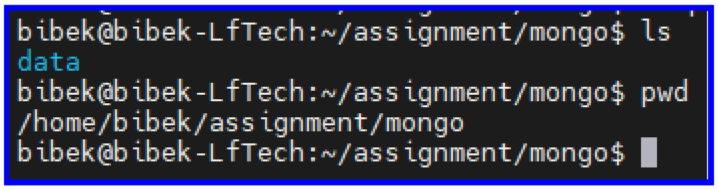
1. Deploy Postgres database using PVC & PV cluster

Making a directory mongo/data in host machine to persist data of postgres

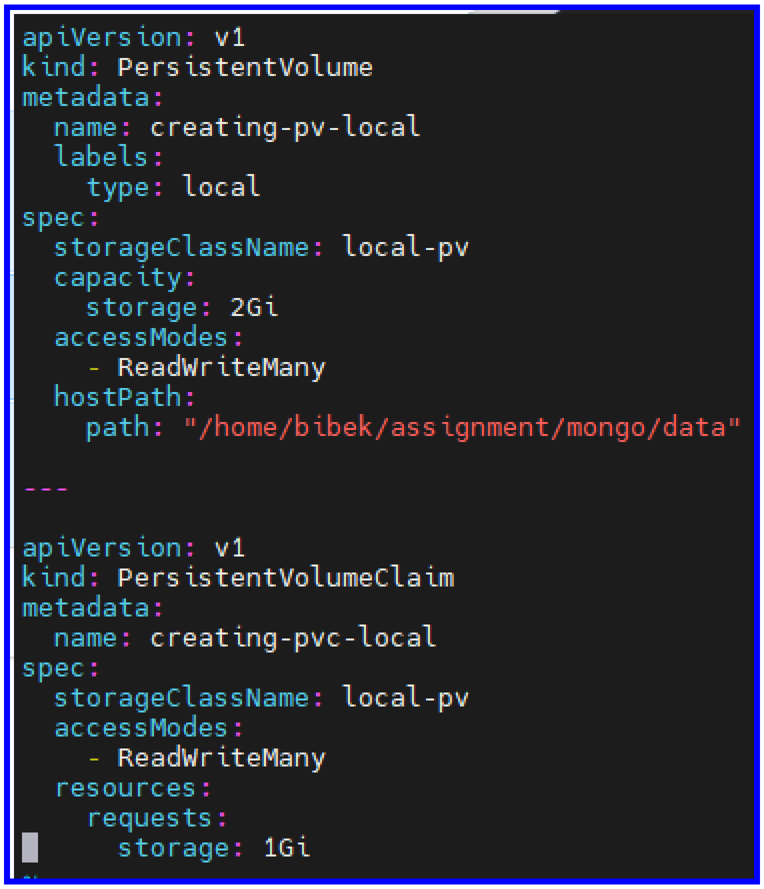
mkdir -p mongo/data



```
bibek@bibek-LfTech:~/assignment/mongo$ ls
data
bibek@bibek-LfTech:~/assignment/mongo$ pwd
/home/bibek/assignment/mongo
bibek@bibek-LfTech:~/assignment/mongo$
```

Creating yaml file for PV and PVC in directory mongo

vi pv-pvc.yaml



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: creating-pv-local
  labels:
    type: local
spec:
  storageClassName: local-pv
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/home/bibek/assignment/mongo/data"

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: creating-pvc-local
spec:
  storageClassName: local-pv
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

Here **storageClassName** of PV and PVC **should be identica**l.

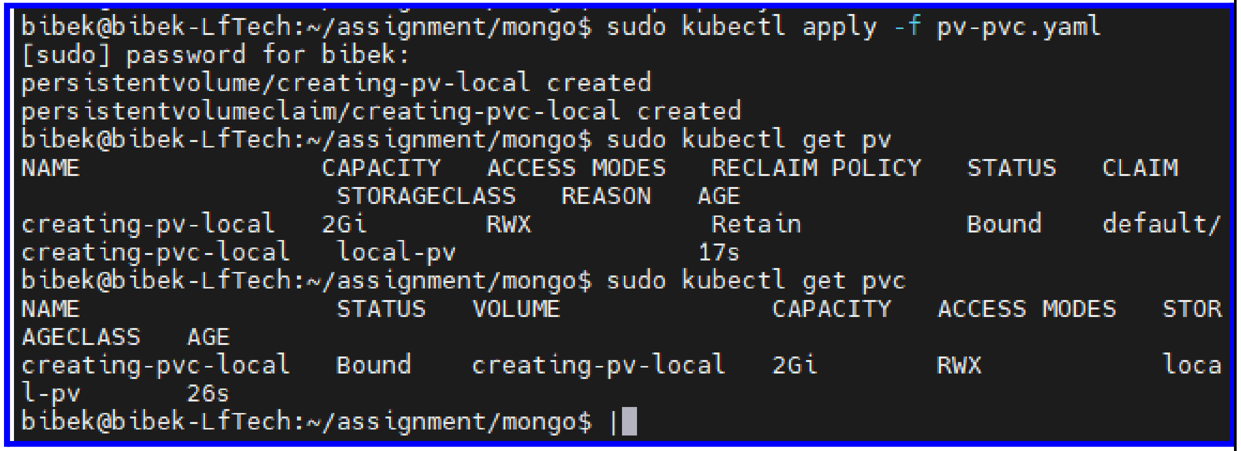
And while PV is created of 2 GB , PVC claims for 1 GB out of 2 GB.

AccessMode represents "ReadWriteMany" which means the volume can be mounted as read-write by many nodes.

We can use other options as well like: **ReadWriteOnce, ReadOnlyMany,** etc

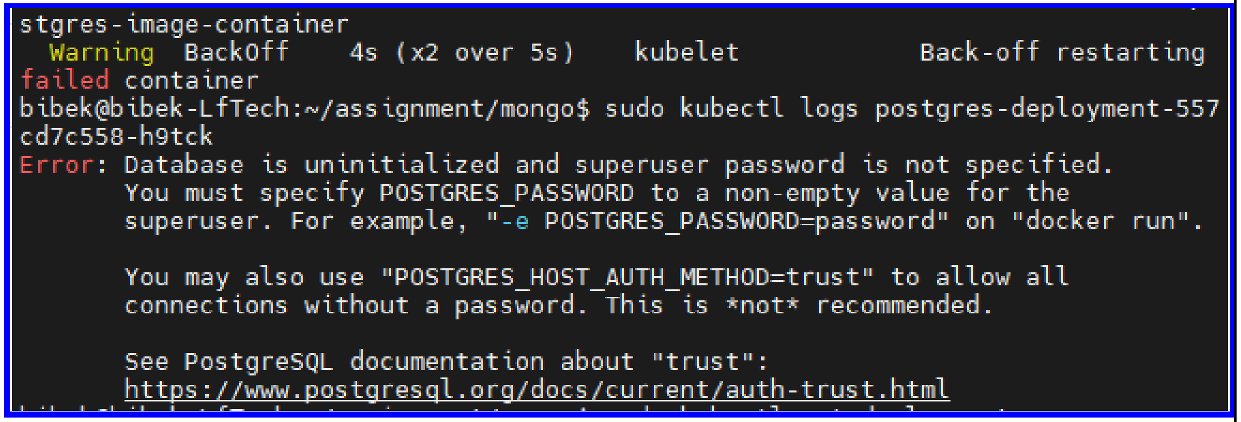
To created pv and pvc applying yaml conf file

sudo kubectl apply -f pv-pvc.yaml



```
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl apply -f pv-pvc.yaml
[sudo] password for bibek:
persistentvolume/creating-pv-local created
persistentvolumeclaim/creating-pvc-local created
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get pv
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
                     STORAGECLASS   REASON   AGE
creating-pv-local   2Gi        RWX            Retain           Bound    default/
creating-pvc-local   local-pv                17s
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get pvc
NAME                 STATUS   VOLUME              CAPACITY   ACCESS MODES   STOR
AGECLASS   AGE
creating-pvc-local   Bound    creating-pv-local   2Gi        RWX            loca
l-pv       26s
bibek@bibek-LfTech:~/assignment/mongo$ |
```

If we run postgres pod without POSTGRES_PASSWORD env we get this error and pod doesn't starts.

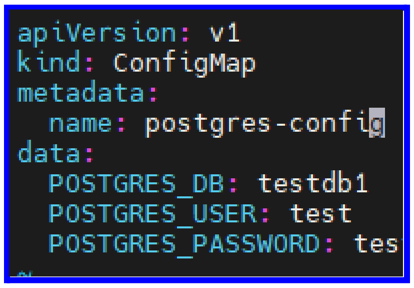


```
stgres-image-container
  Warning  BackOff    4s (x2 over 5s)    kubelet            Back-off restarting
failed container
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl logs postgres-deployment-557
cd7c558-h9tck
Error: Database is uninitialized and superuser password is not specified.
       You must specify POSTGRES_PASSWORD to a non-empty value for the
       superuser. For example, "-e POSTGRES_PASSWORD=password" on "docker run".

       You may also use "POSTGRES_HOST_AUTH_METHOD=trust" to allow all
       connections without a password. This is *not* recommended.

       See PostgreSQL documentation about "trust":
       https://www.postgresql.org/docs/current/auth-trust.html
```

Writing env variables in plain text inside deployment for pod is not a best practice

Creating configmap to pass environment variables.

vi configmap.yaml



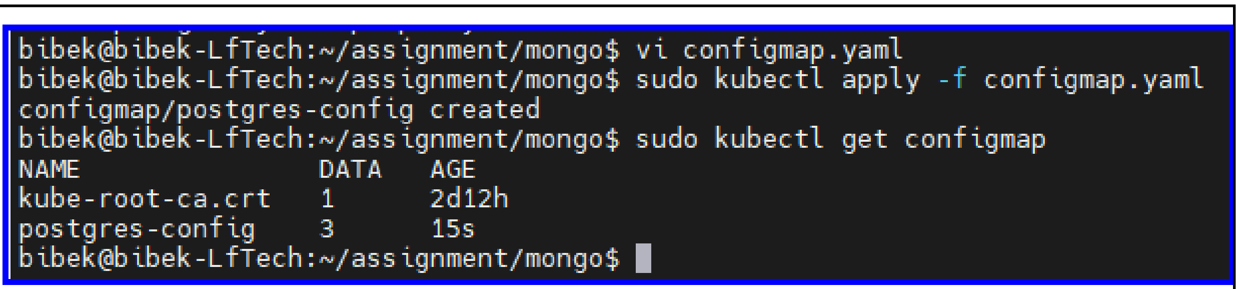
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-config
data:
  POSTGRES_DB: testdb1
  POSTGRES_USER: test
  POSTGRES_PASSWORD: tes
```

I have not shown the password.

Using this configmap we can pass the environment value to postgres-pod

We created config map with kubectl apply command

sudo kubectl apply -f configmap.yaml

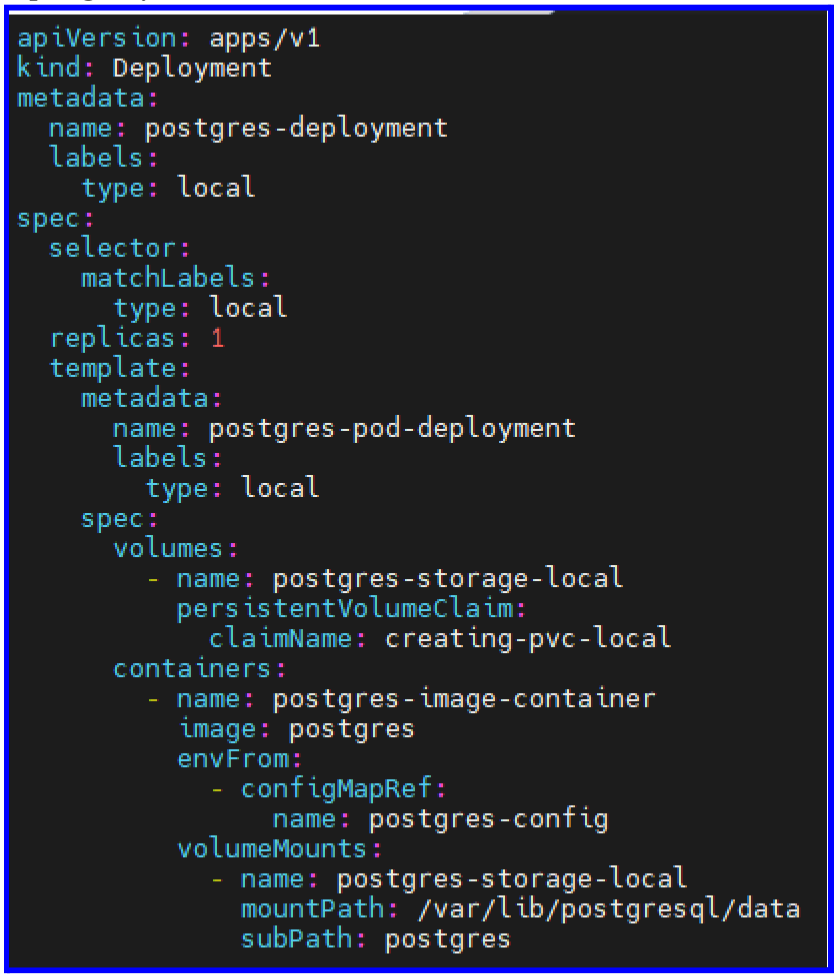


```
bibek@bibek-LfTech:~/assignment/mongo$ vi configmap.yaml
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl apply -f configmap.yaml
configmap/postgres-config created
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      2d12h
postgres-config    3      15s
bibek@bibek-LfTech:~/assignment/mongo$
```

We can see that the configmap is created and running.

<u>Creating yaml file of deployment for postgres-pod</u>

vi postgres.yaml



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
  labels:
    type: local
spec:
  selector:
    matchLabels:
      type: local
  replicas: 1
  template:
    metadata:
      name: postgres-pod-deployment
      labels:
        type: local
    spec:
      volumes:
        - name: postgres-storage-local
          persistentVolumeClaim:
            claimName: creating-pvc-local
      containers:
        - name: postgres-image-container
          image: postgres
          envFrom:
            - configMapRef:
                name: postgres-config
          volumeMounts:
            - name: postgres-storage-local
              mountPath: /var/lib/postgresql/data
              subPath: postgres
```

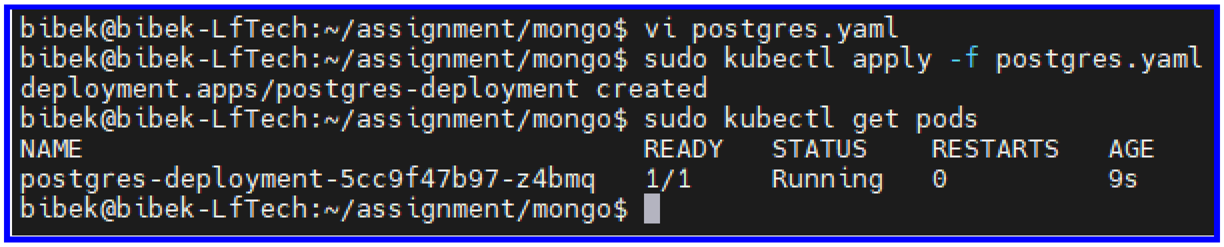
Here we illustrated the name of PVC (**claimName**) in volume section

We passed environment variables using **configMapRef** using **envFrom.**

We mount the postgres pod **/var/lib/postgresql/data** directory with our localhost for volume persisting.

<u>Now creating postgres pod using kubectl apply</u>

sudo kubectl apply -f postgres.yaml

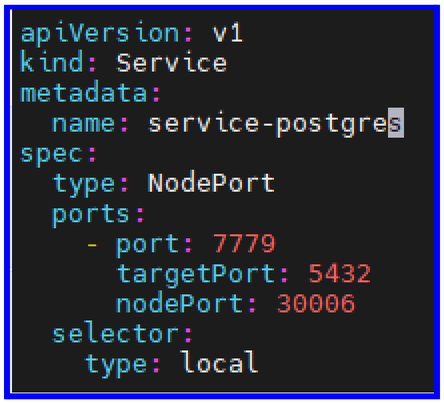


```
bibek@bibek-LfTech:~/assignment/mongo$ vi postgres.yaml
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl apply -f postgres.yaml
deployment.apps/postgres-deployment created
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
postgres-deployment-5cc9f47b97-z4bmq   1/1     Running   0          9s
bibek@bibek-LfTech:~/assignment/mongo$
```

We can see that the postgrespod is created and running.

Now creating service for the postgres-pod

vi service.yaml

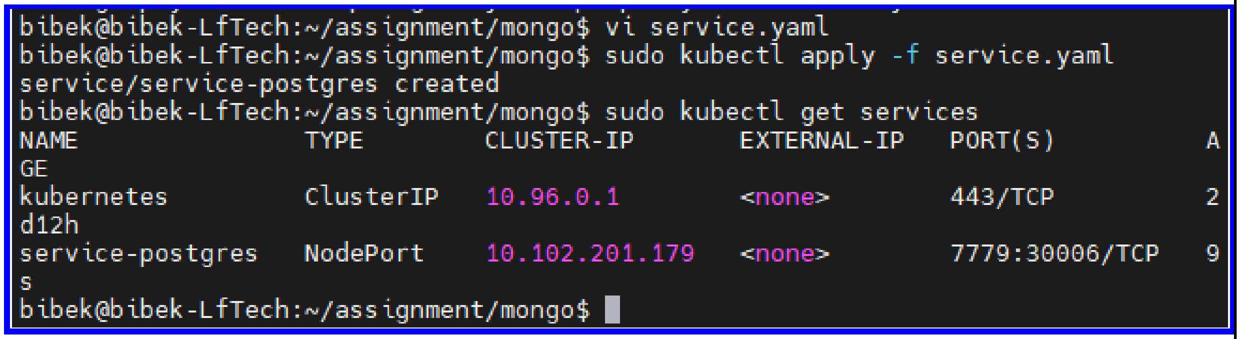


```
apiVersion: v1
kind: Service
metadata:
  name: service-postgres
spec:
  type: NodePort
  ports:
    - port: 7779
      targetPort: 5432
      nodePort: 30006
  selector:
    type: local
```

using NodePort type and assigning static nodeport 30006

Creating service using kubectl command

sudo kubectl apply -f service.yaml



```
bibek@bibek-LfTech:~/assignment/mongo$ vi service.yaml
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl apply -f service.yaml
service/service-postgres created
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get services
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          A
GE
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP          2
d12h
service-postgres   NodePort    10.102.201.179   <none>        7779:30006/TCP   9
s
bibek@bibek-LfTech:~/assignment/mongo$
```

We can see the service is created and running with nodeport 30006.

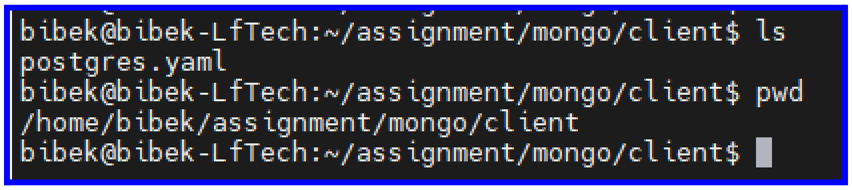
2. Deploy Postgres Client in cluster(psql)

To deploy postgres-client we need postgres image

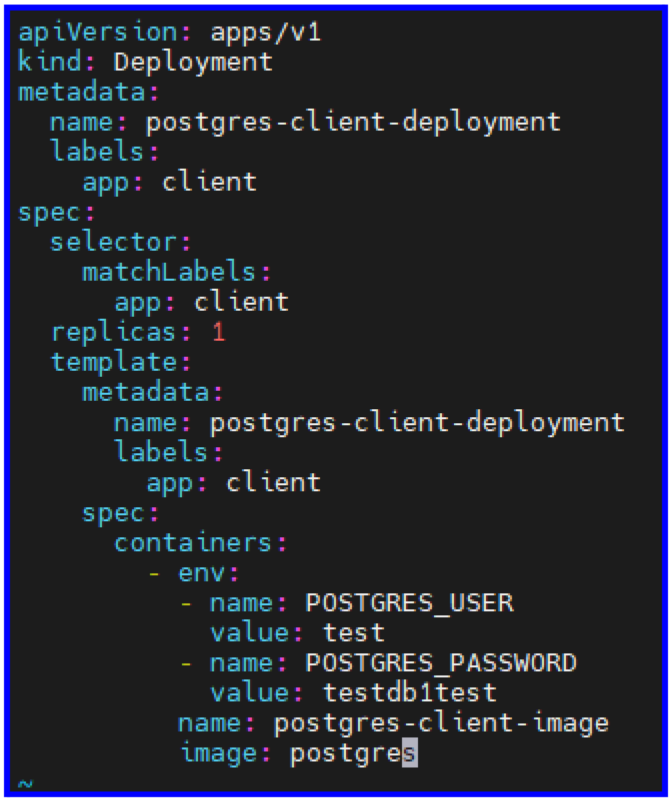
Using psql -h, we can connect to the postgres database pod we have created earlier.

Simply creating a **postgres YAML** with user and database **in client directory**

vi postgres.yaml



```
bibek@bibek-LfTech:~/assignment/mongo/client$ ls
postgres.yaml
bibek@bibek-LfTech:~/assignment/mongo/client$ pwd
/home/bibek/assignment/mongo/client
bibek@bibek-LfTech:~/assignment/mongo/client$
```



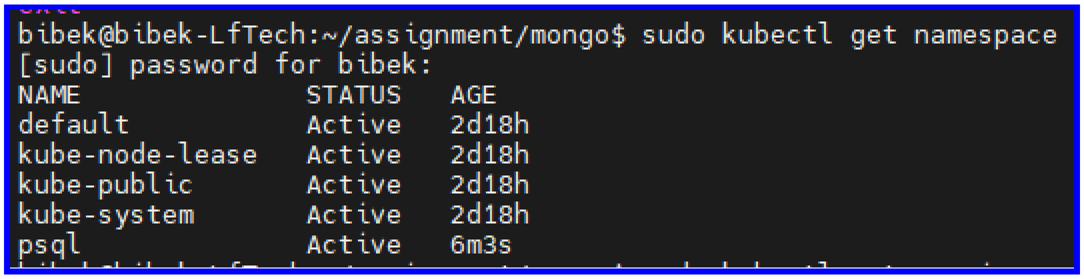
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-client-deployment
  labels:
    app: client
spec:
  selector:
    matchLabels:
      app: client
  replicas: 1
  template:
    metadata:
      name: postgres-client-deployment
      labels:
        app: client
    spec:
      containers:
        - env:
          - name: POSTGRES_USER
            value: test
          - name: POSTGRES_PASSWORD
            value: testdb1test
          name: postgres-client-image
          image: postgres
~
```

Creating namespace psql

sudo kubectl create namespace psql

We can view the created namespace using

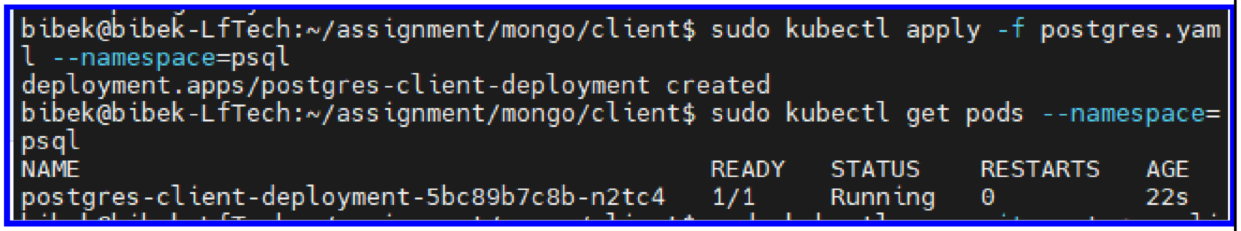
sudo kubectl get namespace



```
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get namespace
[sudo] password for bibek:
NAME              STATUS   AGE
default           Active   2d18h
kube-node-lease   Active   2d18h
kube-public       Active   2d18h
kube-system       Active   2d18h
psql              Active   6m3s
```

<u>Creating **postgres-pod in cluster PSQL** using **kubectl apply** command</u>

sudo kubectl apply -f postgres.yaml --namespace=psql



```
bibek@bibek-LfTech:~/assignment/mongo/client$ sudo kubectl apply -f postgres.yam
l --namespace=psql
deployment.apps/postgres-client-deployment created
bibek@bibek-LfTech:~/assignment/mongo/client$ sudo kubectl get pods --namespace=
psql
NAME                                          READY   STATUS    RESTARTS   AGE
postgres-client-deployment-5bc89b7c8b-n2tc4   1/1     Running   0          22s
```

Here we can see client postgres is created and running in namespace psql.

3. Connect Postgres database from Postgres Client using core-dns's host name.

Going **inside the client** postgres pod (psql-NameSpace) with command **kubectl exec**

sudo kubectl exec -it postgres-client-deployment-5bc89b7c8b-n2tc4 bash --namespace=psql

Changing the user to postgres

su postgres

And executing psql with -h command to connect to service.postgres host

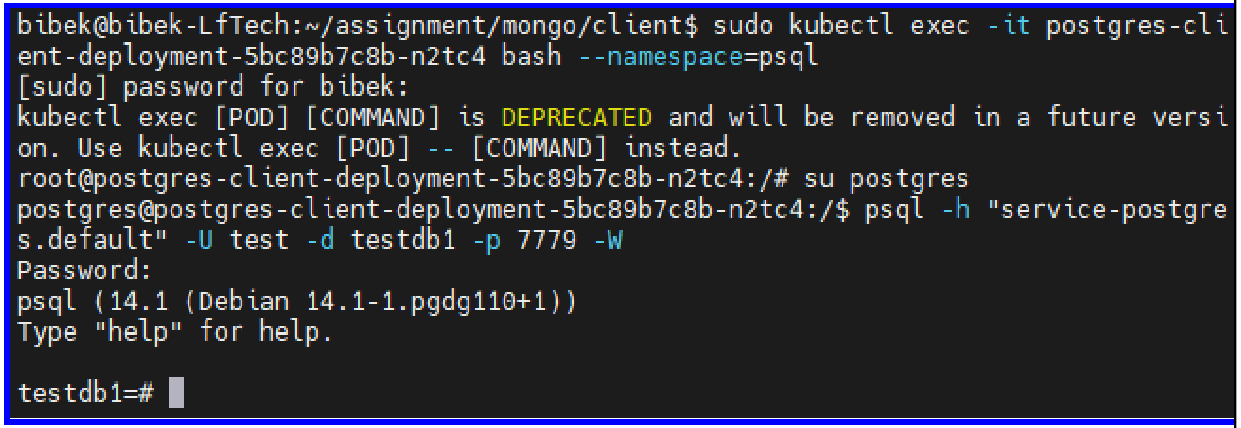
psql -h "service-postgres.default" -U test -d testdb1 -p 7779 -W

Generally hostname is a combination of <**servicename.namespace**>.

Here service is in default namespace that's why host name will be **service-postgres.default**

*And **core DNS** of minikube will resolve its address*

*Here **host is*** **service-postgres.default** *,* *user is* **test,** *DB is* **testdb1** *and connecting to port* **7779** *at service port with **password(-W).***

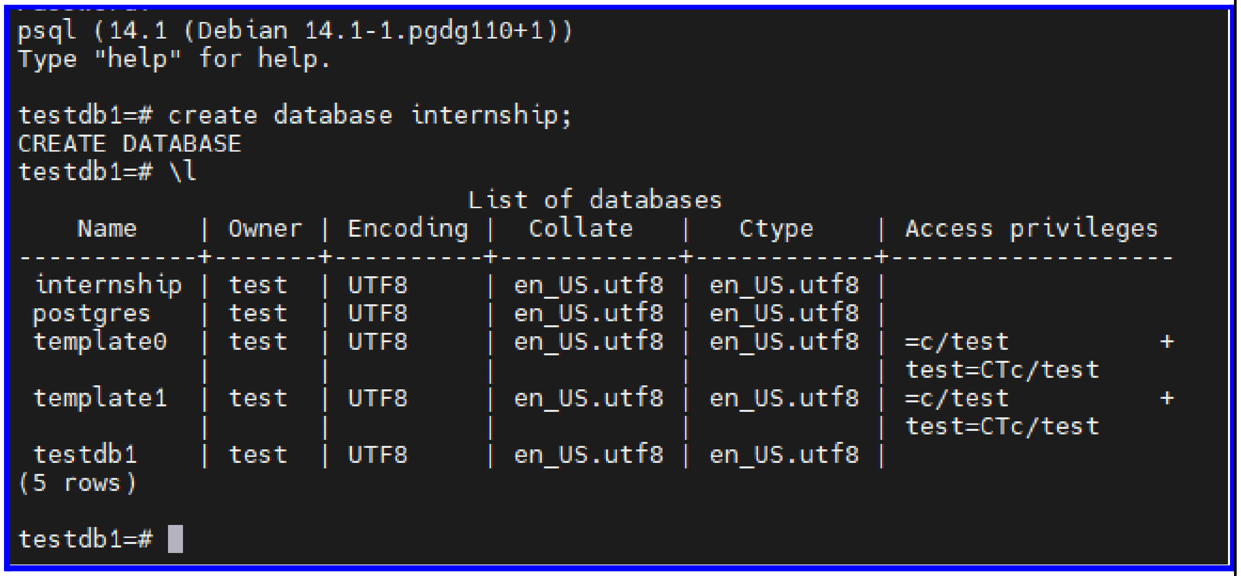


```
bibek@bibek-LfTech:~/assignment/mongo/client$ sudo kubectl exec -it postgres-cli
ent-deployment-5bc89b7c8b-n2tc4 bash --namespace=psql
[sudo] password for bibek:
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future versi
on. Use kubectl exec [POD] -- [COMMAND] instead.
root@postgres-client-deployment-5bc89b7c8b-n2tc4:/# su postgres
postgres@postgres-client-deployment-5bc89b7c8b-n2tc4:/$ psql -h "service-postgre
s.default" -U test -d testdb1 -p 7779 -W
Password:
psql (14.1 (Debian 14.1-1.pgdg110+1))
Type "help" for help.

testdb1=#
```

4. Create a database(internship) and a few tables in the database.

Creating database "***internship***"through **client connectoin to Original postgresDB**

create database internship;



```
psql (14.1 (Debian 14.1-1.pgdg110+1))
Type "help" for help.

testdb1=# create database internship;
CREATE DATABASE
testdb1=# \l
                              List of databases
    Name    | Owner | Encoding |  Collate   |   Ctype    | Access privileges
------------+-------+----------+------------+------------+-------------------
 internship | test  | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres   | test  | UTF8     | en_US.utf8 | en_US.utf8 |
 template0  | test  | UTF8     | en_US.utf8 | en_US.utf8 | =c/test          +
            |       |          |            |            | test=CTc/test
 template1  | test  | UTF8     | en_US.utf8 | en_US.utf8 | =c/test          +
            |       |          |            |            | test=CTc/test
 testdb1    | test  | UTF8     | en_US.utf8 | en_US.utf8 |
(5 rows)

testdb1=#
```

To verify client make changes in Older postgresDB

Getting inside **older postgresDB in default Namespace**

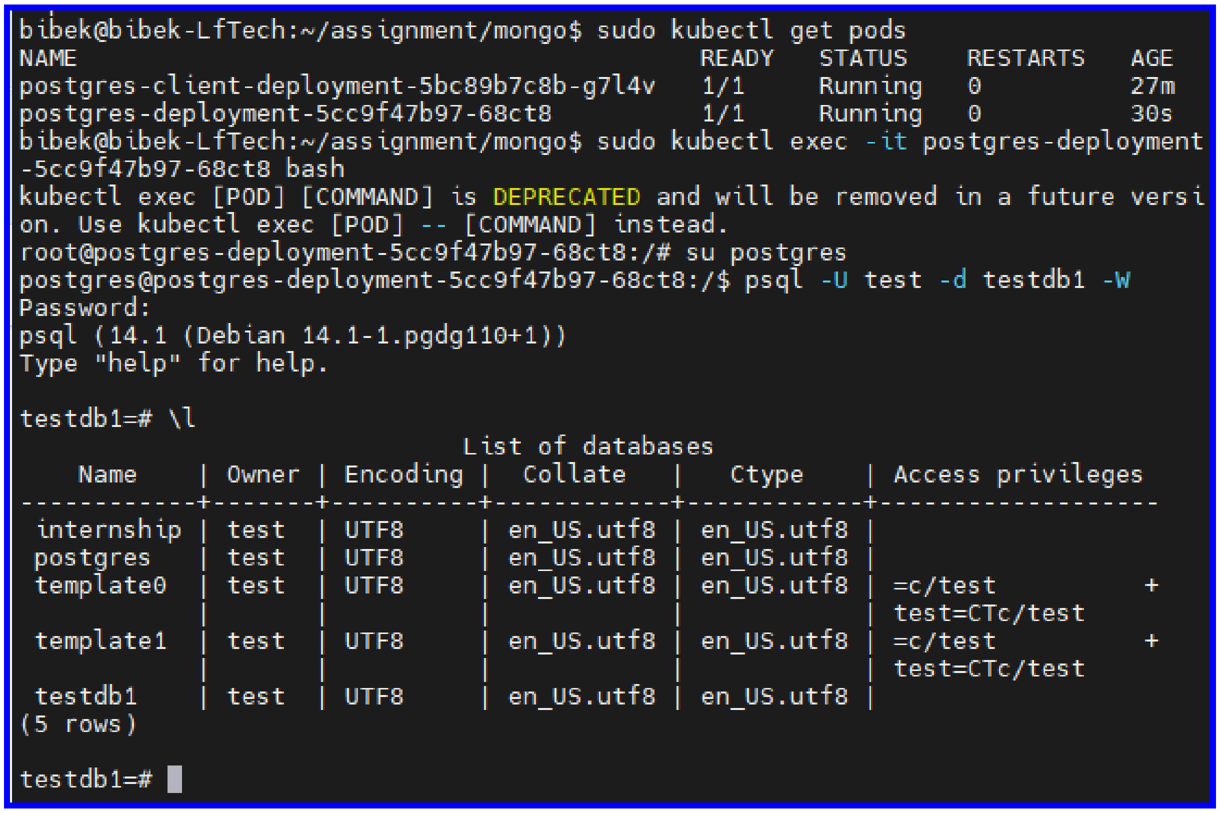
sudo kubectl exec -it postgres-deployment-5cc9f47b97-68ct8 bash

And checking the internship database has been created or not

su postgres

psql -U test -d testdb1 -W

And view all databases



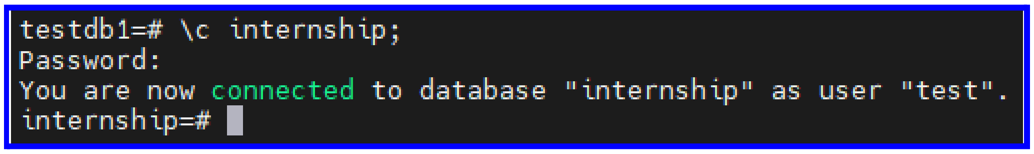
```
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl get pods
NAME                                         READY   STATUS    RESTARTS   AGE
postgres-client-deployment-5bc89b7c8b-g7l4v  1/1     Running   0          27m
postgres-deployment-5cc9f47b97-68ct8         1/1     Running   0          30s
bibek@bibek-LfTech:~/assignment/mongo$ sudo kubectl exec -it postgres-deployment
-5cc9f47b97-68ct8 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future versi
on. Use kubectl exec [POD] -- [COMMAND] instead.
root@postgres-deployment-5cc9f47b97-68ct8:/# su postgres
postgres@postgres-deployment-5cc9f47b97-68ct8:/$ psql -U test -d testdb1 -W
Password:
psql (14.1 (Debian 14.1-1.pgdg110+1))
Type "help" for help.

testdb1=# \l
                              List of databases
    Name    | Owner | Encoding |  Collate   |   Ctype    | Access privileges
------------+-------+----------+------------+------------+--------------------
 internship | test  | UTF8     | en_US.utf8 | en_US.utf8 |
 postgres   | test  | UTF8     | en_US.utf8 | en_US.utf8 |
 template0  | test  | UTF8     | en_US.utf8 | en_US.utf8 | =c/test          +
            |       |          |            |            | test=CTc/test
 template1  | test  | UTF8     | en_US.utf8 | en_US.utf8 | =c/test          +
            |       |          |            |            | test=CTc/test
 testdb1    | test  | UTF8     | en_US.utf8 | en_US.utf8 |
(5 rows)

testdb1=#
```

Here we can see that **internship database** *<u>created through client connection</u>* ***can be seen in original database(*****Older PostgresDB*****)***

<u>Creating tables through client(PSQL-NameSpace) connection to Older PostgresDB</u>

<u>Connecting to database **Internship**</u>

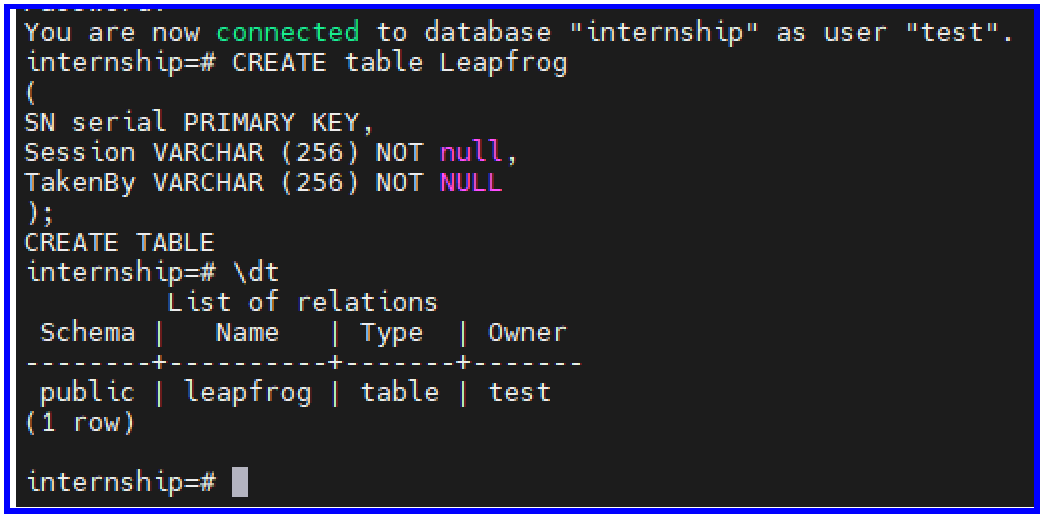
\c internship;



```
testdb1=# \c internship;
Password:
You are now connected to database "internship" as user "test".
internship=#
```

<u>Creating table “**Leapfrog**”</u>

CREATE table Leapfrog
(
SN serial PRIMARY KEY,
Session VARCHAR (256) NOT null,
TakenBy VARCHAR (256) NOT NULL
);



```
You are now connected to database "internship" as user "test".
internship=# CREATE table Leapfrog
(
SN serial PRIMARY KEY,
Session VARCHAR (256) NOT null,
TakenBy VARCHAR (256) NOT NULL
);
CREATE TABLE
internship=# \dt
         List of relations
 Schema |   Name   | Type  | Owner
--------+----------+-------+-------
 public | leapfrog | table | test
(1 row)

internship=# 
```

Name of table *is "***Leapfrog***"*

Three columns are added - **SN (***serial type***), Session (***VARCHAR type***)** *and* **TakenBy (***VARCHAR***).**

Serial type *will assign numeric value itself starting from 1*

VARCHAR type *is used to give character as value* ***i.e. string***

NOT NULL *means the input should not be empty while inserting data in to the table*

PRIMARY KEY *defines the unique ID for the data in that column to be identified while querying.*

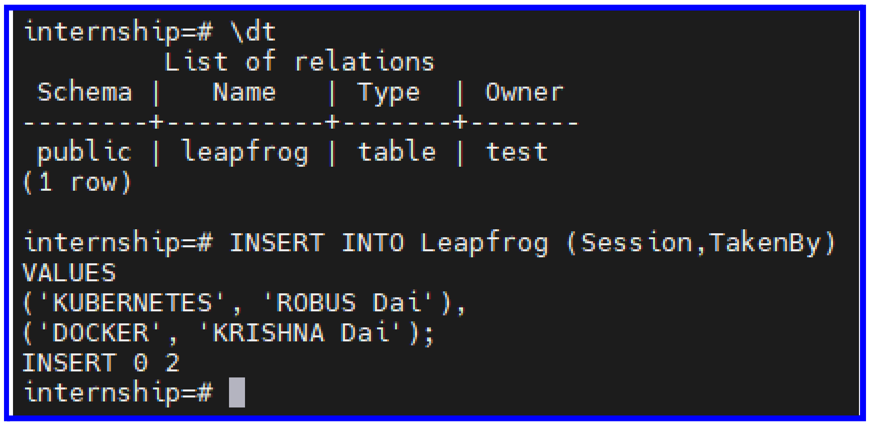
Inserting value into the table Leapfrog

INSERT INTO Leapfrog (Session,TakenBy)

VALUES

('KUBERNETES', 'ROBUS Dai'),

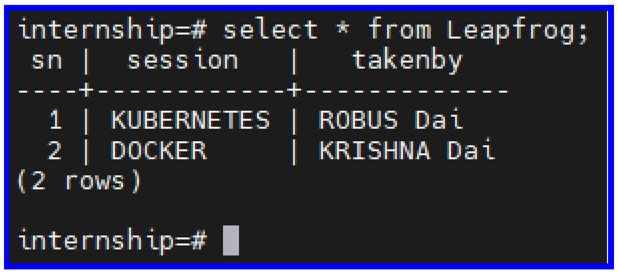
('DOCKER', 'KRISHNA Dai');



```
internship=# \dt
         List of relations
 Schema |   Name   | Type  | Owner
--------+----------+-------+-------
 public | leapfrog | table | test
(1 row)

internship=# INSERT INTO Leapfrog (Session,TakenBy)
VALUES
('KUBERNETES', 'ROBUS Dai'),
('DOCKER', 'KRISHNA Dai');
INSERT 0 2
internship=# 
```

To view data from table Leapfrog

select * from Leapfrog;



```
internship=# select * from Leapfrog;
 sn |   session   |   takenby
----+-------------+-------------
  1 | KUBERNETES  | ROBUS Dai
  2 | DOCKER      | KRISHNA Dai
(2 rows)

internship=#
```

To delete table

drop table Leapfrog;

To delete database

drop database internship;

In this way in Kubernetes(Minikube) Cluster

- Postgres Pod was created using PV and PVC for persisting the data of DB
- Postgres-Client pod was created in new NameSpace
- Made connection to the database using client pod to Postgres-Pod (via CoreDNS hostname)
- And few practices were done (creating database, creating table, inserting values, deleting table, deleting database, etc)

Thank you !!