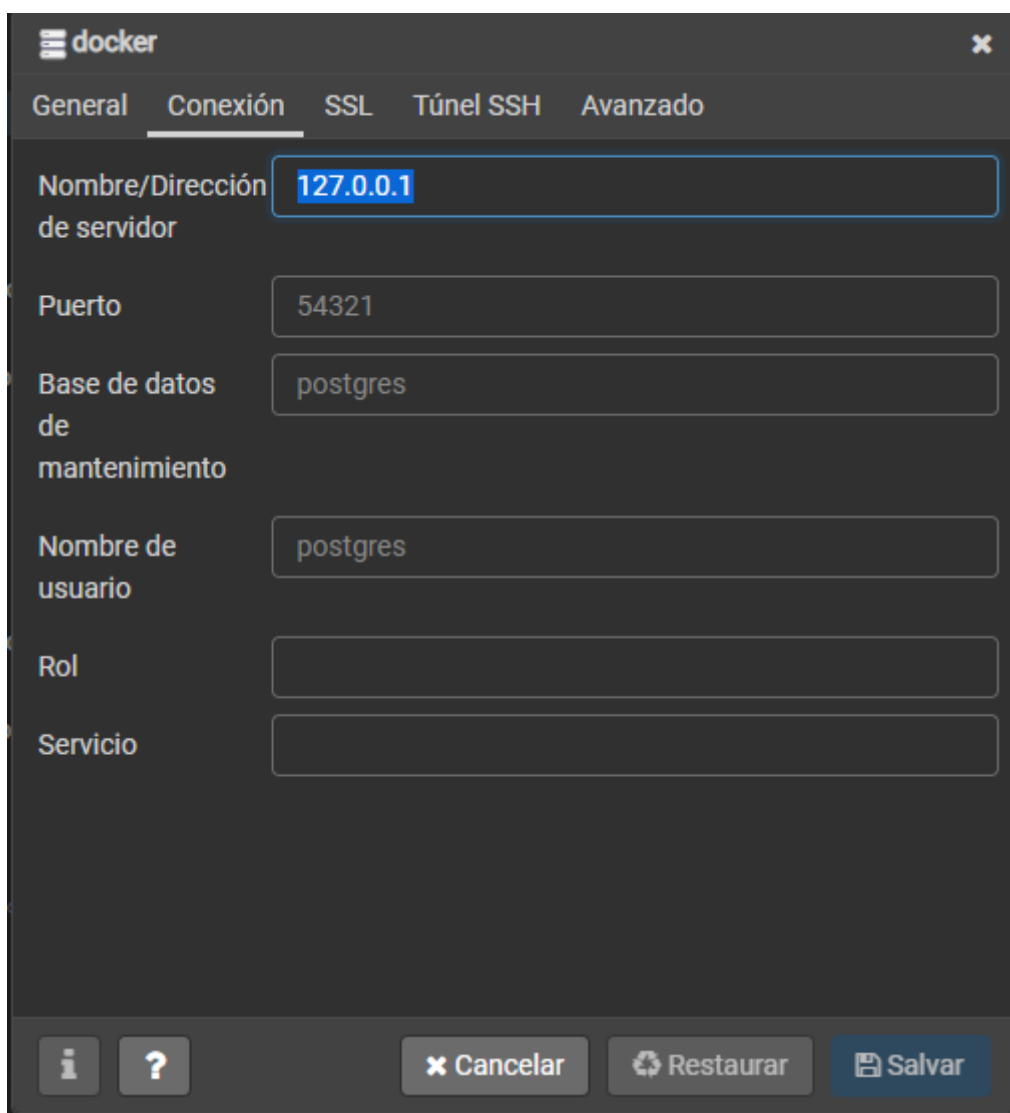


Creación de contenedor Docker con PostgreSQL

- Lo primero es ejecutar el comando **docker pull postgres** para que realice la descarga desde el repositorio donde se encuentra **postgres**.
- Para verificar la descarga se usa **docker image ls**.
- Aquí usamos **docker run --name some-postgres -e POSTGRES_PASSWORD=postgres -d postgres** para ejecutar la imagen de postgres y que se cree el contenedor en docker.
- Para ingresar y utilizar postgres desde la consola se puede usar **docker run -it --rm --link some-postgres:postgres postgres psql -h postgres -U postgres**. La opción de link permite que dos contenedores puedan comunicarse.

Conexión local de MySQL.

- Para hacer la prueba conectamos con PgAdmin y utilizamos las siguientes credenciales:



docker

General **Conexión** SSL Túnel SSH Avanzado

Nombre/Dirección de servidor: 127.0.0.1

Puerto: 54321

Base de datos de mantenimiento: postgres

Nombre de usuario: postgres

Rol:

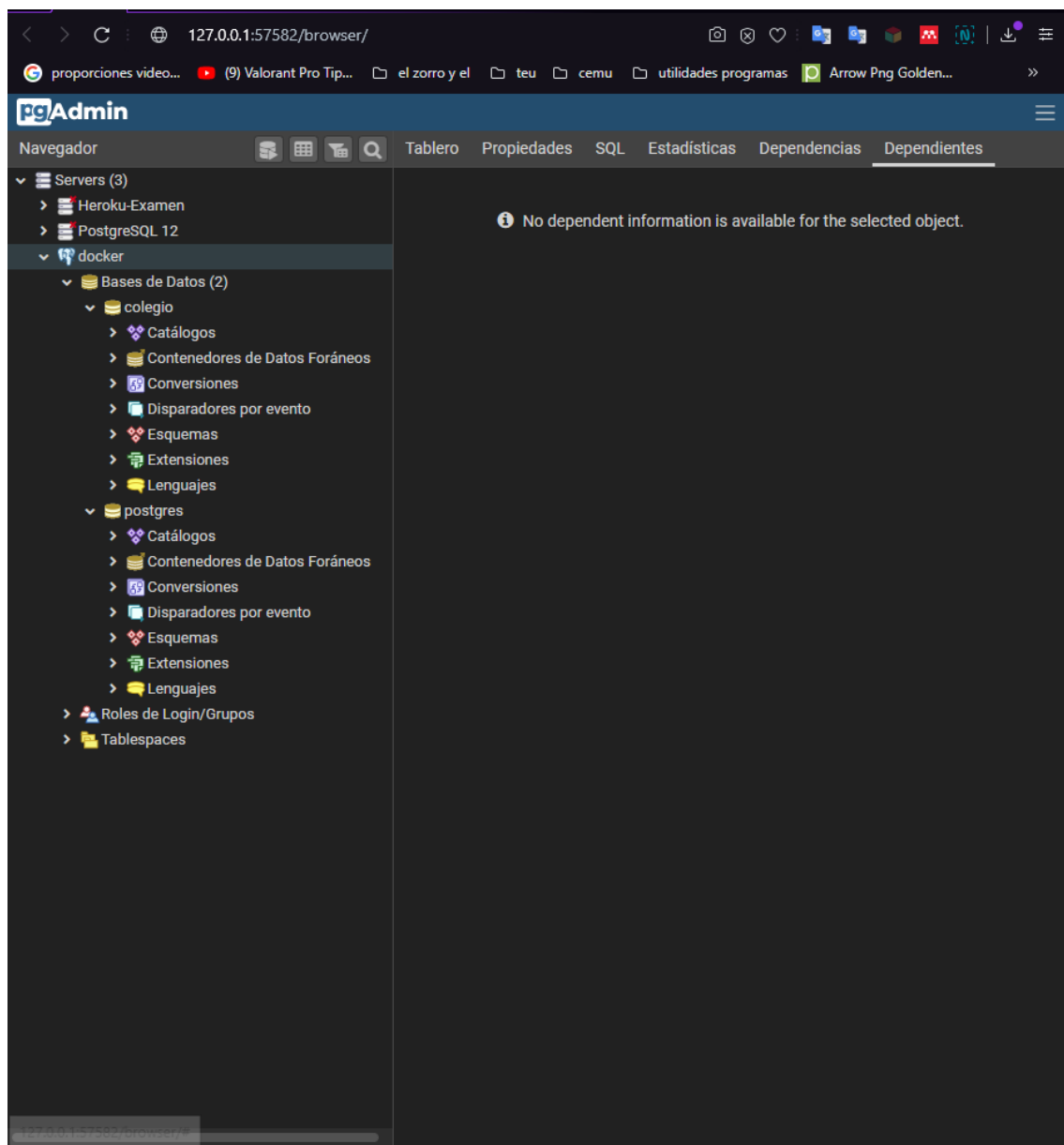
Servicio:

Cancelar Restaurar Salvar

Base de datos visto desde la consola:

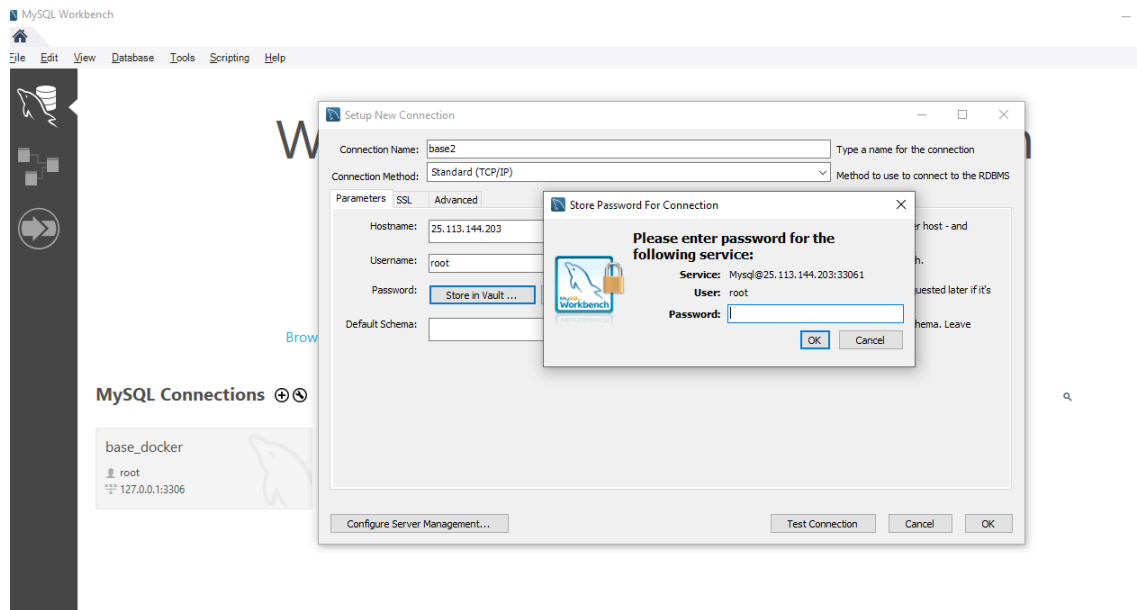
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ecb7863dfa7	marcos/node-web-app	"docker-entrypoint.s..."	14 minutes ago	Up 14 minutes	0.0.0.0:49160->8080/tcp	practical_mccarthy
ba542e627e1c	postgres	"docker-entrypoint.s..."	4 hours ago	Up 4 hours	5432/tcp	kind_shockley
8c1b5864b9d0	postgres	"docker-entrypoint.s..."	4 hours ago	Up 4 hours	0.0.0.0:54321->5432/tcp	some-postgres
357ca3d93302	mysql:5.7	"docker-entrypoint.s..."	5 hours ago	Up 5 hours	33060/tcp, 0.0.0.0:33061->3306/tcp	mysql

Vista local de la base de datos:



Conexión de base de datos con Hamachi:

Base de datos remota:



CONTENEDOR DOCKER PARA SERVIDOR WEB CON NODE.JS

Se crea un carpeta en donde se crea un archivo **package.json** con el siguiente código y se debe correr el comando **npm install** y se crea un archivo **package-lock.json**.

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "First Last <first.last@example.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.16.1"
  }
}
```

Finalmente se crea un archivo **server.js** con el siguiente contenido:

```

'use strict';

const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World');
});

app.listen(PORT, HOST);
console.log(`Running on http://${HOST}:${PORT}`);

```

El siguiente archivo que se crea es un **Dockerfile** que permitirá generar el contenedor docker para node.js.

```

FROM node:10

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm ci --only=production

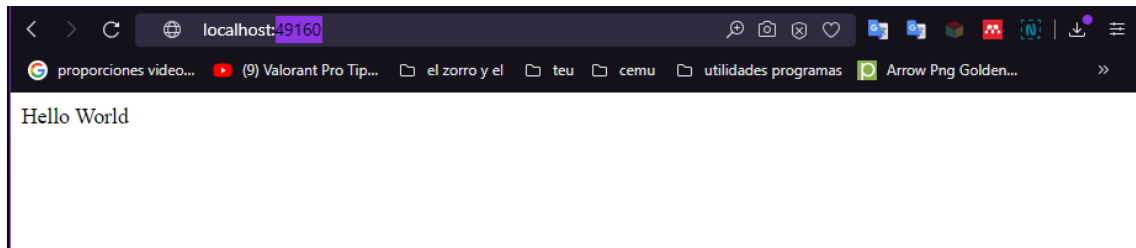
# Bundle app source
COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]

```

Consumo local del servicio web con JS.

Lo siguiente es construir la imagen que se descarga con **docker build -t <your username>/node-web-app** . y luego correr la imagen con **docker run -p 49160:8080 -d <your username>/node-web-app**, para verificar que esté funcionando ingresamos al localhost con su respectivo puerto.



Consumo remoto con Hamachi:

