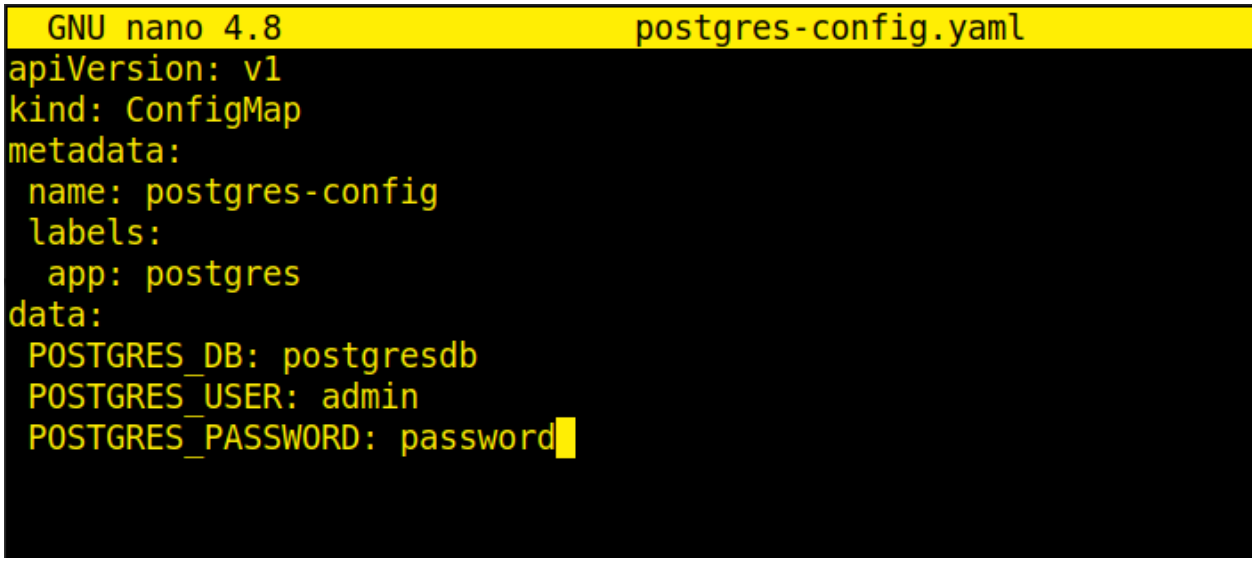
Deploy Postgres database using PVC & PV cluster

The configmap for postgres is created **postgresconfig.yaml as below,**

$ sudo nano postgres-config.yaml

Include the below in the file,

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: postgres-config
 labels:
  app: postgres
data:
 POSTGRES_DB: postgresdb
 POSTGRES_USER: admin
 POSTGRES_PASSWORD: password
```



And also for deployment, **postgres-deployment.yaml as** :

$ sudo nano postgres-deployment.yaml

Include the below in the file,

kind: Deployment

apiVersion: apps/v1

metadata:

name: postgres

spec:

replicas: 1

selector:

matchLabels:

```
  app: postgres
template:
 metadata:
  labels:
       app: postgres
 spec:
       containers:
       - name: postgres
       image: postgres:10.1
       imagePullPolicy: "Always"
       ports:
       - containerPort: 5432
       envFrom:
       - configMapRef:
       name: postgres-config
       volumeMounts:
       - mountPath: /var/lib/postgresql/data
       name: postgresdb
```

volumes:

- name: postgresdb

persistentVolumeClaim:

claimName: postgres-volume-claim

```
  GNU nano 4.8                  postgres-deployment.yaml
kind: Deployment
apiVersion: apps/v1
metadata:
 name: postgres
spec:
 replicas: 1
 selector:
  matchLabels:
   app: postgres
 template:
  metadata:
   labels:
     app: postgres
  spec:
    containers:
     - name: postgres
       image: postgres:10.1
       imagePullPolicy: "Always"
       ports:
        - containerPort: 5432
       envFrom:
        - configMapRef:
           name: postgres-config
       volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgresdb

   volumes:
     - name: postgresdb
       persistentVolumeClaim:
        claimName: postgres-volume-claim
```

Now for Persistent Volume and Persistent Volume Claim (PV and PVC cluster), we create **postgres-volume.yaml** as,

$ sudo nano postgres-volume.yaml

Include the below in the file,

```
kind: PersistentVolume
apiVersion: v1
metadata:
 name: postgrespv-volume
 labels:
  type: local
  app: postgres
spec:
 storageClassName: manual
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 hostPath:
```

```
  path: "/mount/path/data"


---


kind: PersistentVolumeClaim

apiVersion: v1

metadata:

 name: postgres-volume-claim

 labels:

  app: postgres

spec:

 storageClassName: manual

 accessModes:

  - ReadWriteMany

 resources:

  requests:

   storage: 5Gi
```

```
  GNU nano 4.8                    postgres-volume.yaml
kind: PersistentVolume
apiVersion: v1
metadata:
 name: postgrespv-volume
 labels:
  type: local
  app: postgres
spec:
 storageClassName: manual
 capacity:
  storage: 5Gi
 accessModes:
  - ReadWriteMany
 hostPath:
  path: "/mount/path/data"

---

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
 name: postgres-volume-claim
 labels:
  app: postgres
spec:
 storageClassName: manual
 accessModes:
  - ReadWriteMany
 resources:
  requests:
   storage: 5Gi
```

To apply those configurations, following commands are used:

$ kubectl apply -f postgres-config.yaml

$ kubectl apply -f postgres-deployment.yaml

$ kubectl apply -f postgres-volume.yaml

```
lostinserver@lostinserver:~/Documents/K8s/postgres$ kubectl apply -f postgres-co
nfig.yaml
configmap/postgres-config unchanged
lostinserver@lostinserver:~/Documents/K8s/postgres$ kubectl apply -f postgres-de
ployment.yaml
deployment.apps/postgres unchanged
lostinserver@lostinserver:~/Documents/K8s/postgres$ kubectl apply -f postgres-vo
lume.yaml
persistentvolume/postgrespv-volume unchanged
persistentvolumeclaim/postgres-volume-claim unchanged
lostinserver@lostinserver:~/Documents/K8s/postgres$
```

Now, get deployed pods and services with,

$ kubectl get all

```
lostinserver@lostinserver:~/Documents/K8s/postgres$ kubectl get all
NAME                                     READY   STATUS    RESTARTS       AGE
pod/mongodb-standalone-8454974f58-bpl96  1/1     Running   0              72m
pod/nodejsdeploy-9fc8fbd55-7smk2         1/1     Running   1 (107m ago)   2d3h
pod/nodejsdeploy-9fc8fbd55-z47hh         1/1     Running   1 (107m ago)   2d3h
pod/postgres-684c689b4b-x8zpd            1/1     Running   0              4m8s
pod/webserver                            1/1     Running   3 (107m ago)   4d20h

NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
     AGE
service/database     ClusterIP      None            <none>        <none>
     85m
service/kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP
     4d20h
service/nodejsdeploy LoadBalancer   10.107.224.94   <pending>     6080:31538/T
CP   2d3h

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mongodb-standalone    1/1     1            1           78m
deployment.apps/nodejsdeploy          2/2     2            2           4d5h
deployment.apps/postgres              1/1     1            1           4m8s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/mongodb-standalone-8454974f58  1         1         1       72m
replicaset.apps/mongodb-standalone-84695c67f4  0         0         0       78m
replicaset.apps/nodejsdeploy-5b66488cc8        0         0         0       2d3h
replicaset.apps/nodejsdeploy-775b5c64b9        0         0         0       2d3h
replicaset.apps/nodejsdeploy-84fff4767d        0         0         0       4d5h
replicaset.apps/nodejsdeploy-9fc8fbd55         2         2         2       2d3h
replicaset.apps/postgres-684c689b4b            1         1         1       4m8s
lostinserver@lostinserver:~/Documents/K8s/postgres$
```

Now enter the CLI of postgres database with,

$ kubectl exec -it postgres-5d9c946c6f-9mpdx -- psql -h localhost -U admin --password -p 5432:30733 postgresdb

here, **postgres-5d9c946c6f-9mpdx** *is the name of the pod*