**To Setup Postgres database using the custom dockerfile and to create a pod from it.**

* Generally, We have a dockerfile to install and postgres and start its service.
* But, only that setup is not okay to connect to that specific pod or container using DBeaver to access it from anywhere.
* Hence, we have 1st choosen an approach to edit 2 of the configuration files by installing nano editor in the user and then changing postgresql.conf and pg\_hba.conf files.
* **Pros:**

It is working successfully with the docker, because we can restart the docker to get the changes to reflect.

* **Con’s:**

It is not suitable with the pods of K8’s, because when we restart a pod, A new pod is getting created with its default configuration but not the configuration which we have made.

* Then, we have made a decision to copy the 2 conf files from docker/k8’s pod to our local directory 🡪 edit the necessary changes 🡪 Include them inside our dockerfile 🡪 To build a docker image 🡪 push the docker image into the local docker registry. So that we can include that image inside our pods.yaml file.

**1) Therefore, I’ve copied the files from k8’s pod to local directory using below copy commands**

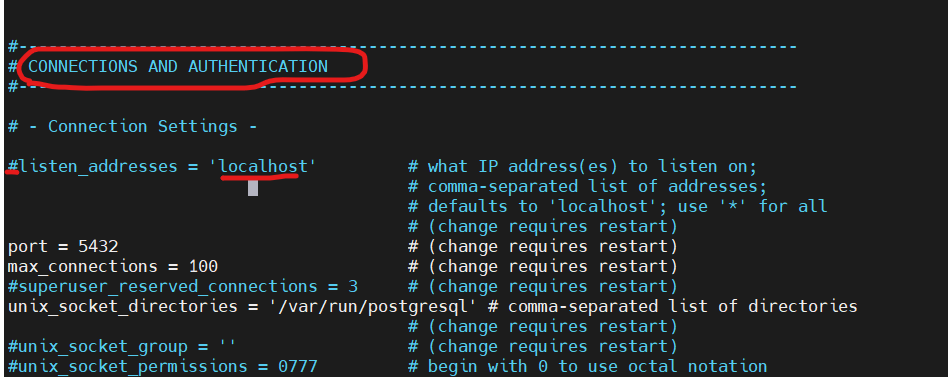
* kubectl cp postgres-pod-f7bb5df55-v56wr:/etc/postgresql/14/main/**postgresql.conf** /home/brrsoftwares/kubernetes\_files/postgres/**postgresql.conf**
* kubectl cp postgres-pod-f7bb5df55-v56wr:/etc/postgresql/14/main/**pg\_hba.conf** /home/brrsoftwares/kubernetes\_files/postgres/**pg\_hba.conf\_bkp**

**2) Changes that I’ve made to my conf files are shown below.**

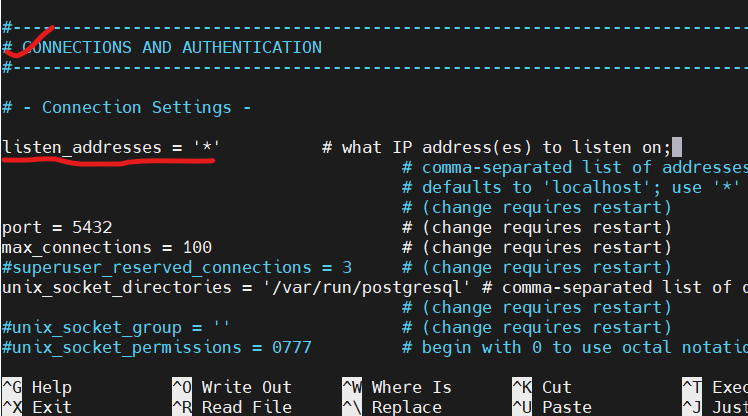
**Editing the postgresql.conf file:**

**Use nano to edit,**

**sudo nano postgresql.conf**

****

So, now scroll down and make changes under **connections and authentication** section by commenting the **listen\_addresses line** and using **‘ \* ‘** in the place of localhost as shown below.

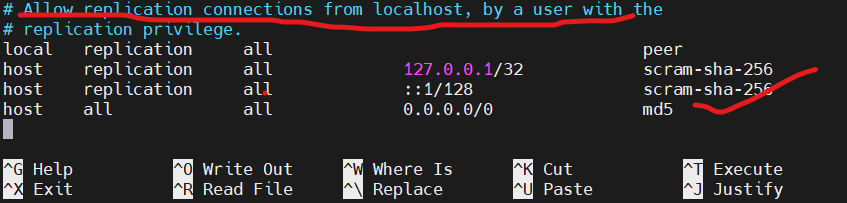


**Note:**

The **listen\_addresses** parameter in the **postgresql.conf** file controls the network interfaces on which PostgreSQL listens for incoming connections. Setting **listen\_addresses** to '\*' allows PostgreSQL to listen on all available network interfaces, enabling connections from both localhost and remote machines.

**Editing the pg\_hba.conf file.**

Add the last line as shown below,



**Which means:**

On the other hand, the **pg\_hba.conf** file specifies the client authentication rules for incoming connections to the PostgreSQL server. The line **host all all all 0.0.0.0/0 md5** in **pg\_hba.conf** allows connections from all IP addresses (**0.0.0.0/0**) to all databases (**all**) for all users (**all**) using the **md5** authentication method. This line essentially permits remote connections from any IP address to the PostgreSQL server, provided the connecting clients provide valid credentials.

**So, using the above copy files to build the Dockerfile**

**Dockerfile of Postgres:**

**FROM ubuntu:latest**

**# Install PostgreSQL and nano**

**RUN apt-get update && \**

**DEBIAN\_FRONTEND=noninteractive apt-get install -y postgresql nano && \**

**apt-get clean**

**# Copy the local postgresql.conf file to the container**

**COPY postgresql.conf /etc/postgresql/14/main/postgresql.conf**

**COPY pg\_hba.conf /etc/postgresql/14/main/pg\_hba.conf**

**# Access PostgreSQL prompt and set password for 'postgres' user**

**RUN service postgresql start && \**

**su - postgres -c "psql -c \"ALTER USER postgres PASSWORD 'Welcome1';\"" && \**

**service postgresql stop**

**# Start PostgreSQL service**

**CMD service postgresql start && \**

**tail -f /dev/null**

**Note:**

* **The DEBIAN\_FRONTEND environment variable is specifically designed for use in Debian-based operating systems, such as Ubuntu, Linux Mint, and Debian itself. These operating systems utilize the apt-get package manager for software package management, and DEBIAN\_FRONTEND is used to control how apt-get interacts with the user during package installations or upgrades.**
* **When apt-get operates in noninteractive mode, it automatically assumes default or predetermined responses to any prompts it would normally display. This allows for automated installation and configuration of packages without requiring manual intervention.**
* **In this Dockerfile, Inside ubuntu image, we have installed the postgres & nano editor, After then copied the locally edited 2 conf files and also started the service of the postgres then switched to the postgres user, switched to database using psql and then altered the password for the user postgres as Welcome1. After doing all changes, stopped the service of the postgres and restarted it, to get the changes to get reflected.**

**Creating a Docker image and pushing it to local docker registry:**

To build a docker image from the Dockerfile run below command.

**sudo docker build -t <dockerimagename> -f <Dockerfile name> .**

**(or)**

**sudo docker build -t <dockerimagename> .**

Firstly, we need to make sure whether our local registry container is up and running and then tagging the docker image with any of your choice name. using below command

**sudo docker tag <dockerimagename> 192.168.29.54:443/<anydockerimagename>**

after tagging,

push the docker image:

**sudo docker push192.168.29.54:443/<anydockerimagename>**

Now, confirm it by checking inside your local docker registry.

i.e., inside your browser 🡪 **192.163.29.54:443/v2/\_catalog**

Therefore, as our custom docker image is available inside our local registry, we can create a pod using it.

Hence, I have used template of the pods.yaml file and service.yaml file from the k8’s official webisite,

**Link :** https://kubernetes.io/docs/tutorials/stateless-application/guestbook/

**Using docker image inside pods.yaml file to create pod**

**sudo nano pods.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: postgres-pod

labels:

app: postgres

role: leader

tier: backend

spec:

replicas: 1

selector:

matchLabels:

app: postgres

template:

metadata:

labels:

app: postgres

role: leader

tier: backend

spec:

containers:

- name: leader

image: "192.168.29.54:443/postgres\_latest" **# My docker image inside local registry.**

ports:

- containerPort: 5432

imagePullSecrets:

- name: password

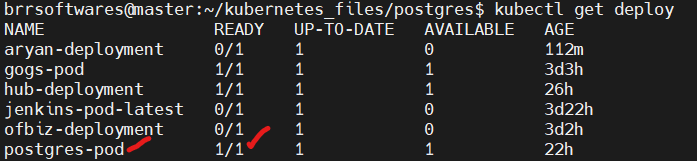
Save the file and run it using below command:

**kubectl apply -f pods.yaml**

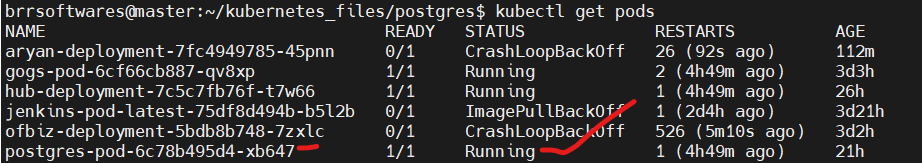
Once the command as got executed,

you can check your deployments and pods using :

**kubectl get deploy**

****

**kubectl get pods**

****

Now, our pod is running, but without service file we cannot connect to our application just by running a pod.

**So, to expose our pod to external world, we need to use service.yaml file**

**sudo nano service.yaml**

apiVersion: v1

kind: Service

metadata:

name: postgres-service

labels:

app: postgres

role: leader

tier: backend

spec:

ports:

- port: 5432

targetPort: 5432

selector:

app: postgres

role: leader

tier: backend

type: LoadBalancer

externalIPs:

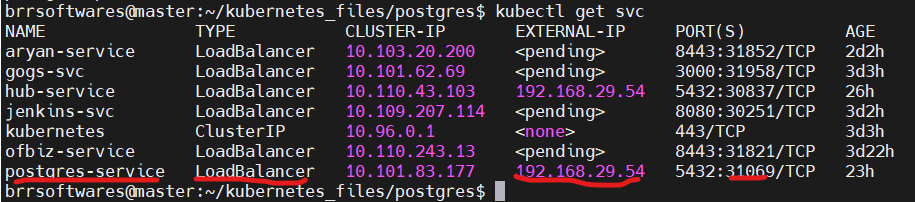
- 192.168.29.54

Save the file and exit. Then the file using the below command:

**kubectl apply -f service.yaml**

After running the above command, you can check it using:

**kubectl get svc**



So. by using this service details, we could be able to connect to our postgres database using DBeaver app.

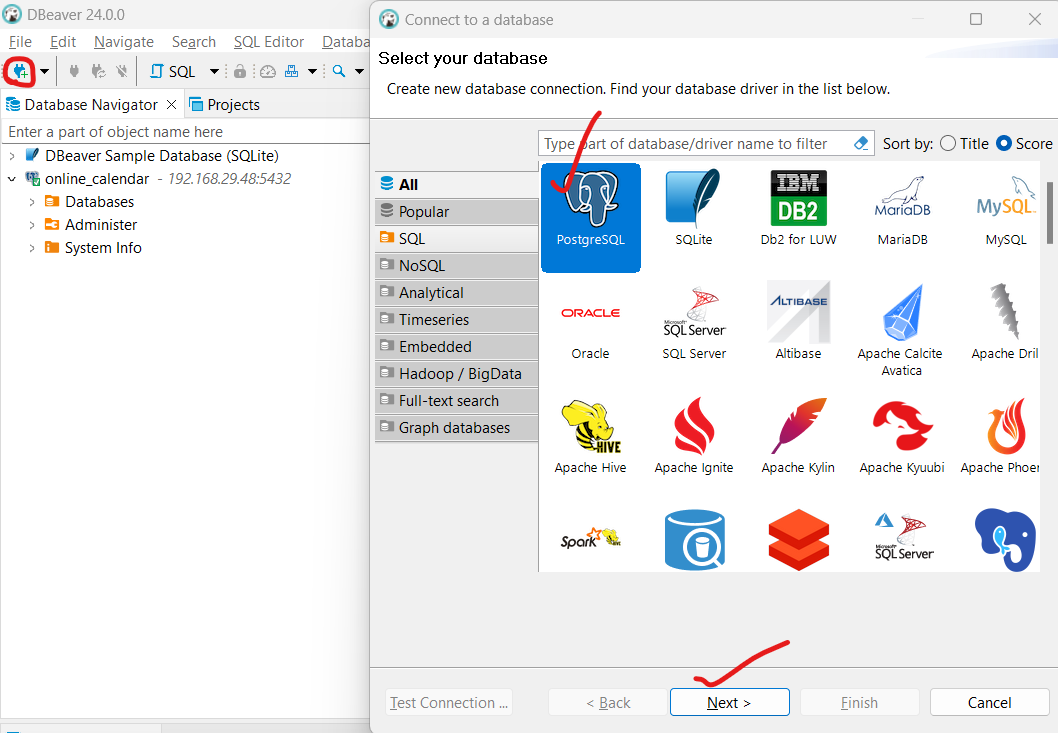
**Using DBeaver to connect remotely with the Postgres K8’s pod:**

Install the DBeaver app from google.com

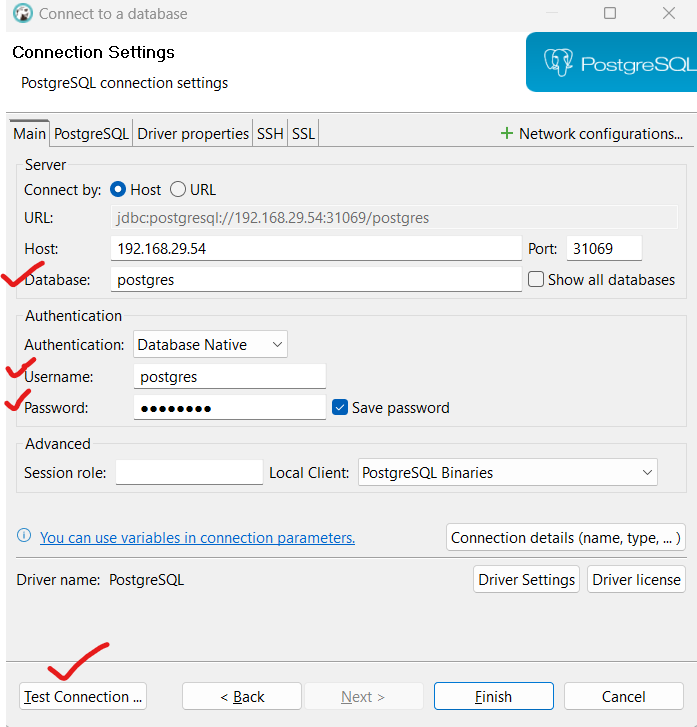
<https://hevodata.com/learn/dbeaver-postgresql/#a6> (follow the instructions to install and setup DBeaver)

After setup,

Click the button below file, so that you can opt to select the databases, then select the postgres database then click on Next, as shown below:

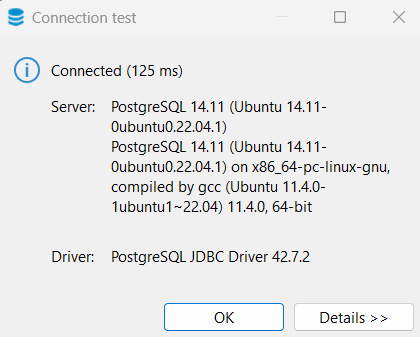


Inside Connection settings, Edit the details as follows:



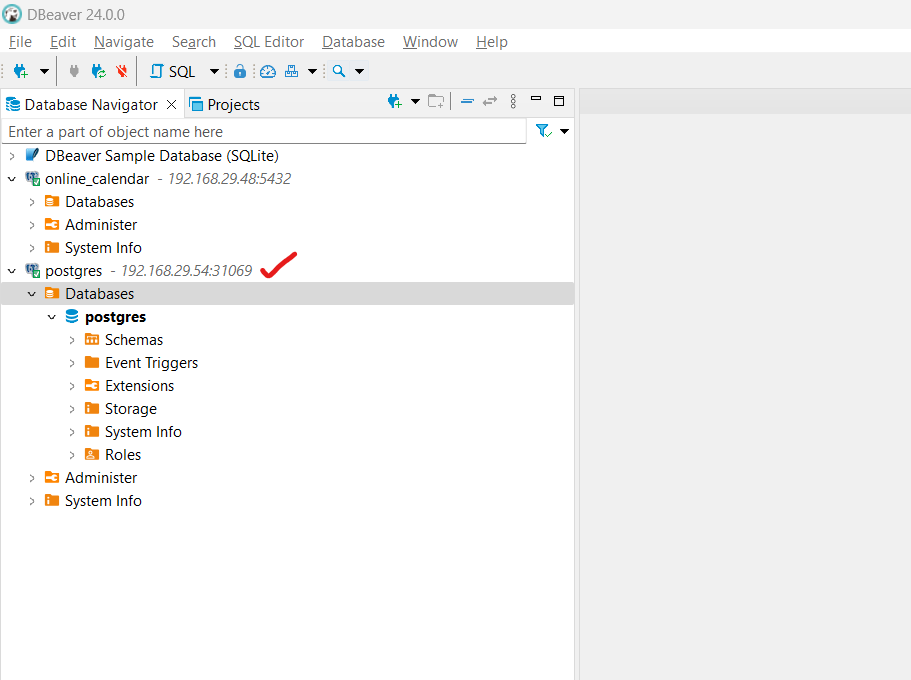
You can enter any database name to which you want to switch and It should be present inside your database.

Password and username will not change for any databases, click on Test connection then you will get an ouput as follows:



Which means, you have successfully connected to your database.

Now, you can proceed and click on finish.



Therefore, successfully we have done it and my pod is running with my configuration file and then finally I’m able to connect to my database using DBeaver.