**1. Introducción**

**1.1. Objetivo**

**Barajando entre varias ideas, he creado un sistema sencillo de servicios orientados a la domótica, que puedan ser accesibles desde fuera. Además, se hará uso de tecnologías de la actualidad como son los ordenadores en miniatura o Raspberry, DNS dinámico y la contenerizacion con Docker.**

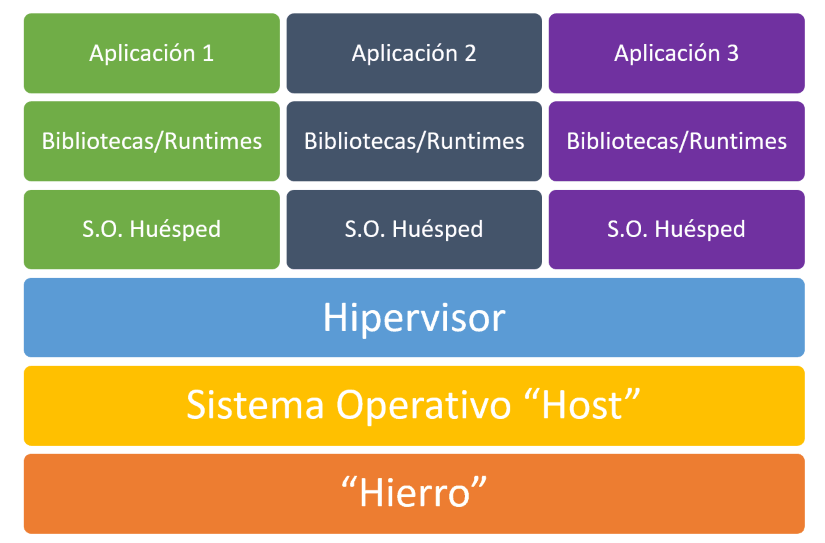
**Gracias a Raspberry, el sistema tiene una mayor portabilidad. Dentro de esta se montará un servidor LEMP. Una de las razones por las que he preferido montar un servidor LEMP antes que un servidor LAMP es porque, aunque solo cambia el servidor web, es un servidor no tan popular por no llevar apache. Aunque Nginx y Apache son iguales en eficiencia, Nginx no tiene tanta popularidad que Apache y es por esto que se usa mas Apache**

**Creación de un servidor LEMP.**

**Máquinas virtuales vs contenedores.**

**Máquinas virtuales.**

Como su propio nombre indica, una máquina virtual es un sistema operativo completo funcionando de manera aislada sobre otro sistema operativo completo.

Esta tecnología permite compartir el hardware de modo que lo puedan utilizar varios sistemas operativos al mismo tiempo.

*Esquema representativo de una máquina virtual.*

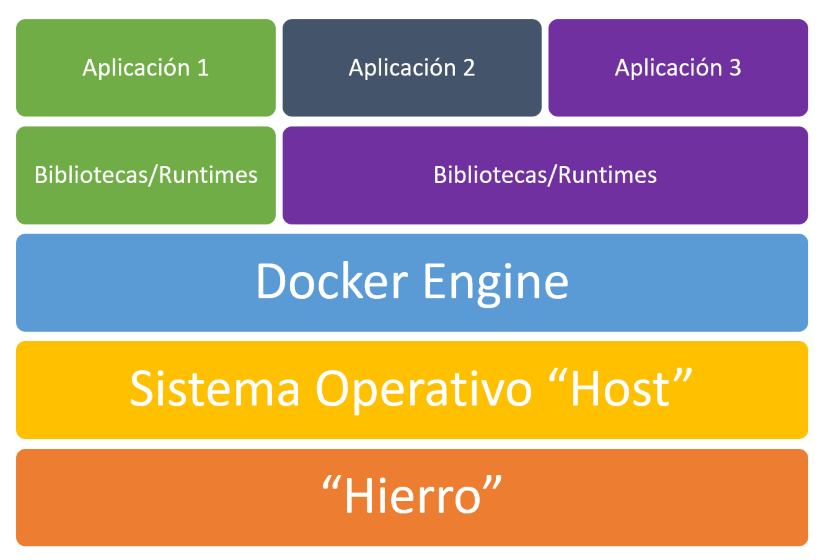
Para que las máquinas virtuales puedan ejecutarse es necesario instalar un componente por encima del sistema operativo, **el hipervisor**. Un hipervisor es un software especializado en exponer los recursos *hardware* que están debajo del sistema operativo, de modo que puedan ser utilizados por otros sistemas operativos.

Esto incluye las CPUs (o *cores*), la memoria y el espacio de almacenamiento además del resto del *hardware*. De este modo se pueden crear **máquinas virtuales** a las que se expone parte del *hardware* de la máquina anfitrión. Estas máquinas virtuales, “engañan" a un sistema operativo convencional para que crea que se está ejecutando sobre una máquina física. Los hipervisores vienen con productos como Hyper-V (incluido gratuitamente con Windows), VirtualBox o VMWare, entre otros.

Contra todo pronóstico, el avance de los hipervisores en los últimos años y las tecnologías orientadas a la virtualización que ofrecen los procesadores modernos, se está reduciendo la pérdida de rendimiento y aunque no sea la mas eficiente, es una manera de compartir hardware.

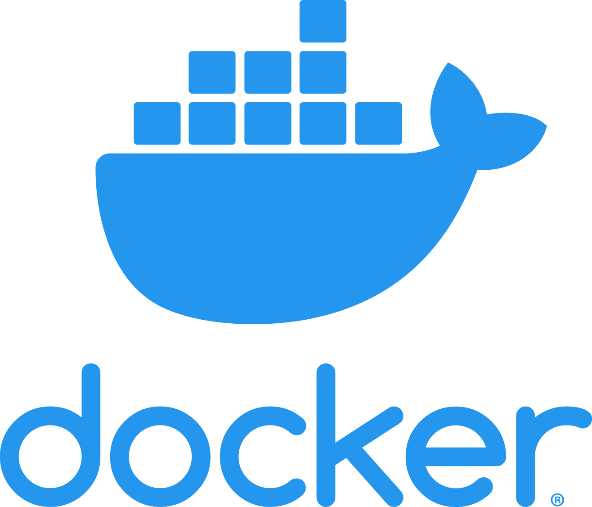
**Contenedores.**

El objetivo principal de los contenedores, es aislar aplicaciones y de generar un entorno replicable y estable para que funcionen, en lugar de albergar un sistema operativo completo (máquinas virtuales) compartiendo los recursos del propio sistema operativo "host" sobre el que se ejecutan. Una de las características principales, es la gestión de recursos entre todos los contenedores optimizando su uso y eliminando la necesidad de tener sistemas operativos separados para conseguir el aislamiento.



*Esquema funcionamiento contenedores.*

Existen numerosos servicios que utilizan esta tecnología (Virtuozzo, LXC/LXD, OpenVZ, DC/OS), pero para este proyecto utilizaremos Docker.



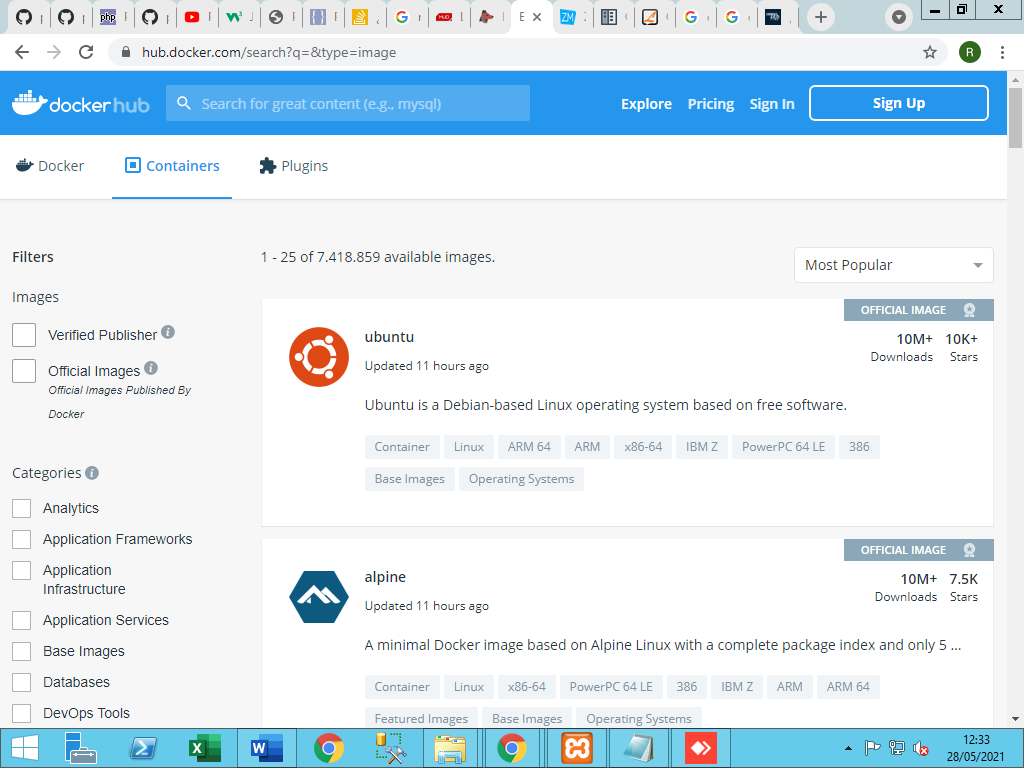
Docker nos permite crear, iniciar, detener, mover o eliminar contenedores. Puede conectar un contenedor a una o más redes, adjuntarle almacenamiento o incluso crear una nueva imagen basada en su estado actual.

De forma predeterminada, un contenedor está relativamente bien aislado de otros contenedores y de su máquina host. Puede controlar qué tan aislados están la red, el almacenamiento u otros subsistemas subyacentes de un contenedor de otros contenedores o de la máquina host.

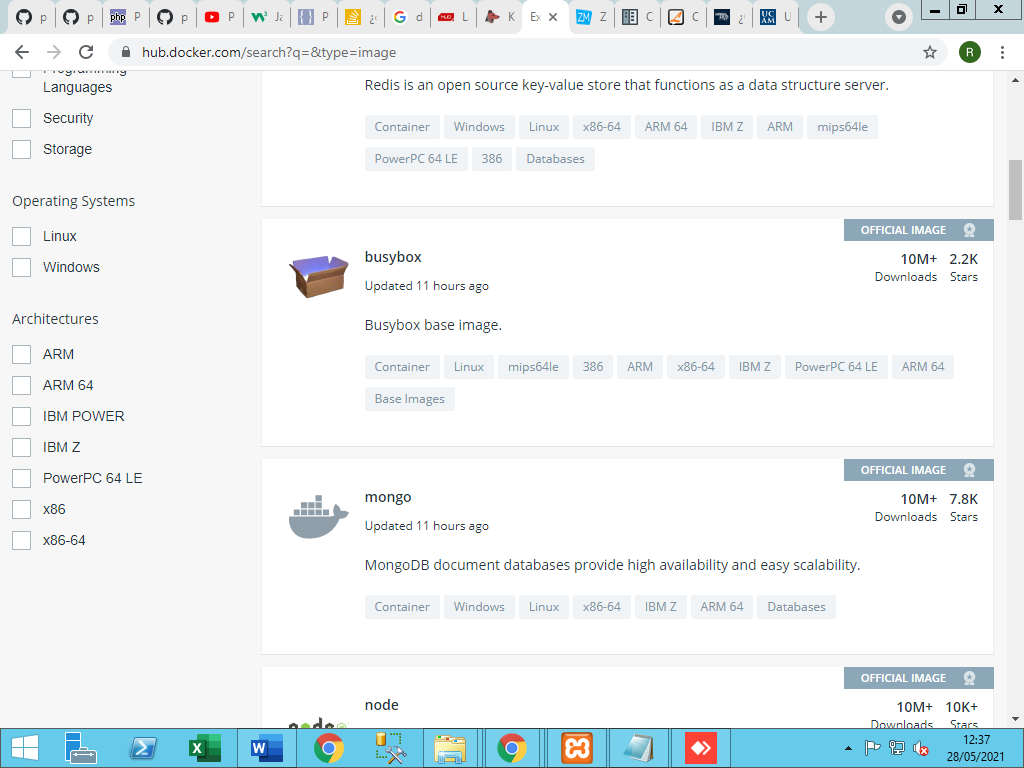
Un contenedor se define por su imagen, así como por las opciones de configuración que le proporcione al crearlo o iniciarlo. Cuando se quita un contenedor, cualquier cambio en su estado que **no esté almacenado en el almacenamiento persistente desaparece.**

Las imágenes que utiliza docker, son proveídas por servicios de repositorios de internet, por lo que hay que tener cuidado a la hora de descargarnos una imagen.

El repositorio por excelencia que usa la mayoría de los usuarios de docker es docker Hub. Docker Hub servicio de repositorio alojado proporcionado por Docker para buscar y compartir imágenes de contenedores con su equipo. Además, podemos crear repositorios de imágenes privados y crear automáticamente imágenes de contenedor desde GitHub y Bitbucket y enviarlos a Docker Hub.

*Repositorio de docker hub*

Una cosa a tener en cuenta es que la gran mayoría de estos contenedores está configurados y creados para trabajar en arquitecturas diferentes a las que usa una Raspberry (ARM/armhf). Por lo que, en caso de no encontrar una imagen para la Raspberry, tendremos las siguientes opciones.

* Buscar en otro repositorio de imágenes
* Crear una imagen con la arquitectura correcta
* Busca en Google y nos saldrán imágenes no oficiales en el mismo Docker Hub pues este solo nos enseña imágenes oficiales y las más populares.

*Filtro de tipo de arquitectura de las imágenes*

**Comparación de LAMP Y LEMP**

Para este proyecto voy a instalar un servidor LEMP (Linux, Nginx, MySql y PHP). Su otra gran alternativa es LAMP (Linux, Apache, Mysql y PHP). La única diferencia entre estos dos “stack’s” es el servidor web. Veamos una comparativa entre **Apache** (LAMP) y **Nginx** (LEMP).

**Apache** es uno de los stacks mas utilizados actualmente, la cual es de las mas robustas y la mas probada por los usuarios. En la red encontramos infinidad de documentación y cualquiera en poco tiempo puede montarlo.

Apache gestiona módulos multiproceso para mantener la misma conexión de diferentes maneras. Brindando como ventaja principal el hecho de que los administradores puedan aplicar criterios para cada conexión. Es capaz de procesar varios idiomas interpretados sin necesidad de conectarse a un software externo. Sin embargo, por su consumo, requiere de más recursos.

**NGINX** trabaja con una arquitectura asíncrona y controlada por eventos, esto significa que los hilos similares se administran bajo un proceso de trabajo, y cada proceso de trabajo contiene unidades más pequeñas llamadas conexiones de trabajo.

Toda esta unidad es la responsable de manejar los hilos de las solicitudes. Las conexiones de trabajo entregan las solicitudes a un proceso de trabajo, que también lo enviará a su turno al proceso maestro. Finalmente, el proceso maestro proporciona el resultado de esas solicitudes.

Puede parecer simple, pero una conexión de trabajo puede atender hasta 1024 solicitudes similares. Debido a eso, NGINX puede procesar miles de solicitudes sin ninguna dificultad. También es por eso que NGINX se convirtió en una excelente opción para sitios web con mucho tráfico como comercio electrónico, motores de búsqueda y almacenamiento en la nube.

**Docker**

**Como añadido, vamos a utilizar Docker y su tecnología de contenerizacion. Pero antes, ¿qué es Docker? Docker nos brinda la capacidad de empaquetar y ejecutar una aplicación en un entorno poco aislado llamado contenedor. El aislamiento y la seguridad le permiten ejecutar muchos contenedores simultáneamente en un host determinado.**

**Los contenedores son livianos y contienen todo lo necesario para ejecutar la aplicación, por lo que no necesita depender de lo que está instalado actualmente en el host. Puede compartir contenedores fácilmente mientras trabaja y asegurarse de que todas las personas con las que comparte obtienen el mismo contenedor que funciona de la misma manera.**

**Una de las razones por las que vamos a utilizar Docker es, por el menor consumo de recursos que al utilizar una maquina virtual, pues vamos a ejecutar nuestro servidor LEMP en una Raspberry la cual no tiene las mismas características que un pc ordinario.**

**INSTALCAION DE DOKCER EN RASPBERRY**

**En la página oficial de Docker podemos encontrar documentación acerca del proceso de instalación. La instalación es relativamente sencilla.**

**Lo primero de todo es actualizar nuestros repositorios y comprobar actualizaciones.**

sudo apt-get update

sudo apt-get upgrade

**Posteriormente instalamos estos paquetes.**

sudo apt-get install -y \

apt-transport-https \

ca-certificates \

curl \

gnupg2 \

software-properties-common \

vim \

fail2ban \

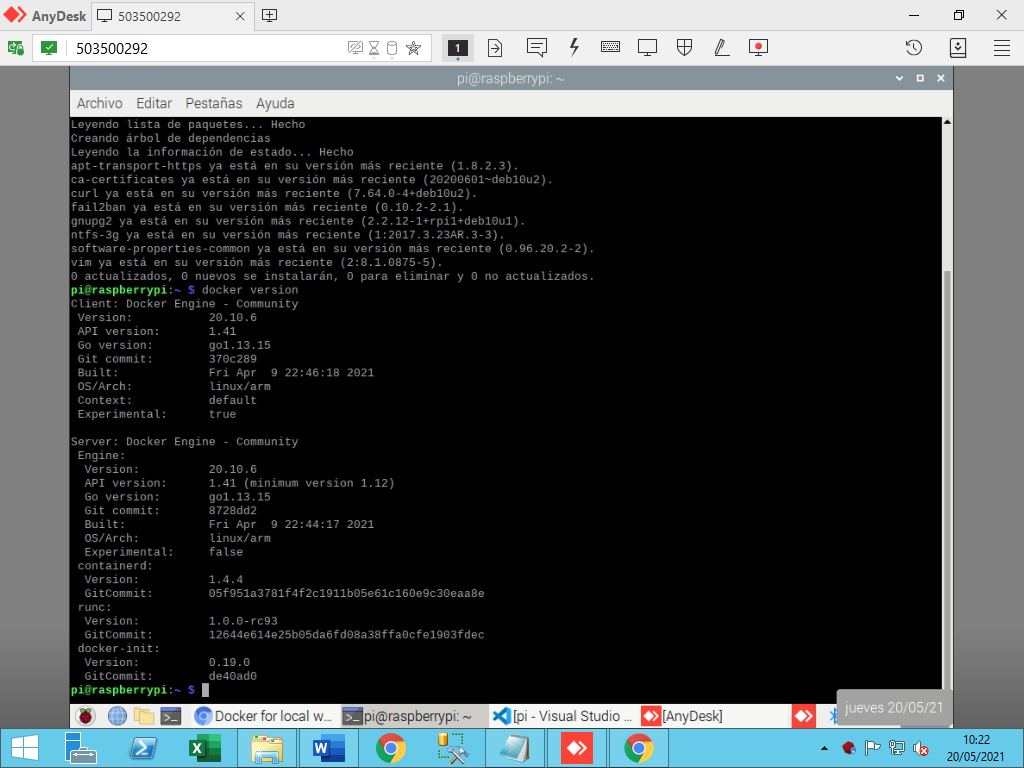
ntfs-3g

**Después ejecutamos el siguiente comando con curl para obtener el script de instalación de Docker.**

curl -sSL https://get.docker.com | sh

**Agregamos nuestro usuario del sistema al grupo docker. Por lo tanto, podrá crear y administrar contenedores, imágenes, volúmenes, etc. de Docker sin privilegios de sudo o superusuario.**

sudo usermod -aG docker pi

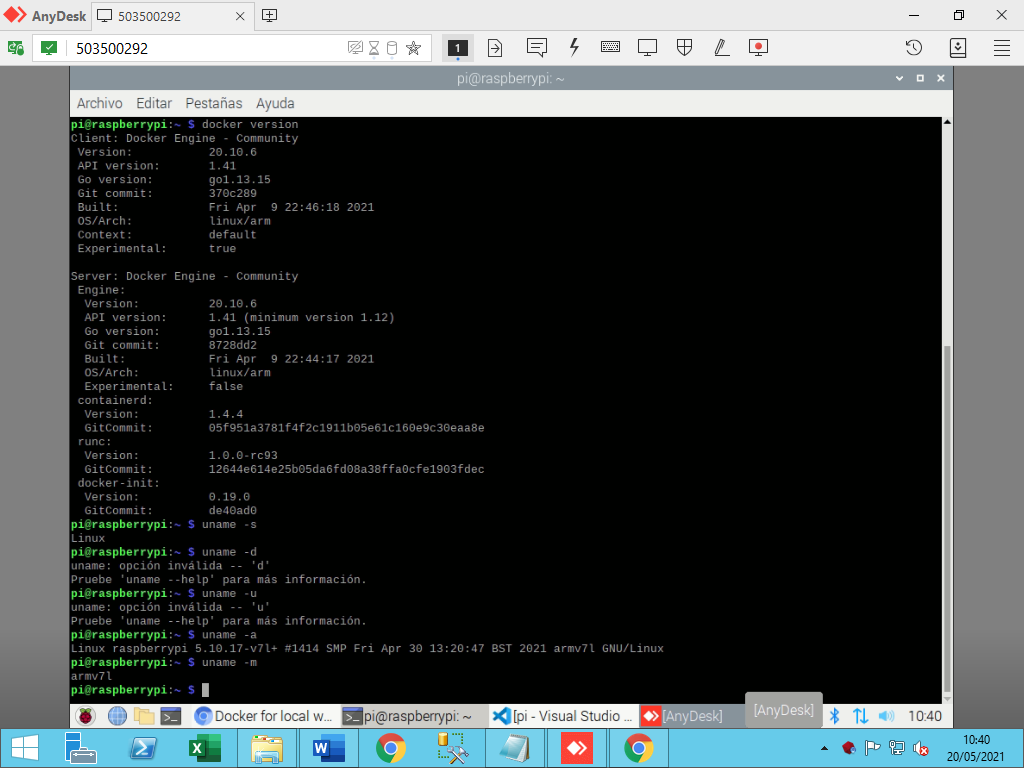
Comprobamos con el comando docker version para comprobar que se ha instalado correctamente.

**INSTALACION DE DOCKER-COMPOSE**

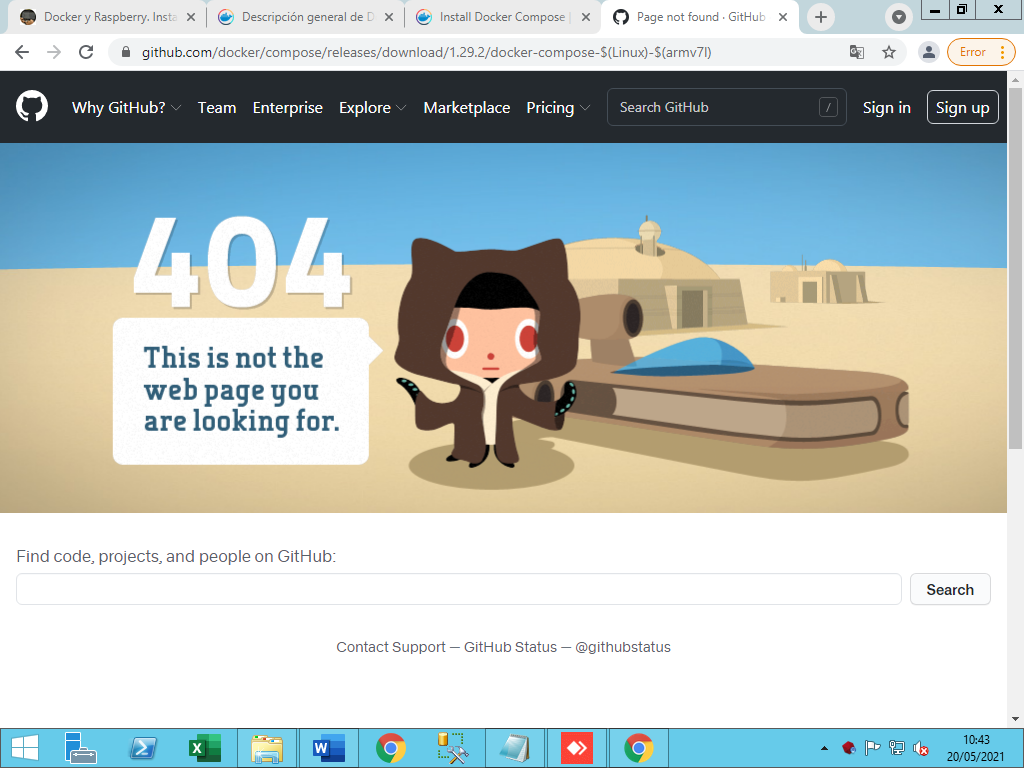
**Ahora vamos instalar docker-compose. Docker-compose es una herramienta para definir, ejecutar y, sobre todo, administrar aplicaciones Docker de varios contenedores pues nos muestra la configuración de todos los contenedores en un solo archivo de extensión YAML.**

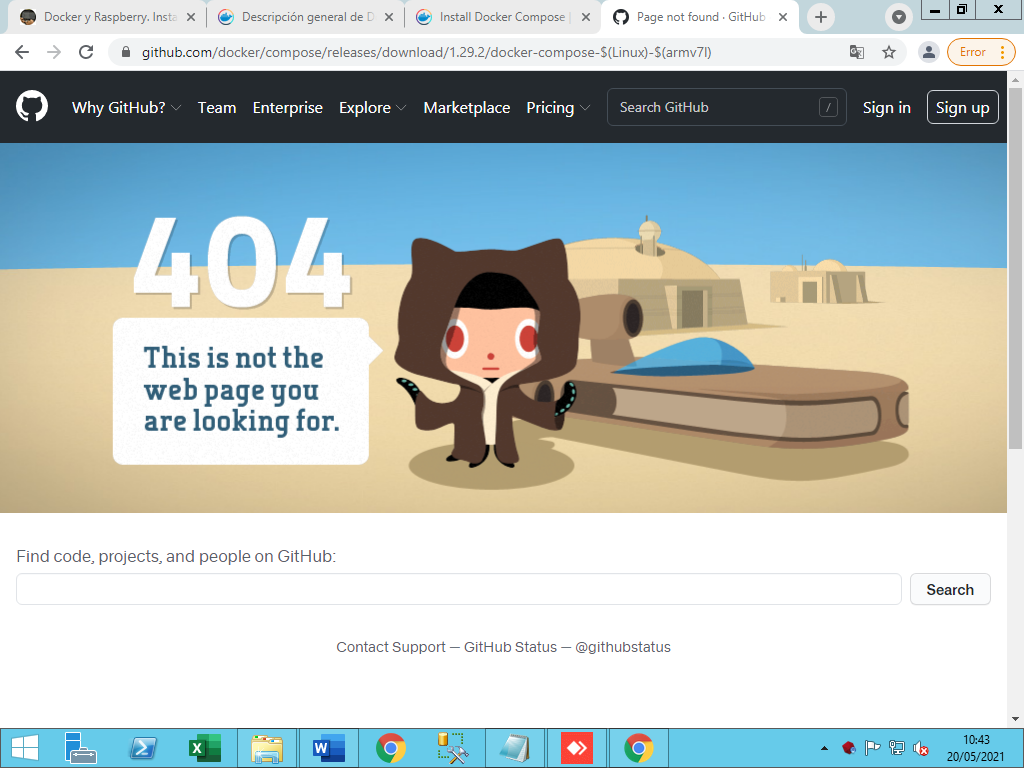
**En la documentación oficial, nos muestra el siguiente comando para su instalación.**

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

**Sin embargo, a este comando se le esta pasando un link con nuestro sistema operativo y nuestra arquitectura la cual no esta disponible en su repositorio de GitHub.**

**Armv7l es la version de nuestro kernel y por consiguiente curl no puede resolver el link compuesto con nuestras variables.**





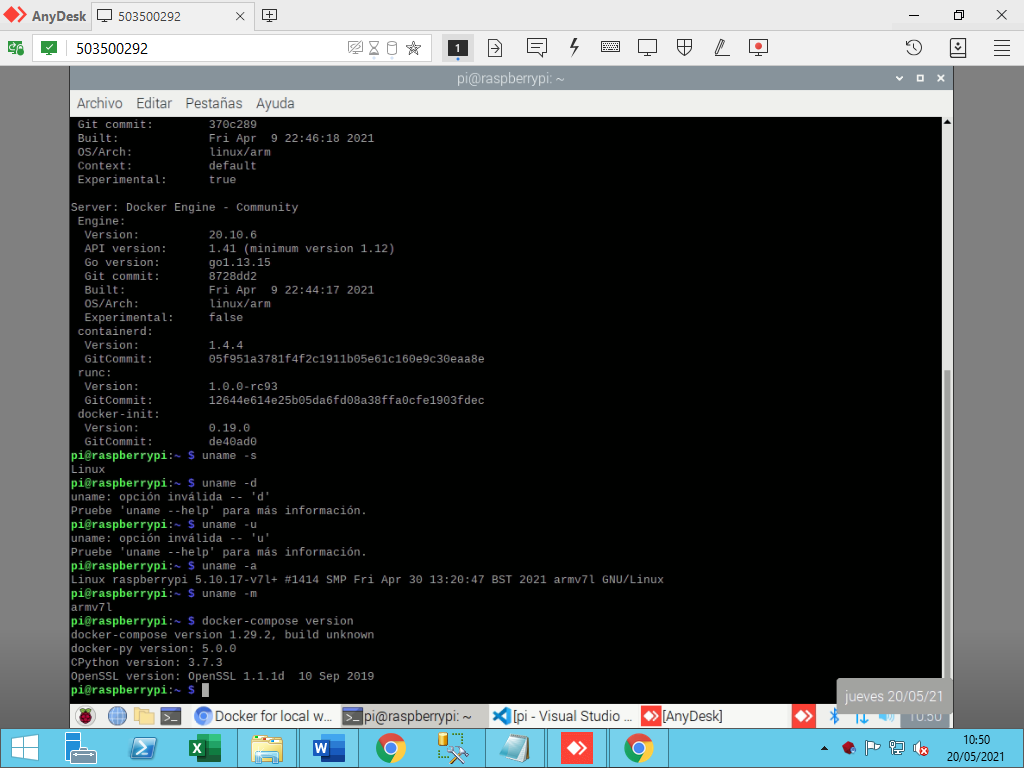
**Por lo que pasamos a la segunda opción, que es instalarlo por pip de Python pues docker esta escrito en su mayoría en este lenguaje. Descargamos los** **siguientes paquetes.**

sudo apt install -y libffi-dev libssl-dev python3-pip

**Instalamos docker-compose con el comando pip3.**

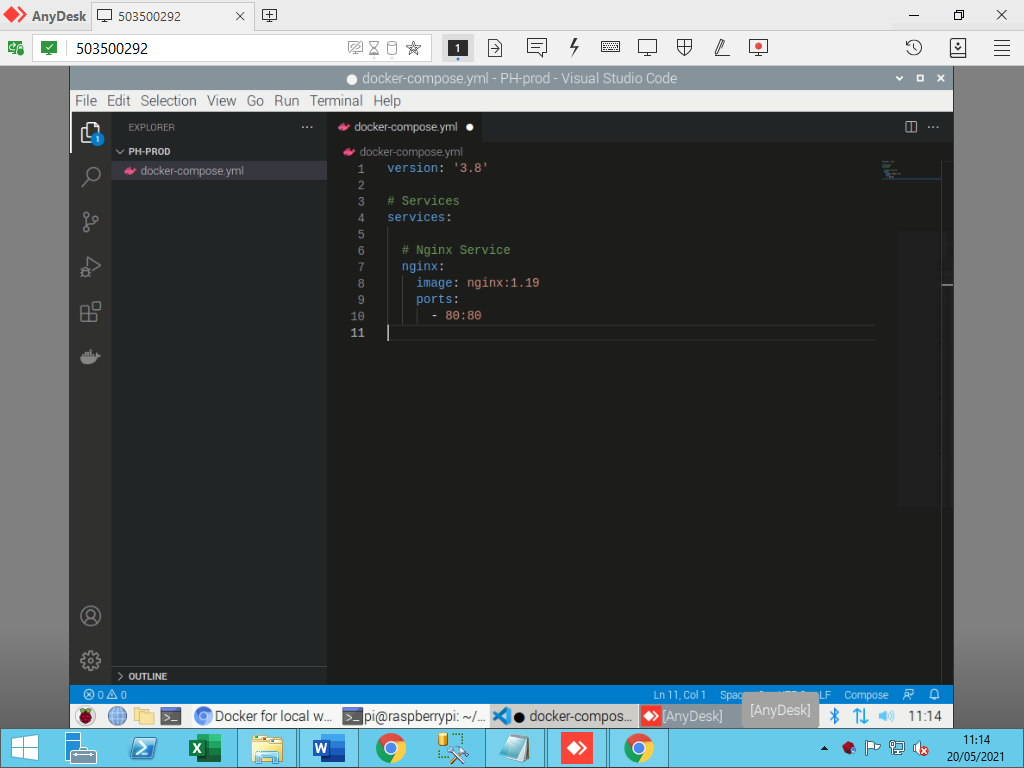
sudo pip3 install docker-compose

**Esperamos uno segundos y comprobamos la version de docker-compose con el comando** docker-compose version **para verificar que se ha instalado de correctamente.**



**CONFIGURACION DE LOS CONTENEDORES EN ARCHIVO.YML**

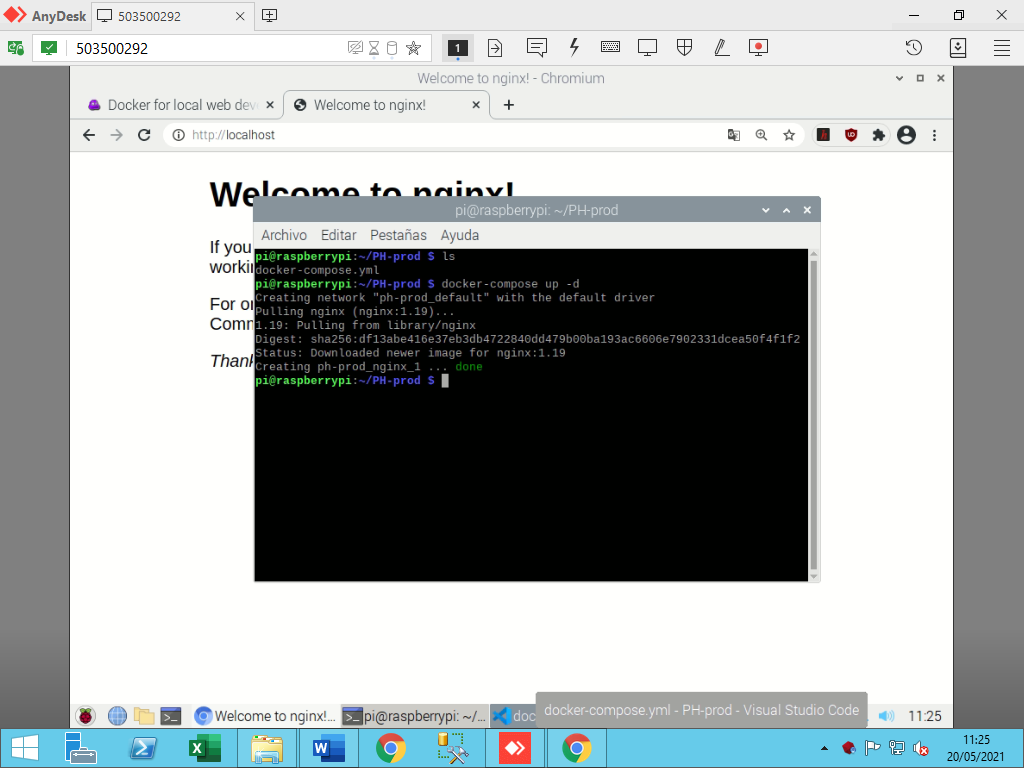
**Empezaremos con el contenedor de Nginx. Antes, especificamos la version de docker-compose que vamos a utilizar y posteriormente una línea para especificar los servicios que se van a desplegar. En el contenedor de Nginx, especificamos la version de imagen que vamos a utilizar y los puertos en modo “bridge” que vamos a utilizar. Las imágenes son descargadas automáticamente desde Docker Hub. No usaremos la ultima version del contenedor “lasted” pues estas versiones son muy recientes y no sabremos su comportamiento al ser tan nuevas.**



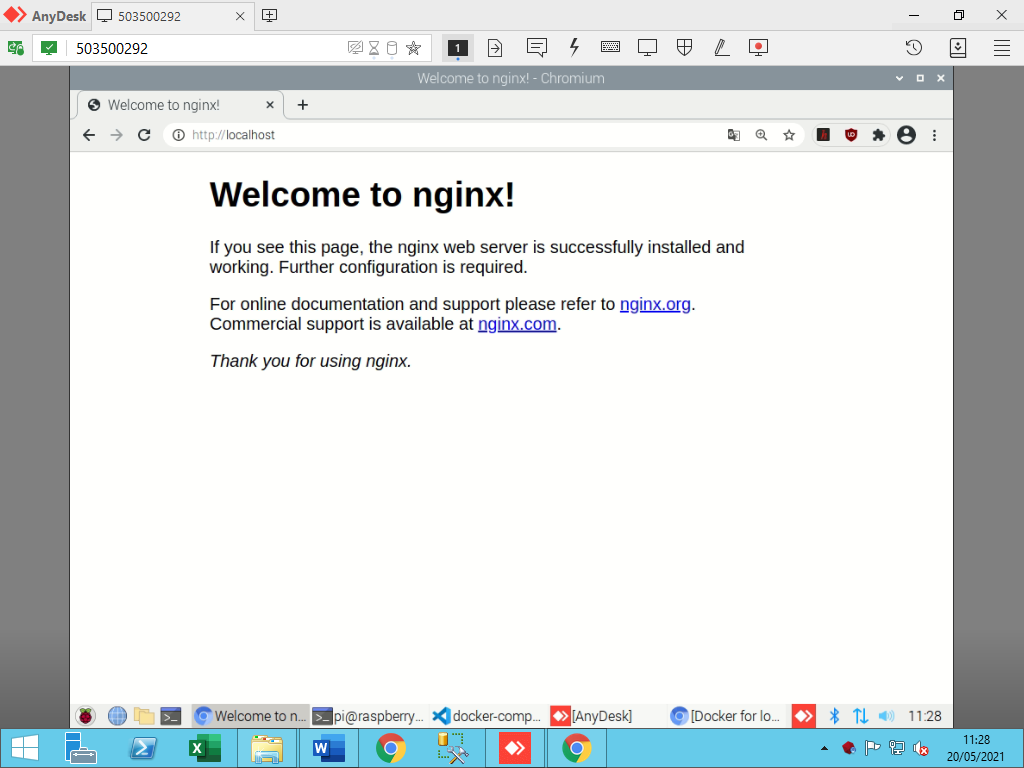
**Levantamos docker-compose con el siguiente comando.**

docker-compose up -d

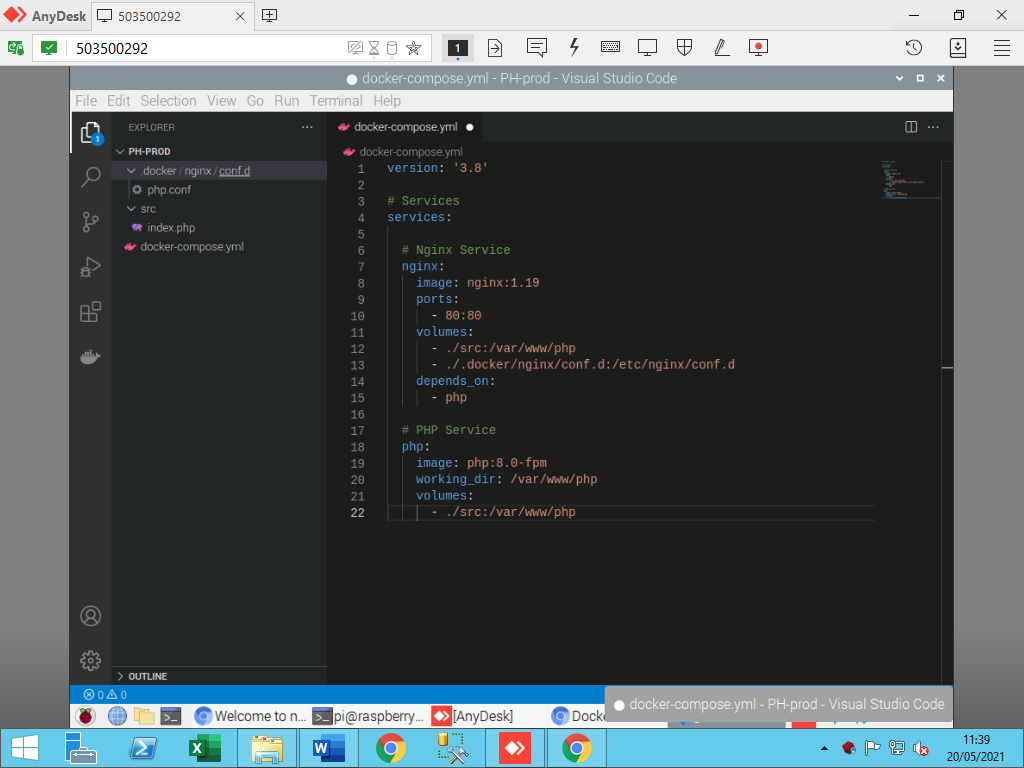
Vemos que crear un nuevo contenedor descargado de Docker Hub



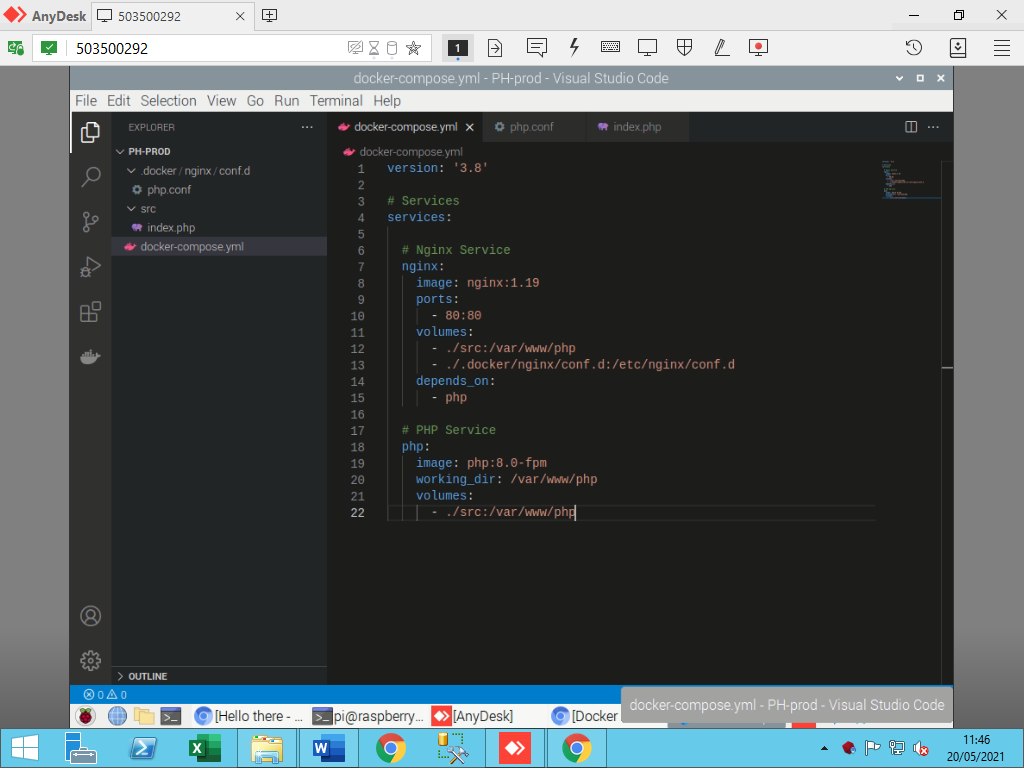
Vamos a nuestro localhost para comprobar que funciona.



Creamos la siguiente estructura de directorios.

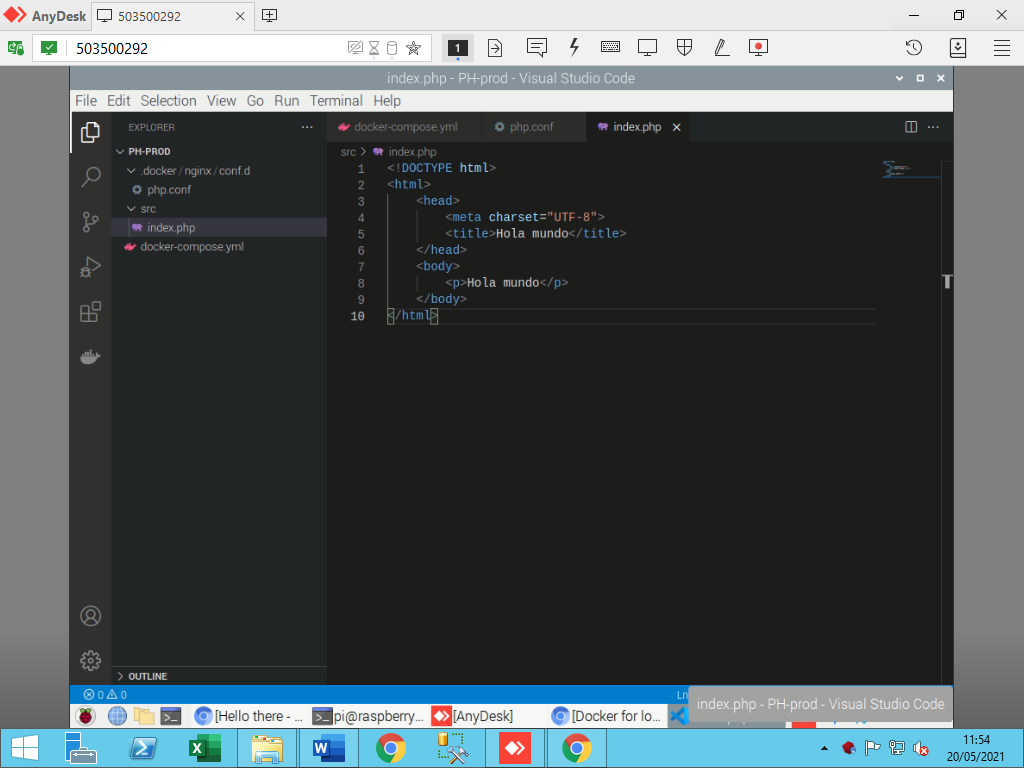


Volvemos a editar el archivo .yml.

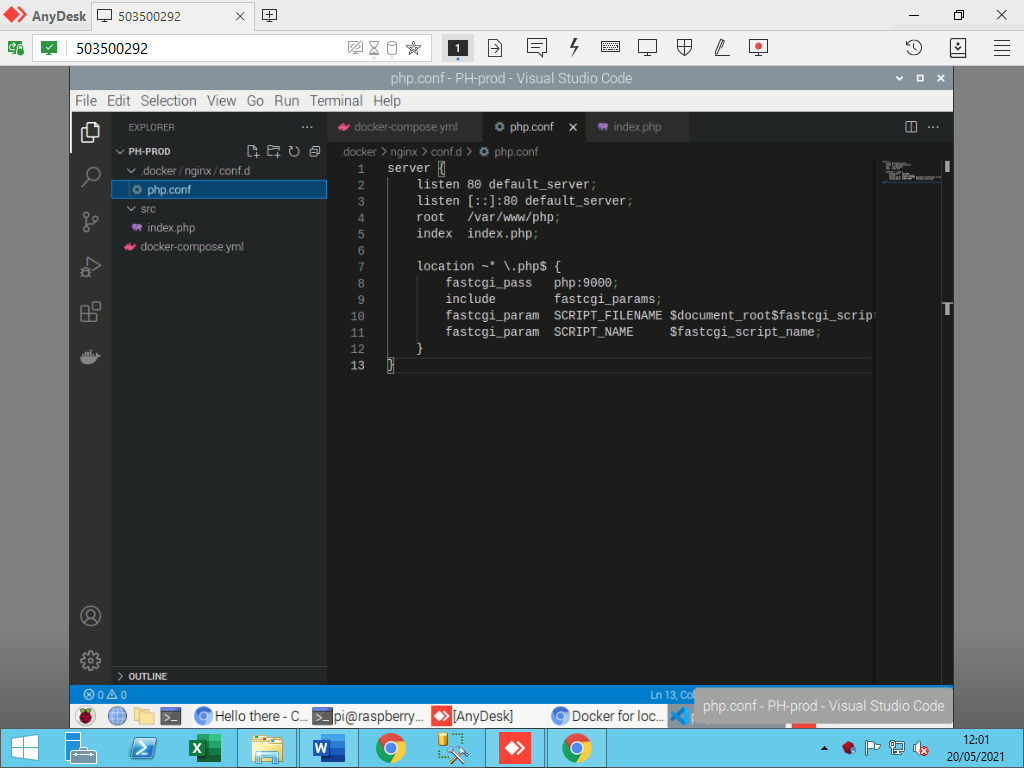


Veremos que el contenedor de php tiene una sección volúmenes los cuales serán montados dentro del contenedor, es decir a la izquierda el directorio local y a la derecha el directorio donde se va a montar dentro del contenedor. Por último, el “depens\_on” establece un orden de inicio de contenedores por lo que asegura que el contenedor PHP se iniciará antes que el de Nginx

En la carpeta .src/ es donde vamos a colocar el código de nuestra aplicación. Dentro de el crearemos un index.php para mostrar un texto de prueba.

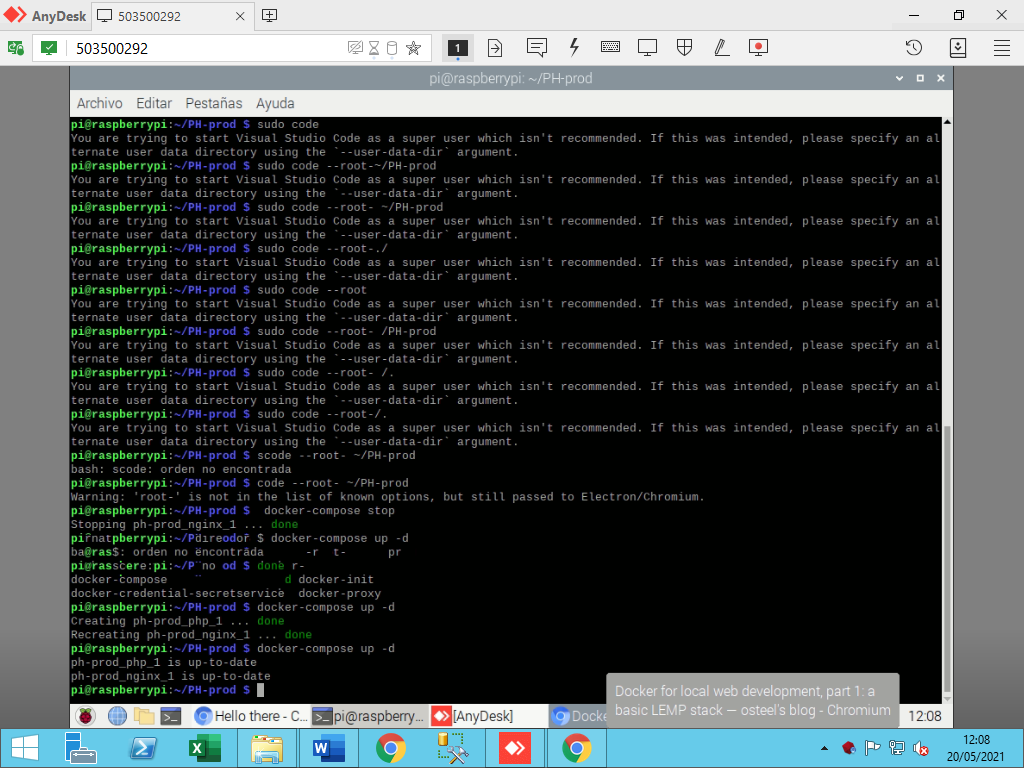


Ahora vamos a los volúmenes agregados al contenedor de Nginx y le montamos la siguiente carpeta donde le añadiremos su correspondiente configuración.

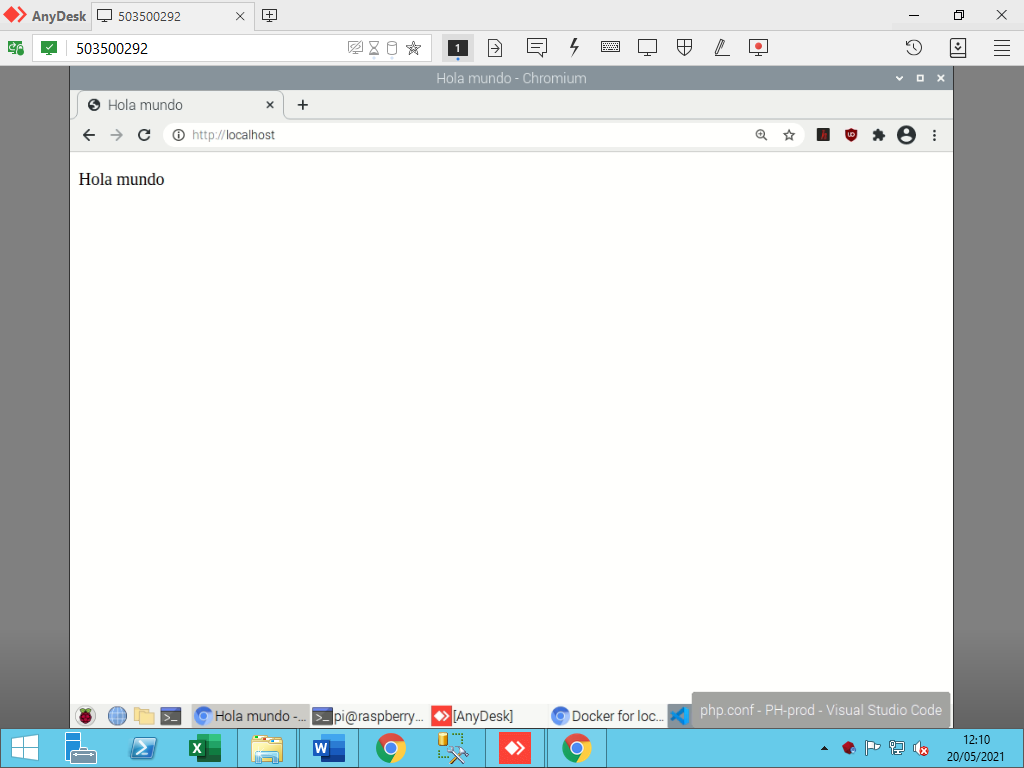


A destacar de esta configuración, la línea 8 donde Nginx reenvía las solicitudes de archivos PHP al puerto 9000 del contenedor PHP, que es el puerto predeterminado en el que escucha PHP-FPM.

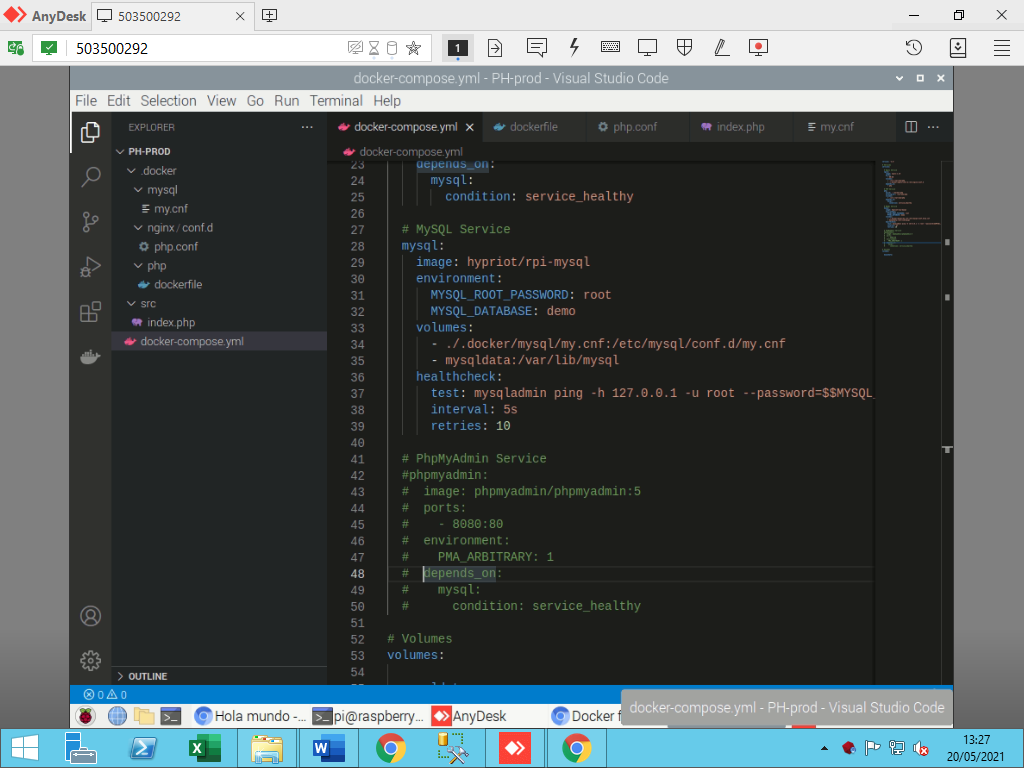
Volvemos a levantar el archivo de dokcer-compose.yml para actualizar todos los archivos.



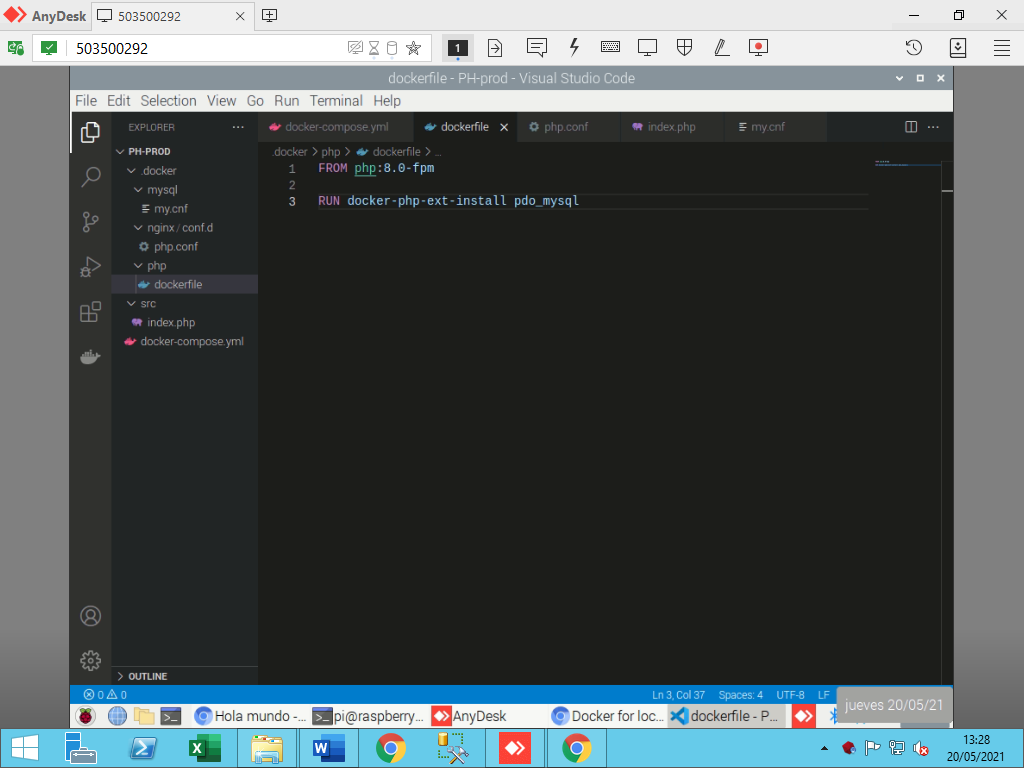
Actualizamos la pagina <http://localhost:80> y se nos mostrara lo que escribimos en nuestro index.php.



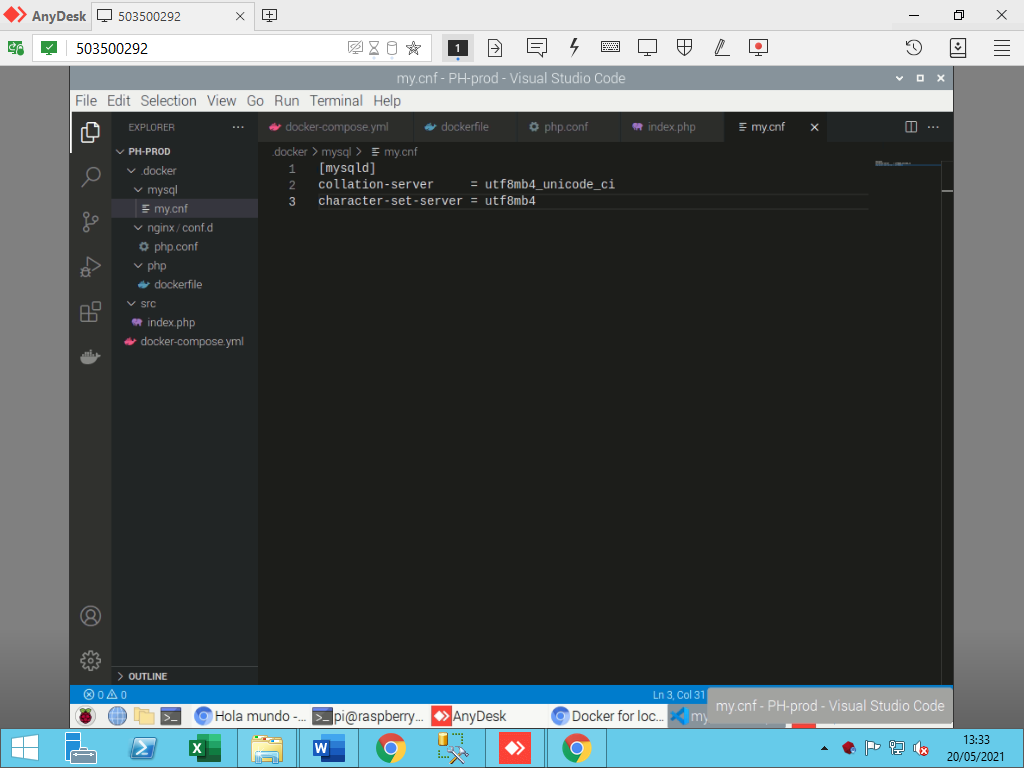
Por último, vamos a instalar MySql como motor de búsquedas. Como los contenedores de docker hub de Mysql no soportan la arquitectura arm, he tenido que descargar una imagen no oficial.



Después creamos el siguiente dokcerfile. Este archivo, permite a PHP utilizar la pdo\_mysql extensión para leer desde una base de datos MySQL.



En la configuración de Mysql (my.cnf) añadimos lo siguiente.



Esto establecerá el juego de caracteres utf8mb4\_unicode\_cipor defecto, que es bastante estándar hoy en día.

Probamos a meternos al contenedor de Mysql con un bash con el siguiente comando.

docker-compose exec mysql bash

Vemos que funciona perfectamente el servidor Mysql

