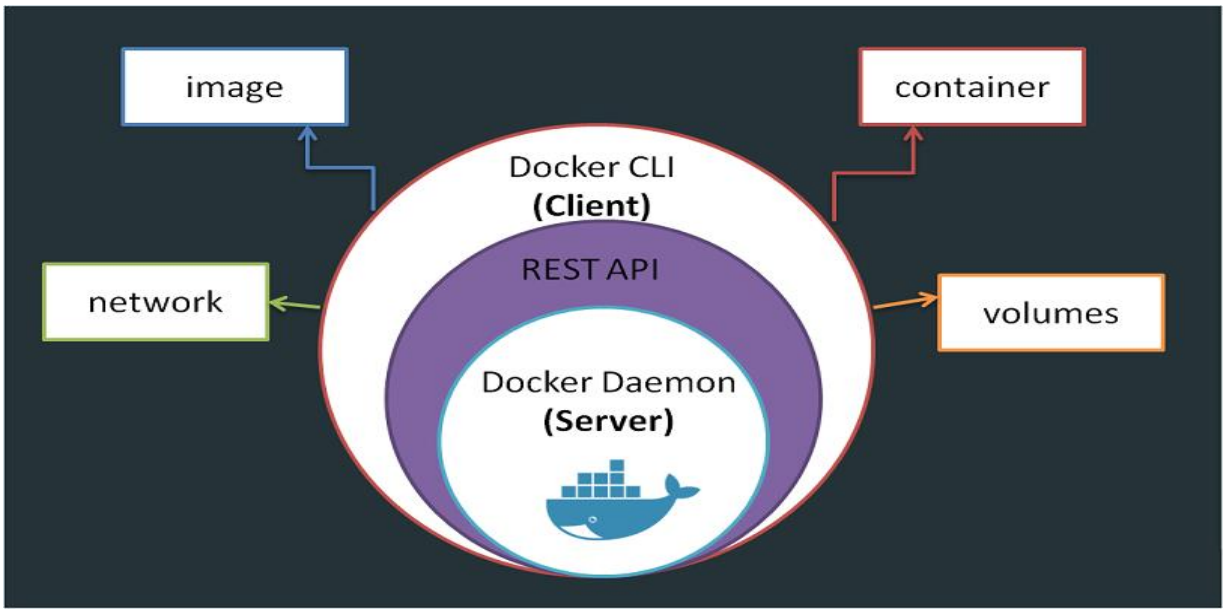


Ilustración 1: Arquitectura Docker

3.2.3 ¿Como funciona Docker?

Docker le proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual que elimina la necesidad de administrar directamente el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores. (AWS Editorial Team, 2021)

3.2.4 ¿Por qué usar Docker?

Docker le permite entregar código con mayor rapidez, estandarizar las operaciones de las aplicaciones, transferir el código con facilidad y ahorrar dinero al mejorar el uso de recursos. Con Docker, obtiene un solo objeto que se puede ejecutar de manera fiable en cualquier lugar. La sintaxis sencilla y simple de Docker le aporta un control absoluto. La amplia adopción significa que existe un gran ecosistema de herramientas y aplicaciones listas para su uso que puede utilizar con Docker. (AWS Editorial Team, 2021)

3.2.5 Ventajas de usar Docker.

Envíe software, más rápido: La frecuencia media de envío de software de los usuarios de Docker es siete veces superior a la de aquellos que no lo usan. Docker le permite entregar servicios aislados con la frecuencia necesaria. (AWS Editorial Team, 2021)

Estandarice las operaciones: Las aplicaciones con contenedores facilitan la implementación, la identificación de problemas y el retorno a una fase anterior para remediarlos. Transfiera de manera sencilla. (AWS Editorial Team, 2021)

3.2.6 Como usar Docker

Puede utilizar los contenedores de Docker como bloque de construcción principal a la hora de crear aplicaciones y plataformas modernas. Docker facilita la creación y la ejecución de arquitecturas de microservicios distribuidos, la implementación de código con canalizaciones de integración y entrega continúas estandarizadas, la creación de sistemas de procesamiento de datos altamente escalables y la creación de plataformas completamente administradas para sus desarrolladores. (AWS Editorial Team, 2021)

3.3 Imágenes

3.3.1 Definición

Una imagen en Docker es un archivo o file que se encuentra compuesto de diversas capas y que se utiliza con el objetivo de ejecutar un código dentro de un contenedor de Docker. Estas imágenes contienen todo el sistema de ficheros inicial en los que se va a basar el contenedor para su funcionamiento, así como su punto de entrada o entrypoint.

Este punto de entrada se refiere a la aplicación o comando que deberá ejecutarse una vez que el usuario lance un contenedor que esté asociado a esa imagen en Docker.

De manera que estas imágenes se encargan de actuar como un script o conjunto de instrucciones útiles para construir un contenedor en Docker, así como una plantilla. De la misma forma, una imagen en esta plataforma funciona como un punto de partida cuando el usuario utiliza Docker.

También debes tener en cuenta que la ejecución de una imagen dentro del sistema de Docker implica que esta tiene la posibilidad de convertirse en una o más instancias de un Docker container o contenedor de Docker.

Una imagen en Docker puede compararse en un sistema de máquinas virtuales con la plantilla o instantánea que utiliza en los entornos de la virtual machine. (KeepCoding, 2022)

3.3.2 Característica de una Imagen Docker

Dentro de los elementos característicos de una imagen en Docker se encuentra que incluye herramientas como los códigos de aplicación, las bibliotecas, dependencias, instrumentos y demás archivos que el sistema requiera para llevar a cabo la ejecución de una determinada aplicación. Además, cuenta con otra característica, como las que te detallamos a continuación:

3.3.3 Capas de una imagen Docker

En lo que respecta al funcionamiento de una imagen en Docker, se puede agregar que estas se usan con el objetivo de crear los contenedores, debido a que no pueden cambiar, lo que contribuye a la creación de contenedores con múltiples capas de imágenes que se superponen sobre las otras. De modo que cada una de estas capas son originadas en la capa inmediatamente anterior, pero son diferentes de esta.

Además, estas capas contribuyen a procesos como la aceleración de las compilaciones de Docker y el aumento de la reutilización, así como la disminución del uso del disco del sistema.

Otro aspecto importante es que las capas de imagen representan archivos con permisos de solo lectura, por lo que, cuando un contenedor es creado por el usuario, el sistema se encarga de agregar una capa de escritura encima de las imágenes de Docker que no pueden modificarse. Esto permite que el desarrollador pueda realizar los cambios que considere necesarios.

3.3.4 Casos de uso de una Imagen

Las imágenes de Docker tienen la posibilidad de llevar a cabo la ejecución de una aplicación en los contenedores del sistema, incluyendo las variables de entorno, los tiempos de ejecución y demás herramientas.

De manera que, cuando una imagen se implementa dentro de un entorno de Docker, puede ejecutarse como un contenedor de la plataforma. Esto se puede complementar usando el comando Docker run, que se encarga de crear contenedores partiendo de una imagen en Docker específica.

Además, estas imágenes pueden implementarse en cualquier host; es decir, los usuarios pueden tomar las capas de imágenes que sean estáticas dentro de un proyecto y usarlas en otro proyecto diferente, lo que permite ahorrar tiempo al evitar recrear la imagen desde el inicio. (KEEPCODING Tech School, s.f.)

3.3.5 Creación de Imágenes Docker

La creación de imágenes para Docker puede tener diversas razones, ya sea porque no puede encontrar un componente o un servicio que desea usar como parte de su aplicación en Docker. También es posible que la aplicación que se está desarrollando se tiene que acoplar para facilitar el envío y la implementación.

Para la creación de imágenes para Docker primero debemos comprender para qué estamos dockerizando o para qué aplicación estamos creando una imagen y cómo se crea la aplicación. Comience por pensar qué podría hacer si desea implementar la aplicación manualmente.

3.3.6 Creación del archivo para la imagen Docker

Ahora que se tienen las instrucciones, vamos a crear un archivo acoplable con estos pasos. Aquí hay una descripción general rápida del proceso de creación de su propia imagen:

Primero hay que crear un archivo Docker llamado “Dockerfile” y escribir las instrucciones para configurar la aplicación en él, como instalar dependencias, donde copiar el código fuente desde y hacia y cuál es el punto de entrada de la aplicación, etc. (Ingnacio Moreno Hojas, StatPlans, s.f.)

3.4 Dockerfile

3.4.1 Definición

Un Dockerfile es un archivo o documento de texto simple que incluye una serie de instrucciones que se necesitan ejecutar de manera consecutiva para cumplir con los procesos necesarios para la creación de una nueva imagen.

A este conjunto de instrucciones se le conoce como línea de comandos y serán los encargados de indicar los pasos a seguir para el ensamblaje de una imagen en Docker, es decir, los elementos necesarios para el desarrollo de un contenedor en Docker.

De manera que las imágenes en Dockerfile se crean a partir de un comando en específico denominado docker build, que se encargará de ofrecer las herramientas para que el sistema siga las instrucciones que el usuario haya indicado en la línea de comandos.

3.4.2 Característica de Dockerfile

Dentro de las características de un Dockerfile se encuentra su ejecución, que se lleva a cabo siguiendo ciertos pasos de fácil aplicación, iniciando con la indicación de las instrucciones que el usuario considere necesario para la creación de la

imagen. Después de esto, se debe ejecutar el comando docker build, que hará que el Dockefile esté disponible para su uso y crear contenedores. Esto implica que esta herramienta funciona bajo el esquema Dockerfile > Docker Image > Docker Container.

Además de esto, un Dockerfile se caracteriza por definir las instrucciones para crear una nueva imagen, pero siempre inicia con una imagen base que ya existe en el sistema. De manera que la imagen creada surge partir de esta base, pero cuenta con una serie de diferencias y propiedades distintas que distingue una de la otra.

3.4.3 Funcionamiento de Dockerfile

En lo que respecta al funcionamiento de un archivo de texto Dockerfile, es posible agregar que cada una de las instrucciones incluidas en esta herramienta para crear una imagen se ubica en una línea distinta y se ejecutan una tras otra, es decir, de forma consecutiva como si se tratara de un script por lotes.

De modo que la creación de una imagen de Docker depende de las instrucciones del Dockerfile. Estas, a su vez, se inician una vez se ejecute el comando docker build dentro de la plataforma de contenedores.

Para el funcionamiento de Dockerfile se debe tener en cuenta también el llamado contexto docker build context, es decir, la herramienta que se encarga de indicar cuáles son los archivos, directorios y ficheros a los que tiene acceso la opción de docker build. Por tanto, los datos e información pertenecientes al directorio fuente se transfieren al docker daemon una vez se ejecute el comando docker build. Esto significa que, para la serie de pasos incluidas en un Dockerfile, podrán tener acceso a la información relacionada con los archivos del build context.

En los casos donde el usuario no necesite utilizar la totalidad de archivos incluidos en el directorio fuente local en el build context, puede acudir a la herramienta de. dockerignore, que se encargará de excluir los elementos que no requiera la imagen final.

3.4.4 Opciones de Dockerfile

Dentro de las opciones más relevantes para un Dockerfile se encuentran herramientas encargadas de labores, como el establecimiento de la imagen base, el cambio de usuario o los elementos preestablecidos para el arranque de un contenedor en Docker, entre otros.

Algunas de estas opciones son:

- FROM: opción de Dockerfile que debe presentarse como la primera instrucción. Cumple con la función de establecer la imagen sobre la que los pasos e imágenes siguientes se desarrollan en el sistema.
- ENV: hace referencia a la opción que indica las variables de entorno que se necesitan en el proceso de construcción de una imagen en Docker y permite la ejecución de los contenedores y sus labores en el sistema.
- RUN: es una de las opciones de mayor importancia y popularidad en Dockerfile. Cumple la labor de ejecutar una instrucción incluida en la línea de comandos de la imagen durante su proceso de construcción. Dockerfile RUN puede escribirse en formato SHELL o bajo la opción de escritura EXEC.
- ADD: este elemento se encarga de las tareas relacionados con la copia de ficheros, directorios y archivos de una imagen en Dockerfile. Se debe tener en cuenta que el uso de la instrucción ADD implica la creación de una nueva capa de imagen, por lo que debes ser cuidadoso al implementar esta opción.
- EXPOSE: es la opción que tiene como labor la definición de las asignaciones referentes a los puertos para los contenedores de la plataforma que se encuentren en su etapa de ejecución. (KeepCoding, s.f.)

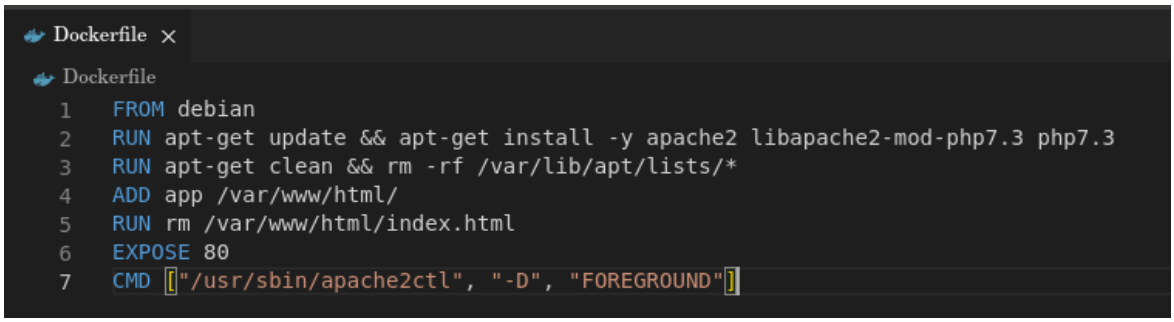


Ilustración 2: Ejemplo de Dockerfile para crear una imagen con una aplicación web desarrollada en PHP

3.5 Contenedores

3.5.1 Definición

Los contenedores son una forma de virtualización del sistema operativo. Un solo contenedor se puede usar para ejecutar cualquier cosa, desde un microservicio o un proceso de software a una aplicación de mayor tamaño, que se usa para agrupar una aplicación con todos sus archivos necesarios en un entorno de ejecución. Como una sola unidad, el contenedor puede moverse con facilidad y ejecutarse en cualquier sistema operativo en cualquier contexto. (NetApp, 2021)

Dentro de un contenedor se encuentran todos los ejecutables, el código binario, las bibliotecas y los archivos de configuración necesarios. Sin embargo, en comparación con los métodos de virtualización de máquinas o servidores, los contenedores no contienen imágenes del sistema operativo. Esto los hace más ligeros y portátiles, con una sobrecarga significativamente menor. En implementaciones de aplicaciones de mayor tamaño, se pueden poner en marcha varios contenedores como uno o varios clústeres de contenedores. Estos clústeres se pueden gestionar mediante un orquestador de contenedores, como Kubernetes.

3.5.2 Beneficios de los Contenedores.

Los contenedores son una forma optimizada de crear, probar, poner en marcha y volver a poner en marcha aplicaciones en varios entornos, desde un portátil local de

un desarrollador hasta un centro de datos on-premises e incluso en la nube. Algunos de los beneficios de los contenedores son:

- Menos sobrecarga: Los contenedores requieren menos recursos del sistema que los entornos de máquinas virtuales tradicionales o de hardware porque no incluyen imágenes del sistema operativo.
- Mayor portabilidad**:** Las aplicaciones que se ejecutan en contenedores se pueden poner en marcha fácilmente en sistemas operativos y plataformas de hardware diferentes.
- Funcionamiento más constante: Los equipos de DevOps saben que las aplicaciones en contenedores van a ejecutarse igual, independientemente de dónde se pongan en marcha.
- Mayor eficiencia: Los contenedores permiten poner en marcha, aplicar parches o escalar las aplicaciones con mayor rapidez.
- Mejor desarrollo de aplicaciones: Los contenedores respaldan los esfuerzos ágiles y de DevOps para acelerar los ciclos de desarrollo, prueba y producción. (NetApp, 2021)

Al usar contenedores, los usuarios evitan que se produzcan bloqueos debido a entornos incompatibles y obtienen un rendimiento uniforme en todos los equipos. Los desarrolladores pueden entonces enfocarse en la propia aplicación y no en la corrección de errores o en reescribirla para diferentes entornos de servidores. Y, sin sistema operativo, los contenedores ofrecen una manera eficiente de que los desarrolladores puedan desplegarlos en clúster, donde los contenedores individuales contienen componentes únicos de aplicaciones complejas. Al dividir los componentes en contenedores separados, los desarrolladores también pueden actualizar componentes individuales en lugar de reprocesar toda la aplicación. (AWS Editorial Team, 2021)

3.5.3 Casos de uso de contenedores

Algunas de las formas más habituales en las que las organizaciones usan los contenedores son:

- El rehospedaje de las aplicaciones existentes en arquitecturas de nube modernas: Algunas organizaciones utilizan contenedores para migrar las aplicaciones existentes a entornos más modernos. Aunque esta práctica ofrece algunos de los beneficios básicos de la virtualización de sistemas operativos, no ofrece todas las ventajas de una arquitectura de aplicaciones modular basada en contenedores. (AWS Editorial Team, 2021)
- Refactorización de las aplicaciones existentes para contenedores**:** Aunque la refactorización requiere mucho más que la migración del rehospedaje, ofrece todos los beneficios en un entorno de contenedores.
- Desarrollo de nuevas aplicaciones nativas del contenedor: Al igual que la refactorización, este método permite disfrutar de todos los beneficios de los contenedores.
- Más compatibilidad con las arquitecturas de microservicios**:** Las aplicaciones distribuidas y los microservicios se pueden aislar, poner en marcha y escalar más fácilmente utilizando elementos básicos de contenedores individuales.
- Soporte de DevOps para la integración y la puesta en marcha continuas (CI/CD): La tecnología de contenedores permite la creación, la prueba y la puesta en marcha optimizadas a partir de las mismas imágenes de contenedores.
- Una puesta en marcha más sencilla de tareas y trabajos repetitivos: Los contenedores se ponen en marcha para dar soporte a uno o varios procesos parecidos que, a menudo, se ejecutan en segundo plano, como las funciones ETL o los lotes de tareas en contenedores. (AWS Editorial Team, 2021)

3.6 Docker Compose

3.6.1 Definición

Es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de esta forma es más sencillo crear contendores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc. Nos permite lanzar un solo comando para crear e iniciar todos los servicios desde su configuración (YAML), esto significa que puedes crear diferentes contenedores y al mismo tiempo diferentes servicios en cada contenedor, integrarlos a un volumen común e iniciarlos y/o apagarlos, etc. Este es un componente fundamental para poder construir aplicaciones y microservicios.

Docker-Compose funciona en todos los entornos: production, staging, development, testing, así como flujos de trabajo basados en Continuous Integration(CI). (Barrios, 2020)

3.6.2 Características de Docker-Compose

Estas características pueden ser detalladas de la siguiente forma:

- Multiplicidad de entornos en un solo host: Esta característica de Docker Compose indica que la herramienta tiene la posibilidad de utilizar un nombre de un determinado proyecto, con el objetivo de aislar los entornos entre sí.
- Conservar los datos de volumen: permitir la preservación de la información.

- Recrear contendores que hayan sido modificados: propiedad de almacenar en el caché toda la información relacionada con los ajustes y configuraciones usados para crear un contenedor. En los casos donde un servicio que no ha cambiado se reinicia, Compose se encarga de reutilizar los containedores que ya existen.
- Variables y movimientos de una composición entre ambientes: admite variables de entorno en su archivo, que pueden usarse con el objetivo de

personalizar su composición para un gran número de entornos o usuarios diferentes. (KEEPCODING Tech School, s.f.)

3.6.3 Estructura de un fichero Docker-Compose.yml

- version (Obligatorio. Si no se indica una versión se trataría de la versión 1 que está en desuso)
- services (Obligatorio. Debe incluir al menos un servicio). Que nos servirá para configurar todos nuestros contenedores partiendo de una imagen base, además de poder indicar variables de entorno o establecer ubicaciones.
- volumes (Opcional). Donde podremos definir la persistencia de datos de nuestros contenedores.
- networks (Opcional). Para definir las redes con las que conectar los distintos servicios. (I.E.S CELIA VIÑAS CIBERSEGURIDAD, 2021)

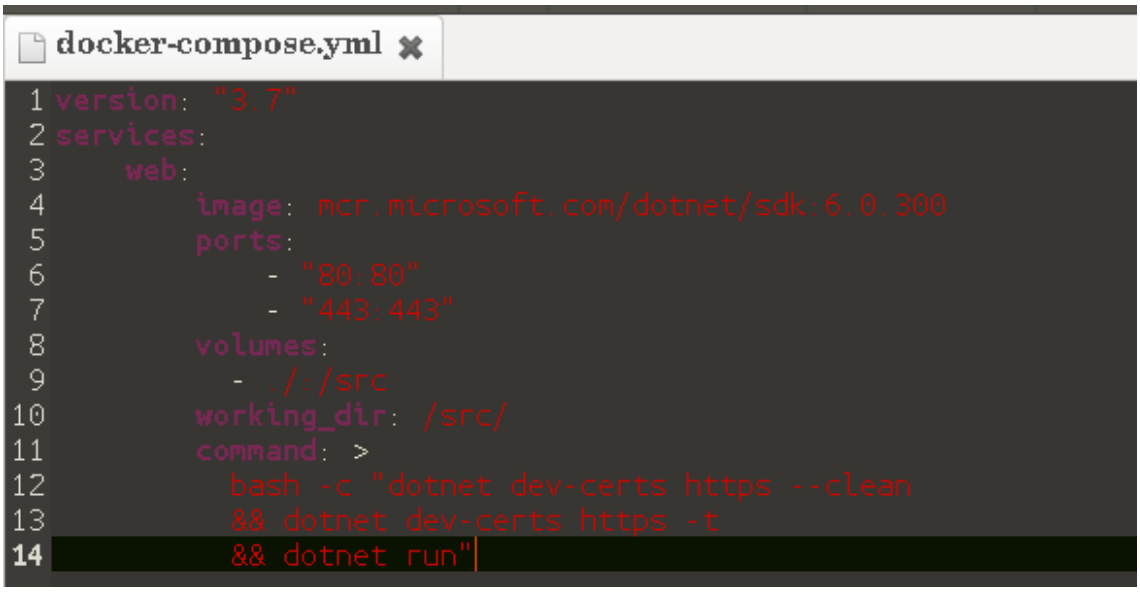


Ilustración 3: Fichero docker-compose.yml para un contenedor para una aplicación en .net

3.7 Docker hub

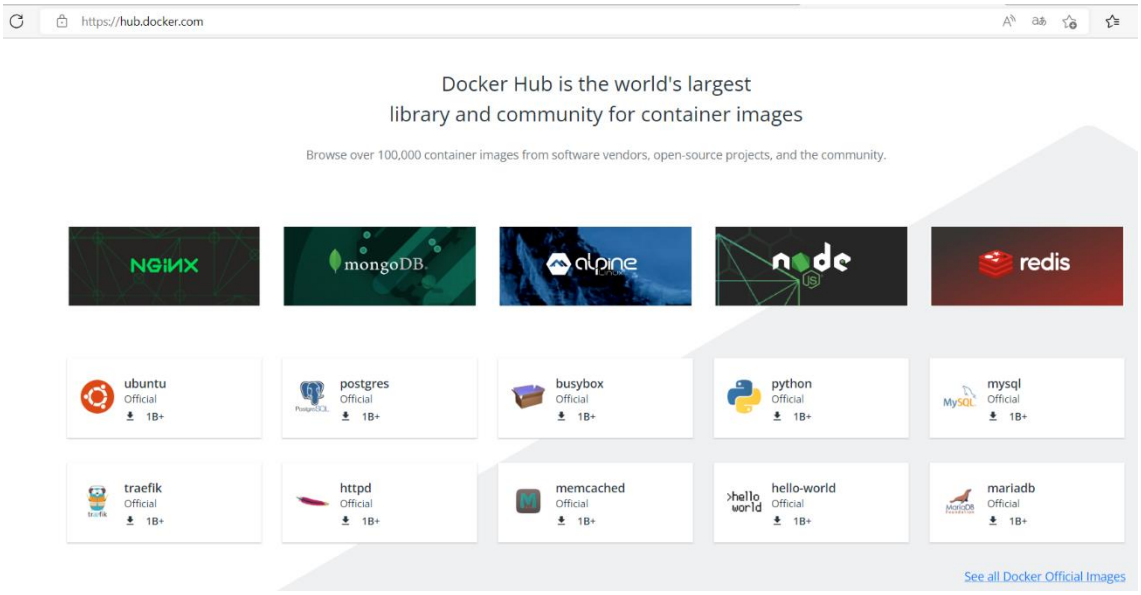


Ilustración 4: Página oficial de Docker hub

3.7.1 Definición

Es un servicio proporcionado por Docker para encontrar y compartir imágenes de contenedores en nuestros ordenadores. El mismo, presenta las siguientes características o elementos principales:

- Repositorios: Para la gestión (push and pull) de las imágenes de los contenedores.
- Equipos y organizaciones: Para la gestión del acceso a los depósitos privados de imágenes de contenedores.
- Imágenes oficiales: Son imágenes de alta calidad de los contenedores proporcionadas por Docker.
- Imágenes de editores: Son imágenes de alta calidad de contenedores proporcionadas por proveedores externos. Estás imágenes certificadas, también incluyen soporte y garantía de compatibilidad con Docker Enterprise.
- Creaciones: Imágenes de contenedores propias creadas desde GitHub y Bitbucket, que luego son cargadas a Docker Hub.

- Webhooks: Acciones programadas sobre un repositorio cargado para integrar Docker Hub con otros servicios. (Desde Linux, s.f.)

3.8 YAML

3.8.1 Definición

YAML es un lenguaje de serialización de datos que suele utilizarse en el diseño de archivos de configuración. Para algunas personas, YAML significa otro lenguaje de marcado más; para otras, es un acrónimo recursivo que quiere decir YAML no es un lenguaje de marcado, lo que enfatiza la idea de que se utiliza para los datos, no para los documentos.

Además, es un lenguaje de programación popular debido a que las personas pueden comprenderlo fácilmente y utilizarlo con otros lenguajes. Gracias a la flexibilidad y la accesibilidad que lo caracterizan, la herramienta de automatización de Ansible lo utiliza para crear procesos de automatización en forma de playbooks de Ansible.

3.8.2 Sintaxis de YAML

YAML cuenta con funciones de Perl, C, XML, HTML y otros lenguajes de programación, y se basa en JSON, lo cual genera que los archivos JSON sean compatibles con YAML.

En YAML, se usa la sangría al estilo Python para indicar la incorporación de un elemento de código dentro de otro. No se admiten los caracteres de tabulación, así que se usan los espacios en blanco, ni tampoco hay símbolos comunes de formato, como las llaves, los corchetes, las etiquetas de cierre o las comillas. Los archivos YAML usan las extensiones yml o yaml.

Al crear un archivo YAML, es necesario asegurarse de que sea válido y siga las reglas sintácticas.

3.8.3 Los usos de YAML

Uno de los usos más comunes es la creación de archivos de configuración. Se recomienda utilizar YAML en lugar de JSON para escribir los archivos de configuración porque es un lenguaje más fácil de comprender, aunque ambos pueden usarse de manera indistinta en la mayoría de los casos.

Además de Ansible, hay otros elementos que utilizan YAML, como las implementaciones y los recursos de Kubernetes. Una de las ventajas de utilizarlo es que se pueden agregar los archivos a un control de versiones, como Github, para rastrear y auditar los cambios

3.8.4 YAML para Kubernetes

El funcionamiento de Kubernetes depende del estado definido y del real. Los objetos de Kubernetes representan el estado de un clúster, y comunican a la plataforma cuál es la apariencia de la carga de trabajo que usted desea. Se pueden crear recursos de Kubernetes, como los pods, los objetivos y las implementaciones, con los archivos YAML.

Cuando cree un objeto de Kubernetes, para lo cual puede utilizar su propia API, deberá incluir especificaciones para definir su estado deseado. La solicitud a la API incluirá las especificaciones del objeto en JSON, pero lo más común es que usted proporcione la información obligatoria a kubectl como un archivo YAML. Kubectl convertirá el archivo al lenguaje YAML por usted cuando haga la solicitud a la API.

Una vez que se crea y se define el objeto, Kubernetes se encarga de garantizar su permanencia.

Los desarrolladores o los administradores de sistemas indican el estado definido a través de los archivos YAML o JSON que envían a la API de Kubernetes. La plataforma utiliza un controlador para analizar la diferencia entre el nuevo estado definido y el real en el clúster. (Red Hat,Inc, 2021)

3.9 Kubernetes

3.9.1 Definición

Kubernetes o k8s, es una plataforma open source que automatiza las operaciones de los contenedores de Linux. Elimina muchos de los procesos manuales involucrados en la implementación y escalabilidad de las aplicaciones en contenedores. En otras palabras, puede crear un clúster de grupos de hosts que ejecutan contenedores de Linux, y Kubernetes lo ayuda a administrar con facilidad y eficacia esos clústeres. Estos clústeres pueden abarcar hosts en nubes públicas, privadas o híbridas. Por este motivo, Kubernetes es la plataforma ideal para alojar aplicaciones nativas de la nube que requieren una expansión rápida, como la transmisión de datos en tiempo real a través de Apache Kafka.

3.9.2 Las posibilidades que ofrece Kubernets

La ventaja principal de usar Kubernetes en su entorno es que obtiene una plataforma para programar y ejecutar los contenedores en clústeres de máquinas virtuales (VM) o físicas, especialmente si se encuentra en pleno proceso de optimización del desarrollo de las aplicaciones para la nube.

A grandes rasgos, le permite implementar una infraestructura basada en contenedores en los entornos de producción, y depender completamente de ella. Además, puede realizar muchas de las mismas tareas que haría con otras plataformas de aplicaciones o sistemas de gestión, pero en sus contenedores, ya que Kubernetes automatiza las tareas operativas.

Los desarrolladores pueden crear aplicaciones en la nube con Kubernetes como una plataforma de tiempo de ejecución utilizando los patrones, que son las herramientas necesarias para diseñar aplicaciones y servicios basados en los contenedores. (Red Hat,Inc, 2020)

3.9.3 Que son los contenedores en Kubernetes

A medida que los desarrolladores implementan cada vez más software para un conjunto diverso de entornos informáticos con diferentes nubes, entornos de prueba, portátiles, dispositivos, sistemas operativos y plataformas, resulta más importante que el software se ejecute de manera confiable. Aquí es donde entran en juego los contenedores: agrupan una aplicación con todo su entorno de tiempo de ejecución. En este sentido, los contenedores se consideran una forma de virtualización, ya que proporcionan una "burbuja" en la que se puede ejecutar la aplicación con las bibliotecas, dependencias y sistemas operativos correctos. No obstante, los contenedores son más pequeños que las máquinas virtuales, ya que contienen solo los recursos que necesita la aplicación, y nada más. (OCL, s.f.)

3.9.4 Comparación entre Kubernetes y Docker

Docker es una plataforma de contenedorización y un tiempo de ejecución de contenedores, mientras que Kubernetes es una plataforma para ejecutar y gestionar contenedores a partir de numerosos tiempos de ejecución de contenedores. Kubernetes admite varios tiempos de ejecución de contenedores, incluido Docker.

La irrupción de Docker en el año 2013 supuso el comienzo de la era moderna de los contenedores y el inicio de un modelo informático basado en microservicios. Dado que los contenedores no tienen un sistema operativo propio, facilitan el desarrollo de microservicios escalables y poco vinculados, ya que permiten que los equipos empaqueten aplicaciones de forma declarativa, junto con sus dependencias y su configuración, como imágenes de contenedor.

Sin embargo, a medida que las aplicaciones fueron ganando en complejidad para incluir contenedores distribuidos en distintos servidores, empezaron a plantearse problemas; por ejemplo, cómo coordinar y programar varios contenedores, cómo permitir la comunicación entre contenedores o cómo escalar instancias de contenedor. Kubernetes se creó para resolver este tipo de problemas.

En la tecnología de contenedores, Docker y Kubernetes son dos herramientas de referencia. Tal vez te preguntes cuál de ellas es mejor. A menudo, la clave no está en elegir una o la otra, sino en saber cómo pueden utilizarse conjuntamente para obtener resultados óptimos. (CHANDLER HARRIS, s.f.)

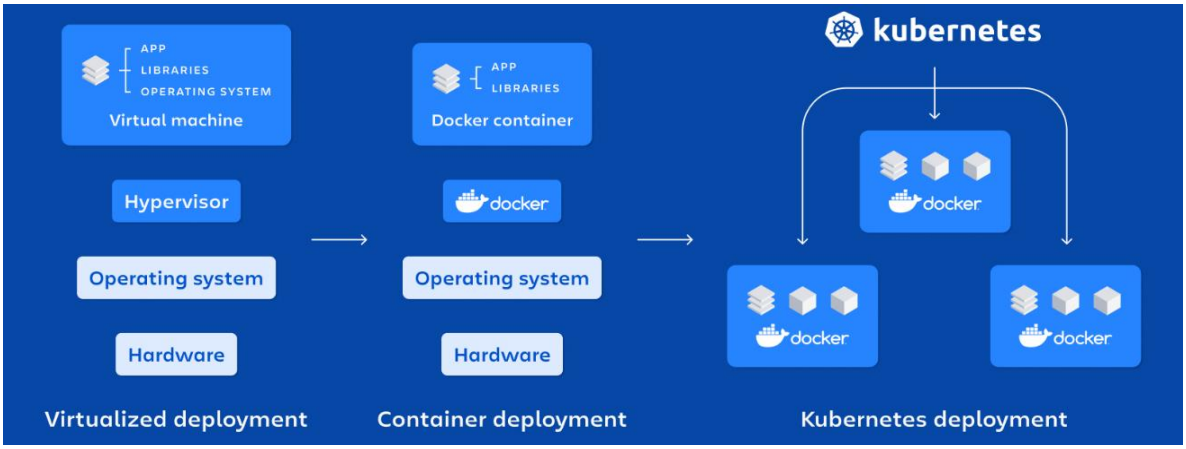
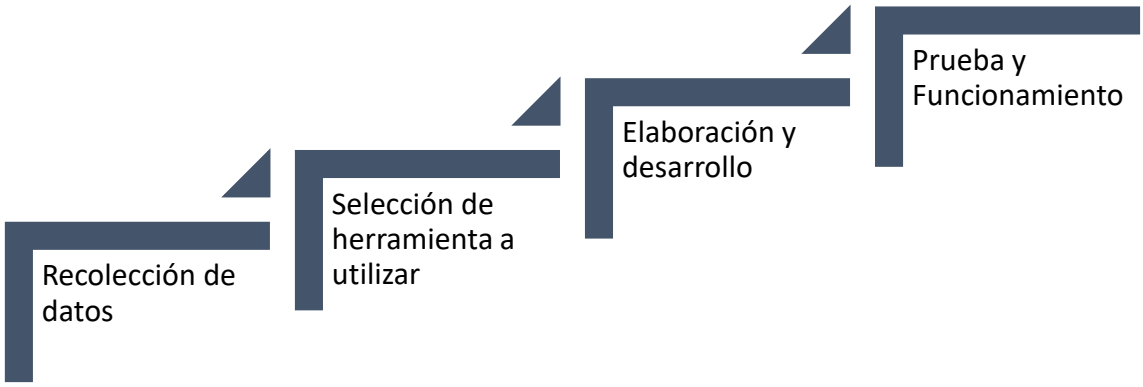


Ilustración 5: Docker vs Kubernetes

4 Diseño Metodológico

4.1 Tipo de estudio

El presente estudio investigativo tiene un enfoque aplicativo con la finalidad de construir ambientes de desarrollo eficientes para el despliegue de aplicaciones.



4.2 Etapa I: Recolección de datos.

En la primera etapa del trabajo se recolecta información necesaria e indispensable para poder conocer las necesidades de los estudiantes, Se tomo en cuenta la información ya existente sobre el problema de investigación.

4.3 Etapa II: Selección de herramientas a utilizar.

4.3.1 Recursos hardware

Material	Descripción
Laptops	Se utilizaron para el desarrollo de esta investigación las máquinas de marca Toshiba, DELL, ASUS, cada una con distintas características. La memoria RAM consta de 4 - 12 GB y el procesador 2.4 GHz correlativamente.

4.3.2 Recursos software.

Software	Descripción
Ubuntu MATE 20.04	Es una distribución Linux basada en Ubuntu que usa el entorno de escritorio MATE.
Ubuntu 20.04 y 22.04	Sistema operativo GNU/Linux de código abierto basado en Debian. Actualmente se ejecuta en operadores de escritorio y servidores.
Firefox y Chrome	Navegadores web, diseñados para navegar por internet de manera rápida.
Docker versión nativa Linux	Es un proyecto de código abierto que automatiza la implementación de aplicaciones dentro de los contenedores permitiendo empaquetar una aplicación con sus dependencias en un contenedor.

Docker Compose	Es una herramienta que se desarrolló para ayudar a definir y compartir aplicaciones de varios contenedores.
Visual Studio Code	Es un editor de código optimizado con soporte para operaciones de desarrollo como depuración, ejecución de tareas y control de versiones.

4.4 Etapa III: Elaboración y desarrollo.

En esta etapa se elaboraron guías para familiarización de la tecnología Docker, donde de igual forma se elaboraron los ficheros yaml para la ejecución de ambientes aislado para los lenguajes seleccionados.

4.5 Etapa IV: Prueba y funcionamiento.

En esta etapa del proyecto hacemos las últimas pruebas antes de presentar la investigación final.

5 Resultados

En este acápite se estará desarrollando la parte práctica de esta investigación donde se elaboraron guías en un orden que facilite la compresión de Docker y su facilidad para la creación de contenedores o ambientes para desarrollo, iniciando desde comandos básicos, ejemplos sencillos y guiados como también ejercicios propuestos para motivar a iniciarse solos en esta tecnología, de igual forma se explicara detalladamente como crear, configurar y utilizar los entornos de desarrollos para los lenguajes seleccionados usando Docker-compose.

Programación

- Guía 1: Familiarización con Docker.
- Guía 2: Imágenes con Docker.
- Guía 3: Utilización de volúmenes para compartir código fuente en Docker.
- Guía 4: Creación y comunicación entre múltiples contenedores.
- Guía 5: Docker-compose para el manejo de contenedores

Formato de las guías

- **Título de la guía**
- **Introducción**: Describe lo que se va realizar en la práctica además de servir de unión de la práctica anterior.
- **Duración estimada en la realización de la guía.**
  Tiempo estimado para dar solución a la guía.
- **Desarrollo:** Procedimientos, funcionalidad comandos a implementar capturas de pantallas y soluciones a errores para llevar a cabo la práctica.
- **Ejercicios:** Propuestos: ejercicios prácticos para poner en práctica lo antes realizado.

5.1 Guía 1: Familiarización con Docker

En la presente guía se abordan los principales comandos de Docker para verificar versiones disponibles en el equipo y manejo de los contenedores.

Duración estimada en la realización de la guía

45 minutos.

Desarrollo

Comandos básicos de Docker.

Mostrar la versión de Docker.

```
sudo docker –versión
```

Mostrar la ayuda de Docker.

```
sudo docker –help
```

CONTENEDORES.

Los contenedores son una forma de virtualización del sistema operativo, un solo contenedor se puede usar para ejecutar cualquier cosa, desde un microservicio o un proceso de software a una aplicación de mayor tamaño. Dentro de un contenedor se encuentran todos los ejecutables, el código binario, las bibliotecas y los archivos de configuración necesarios.

Listar todos los contenedores.

```
sudo docker ps -a
```

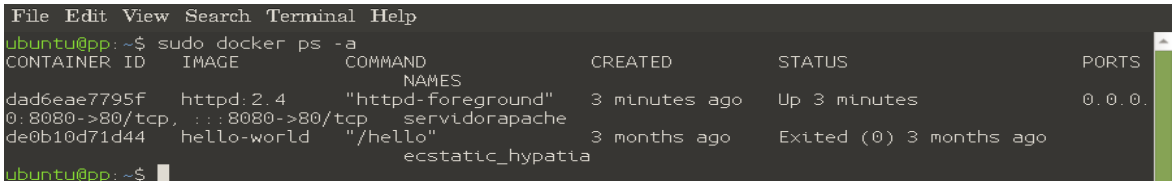


Ilustración 6: Visualización de contenedores existentes

Ejecutar un contenedor ya creado.

```
sudo docker start < Nombre o id del contenedor>
```

Listar contenedores en ejecución.

```
sudo docker ps
```

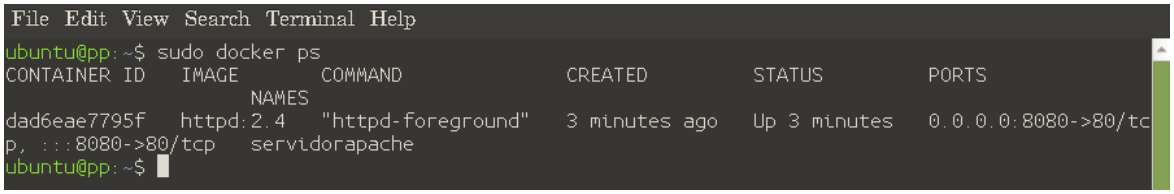


Ilustración 7:Contenedores en ejecución

Detener un contenedor.

```
sudo docker stop <Nombre de contenedor o id>
```

Reiniciar un contenedor.

```
sudo docker restart <Nombre o id del contenedor>
```

Eliminar un contenedor.

```
sudo docker rm < Nombre o id del contenedor >
```

Acceder a un contenedor**.**

```
$ sudo docker exec -it <Nombre o id d del contenedor> /bin/bash
```

Crear un contenedor con todas sus configuraciones para poner en prácticas los comandos anteriores.

Un contenedor se crea a partir de una imagen, en este post solo descargaremos la imagen, más adelante profundizaremos que son imágenes.

Descargar la imagen del servidor apache, esta es la imagen oficial de apache (httpd). En este caso la imagen httpd:2.4

```
sudo docker pull httpd:2.4
```

Crea el contenedor.

```
sudo docker run -p 8080:80 --name servidorapache -d httpd:2.4
```

-p: Permite mapear los puertos entre nuestra máquina local y el contenedor. En este caso, mapea el puerto 8080 de la máquina local con el puerto 80 del contenedor.

--name: Asigna un nombre al contenedor, Si no le asignamos, Docker nos asignará un nombre automáticamente.

-d: Ejecuta el contenedor en segundo plano.

httpd:2.2: Nombre de la imagen que a usar en el contenedor.

```
File  Edit  View  Search  Terminal  Help
ubuntu@pp:~$ sudo docker run -p 8080:80 --name servidorapache -d httpd:2.4
dad6eae7795f5aae3b7cf19352104322d239a3eee4bd3f070ce9b74ec1f177b3
ubuntu@pp:~$
```

Ilustración 8: Creando un contenedor Docker

Asegurarse de que el contenedor está configurado para ejecutar los scripts PHP, ve al navegador escribe http://localhost:8080

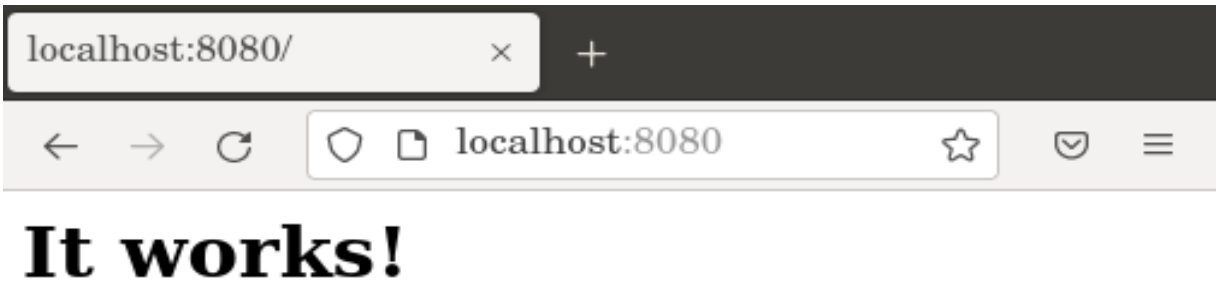


Ilustración 9: Salida de la aplicación servidorapache

Accede al contenedor y modifica el archivo index.html de apache esto solo por cuestión de practica ya que en un entorno de desarrollo no es algo practico.

Accede al contendor para modificar el archivo index.php del servidor apache.

```
sudo docker exec -it servidorapache /bin/bash
```

```
File  Edit  View  Search  Terminal  Help
ubuntu@pp:~$ sudo docker exec -it servidorapache /bin/bash
root@dad6eae7795f:/usr/local/apache2#
```

Ilustración 10: Accediendo al contenedor servidorapache

Accede al directorio htdocs de apache2

```
cd htdocs
```

Edita el archivo index.html, antes debe instalar nano (editor de texto) en el contenedor.

```
sudo apt update && apt install nano
```

```
nano index.html
```

```
<html><body><h1> Lo hiciste muy bien todo funciona! </h1></body></html>
```

Recarga la página en el navegador se notan el pequeño cambio realizado.

Ilustración 11: Edición de la salida de aplicación servidorapache

Ya sabes que un contenedor es creado a partir de una imagen base a la que se puede modificar para adaptarla nuestras necesidades, pero aún falta mucho por recorrer, como crear volúmenes para que los datos persistan, crear nuestras propias imágenes. Pero eso lo veremos más adelante.

Ejercicio propuesto.

Crear un contenedor con la imagen php:7.4-apache de la página oficial de Docker https://hub.docker.com

Usar el puerto 8000 para nuestra maquina local y el 80 para el contenedor. Acceder al contenedor y crear o modificar el archivo php que muestre la información sobre el PHP (phpinfo) que tengamos en nuestro contenedor.

5.2 Guía 2: Imágenes con Docker

En esta guía se abordará sobre imágenes Docker siendo el objetivo de esta conocer los comandos para trabajar con imágenes ya que estas son plantillas de solo lectura, donde una imagen puede contener el sistema de archivo de un sistema operativo, y tienen todo lo necesario para construir contenedores

Duración estimada en la realización de la guía

45 minutos.

Desarrollo

Principales instrucciones para trabajar con imágenes:

Mostrar las imágenes en el registro local.

```
sudo docker images
```

Descargar la última versión de la imagen indicada.

```
sudo docker pull <Nombre de la imagen>
```

Eliminar imágenes.

```
sudo docker rmi <Nombre o Id de la imagen>
```

Nota: No puede eliminar una imagen si tiene algún contenedor creado a partir de ella.

El objetivo principal de esta práctica es la creación de una imagen personalizada que contenga una aplicación, ya que las imágenes se utilizan para crear contenedores, y nunca cambian.

Para crear la imagen partimos de una imagen padre a la que le iremos añadiendo cosas o capas. Elegir una imagen base confiable, usar una con el sistema operativo Debian a partir de ella ir agregando capas lo que sería como agregarle una nueva herramienta a un sistema operativo.

Las configuraciones necesarias para crear la imagen se detallan en un archivo llamado Dockerfile este es un fichero o archivo de texto donde está el conjunto de instrucciones que serán ejecutadas de forma secuencial para construir una nueva imagen Docker. Cada una de estas instrucciones crea una nueva capa en la imagen que estamos creando.

Hay varias instrucciones que se pueden usar en la construcción de un Dockerfile, veamos las principales instrucciones que se pueden usar:

FROM: Sirve para especificar la imagen sobre la que voy a construir la mía.
Si la imagen ya está descargada en nuestro equipo solo la obtiene si no es así procederá a descargarla del sitio oficial de Docker.
RUN: Ejecuta una orden creando una nueva capa. En este caso es muy importante que pongamos la opción -y porque en el proceso de construcción no puede haber interacción con el usuario.
ADD: Es similar a COPY, pero tiene funcionalidades adicionales como especificar URLs y tratar archivos comprimidos.
COPY: Para copiar ficheros desde mi equipo a la imagen. Esos ficheros deben estar en el mismo contexto (carpeta o repositorio) donde está el Dockerfile.
WORKDIR: Establece el directorio de trabajo dentro de la imagen que estoy creando donde posteriormente se usaran las órdenes RUN, COPY, ADD, CMD o ENTRYPOINT.
EXPOSE: Nos da información acerca de qué puertos tendrá abiertos el contenedor cuando se cree uno en base a la imagen que estamos creando. Es meramente informativo.

Ya estudiamos el mecanismo de creación de imágenes Docker con los ficheros Dockerfile vámonos a la práctica.

Creación una imagen con el servidor web Apache2 y una simple app de HTML.

Crear un directorio el que será el área de trabajo, crea un archivo Dockerfile y en un directorio llamado app donde agregas un script index.html

Index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Images-docker</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
  <p>Listo ya tenemos una imagen con una app web! </p>
</body>
</html>
```

Dockerfile

```
FROM debian
RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm -rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2

ADD app /var/www/html/
EXPOSE 80
CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Instala el servidor web apache2 y para reducir el tamaño, borramos la caché de paquetes apt y la lista de paquetes descargada, creamos varias variables de entorno (en este ejemplo no se van a utilizar, pero se podrían utilizar en cualquier fichero del contexto, por ejemplo para configurar el servidor web), exponemos el puerto 80, copia el directorio app que contiene el fichero index.html al DocumentRoot (directorio por omisión de apache) y finalmente indica el comando que se va a ejecutar al crear el contenedor y además al usar el comando ENTRYPOINT, no permite ejecutar ningún otro comando durante la creación.

Generar la imagen.

```
sudo docker build -t mi-version-apache .
```

Con el comando **build**, la opción **-t** indica el nombre de la nueva imagen, con el (**.**) manda todos los ficheros del contexto

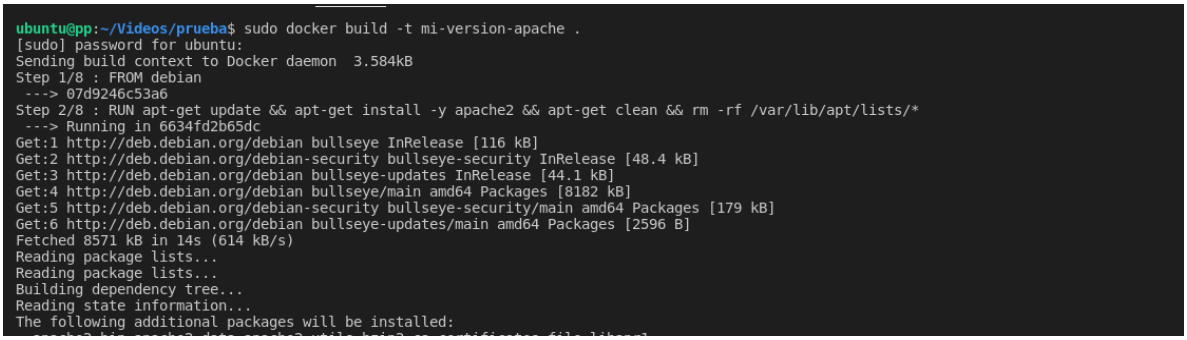


Ilustración 12: Creación de la imagen mi-version-apache

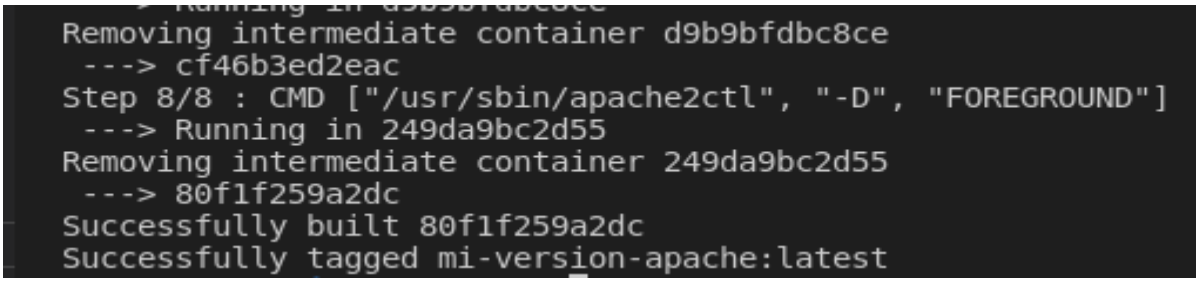


Ilustración 13: Salida correcta al crear la imagen mi-version-apache

Comprobar que esta generada la nueva imagen.

```
sudo docker images
```

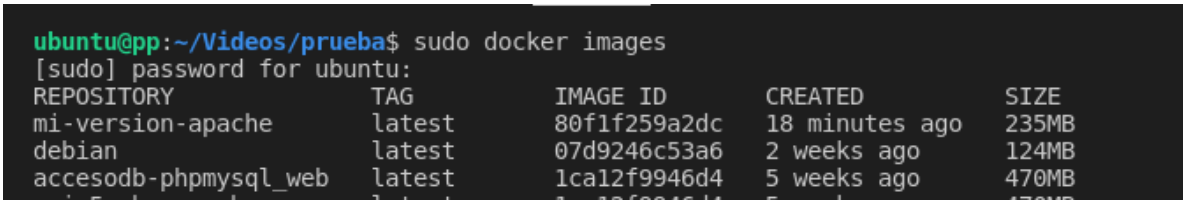


Ilustración 14: Lista de imágenes

A continuación, crear un contenedor a partir de la nueva imagen:

```
sudo docker run -p 80:80 --name servidor_web mi-version-apache
```

Acceder al navegador para ver la página web:

http://localhost/:80

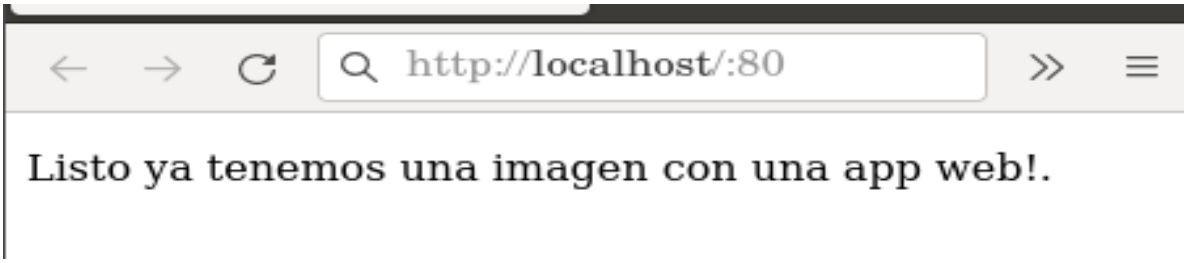


Ilustración 15: salida de la ejecución de la imagen mi-version-apache

Ya has creado la imagen con una aplicación ya incluida, a partir de una imagen base.

Creación de una imagen de Docker para una base de datos MySQL personalizada

También puedes crear una imagen que contenga una base de datos, lo que nos facilitaría probar código y escribir datos sin instalar y configurar muchas herramientas, en nuestra máquina.

Crea los directorios

```
mkdir -p /mi-mysql/scripts
cd mi-mysql/scripts/
```

Creando scripts SQL

Crea un directorio de trabajo. Además, cree un subdirectorio donde almacenara los scripts.sql

Personalizar la base de datos con una tabla llamada empleados. La tabla debe contener unas filas con algunos empleados (nombre, apellido, departamento y correo electrónico).

CreateTable.sql. Crea una tabla llamada empleados donde se agregará cuatro columnas a la tabla.

```
CREATE TABLE empleados (
nombre varchar(25),
apellido varchar(25),
departamento varchar(15),
email varchar(50) );
```

InsertData.sql. contiene la declaración para insertar datos en la tabla 'empleados'

```
INSERT INTO empleados (nombre, apellido, departamento, email)
VALUES ('Lorenzo', 'Vanthillo', 'IT', 'lvthillo@mail.com'),
      ('Luis', 'Gomes', 'Informatica', 'lgomes@mail.com'),
      ('Maria', 'Perez', 'RRHH', 'mperez@mail.com');
```

Ahora que los scripts están listos, escribir un Dockerfile para crear la imagen de Docker (basada en la imagen oficial de MySQL)

Dockerfile

```
# Imagen oficial MySQL (Imagen base)
FROM mysql
```

```
# Nombre de la base de datos
ENV MYSQL_DATABASE company

# Agregamos el contenido del directorio sql-scripts/ a la imagen
# Todos los scripts en docker-entrypoint-initdb.d/ son automáticamente
# ejecutados durante el inicio del contenedor

COPY ./scripts/ /docker-entrypoint-initdb.d/
#Exponemos el puerto
EXPOSE 3306
```

Crea la imagen

```
sudo docker build -t my-mysql .
```

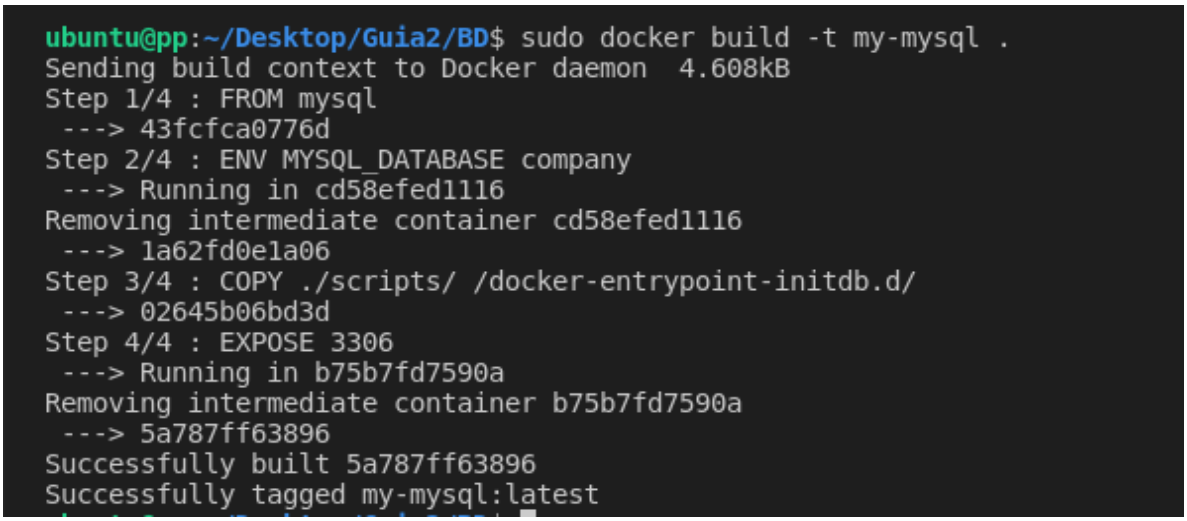


Ilustración 16: Creando la imagen my-mysql

Iniciar un contenedor MySQL con la nueva imagen.

```
sudo docker run -d -p 3306:3306 --name my-mysql -e
MYSQL_ROOT_PASSWORD=root my-mysql
```

Verifica si está ejecutándose el contenedor

```
sudo docker ps
```

```
ubuntu@pp:~/Desktop/Guia2/BD$ sudo docker run -d -p 3306:3306 --name my-mysql -e MYSQL_ROOT_PASSWORD=root my-mysql
6e6d946bf8c548702e4f24e8e88bbbdf9ed3a268c3b2c35d84ce235e3c5f0cc2
ubuntu@pp:~/Desktop/Guia2/BD$ sudo docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS
 NAMES
6e6d946bf8c5   my-mysql   "docker-entrypoint.s…"   54 seconds ago   Up 53 seconds   0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp
 my-mysql
```

Ilustración 17: Creando y ejecutándose el contenedor my-mysql

Verificar el contenido de la base de datos. Hacerlo con exec en el interior del contenedor.

```
sudo docker exec -it my-mysql /bin/bash
```

Accede a la consola mysql

La contraseña para acceder será la agregada al crear el contenedor con la variable de entorno (MYSQL_ROOT_PASSWORD)

```
mysql -u root -p
```

Verifica si existe la base de datos

```
show databases;
use company
```

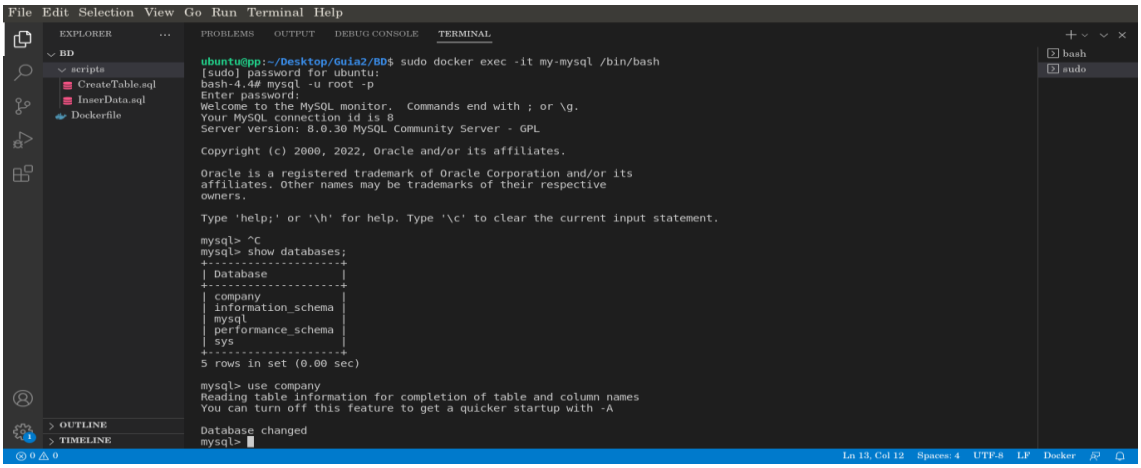


Ilustración 18: Consola de MySQL mostrando la base de datos company

Verifica si están los datos que se agregaron en el archivo InsertData.sql

```
select * from empleados;
```

Ilustración 19: Contenido de la tabla empleados de la base de datos company

¡Funciona! ¡Tenemos la imagen de Docker de base de datos MySQL personalizada! Esta es una gran solución para el desarrollo local entre múltiples desarrolladores. Al compartir la imagen de Docker, todos los desarrolladores pueden usar la base de datos simplemente iniciando un contenedor desde la imagen.

Sin embargo, es importante tener en cuenta que esta no siempre es la mejor solución ya que si inserta una gran cantidad de datos, el tamaño de su imagen aumentará significativamente lo que necesitará crear una nueva imagen cuando desee actualizar los datos.

En conclusión, las imágenes Docker se basan en un sistema de archivos en capas que ofrece muchas ventajas y beneficios para los casos de uso para los que son diseñados los contenedores, como ser ligeros y compartir partes comunes que muchos contenedores pueden desplegar y ejecutar en la misma máquina económicamente.

5.3 Guía 3: Utilización de volúmenes para compartir código fuente en Docker

En esta guía se abordarán los comandos para el manejo y funcionamiento de los volúmenes, los que están relacionados con almacenamiento y persistencias de los datos de los contenedores, como funcionan y como implementarlos.

Duración estimada en la realización de la guía

60 minutos.

Desarrollo

Volúmenes

Los volúmenes son el mecanismo preferido para conservar los datos generados y utilizados por los contenedores de Docker. Si bien los montajes de enlace dependen de la estructura del directorio y el sistema operativo de la máquina host, Docker administra completamente los volúmenes. Puede administrar volúmenes mediante los comandos de la CLI de Docker o la API de Docker.

Para poder guardar y compartir datos entre contenedores, Docker ideó el concepto de volúmenes. En pocas palabras, los volúmenes son directorios (o archivos) que están fuera del sistema de archivos de unión predeterminado y existen como directorios y archivos normales en el sistema de archivos del host.

Los volúmenes suponen el modo en el que Docker permite persistir los datos de tu aplicación. La aloja fuera del propio contenedor, en el propio sistema de archivos del host donde está corriendo Docker, de tal manera que se puede cambiar, apagar o borrar el contenedor sin que afecte a los datos**.**

Ventajas de usar volúmenes

Los volúmenes son bastante útiles porque permiten compartirse entre contenedores, o el propio host. Eso nos permite cosas como:

- Montar el código fuente de una aplicación web, dentro de un volumen, accesible desde el contenedor web y así ver en tiempo real los cambios durante el desarrollo.
- Consultar todos los logs cómodamente desde un contenedor dedicado.
- Hacer backups de un contenedor desde otro dedicado, o recuperar esos mismo backups hacia nuestro host.
- Compartir la misma información entre varios contenedores sin duplicarla. Por ejemplo, la información relativa al entorno: desarrollo, integración, preproducción, producción.

En Docker se usan un grupo de comandos disponibles para interactuar con volúmenes, algunos de estos son:

Crear un volumen

```
sudo docker volume create <Nombre del volumen>
```

Listar volúmenes

```
sudo docker volume ls
```

Inspeccionar volúmenes

```
sudo docker volume inspect <Nombre del volumen>
```

```
File  Edit  View  Search  Terminal  Help
ubuntu@pp:~$ sudo docker volume inspect vol1
[
    {
        "CreatedAt": "2022-10-25T00:15:45-06:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
        "Name": "vol1",
        "Options": {},
        "Scope": "local"
    }
]
ubuntu@pp:~$
```

Ilustración 20: Información sobre un volumen

Si inspecciona un volumen veras la información relacionada con este, donde está localizado en nuestro sistema, su nombre y la fecha de creación.

Borrar volúmenes

```
sudo docker volume rm <Nombre del volumen>
```

No podemos borrar un volumen que esté en uso por un contenedor, por lo que es necesario detener y borrar primero su contenedor.

Limpiar volúmenes, si se tiene volúmenes que no están asociados a ningún contenedor podemos deshacernos de todos ellos con un único comando.

```
sudo docker volume prune
```

Tipos de Volúmenes

Volúmenes de host

Estos volúmenes se crean en el docker host, se mapeará un directorio del docker host al directorio donde la aplicación guarda los datos en el contenedor.
De este modo, cualquier información que la aplicación guarde en el directorio del contenedor también se guardara en el docker host.

Este volumen no se borra con el comando:

```
sudo docker rm -v <Nombre del contenedor>
```

Volúmenes de datos sin nombres

Este tipo de volumen es creado y manejado por docker, la información se guarda en un directorio que crea docker en el docker root dir con un nombre aleatorio.

Se crean cuando se levanta un contenedor y no se le asigna un nombre concreto, mediante el comando docker run, por ejemplo:

```
sudo docker run -v /data alpine:3.4
```

-v: crea el volumen en la ruta /data.

Esto nos crea un volumen asociado al contenedor creado.

```
File  Edit  View  Search  Terminal  Help
ubuntu@pp:~$ sudo docker volume ls
DRIVER    VOLUME NAME
local     6ad81129e2d50e81772fdfc683c771b2ccb95d11a871f6d2162af03f790b4f0e
ubuntu@pp:~$
```

Ilustración 21: Nombre de un volumen

Volúmenes de datos con nombres

Estos volúmenes no dependen de ningún contenedor concreto, y se pueden montar en cualquier contenedor. Se crean específicamente usando el comando docker volume create, o al ejecutar un contenedor si le damos un nombre en la línea de comandos.

Los volúmenes con nombre son excelentes si simplemente queremos almacenar datos, ya que no tenemos que preocuparnos en dónde se almacenan dichos datos. Al igual que los volúmenes de host este tipo de volumen no se borra con el comando:

```
sudo docker rm -v <Nombre del contenedor>
```

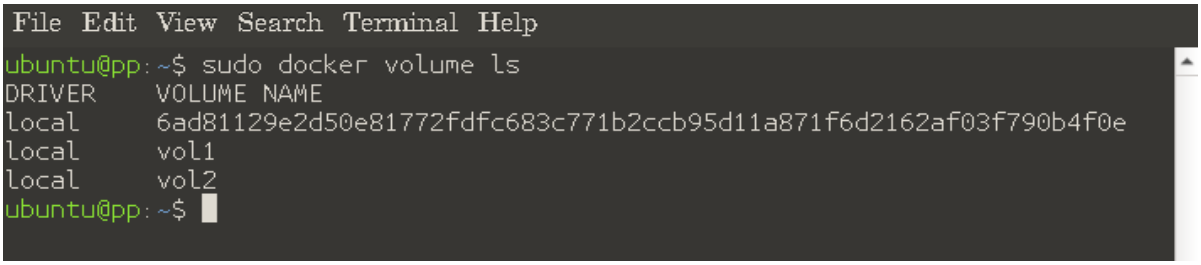
Crea un nuevo volumen

```
sudo docker volume create --name vol1
```

```
File  Edit  View  Search  Terminal  Help
ubuntu@pp:~$ sudo docker volume create --name vol1
vol1
ubuntu@pp:~$
```

Ilustración 22: Creando un volumen

```
sudo docker run -v vol2:/data alpine:3.4
```



Bind mounts

Otras veces interesa montar ficheros o carpetas desde la máquina *host*. En este caso, se puede montar la carpeta o el fichero especificando la ruta completa desde la máquina *host*, y la ruta completa en el contenedor. Es posible también especificar si el volumen es de lectura y escritura (por defecto) o de solo lectura.

Este caso es ideal para recuperar backups o ficheros generados en un contendor, en vistas a su utilización futura por parte de otros contenedores o del mismo host. Para usar montajes en un contenedor, tiene dos opciones de sintaxis para usar, --mount y -v. La diferencia más notable entre las dos opciones es que --mount es más detallado y explícito, mientras que la sintaxis -v es más una forma abreviada de la sintaxis --mount. Combina todas las opciones que pasa a --mount en un campo.

- v o - -volume: consta de tres campos, separados por dos puntos (:). Los campos deben estar en el orden correcto y el significado de cada campo no es inmediatamente obvio.

En el caso de volúmenes con nombre, el primer campo es el nombre del volumen y es único en una máquina host determinada. Para volúmenes anónimos, se omite el primer campo.

El segundo campo es la ruta donde se monta el archivo o directorio en el contenedor. El tercer campo es opcional y es una lista de opciones separadas por comas,

La sintaxis --mount es más detallada que -v o --volume, pero el orden de las teclas no es significativo y el valor de la bandera es más fácil de entender.

El tipo de montaje, que puede ser bind, volumen o tmpfs. Este tema trata sobre los volúmenes, por lo que el tipo siempre es volumen.

El origen del montaje, ara volúmenes con nombre, este es el nombre del volumen. Para volúmenes anónimos, este campo se omite. Puede especificarse como source o src.

El destino toma como valor la ruta donde está montado el archivo o directorio en el contenedor. Puede especificarse con la etiqueta destination(dst) o,target.

La opción de solo lectura, si está presente, hace que el montaje de vinculación se monte en el contenedor como de solo lectura. Puede especificarse como de solo lectura “ro”.

Si inicia un contenedor con un volumen que aún no existe, Docker crea el volumen por usted.

Especifique el montaje de enlace usando el indicador --mount ejecutando:

```
sudo docker run --name postgres-db -e POSTGRES_PASSWORD=password--
mount type=bind,source="$pwd",target=/var/lib/postgresql/data -p 2000:5432 -d
postgres
```

--name: nombre del contenedor.
-e: crear variables de entorno
--mount: crear volumen de montaje del contenedor
Type: specifica el tipo de montaje, puede ser bind, volulme, tmpfs.
Target: ruta destino del contenedor.
-p: mapear un puerto.
-d: ejecutamos el contenedor en segundo plano.

Use este código para especificarlo con el indicador -v:

```
sudo docker run --name postgres-db2 -e POSTGRES_PASSWORD=password´
-v "$pwd":/var/lib/postgresql/data -p 2000:5433 -d postgres
```

Tenga en cuenta que, en ambos casos, se debe especificar $pwd, el directorio de trabajo, como origen. Básicamente, le estamos diciendo a Docker que cree el montaje de enlace en el directorio en el que nos encontramos actualmente.

Comparaciones de volúmenes con nombre y montaje de enlaces.

Los montajes de enlace y los volúmenes con nombre son los dos tipos principales de volúmenes que vienen con el motor de Docker. Sin embargo, hay controladores de volumen adicionales disponibles para admitir otros casos de uso (SFTP, Ceph, NetApp, S3 y más).

	Volúmenes con nombre	**Montaje de enlaces (binds mount)**
Ubicación del Host	Docker elige la ubicación	Tú eliges la ubicación
Ejemplo de montaje (usando -v)	my-volume:/usr/local/data	/path/to/data:/usr/local/data
Rellena el nuevo volumen con el contenido del contenedor	Si	No
Admite controladores de volúmenes	Si	No

Ejemplo práctico.

Crear un directorio en el escritorio llamado “testvolumen” donde se colocará un archivo con el nombre index.html con un texto que muestre “hola mundo”, esta será la ruta absoluta del host.

Crear dos contenedores en docker, el primero con la imagen apache y el segundo con la imagen de Nginx, usar un volumen para compartir el código entre ambos contenedores.

Copiar el siguiente código en el archivo index.html, cuyo archivo se encuentra en el directorio llamado “testvolumen”

index.html

```
DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Testvolumen</title>
</head>
<body>
<h1>Hola mundo</h1>
</body>
</html>
```

Solución

Creando el primer contenedor con la imagen de apache.

```
sudo docker run -d -it -p 8000:80 --name servidorapache -v
/home/angel/Escritorio/testvolumen:/usr/local/apache2/htdocs httpd
```

Contenedor apache activo.

Si usa la orden Docker ps puede observar que el contenedor apache esta activo en el puerto 8000.

Abrir el navegador de preferencia y colocar http://localhost:8000 para mostrar la página html antes creada.

Ilustración 23: Salida del contenido del volumen testvolumen

Detener y eliminar el contenedor anterior y crea el nuevo con la imagen Nginx

```
sudo docker stop servidorapache
sudo docker rm servidorapache
```

Creando el segundo contenedor con la imagen de Nginx.

```
sudo docker run -d -p 3000:80 --name servidorNginx -v /home/angel/Escritorio/testvolumen:/usr/share/nginx/html nginx
```

Entrar nuevamente al navegador y colocar localhost, pero ahora en el puerto 3000 como se aprecia en la imagen de abajo.

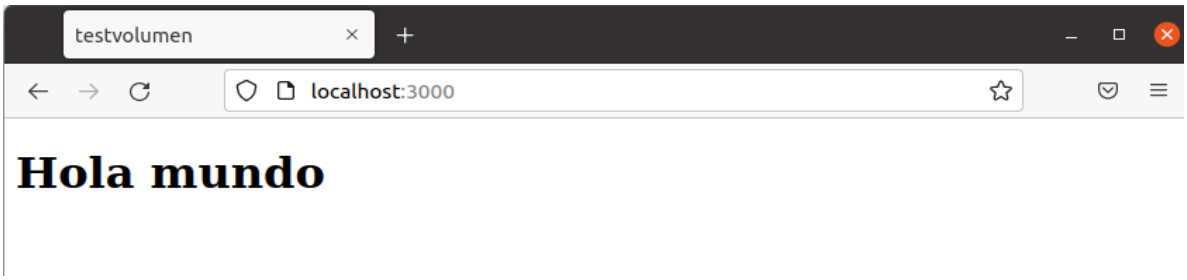


Ilustración 24: Salida del volumen testvolumen usando otro contenedor

Se muestra el mismo contenido, esto es debido a que los contenedores comparten el mismo volumen.

Ejercicios propuestos.

Creación y uso de volúmenes

1) Crear los siguientes volúmenes con la orden docker volume: *volumen_datos* y *volumen_web*.
2) Una vez creados estos contenedores:
    a. Arrancar un contenedor llamado c1 sobre la imagen php:7.4-apache que monte el volumen_web en la ruta /var/www/html y que sea accesible en el puerto 8080.
    b. Arrancar un contenedor llamado c2 sobre la imagen mariadb que monte el volumen_datos en la ruta /var/lib/mysql y cuya contraseña de root sea admin.
3) Intenta borrar el volumen *volumen_datos*, para ello tendrás que parar y borrar el contenedor c2 y tras ello borrar el volumen.
4) Copia o crea un fichero index.html al contenedor c1, accede al contenedor y comprueba que se está visualizando.
5) Borra el contenedor c1 y crea un contenedor c3 con las mismas características que c1 pero mapeando el puerto 8081.

Bind mount para compartir datos

1. Crea una carpeta llamada saludo y dentro de ella crea un fichero llamado index.html con el siguiente contenido (Deberás sustituir ese **** por tu nombre.): <h1>HOLA SOY **** </h1>
2. Una vez hecho esto arrancar dos contenedores basados en la imagen php:7.4-apache que hagan un bind mount de la carpeta saludo en la carpeta

/var/www/html del contenedor. Uno de ellos vamos a acceder con el puerto 8181 y el otro con el 8282. Y sus nombres serán c1 y c2.

3. Modifica el contenido del fichero ~/saludo/index.html.
4. Comprueba que puedes seguir accediendo a los contenedores, sin necesidad de reiniciarlos.

5.4 Guía 4: Creación y comunicación entre múltiples contenedores

En esta guía se profundizará los comandos para la administración, comunicación de contenedores desde la CLI.

Docker proporciona mecanismos llamados enlaces para configurar la comunicación entre contenedores. Si un nuevo contenedor es enlazado a un contenedor existente, el nuevo contenedor obtendrá la información de conexión de un contenedor existente a través de las variables de entorno.

Esto proporciona un camino fácil para establecer comunicación entre dos contenedores proporcionando al nuevo contenedor la información explícita sobre cómo acceder a su compañero.

Duración estimada en la realización de la guía

45 minutos.

Desarrollo

Esta guía te mostrara cómo podemos usar el entorno de desarrollo de Docker para:

- Configurar y ejecutar una instancia de servidor PHP Apache local.
- Funcione un sitio web dinámico impulsado por PHP.
- Configurar una base de datos MySQL para ejecutar secuencias de comandos SQL, obtener registros y mostrarlos en un sitio web basado en PHP.
- Conectar phpMyAdmin para una mejor gestión de la base de datos

Creación y configuración de los contenedores.

Descargar las imágenes

```
sudo docker pull mysql:5.7
```

```
sudo docker pull php:7.4-apache
```

```
sudo docker pull phpMyAdmin
```

Verificar las imágenes.

```
sudo docker images
```

Crea el primer contendor a partir de la imagen MySQL previamente descargada en el sitio web será indispensable una base de datos eso es lo que configuraremos.

Ejecuta la imagen de MySQL, con todas las configuraciones necesarias.

```
sudo docker run --name base-de-datos -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=prueba -e MYSQL_USER=usuario1 -e MYSQL_PASSWORD=pass1234 -v /home/ubuntu/Desktop/Guia4/mysql:/var/lib/mysql -d mysql:5.7
```

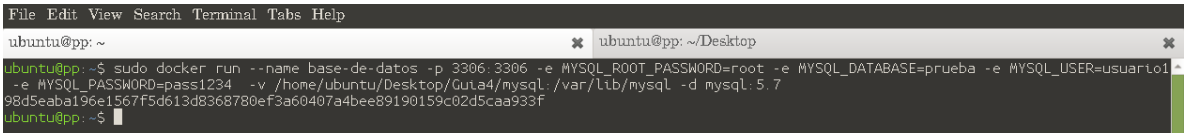


Ilustración 25: Creación del contenedor base-de-datos

-p: Permite mapear los puertos entre nuestra máquina local y el contenedor. En este caso, estamos mapeando el puerto 3306 de nuestra máquina local con el puerto 3306 del contenedor.

--name: Asignarle un nombre a nuestro contenedor. Si no le asignamos Docker nos asignará un nombre automáticamente.

-v: Pasamos la ruta de nuestro volumen en nuestra maquina local y copiamos ese volumen en el directorio por omisión de MySQL en el contenedor.

-e: Pasarle al contenedor variables de entorno.

MYSQL_ROOT_PASSWORD= Contraseña del usuario root.

MYSQL_DATABASE= Nombre de la base de datos

MYSQL_USER= Usuario para esa base de datos

MYSQL_PASSWORD= Contraseña para el usuario1

-d: Ejecuta el contenedor en segundo plano.

mysql:5.7: Nombre de la imagen a usar en el contenedor.

Verifica que el contenedor este creado y ejecutando**.**

```
sudo docker ps
```

Para confirmar que la base de datos está en funcionamiento, conéctese a ella

Entra al contenedor

```
sudo docker exec -it base-de-datos /bin/bash
```

Entra a la consola MySQL

```
mysql -u root -p
```

Nos pedirá una contraseña debe ingresar la que se escribió cuando creo el contenedor la contraseña a usar es la que pasamos en el parámetro MYSQL_ROOT _PASSWORD.

Verifica si se creó la base de datos

```
show databases;
```

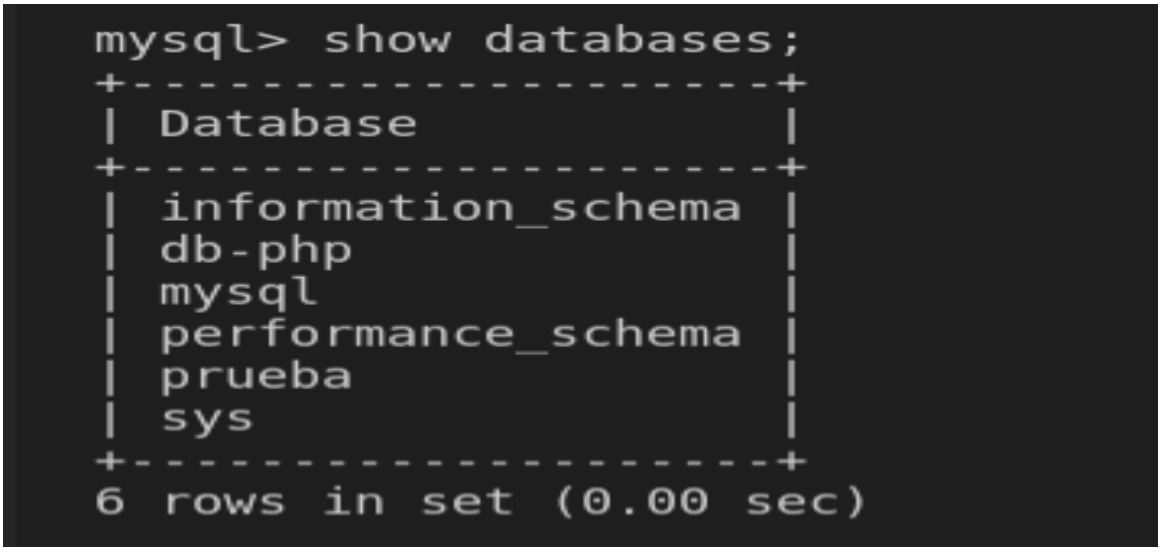


Ilustración 26: Vista de las bases de datos en la consola de MySQL

Usa la base de datos creada.

```
use prueba;
```

Crea una tabla

```
create table usuarios (username varchar(20)primary key not null,nombre varchar(30),correo varchar(20),clave varchar(20));
```

Inserta usuarios

```
insert into usuarios values ('victor2022', 'Victor Gonzalez', 'victorg@gmail.com','miclave'),('gaby22', 'Gabriela Rivera', 'gabyr@gmail.com','miclave2'),('jose22', 'Miguel Obando', 'miguelj@gmail.com','miclave3');
```

Consulta los datos.

```
SELECT * from usuarios;
```

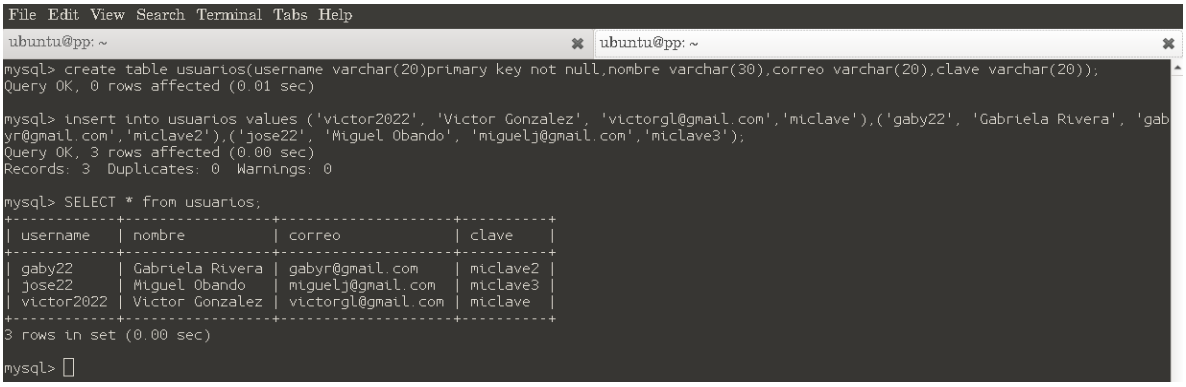


Ilustración 27: Creación de la tabla y muestra el contenido

Crea el segundo contenedor con la imagen php-apache

Ejecuta la imagen de php:7.4-apache, con todas las configuraciones necesarias.

```
sudo docker run -p 8080:80 -v /home/ubuntu/Desktop/Guia4/sweb:/var/www/html
--name php-apache -d --link base-de-datos php:7.4-apache
```

-p: Mapear los puertos entre nuestra máquina local y el contenedor. En este caso, estamos mapeando el puerto 8000 de nuestra máquina local con el puerto 80 del contenedor.

-v: Ruta donde están los archivos de nuestra página web(volumen) ya deben estar creados los directorios para evitar restricciones del root. Copiar ese volumen que contiene nuestra aplicación php al directorio por omisión del servidor apache en el contenedor.

--name: Nombre al contenedor.

-d: Ejecutar en segundo plano.

--link: Propiedad para enlazar contenedores. Se le pasa el nombre del contenedor con el que se comunicara.

Php:7.4-apache: Nombre de la imagen que se va a utilizar.

Verifica que los contenedores están creados y se ejecutan.

```
sudo docker ps
```

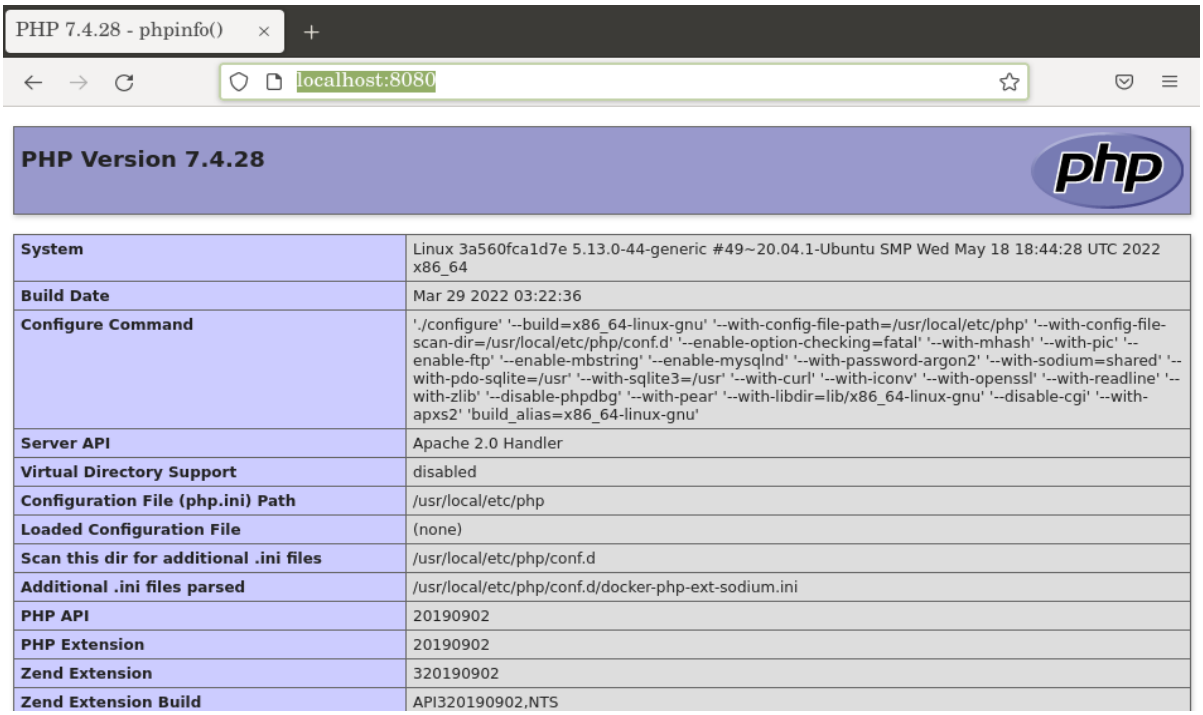
Crea un archivo index.php en el directorio de trabajo para verificar que funciona el servidor web.

index.php

```
<?php
phpinfo();
?>
```

En el navegador escribe.

http://localhost:8080/



System	Linux 3a560fca1d7e 5.13.0-44-generic #49~20.04.1-Ubuntu SMP Wed May 18 18:44:28 UTC 2022 x86_64
Build Date	Mar 29 2022 03:22:36
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' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' '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	/usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,NTS

Ilustración 28: Información sobre la configuración PHP

Verificar la conexión entre contenedores.

Hasta acá ambos servidores funcionan correctamente ahora debe corroborar que se están comunicando entre sí.

Edita el archivo index.php para crear la conexión a la base de datos del contenedor base-de-datos.

index.php

```
<?php
$servername = "base-de-datos:3306";
$username = "root";
$password = "root";
$database = "prueba";
$db= mysqli_connect($servername,$username,$password,$database);
?>
<!DOCTYPE html>
<html>
<head>
        <title>Mostar datos</title>
</head>
<body>
<br>
<table  border="4px" >
 <tr >
        <td>UserName</td>
        <td>Nombre</td>
        <td>Correo</td>
        <td>Cave</td>
   </tr>
        <?php
        $sql="SELECT * from usuarios";
        $result=mysqli_query($db,$sql);
        while ($mostrar=mysqli_fetch_array($result)){
        ?>
   <tr>
            <td><?php echo $mostrar['username'] ?></td>
            <td><?php echo $mostrar['nombre'] ?></td>
            <td><?php echo $mostrar['correo'] ?></td>
            <td><?php echo $mostrar['clave'] ?></td>
        </tr>
        <?php
         }
```

```
        ?>
</table>
</body>
```

Para la conexión al servidor de base de datos se pasa el nombre del contendor donde se aloja este servicio y el puerto de conexión del servidor (servername) el resto seria usuario, contraseña y la base de datos. Ve al navegador y si actualizas la página nos dará un error similar al siguiente.

```
Fatal error: Uncaught Error: Call to undefined function mysqli_connect() in
/var/www/html/index.php:7 Stack trace: #0 {main} thrown in
/var/www/html/index.php on line 7
```

Lo que sucede es que el servidor de PHP no tiene una extensión para conectarse con MYSQL entones el por qué no reconoce la función de conexión, para resolverlo debe instalarla.

Como se agregará paquetes de mysqli al contenedor que es una extensión PHP que conectará el PHP Apache al servidor MySQL. Para poder agregar esta extensión al contenedor php se tiene que entrar al propio contenedor.

```
sudo docker exec -it php-apache /bin/bash
```

Instala la extensión necesaria.

```
docker-php-ext-install mysqli pdo pdo_mysql
```

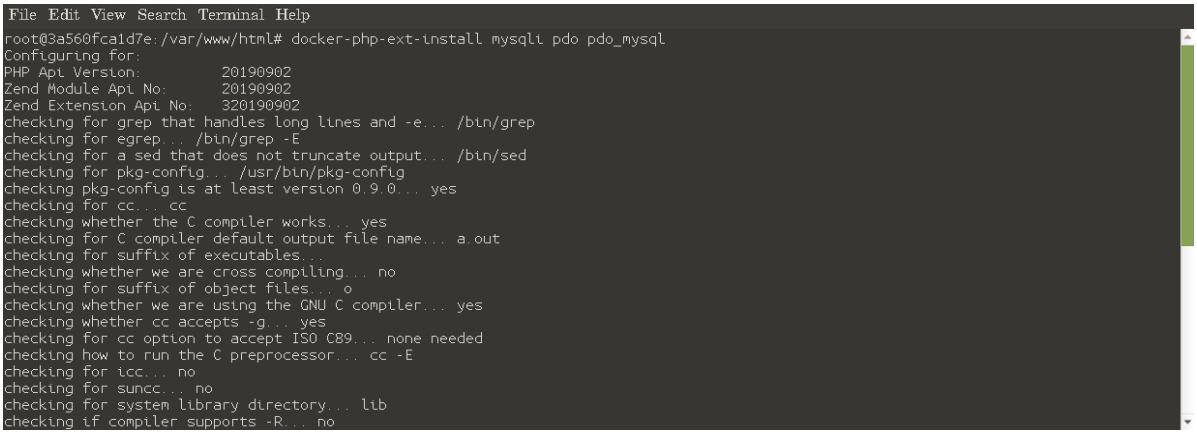


Ilustración 29: Instalación de la extensión pdo_mysql

Una vez que se instala la extensión salir del contendor y reiniciar.

```
sudo docker restart php-apache
```

Actualizar la página en el navegador.

Ilustración 30: Salida del del contenedor php-apache

El siguiente paso será crear un tercer contenedor con la imagen de phpmyadmin que se conecte a MySQL para que sea mucho más cómodo gestionar la base de datos o crear nuevas.

Ejecuta la imagen phpmyadmin con sus configuraciones necesarias.

```
sudo docker run -p 8000:80 --name mi-phpmyadmin --link base-de-datos:db -d phpMyAdmin
```

-p: Mapear los puertos entre nuestra máquina local y el contenedor.

--name: Nombre del contenedor.

--link: Proporciona acceso a otro contenedor en nuestra máquina. Se le pasa el nombre del contenedor y el recurso al que accederá en mysql.

-d: Ejecutar en segundo plano.

phpmyadmin: Nombre de la imagen que va a utilizar.

En el navegador abrir una pestaña nueva y escribir

http://localhost:8000/

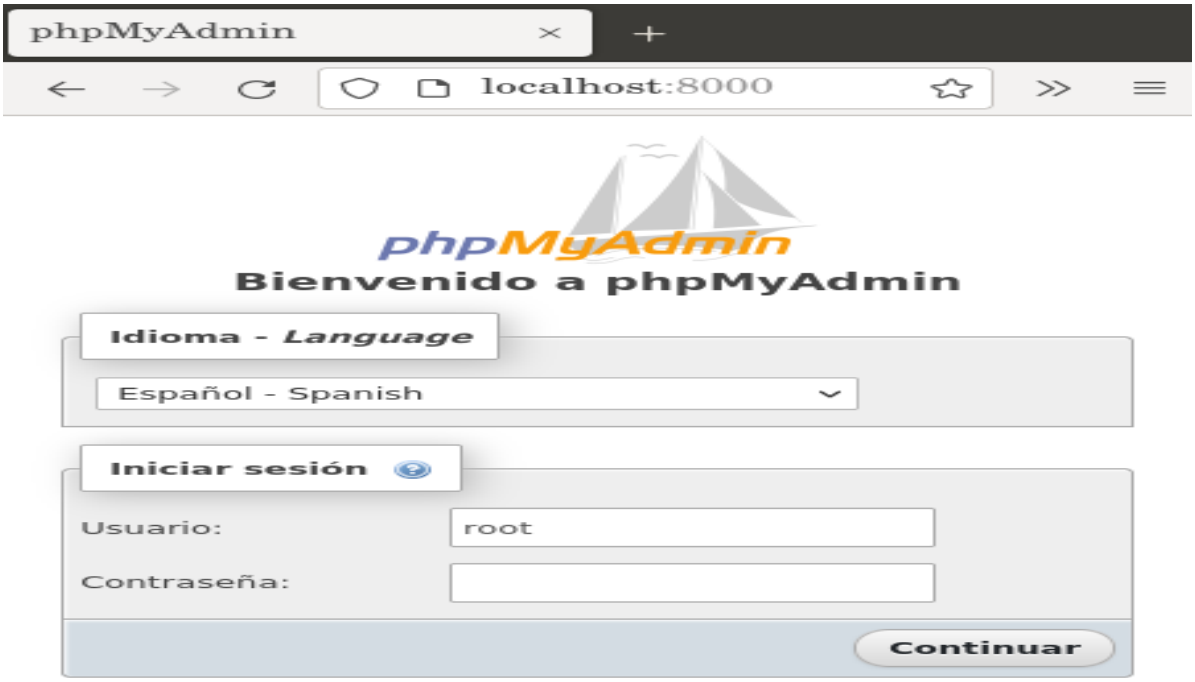


Ilustración 31: Inicio de sesión de phpMyAdmin

Inicia sesión y verifica que está la base de datos con los datos que ingresamos por consola cuando creamos el contendor base-de-datos.

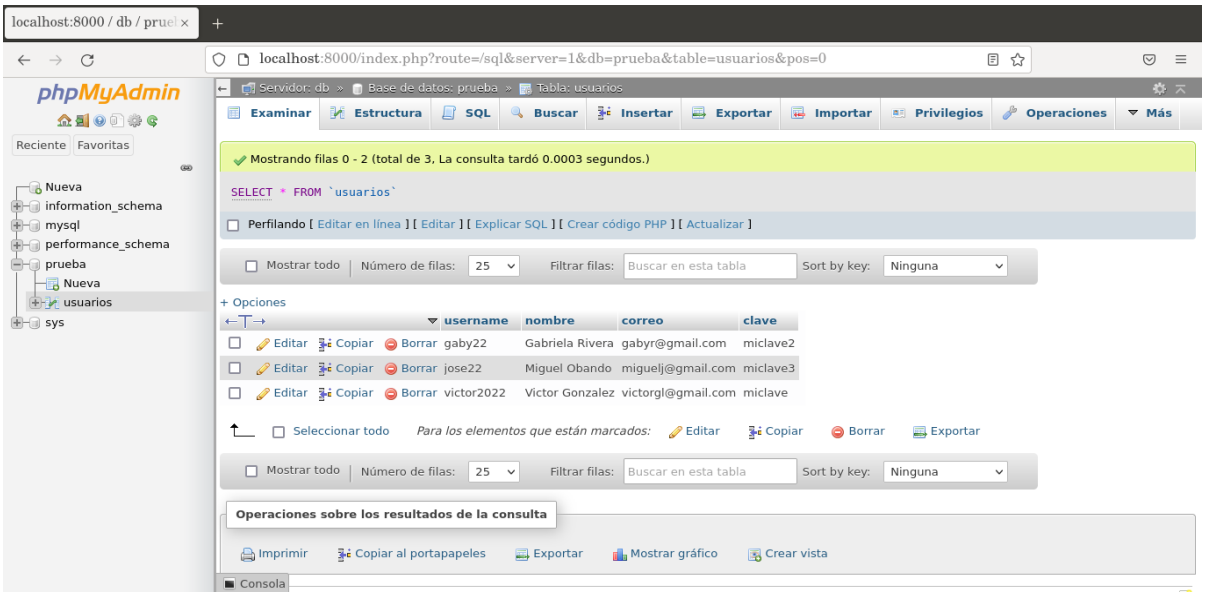


Ilustración 32: Vista de la tabla y sus valores en phpMyAdmin

Listo ya puedes a empezar a trabajar en tu proyecto web tienes todo tu servidor web, base de datos y phpMyAdmin para el manejo más cómodo para tus bases de datos.

5.5 Guía 5. Docker-compose para el manejo de contenedores

En esta guía se abordará sobre docker-compose que es una herramienta para definir y ejecutar aplicaciones Docker multicontenedor que permite simplificar el uso de Docker a partir de archivos YAML, de esta forma es más sencillo crear contendores que se relacionen entre sí, conectarlos, habilitar puertos, volúmenes, etc. Nos permite lanzar un solo comando para crear e iniciar todos los servicios desde su configuración (YAML), esto significa que puedes crear diferentes contenedores y al mismo tiempo diferentes servicios en cada contenedor, integrarlos a un volumen común e iniciarlos y/o apagarlos, etc. Este es un componente fundamental para poder construir aplicaciones y microservicios.

Docker-Compose funciona en todos los entornos: production, staging, development, testing.

Duración estimada en la realización de la guía

40 minutos.

Desarrollo

Comandos básicos de docker-compose

Mostrar la versión de Docker-compose

```
sudo docker-compose –versión
```

Mostrar la ayuda de Docker-compose.

```
sudo docker-compose –help
```

Iniciar todo el entorno o los servicios especificados en el archivo docker-compose.yml.

```
sudo docker-compose up -d
```

Listar contenedores ejecutándose en el contexto de docker-compose.

```
sudo docker-compose ps
```

Detener el/los contenedores.

```
sudo docker-compose down
```

La configuración que se hizo de forma manual con docker en la guía anterior se hará en un solo archivo el docker-compose donde solo bastará ejecutar un solo comando y ya tendríamos nuestros servicios y contenedores creados.

Configuración del servidor.

Crea una carpeta donde se guardará el proyecto.

Crear el archivo docker-compose.yml

Una cuestión importante que se debe tener en cuenta al trabajar con los ficheros yml es que haremos uso de tabulaciones para identar el código vayamos escribiendo.

```
version: '3.7'
services:
  php-apache:
    container_name: my-php-apache
    image: php:7.4-apache
     volumes:
      - ./php/src:/var/www/html/
    ports:
```

```
    - 8080:80
  networks:
    - lan-networks
```

version: Seleccionar la versión de Docker que desea usar.

```
version: '3.7'
```

services: Servicios que se crearan.

```
services:
   php-apache
```

Para el contenedor PHP-Apache, debe especificar los siguientes.

container_name: Este es solo un nombre aleatorio como le gustaría nombrar a su contenedor PHP.

```
container_name:
```

image: Imagen base de la cual se creará el contenedor, esta es imagen oficial de PHP con apache ya integrado php:7.4-apache.

```
image:php:
```

volumes: En su directorio de trabajo en la carpeta src estarán sus archivos de código fuente php. Si tuviera que ejecutar un script PHP, ese archivo tendría que

estar en ese directorio. Y el contenido de ese volumen sera copiado a la ruta por omisión del servidor apache.

```
volumes:

 - ./src:/var/www/html/
```

ports: Asignación de puerto en su computadora local, al puerto del contenedor que funciona como servidor.

```
   ports:
     - 8080:80
```

networks: Crea una red que se conectaran los demás contenedores para compartir sus variables de entorno.

```
networks
    - lan-networks
```

Las aplicaciones se ejecutan dentro de contenedores, de manera independiente unas de otras. Sin embargo, estas aplicaciones necesitan comunicarse entre ellas

Dirígete al directorio de tu proyecto ./src, crea un index.php.

```
<?php

echo "Hola, este es el contenedor my-php-apache";

?>
```

Ejecuta el archivo docker-compose.yml

```
sudo docker-compose up -d
```

Eso extraerá toda la información creará y ejecutará el contenedor.

Verificar si se está ejecutando el contenedor.

```
sudo docker-compose ps
```

```
      Name                    Command               State                         Ports
-------------------------------------------------------------------------------------------------
my-php-apache       docker-php-entrypoint apac ...   Up      0.0.0.0:8080->80/tcp,:::8080->80/tcp
```

Ilustración 33: contenedor my-php-apache ejecutando

Para asegurarse de que el contenedor esté configurado y ejecutando los scripts PHP, abre el navegador y escribe.

http://localhost:8080/

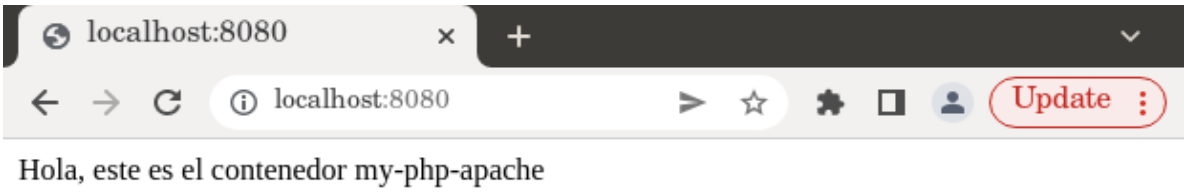


Ilustración 34: Vista de la ejecución del contendor

Configuración de MySQL.

Agrega el servicio MySQL al archivo docker-compose.yml

```
db:
    container_name: mi-db
    image: mysql:5.7
    environment:
        MYSQL_ROOT_PASSWORD: root
        MYSQL_DATABASE: prueba
        MYSQL_USER: usuario1
        MYSQL_PASSWORD: pass1234
    ports:
        - "3306:3306"
```

```
  networks
    - lan-networks
```

Para MySQL necesitamos configurar:

MYSQL_ROOT_PASSWORD= Contraseña del usuario root.

MYSQL_DATABASE= Nombre de la base de datos

MYSQL_USER= Usuario para esa base de datos

MYSQL_PASSWORD= Contraseña para el usuario1

Pruebe si el contenedor funciona como se espera.

Dirígete al archivo index.php y cambia el código por el siguiente, lo que se agrego es una conexión a la base de datos.

```
<?php
$servername = "db";
$username = "root";
$password = "root";
$database = "prueba";
$db = mysqli_connect($servername, $username, $password, $database);
if (!$db) {
    die("Fallo la conexion: " . mysqli_connect_error());
}
echo "Conexión exitosa!";
mysqli_close($db);
?>
```

Si ejecuta el archivo docker-compose.yml ambos servicios se crearán, pero si recordamos en la práctica anterior dio este error.

```
Fatal error: Uncaught Error: Call to undefined function mysqli_connect() /var/www/html/index.php:7 Stack trace: #0 {main} thrown /var/www/html/index.php on line 7
```

Para solucionarlo necesita agregar la extensión mysqli de MySQL a la imagen de php:7.4-apache para que php pueda conectarse con MySQL

En el directorio de su proyecto cree un archivo Docker, asígnele el nombre Dockerfile y agregue las siguientes configuraciones de PHP.

```
FROM php:7.4-apache
RUN apt-get update && apt-get upgrade -y
RUN docker-php-ext-install mysqli pdo pdo_mysql
```

Ahora necesita construir esta imagen personalizada dentro del servicio php-apache en el docker-compose.yml. php-apache también depende del servicio db para conectarse a MySQL especificando un depends_on:

Modifica el archivo docker-compose.yml

```
version: '3.7'
services:
  php-apache:
    container_name: my-php-apache
    build:
      context: ./
      dockerfile: Dockerfile
    depends_on:
      - db
    volumes:
      - ./src:/var/www/html/
    ports:
      - 8080:80
    networks:
      - lan-networks
```

```
 db:
   container_name: mi-db
   image: mysql:5.7
   environment:
     MYSQL_ROOT_PASSWORD: root
     MYSQL_DATABASE: prueba
     MYSQL_USER: usuario1
     MYSQL_PASSWORD: pass1234
   ports:
     - "3306:3306"

   networks:
     - lan-networks
networks:
 lan-networks:
```

Ejecutar sudo docker-compose up -d para extraer y configurar el entorno MySQL y se agregará y creara el nuevo contenedor mi-db que es el de mysql.

Recargamos la página en nuestro navegador.

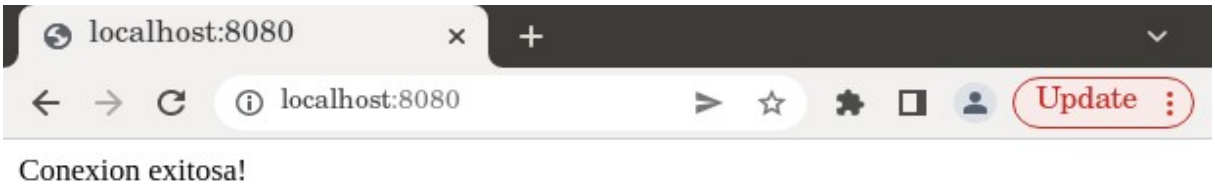


Ilustración 35: Muestra la conexión exitosa entre contenedores

El entorno PHP Apache y MySQL ahora están configurados, y puede comenzar a desarrollar su aplicación.

Configuración de phpMyAdmin.

La conexión MySQL ahora está bien. Vea cómo puede obtener algunos datos de una base de datos MySQL y mostrarlos en una página web utilizando scripts PHP.

La aplicación interactúa con una base de datos; probablemente querrá una interfaz para interactuar con sus datos. Agregue servicios PHPMyAdmin para proporcionarnos una interfaz para interactuar con la base de datos MySQL.

Agregue un servicio PHPMyAdmin como se muestra a continuación.

```
phpmyadmin:
    image: phpmyadmin
    links:
      - db
    networks:
      - lan-networks
    ports:
      - 8000:80
    environment:
        PMA_ARBITRARY: 1
        PMA HOST: db
        MYSQL_ROOT_PASSWORD: root
networks:
  lan-networks:
```

PMA HOST: servidor de base de datos
MYSQL_ROOT_PASSWORD: contraseña del usuario de mysql

Detener los contenedores y los volver a ejecutar para que cree el tercer contenedor de phoMyAdmin

Abrir una nueva pestaña del navegador y escribir.
http://localhost:8000/

Para iniciar sesión en el panel Phpmyadmin, use:

Servidor**:** db que es el nombre del servicio de base de datos en el docker-compose.yml

El nombre puede ser el root con la contraseña de la variable MYSQL_ROOT_PASSWORD: root

O bien el usuario y la contraseña establecidas en las variables de entorno

MYSQL_USER: usuario1

MYSQL_PASSWORD: pass1234

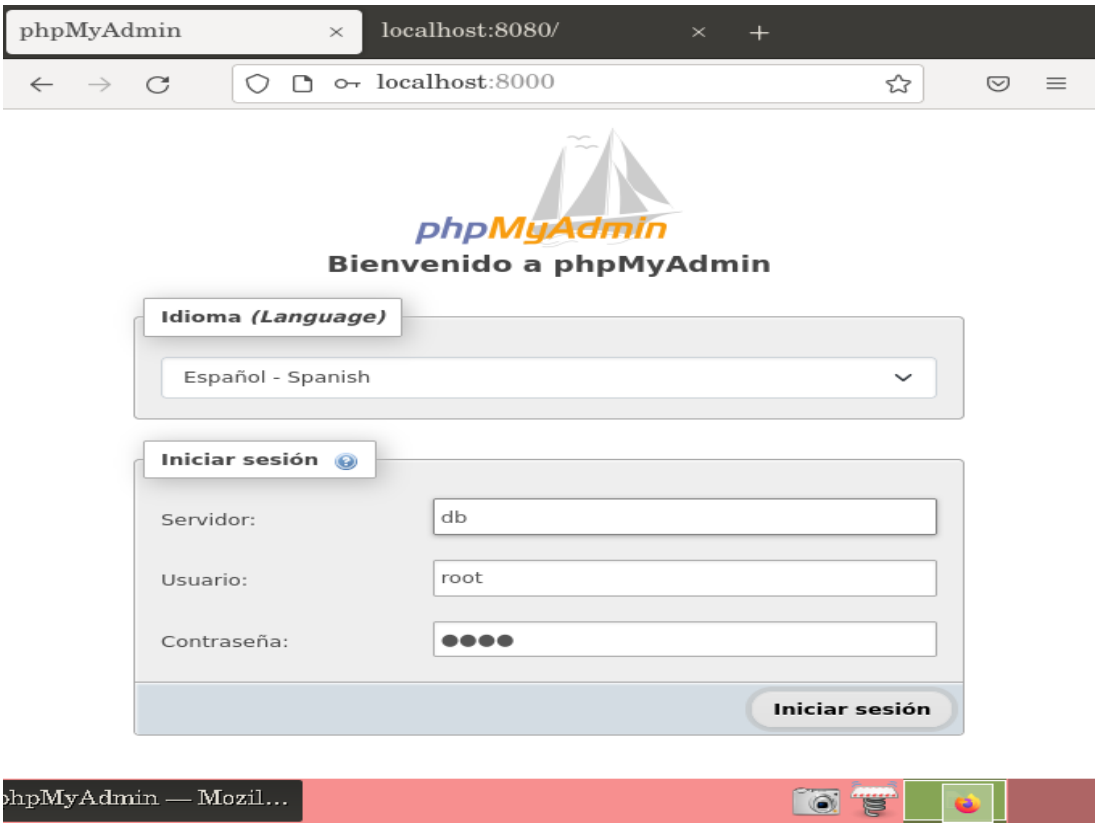


Ilustración 36: Inicio de sesion de phpMyadmin

Ahora puede ver que la base de datos que fue definida y configurada, ya puede comenzar a interactuar con Phpmyadmin.

En nuestra base de datos “prueba” cree una tabla llamada “personas” con cuatro columnas id, nombre, correo, sexo y rellene algunos registros.

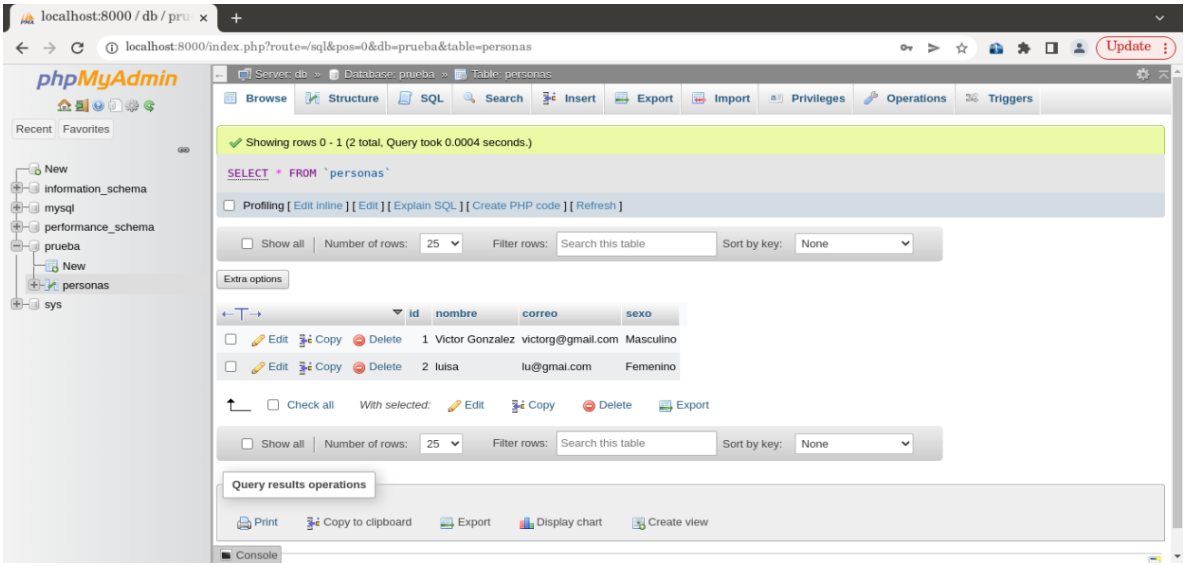


Ilustración 37: Tabla personas de la base de datos prueba

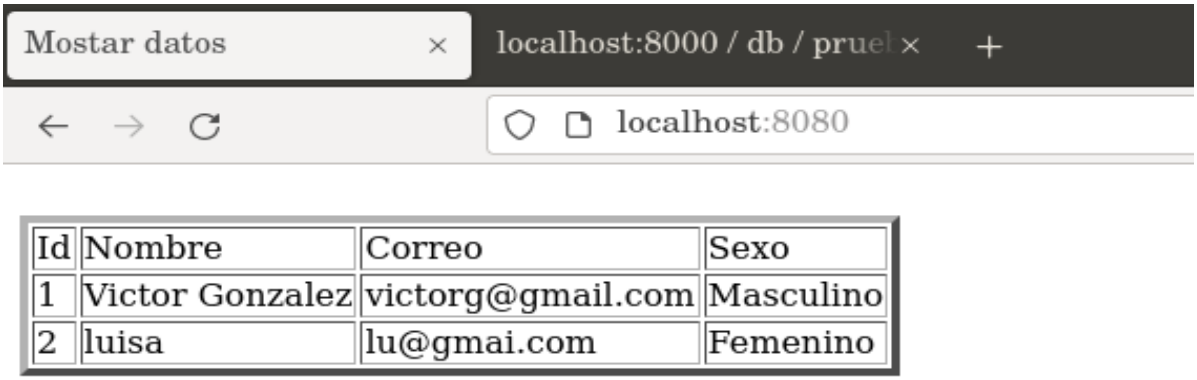
Edita el archivo index.php

```
<?php
$db=mysqli_connect('db','root','root','prueba')
?>

<!DOCTYPE html>
<html>
<head>
        <title>Mostar datos</title>
</head>
<body>
<br>
<table  border="4px" >

<tr >
        <td>Id</td>
        <td>Nombre</td>
        <td>Correo</td>
        <td>Sexo</td>
   </tr>
        <?php
        $sql="SELECT * from personas";
        $result=mysqli_query($db,$sql);
```

```
        while ($mostrar=mysqli_fetch_array($result)){
        ?>
    <tr>
            <td><?php echo $mostrar['id'] ?></td>
            <td><?php echo $mostrar['nombre'] ?></td>
            <td><?php echo $mostrar['correo'] ?></td>
            <td><?php echo $mostrar['sexo'] ?></td>
        </tr>
        <?php
         }
        ?>
</table>
</body>
```

Actualice para ver los resultados.

http://localhost:8080/



Id	Nombre	Correo	Sexo
1	Victor Gonzalez	victorg@gmail.com	Masculino
2	luisa	lu@gmai.com	Femenino

Ilustración 38: Tabla prueba mostrada en el navegador

5.6 Entorno de desarrollo para PHP

Es un lenguaje de programación de código abierto del lado del servidor que se utiliza principalmente para crear páginas web dinámicas. La abreviatura nació originariamente de “Personal Home Page Tools”, aunque hoy en día se ha convertido en el acrónimo recursivo para “**PHP**: Hipertextos Preprocessor”

Software y servicios incluidos en el contenedor.

php:8.0-apache
MySQL :5.7
PhpMyAdmin: latest

Los archivos están disponibles en el siguiente enlace:

https://estunanleonedu-my.sharepoint.com/:u:/g/personal/victor_gonzalez15_est_unanleon_edu_ni/EVCdSmu6wSlAonor86KgyAUBzA9LxN8_K80votofAqdmYw?e=Pv8cwP

El cual contiene los siguientes archivos de Docker y un directorio app donde se alojará el código fuente de la aplicación.

Dockerfile

```
FROM php:8.0-apache

RUN apt-get update && apt-get upgrade -y

RUN docker-php-ext-install mysqli pdo pdo_mysql

EXPOSE 8000
```

docker-compose.yml

```
version: '3.7'

services:
############## App de PHP ###############
  web:
    build:
     context: ./
     dockerfile: Dockerfile
    container_name: app-php
    depends_on:
      - php-db
    volumes:
     - ./app:/var/www/html/
    ports:
     - 8000:80
    networks:
    - lan-networks
####################Servidor de base de datos ########
  php-db:
    image: mysql:5.7
    container_name: php-db
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: bd-php
      MYSQL_USER: devuser
      MYSQL_PASSWORD: devpass
    volumes:
     -./mysql-data:/var/lib/mysql
    ports:
      - 3306:3306
    networks:
    - lan-networks
#################phpMyAdmin #################
  phpmyadmin:
    image: phpmyadmin
   container_name: php-phpmyadmin
    links:
     - php-db
```

```
    networks:
      - lan-networks
    ports:
      - 8080:80
    environment:
        PMA_ARBITRARY: 1
        PMA_HOST: php-db
        MYSQL_ROOT_PASSWORD: root
networks:
  lan-networks:
```

Ejecute el archivo docker-compose

Use el comando **sudo docker-compose up -d,** vera algo así:

Ilustración 39: Ejecución de las instrucciones dadas en docker-compose.yml

Debe tener acceso a internet la primera vez que ejecute este comando ya que se descargarán las imágenes de Docker correspondiente al contenedor.

Use el comando **sudo docker compose ps** para verificar que se han creado los contenedores.

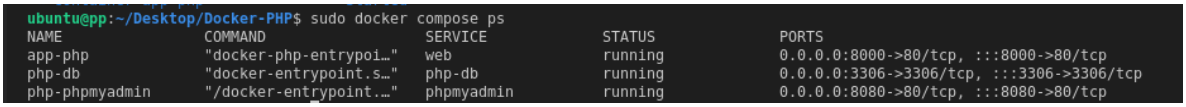
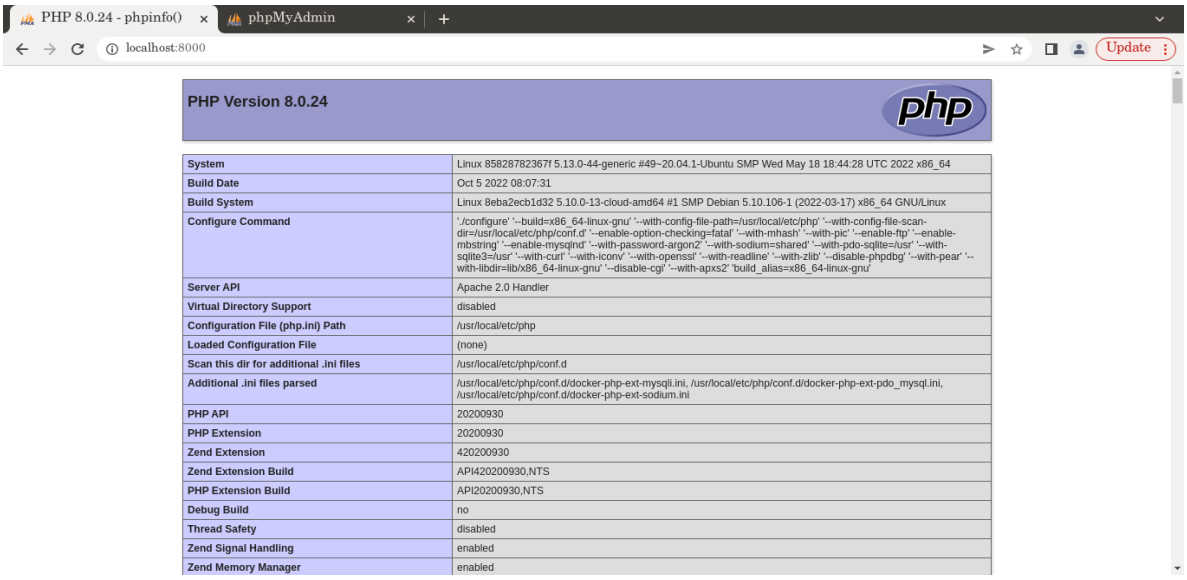


Ilustración 40: Contenedores en ejecución

Una vez creado el ambiente de desarrollo es posible empezar a utilizarlo para la creación de páginas web en lenguaje PHP.

Accede a: http://localhost:8000/



System	Linux 85828782367f 5.13.0-44-generic #49~20.04.1-Ubuntu SMP Wed May 18 18:44:28 UTC 2022 x86_64
Build Date	Oct 5 2022 08:07:31
Build System	Linux 8eba2ecb1d32 5.10.0-13-cloud-amd64 #1 SMP Debian 5.10.106-1 (2022-03-17) x86_64 GNU/Linux
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' '--with-pic' '--enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-password-argon2' '--with-sodium=shared' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-iconv' '--with-openssl' '--with-readline' '--with-zlib' '--disable-phpdbg' '--with-pear' '--with-libdir=lib/x86_64-linux-gnu' '--disable-cgi' '--with-apxs2' '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	/usr/local/etc/php/conf.d/docker-php-ext-mysqli.ini, /usr/local/etc/php/conf.d/docker-php-ext-pdo_mysql.ini, /usr/local/etc/php/conf.d/docker-php-ext-sodium.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled

Ilustración 41: Phpinfo

Accede a: http://localhost:8080/

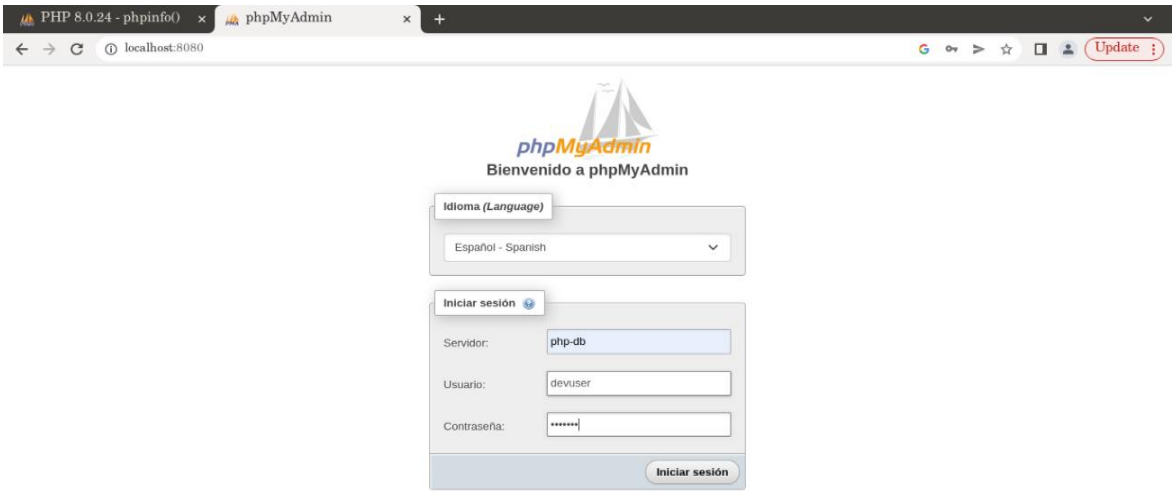


Ilustración 42: PhpMyAdmin

5.7 Entorno de desarrollo para Laravel

Laravel es un popular framework de PHP. Permite el desarrollo de aplicaciones web totalmente personalizadas de elevada calidad. Laravel es un framework PHP. Es uno de los frameworks más utilizados y de mayor comunidad en el mundo de Internet.

Como framework resulta bastante moderno y ofrece muchas utilidades potentes a los desarrolladores, que permiten agilizar el desarrollo de las aplicaciones web.
Laravel pone énfasis en la calidad del código, la facilidad de mantenimiento y escalabilidad, lo que permite realizar proyectos desde pequeños a grandes o muy grandes. Además, permite y facilita el trabajo en equipo y promueve las mejores prácticas.
Esta propuesta de contenedores que permiten ejecutar Laravel está compuesta por los siguientes servicios:

mysql:8.0
phpMyAdmin: latest
php:8.0-apache
Laravel-9x

Los recursos para este entorno están disponibles en el siguiente enlace.

https://estunanleonedu-my.sharepoint.com/:u:/g/personal/victor_gonzalez15_est_unanleon_edu_ni/ERu0g6eYyQ5Bv68lCHyEt7sBjpXLdhhdGr6lEHtDVxtX7g?e=HFfZV2

El archivo .zip contiene archivos Dockers y la app de Laravel y sus dependencias.

Dockerfile.

```
FROM php:8.0-apache
# 1. paquetes de desarrollo
RUN apt-get update
RUN apt-get install -y \
    git \
    zip \
    curl \
    sudo \
    unzip \
    libicu-dev \
    libbz2-dev \
    libpng-dev \
    libjpeg-dev \
    libmcrypt-dev \
    libreadline-dev \
    libfreetype6-dev \
    g++

RUN curl -sL https://deb.nodesource.com/setup_12.x | bash -
RUN apt-get install -y nodejs

# 2. Configuraciones apaches + document root
ENV APACHE_DOCUMENT_ROOT=/var/www/html/public
RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/sites-available/*.conf
RUN sed -ri -e 's!/var/www/!${APACHE_DOCUMENT_ROOT}!g' /etc/apache2/apache2.conf /etc/apache2/conf-available/*.conf

# 3. mod_rewrite para reescritura de URL y mod_headers para encabezados adicionales .htaccess como Access-Control-Allow-Origin-
RUN a2enmod rewrite headers

# 4. Configuración base de php, luego agregamos extensiones
RUN mv "$PHP_INI_DIR/php.ini-development" "$PHP_INI_DIR/php.ini"

RUN docker-php-ext-install mysqli pdo pdo_mysql

# 5. composer
```

```
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

Docker-compose.yml

```
version: '3.7'

services:
################## App Laravel ##################
  laravel-app:
    build:
      context: '.'
    container_name: laravel-app
    volumes:
      - ./laravel:/var/www/html
    ports:
      - 8000:80

    networks:
    - lan-laravel
############## Servidor de base de datos ########
  laravel-db:
    image: mysql:8.0
    container_name: laravel-db
    volumes:
      - ./sql-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=laravel
      - MYSQL_USER=user1
      - MYSQL_PASSWORD=pass1
    ports:
      - 3306:3306
    networks:
    - lan-laravel
############################ phpMyAdmin ############
  phpmyadmin:
    image: phpmyadmin
    links:
      - laravel-db
    networks:
      - lan-laravel
```

```
    ports:
      - 8080:80
    environment:
      PMA_ARBITRARY: 1
      PMA_HOST: laravel-db
      MYSQL_ROOT_PASSWORD: root

networks:
  lan-laravel:
```

container-app.sh

```
#!/bin/bash

docker exec -it laravel-app bash -c && find ./ -type d -exec chmod g+rwx {} \; \
                                    && chgrp -R www-data . \
                                    && umask 0020 \
                                    && /bin/bash"
```

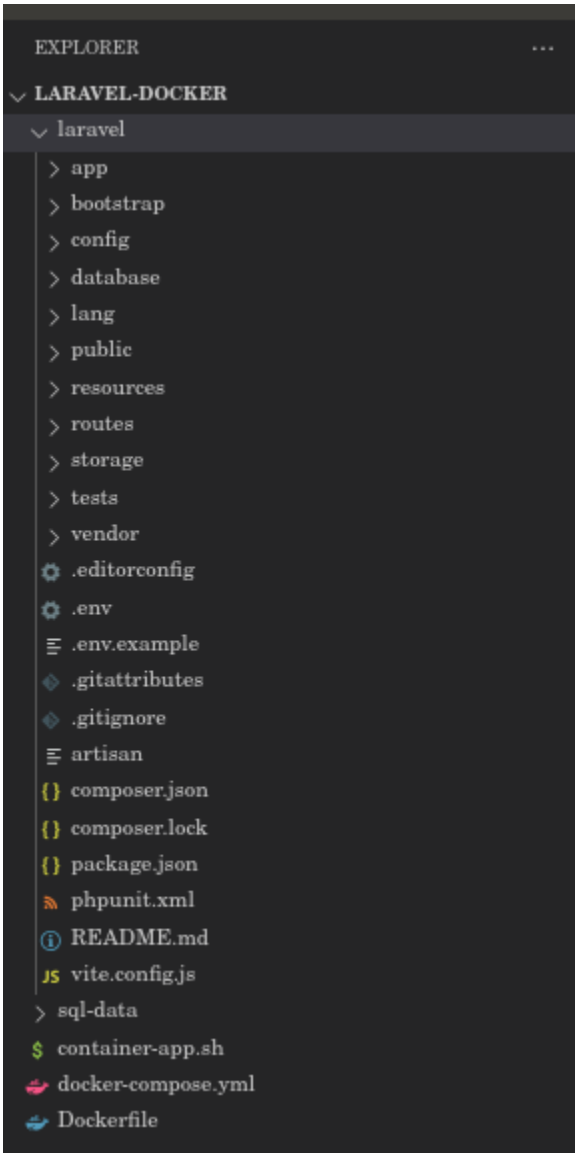


Ilustración 43: Estructura de la aplicación

Ahora construya el entorno y póngalo en funcionamiento. También instale dependencias de composer, así como algunos comandos artisan.

Use el comando

```
sudo docker-compose build && sudo docker-compose up -d
```

```
ubuntu@pp:~/Desktop/Laravel-Docker$ sudo docker-compose build && sudo docker-compose up
[sudo] password for ubuntu:
Sorry, try again.
[sudo] password for ubuntu:
laravel-db uses an image, skipping
phpmyadmin uses an image, skipping
Building laravel-app
Sending build context to Docker daemon  209.4kB
Step 1/12 : FROM php:8.0-apache
 ---> 0043a9962c76
Step 2/12 : RUN apt-get update
 ---> Running in fb9bd5b45679
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main amd64 Packages [8184 kB]
Get:5 http://deb.debian.org/debian-security bullseye-security/main amd64 Packages [190 kB]
Get:6 http://deb.debian.org/debian bullseye-updates/main amd64 Packages [6340 B]
Fetched 8588 kB in 9s (977 kB/s)
Reading package lists...
Removing intermediate container fb9bd5b45679
 ---> eb220663bed3
```

Ilustración 44:Construyendo la imagen Laravel y ejecutándola

Una vez que todos los contenedores estén en funcionamiento, puede verificarlos mediante

```
sudo docker-compose ps
```

```
ubuntu@pp:~/Desktop/Laravel-Docker$ sudo docker-compose ps
[sudo] password for ubuntu:
          Name                        Command               State                         Ports
-------------------------------------------------------------------------------------------------------------------------
laravel-app                  docker-php-entrypoint apac ...   Up      0.0.0.0:8000->80/tcp,:::8000->80/tcp
laravel-db                   docker-entrypoint.sh mysqld      Up      0.0.0.0:3306->3306/tcp,:::3306->3306/tcp, 33060/tcp
laravel-docker_phpmyadmin_1  /docker-entrypoint.sh apac ...   Up      0.0.0.0:80->80/tcp,:::80->80/tcp
```

Ilustración 45:Observamos los contenedores que están corriendo

Necesitará agregar unos permisos y actualizar composer y artisan para ello ejecute el script **container-app.sh** y ejecutar los comandos **composer install y php artisan key:generate**

Nota: Para acceder al contenedor laravel-app siempre utilizar el container-app.sh

```
./container-app.sh
```

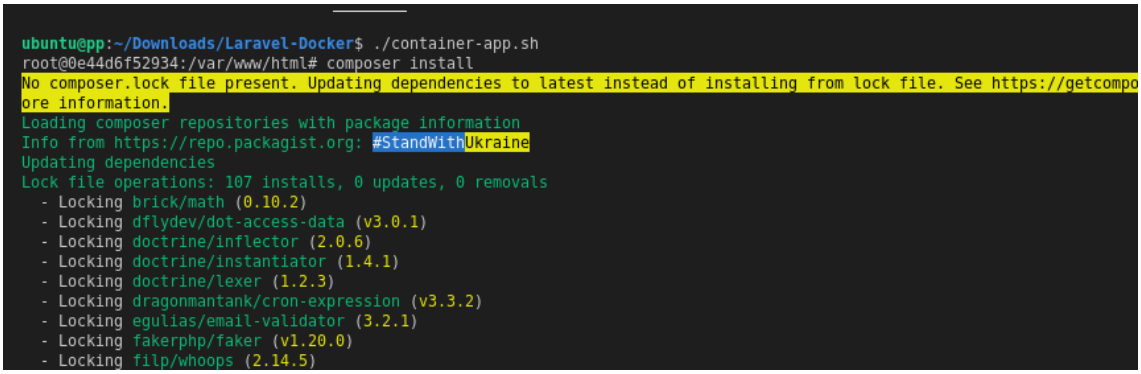


Ilustración 46: Instalación de Composer

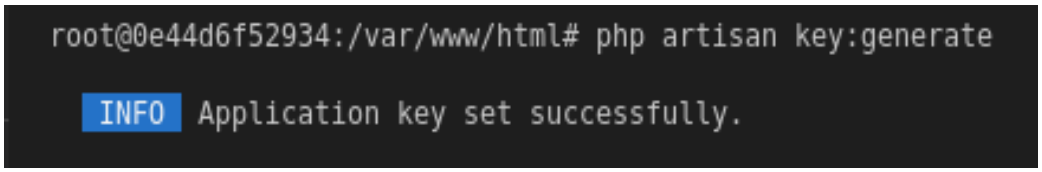


Ilustración 47: Actualizar la clave de la aplicación

Abrir en el navegador dos pestañas en una escriba: http://localhost:8000/ y en la otra http://localhost:8080/

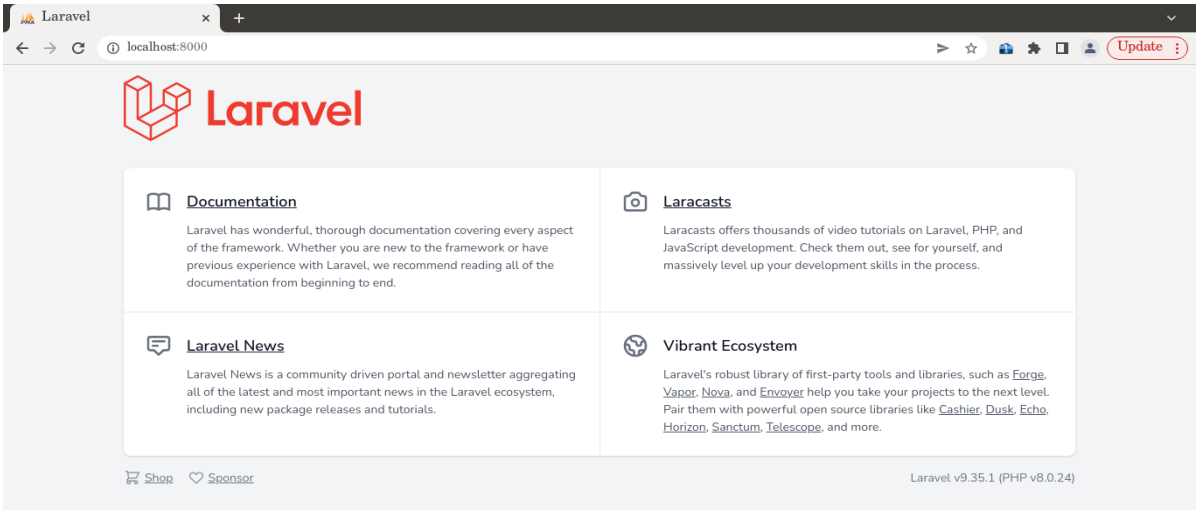


Ilustración 48: Se observa la página de inicio de Laravel

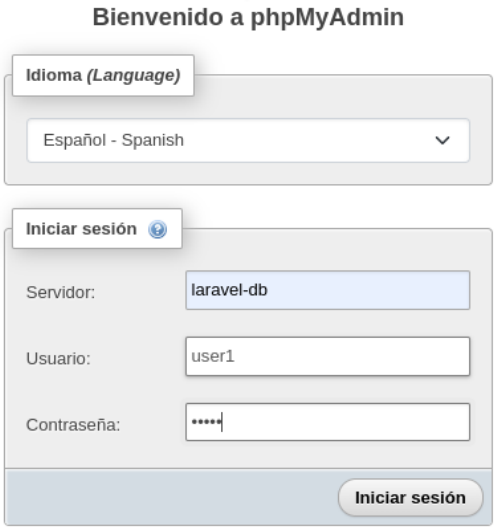


Ilustración 49: Inicio de sesión de phpMyAdmin

Para comprobación del funcionamiento del entorno se instala livewire para generar un CRUD. El Scanffold para almacenar registro de profesor la información se guardará en el contenedor externo de MYSQL

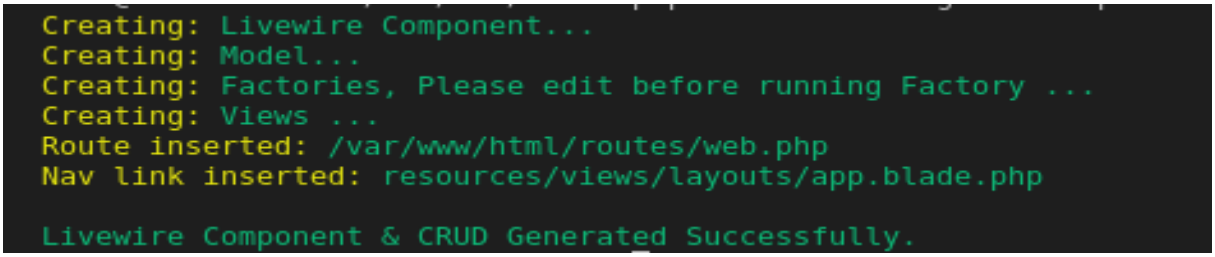


Ilustración 50: Directorio creado por el scanffold

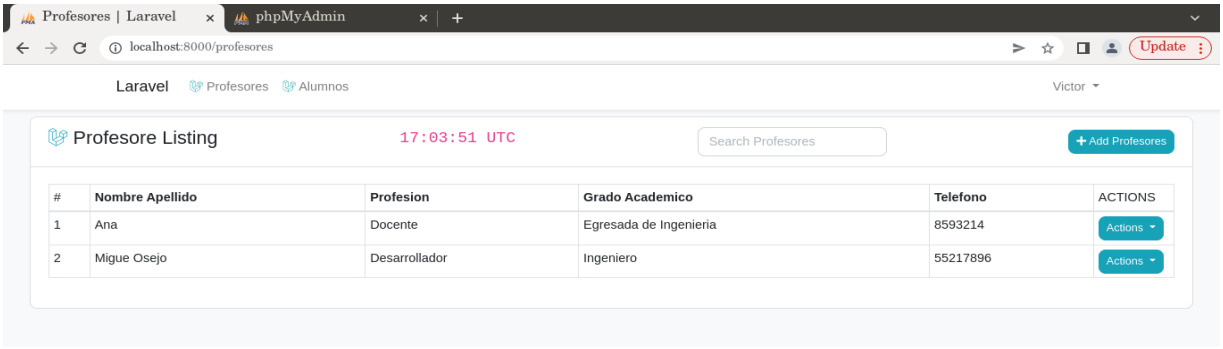


Ilustración 51: Datos agregados

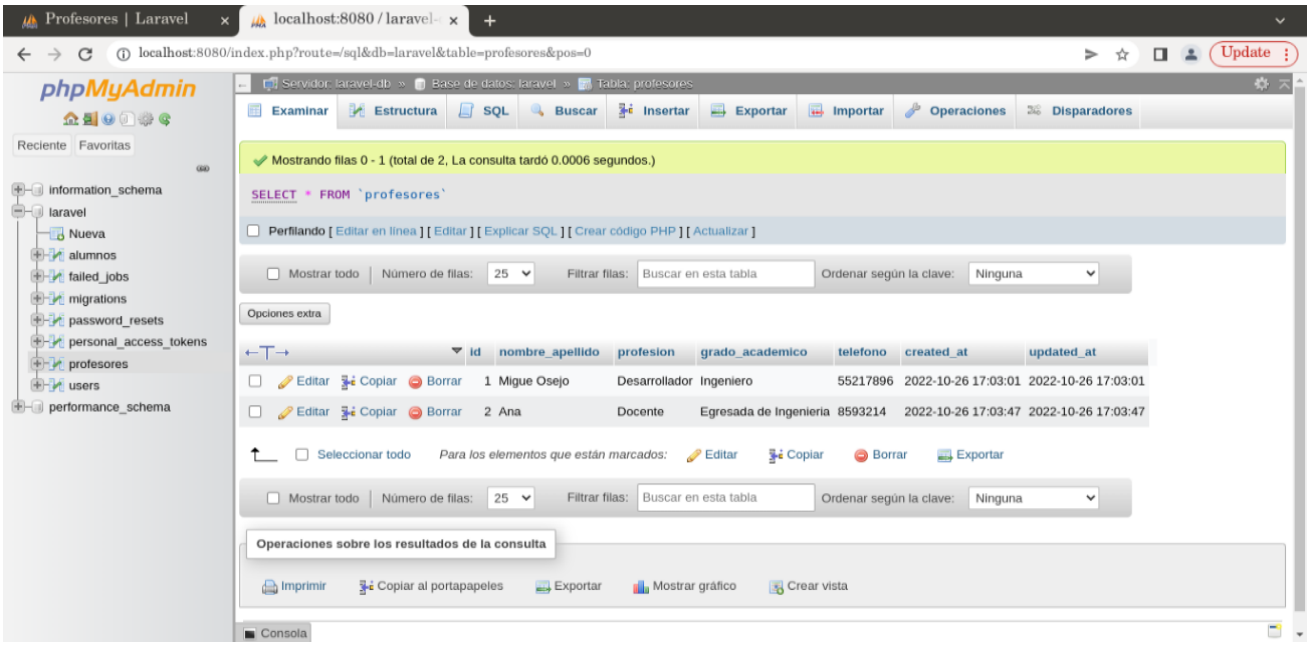


Ilustración 52: Datos en PhpMyAdmin

5.8 Entorno de desarrollo para ASP .NET

ASP.NET es un framework de desarrollo web del lado del servidor creado por Microsoft y altamente desarrollado. Se utiliza para crear páginas web dinámicas, aplicaciones web y servicios basados en web. Después de décadas de desarrollo, el framework sigue existiendo bajo el nombre de ASP.NET Core.

Esta propuesta de entorno de desarrollo para ASP .NET se utilizarán los siguientes sistemas o softwares:

mcr.microsoft.com/dotnet/sdk: 6.0.300
MySQL: 5.7
phpMyAdmin: latest

Los recursos para este entorno estarán disponibles en el siguiente enlace:

https://estunanleonedu-my.sharepoint.com/:u:/g/personal/victor_gonzalez15_est_unanleon_edu_ni/EewCsLW58FZCqcEAUP8eF98BrOdidA1feP6YsPXvmEu1-Q?e=IOVxNj

Esta carpeta contiene el archivo docker-compose.yml, junto a las dependencias y archivos de una app creada en ASP .NET

docker-compose.yml

```
version: "3.7"
services:
################### App ###############
   web:
      image: mcr.microsoft.com/dotnet/sdk:6.0.300
      container_name: App-net
      depends_on:
        - db-asp
      links:
        - db-asp
      ports:
```

```
      - "80:80"
      - "443:443"
    volumes:
      - ./:/MvcTest
    working_dir: /MvcTest/
    command: >
      bash -c " dotnet dev-certs https --clean
      && dotnet dev-certs https -t
      && tail -F anything"
    networks:
      - asp-networks
###################### Servicio de Base de datos ##############
  db-asp:
    image: mysql:5.7
    container_name: db-NET
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: ASP-net
      MYSQL_USER: usuario1
      MYSQL_PASSWORD: user123
    volumes:
      - ./mysql-data:/var/lib/mysql
    ports:
      - 3306:3306
    networks:
     - asp-networks
############################### phpMyadmin ##################
  phpmyadmin:
    container_name: Asp-phpmyadmin
    image: phpmyadmin
    links:
     - db-asp
    networks:
     - asp-networks
    ports:
     - 8080:80
    environment:
      PMA_ARBITRARY: 1
      PMA_HOST: db-asp
      MYSQL_ROOT_PASSWORD: root
```

```
networks:
  asp-networks:
```

app-net.sh

```
#! /bin/bash
sudo docker exec it App-net bash -c “umask 0020 \
                                        && /bin/bash“
```

Ejecute el comando **sudo docker-compose up -d** en su terminal (antes verifique que tenga conexión a internet)

Verificar si los contenedores están ejecutándose con **sudo docker compose ps**

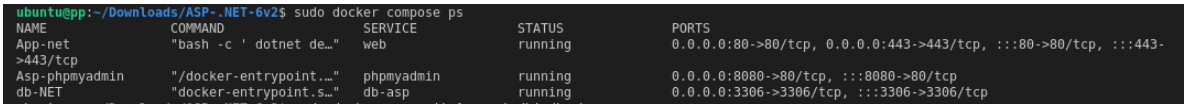


Ilustración 53: Contenedores en ejecución

Entrar al contenedor utilizando el script **app-net.sh**

Nota: Para acceder al contenedor App-net siempre utilizar el script app-net.sh

```
./app-net.sh
```

Compilar el proyecto

```
dotnet run
```

Ilustración 54: Salida del dotnet run

Ir a http://localhost:80/

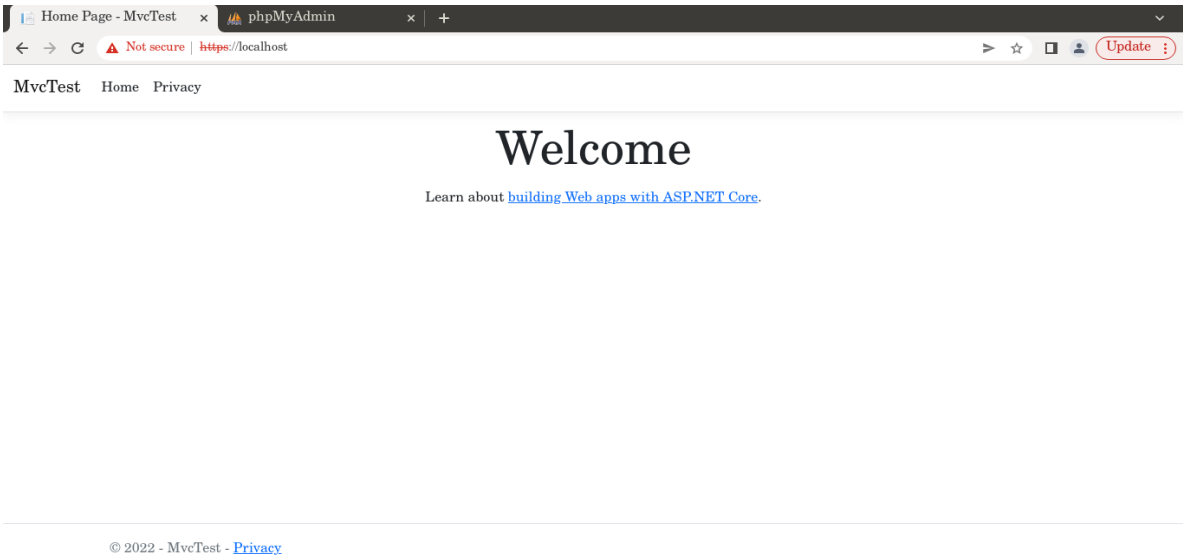


Ilustración 55: página de inicio de la aplicación

Ir a http://localhost:8080/

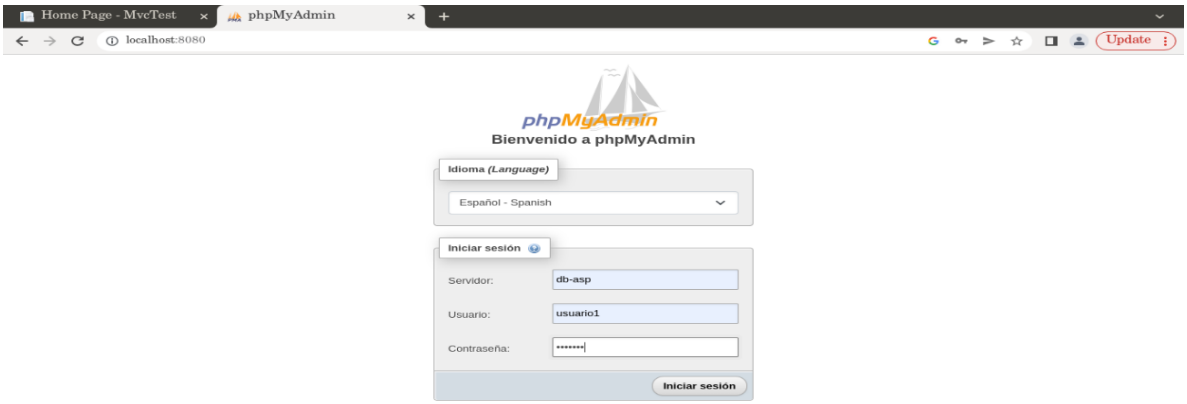


Ilustración 56: Inicio de sesión de phpMyAdmin

Para demostración del entorno de desarrollo se generó un scanffold que guardara registro en una base de datos

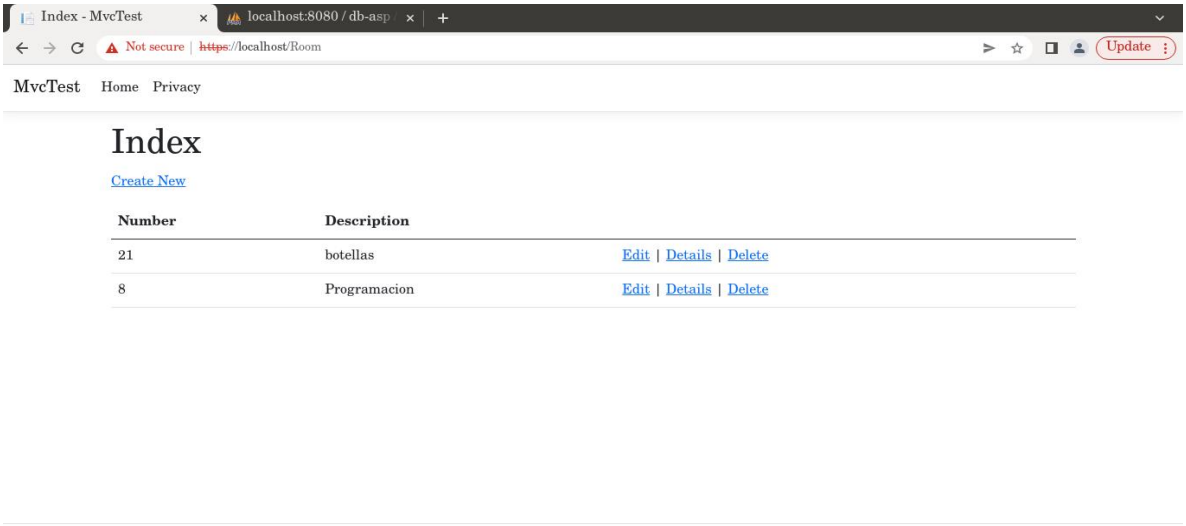


Ilustración 57: Introducción de datos

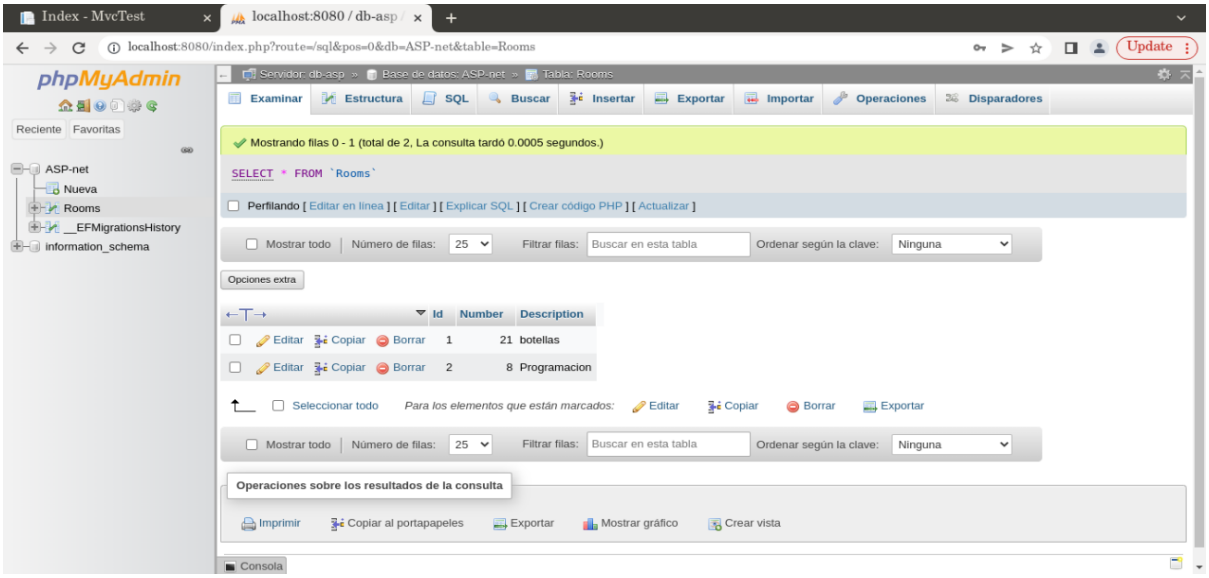


Ilustración 58: Datos guardados

5.9 Entorno de desarrollo para Ruby On Rails

Ruby on Rails, también conocido como RoR o Rails, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma del patrón Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración.

Esta propuesta de entorno de desarrollo para Ruby on Rails permitirá crear aplicaciones web, en este contenedor se utilizarán los siguientes sistemas o softwares:

MySQL:5.7
PhpMyAdmin: latest
Ruby:2.6.
Rails:5.0.7.

Descargar la carpeta Ruby-on-Rails el recurso está disponible en el siguiente enlace:

https://estunanleonedu-my.sharepoint.com/:u:/g/personal/victor_gonzalez15_est_unanleon_edu_ni/ETVG_c_sWiBHo2ntI9MCrNgBRDRt8tDXdndKXa3n17cUpA?e=084yFd

Este directorio contiene los archivos docker-compose.yml, Dockerfile y app-ruby.sh junto a las dependencias y archivos de una app creada en Ruby on Rails

docker-compose.yml

```
version: '3.7'
services:
######################### Servidor de Base de datos ##############
    db:
        image: mysql:5.7
        container_name: Ruby-db
        environment:
            MYSQL_ROOT_PASSWORD: root
            MYSQL_DATABASE: Ruby
            MYSQL_USER: user
            MYSQL_PASSWORD: password
        ports:
            - "3306:3306"
        volumes:
            - ./sql-data:/var/lib/mysql
        networks:
            - lan-Rails
############################ App de Ruby on Rails ##############
    app:
        build: .
        container_name: app-ruby
        command: bundle exec rails s -p 8000 -b '0.0.0.0'
        volumes:
            - ".:/app"
        ports:
            - "8000:8000"
        links:
            - db
        networks:
            - lan-Rails
        environment:
            DB_NAME: Ruby
            DB_USER: user
            DB_PASSWORD: password
            DB_HOST: db
############################ phpMyAdmin #################
    phpmyadmin:
        image: phpmyadmin
```

```
        container_name: Ruby-phpmyadmin
        links:
            - db
        networks:
            - lan-Rails
        ports:
         - 8080:80
        environment:
            PMA_ARBITRARY: 1
            PMA_HOST: db
            MYSQL_ROOT_PASSWORD: root
networks:
    lan-Rails:
```

Dockerfile

```
FROM ruby:2.5.1
RUN apt-get update -qq && apt-get install -y build-essential libpq-dev nodejs
RUN mkdir /app
WORKDIR /app
ADD Gemfile /app/Gemfile
ADD Gemfile.lock /app/Gemfile.lock
RUN bundle install
ADD . /app
```

App-ruby.sh

```
sudo docker exec -it app-ruby bash -c " umask 0020 \
                                        && /bin/bash "
```

Accede a la carpeta previamente descargada y abre los archivos con el editor de texto favorito.

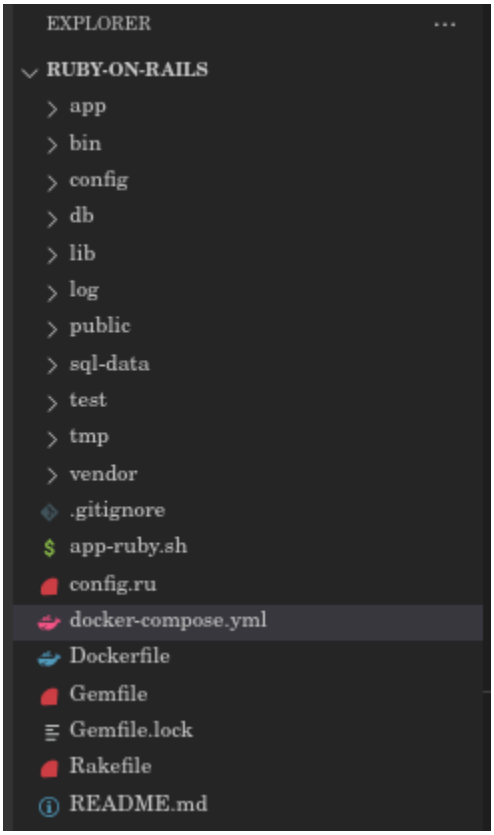


Ilustración 59: Estructura del proyecto Ruby on Rails

Ejecute este comando en su terminal (antes verifique que tenga conexión a internet)

```
sudo docker-compose up -d "
```

Ir a http://localhost:8000/ verá el logotipo de Rails

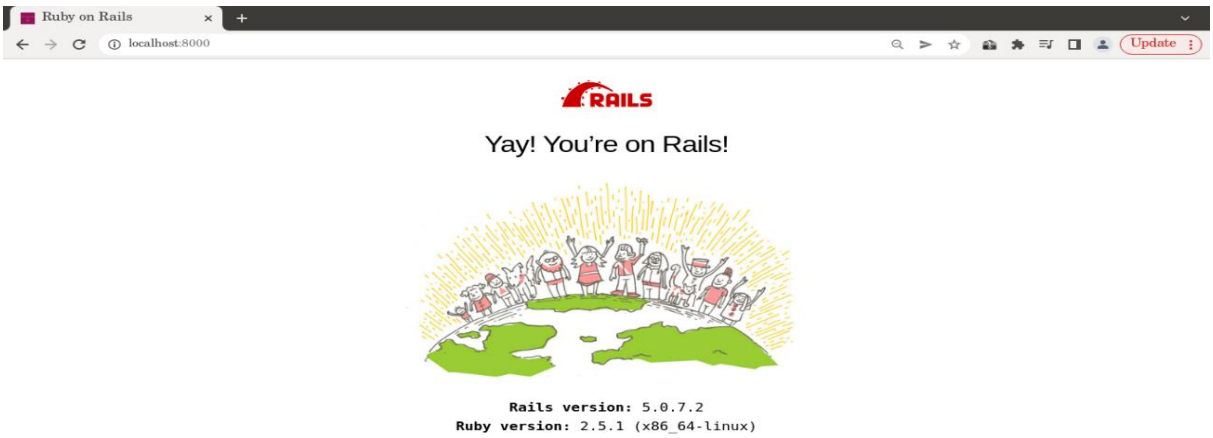


Ilustración 60: Logo de Rails

Tambien acceder a phpMyAdmin

http://localhost:8080/

Terminado el proceso de construcción de los contenedores, estarán listos para utilizarse

Nota: Para acceder al contenedor app-ruby siempre utilizar el script app-ruby.sh

```
./app-ruby.sh
```

Para comprobar el funcionamiento del contenedor se creó un CRUD usando un Scaffolding que en Ruby on Rails se refiere a la generación automática de un conjunto de un modelo, vistas y un controlador que generalmente se usa para una sola tabla de base de datos. Le generará automáticamente un controlador, modelo y vistas listos para usar con una interfaz web.

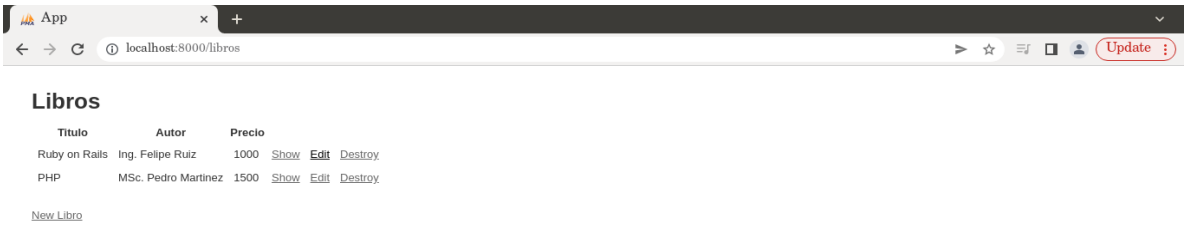


Ilustración 61: Datos agregados a la base de datos

Estos datos también se ven reflejados en phpMyAdmin

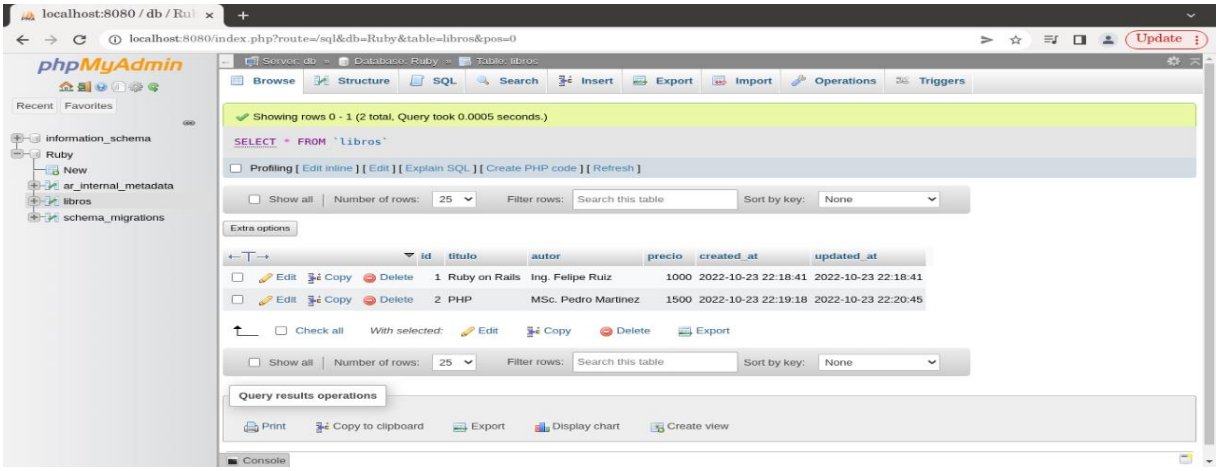


Ilustración 62: Datos vistos desde phpMyAdmin

6 Conclusiones

Al concluir este trabajo monográfico, considerando que se ha logrado llevar a cabo los objetivos propuestos se puede afirmar que con la documentación aportada se ha obtenido conceptos básicos para instruir y guiar a los estudiantes en el uso de los contenedores.

Con las guías elaboradas y resueltas contribuyen a una mejor comprensión para los estudiantes acerca de esta tecnología, explicando paso a paso los comandos básicos hasta como elaborar una aplicación multi contenedor.

Los ambientes de desarrollo creado con Docker Compose podrán ser usadas como material de apoyo para la realización de tareas asignadas en algunos componentes logrando también implementarse en desarrollo de aplicaciones web.

Finalmente, con los elementos antes mencionados se puede afirmar que mediante la realizaron de pruebas se logró tener una visión de usuario final, validando la usabilidad de los entornos y comprobando que lo planteado había sido cumplido.

7 Recomendaciones

Una vez finalizado este trabajo se propone una serie de recomendaciones:

- Usar este documento como guía y manejo en la creación de cada entorno propuesto cuidando las anotaciones de comandos para la creación y ejecución de estos de manera exitosa.
- Si bien los contenedores funcionan muy bien en muchos escenarios las practicas se recomienda utilizarlos en distribuciones estables de Linux
- Hacer pruebas de funcionamiento en el sistema Windows para extender su uso y de haber problemas de compilación o depuración intentar corregirlos para usarse en este sistema.

8 Referencias Bibliográficas

Abel, H. T. (2017). *Arquitectura para el desarrollo e implementacion de servicios web.* Puno, Peru: Universidad Nacional del Altiplano.

AWS Editorial Team. (2021). *aws*. Recuperado el 29 de Noviembre de 2021, de https://aws.amazon.com/es/docker/

CHANDLER HARRIS, J. C. (s.f.). *atlasisan.com*. Obtenido de atlassian.com: https://www.atlassian.com/es/microservices/microservices-architecture/kubernetes-vs-docker#:~:text=Docker%20es%20una%20plataforma%20de,ejecuci%C3%B3n%20de%20contenedores%2C%20incluido%20Docker.

Desde Linux. (s.f.). *desdelinux.net*. Obtenido de desdelinux.net: blog.desdelinux.net/docker-hub-aprendiendo-tecnologia-docker/

Edwin Lacayo, J. M. (2020). *Desarrollo de una aplicación web para la representación gráfica de topologías de red creadas entre contenedores Docker.* León.

Equipo Ekon. (1 de Octubre de 2020). *www.ekon.es*. Obtenido de https://www.ekon.es/blog/entornos-desarrollo-software/

Erdwan Mazariego, L. M. (2022). *Implementación de contenedores Docker como herramienta de virtualización liviana .* León.

Genesis Delgado, H. M. (2017). *Distribución Linux Ubuntu 16.04 que incluya paquetes de software pre-instalados utilizados en las asignaturas de las áreas de Redes y Programación de las carreras que ofrece el Departamento de Computación de la UNAN-León.* León.

I.E.S CELIA VIÑAS CIBERSEGURIDAD. (19 de Mayo de 2021). *iescelia.org*. Obtenido de iescelia.org: https://iescelia.org/ciberseguridad/serie-docker-estructura-de-configuracion-de-un-archivo-docker-compose/

Ingnacio Moreno Hojas, StatPlans. (s.f.). *Stat Developer*. Obtenido de Stat Developer: https://www.statdeveloper.com/creacion-de-imagenes-para-docker/

INSOR. (2017). *GUÍA PARA SEPARACIÓN DE AMBIENTES* (Vol. 01). Bogota, Colombia. Recuperado el 29 de Noviembre de 2021

KeepCoding. (s.f.). *KeepCoding*. Obtenido de KeepCoding: https://keepcoding.io/blog/que-es-dockerfile/#:~:text=Un%20Dockerfile%20es%20un%20archivo,creaci%C3%B3n%20de%20una%20nueva%20imagen.

KEEPCODING Tech School. (s.f.). *KEEPCODING*. Obtenido de KEEPCODING: https://keepcoding.io/blog/que-es-docker-compose/

KeepCoding, R. (28 de Abril de 2022). *keepcoding Tech School*. Obtenido de https://keepcoding.io/blog/que-es-una-imagen-en-docker/

Leaf Group. (1 de Febrero de 2018). *Cuida tu Dinero*. (M. Zeske, Editor) Obtenido de https://www.cuidatudinero.com/13098460/definicion-de-un-ambiente-de-prueba

NetApp. (2021). *NetApp*. Recuperado el 29 de Noviembre de 2021, de https://www.netapp.com/es/devops-solutions/what-are-containers/

OCL. (s.f.). *oracle.com*. Obtenido de oracle.com: https://www.oracle.com/mx/cloud/cloud-native/container-engine-kubernetes/what-is-kubernetes/#link9

Red Hat,Inc. (27 de marzo de 2020). *redhat*. Obtenido de redhat: https://www.redhat.com/es/topics/containers/what-is-kubernetes#:~:text=Kubernetes%20(tambi%C3%A9n%20conocida%20como%20k8s,que%20se%20alojan%20en%20ellos.

Red Hat,Inc. (18 de junio de 2021). *redhat*. Obtenido de redhat: https://www.redhat.com/es/topics/automation/what-is-yaml

Tirado Zatarain, L. M. (2017). *Implementación de Docker en la.* Mazatlan: SEP.

Anexos

Cronograma de Actividades