Universidad Latina de Costa Rica

BIT-28 Sistemas Operativos II

Laboratorio #4

**Contenedores**

**Instrucciones**

Trabaje individualmente. Siga los siguientes puntos paso a paso. Debe trabajar en Linux (Ubuntu). Debe entregar un reporte al final de la clase en el Moodle.

Para el presente laboratorio debe contar con al menos una máquina Ubuntu 18.

**Objetivos**

¿Qué son los contenedores?

Comprender las diferencias entre Docker y LXD

Instalar Docker

Administrar contenedores Docker

Administrar Docker con interfaz gráfica Portainer

**¿Qué son los contenedores?**

Los diferentes tipos de virtualización como la emulación (VirtualBox), para virtualización (Xen), o virtualización completa (VMWare, KVM) nos permite ejecutar múltiples servidores virtuales en un solo servidor físico. Asignamos CPU, RAM y espacio en disco a estas máquinas virtuales, y se ejecutan como si fueran un servidor real. Sin embargo, también hay debilidades con las máquinas virtuales. Quizás lo más obvio es la asignación de recursos que asigna a una VM, que probablemente se están desperdiciando. Por ejemplo, quizás se hayan asignado 512 MB de RAM a una máquina virtual. ¿Qué sucede si la aplicación rara vez usa más de 100 MB de RAM? Eso significa que la mayoría de las veces, 412 MB de RAM, que de otro modo podrían usarse para un propósito útil simplemente está inactivo. Lo mismo puede decirse de la CPU también. Hoy en día, las soluciones de máquinas virtuales tienen formas de compartir recursos no utilizados, pero efectivamente, la eficiencia de los recursos es una debilidad natural de la plataforma.

Mientras que las máquinas virtuales suelen tener una CPU dedicada, los contenedores comparten la CPU con el host. Las máquinas virtuales generalmente también tienen su propio núcleo, pero los contenedores comparten el núcleo del host. Sin embargo, los contenedores están segregados. Al igual que una máquina virtual no puede acceder al sistema de archivos del host, un contenedor tampoco puede hacerlo (a menos que se configure explícitamente para eso). Quizá, una manera de pensar qué es un contenedor, es verlo como un sistema de archivos. El contenedor en sí contiene una estructura de archivos que coincide con la de la distribución en la que se basa. Un contenedor basado en Ubuntu Server, por ejemplo, tendrá el mismo diseño de sistema de archivos que una instalación real de Ubuntu Server. Imagine copiar todos los archivos y carpetas de una instalación de Ubuntu, ponerlo todo en un único directorio segregado y tener los contenidos binarios del sistema de archivos ejecutados como un programa (sin tener que levantar otro sistema operativo). Lo que implica que un contenedor usaría solo los recursos que necesitaba.

La principal diferencia con los contenedores es que cada contenedor generalmente hace una cosa. Por ejemplo, quizás un contenedor contiene un sitio web alojado o contiene una sola aplicación. Las máquinas virtuales a menudo se crean para realizar muchas tareas, como un servidor web que aloja diez sitios web.

¿Cuándo deberías usar contenedores? Sugiero que considere los contenedores cada vez que ejecute una aplicación web o algún tipo de servicio, y se beneficiaría de compartir recursos en lugar de dedicar memoria o CPU. No todas las aplicaciones se ejecutarán bien en un contenedor, pero al menos es algo a considerar.

**Diferencias entre Docker y LXD**

LXD (pronunciado Lex-D) encuentra sus raíces en LXC. LXC es la abreviatura de Linux Containers (pronunciado Lex-C) y es otra implementación de contenedores, muy similar a Docker. Esta tecnología utiliza la función de grupos de control (cgroups) del kernel de Linux, que aísla los procesos y puede segregarlos entre sí. Esto mejora la seguridad, ya que los procesos no deberían poder leer datos de otros procesos a menos que haya una buena razón para hacerlo. LXC lleva el concepto de segregación aún más lejos, al crear una implementación de virtualización basada únicamente en la ejecución de aplicaciones en un entorno aislado que coincide con el entorno de un sistema operativo. Puede ejecutar contenedores LXC en casi todas las distribuciones de Linux disponibles en la actualidad.

LXD es, al menos, en parte, específico de Ubuntu. Eso no es completamente cierto en estos días, ya que la tecnología también ha sido diseñada para funcionar en otras plataformas. Esto se debe al hecho de que el software LXD se distribuye a través de paquetes snap, por lo que cualquier distribución de Linux que pueda instalar paquetes snap debería poder instalar LXD. Creado inicialmente por Canonical (la compañía detrás de Ubuntu), LXD mejora LXC y le brinda características adicionales que de otro modo no tendría, como instantáneas y migración. LXD no reemplaza a LXC; en realidad lo utiliza para proporcionar su tecnología base (incluso utiliza el comando lxc para administrarlo). Quizás la mejor manera de pensar en LXD es LXC como una capa de administración adicional en la parte superior, que agrega características adicionales.

¿En qué se diferencia LXD / LXC de Docker? La principal diferencia es que, si bien ambas son soluciones de contenedor y resuelven el mismo objetivo de una manera muy similar, LXD es más similar a una máquina virtual real, mientras que Docker se esfuerza más por diferenciarse de eso. En comparación, los contenedores Docker son transaccionales (cada tarea se ejecuta en una capa separada) y generalmente tiene un comando ENTRYPOINT que se ejecuta dentro del contenedor. Esencialmente, LXC tiene un sistema de archivos al que puede acceder directamente desde el sistema operativo host, y tiene un enfoque más simple para los contenedores. Puede pensar en LXC como una forma de contenedor de máquina que emula estrechamente una máquina virtual, y un Docker como un contenedor de aplicación que proporciona la base necesaria para ejecutar una aplicación. Independientemente de estas diferencias, ambas tecnologías se pueden usar de la misma manera y brindan soporte para casos de uso idénticos.

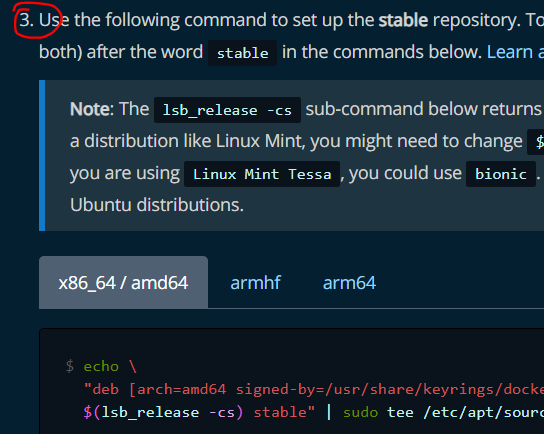
¿Cuándo debería usar Docker y cuándo debería usar LXD? Se recomienda aprender ambos, ya que no son demasiado difíciles de aprender. Docker es más una herramienta de uso general. Puede ejecutar contenedores Docker en Linux, macOS e incluso Windows. Es una buena opción si desea crear un contenedor que se ejecute en todas partes. LXD es generalmente el mejor para entornos Linux, aunque Docker también funciona muy bien en Linux. El sistema operativo en el que está ejecutando su solución de contenedor es de poca importancia hoy en día, ya que la mayoría de las personas usan un servicio de contenedor para ejecutar contenedores en lugar de un servidor real que usted administra. En el futuro, si se empeña en la contenedorización, es posible que renuncie por completo al sistema operativo y simplemente lo ejecute en un servicio como el **Elastic Container Service** (ECS) de Amazon o Kubernetes.

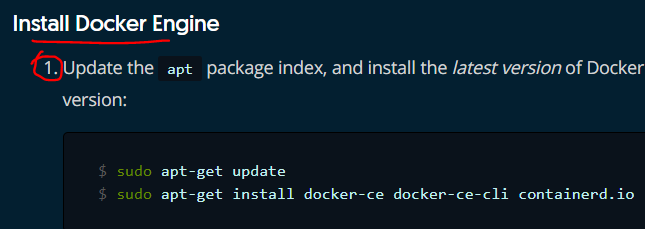
Otro beneficio de Docker es el **Docker Hub**, que puede usar para descargar contenedores que otros han hecho o incluso cargar el suyo para que otros lo usen. El beneficio aquí es que si alguien ya ha resuelto el objetivo que está tratando de lograr, puede beneficiarse de su trabajo en lugar de comenzar desde cero, y también puede beneficiarse de su trabajo. Esto ahorra tiempo y, a menudo, es mejor que crear una solución desde cero. Sin embargo, siempre asegúrese de revisar los recursos de terceros antes de utilizarlos en su organización.

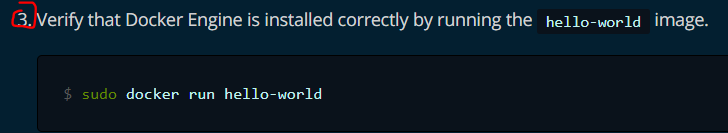
**Instalar Docker en Ubuntu 20.04**

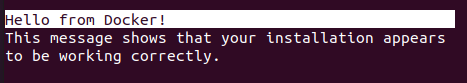
Seguir los pasos en <https://docs.docker.com/engine/install/ubuntu/>

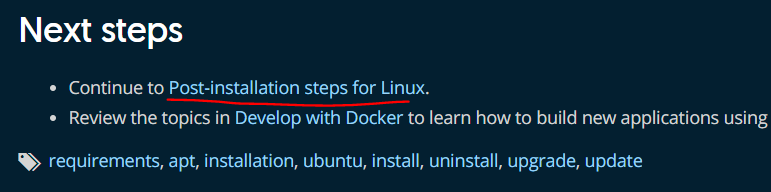


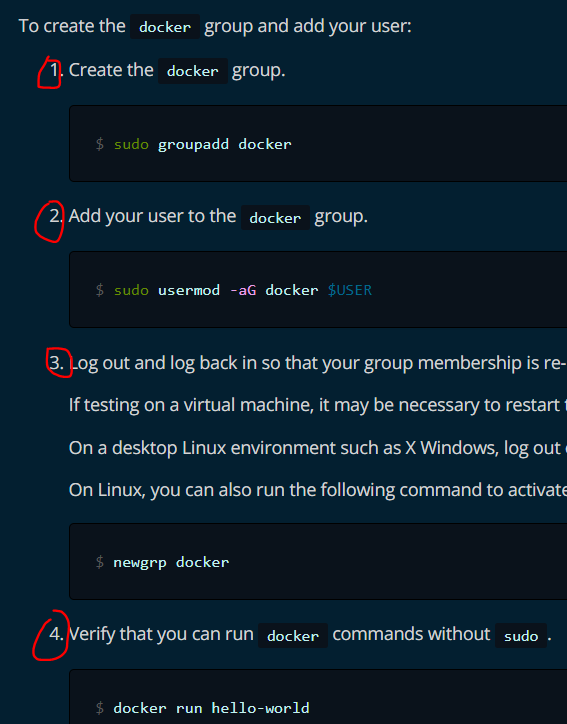


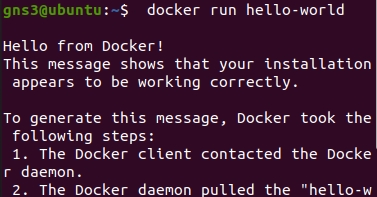












**Utilizar Docker**

En resumen, un contenedor no es más que un proceso en ejecución, con algunas características de encapsulación adicionales aplicadas para mantenerlo aislado del host y de otros contenedores. Uno de los aspectos más importantes del aislamiento de contenedores es que cada contenedor interactúa con su propio sistema de archivos privado; este sistema de archivos es proporcionado por una imagen Docker. Una imagen provee todo lo necesario para ejecutar una aplicación, como código o binarios, runtimes, dependencias, etc. Una imagen podría proporcionar un ambiente Debian o un servidor LAMP, por ejemplo.

Muchos proyectos mantienen una imagen oficial y también los usuarios aportan sus propias imágenes. El repositorio o registro oficial es [Docker Hub](https://hub.docker.com), el cual será utilizado a menos que se especifique otro diferente. También es posible crear una imagen propia tomando una imagen existente como base. Esta se define en un archivo llamado Dockerfile.

Las imágenes están construidas por varias capas que pueden ser compartidas, por ejemplo varias imágenes pueden estar basadas en Debian por lo que la respectiva capa será descargada una sola vez y utilizada por todas las imágenes que lo requieran.

Sobre la imagen se ejecuta el contenedor y es aquí donde se escriben los cambios, por ejemplo los archivos que se suban a un Nextcloud o los datos que se inserten en un MariaDB. Si se borra el contenedor, todos los datos contenidos se borrarán también. Para resguardar los datos se utilizan volúmenes persistentes que se montan dentro del contenedor pero cuyo sistema de archivos está fuera de este.

**Ejemplo servidor web**

Algo muy común es ejecutar varios contenedores, uno con un servidor web y otro con una base de datos; donde se expongan uno o más puertos para acceder servicios desde otros hosts; y donde se definan volúmenes persistentes que se monten en el contenedor, esto para evitar que se pierdan datos en caso de eliminar un contenedor.

El primer paso es definir una red exclusiva para los contenedores. Si no se creara una red, los contenedores utilizarían la red predeterminada, llamada bridge, sin embargo, esta tiene menos características y no ofrece un aislamiento seguro.

docker network create misitio-net



El servidor de base de datos se ejecutará en su propio contenedor, aparte del servidor web. De la siguiente manera:

docker run -d --name **mysql11** \

--network misitio-net \

-e MYSQL\_ROOT\_PASSWORD=carlos.123 \

-e MYSQL\_USER=carlos \

-e MYSQL\_PASSWORD=carlos.123 \

-e MYSQL\_DATABASE=misitiodb \

-v websitedbvolume:/var/lib/mysql \

mariadb:latest

Donde:

-d indica que el contenedor se ejecutará en segundo plano regresando el control al promp.

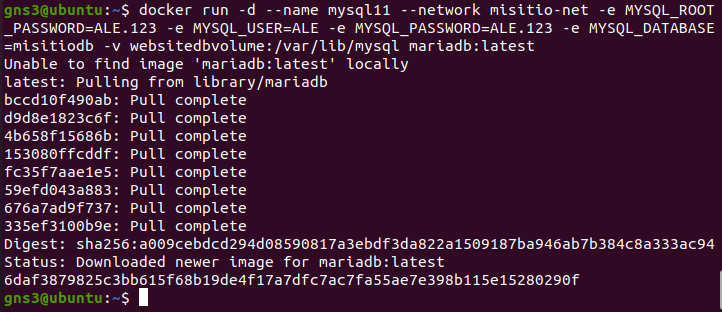
--name define un nombre. Si no se definiera, docker creará algún nombre.

--network indica a cuál red, previamente definida, debe conectarse. Puede conectarse a más de una red.

-e permite asignar valores a las variables de entorno definidas en el contenedor por los desarrolladores.

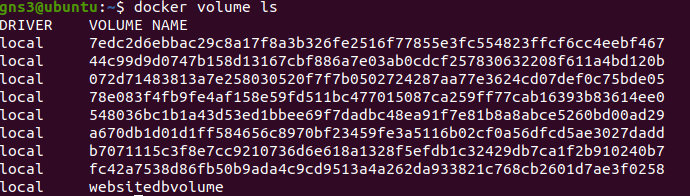
-v monta un volumen dentro del contenedor. En este caso se definió el volumen websitedbvolume, el cual es gestionado por docker.

mariadb:latest corresponde al nombre de la imagen y la etiqueta. Si la imagen no existiera, docker la descargará.

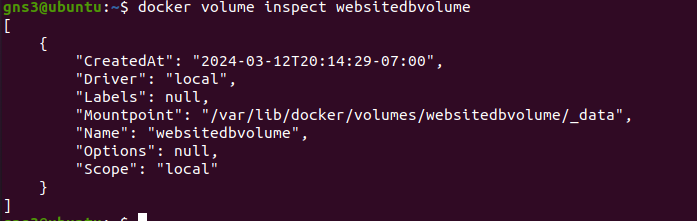


Puede confirmar la creación del volumen y la información del mismo, de la siguiente manera:

docker volume ls



docker volume inspect websitedbvolume



Las bases de datos del contenedor se guardarán aparte, en este volumen, en la ruta indicada en Mountpoint en la salida del segundo comando.

Antes de crear el contenedor para el servidor web, crearemos un directorio que usaremos como volumen, mostrando otra forma de gestionar los volúmenes en docker.

mkdir public\_html

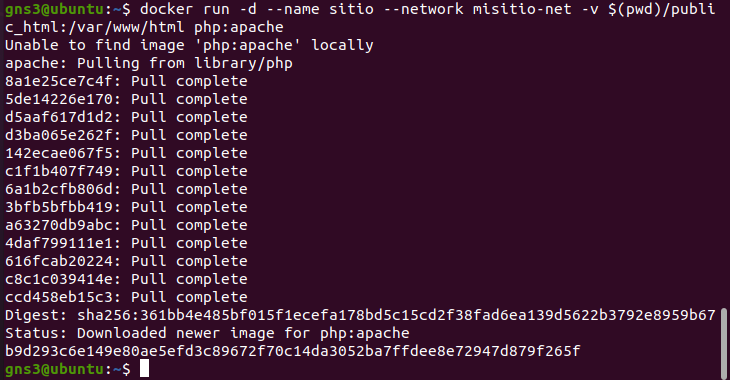
El contendedor del servidor web se ejecuta de la siguiente manera:

docker run -d --name **sitio** \

--network misitio-net \

-v $(pwd)/public\_html:/var/www/html \

php:apache



Donde:

-d indica que el contenedor se ejecutará en segundo plano regresando el control al promp.

--name define un nombre. Si no se definiera, docker creará algún nombre.

--network indica a cuál red, previamente definida, debe conectarse. Puede conectarse a más de una red.

-p indica que puerto se expondrá. A la izquierda se indica un puerto libre del host, y a la derecha el puerto a exponer del contenedor.

-v monta un volumen dentro del contenedor. En este caso se indicó la ruta a un directorio. $(pwd) indica la ruta actual, de manera que genere la dirección a una ruta absoluta.

php:apache corresponde al nombre de la imagen y la etiqueta. Si la imagen no existiera, docker la descargará.

Todo lo que se coloque en el directorio public\_html será publicado por el servidor web, pues dicho directorio está montado, en el contenedor, como /var/www/html, que es la ruta predeterminada del servicio web. Pruebe creando un archivo index.html o index.php.

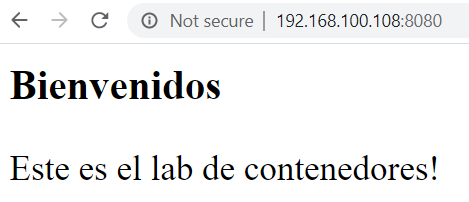
Docker ijnspect sitio y te despliegfa la info

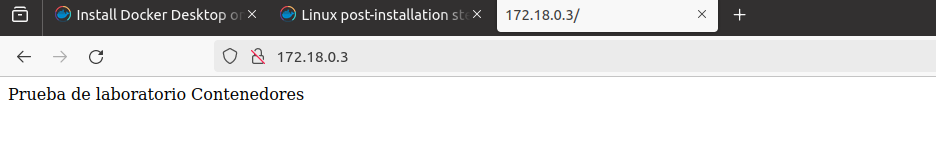
***Para ver el sitio web, abra un navegador web e ingrese a la dirección*** http://xx.xx.xx.xx:80

Donde:

xx.xx.xx.xx es la dirección IP o nombre de dominio de su servidor. Sustituya por localhost si está probando en su computadora.

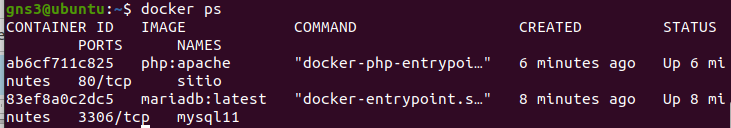
8080 es el puerto con el que se mapeo el puerto 80 del contenedor.





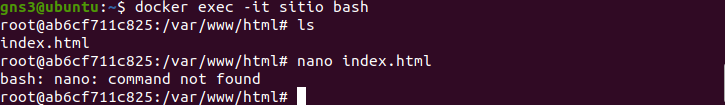
En este punto debería tener en ejecución dos contenedores, llamados misitiodb y sitio. Puede corroborarlo así:

docker ps



Si desea correr comandos en la terminal de un contenedor que esté en ejecución, puede hacerlo de la siguiente manera:

docker exec -it sitio bash



Donde:

-it indica que abra una sesión interactiva y que además proporcione una consola.

website corresponde al nombre del contenedor, el cual fue definido cuando se creó.

bash indica el programa a ejecutar.

Si desea eliminar los contenedores ejecute los siguientes pasos (Hacerlo para la siguiente parte).

docker stop sitio

docker rm sitio

docker stop misitio

docker rm misitio

El volumen del contenedor websitedb seguirá existiendo y almacenando los archivos de la o las bases de datos. Si desea borrarlo ejecute (Hacerlo para la siguiente parte):

docker volume rm websitedbvolume

De una manera similar, puede eliminar la red que se definió para dichos contenedores (Hacerlo para la siguiente parte).

docker network rm misitio-net

Por el momento conservare la configuración

**Docker Compose**

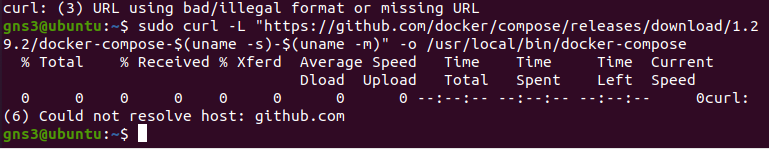
Docker Compose es una herramienta que permite definir uno o más contenedores Docker en un único archivo de configuración y ejecutarlo con un simple comando. Es una excelente alternativa para definir y correr aplicaciones complejas.

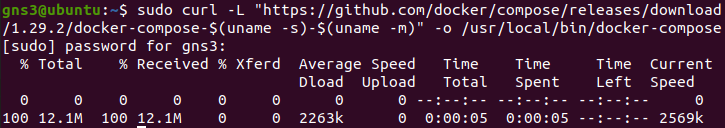
Básicamente se debe definir un archivo, en formato YAML, y guardarlo con el nombre docker-compose.yml .

**Instalación en Ubuntu**

Para instalar docker-composer V1, realizar lo siguiente:

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose





Y, le damos los permisos de ejecución:

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

Para corroborar que funciona, ejecute:

docker-compose –version



Nota:

\*\* Si desea instalar docker composer V2: <https://docs.docker.com/compose/cli-command/#install-on-linux>

\*\* Si desea eliminar docker-composerV1: sudo rm /usr/local/bin/docker-compose

**Ejemplo servidor web con un solo comando**

El ejemplo del servidor web, creamos un archivo docker-compose.yml:

version: '3'

services:

sitio:

image: php:apache

container\_name: sitio

ports:

- 8081:80

volumes:

- ./public\_html:/var/www/html

networks:

- misitio-net

misitiodb:

image: mariadb:latest

container\_name: misitiodb

volumes:

- websitedbvolume:/var/lib/mysql

environment:

MYSQL\_ROOT\_PASSWORD: "carlos.123"

MYSQL\_USER: 'carlos'

MYSQL\_PASSWORD: 'carlos.123'

MYSQL\_DATABASE: misitiodb

networks:

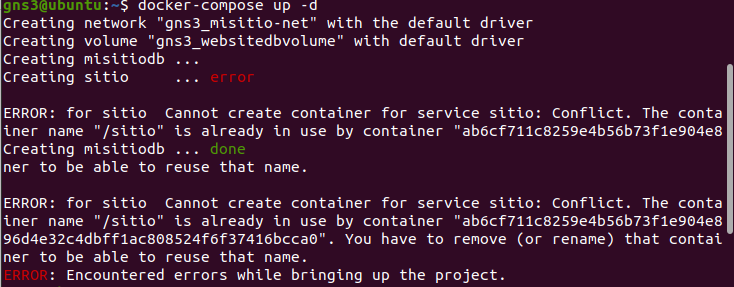
- misitio-net

networks:

misitio-net:

volumes:

websitedbvolume:



Consideraciones:

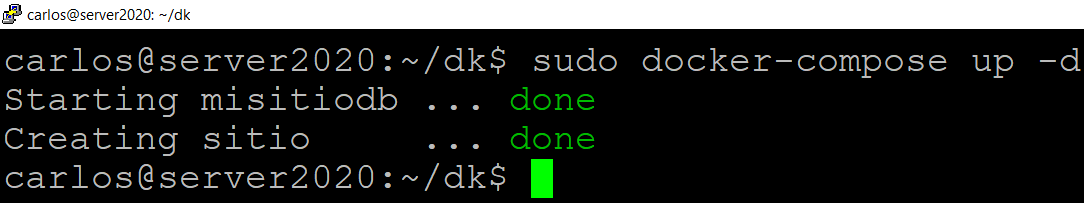
* Los servicios corresponden a cada uno de los contenedores.
* Si no se indica un nombre en container\_name, docker-compose le asignará un nombre.
* Las redes y volúmenes se deben declarar, en este caso al final del archivo.
* En este caso se utilizó el puerto 8081 del host, esto para evitar conflicto en caso de que todavía el ejemplo esté en ejecución.

Guarde los cambios y, como se había indicado, ejecute:

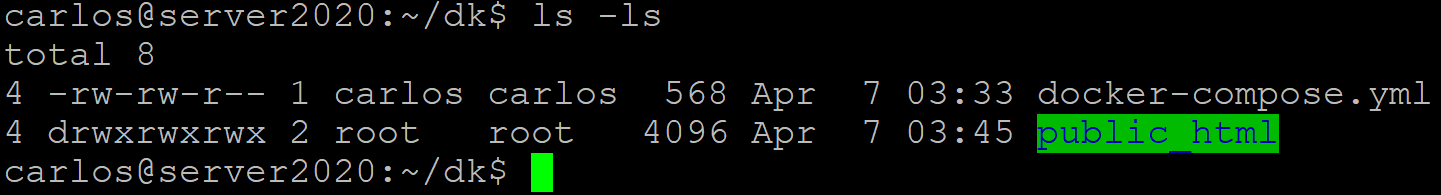
docker-compose up -d

Donde:

-d indica que el contenedor se ejecutará en segundo plano regresando el control al promp.



Observe que se creó un directorio **public\_html**, inserte en dicho directorio una página **index.html**

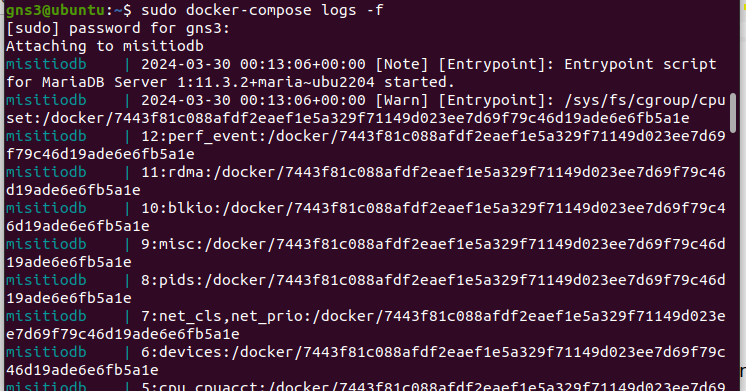


Pruebas, ahora realice una solicitud al puerto **8081**:



Puede monitorear la ejecución de los contenedores con:

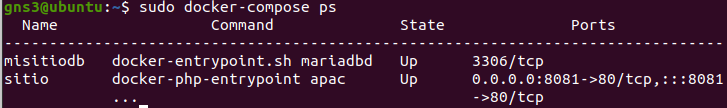
sudo docker-compose logs -f



Presione Control+C para volver al promp.

Para ver los contenedores existentes ejecute:

sudo docker-compose ps



Si desea ejecutar comandos dentro de un contenedor, puede hacerlo de la siguiente manera:

sudo docker-compose exec website bash

Donde:

website corresponde al nombre del servicio.

bash indica el programa a ejecutar.

Para eliminar los contenedores ejecute, siempre dentro del directorio con el archivo docker-compose.yml, los siguientes comandos:

sudo docker-compose stop

sudo docker-compose rm

**Ejemplo de creación de imagen en Docker**

Creamos un archivo Dockerfile

FROM php:8.1-apache

RUN apt-get update && apt-get install -y \

libfreetype6-dev \

libjpeg62-turbo-dev \

libmcrypt-dev \

libicu-dev \

libxml2-dev \

vim \

wget \

unzip \

&& docker-php-ext-install -j$(nproc) iconv intl opcache \

&& docker-php-ext-install -j$(nproc) gd \

&& docker-php-ext-install pdo\_mysql exif gettext mysqli

Cerrar el archive y ejecutar el liguiente comando

docker build -t miprimeraimagen:v1 .

Documentación de imagen base es: <https://hub.docker.com/_/php/>

