GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

PFM02. Infraestructura

Sistemas de Gestión Empresarial
2 Curso // CFGS DAM // Informática y Comunicaciones
Alfredo Oltra

Cicles Formatius

ÍNDEX

Versión: 230914.2238

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

☑ Atención.

Interesante.

PFM02. INFRAESTRUCTURA

1 INTRODUCCIÓN

Antes de empezar a codificar es necesario crear y configurar el entorno de trabajo, es decir, todas aquellas herramientas que nos van a ser necesarias para programar, compartir, testear y documentar el código.

En principio podríamos pensar que con un editor de código y un navegador podría ser suficiente, pero no es así. Estamos en un proyecto en el que las modificaciones que vamos a realizar afectan a la parte servidora. Por lo tanto, vamos a necesitar una copia del servidor y toda su configuración en local para hacer las pruebas necesarias.

En resumen, crear el entorno (y sobre todo configurarlo) resulta fundamental para el trabajo diario, pero no es una tarea fácil. Normalmente una elección definitiva nunca se da (versiones nuevas, herramientas más potentes o que se adecúan mejor a nuestra problemática, cambios en la estructura de los equipos…). En esta fase vamos a conocer con un poco de profundidad los problemas a los que nos enfrentamos y las diferentes posibilidades que tenemos para solucionarlos, para al final, optar por una e implantarla.

2 OBJETIVO

El objetivo de esta fase es la creación y configuración de un entorno de trabajo estable preparado para la realización del desarrollo del proyecto.

3 HERRAMIENTAS

☑ No instales todavía ninguna de las herramientas que se comentan en este apartado. El proceso de instalación se explica en un apartado posterior.

El primer paso consiste en definir cuáles van a ser las herramientas que vamos a necesitar a lo largo del desarrollo. En general, una buena planificación nos llevaría también a definir con qué herramientas y tecnologías vamos a trabajar, al menos en lo que hace referencia a las transversales o troncales que, por otra parte, son las más importantes[1].

Obviamente, nuestro caso es un poco especial y las tecnologías (*Odoo* y *Python*) vienen definidas por los contenidos impartidos durante el curso aunque, como se irá viendo en otras fases del proyecto, se dispondrá de cierta flexibilidad por parte del alumno a la hora de elegir alguna tecnología siempre y cuando cuente con el beneplácito del profesor. Es por eso que vamos únicamente a centrarnos en la elección de las herramientas de trabajo y de aquellas tecnologías que nos vienen impuestas.

El listado de herramientas que se necesitan desarrollar el PFM son[2]:

- **Sistema operativo**. Parece una obviedad, pero es un buen momento para hacer un cambio y montar un buen sistema operativo de desarrollo2. Mis recomendaciones, en orden, son *GNU/Linux* (por ejemplo *KUbuntu*), *MacOS* y en último lugar un *Windows 11 Pro*.

  ☑ Las versiones *Home* de *Windows* no está soportadas.

- **Editor de código**. La elección es libre. Mi recomendación es Visual Studio Code, pero otras opciones como Sublime o PyCharm pueden ser igualmente buenas
- **Sistema de control de versiones**. *git*, utilizando como repositorio externo GitHub. Va a ser fundamental tanto para compartir el desarrollo o realizar un seguimiento del mismo.

  Por lo que respecta a GitHub lo utilizaremos para poder trabajar de manera remota y desacoplada con respecto al profesor. Su objetivo es convertirse en el punto común de control del proyecto, dónde no sólo almacenaremos los commits que vayamos realizando sino también llevando un control de los errores que van apareciendo, las propuestas de diferentes mejoras, etc
- **Cliente gráfico para sistema de control de versiones (O)**. Para realizar las tareas diarias de seguimiento de versiones con mayor facilidad es recomendable disponer de un cliente gráfico. Hoy en día los editores de código suelen permitir la instalación de plugins que integran el acceso al repositorio directamente, pero las aplicaciones externas suelen permitir más opciones y ser más productivas. En sistemas *MacOS* o *Windows* mi recomendación es *SourceTree* o *gitKraken*, mientras que *gitg*, *gitKraken* o *SmartGit* en sistemas GNU/Linux (los dos últimos propietarios pero con licencia gratuita para uso no comercial).

1 El día a día puede ir incluyendo tecnologías de apoyo o secundarias para resolver problemas concretos
2 Con (O) se indican aquellas herramientas que no son vitales para el funcionamiento del sistema, pero que puede ser interesante disponer de ellas.

☑ Es posible utilizar extensión incluidas en los editores de código como VSCode, pero sus opciones son más limitadas en algunos aspectos, aunque para hacer un seguimiento de commits no son problemáticos.

- **Servicio SSH (O).** Aunque no es obligatorio, el uso de SSH para comunicarnos con GitHub es muy recomedable. Si todavía no posees una clave debes generarte una en tu ordenador (por ejemplo con *ssh-keygen*) y registrarla dentro de tu perfil de GitHub. Puedes encontrar más información aquí.

- **Navegadores web**. Como mínimo se deberá utilizar dos navegadores. La recomendación es *Chromium/Chrome* y Firefox Quantum, aunque otras opciones como Opera o Safari son también válidas. El hecho de utilizar al menos dos es posibilitar la comprobación del buen funcionamiento de las renderizaciones en entornos diferentes.

- **Servidor ERP**. Obviamente es la clave de todo el proyecto. En nuestro caso el servidor elegido ha sido Odoo.

- **Base de datos**. Para conseguir persistencia en el servidor utilizaremos como sistema gestor de base de datos postgreSQL.

4 INFRAESTRUCTURA

4.1 Máquina nativa

En general, la primera opción que tendríamos para construir el entorno de trabajo sería utilizar nuestro ordenador habitual. De esta manera, instalando de forma nativa todas las herramientas necesarias, podríamos llegar a obtener un entorno bastante optimizado, ya que, a priori, no existiría ninguna capa entre el hardware y las aplicaciones que ralentizarse el trabajo.

Sin embargo este tipo de infraestructuras tienen dos problemas importantes; por una parte es posible que alguna o varias de las herramientas necesarias no se encuentren disponibles para nuestro sistema operativo y por otra, que pueden generarse conflictos con otras aplicaciones parecidas a la hora de actualizar a versiones superiores. Por poner un ejemplo relativo a esto último, no sería nada descabellado que necesitásemos tener dos o más versiones de *Odoo* o de un sistema gestor de base datos instaladas para mantener la compatibilidad con dos proyectos con los que estemos trabajando de manera simultánea[3]. Muchas de estas situaciones se pueden solucionar tocando la configuración y con una buena dosis de paciencia, pero ni que decir tiene que suponen un coste de mantenimiento bastante alto.

4.2 Máquina virtual

Una alternativa que mejora parte los problemas anteriores es el trabajo con máquinas virtuales. Podríamos crearnos tantas máquinas virtuales como entornos de desarrollo diferentes quisiéramos. Si necesitamos un entorno nuevo, se crea una máquina virtual nueva. Si la herramienta sólo está disponible para un SO, se crea una máquina con ese sistema. Además, tendríamos un valor añadido: una máquina virtual se encuentra aislada del anfitrión, aportando un nivel de seguridad y de protección de datos mayor.

Pero las máquinas virtuales también tienen problemas. El principal es la bajada de rendimiento: si el entorno necesario es muy exigente es muy probable que, salvo que dispongamos de un ordenador muy potente, el uso de la máquina virtual nos lastre en el trabajo diario. Por otra parte, ese aislamiento del host se puede convertir en algo tedioso cuando deseemos intercambiar información con él o entre máquinas virtuales. Por último, con mucha probabilidad tendremos que tener el mismo software instalado varias veces lo cual complica las labores de mantenimiento cuando, por ejemplo, deseemos actualizar a una nueva versión o modificar una configuración de funcionamiento.

4.3 Contenedores

En los últimos años ha empezado a coger fuerza una tercera vía, el uso de contenedores, que intenta adoptar las ventajas y descartar los inconvenientes de las anteriores. Pero ¿qué es un contenedor? De una manera muy simplista podríamos definir un contenedor como una especie de máquina virtual de aplicaciones. Profundizando un poco, un contenedor es espacio

3 O incluso no tan simultánea. Es muy habitual en la vida real estar trabajando en la creación de un nuevo proyecto con las últimas versiones de las tecnologías, pero que haya que seguir realizando un mantenimiento de otro proyecto desarrollado previamente con versiones anteriores.

de memoria en el que se crean las condiciones necesarias para que una o varias aplicaciones puedan ser ejecutadas, entendiendo por condiciones necesarias la existencia no solamente de la aplicación instalada, sino también todas aquellas librerías y ficheros de configuración que son necesarios para que funcione sin ningún tipo de problema. Además ese contenedor se encuentra, en principio, aislado del resto del sistema que lo contiene, aunque puede comunicarse con él, y por lo tanto con otros contenedores, a través de puertos o mediante volúmenes (espacios de disco duro).

A efectos del usuario esto aporta varias ventajas.

- El trabajo con los contenedores es transparente. Un usuario arranca desde su ordenador el contenedor que le interesa (para él es algo parecido a arrancar una aplicación) pero sabiendo que este no va a interferir con su sistema. El usuario trabaja de manera normal, como si la instalación de la aplicación hubiera sido nativa.
- Menos sobrecarga del sistema. Cada contenedor tiene lo necesario y sólo lo necesario para hacer funcionar la aplicación, por lo que hay menos exigencia de recursos y los arranques y paradas son más rápidos que con una máquina virtual.
- Al ser lo más habitual que su ámbito sea de una o muy pocas aplicaciones, es frecuente que alguien haya creado el contenedor[4] y lo comparta para su uso, de manera que únicamente haya que descargarlo de internet.
- A diferencia de las máquinas virtuales, sólo necesitamos tener contenedores de aquellas cosas específicas de cada entorno, con los beneficios en cuestión de mantenimiento que eso implica. Por ejemplo, el IDE puede ser siempre el mismo.
- Permite crear muy fácilmente entornos que sean combinaciones diferentes de contenedores. Por ejemplo, hacer un cambio de un entorno *MySQL + PhpMyAdmin* a otro *MariaDB + phpMyAdmin* es tan sencillo como cerrar el contenedor de *MySQL* y abrir el de *MariaDB*.
- A la hora de desplegar es mucho más sencillo llevarse al servidor una serie de contenedores que ya se sabe que funcionan correctamente, que realizar una instalación y configuración de todas las herramientas.

  Como se puede comprobar, esta es la opción más versátil y eficaz de las propuestas, por lo que va a ser la que utilizaremos para el montaje de la infraestructura del entorno.

4 Realmente se denominan imágenes.

5 DOCKER

Docker es una aplicación que, trabajando como un servicio, nos va a permitir la gestión de los contenedores. Aunque a nivel de usuario nos vamos a limitar a instalar, arrancar y parar contenedores conviene como mínimo tener claro algunos conceptos, sobre todo por lo que se refiere a nomenclatura.

El elemento base de trabajo de *Docker* son las imágenes. Una imagen no es ni más ni menos que una especie de máquina virtual congelada almacenada en el disco duro. Eso sí, una máquina virtual reducida y centrada en una o pocas aplicaciones. Los contenedores se crean a partir de una imagen. Dicho con otras palabras, un contenedor no es ni más ni menos que la representación en memoria de la imagen almacenada en nuestro disco duro.

Es posible definir una empresa de cualquier sector, pero es obligatorio que sea de una mediano o gran empresa. La implantación en una empresa pequeña no permite sacarle el juego necesario al ERP y por tanto al PFM.

Por poner un símil (no del todo correcto), una imagen podría ser como un programa instalado en nuestro disco duro, con sus configuraciones y sus librerías. Por cada vez que lo arrancamos generamos un proceso que sería el que haría las veces de contenedor. Cada uno de esos procesos (contenedores), aunque parte de un mismo programa (imagen) tiene su ciclo de vida independiente del resto, pudiendo crear objetos, otros procesos, etc.

Para obtener las imágenes tenemos dos opciones:

- Obtenerla directamente internet. Existe un repositorio público de imágenes (Docker Hub) donde muchos de los desarrolladores crean ya las imágenes de sus aplicaciones para que puedan ser utilizadas de manera muy sencilla
- Construirlas. Para ello se crea un fichero de definición de imágenes denominado Dockerfile. Este fichero especifica los comandos de instalación de la(s) aplicación(es) que se quieren incluir en la imagen. Aunque es posible partir de cero, lo más habitual es partir de una imagen previa e incluir lo que necesitemos.

Veamos un ejemplo: MySQL tiene publicadas en Docker Hub las imágenes de sus SGBD desde la versión 5 hasta la última. Si necesitamos un MySQL 5.7 y un 8 para diferentes proyectos podemos descargar ambas imágenes y en función de qué es lo que nos haga falta en cada momento arrancar (crear el contenedor de) una o la otra.

En el caso que quisiéramos realizar alguna modificación de esa imagen podríamos definir un *Dockerfile*, por ejemplo:

```
FROM mysql:5.7
COPY my.cnf /etc/mysql/conf.d/my.cnf
CMD ["mysqld"]
EXPOSE 3306
```

Donde, a partir de la imagen oficial de la versión 5.7, modificamos la configuración de arranque con una serie de valores que están definidos en el fichero *my.conf*.

Existe otra opción de crear imágenes a partir de una existente que consiste en arrancar el contenedor, realizar modificaciones, salir y realizar un commit de la imagen. Este proceso nos genera otra imagen basándose en la primera pero que incluye las modificaciones realizadas.

Una de las cosas más útiles del uso de contenedores es poder crear imágenes de aplicaciones concretas y combinarlas. En ese caso sería interesante disponer de algún sistema que nos permitiera definir cómo se van a coordinar entre sí los contenedores resultantes. Es en este contexto donde entra *Docker Compose*, una aplicación de la familia Docker qué va a permitir que creemos scripts que definan esa interacción. El inconveniente de este método es que el gasto en memoria y disco duro es mayor.

En este ámbito los contenedores pasan a denominarse servicios, ya que el objetivo es que se proporcionen servicios (valga la redundancia) entre ellos.

6 INSTALACIÓN. FASE 1. HERRAMIENTAS DEL HOST

En esta primera fase vamos a instalar las herramientas generales, aquellas que no dependen únicamente del proyecto. Es posible que muchas de ellas ya las tengas instaladas. En principio todas las herramientas para trabajar con el PFM en esta fase se puede instalar para cualquiera de los 3 sistemas operativos de escritorio más utilizados: *Linux, MacOS y Windows*.

☑ En sistemas Windows es altamente recomendado utilizar WSL2. De hecho, los comandos indicados se indican para *Linux, MacOS o WSL*.

1. Navegadores web. Su instalación resulta muy sencilla en cualquiera de los sistemas que hayas elegido para trabajar. Recuerda que como mínimo debes tener dos instalados.

2. *git*. La instalación de *git* no supone demasiados problemas.

   - MacOS. Si tienes instalado *XCode* lo tendrás por defecto (tal vez no una de las últimas versiones, pero valdrá)[5]. En caso contrario lo puedes descargar de la web de *git*.

   - Windows. La instalación la puedes realizar sin demasiados problemas. La descarga la puedes realizar desde la web de *git*.

   - Linux. La instalación se realiza desde el terminal mediante el comando:

     ```
     $ sudo apt-get install git
     ```

3. Cliente gráfico para git. La instalación de cualquiera de las opciones elegidas es muy sencilla ya sea utilizando el instalador (*.dmg, .deb* o *.msi*) o con apt-get. Ante cualquier duda conviene consultar las instrucciones del desarrollador.

4. *Docker* y *Docker Compose*. La instalación de *Docker* es relativamente sencilla siguiendo los pasos indicados en la página web oficial. La solución más común es la de instalar *Docker Desktop*, un cliente gráfico disponible tanto para *Windows, MacOS* como *Linux* y que incorpora *Docker* (llamado oficialmente *Docker Engine*) y *Docker Compose* (aparte de otros elementos interesantes como *Kubernetes*). En sistemas Linux existe la opción de instalar directamente (sin cliente gráfico) *Docker Engine*[6] *(*el proceso descrito instala también el plugin para *Compose).*

☑ No instales *Docker* desde los repositorios de Ubuntu o similar. Esos repositorios priorizan la estabilidad de las versiones sobre las novedades de las mismas, por lo que suelen ser versiones muy desactualizadas.

5 Si tienes algún problema siempre puedes ejecutar en un terminal `xcode-select –install` para intentar corregirlo.

6 De las opciones comentadas, la recomendada es usar el repositorio *apt*

7 INSTALACIÓN. FASE 2. PUESTA EN MARCHA

Para esta fase 2, el proceso que se ha de seguir es muy similar al descrito en la documentación de la unidad 2 del módulo[7]. Como ya hemos comentado, vamos a trabajar con contenedores *Docker* y a organizar su funcionamiento con *Compose*.

☑ A partir de aquí recuerda que es necesario tener *Docker* arrancado.

☑ Es posible que los usuarios de Linux necesiten ejecutar las instrucciones relacionadas con *docker* con privilegios de *root*, es decir, deben ejecutarse con el comando *sudo*. En estos casos es recomendable crear un grupo llamado *docker* y añadir a nuestro usuario en él. Más información aquí.

.

☑ En el Anexo I se han incluido las soluciones a posibles situaciones especiales que se pueden dar en el proceso de instalación de Docker.

1. Es conveniente crear en local una carpeta llamada *Odoo_dev* o *[nombre de la empresa de la fase 1]_dev* (en el resto del documento haremos referencia a ella como *Odoo_dev*), en la que ir dejando todo lo necesario para el proyecto: documentación, código, configuraciones, datos…

2. Clonar en su interior *odoodock*

```
$ cd odoo_dev
$ git clone https://github.com/aoltra/odoodock.git
```

3. Entrar en la carpeta *odoodock*

```
$ cd odoodock
```

4. Copiar el fichero *.env-example* a *.env*

```
$ cp .env-example .env
```

5. Modificar el fichero *.env* para adapartalo a nuestras necesidades (O).

En un principio la versión de ejemplo es suficiente para empezar a trabajar.

6. Copiar el fichero *.services-example* a *.services*

```
$ cp .services-example .services
```

7 Con (O) se indican aquellos pasos que no son vitales para la instalación del sistema.

7. Modificar el fichero *.services* para incluir los servicios que deseamos arrancar. Los servicios se separan por espacios y van entrecomillados.

> En un principio únicamente es suficiente arrancar el servicio *web* para empezar a trabajar.

> El servicio *web* es obligatorio para arrancar *odoo*. De manera automática se arranca también el servicio *db*.

8. Asignar permisos de ejecución para el usuario al fichero `up.sh` y `create-module.sh`.

```
$ chmod u+x ./up.sh
$ chmod u+x ./create-module.sh
```

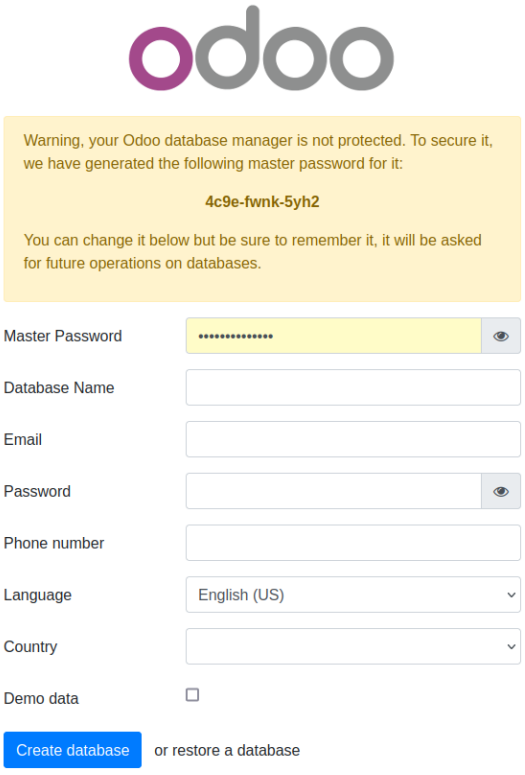
9. Arrancar los servicios

```
$ ./up.sh
```

10. Comprobar que los contenedores están en ejecución (O). Conviene comprobar que el estado de todos los contenedores es *UP*, al ejecutar

```
$ docker compose ps    o    $ docker ps
```

11. Para comprobar que todo ha ido correctamente, acceder desde un navegador a *localhost:8069*, donde debe aparecer la página del selector de la base de datos.



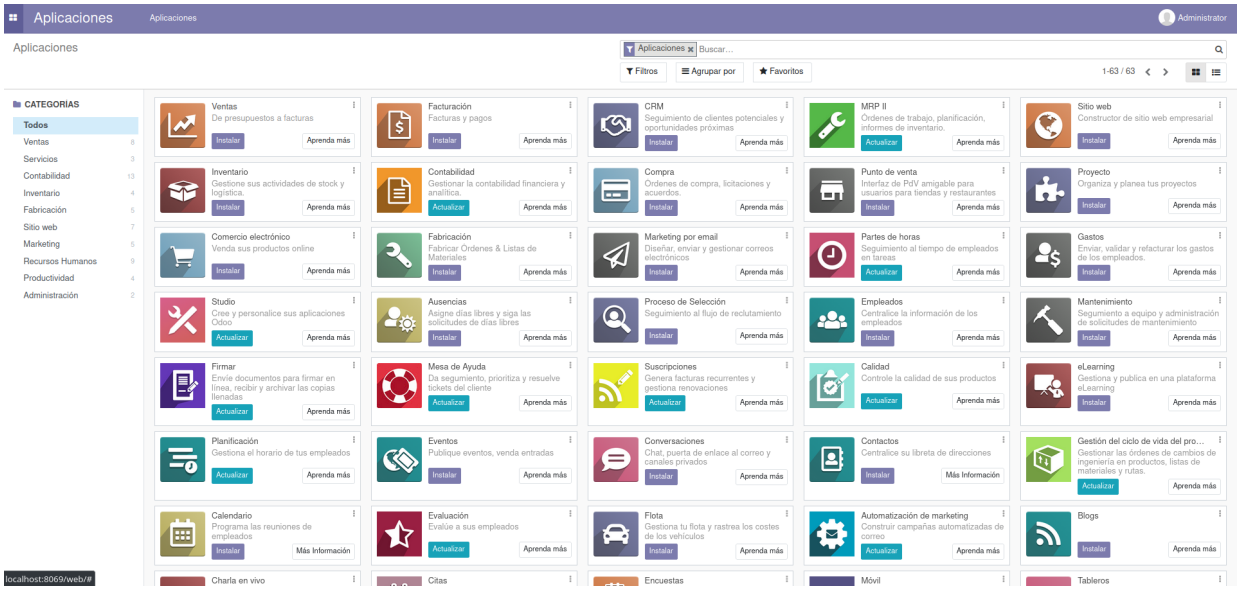
En caso de que esa pantalla no aparezca hay que dar por hecho que existe algún problema con los contenedores. Para obtener información de qué ha sucedido lo mejor apagarlos y rearrancar compose de manera manual sin la opción *-d (detach mode)*, de tal manera que se muestre por consola toda la información del arranque.

```
$ docker compose up.
```

12. Configurar los valores y crear la base de datos.

Es recomendable almacenar el *master password* en un fichero aparte.

13. Si todo ha ido correctamente, una vez finalizada la creación de la base de datos, deberá cargarse en el navegador la página *Aplicaciones*.



Toda esta información (y más) la puedes ver detallada en la documentación online de odoodock.

8 EL DÍA A DÍA

Una vez montada toda la infraestructura, el trabajo diario es bastante sencillo. Únicamente hay entrar dentro de la carpeta *odoo_dev/odoodock* y ejecutar `./up.sh` cada vez que vayamos a trabajar y cerrar los contenedores una vez hayamos acabado (no es que sea necesario, pero ahorrará recursos en el sistema). Para ello simplemente ejecutamos desde *odoo_dev/odoodock*:

```
$ docker-compose down
```

Aunque el procedimeinto es básicamente el mismo, cuando codifiquemos habrá que realizar algunos pasos más que ya comentaremos en las fases correspondientes.

9 ENTREGA

La entrega de esta fase consistirá en la creación de un pequeño video y de la creación de una copia de seguridad de la bases de datos.

- El video consistirá en una pequeña demo en la que:
  1. Arranques los contenedores,
  2. Entres en *Odoo* con un usuario que sea tu NIA en la base de datos llamada C*URSO_APELLIDO1_NOMBRE_NIA*, donde:
     - *CURSO* son los dos últimos digitos del año inicial y final del curso,
     - *APELLIDO1* será tu primer apellido en mayúsculas y sin tíldes (si es compuesto elimina los espacios)
     - *NOMBRE* será tu nombre en mayúsculas y sin tíldes (si es compuesto elimina los espacios)
     - *NIA* será tu NIA
     - Por ejemplo, si tu nombre es *Lucía Del Valle* con el *NIA* es *123456* y es el curso que comienza en el 2022, la base de datos debe llamarse *2223_DELVALLE_LUCIA_123456*
  3. Accedas al modo desarrollador.
  4. Accedas a ajustes (icono de cuadrados en la esquina superior izquierda) y crees un usuario con login *profesor* y correo *profesor@ceedcv.es*.
  5. Crees la copia de la base de datos (en formato *.zip)*
- El video ha de llamarse *instalacion.ext* (la extensión es la que tu consideres en función del formato de grabación) y ha de incluir tu voz **presentándose** y explicando los pasos.

☑ El video debe grabarse con un programa de captura de pantalla (como OBS). En ningún caso debe ser una grabación realizada con un móvil. Además la voz ha de ser clara. En cualquiera de los otros casos la fase será rechazada.

☑ La entrega se realizará en la tarea *PFM-F2. Entrega*, dentro de la plataforma *Aules,* incluyendo ambos ficheros. En caso de que el tamaño del video fuera muy alto (no debería) es posible subirlo a alguna plataforma de manera **privada** e indicar el enlace mediante un fichero *.txt*. La fecha aconsejada de entrega será el 7 de noviembre de 2023 a las 23:55.

10 CONSEJOS

- **Leer**. Simplemente lee con tranquilidad y completamente las instrucciones que se te proporcionan. Copiar y pegar los comandos que vayas a ir encontrando puede ser que te funcione, pero es bastante probable que te cree problemas. Además, no habrás aprendido nada.

- **Curiosear**. Tienes a tu alcance un sistema docker con varios servicios, construcciones de imagenes, scripts de arranque… Ahora es el momento de copiar, modificar y probar. Al final siempre puedes borrar todo (imagenes, carpetas, contendores…) y volver a empezar

- Para cualquier duda sobre el enunciado o sobre posible información a incluir en la empresa puedes utilizar el foro *PFM-F2. Dudas planificación.*

11 ANEXO I. POSIBLES PROBLEMAS

Soluciones a posibles situaciones que se pueden dar en el proceso de instalación de Docker.

11.1 Linux

11.1.1 Colisiones de puertos

Aunque lo incluyo en *Linux*, este problema puede aparecer en cualquier SO. Aparece cuando ya tienes previamente instalado en tu máquina las aplicaciones que vas a utilizar desde *Docker*. En ese caso, es posible que tanto la aplicación nativa como la de *docker* vayan a hacer uso del mismo puerto de comunicación, creando una colisión. En ese caso es necesario parar las aplicaciones nativas, ya sea cerrando *postgreSQL* u *Odoo* local. Por ejemplo desde distribuciones *Debian* (como *Ubuntu*),se pueden parar los demonios ejecutando:

```
$ sudo sudo service odoo-server stop
$ sudo service postgresql stop
```

11.1.2 Acceso denegado / Imposibilidad de conectar con el demonio *Docker*

Como se indica en la documentación, desde Linux es necesario ejecutar los comandos con *sudo*. Los posibles errores que pueden surgir por no ejecutarlo de esa manera son muy variados, por lo que hay que asegurarse siempre que la ejecución se ha realizado como *root*.

En el caso de que quieras evitar escribir sudo cada vez, es posible añadir a tu usuario a un grupo llamado docker, el cual si tiene permisos para correr contenedores. Para ello:

```
$ sudo groupadd docker
$ sudo usermod -aG docker loginusuario
```

donde *loginusuario* es el nombre de tu usuario.

11.1.3 Problemas de arranque de contenedor *PostgreSQL*

En caso que el arranque del contenedor de *PostgreSQL* falle dando un error de permiso la solución (en desarrollo) consiste en añadir en el dockerfile, para el servicio de bases de datos, una indicación de que el usuario con el que va a funcionar va a ser *root*

```
db:
  image: postgres:10
  ...
  user: root
  ...
```

11.2 Windows 10/11 Pro

11.2.1 Hyper-V

Al arrancar *Docker* es posible que aparezca un mensaje indicando la necesidad de habilitar *Hyper-V* y que, en caso de hacerlo, *Virtual Box* dejará de funcionar. Efectivamente ambas dos opciones no pueden simultanearse, pero pueden convivir activando y desactivando la opción desde un terminal con permisos de administrador ejecutando:

```
bcdedit /set hypervisorlaunchtype off
```

para desactivar *Hyper-V* o

```
bcdedit /set hypervisorlaunchtype auto
```

para activarlo.

Posteriormente es necesario reiniciar el equipo.

11.2.2 Error de uso en MinTTY Git Bash

Al ejecutar:

```
docker compose exec workspace bash
```

es posible que el terminal devuelva el siguiente aviso:

the input device is not a TTY. If you are using mintty, try prefixing the command with 'winpty'

la solución consiste en ejecutar el comando prefijándolo con *mintty*:

```
mintty docker-compose exec workspace bash
```