VIRTUALIZACIÓN con Proxmox

Elaborado por:

Edwin Paul Paute Chalco
Juan Francisco Quizhpi Fajardo

Docente:

Cristian Fernando Timbi Sisalima

Tema:

Proxmox

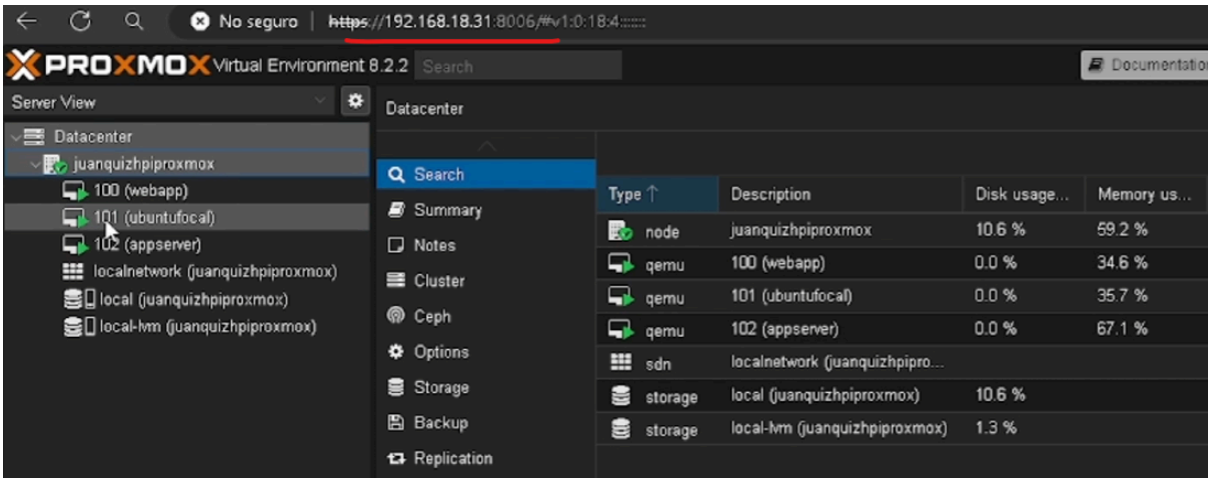
Periodo Académico:

64

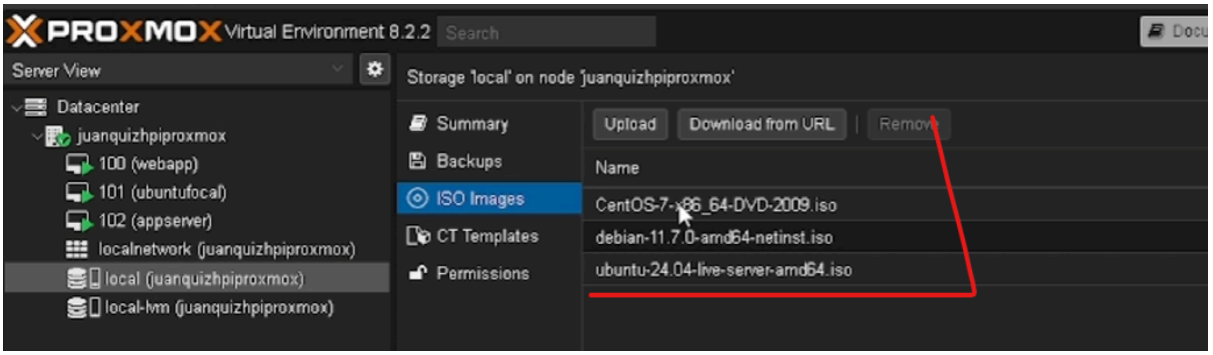
Recomendaciones iniciales
Para crear el proyecto al momento de configurar las máquinas debemos empezar por la VM de base de datos, VM del BackEnd y finalmente por la del FrontEnd

Configuración de Proxmox

Una vez descargado e instalado proxmox de tipo bare metal ingresamos a la página web de esta por medio de la IP 192.168.18.31:8006 una vez dentro ingresamos nuestras credenciales y procedemos a crear las imágenes



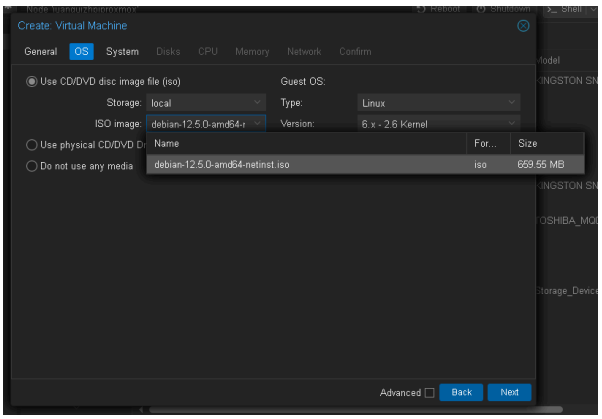
También vamos a proceder a descargar las imágenes de 3 distribuciones de linux en este caso optamos por Centos, Debian y Ubuntu Server.

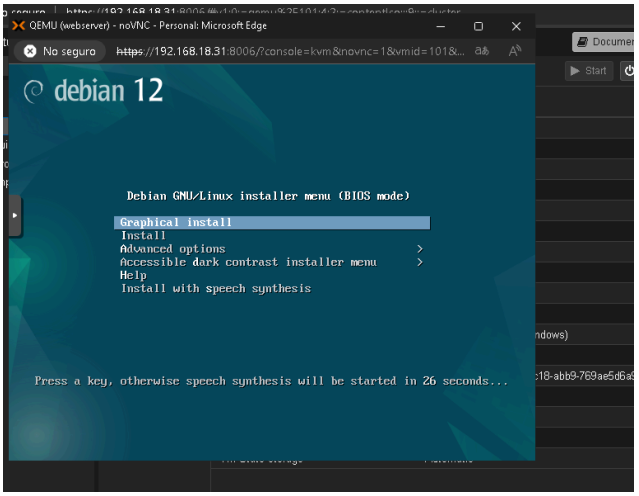


Creación de las máquinas virtuales

Ahora procedemos a instalar las máquinas virtuales dentro de nuestro hipervisor de proxmox
Máquina virtual para FrontEnd o angular (Debian)

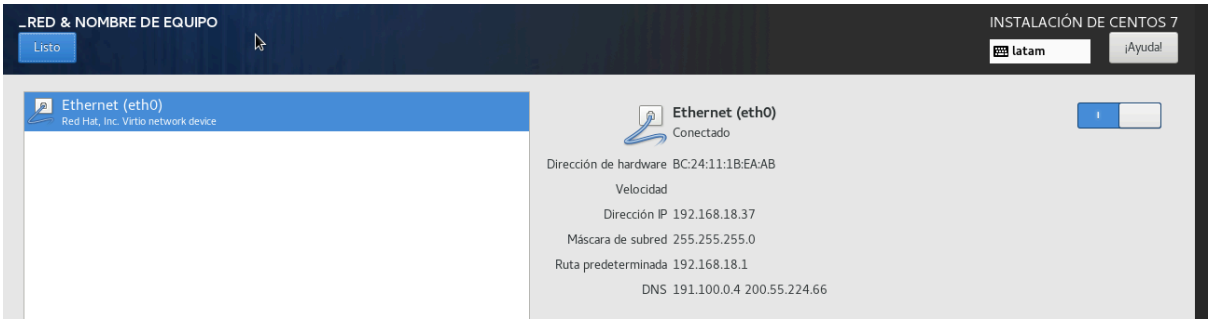
Instalamos en primer lugar nuestra imagen de Debian





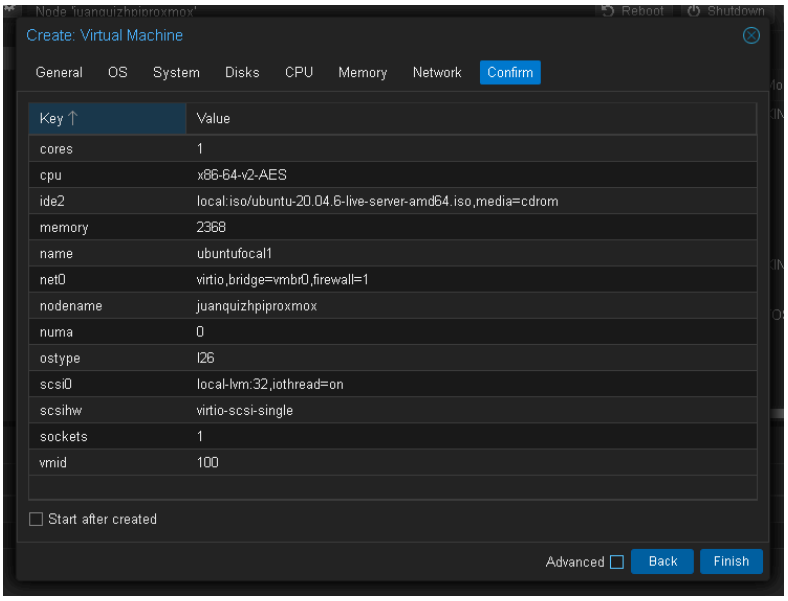
Máquina virtual para BackEnd (Centos 7)

Ahora procedemos a instalar Centos dentro de nuestro hipervisor, un paso importante en la instalación será la activación de interfaz de red para que por medio de DHCP se asigne la IP automáticamente.

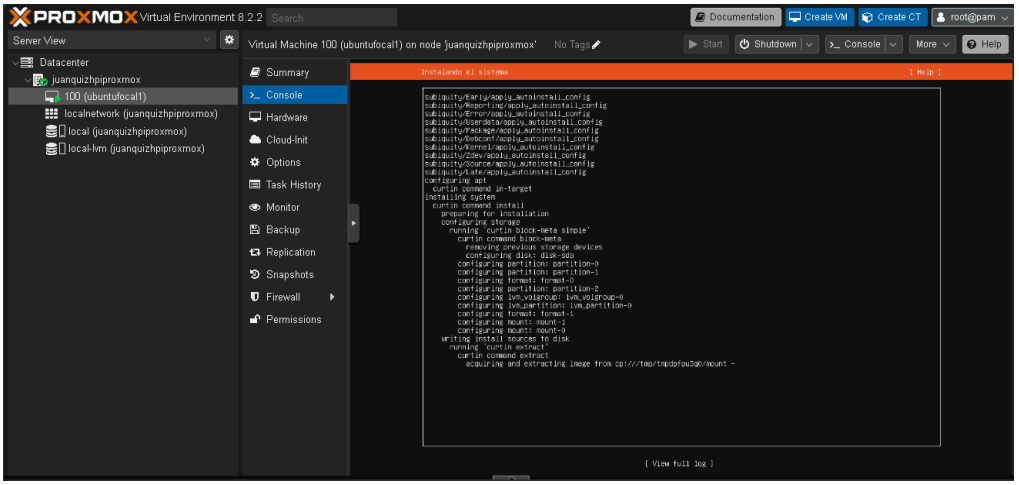


Máquina virtual para base de datos (Ubuntu)

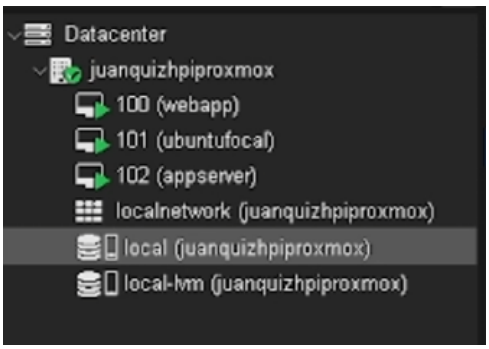
Ahora procedemos a instalar ubuntu dentro de nuestro entorno de hipervisor proxmox



Realizamos una instalación típica de una distribución de linux cabe mencionar que se debe instalar SSH para poder usar la VM instalada.



Instancias creadas para el despliegue



Configuración de la VM para el FrontEnd

En primer lugar vamos a ejecutar los siguientes comando para actualizar los paquetes de Nginx

```
sudo apt update
sudo apt install nginx -y
```

Luego ejecutaremos los comandos necesarios para la instalación de Git y Node.js

```
sudo apt install git -y
curl -sL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Ahora procedemos a ejecutar los comandos para la instalación de Angular para poder crear el proyecto en modo producción.

```
sudo npm install -g @angular/cli@17
```

Ahora procedemos a clonar nuestro proyecto de angular y entramos dentro de la carpeta donde se encuentra nuestro repositorio del frontend

```
git clone https://github.com/JuanQuizhpi/frontendcarroapp.git
cd frontendcarroapp
```

Dentro del repositorio procedemos a la construcción de la aplicación con los siguientes comandos

```
npm install
ng build --configuration=production
```

Ahora procederemos a la configuración de Nginx para servir la aplicación de Angular ejecutamos el siguiente comando

```
sudo nano /etc/nginx/sites-available/default
```

Vamos a configurar nuestro nuestro Nginx, se debe considerar el proxy_pass para redireccionar las peticiones a la máquina donde se encuentra el backend. De este modo tenemos que nuestra configuración queda de la siguiente manera.

```
server {
        listen 80;
#       listen [::]:80;

        server_name example.com;

        root /home/debianweb/frontendcarroapp/dist/frontendcarroapp/browser;
        index index.html;

        location / {
                try_files $uri $uri/ =404;
        }
}

server{
        listen 8080;
        location /{
                proxy_pass http://192.168.18.37:8080/;

        }


}
```

Lo que hacemos al configurar el proxy pass es reenviar las solicitudes HTTP recibidas a otro servidor, en este caso denominado servidor de destino o backend. En contexto Nginx actuará como como un intermediario entre los clientes de frontend y el servidor de backend el cual se encargará del manejo del manejo de las solicitudes para consumir los servicios.

Ahora reiniciamos nuestro nginx y procedemos a ver el estatus de nuestro servidor de servidor con los siguientes comandos

```
sudo systemctl restart nginx
sudo systemctl status nginx
```

Configuración de la VM del BackEnd

En primer lugar, tomando en cuenta que estamos trabajando en Centos los comandos van a variar para la configuración, seguiremos los siguientes pasos. Primero vemos que nuestro sistema está actualizado con el siguiente comando.

```
sudo yum update -y
```

Luego vamos a realizar la instalación de OpenJDK 11 para esto nos ayudaremos del siguiente comando

```
sudo yum install -y java-11-openjdk-devel
```

Ahora vamos a proceder a la instalación del WildFly para esto vamos a navegar hacia el directorio donde deseamos descargar el WildFly con el siguiente comando

```
cd /opt
```

Ahora vamos a proceder a descargar WildFly con ayuda del wget tomar en cuenta descargar este antes de usar su comando

```
sudo yum install wget
sudo wget https://github.com/wildfly/wildfly/releases/download/27.0.0.Final/wildfly-27.0.
```

Ahora procedemos a extraer el contenido del archivo descargado con el siguiente comando

```
sudo tar -xvzf wildfly-27.0.0.Final.tar.gz
```

Ahora cambiamos el nombre del directorio para facilitar el uso del mismo con el siguiente comando

```
sudo mv wildfly-27.0.0.Final /opt/wildfly
```

También cambiamos los permisos para permitir la ejecución del mismo con el siguiente comando

```
sudo chown -R atlas_ep26:atlas_ep26 /opt/wildfly
```

Ahora procedemos a descargar el driver de JDBC para PostgreSQL y lo ubicamos el el directorio de deployments con los siguientes comandos

```
cd /opt/wildfly/standalone/deployments
sudo wget https://jdbc.postgresql.org/download/postgresql-42.3.1.jar
```

También para el correcto funcionamiento vamos a descargar las carpetas del siguiente repositorio y los vamos a ubicar en la carpeta modules del WildFly

```
sudo yum install -y git
git clone https://github.com/JuanQuizhpi/libreriasJDBC.git
cd libreriasJDBC
sudo cp -r ./ /opt/wildfly/modules/
```

Ahora procedemos a configurar el standalone.xml con ayuda del siguiente comando

```
sudo nano /opt/wildfly/standalone/configuration/standalone.xml
```

Una vez dentro debemos tener configurado el standalone.xml de la siguiente manera. Recordar que cada vez que editemos ejecutaremos (Ctrl+X, Y, Enter) para salir. En este caso configuraremos el datasource con la dirección IP donde se encuentra nuestra VM de la base de datos. Esta es 192.168.18.36.

```
<datasource jndi-name="java:/jdbc/demojpaDS" pool-name="demojpaDS" enabled="true" use-java-context="true" >
<connection-url>jdbc:postgresql://192.168.18.36:5432/carrobase</connection-url>
<driver>postgresql</driver>
<security>
        <user-name>carroadmin</user-name>
        <password>admin</password>
</security>
</datasource>
<drivers>
    <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
    </driver>
    <driver name="postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
    </driver>
```

Una vez completado esa parte abrimos los puertos necesarios para la administración de WildFly con los siguientes comandos

```
sudo firewall-cmd --zone=public --add-port=8080/tcp --permanent
sudo firewall-cmd --zone=public --add-port=9990/tcp --permanent
sudo firewall-cmd --reload
```

Ahora regresamos a la carpeta home en donde en un principio descargamos las librerías JDBC. Una vez ahí ejecutamos los siguientes comandos. Además en esta parte lo que vamos a hacer copiar el war en deployments del WildFly

```
git clone https://github.com/JuanQuizhpi/carrobe.git
cd carrobe
sudo cp target/carrobe.war /opt/wildfly/standalone/deployments/
```

Finalmente para poder iniciar el servidor de aplicaciones ejecutamos el siguiente comando

```
sudo sh /opt/wildfly/bin/standalone.sh -b 0.0.0.0
```

Configuración del servidor de Base de datos

En primera instancia vamos a actualizar y instalar todo lo necesario para nuestra VM con los siguientes comandos

```
sudo apt update && sudo apt upgrade -y
```

Ahora procedemos a instalar postgres para este caso se instala la versión 12

```
sudo apt install -y postgresql postgresql-contrib
```

Ahora procedemos a iniciar el postgres con el siguiente comando

```
sudo systemctl start postgresql
```

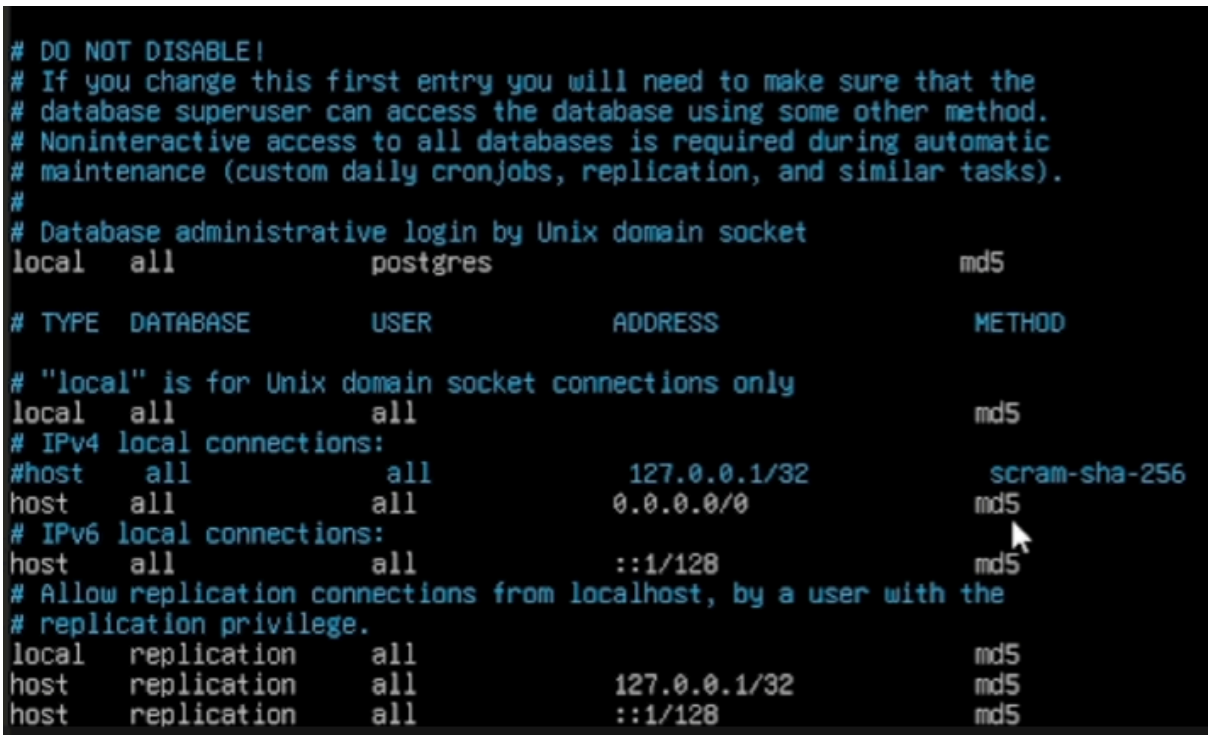
Ahora verificamos el status de nuestro postgres con el siguiente comando

```
sudo systemctl status postgresql
```

Ahora ingresamos al archivo de configuración de postgres para permitir las conexiones desde otras máquinas. Usaremos el siguiente comando.

```
sudo nano /etc/postgresql/12/main/pg_hba.conf
```

Una vez dentro cambiamos la línea que controla el acceso a la base de datos para que permita conexiones de todas las IPs (0.0.0.0/0). Para la primera vez del **md5** en el usuario postgres debe ser **peer** o no tendremos acceso para crear la tabla.

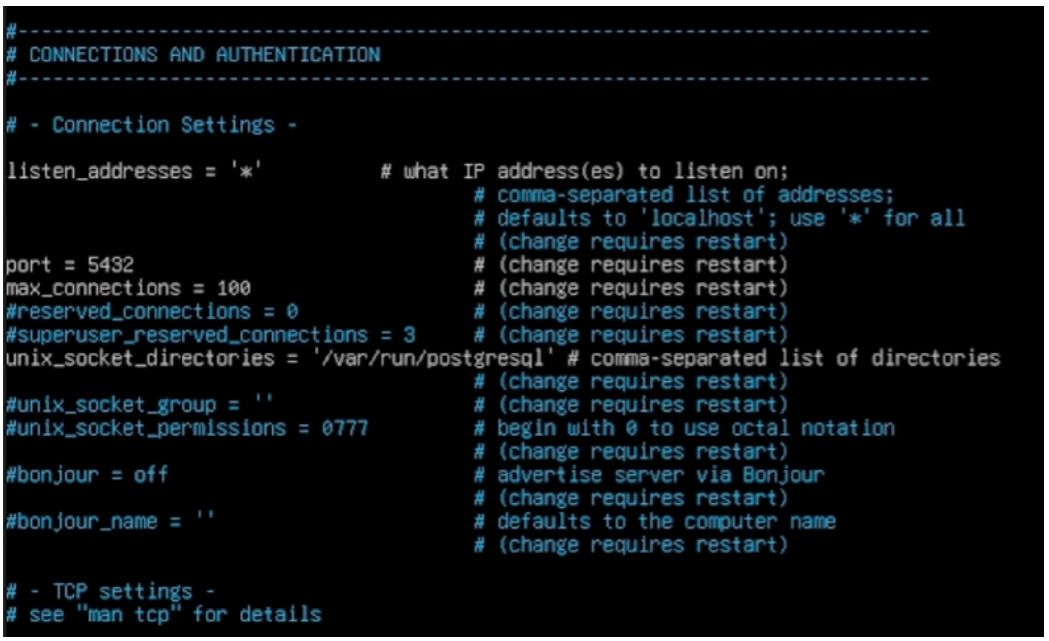


```
# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local   all             postgres                                md5

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     md5
# IPv4 local connections:
#host    all             all             127.0.0.1/32            scram-sha-256
host    all             all             0.0.0.0/0               md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     md5
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
```

Ahora también vamos a entrar al siguiente archivo de configuraciones de postgres para permitir que todas las interfaces de red escuchen a PostgresSQL. Con el siguiente comando

```
sudo nano /etc/postgresql/12/main/postgresql.conf
```

Ahora buscaremos la siguiente línea y la configuramos de esta manera



```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#reserved_connections = 0               # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''                      # defaults to the computer name
                                # (change requires restart)

# - TCP settings -
# see "man tcp" for details
```

Una vez terminado ese proceso ejecutamos el comando para reiniciar y aplicar las configuraciones

```
sudo systemctl restart postgresql
```

Ahora vamos a ingresar a la consola de postgres para crear el usuario con el siguiente comando.

```
sudo -u postgres psql
```

En esta parte ya estaremos en la consola y ejecutamos los siguientes comandos para crear la base de datos estos variarán en función del datasource ya configurado en el wildfly.

```
##Comandos en la consola de postgres

CREATE DATABASE carrobase;
CREATE USER carroadmin WITH PASSWORD 'admin';
GRANT ALL PRIVILEGES ON DATABASE carrobase TO carroadmin;
#Para salir de la consola de postgres
\q
```

Ahora vamos a permitir las conexiones al puerto 5432 de PostgreSQL con el siguiente comando

```
sudo ufw allow 5432/tcp
```

Ahora debemos considerar lo siguiente cuando se cree la base de datos por primera vez se debe considerar que la configuración del usuario postgres deberá estar en modo peer no md5. Una vez ya se haya creado el data source procedemos a cambiarle la clave al usuario con los siguientes comandos.

```
sudo -u postgres psql

#desde postgres

ALTER USER postgres WITH PASSWORD 'admin';
\q

#Fuera de postgres

sudo systemctl restart postgresql
```

Ahora para cuando nos de el error `Acceso denegado al schema public` debemos ejecutar este comando para arreglarlos.

```
psql -U postgres -d carrobase

#Desde la consola psql
#Permitir al usuario usar el esquema `public`
GRANT USAGE ON SCHEMA public TO carroadmin;

#Permitir al usuario crear objetos en el esquema `public`
GRANT CREATE ON SCHEMA public TO carroadmin;

#Permitir al usuario acceder y modificar todas las tablas existentes en el esquema `public`
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO carroadmin;

#Configurar permisos predeterminados para futuras tablas
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO

\dn+

\dp
```

Ahora nos vamos a conectar al postgres de nuevo para verificar que se creó la base con el siguiente comando

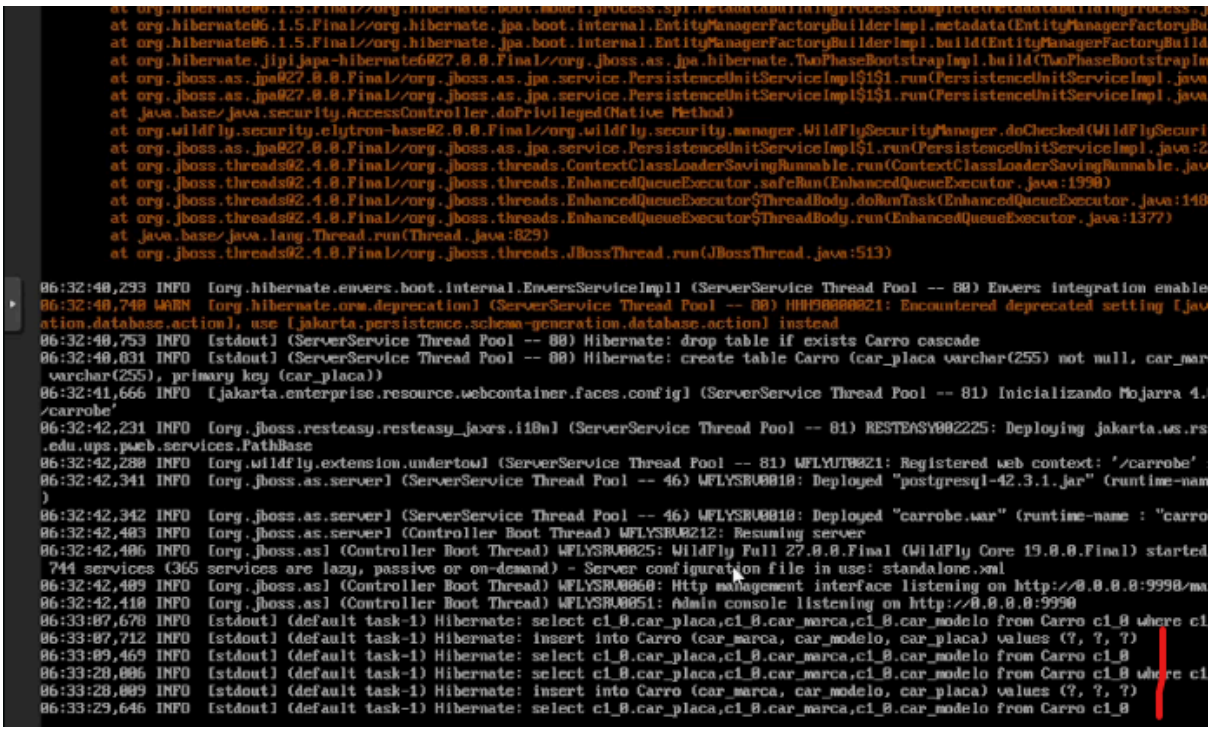
```
psql -U carroadmin carrobase -W
```

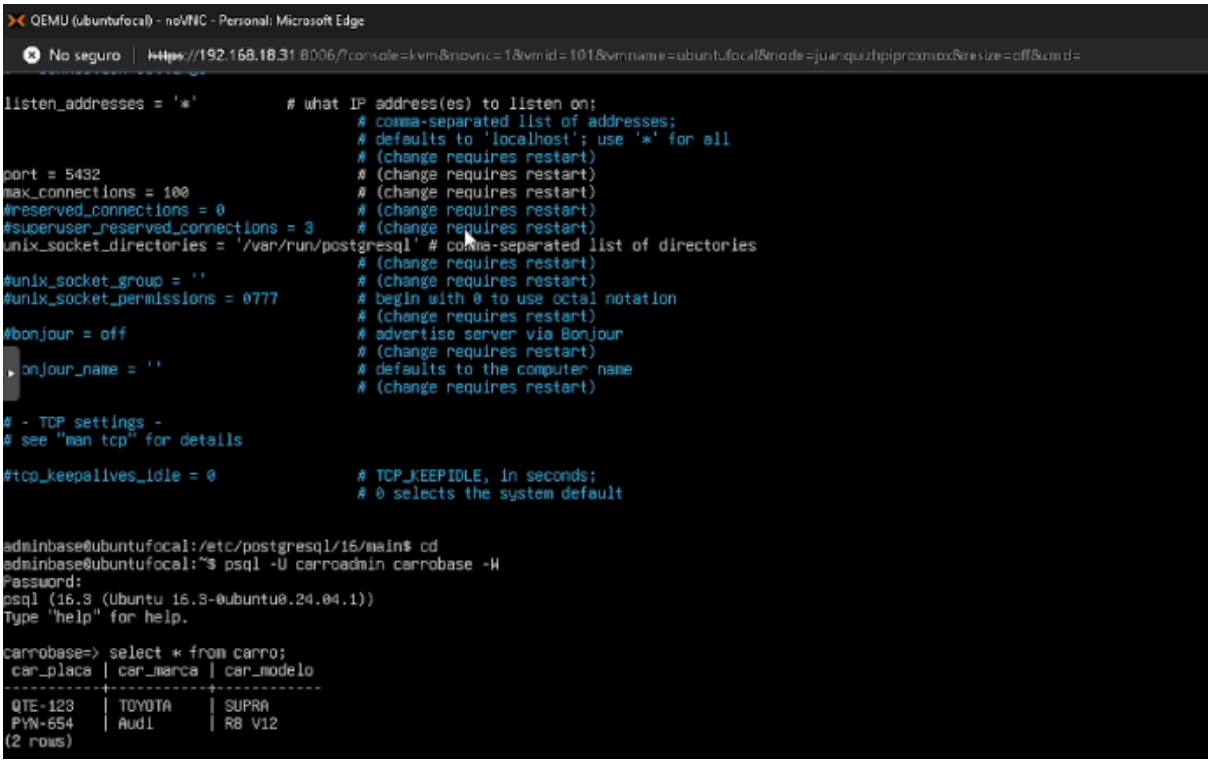
Datos a tomar en cuenta

Para conectar el FrontEnd con el BackEnd en la ruta con la que consumimos los servicios cambiamos el **localhost** por la IP nuestra de la máquina de FrontEnd y gracias al proxy_pass se redireccionará a la máquina del BackEnd . En este caso se usará la siguiente dirección IP 192.168.18.35.

```
private apiUrl = 'http://192.168.18.35:8080/carrobe/rs/carros';

constructor(private http: HttpClient) { }

save(carro: Carro) {
  return this.http.post<any>("http://192.168.18.35:8080/carrobe/rs/carros", carro)
}

getAll(){
  return this.http.get<any>("http://192.168.18.35:8080/carrobe/rs/carros/list");
 }

 delete(carro: Carro) {
  const url = `http://192.168.18.35:8080/carrobe/rs/carros?placa=${carro.placa}`;
  return this.http.delete(url);
}

update(Carro: Carro) {
  const url = 'http://192.168.18.35:8080/carrobe/rs/carros';
  return this.http.put(url, Carro);
}
}
```

Prueba de la Aplicación





Videos

https://youtu.be/QalvqveEDtg

https://youtu.be/8CydxTlnx-8

Enlaces a los repositorios usados para el despliegue

https://github.com/JuanQuizhpi/frontendcarroapp.git
https://github.com/JuanQuizhpi/libreriasJDBC.git
https://github.com/JuanQuizhpi/carrobe.git