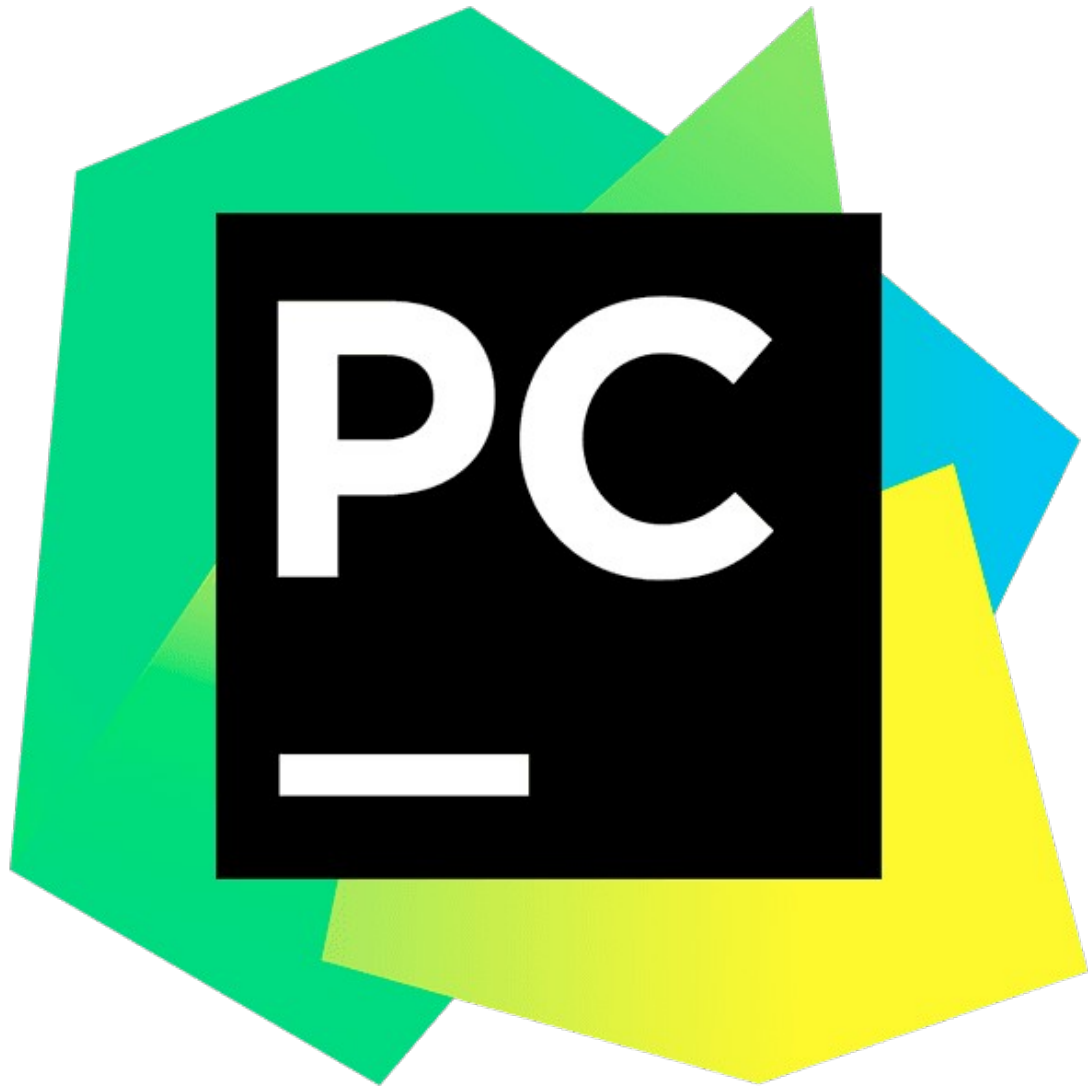
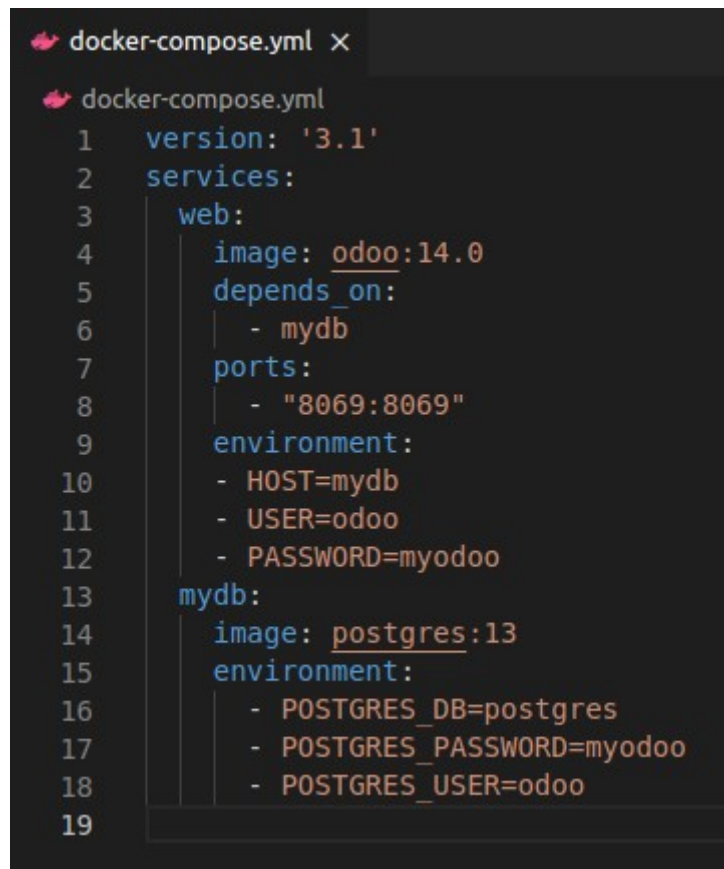


INSTALACION ODOO Y POSTGRES



Lo primero de todo es crear un archivo docker-compose.yml, que contenga la siguiente estructura:



```
docker-compose.yml x
docker-compose.yml
1  version: '3.1'
2  services:
3    web:
4      image: odoo:14.0
5      depends_on:
6        - mydb
7      ports:
8        - "8069:8069"
9      environment:
10       - HOST=mydb
11       - USER=odoo
12       - PASSWORD=myodoo
13    mydb:
14      image: postgres:13
15      environment:
16       - POSTGRES_DB=postgres
17       - POSTGRES_PASSWORD=myodoo
18       - POSTGRES_USER=odoo
19
```

En este documento se va a explicar detalladamente como a través de Docker podemos crear un contenedor de Odoo y de PostgreSQL y enlazarlos para trabajar en remoto.

Lo primero de todo es crear un documento llamado “docker-compose.yml”, en el que definiremos las imagenes, credenciales, puertos y volúmenes de cada servicio, en este caso, de Odoo y PostgreSQL:

En este documento usamos las imagenes “postgres:13” y “odoo:14.0”, le asignamos el puerto 5432 a PostgreSQL y el 8069 a Odoo, y le decimos al servicio de Odoo que dependa del servicio db, que es el contenedor de la base de datos. Además asignamos un nombre, contraseña y usuario a la base de datos, postgres, odoo y odoo respectivamente.

A continuacion lanzamos el comando `docker-compose up -d` para levantar los contenedores.

```
dam2b@perlanegra22:~/Documentos$ docker-compose up -d
Starting dam22_postgresql ...
Starting dam22_postgresql ... error

ERROR: for dam22_postgresql Cannot start service db: driver failed programming external connectivity on endpoint dam22_postgresql: Error starting userland proxy: listen tcp4 0.0.0.0:5432: bind: address already in use

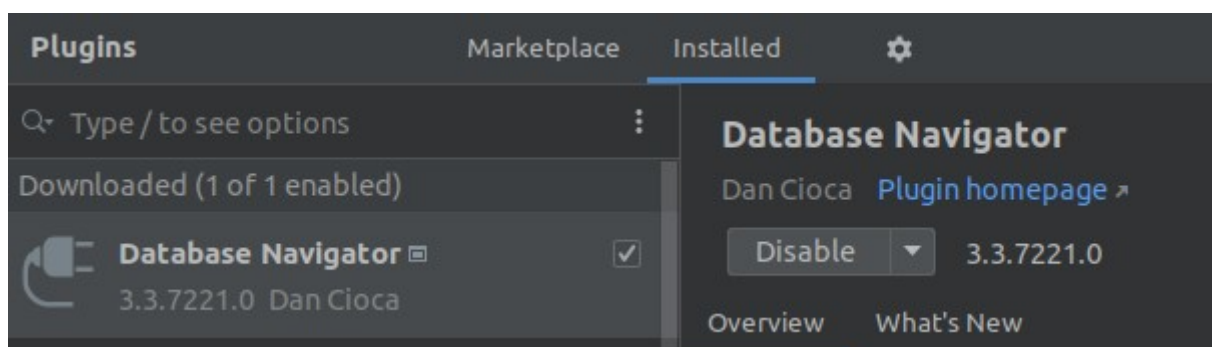
ERROR: for db Cannot start service db: driver failed programming external connectivity on endpoint dam22_postgresql: Error starting userland proxy: listen tcp4 0.0.0.0:5432: bind: address already in use
ERROR: Encountered errors while bringing up the project.
dam2b@perlanegra22:~/Documentos$ sudo lsof -i :5432
[sudo] contraseña para dam2b:
COMMAND  PID    USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
postgres 1612 postgres 5u    IPv6  32570      0t0  TCP localhost:postgresql (LISTEN)
postgres 1612 postgres 6u    IPv4  32571      0t0  TCP localhost:postgresql (LISTEN)
```

En este caso ha habido un error debido a que el puerto 5432 estaba ocupado por otro proceso. Para solucionar este problema buscamos que proceso está ocupando el puerto con el comando `sudo lsof -i :5432`, cogemos el PID del proceso y lo eliminamos con el siguiente comando: `sudo kill 1612`. A continuacion lanzamos de nuevo el `docker-compose up` y funciona:

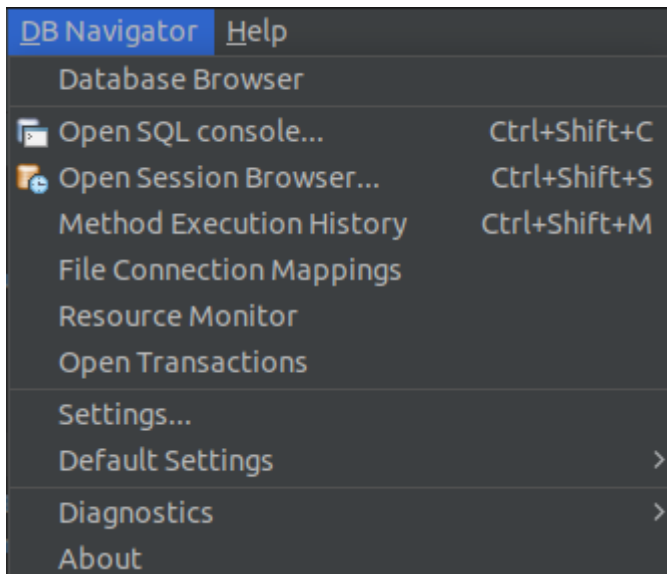
```
dam2b@perlanegra22:~/Documentos$ sudo lsof -i :5432
[sudo] contraseña para dam2b:
COMMAND  PID    USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
postgres 1612 postgres 5u    IPv6  32570      0t0  TCP localhost:postgresql (LISTEN)
postgres 1612 postgres 6u    IPv4  32571      0t0  TCP localhost:postgresql (LISTEN)
dam2b@perlanegra22:~/Documentos$ kill 1612
bash: kill: (1612) - Operación no permitida
dam2b@perlanegra22:~/Documentos$ sudo kill 1612
dam2b@perlanegra22:~/Documentos$ sudo lsof -i :5432
dam2b@perlanegra22:~/Documentos$ docker-compose up -d
Starting dam22_postgresql ... done
Creating dam22_odoo ... done
dam2b@perlanegra22:~/Documentos$
```

Una vez hecho todo esto ya tendremos los contenedores corriendo, queda enlazar Docker a Pycharm para acceder a la base de datos.

Para esto podemos hacer dos cosas, conectarnos a Docker desde la ventana `services`, que se encuentra en la zona de abajo del IDE y seleccionar Docker, o también podemos instalar un plugin para acceder a bases de datos:



Una vez hecho esto accedemos al navegador:



Y rellenamos los datos con las credenciales establecidas en el docker-compose.yml:

