**CIFP Rodolfo Ucha Piñeiro**

**Puesta en Producción Segura**

**PRÁCTICA 2: SECDEVOPS**

**Nombre: Miguel Angel Jaimes Linares**

**Curso: CE Ciberseguridad en entornos TI**

# 

Contenido

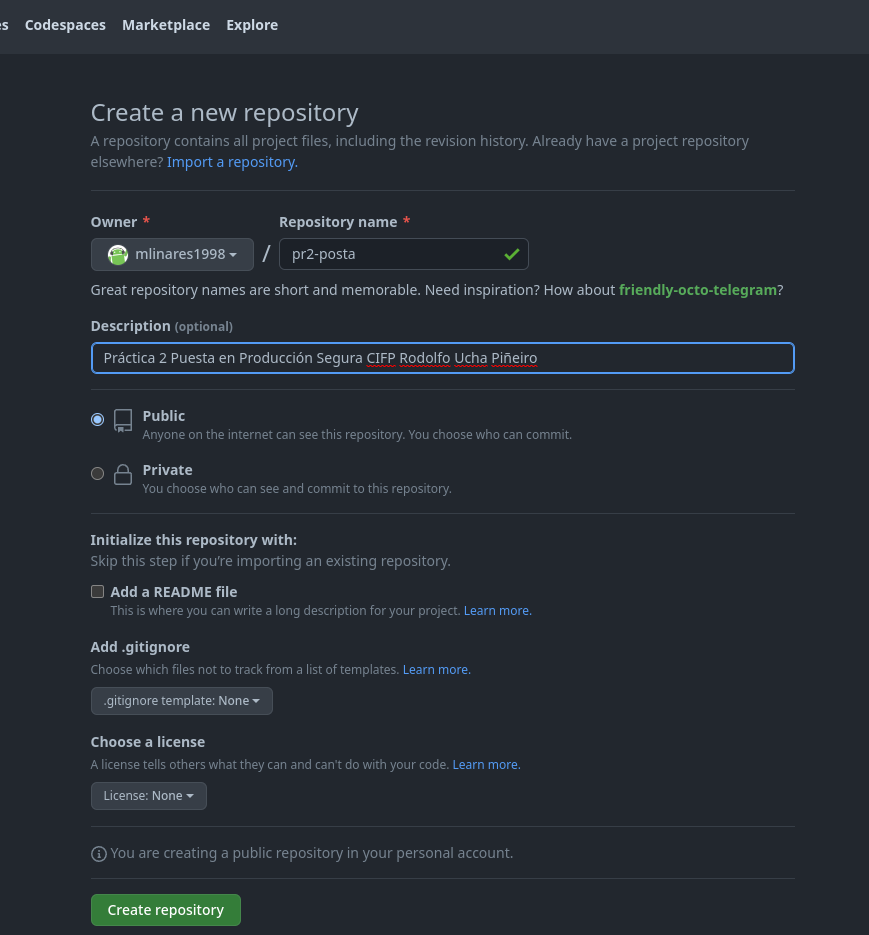
[Crear nuestro repositorio en GitHub 3](#_Toc125745879)

[Crear Dockerfile que copia la aplicación directamente desde el anfitrión 8](#_Toc125745880)

[Crear Dockerfile que prepara la aplicación desde una imagen de Ubuntu 12](#_Toc125745881)

# Crear nuestro repositorio en GitHub

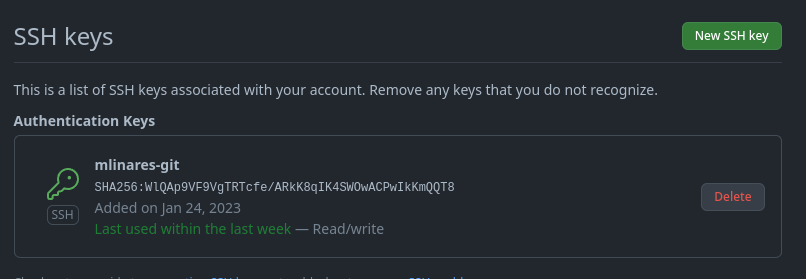
* Creamos un nuevo repositorio en GitHub para la realización de la práctica.

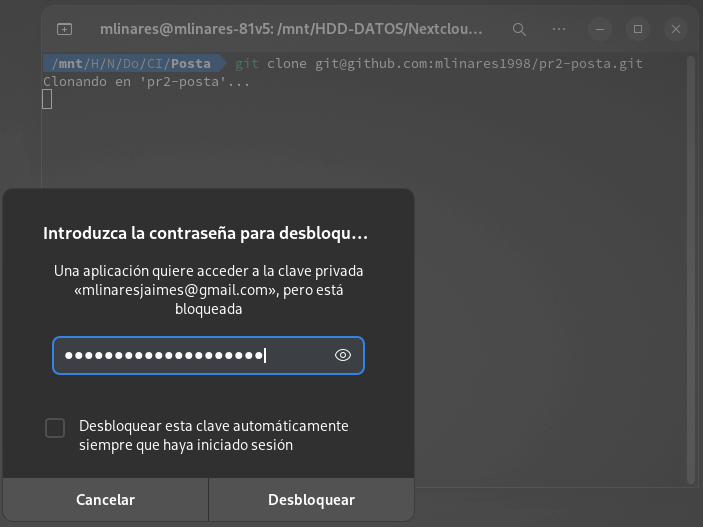


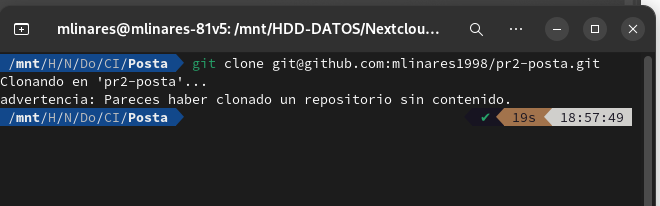
* Clonamos el repositorio en local usando git clone

**git clone** [**git@github.com:mlinares1998/pr2-posta.git**](mailto:git@github.com:mlinares1998/pr2-posta.git)

**NOTA: Estoy usando una clave SSH asociada a mi cuenta GitHub como mecanismo de autenticación con GitHub.**

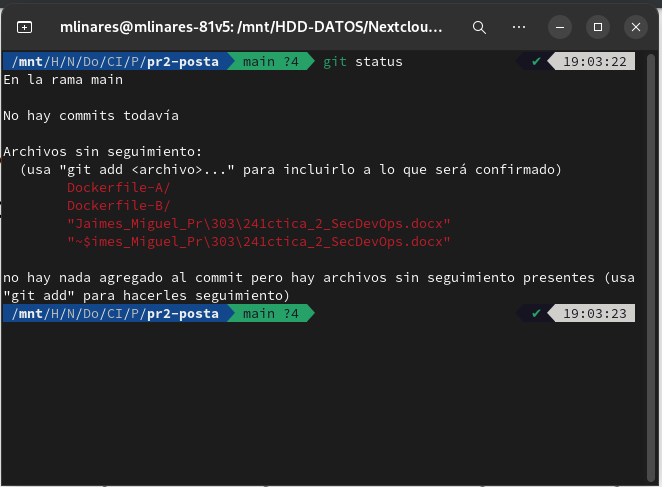
****

****

****

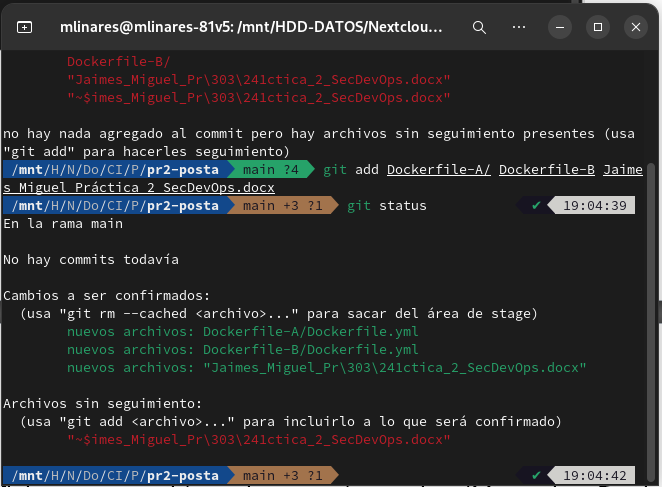
Nos ha clonado el repositorio vacío, listo para poder agregar nuestros archivos.

* Usamos **git status** para ver el estado de nuestro repo local.

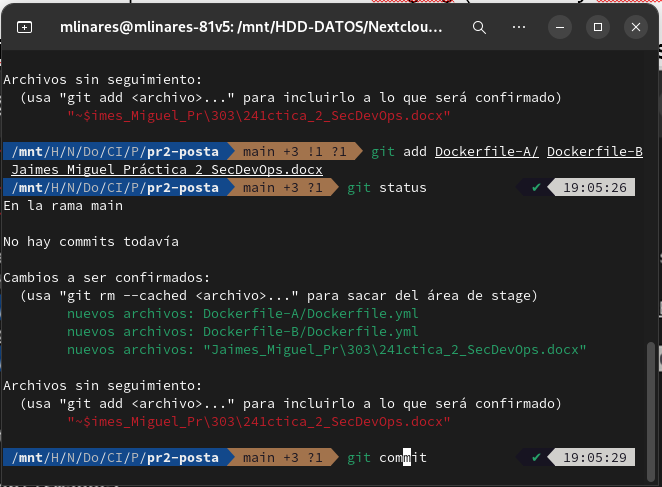


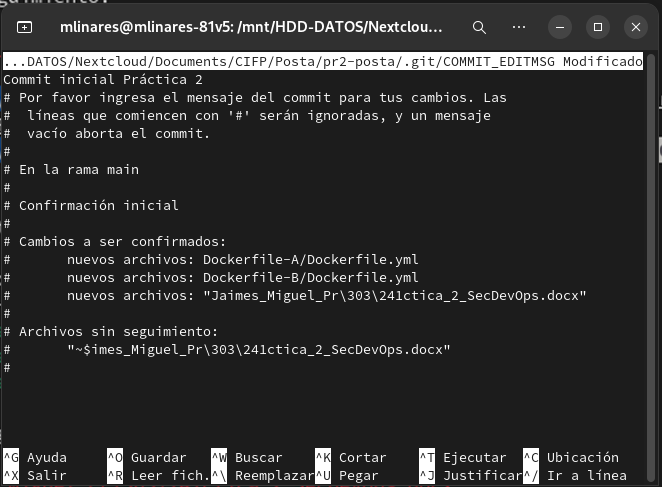
* Agregamos los ficheros requeridos a la zona de staging (Memoria y Dockerfile vacios)

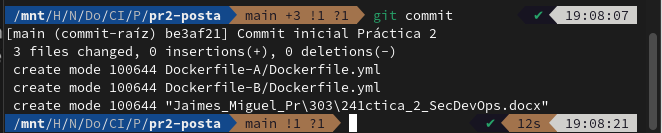
**git add Dockerfile-A/ Dockerfile-B Jaimes\_Miguel\_Práctica\_2\_SecDevOps.docx**



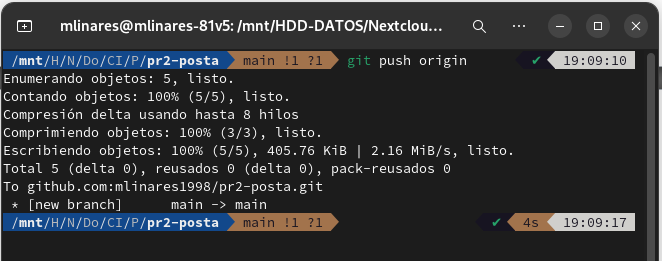
* Hacemos commit de nuestros cambios con **git commit**

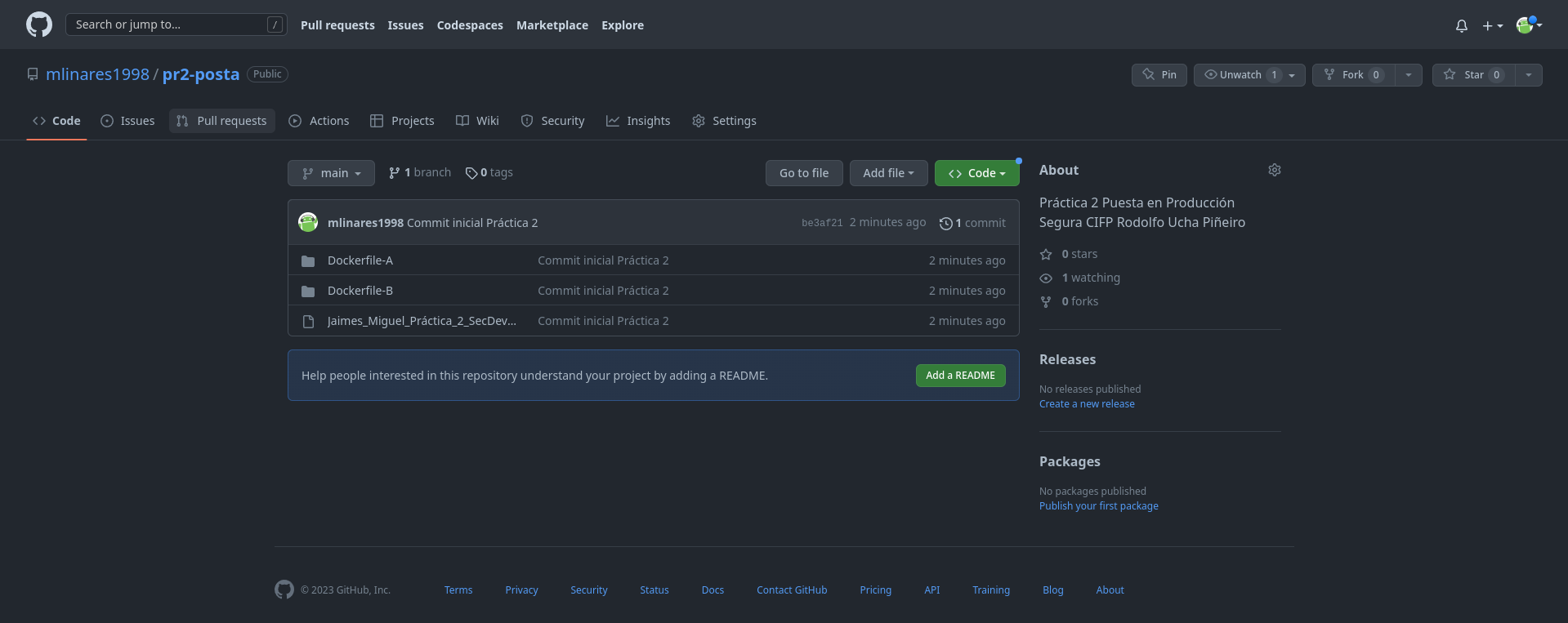






* Usamos **git push** para subir nuestros cambios al repo.





URL del repo: <https://github.com/mlinares1998/pr2-posta>

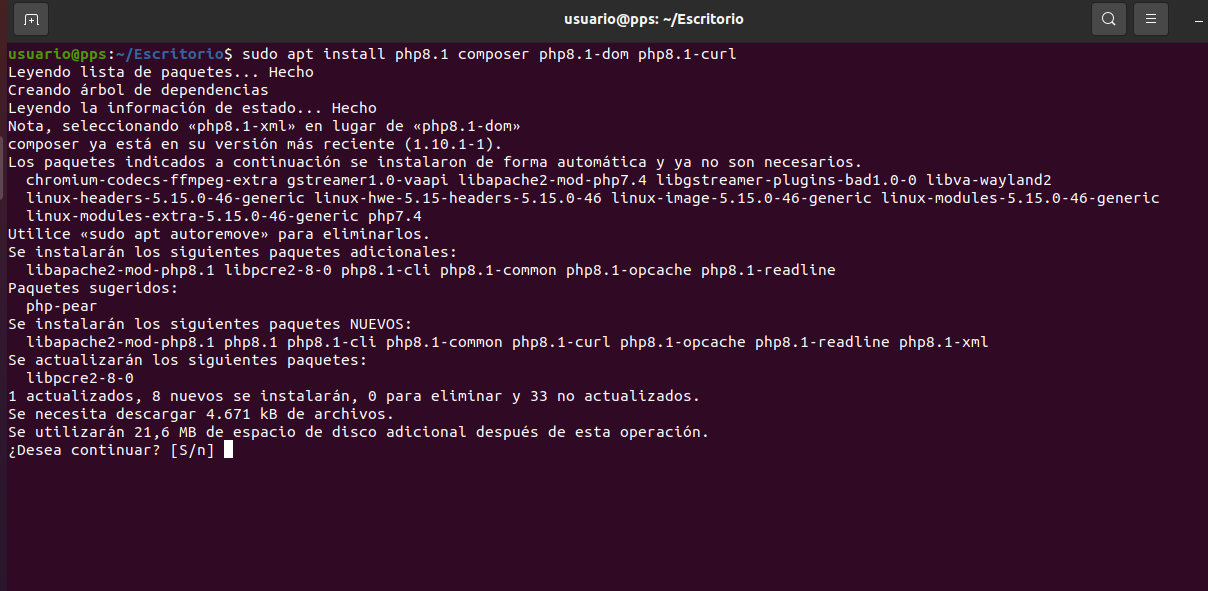
# Crear Dockerfile que copia la aplicación directamente desde el anfitrión

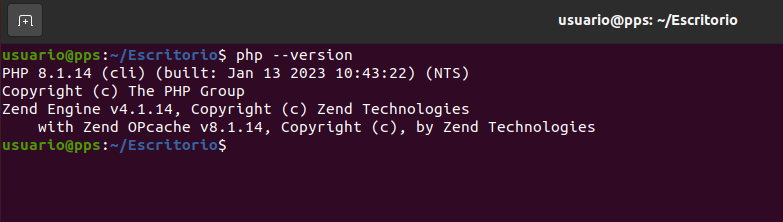
* Crea un Dockerfile que partiendo de una imagen PHP genera una imagen que copia una aplicación en PHP a un directorio del contenedor. Esta aplicación se debe copiar directamente desde un directorio del anfitrión.

Usaremos la siguiente aplicación demo:

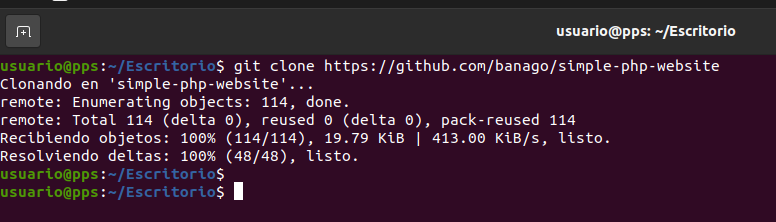
<https://github.com/banago/simple-php-website>

* Instalamos las dependencias necesarias con   
  **apt install php8.1 composer php8.1-dom php8.1-curl**



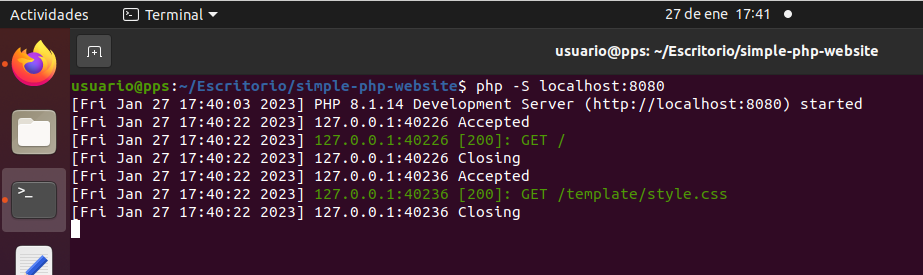


* Clonamos el repositorio



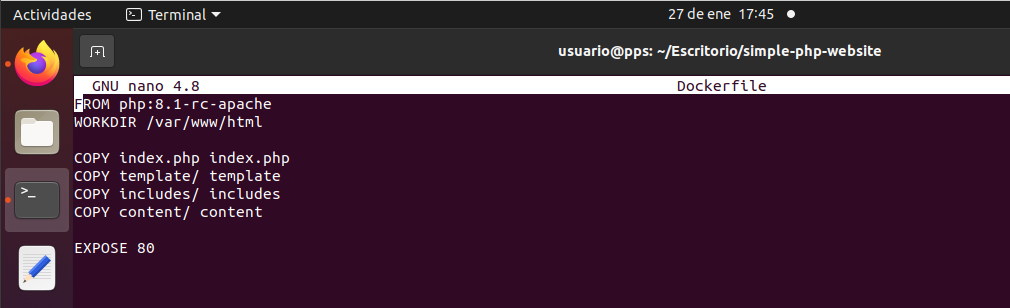
* Lanzamos la aplicación con el servidor de desarrollo de PHP para comprobar que funciona.

**php –S localhost:8080**





* Creamos nuestro fichero Dockerfile (En mi caso directamente usando nano).

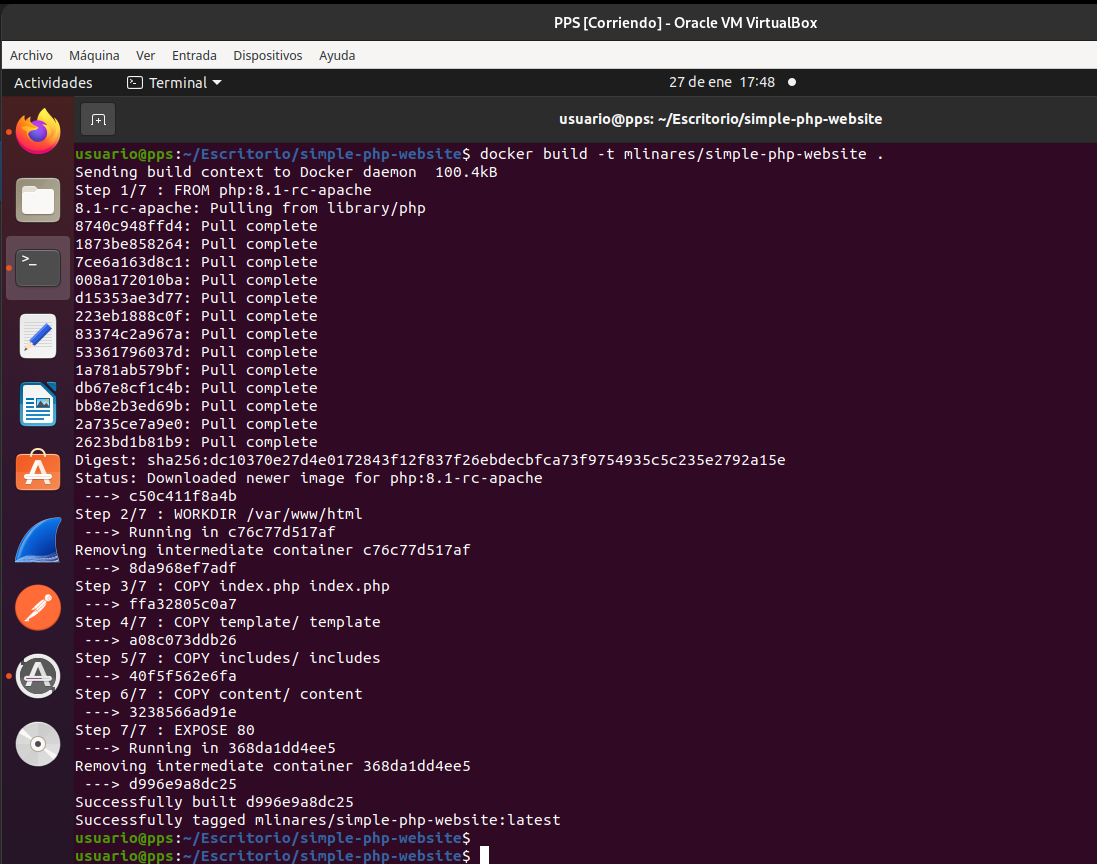


Este Dockerfile usara la imagen de PHP 8.1 con el servidor Apache pre-configurado, copiara el fichero index.php y carpetas requeridas a la carpeta /var/www/html del contenedor. Finalmente expone el servicio desde el puerto 80.

* Construimos la imagen.

Usamos el comando **docker build –t mlinares/simple-php-website**

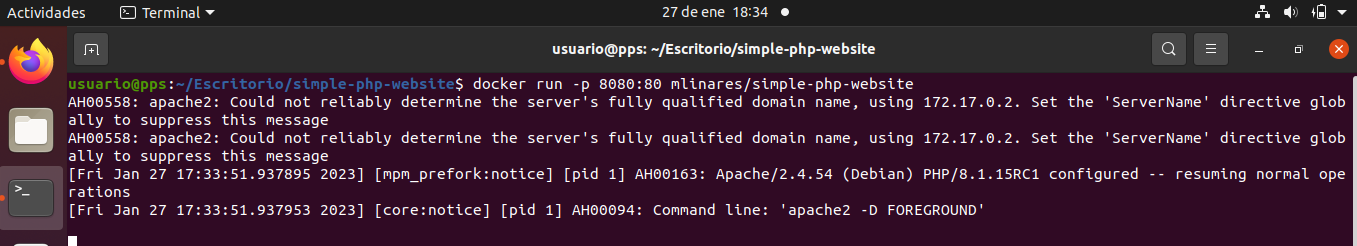
Nos construirá una imagen con el nombre especificado usando el Dockerfile de la carpeta en la que estamos posicionados.

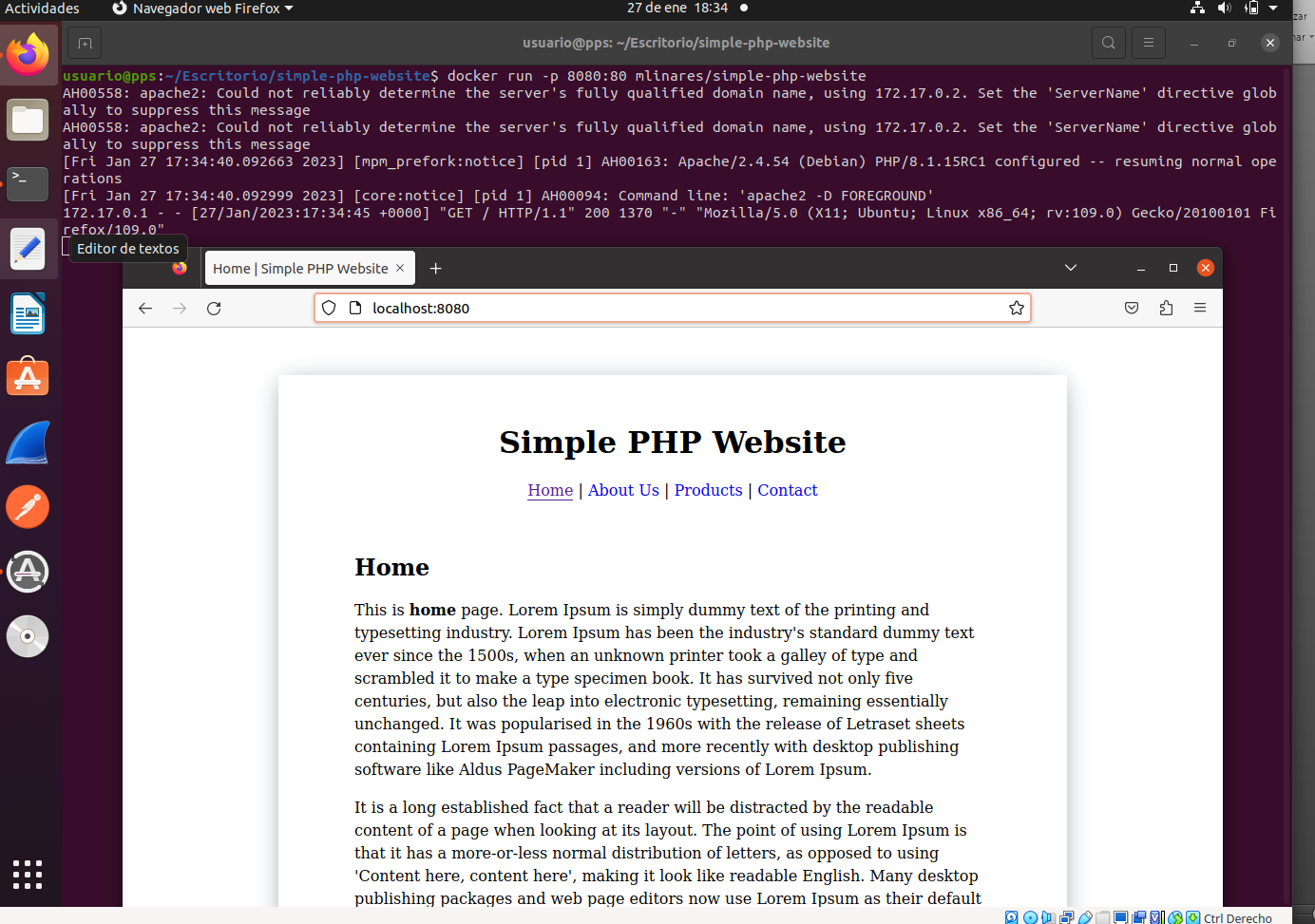


* Lanzamos la imagen y comprobamos que funciona correctamente.

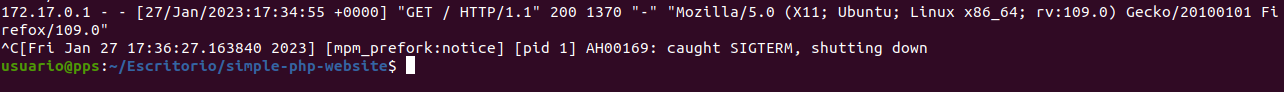
**docker run –p 8080:80 mlinares/simple-php-website**

Con el atributo –p enlazamos el puerto 8080 de nuestra máquina real con el puerto 80 interno del contenedor.





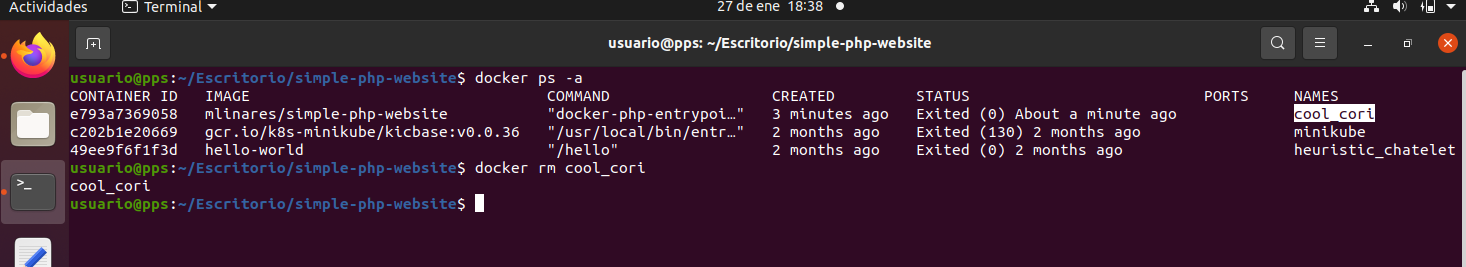
* Paramos el contenedor pulsando CTRL+C debido a que automáticamente hemos entrado en modo attached / interactivo.



* Eliminamos el contenedor

Usamos **docker ps –a** para obtener el slug del contenedor y su id (cool\_cori)

Luego eliminamos el contenedor usando **docker rm cool\_cori**



# Crear Dockerfile que prepara la aplicación desde una imagen de Ubuntu

* Crea un Dockerfile que partiendo de una imagen Ubuntu genera una imagen que:

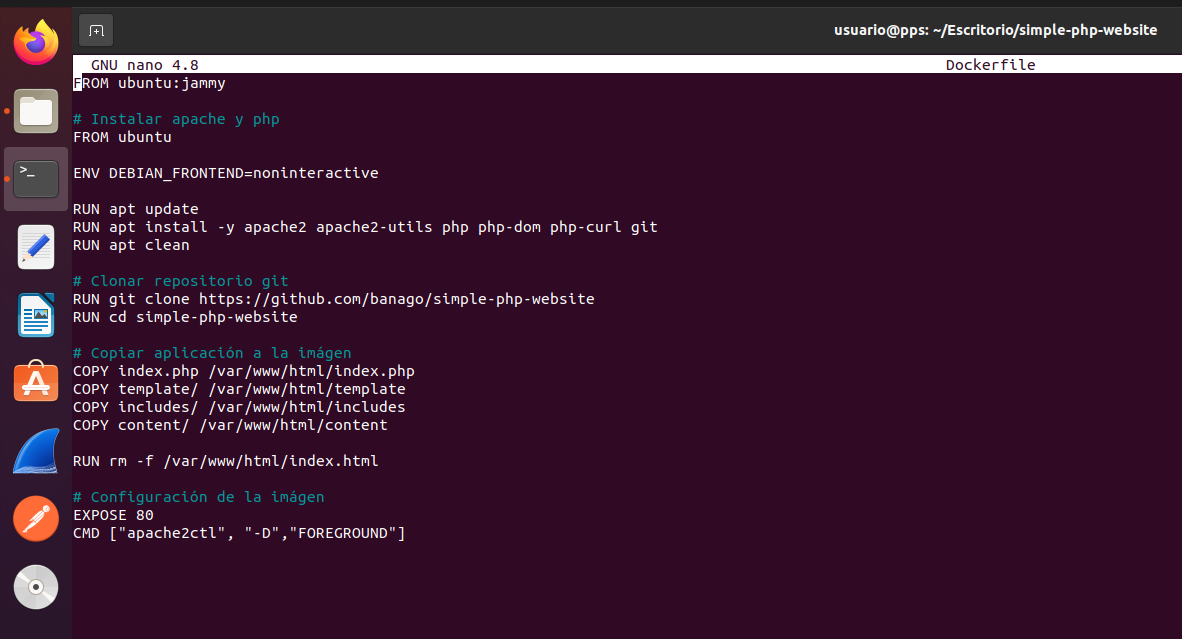
1. Instala Apache, de forma que se exponga el puerto 80.

2. Instala PHP.

3. Copia una aplicación web en PHP al directorio de Apache que expone las páginas web (Usamos la misma anterior). Esta aplicación se debe descargar automáticamente mediante algún comando como git clone o curl. Para facilitar las cosas, debe de ser una aplicación sencilla que no emplee bases de datos (ya que si no también habría que instalar un MySQL).

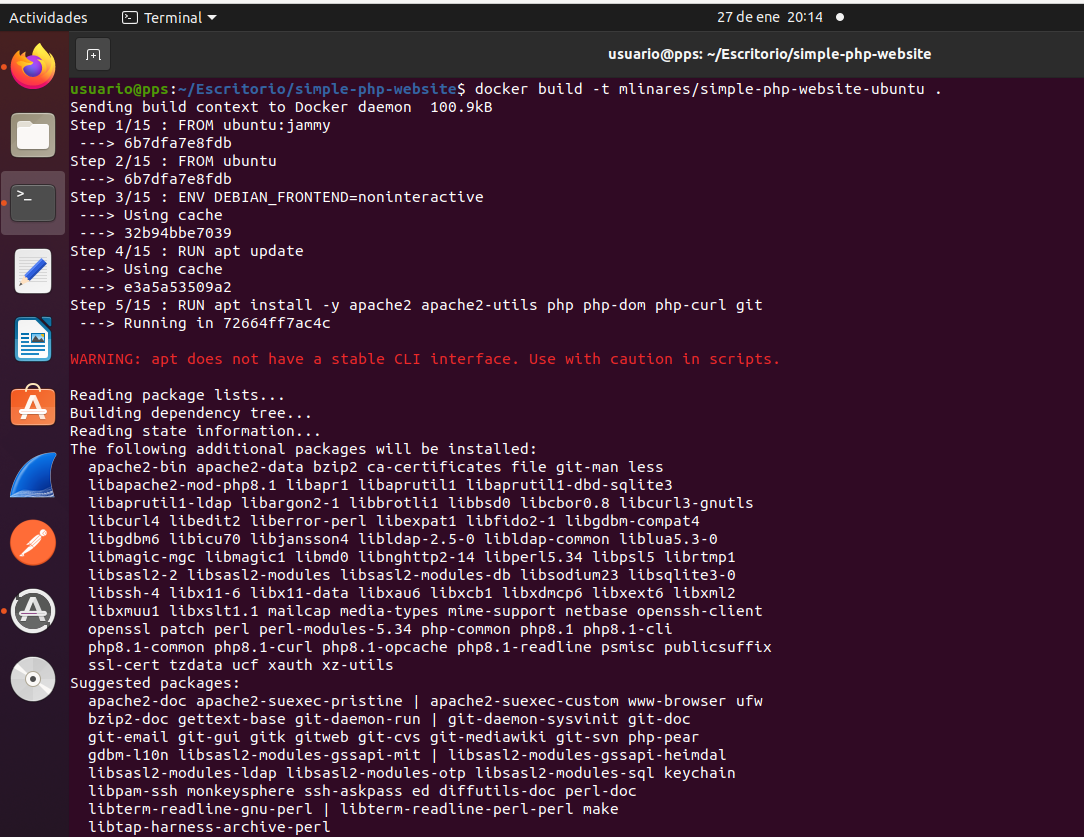
Este Dockerfile parte de una imagen de Ubuntu 22.04, la variable DEBIAN\_FRONTEND previene que al construir la imagen la shell pueda solicitar un input al usuario como ocurre al actualizar TZDATA.

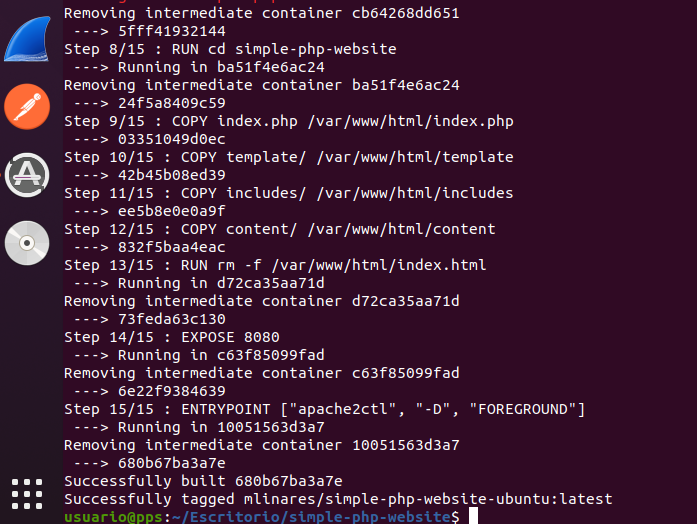
Posteriormente se instala Apache, PHP y git, se clona el repositorio, se copian los archivos en /var/www/html, se elimina el html por defecto incluido en apache y se empaqueta la aplicación, exponiendo el puerto 80 y lanzando Apache con CMD.



* Construimos la imagen con el comando

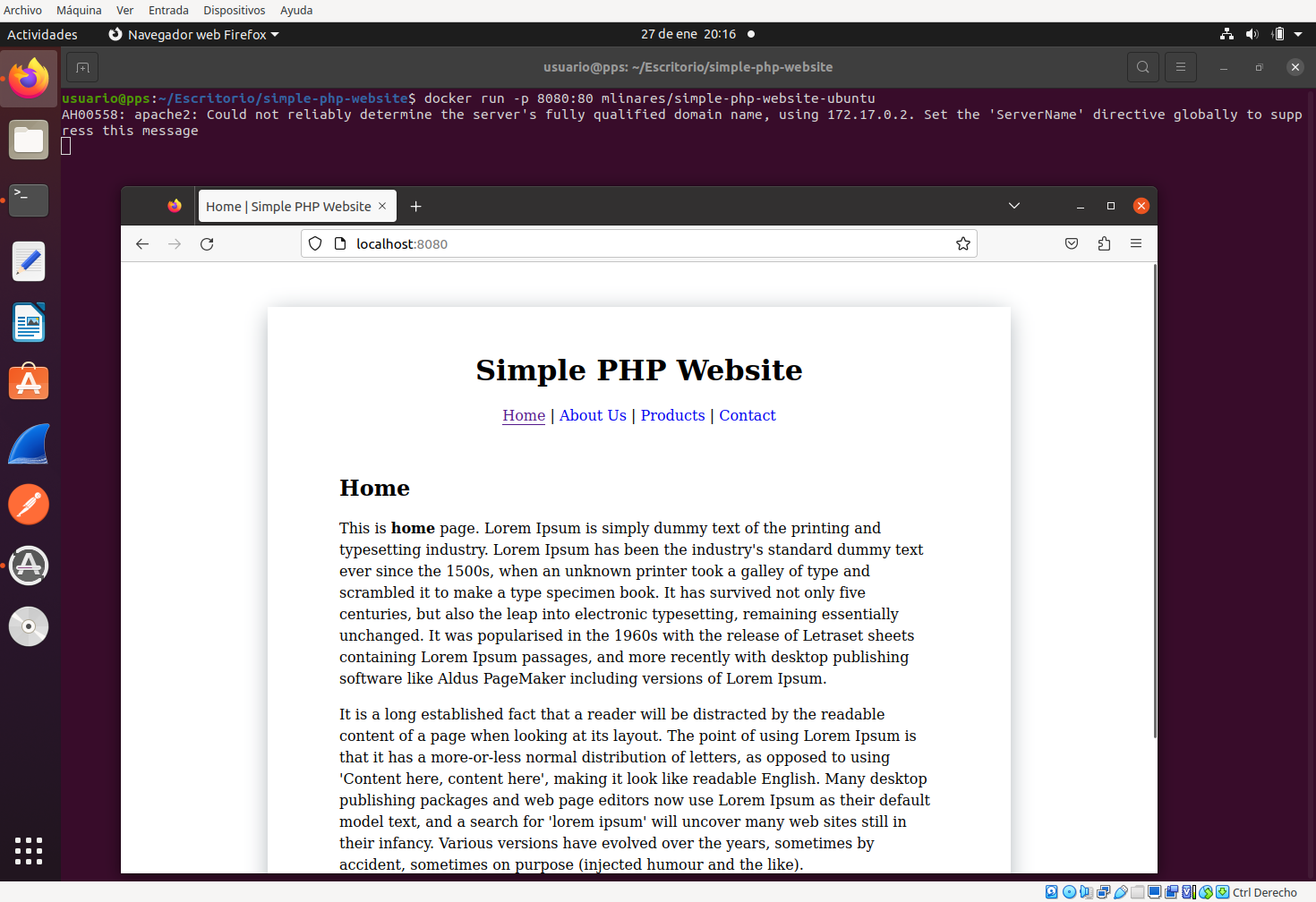
**docker build -t mlinares/simple-php-website-ubuntu .**

****

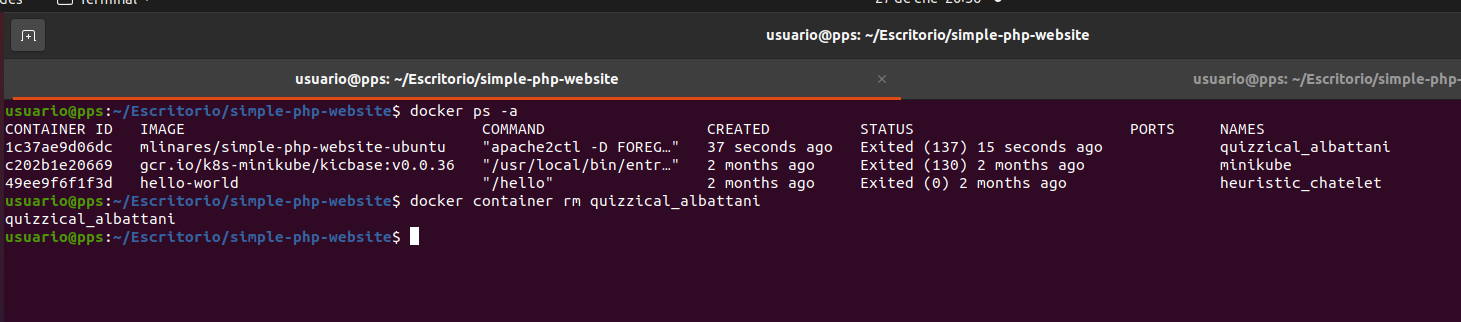
****

* Levantamos la imagen con el comando

**docker run –p 8080:80 mlinares/simple-php-website-ubuntu**

****

* Detenemos el contenedor y lo eliminamos

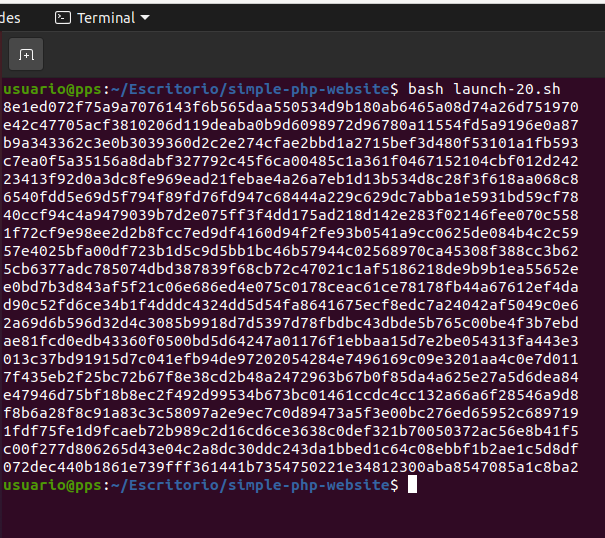
****

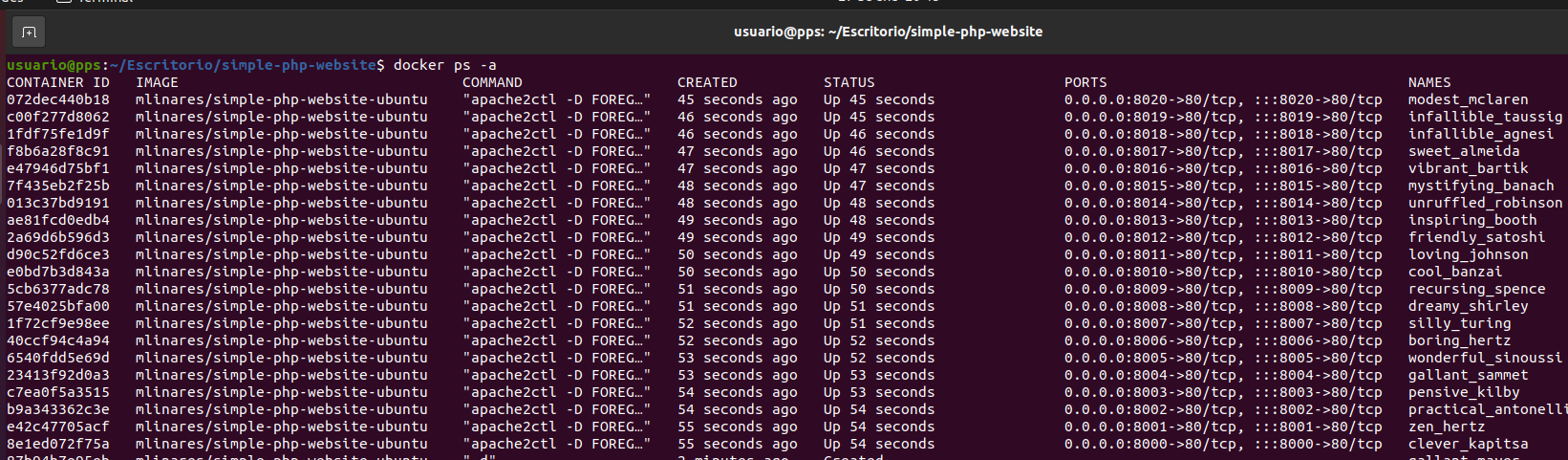
* Emplea un comando para lanzar 20 contenedores de la segunda imagen, cada uno mapeado en un puerto distinto del anfitrión. Cuando veas que funcionan, para y borra dichos contenedores.

Usamos el siguiente script de bash

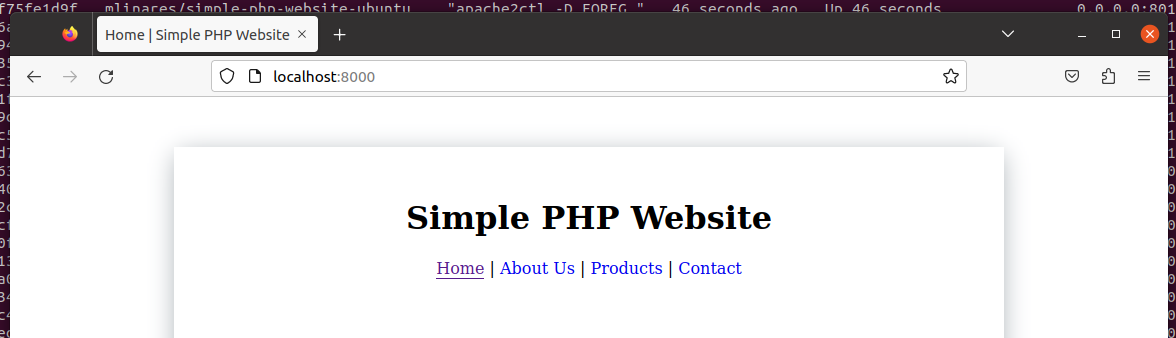
****

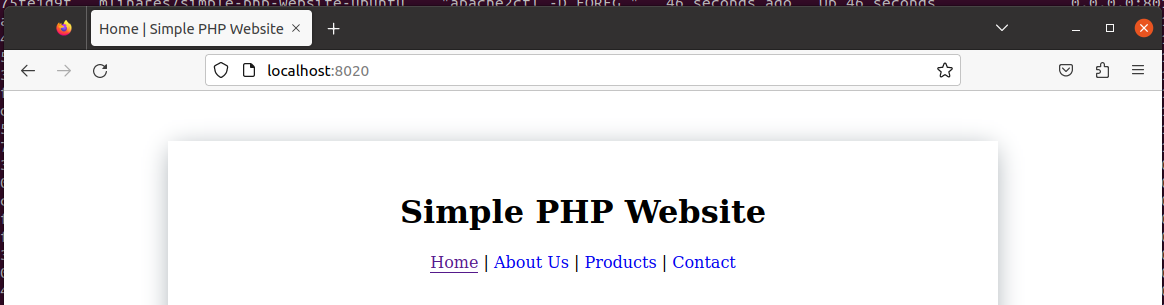
* Observamos en la terminal que se han lanzado los 20 contenedores

****

****

* Intentamos acceder a los servidores alojados en los puertos 8000 y 8020 como prueba. Funcionan correctamente.

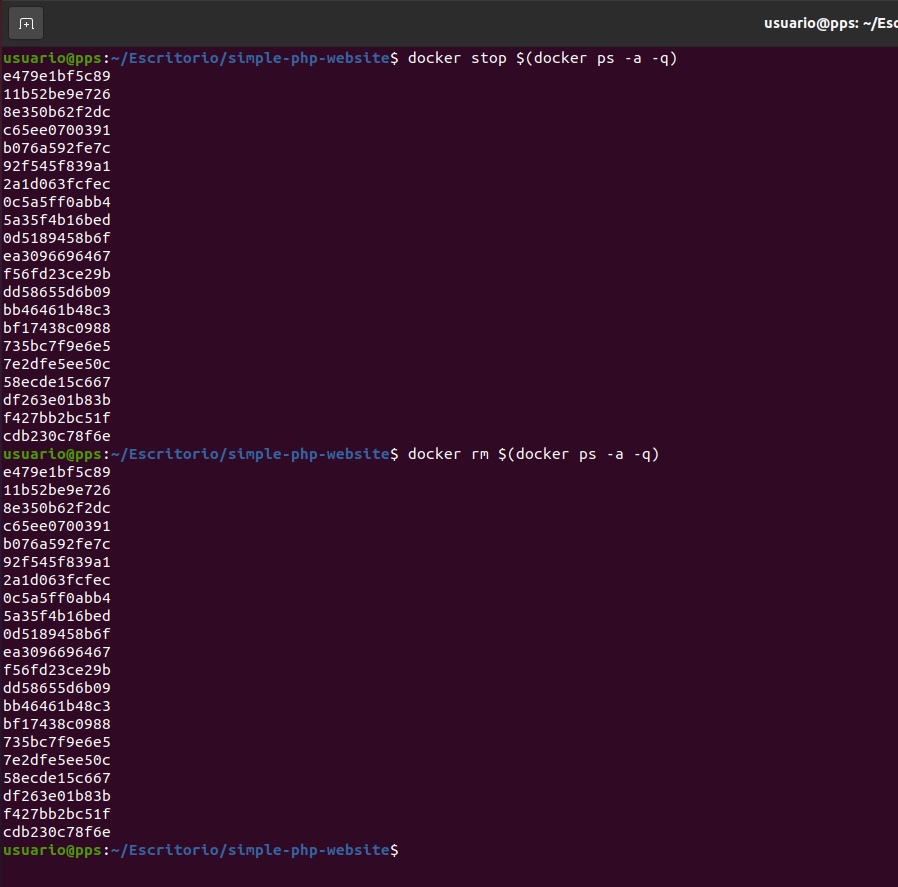
****

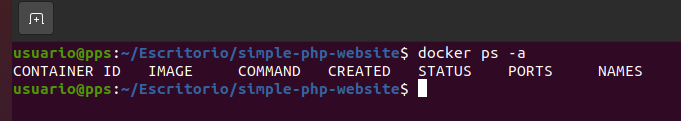
****

* Usamos estos dos comandos para detener y eliminar todos los contenedores.

**docker stop $(docker ps -a -q)**

**docker rm $(docker ps -a -q)**

****

****