Curso de Especialización de Ciberseguridad en Entornos de Tecnología de la Información (CETI)

Puesta en Producción Segura

UD05. Usando Docker.
Tarea Online.

JUAN ANTONIO GARCIA MUELAS

INDICE

1.- Descripción de la tarea.

Caso práctico

dotCloud, Inc.. *Docker Logo* (Apache License 2.0)

Julián se ha unido a un grupo de trabajo para la creación de un aplicativo web para entornos web, sabe que sus compañeros están usando Docker y aún tiene cierto temor al uso de esta tecnología por desconocimiento por lo que va repasando cada paso que da.

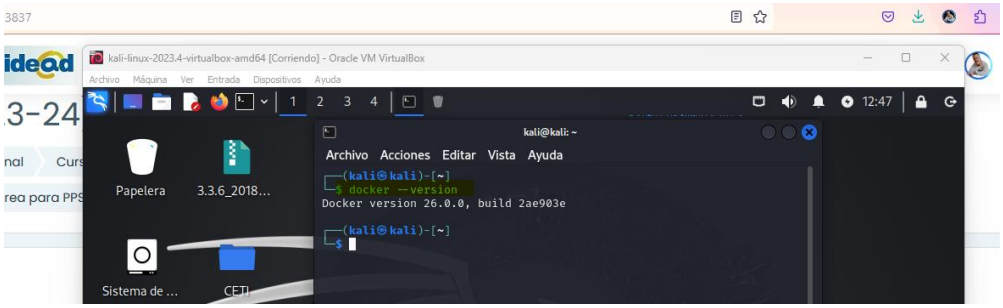
¿Qué te pedimos que hagas?

Esta actividad tiene parte teórica y parte práctica.

✓ **Apartado 1: Manejo básico de contenedores con Docker**

1. En las unidades 2 y 3 ya se ha utilizado Docker así que ya debes tenerlo instalado, en caso contrario debes instalarlo. Puedes instalarlo en tu equipo o crear una máquina virtual e instalarlo en la misma (en ese caso te recomendamos utilizar una máquina Ubuntu porque va a ser muy sencillo).

   Inicio la VM y compruebo la instalación de docker de tareas anteriores.

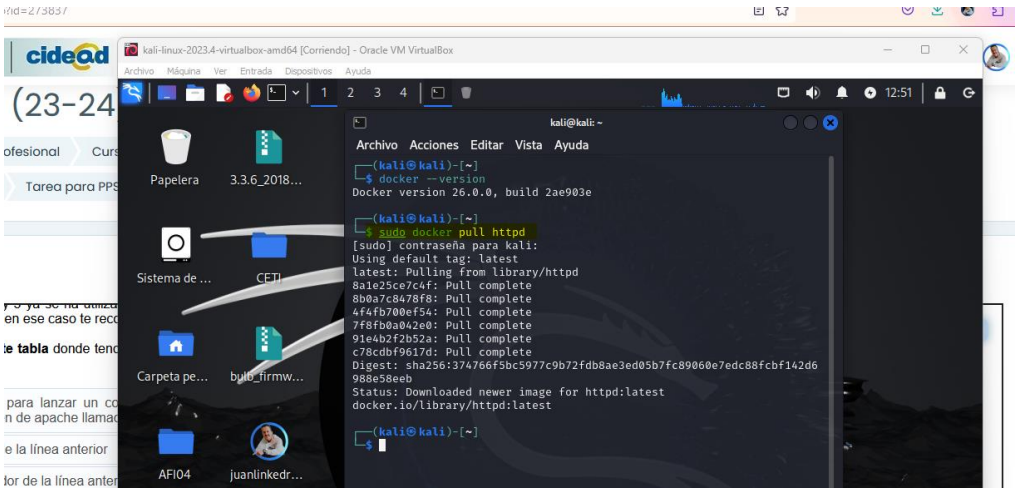


2. **Rellena la siguiente tabla** donde tendrás que utilizar los comandos básicos de Docker. Además, **ejecuta esos comandos en tu sistema e incluye pantallazos de la ejecución de esos comandos.**

Línea de comandos para lanzar un contenedor de nombre contenedor1 basado en una imagen de apache llamada httpd	Incluyo Apache con: **`docker pull httpd`** Creo y lanzo el contenedor con: **`sudo docker run --name contenedor1 -p 8080:80 -d httpd`**
Parar el contenedor de la línea anterior	**`docker stop contenedor1`**
Continuar el contenedor de la línea anterior	**`docker start contenedor1`**
Terminar el contenedor de la línea anterior	**`docker stop contenedor1`** **`docker rm contenedor1`**
Ver los contenedores en ejecución	**`docker ps`**

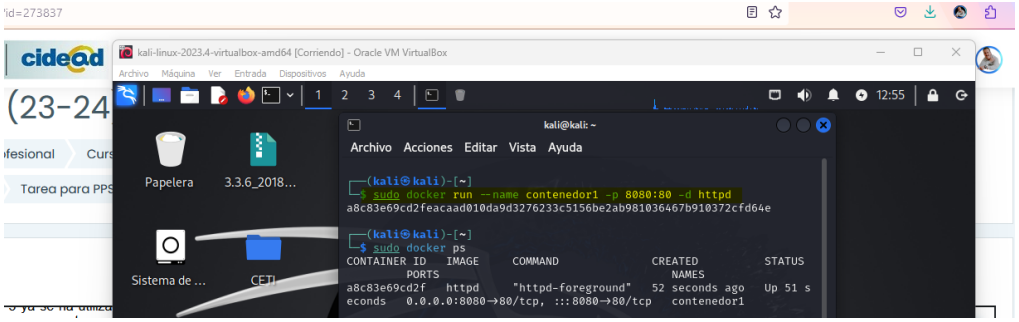
Como hemos visto en la tabla, podemos comenzar instalando el proyecto Apache descargando la imagen **`httpd`**

`sudo docker pull httpd`

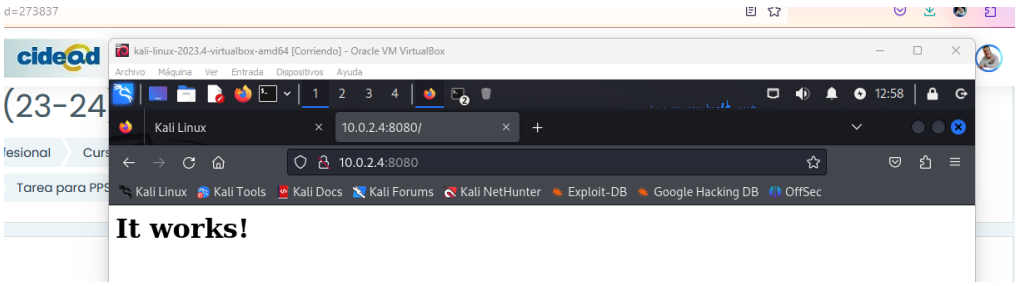


Luego podemos ya lanzar el contenedor, teniendo como base la imagen **`httpd`**, asignando el puerto **`8080`**, y lo ejecutamos en segundo plano.

Tras esto, podemos listar los contenedores activos para comprobar que lo hemos hecho correctamente.



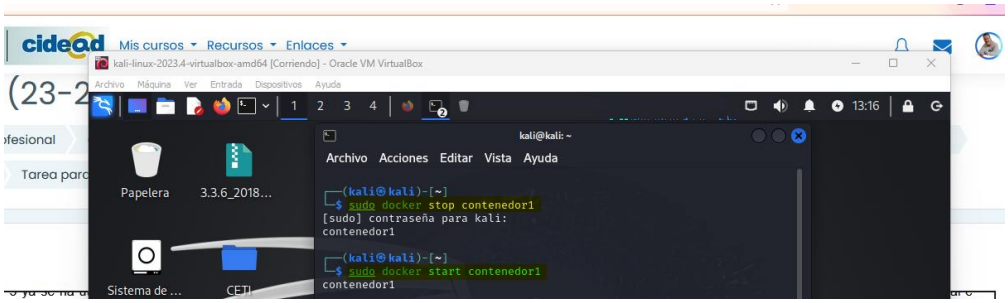
También podemos abrir el navegador y comprobar que conecta desde el puerto asignado.



Paramos el contenedor con **`docker stop contenedor1`**.

Si solo lo queremos pausar, utilizamos **`docker pause contenedor1`**.

Continuamos el contenedor. Podemos hacerlo con **`start`** o con **`restart`** (este comprueba si está activo, lo detiene y lo vuelve a lanzar).



Terminamos con el contenedor parando y eliminando o forzando la eliminación (**-f**).

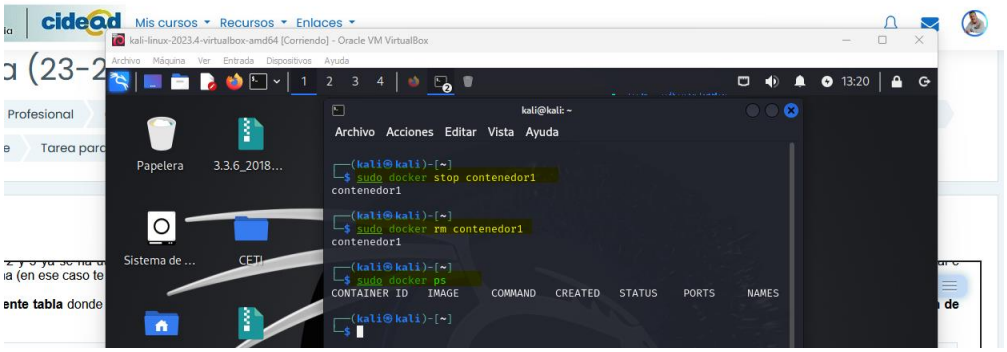
Por claridad del proceso, muestro la primera opción:

```
docker stop contenedor1
docker rm contenedor1
```

También tenemos la opción **`docker kill contenedor1`** para matarlo, si se congela y no responde, por ejemplo.

Comprobamos los contenedores en ejecución:

```
docker ps
```



Vemos que ya no hay ningún contenedor activo.

3. **Responde a la siguiente pregunta** ¿Qué ventajas aporta trabajar con contenedores?

Portabilidad: Los contenedores son portables, o lo que es lo mismo, pueden ejecutarse en cualquier sistema operativo que tenga Docker instalado, facilitando el desarrollo y la implementación de aplicaciones.

Aislamiento: Los contenedores se ejecutan en un entorno aislado, lo que significa que no tienen acceso directo al sistema operativo o a otras aplicaciones que se ejecutan en el mismo host, mejorando la seguridad y confiabilidad de las aplicaciones desarrolladas.

Eficiencia de recursos: Los contenedores son más eficientes en el uso de recursos que las máquinas virtuales, porque no necesitan de un sistema operativo completo (comparten el kernel).

Escalabilidad: Los contenedores son de por sí muy livianos y se pueden escalar fácilmente para aumentar o disminuir la capacidad de una aplicación. Esto puede ser útil para aplicaciones que necesiten escalar horizontalmente.

Facilidad de uso: Tiene una interfaz de línea de comandos relativamente simple para administrar los contenedores.

Colaboración: Los contenedores facilitan la colaboración entre equipos de desarrollo, pudiendo compartir imágenes con todo lo necesario para ejecutar una aplicación.

Agilidad: Por lo anterior, permiten a las empresas ser más ágiles en su desarrollo e implementación de software, con desarrollos, pruebas e implementaciones más rápidas.

✓ Apartado 2: Dockerfile

1. Dado el siguiente archivo Dockerfile

```
FROM ubuntu
RUN apt-get update && apt-get install -y apache2
EXPOSE 80
ADD ["index.html","/var/www/html/"]
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Rellena la siguiente tabla donde tendrás que indicar **para que sirve cada línea**

`FROM ubuntu`	Ubuntu es la imagen base del contenedor.
`RUN apt-get update && apt-get install -y apache2`	Actualiza todos los paquetes de Ubuntu e instala el servidor **`Apache2`**
`EXPOSE 80`	Puerto del servidor Apache del contenedor visible desde el exterior.
`ADD ["index.html","/var/www/html/"]`	Añade el archivo **`index.html`** al directorio **`/var/www/html`**. Así será visible al visitar el contenedor.
`ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]`	Al iniciar el contenedor se ejecuta como demonio (**`-D`**) **`apache2ctl`** en primer plano (**`FOREGROUND`**). Al verse la salida en primer plano, puede ser útil para depurar.

2. Responde a las siguientes preguntas

2.1. ¿Para qué es útil un archivo Dockerfile?

Los Dockerfile son archivos que definen las instrucciones para crear una imagen de Docker (un paquete ligero, portátil y autosuficiente con todo lo necesario para ejecutar una aplicación).

Por esa misma definición pueden ser útiles para:

Automatizar la creación de imágenes: Permitiendo crear imágenes de forma consistente, reproducible e independiente del entorno en el que se ejecuten.

Compartir imágenes: Podemos compartir las imágenes entre desarrolladores o equipos facilitando la colaboración y la reutilización del código.

Aislar las aplicaciones: Cada imagen se ejecuta en su propio contenedor, aisladas del sistema operativo y de otras aplicaciones, mejorando la seguridad.

Facilitar la implementación: Podemos implementar aplicaciones en cualquier entorno con Docker instalado, sin necesidad alguna de modificar el código.

Reducir el tamaño de las imágenes: Se pueden optimizar las imágenes para que sean lo más pequeñas posible, reduciendo tiempo de descarga y espacio en disco.

2.2. ¿Qué línea de comandos debes ejecutar para crear una imagen nueva con ese archivo Dockerfile de nombre mi-apache versión 1.0?

Podemos crear una imagen de Ubuntu con servidor web **Apache2** que muestre el archivo **index.html** desde el puerto **80** con:

```
docker build -t mi-apache:1.0
```

build buscará el directorio el archivo para crear la imagen con el nombre y versión que le estamos pasando.

✓ **Apartado 3: Opciones avanzadas y docker-compose**

1. Dada la siguiente línea de comandos que lanza un contenedor

docker run -d -p 33060:3306 --name mysql-db1 -e MYSQL_ROOT_PASSWORD=secret -v ./mysql:/var/lib/mysql mysql:5.7.28

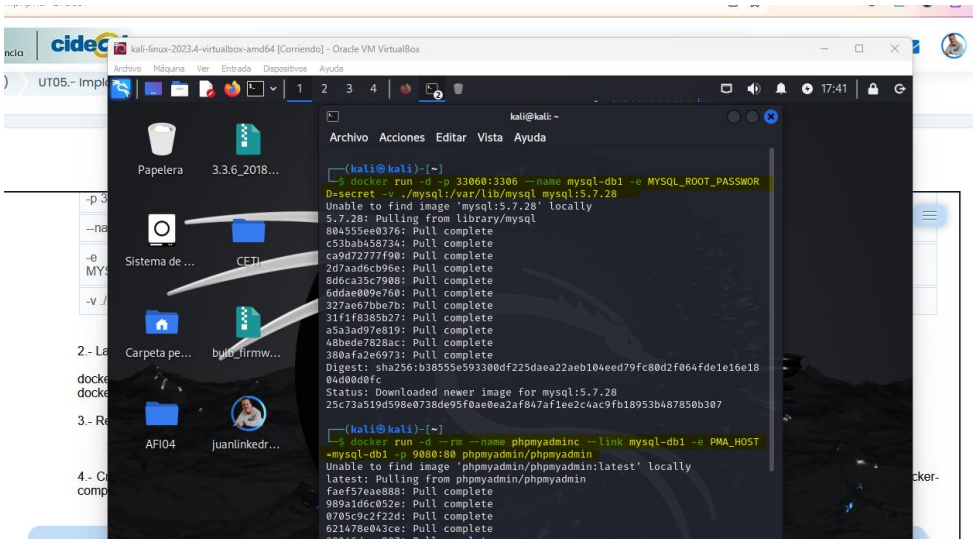
Rellena la siguiente tabla indicando para que sirve cada opción

`-d`	Inicia y ejecuta el contenedor en segundo plano.
`-p 33060:3306`	Todas las solicitudes que entran por el puerto **33060** de la máquina, se reenvían al **3306** del contenedor (**MySQL**).
`--name mysql-db1`	Nombre asignado al contenedor.
`-e MYSQL_ROOT_PASSWORD=secret`	Valor de la contraseña de la cuenta **root**.
`-v ./mysql-data:/var/lib/mysql`	Monta un volumen del directorio **mysql-data** en la ruta **/var/lib/mysql** para almacenar los datos y que no se pierdan tras detener el contenedor.

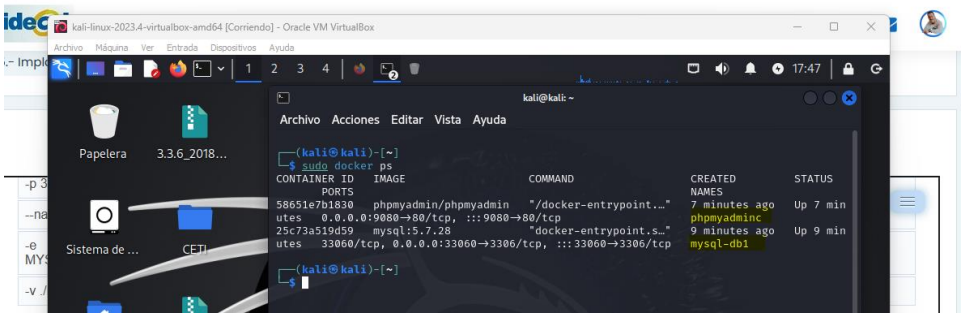
2. Lanza dos contenedores con las siguientes líneas en una terminal y **muestra pantallazos de la ejecución de esos comandos (docker ps -a)**

docker run -d -p 33060:3306 --name mysql-db1 -e MYSQL_ROOT_PASSWORD=secret -v ./mysql:/var/lib/mysql mysql:5.7.28
docker run -d --rm --name phpmyadminc --link mysql-db1 -e PMA_HOST=mysql-db1 -p 9080:80 phpmyadmin/phpmyadmin

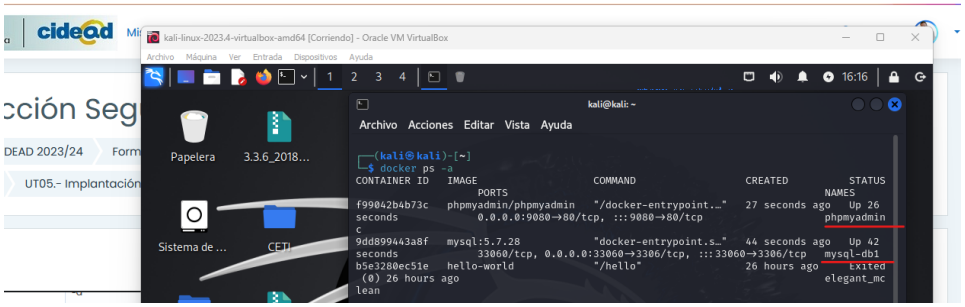
Lanzamos los dos contendores desde nuestra terminal y esperamos que se ejecuten.



Hacemos un **`docker ps`** para comprobarlo.



O con docker ps -a para ver todos los contenedores



Con los dos contenedores en ejecución podemos pasar al siguiente punto.

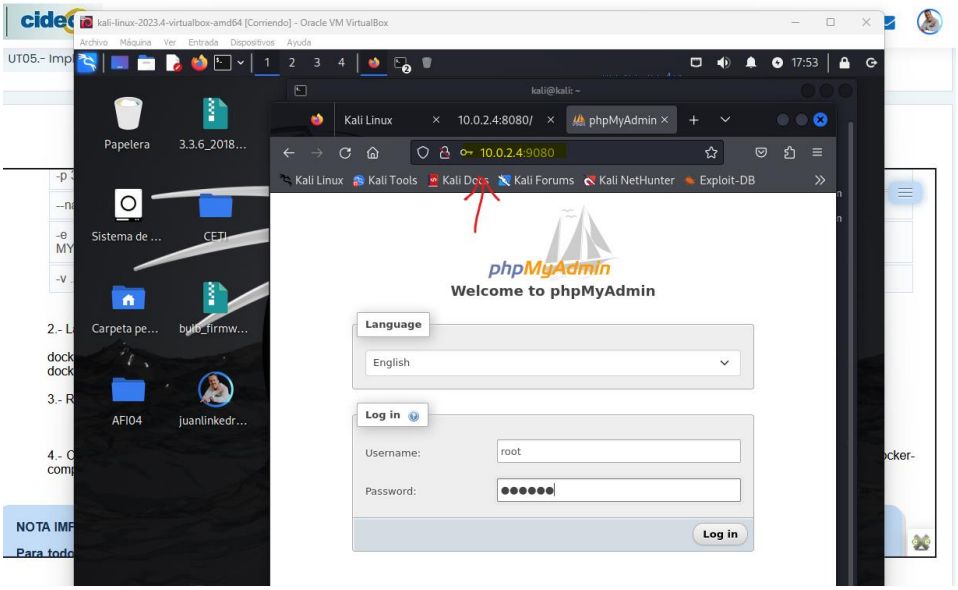
3. Responde a las siguientes preguntas:

3.1. ¿Puedo acceder a esa base de datos mysql desde un navegador (ten en cuenta que has lanzado un contenedor de phpmyadmin)?

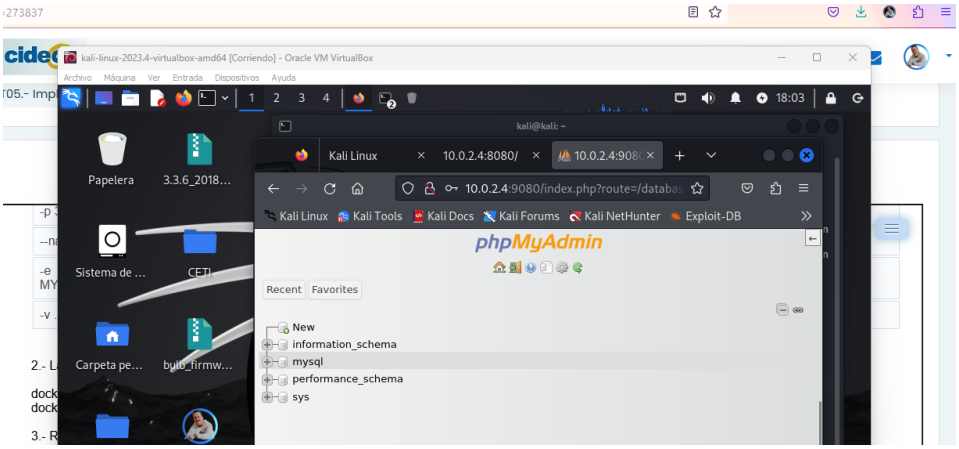
Si es posible muestra un pantallazo del acceso desde el navegador.

Con el segundo contenedor, hemos "linkeado" la conexión, por lo que la forma más sencilla para acceder desde el navegador será con la **`IP`** de nuestra máquina y el puerto que se asignó a la conexión desde el contenedor (**`9080`**). Además, debería poder loguearme con un usuario **`root`** y contraseña **`secret`**, como se definió en el primer contenedor.

Probamos entonces la conexión.

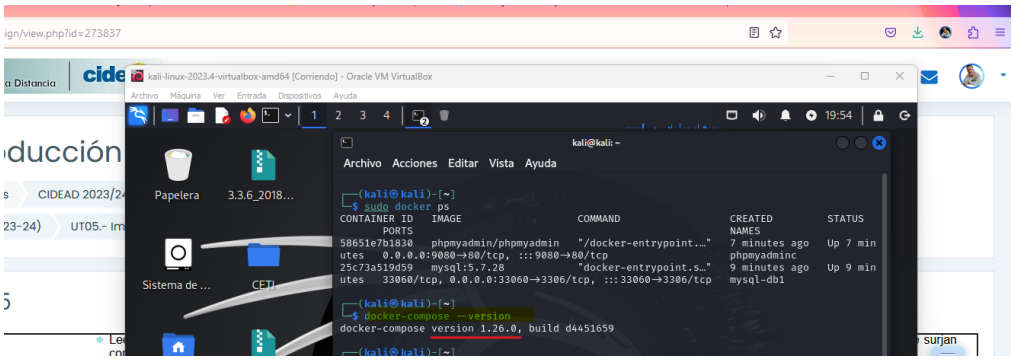


Vemos que accede con los datos configurados.



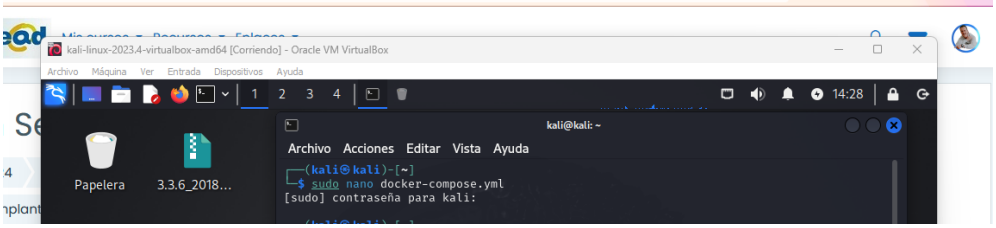
4. Crea un archivo **docker-compose.yml** que permita lanzar esos dos contenedores utilizando docker compose. Incluye el contenido del archivo docker-compose.yml y pantallazos de la ejecución del mismo.

Comprobamos que tenemos una versión de **`docker-compose`** instalada.



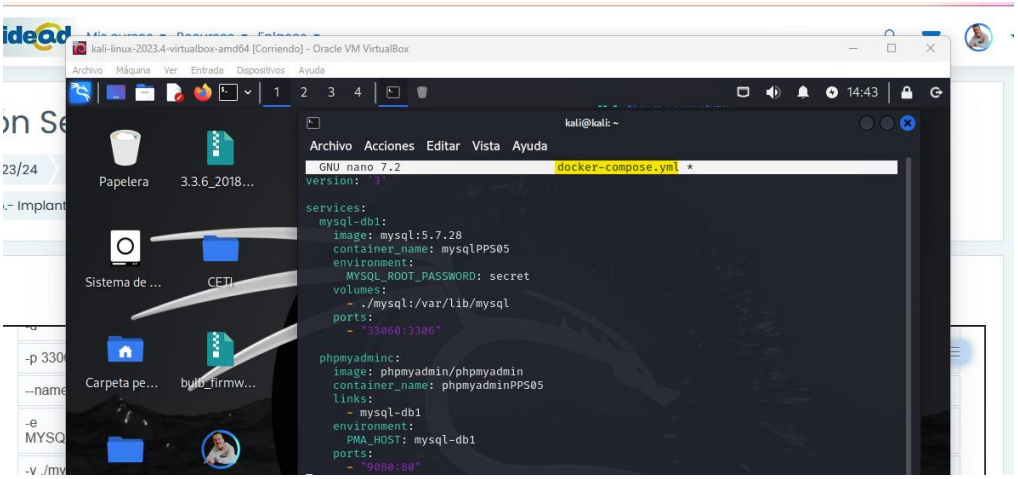
Debemos crear con los datos que nos han aportado el archivo **`docker-compose.yml`**

```
sudo nano docker-compose.yml
```

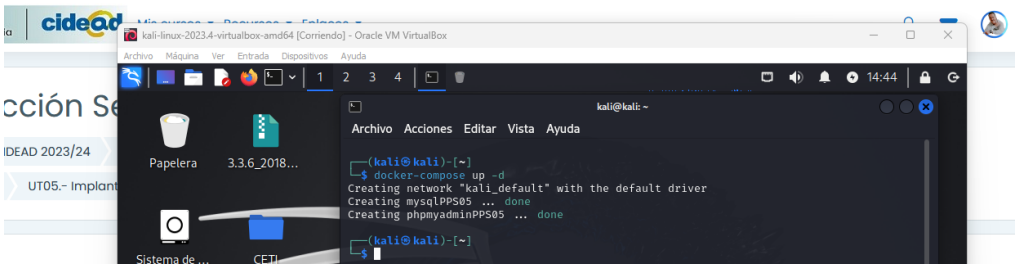



```
version: '3'

services:
  mysql-db1:
    image: mysql:5.7.28
    container_name: mysqlPPS05
    environment:
      MYSQL_ROOT_PASSWORD: secret
    volumes:
      - ./mysql:/var/lib/mysql
    ports:
      - "33060:3306"

  phpmyadminc:
    image: phpmyadmin/phpmyadmin
    container_name: phpmyadminPPS05
    links:
      - mysql-db1
    environment:
      PMA_HOST: mysql-db1
    ports:
      - "9080:80"
```

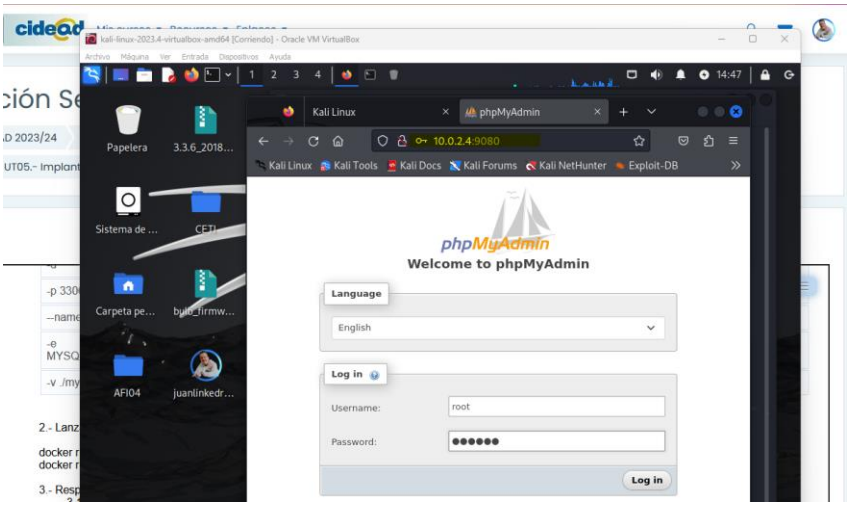
Por coherencia y convención, deberían de haberse guardado los datos como **mysql** y **phpmyadmin** para que sean más cortos, sin guiones, más claros, legibles y sencillos de utilizar, pero dado el enunciado, los mantengo así.

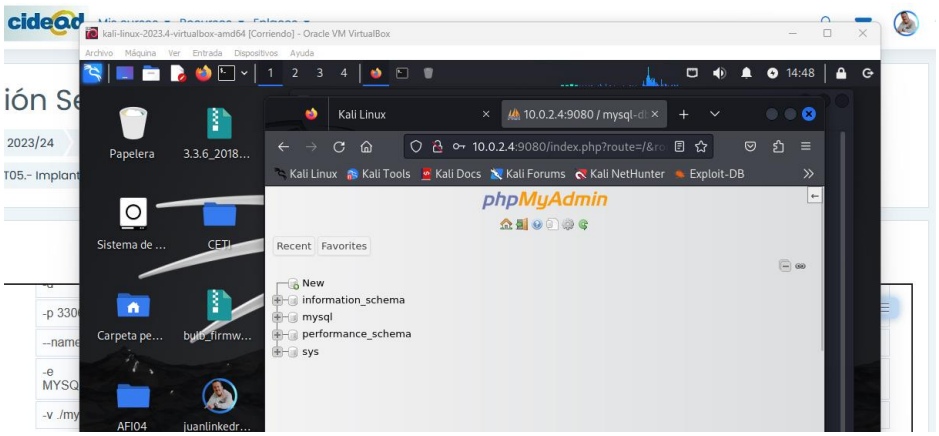


Lanzamos la instalación con la opción **-d** para ejecutarlo en segundo plano y esperando no ver algún error (en ese caso, nos lo mostrará de forma bastante precisa)

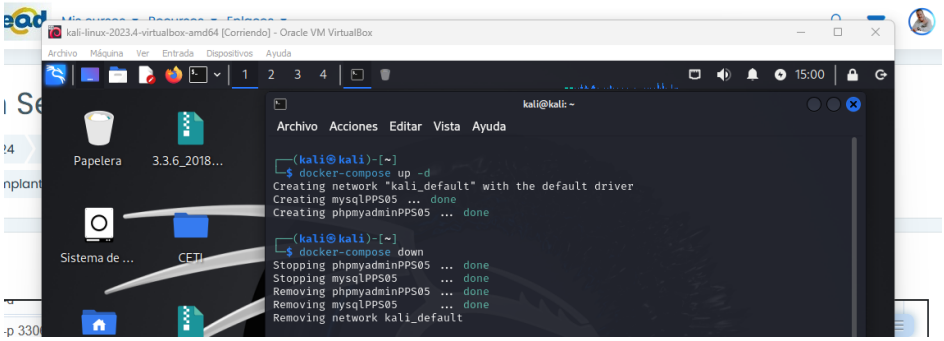


Podemos comprobar que accedemos de la misma manera que anteriormente.





Tras verlo todo correcto, podremos cerrarlos con **`docker-compose down`**.



Webgrafía

https://www.mecd.es/cidead/aulavirtual/course/view.php?id=2384#section-5

https://docs.docker.com/engine/reference/commandline/cli/

https://www.hostinger.es/tutoriales/como-crear-contenedor-docker

https://docs.docker.com/engine/reference/run/

https://docs.docker.com/reference/dockerfile/

https://docs.docker.com/compose/

https://iesgn.github.io/curso_docker_2021/sesion5/comando.html