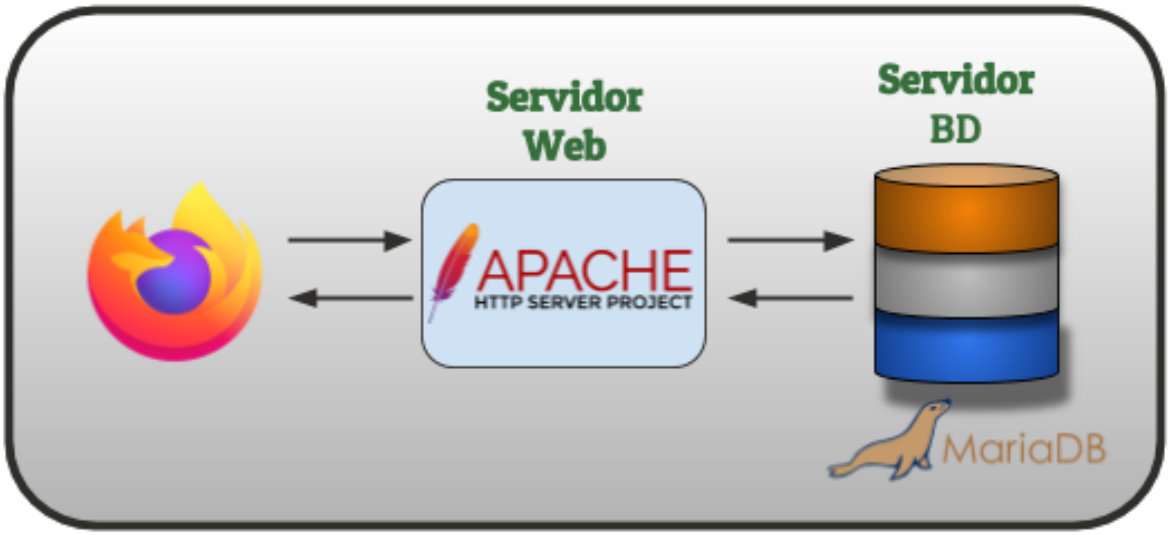
7. Aplicaciones multicapa con docker-compose

Hasta ahora hemos estado hablando de **contenedores en solitario** pero la realidad es que **las aplicaciones actuales están formadas de varias aplicaciones o servicios. TÍPICAMENTE** podríamos decir que tenemos **al menos los siguientes elementos**:



Juan Diego Pérez Jiménez. *Arquitectura típica de una aplicación multi-capa* (Dominio público)

- Una **CAPA DE PRESENTACIÓN** que hace referencia al cliente que obtiene los datos pudiendo ser un **navegador, una app móvil** etc..
- Una **CAPA LÓGICA** que típicamente está representada por un **servidor web, servidor de aplicaciones** etc..
- Una **CAPA DE DATOS** que reside normalmente en un **servidor de base de datos**, ya sea relacional o no.

Además **nos podemos encontrar aplicaciones con arquitecturas mucho más complejas** con diversos servidores de bases de datos, distintos APIs independientes, servidores de autentificación etc... Aunque estas aplicaciones existen, lo cierto es que **para el objetivo del curso nos vamos a conformar con aplicaciones como las descritas anteriormente**, con una capa de presentación, una capa de lógica y una capa de datos.

Precisamente ya vimos por encima en el módulo anterior un ejemplo de este tipo de aplicaciones donde teníamos:

- Un contenedor con un servidor web Apache, con el módulo PHP instalado y el código de Wordpress descargado en la carpeta adecuada.
- Un contenedor con un servidor de base de datos MariaDb para guardar los datos de la aplicación.

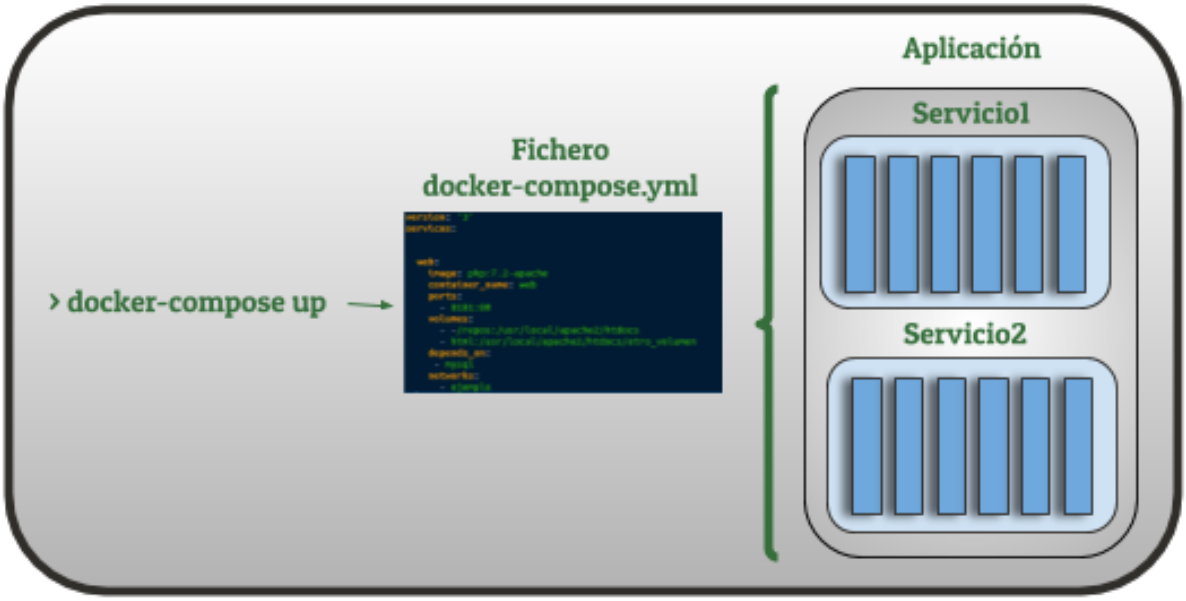
Esos contenedores **los habíamos creado y configurado por separado aunque conformaban una única aplicación.** Tenía que **arrancarlos y configurarlos** uno a uno **de manera manual**. Eso no es ideal y además no es ágil, cada vez que quiero poner en funcionamiento este tipo de arquitectura deberé repetir todo el proceso de manera paso a paso. **LO IDEAL** sería:

- **Hacer todo de manera declarativa** para que no tenga que repetir todo el proceso cada vez.
- Poner en funcionamiento **todos los contenedores** que necesita mi aplicación **de una sola vez** y debidamente configurados.
- Garantizar que los contenedores se arrancan **en el orden adecuado**. Por ejemplo: Mi aplicación no podrá funcionar debidamente hasta que no esté el servidor de bases de datos funcionando en marcha.
- **Asegurarnos de que hay comunicación** entre los contenedores que pertenecen a la aplicación.

Para todo esto tenemos la herramienta **DOCKER-COMPOSE.** Si tuviéramos que definir esta herramienta diríamos:

"DOCKER-COMPOSE ES UNA HERRAMIENTA PARA DESPLEGAR GRUPOS DE CONTENEDORES QUE FORMAN PARTE DE UNA MISMA APLICACIÓN O UN MISMO ENTORNO"

El proceso para conseguir esto se describe de manera general en la siguiente imagen:



Juan Diego Pérez Jiménez. *Proceso general de docker-compose* (Dominio público)

Los pasos son los siguientes:

1. **Describo de manera declarativa** todo los contenedores que conforman mi aplicación en el fichero **docker-compose.yml.** Este fichero es un un fichero con formato YAML.
2. Al ejecutar **docker-compose up** se levanta toda la aplicación, es decir, todos los contenedores que la conforman.

Profundizaremos en el contenido de ese fichero y en las posibilidades más relevantes que nos proporciona la herramienta docker-compose a lo largo de este módulo.

7.1 Instalación de docker-compose

La instalación de docker-compose es un proceso muy sencillo. Si somos usuarios de MAC y Windows no tendremos que instalar nada ya que docker-compose es una de las herramientas que por defecto se incluyen dentro de Docker Desktop. La instalación de Docker Desktop ya la vimos para Windows 10 en el módulo 1 de este mismo curso.

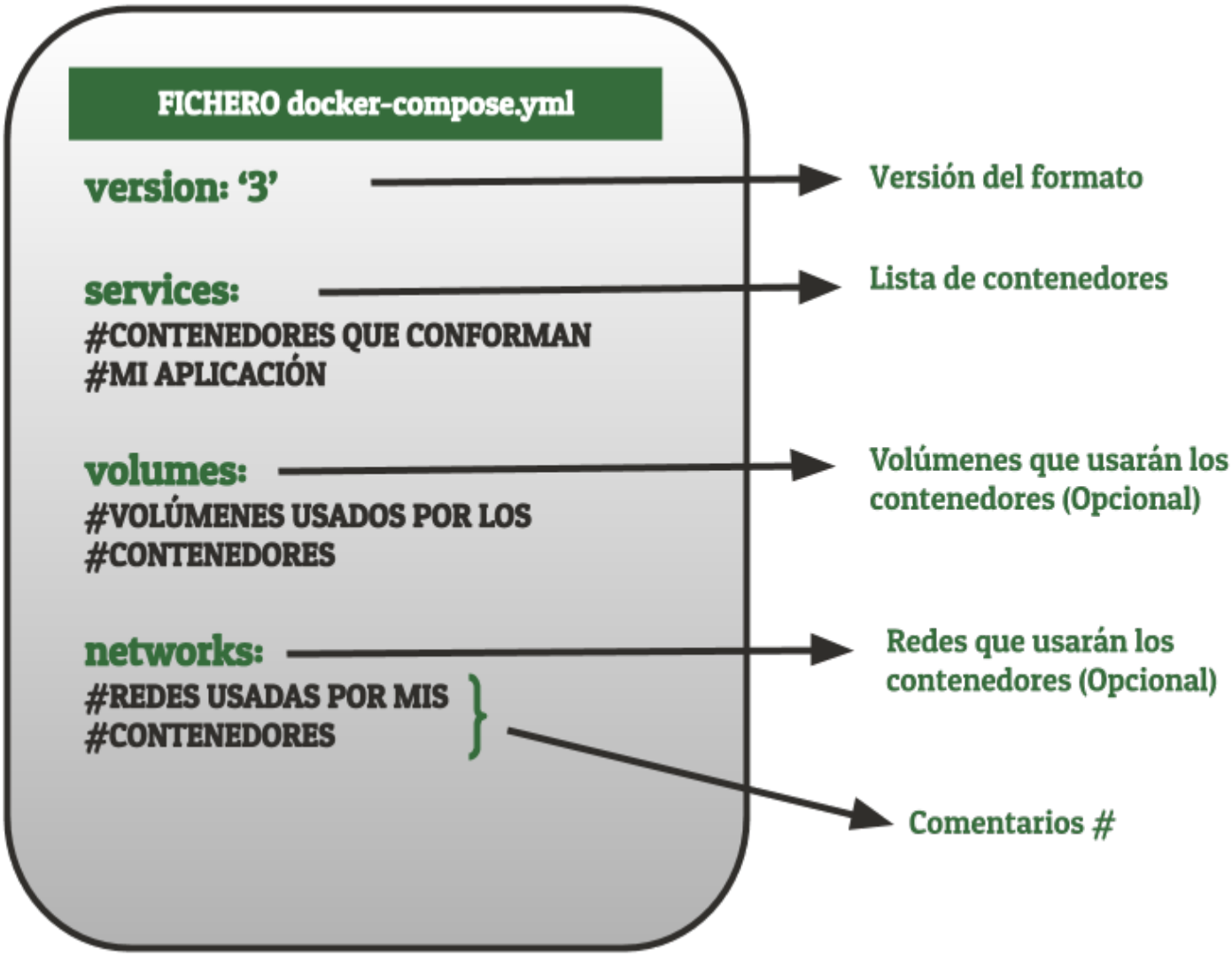
Si somos usuarios de Linux su instalación se realiza únicamente con dos pasos:

```
# Descarga del fichero mediante la orden curl y colocación en el directorio adecuado. Actualmente (Enero 2021) la versión vigente es la 1.27.4

> sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# Concesión de los permisos de ejecución

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

# Comprobación de que la instalación está correcta.

> docker-compose --version

docker-compose version 1.27.4, build 1110ad01
```

IMPORTANTE: EVIDENTEMENTE PARA QUE FUNCIONE DOCKER-COMPOSE TENEMOS QUE TENER DOCKER INSTALADO DE MANERA PREVIA.

7.2 El archivo docker-compose.yml

Como ya hemos dicho en el apartado anterior el fichero docker-compose.yml es un fichero en formato YAML que contiene las instrucciones para crear y configurar los servicios que van a constituir mi aplicación o entorno. Su estructura general es la siguiente:



Juan Diego Pérez Jiménez. *Estructura general de un docker-compose.yml* (Dominio público)

Dentro de cada una de estas secciones tenemos multitud de opciones. Es imposible tratar todas de manera detallada así que nos vamos a centrar en ejemplos para conseguir:

- Asociar contenedores e imágenes a los servicios creados.
- Especificar las distintas variables de entorno que pueden tener los contenedores creados. Recordad que hay que consultar siempre la página de cada imagen en DockerHub.
- Establecer las redirecciones de puertos si fueran necesarias.
- Persistir los datos de los contenedores usando bind mounts o volúmenes . Definiremos los volúmenes, si es necesario, para que sean usados por los contenedores.
- Definir redes para asociarlas a los contenedores en caso de que queramos que no usen la red por defecto. En el módulo 5 hablamos de las diferencias entre la red por defecto y las redes creadas por el usuario.
- Establecer orden de inicio para los contenedores que componen mi aplicación.

ASOCIAR CONTENEDORES A SERVICIOS

```
version: '3'
```

```
 # Empieza la sección de servicios.

 services:

   # Declaro un servicio con nombre miapache

   miapache:

    # Ese contenedor usará como imagen de base la imagen httpd(Servidor Apache) de DockerHub.

     image: httpd

     # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)

     container_name: web
```

ESPECIFICAR LAS VARIABLES DE ENTORNO PARA LOS CONTENEDORES

```
version: '3'

 # Empieza la sección de servicios.

 services:

   # Declaro un servicio con nombre datos

   datos:

    # Ese contenedor usará como imagen de base la imagen mariadb (Servidor de base de datos) de DockerHub.

     image: mariadb

     # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)

     container_name: bd

     # Establezco las variables de entorno para configurar el servicio

     environment:

       MYSQL_ROOT_PASSWORD: 123456

       MYSQL_DATABASE: test
```

```
      MYSQL_USER: pepe

      MYSQL_PASSWORD: pepe
```

ESTABLEZCO LA REDIRECCIÓN DE PUERTOS SI FUERA NECESARIO

```
version: '3'

 # Empieza la sección de servicios.

 services:

   # Declaro un servicio con nombre miapache

   miapache:

    # Ese contenedor usará como imagen de base la imagen httpd(Servidor Apache) de DockerHub.

     image: httpd

    # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)

     container_name: web

    # Establezco la redirección de puertos

     ports:

      - 8080:80
```

PERSISTO LOS DATOS USANDO UN BIND MOUNT PARA EL SERVIDOR WEB

```
version: '3'

 # Empieza la sección de servicios.

 services:

   # Declaro un servicio con nombre miapache

   miapache:

    # Ese contenedor usará como imagen de base la imagen httpd(Servidor Apache) de DockerHub.
```

```
    image: httpd

    # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)

    container_name: web

    # Establezco un bind bound de la carpeta src de mi equipo en la carpeta /app del contenedor.

    volumes:

     - "./src:/app"

     # Notación alternativa a lo anterior

     - type: bind

       source: "./src"

       target: /app
```

PERSISTIR LOS DATOS PARA QUE EL SERVIDOR DE BASE DE DATOS USE UN VOLUMEN

```
version: '3'

 services:

   # Declaro un servicio con nombre datos

   datos:

    # Ese contenedor usará como imagen de base la imagen mariadb (Servidor de base de datos) de DockerHub.

     image: mariadb

    # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)

    container_name: bd

    .........

    # Establezco que los datos de la base de datos van a persistir en el volumen
```

```
datosapp que se montará en /var/lib/mysql
    volumes:
     - "datosapp:/var/lib/mysql"
    # Notación alternativa a lo anterior
    volumes:
     - type: volume
       src: datosapp
       target: "var/lib/mysql"
# Sección para la definición de los volúmenes. Está al mismo nivel de la sección services
 volumes:
  datosapp: local
```

DEFINIR REDES PARA ASOCIARLAS A LOS CONTENEDORES

```
version: '3'
# Empieza la sección de servicios.
 services:
  # Declaro un servicio con nombre miapache
  miapache:
   # Ese contenedor usará como imagen de base la imagen httpd(Servidor Apache) de DockerHub.
    image: httpd
    # Le doy nombre al contenedor cuando arranque (equivalente al flag --name de docker un)
    container_name: web
    # Establezco el nombre de red para el contenedor
```

```
    hostname: web
    # Establezco la red o redes a las que se va a conectar el contenedor
    networks:
      - ejemplo
...
# Sección de definición de redes. Está al mismo nivel que services y volumes
networks:
 # Definición de la red ejemplo
 ejemplo:
  # Tipo de red
  driver: bridge
  # Opciones de la red
  ipam:
    driver: default
    config:
      subnet: 172.20.0.0/16
```

ESTABLECER ORDEN DE INICIO DE LOS CONTENEDORES

```
version: '3'

# Empieza la sección de servicios.

 services:

  # Declaro un servicio con nombre miapache

  miapache:

   # Ese contenedor usará como imagen de base la imagen httpd(Servidor Apache) de DockerHub.

    image: httpd

    ....

   # Este contenedor arrancará después el contenedor del servicio datos

    depends_on:
      - datos

 # Comienzo de descripción del servicio de datos

  datos:

   image: mariadb

    ....
```

NOTA: LA NOTACIÓN YAML ESTABLECE 2 ESPACIOS PARA LA TABULACIÓN DE LOS DISTINTOS NIVELES:

EJEMPLO COMPLETO

docker-compose.yml que une todo lo expuesto anteriormente:

```
version: '3'
services:
#------------------------------------------------------------------------
----
# SERVICIO SERVIDOR WEB (php:7.4-apache con mysqli y código WP descargado
#------------------------------------------------------------------------
----
web:
# IMAGEN USADA
```

```
image: jperjim398/miwp
# NOMBRE QUE LE VOY A DAR AL CONTENEDOR
container_name: web
# REDIRECCIÓN DE PUERTOS
ports:
- 8181:80
# SERVICIOS QUE TIENEN QUE ARRANCAR ANTES DE ARRANCAR ESTE
depends_on:
- datos
# REDES A USAR
networks:
- ejemplo
#----------------------------------------------
--
# SERVICIO SERVIDOR DE BASE DE DATOS MARIADB
#----------------------------------------------
--
# NOMBRE DEL SERVICIO
datos:
# IMAGEN USADA
image: mariadb
# NOMBRE QUE LE VOY A DAR AL CONTENEDOR
container_name: bd
# LISTA DE VALORES DE ENTORNO CON SUS VALORES
environment:
MYSQL_ROOT_PASSWORD: 123456
MYSQL_DATABASE: wordpress
MYSQL_USER: pepe
MYSQL_PASSWORD: pepe
# REDIRECCIÓN DE PUERTOS
ports:
- 3316:3306
# VOLÚMENES A USAR POR EL CONTENEDOR
volumes:
# DOCKER VOLUME
- data:/var/lib/mysql
# REDES A USAR
networks:
- ejemplo
# DEFINICIÓN DE VOLÚMENES DOCKER A USAR POR LOS SERVICIOS (OPCIONAL)
volumes:
data:
# TIPO DE DRIVER
```

```
driver: local

# DEFINICIÓN DE LAS REDES A USAR POR LOS SERVICIOS (OPCIONAL)
networks:
# NOMBRE DE LA RED
ejemplo:
# DRIVER DE LA RED
driver: bridge
ipam:
driver: default
config:
- subnet: 172.20.0.0/16
```

REFERENCIA COMPLETA docker-compose.yml

Podemos encontrar una referencia completa sobre el fichero docker-compose.yml en la documentación oficial: https://docs.docker.com/compose/compose-file/compose-file-v3/

Y una extensión para manejar este tipo de ficheros en Visual Studio Code: https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml

7.3 La orden docker-compose

PUESTA EN MARCHA DE LOS SERVICIOS

Una vez hemos creado el archivo docker-compose.yml tenemos que empezar **a trabajar con él**, es decir a crear los contenedores que describe su contenido.

Esto lo haremos **mediante el ejecutable docker-compose**.**ES IMPORTANTE DESTACAR QUE DEBEMOS INVOCARLA DESDE EL DIRECTORIO EN EL QUE SE ENCUENTRA EL FICHERO docker-compose.yml.**

Esta herramienta (docker-compose) tiene muchos subcomandos y estos subcomandos, a su vez, tienen u montón de opciones. Nos vamos a centrar en ejemplificar con las combinaciones más comunes y además pondremos una lista de aquellas que también pueden resultar interesantes:

```
#Obtener la versión de docker-compose.

> docker-compose --version

# Crear los contenedores (servicios) que están descritos en el docker-compose.yml.
```

```
> docker-compose up

# Crear en modo detach los contenedores (servicios) que están descritos en el docker-compose.yml. Eso significa que no muestran mensajes de log en el terminal y que se  nos vuelve a mostrar un prompt.

> docker-compose up -d

# Detiene los contenedores que previamente se han lanzado con docker-compose up.

> docker-compose stop

# Inicia los contenedores descritos en el docker-compose.yml que estén parados.

> docker-compose run

# Pausa los contenedores que previamente se han lanzado con docker-compose up.

> docker-compose pause

# Reanuda los contenedores que previamente se han pausado.

> docker-compose unpause

# Reinicia los contenedores. Orden ideal para reiniciar servicios con nuevas configuraciones.

> docker-compose restart

# Para los contenedores, los borra  y también borra las redes que se han creado con docker-compose up (en caso de haberse creado).

> docker-compose down

# Para los contenedores y borra contenedores, redes y volúmenes

> docker-compose down -v

# Muestra los logs del servicio llamado servicio1 que estaba descrito en el docker-compose.yml.

> docker-compose logs servicio1

# Ejecuta una orden, en este caso /bin/bash en un contenedor llamado servicio1 que estaba descrito en el docker-compose.yml
```

```
> docker-compose exec servicio1 /bin/bash
```

Algunos otros subcomandos interesante son:

- **docker-compose build** que ejecutaría, si está indicado, el proceso de construcción de una imagen que va a ser usado en el docker-compose.yml a partir de los ficheros Dockerfile que se indican.
- **docker-compose top** que muestra los procesos que están ejecutándose en cada uno de los contenedores de los servicios.

IMPORTANTE (REPITIENDO): DEBEMOS LLAMAR A LA DOCKER-COMPOSE DESDE EL DIRECTORIO DONDE ESTÁ EL FICHERO DOCKER-COMPOSE.YML

REFERENCIA COMPLETA DOCKER-COMPOSE

Podéis encontrar la referencia completa de la orden docker-compose en el siguiente enlace: https://docs.docker.com/compose/reference/