

Universidad de El Salvador.

Facultad Multidisciplinaria de Occidente.

Departamento de Ingeniería y Arquitectura

Catedra: Curso progra 2.5



Objetivo: que el estudiante entienda o al menos tenga una noción de la estructura básica de un proyecto Enterprise de Java así como la configuración necesaria del servidor de aplicaciones para poder desplegar el proyecto **con Docker**.

Requisitos:

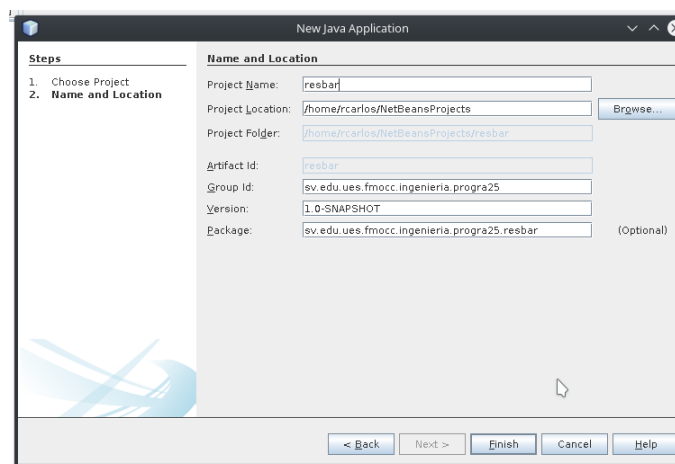
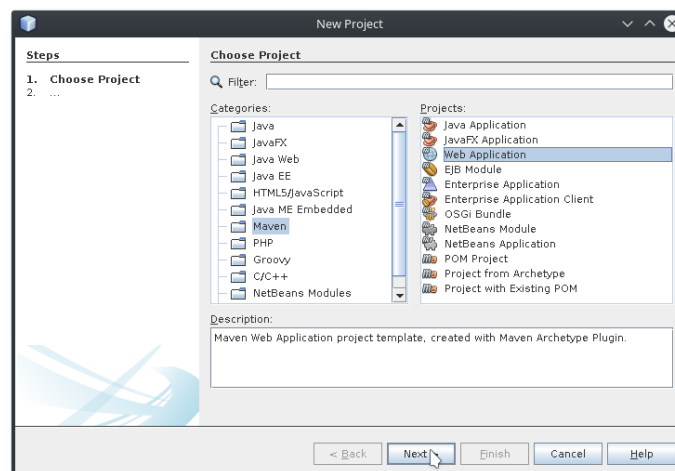
-Docker ce

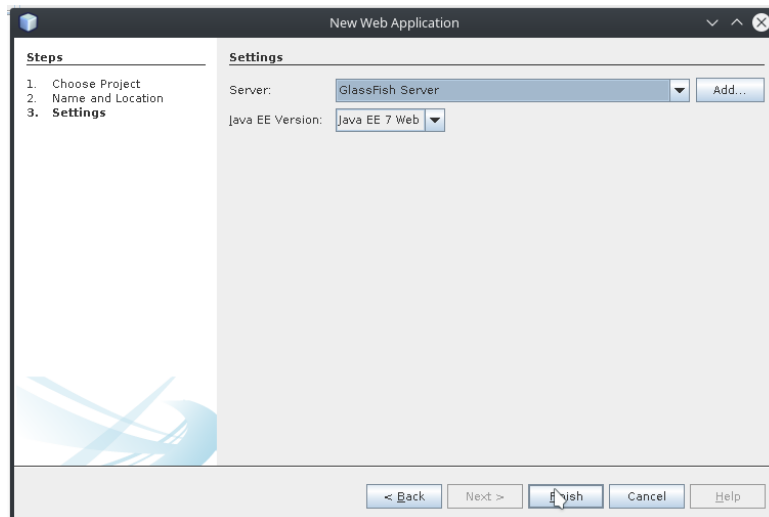
-Docker compose

Empecemos...

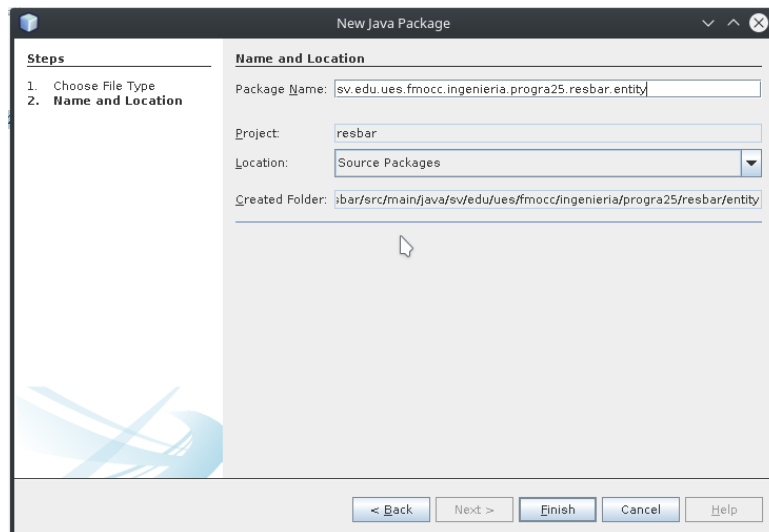
Para empezar debemos levantar un contenedor con la base de datos que nos sirva de ayuda para que el asistente de NetBeans pueda mapear la base de datos, para hacer esto en el repositorio <https://github.com/rcarlos97/recursos> hay una carpeta “/bd” que pueden utilizar de la misma forma que se usó durante la clase (entrar a la carpeta /bd y ejecutar el comando “docker-compose up”)

Luego debemos crear un proyecto java web con maven:

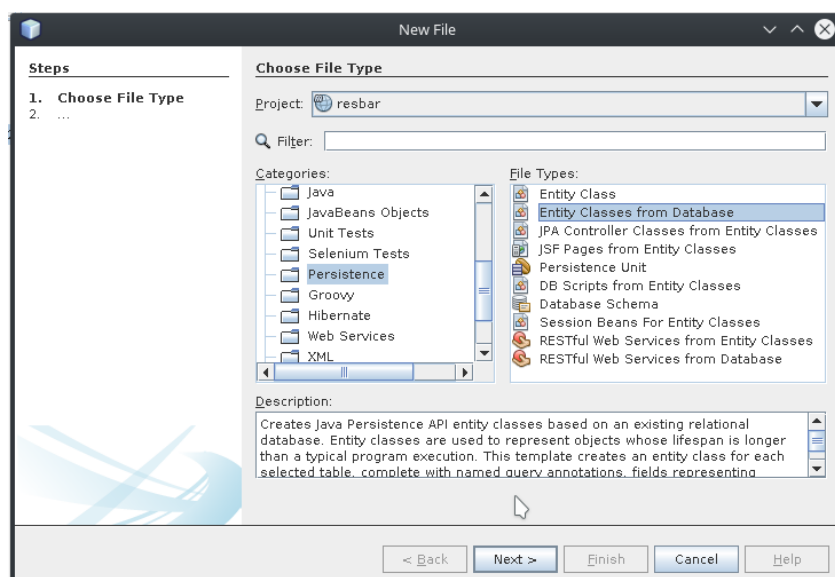




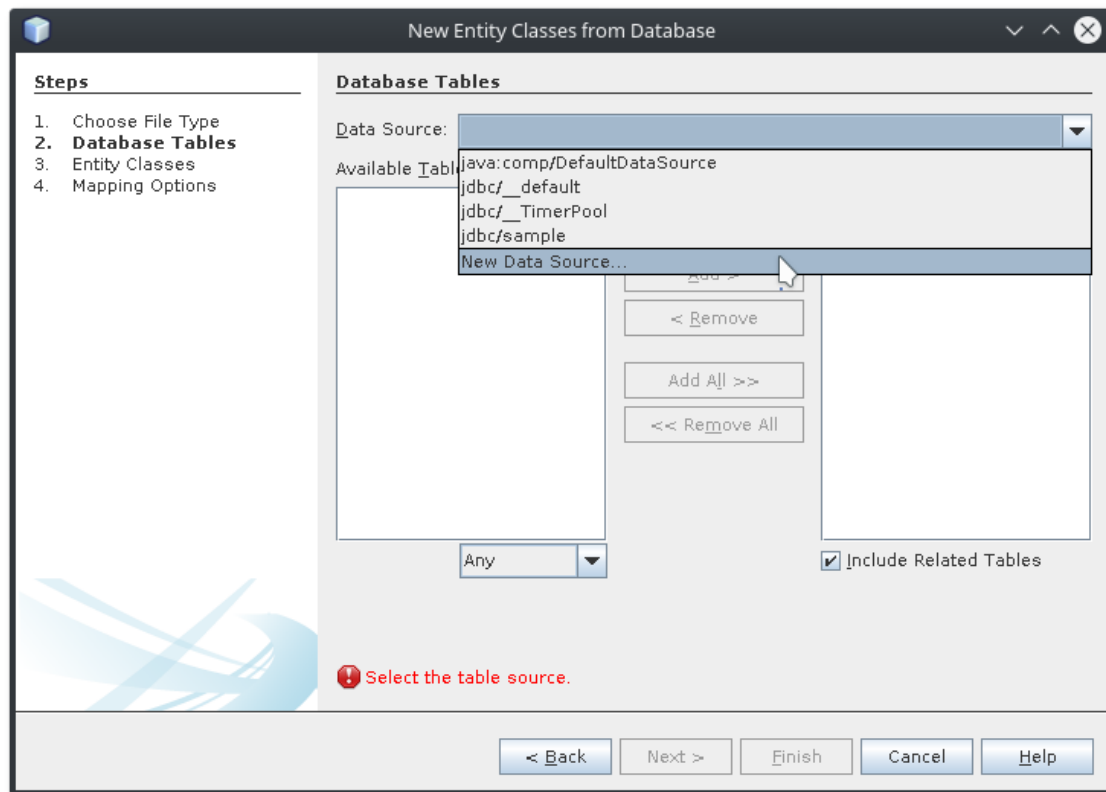
Creamos un nuevo paquete:



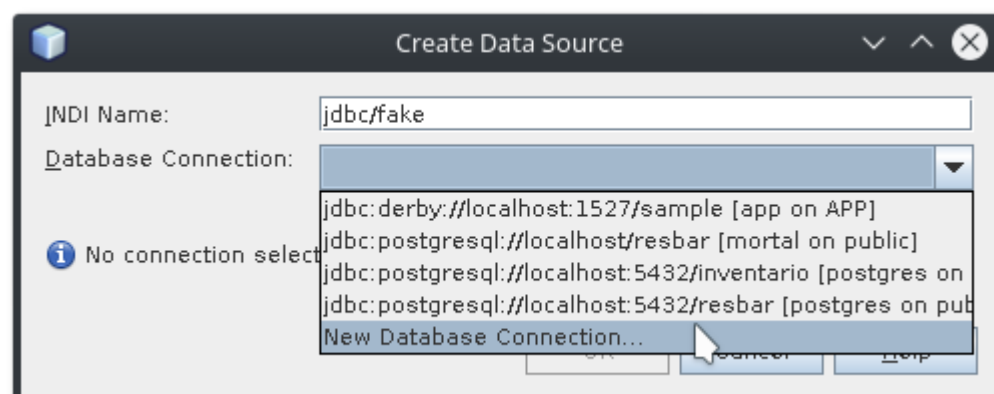
El paquete creado anteriormente servira para el mapeo de la base de datos con JPA y una vez creado sobre el proyecto damos **Click derecho > New > Other** y en el asistente abierto seleccionar:



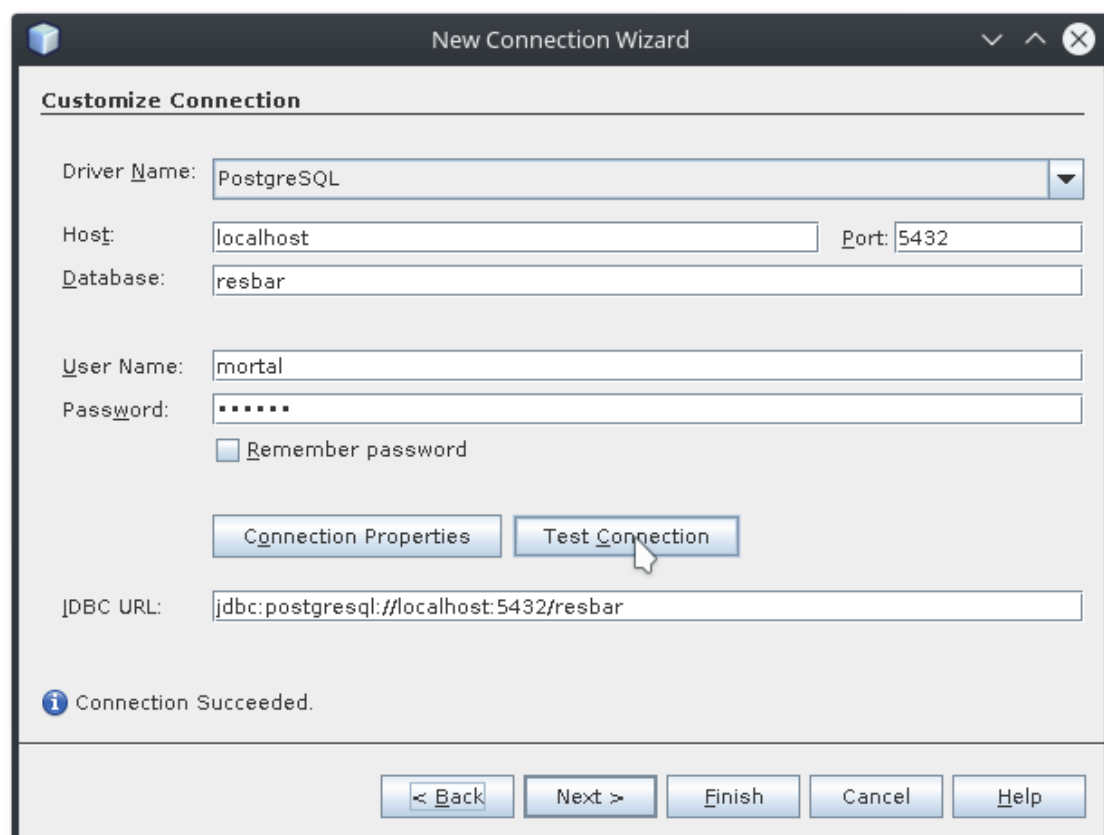
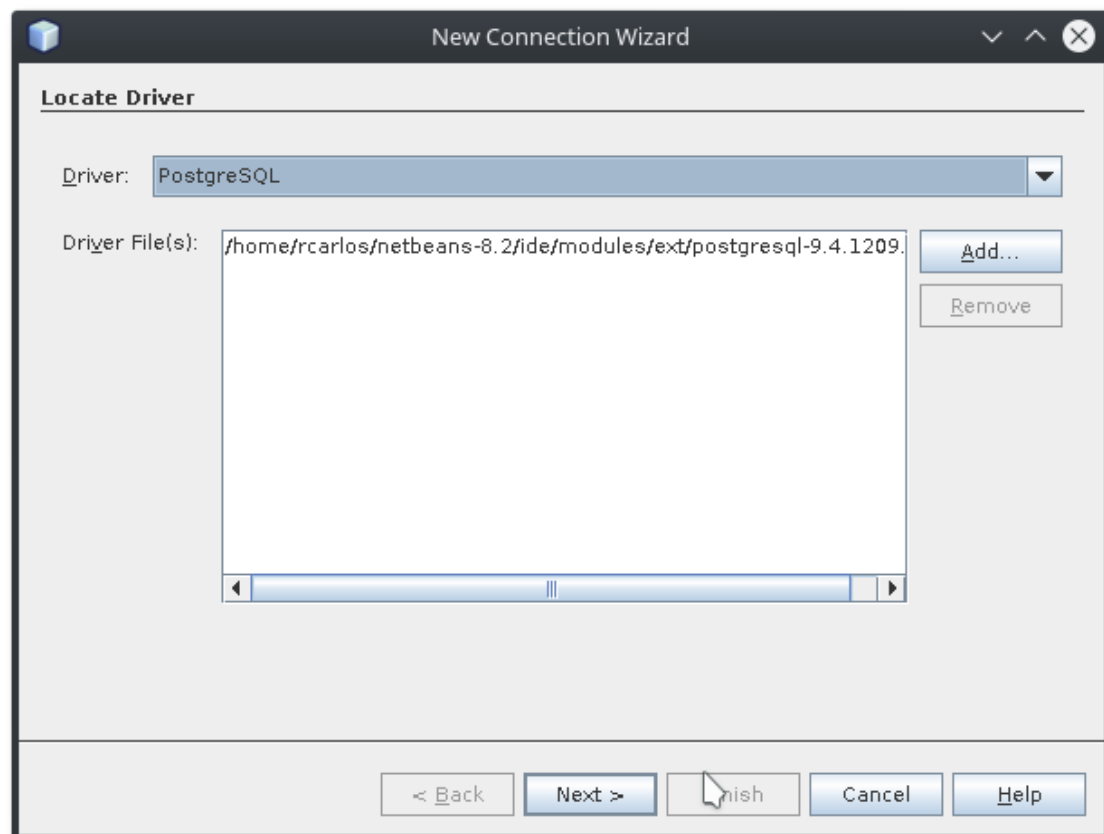
Lo siguiente es seleccionar el datasource que se crea en el servidor de aplicaciones pero hasta este momento no se ha creado ningun datasource ya que el servidor de aplicaciones se va a desplegar hasta el final. Es en este punto donde se vuelve un poquito complicado usar el asistente y lo mejor seria crear todo el mapeo de entidades a pie con "New Classs" pero hacerlo se vuelve complicado por mucha razones es por eso que vamos a ver como "engañar al asistente" y para hacerlo, en la siguiente ventana se va selecciones "New Data Source"

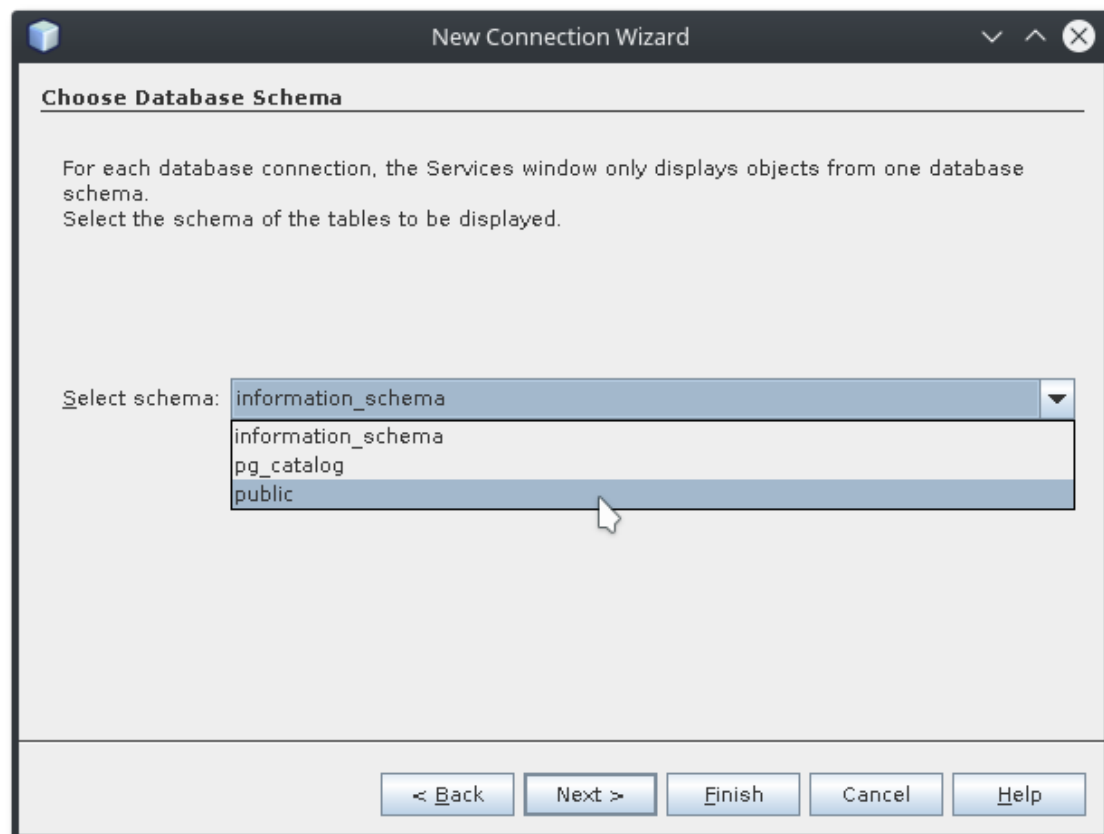


Esto les va a mostrar otro cuadro de dialogo donde deberan poner el nombre del datasource y crear una nueva conexión local asi(tambien pueden usar una conexión ya existente):

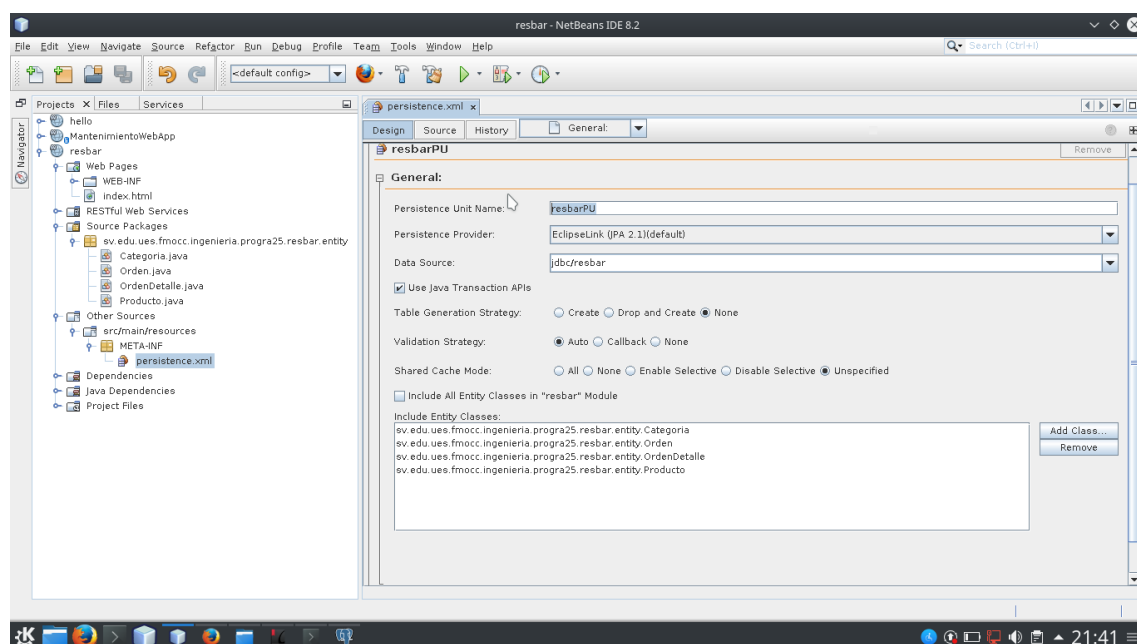


Si seguimos el paso de crear una nueva conexión, los pasos a seguir son:





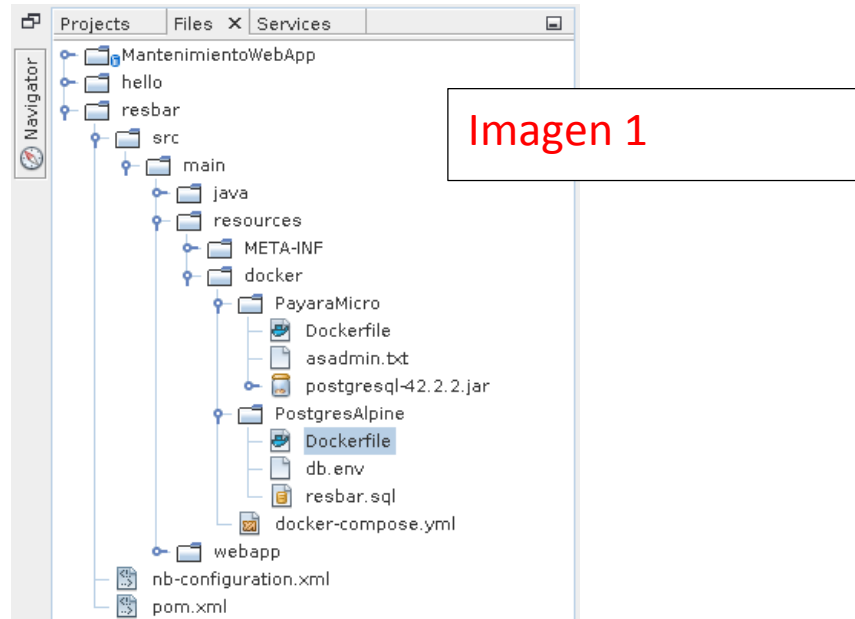
Continuando, se tiene que cambiar el nombre de la unidad de persistencia por uno mas facil de recordar y añadir todas las clase mapeadas por JPA(ver guia anterior si olvido como hacer esto, en la pagina 13 y el ultimo parrafo de la pagina 12, por sino encuentran el parra es el que tiene en negrita **"Include All Entity Classes in "resbar" Module"**). [Tambien se debera cambiar el apratado "Data Source" poniendo "jdbc/resbar" que mas adelante se creara con docker.](#)



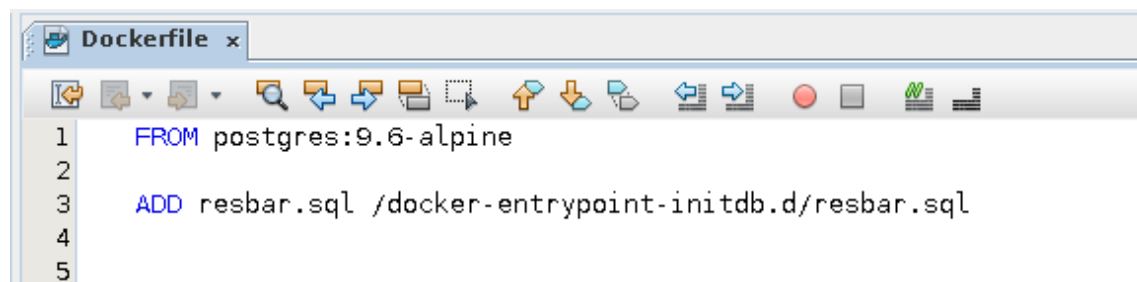
El siguiente paso es crear los Sessions Beans For Entity Classs, sino recuerdo com hacerlo, puede ver la guia anterior de la pagina 14 en adelante.

Con esto ya tenemos lista la configuración del proyecto, solo falta añadir los archivos necesarios para poder desplegarlo con docker para esto, en su proyecto deberán crear el directorio /docker en la ruta /src/main/resources quedando como ruta final /src/main/resources/docker. Para hacerlo mas facil, en el repositorio <https://github.com/rcarlos97/recursos> hay una carpeta "/proyecto" que dentro tiene la carpeta "/docker", pueden copiar esta carpeta "/docker" a su proyecto.

Una vez añadido esta carpeta a su proyecto, podemos analizar su contenido:



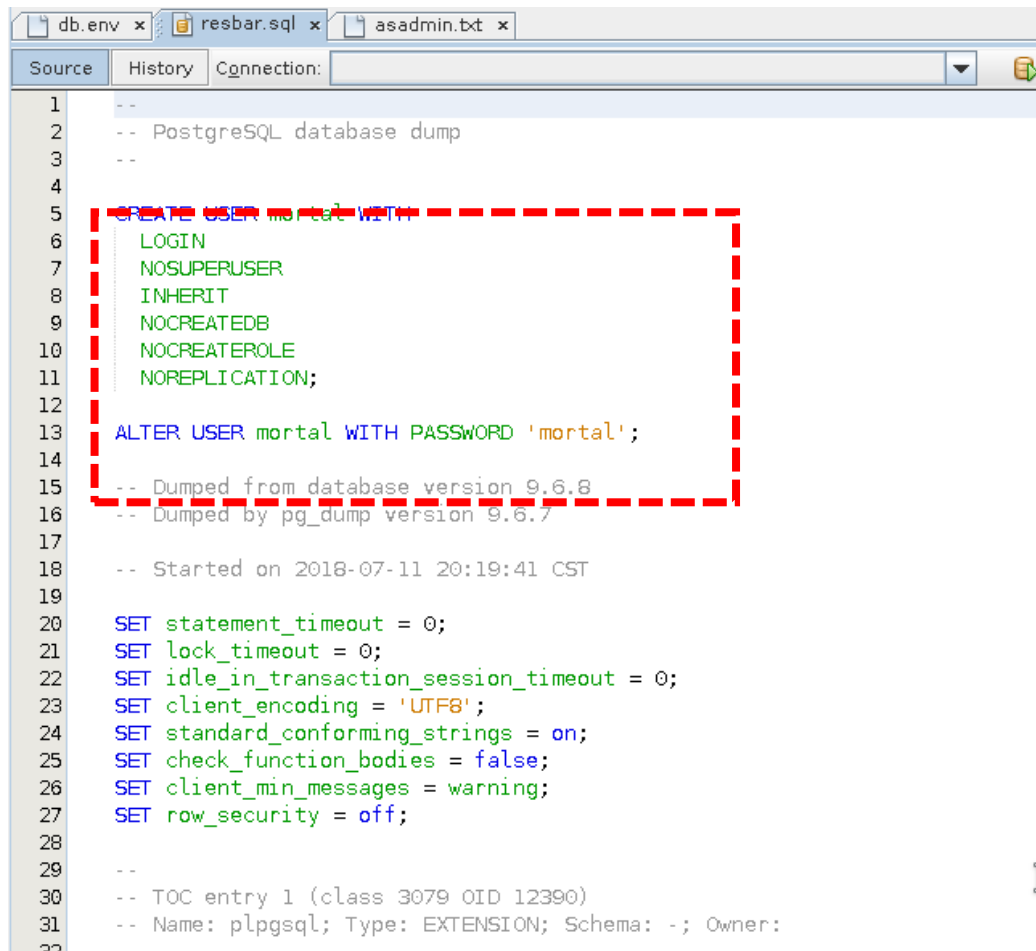
Dockerfile para postgres



Aquí solo son necesarias 2 comandos el Primero es "FROM" que sirve para indicar que imagen docker para postgres vamos a usar

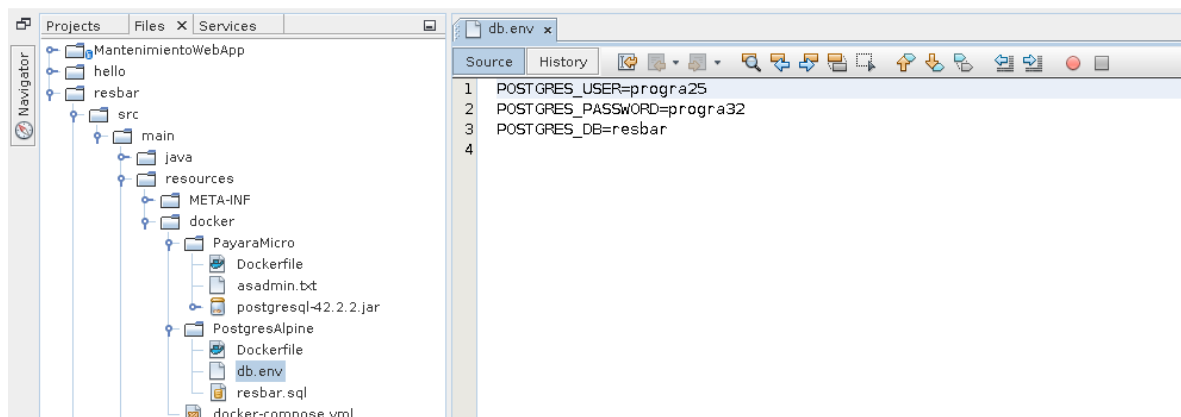
El segundo comando es "ADD" y sirve para añadir el script SQL para la creación de las tablas, este script también incluye sentencias SQL para crear el usuario "mortal" con la contraseña "mortal" que solo posee permisos de lectura y escritura en la base de datos (no tiene permisos para crear tablas o cosas que solo debería poder hacer un usuario administrador y ahí el origen de su nombre, es un simple "mortal" que solo puede administrar los datos de la base de datos pero nunca la estructura de la base de datos).

Este script debe estar junto al Dockerfile para poder añadirlo con el comando "ADD" justo como se puede ver en la "Imagen 1"



```
1  --
2  -- PostgreSQL database dump
3  --
4
5  CREATE USER mortal WITH
6  LOGIN
7  NOSUPERUSER
8  INHERIT
9  NOCREATEDB
10 NOCREATEROLE
11 NOREPLICATION;
12
13 ALTER USER mortal WITH PASSWORD 'mortal';
14
15 -- Dumped from database version 9.6.8
16 -- Dumped by pg_dump version 9.6.7
17
18 -- Started on 2018-07-11 20:19:41 CST
19
20 SET statement_timeout = 0;
21 SET lock_timeout = 0;
22 SET idle_in_transaction_session_timeout = 0;
23 SET client_encoding = 'UTF8';
24 SET standard_conforming_strings = on;
25 SET check_function_bodies = false;
26 SET client_min_messages = warning;
27 SET row_security = off;
28
29 --
30 -- TOC entry 1 (class 3079 OID 12390)
31 -- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
32 --
```

Archivo db.env



Aquí se definen variables de entorno con un usuario y su contraseña (este usuario si tendrá todos los permisos por si necesitamos hacer cambios en la estructura de la base de datos, contrario al usuario “mortal” que solo sirve para acceder a los datos) y el nombre de la base de datos que se va a utilizar.

Dockerfile de payara

```
Dockerfile x
1 # Agregando la imagen 192.168.0.15:5000/
2 FROM payara/micro:latest
3
4 # Descargando el JDBC
5 # RUN wget -O $PAYARA_PATH/postgresql-42.2.2.jar https://jdbc.postgresql.org/download/postgresql-42.2.2.jar (alternativa en lugar de COPY)
6 COPY postgresql-42.2.2.jar $PAYARA_PATH/
7
8 # Agregando el archivo de comandos asadmin
9 COPY asadmin.txt $PAYARA_PATH/
10
11 # Agregando nuestra aplicacion web al directorio autodeploy
12 COPY resbar-1.0-SNAPSHOT.war $PAYARA_PATH/
13
14 # Agregando la libreria, Ejecutamos los comandos asadmin(Pool y Resources) y hacemos Deploy
15 ENTRYPOINT ["java", "-jar", "payara-micro.jar", "--addLibs", "postgresql-42.2.2.jar", "--postbootcommandfile", "asadmin.txt", "--deploy", "resbar-1.0-SNAPSHOT.war"]
16
```

-Comando FROM es el mismo utilizado en el dockerfile de postgres, solo cambia la imagen docker que utilizara, en este caso, payara/micro:latest(recomendación: cambiar “latest” por “5.182”)

-Comando COPY, es una alternativa para el comando “ADD” ya utilizado y solo sirve para copiar el driver de postgres a una ruta establecida(“\$PAYARA_PATH” es equivalente a “/opt/payara”)

-Necesitamos copiar tambien el “asadmin.txt” y **EL PROYECTO EMPAQUETADO** que vamos a desplegar(resbar-1.0-SNAPSHOT.war)

-Comando ENTRYPOINT este solo sirve para ejecutar comando dentro del contenedor(ver documentacion de payara para saber que hace cada uno de ellos o en docker hub de la imagen payara/micro)

Hemos copiado el archivo “asadmin.txt” pero, ¿para que sirve este archivo?, veamos que tiene:

```
asadmin.txt x Dockerfile x
Source History
1 #El pool de conexiones se llama "postgres_resbar_local"
2
3 #Creando el pool de conexiones
4 create-jdbc-connection-pool --datasourceclassname org.postgresql.ds.PGSimpleDataSource --restype javax.sql.DataSource --property user=mon
5
6 #Probando el pool de conexiones
7 ping-connection-pool postgres_resbar_local
8
9 #Creando el resource de conexiones
10 create-jdbc-resource --connectionpoolid postgres_resbar_local jdbc/resbar
```

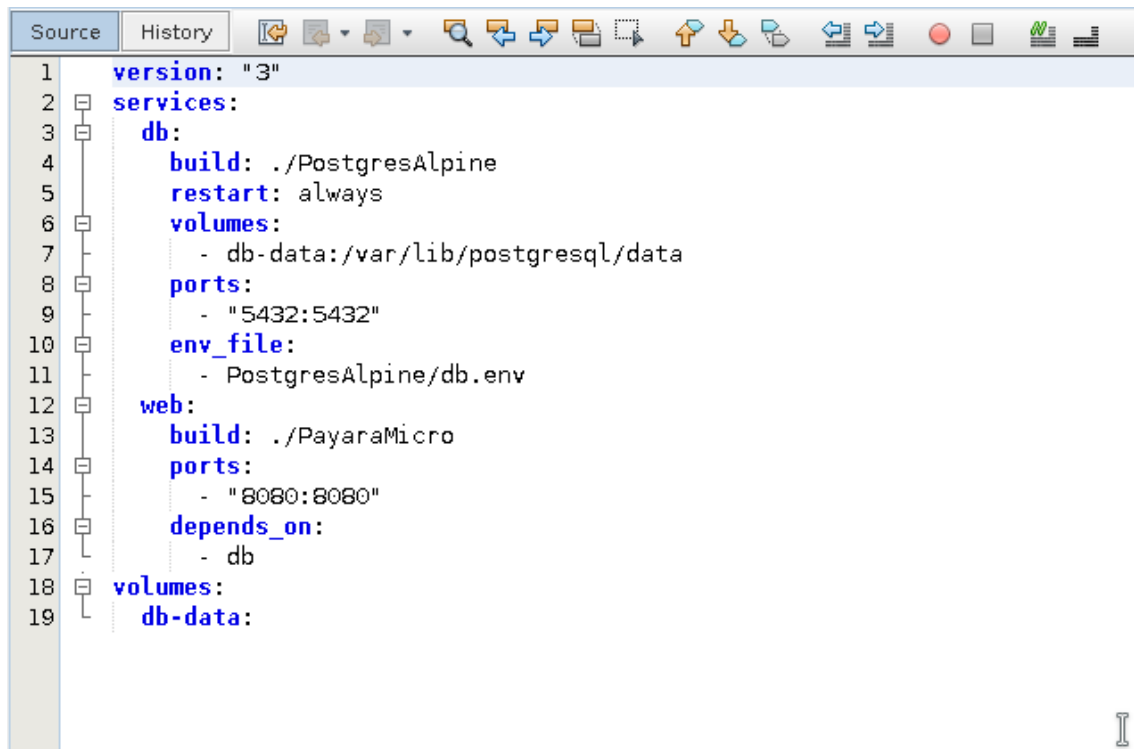
¿Recuerdan todo el relajo que se hace para crear un pool de conexiones y un datasource de forma grafica? Pues, aquí lo hacemos con comandos y es en este archivo(asadmin.txt) donde deben escribir los comandos.

¿Recuerda que anteriormente dijimos que mas adelante se crearia el datasoruce(**pagina 5, letras azules**) es aquí en el archivo asadmin.txt en la linea 10 donde se crea y como pueden ver, final de la linea 10 se puede ver donde dice “jdbc/resbar” y si, es el mismo que pusimos en la unidad de persistencia de la pagina 5?

NOTA EXTRA: “jdbc/resbar” solo es el nombre, podeis cambiarlo por cualquier nombre pero debeara cambiarse tando en el archivo “asadmin.txt” como en la unidad de persistencia. Tener en cuenta esto, para futuros proyectos...

Para mas informacion de cada comando consultar el siguiente enlace:
<https://blog.payara.fish/una-introduccion-a-los-pools-de-conexiones-en-payara-server>

Finalizando el apartado de docker, se adjunta una captura del archivo “docker-compose.yml”



```
1 version: "3"
2 services:
3   db:
4     build: ./PostgresAlpine
5     restart: always
6     volumes:
7       - db-data:/var/lib/postgresql/data
8     ports:
9       - "5432:5432"
10    env_file:
11      - PostgresAlpine/db.env
12  web:
13    build: ./PayaraMicro
14    ports:
15      - "8080:8080"
16    depends_on:
17      - db
18  volumes:
19    db-data:
```

En resumidas cuentas, se declaran los servicios “db” y “web” que usan la sentencia “build” para construir las imágenes en base a los archivos dockerfile que estan en las ruta relativas “./PostgresAlpine” y “./PayaraMicro” ya que estas carpetas poseen un archivo “Dockerfile” cada una, explicado anteriormente.

Con todo lo anterior explicado solo falta añadir un plugin al proyecto maven en su archivo POM.xml que sirva para copiar **EL PROYECTO EMPAQUEDADO desde** el directorio /target donde los guarda maven al dar “clean and build” **hasta** el directorio “src/main/resources/docker/PayaraMicro” y poder construir correctamente la imagen con el archivo dockerfile ya explicado.

Plugin:



```
84
85 <!-- plugin para copiar el war -->
86 <plugin>
87   <groupId>org.codehaus.mojo</groupId>
88   <artifactId>exec-maven-plugin</artifactId>
89   <version>1.6.0</version>
90   <configuration>
91     <executable>cp</executable>
92     <arguments>
93       <argument>${project.basedir}/target/resbar-1.0-SNAPSHOT.war</argument>
94       <argument>${project.basedir}/src/main/resources/docker/PayaraMicro</argument>
95     </arguments>
96   </configuration>
97   <executions>
98     <execution>
99       <phase>verify</phase>
100       <goals>
101         <goal>exec</goal>
102       </goals>
103     </execution>
104   </executions>
105 </plugin>
```

Antes de “dar clean and build” se recomienda detener el contenedor para la base de datos levantado al inicio de la guía. Para esto debera entrar al directorio /bd de donde haya clonado el repositorio y ejecutar “docker-compose down”

Una vez realizado todos los pasos anteriores, solo con darle “clean and build” sera suficiente para copiar el war a un directorio donde pueda usarlo docker(src/main/resources/docker).

Luego de darle clean and build podemos entrar desde la consola al directorio “src/main/resources/docker” y en este directorio ejecutar el comando “docker-compose up”

NOTA: cada vez que se hagan cambios en el proyecto se debera borrar la imagen “docker-web:latest” con el comando “docker-rmi docker-web:latest” asi como tambien detener los contenedores ejecutandose actualmente para ello se debe entrar al directorio “src/main/resources/docker” y ejecutar “docker-compose down”

DESAFIO: Añadir el plugin de docker y su dependencia al POM.xml para que con solo hacer “clean and build” puedan desplegar el proyecto.(Esta en la guia de maven como hacerlo xd)

DUDAS: Pueden hacerlas en el grupo de facebook o escribiendole a alguno de los instructores :v

SOLUCIÓN: Pueden ver en el repositorio <https://bitbucket.org/rcarlos97/resbar/src/master/> , este tiene 2 branch: **master** y **solucionGuia**, al clonarlo pueden cambiar a de branch con “git checkout solucionGuia”, ingresar al directorio “/src/main/resources/docker” y hacer docker “compose up”, para ver que el proyecto se ha desplegado correctamente solo se abre el navegador y se accede a la direccion <http://localhost:8080/resbar-1.0-SNAPSHOT>

CURIOSIDAD: Si hacen lo mismo del paso “docker-compose up” en la branch **master** podran ingresar a la url: <http://localhost:8080/resbar-1.0-SNAPSHOT/ws/categoria/all> y eso, es REST.