

Task 6.3D: PostgreSQL Replication in Kubernetes

I. Database Selection and Replication Features

For this project, I selected PostgreSQL as the database platform due to its mature, robust, and widely adopted replication features. PostgreSQL supports several replication methods, but I focused on “Streaming Replication”, which is a physical, WAL-based (Write-Ahead Log) approach. This method enables near real-time synchronization of data from a primary (read/write) node to one or more replicas (read-only nodes).

Key Replication Features:

- **Asynchronous and Synchronous Replication:** By default, streaming replication is asynchronous, meaning the primary does not wait for replicas to confirm receipt of WAL data. However, synchronous mode can be enabled for stronger consistency, at the cost of some performance.
- **Multiple Replicas:** PostgreSQL allows multiple standby nodes, supporting both high availability and horizontal read scaling.
- **Hot Standby:** Replicas can serve read-only queries while replaying WAL data, improving resource utilization.
- **Point-in-Time Recovery (PITR):** WAL archiving and base backups allow recovery to any point in time, which is essential for disaster recovery.

I maintained my choice of PostgreSQL from Task 6.2C, as its open-source nature, strong community support, and clear documentation made it ideal for learning and demonstration purposes.

II. Deployment Steps and Challenges

The deployment was performed on an Ubuntu VM running in Oracle VirtualBox, with Docker and Minikube providing the Kubernetes environment.

The process was divided into several phases:

Phase 1 – Kubernetes Resources Creation

A/ Secrets Management:

- I used Kubernetes Secrets to securely store sensitive credentials like:
 - POSTGRES_USER
 - POSTGRES_PASSWORD
 - REPLICATION_USER

- REPLICATION_PASSWORD

B/ Persistent Storage:

- PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs) were defined to ensure that PostgreSQL data persisted across pod restarts and rescheduling.
- I initially attempted to use a static hostPath PV, but later switched to dynamic provisioning for better compatibility with StatefulSets.

C/ StatefulSet Configuration:

- A StatefulSet was used to deploy two PostgreSQL pods (“postgres-0” as primary, “postgres-1” as replica).
- StatefulSets provide stable network identities and persistent storage for each pod, which is crucial for database workloads.

Phase 2 – Primary Pod Configuration

A/ Replication Setup

A dedicated replication user was created with this SQL command - “CREATE USER replicator WITH REPLICATION ENCRYPTED PASSWORD 'replicatorpass';” through accessing the primary pod, “postgres-0”.

- This user was granted only the necessary privileges for replication, following the principle of least privilege.

B/ “pg_hba.conf” Adjustments:

In the same pod, I edited its “pg_hba.conf” to allow replication connections between the primary and replica pod through the setting up a common subnet that housing these pods, by adding the following line - “host replication replicator 10.1.0.0/16 md5”

Phase 3 – Replica Pod Setup

To automate the replica's setup, I added an init container to the StatefulSet.

- This container checks if the pod is “postgres-1” and, if so, uses “pg_basebackup” to clone the primary's data directory, creates the “standby.signal” file, and configures “primary_conninfo” for streaming replication.

Errors and Troubleshooting

A/ PersistentVolumeClaim (PVC) Binding Issue

- Upon initial deployment, the pods were stuck in a “Pending” state due to unbound PVCs.
- Technical Detail: StatefulSets generate unique PVCs for each pod (e.g., “postgres-data-postgres-0”), but my static PV did not match these names.
- Resolution: I removed the manual PV/PVC and allowed Kubernetes to dynamically provision storage, which resolved the binding issue.

B/ “pg_ctl” Command Not Found

- When attempting to stop PostgreSQL for maintenance, the “pg_ctl” command was not found in the default PATH.
- Technical Detail: The official PostgreSQL container does not include “/usr/lib/postgresql/14/bin” in the PATH for the “postgres” user.
- Resolution: I used command “find / -name pg_ctl” to locate the binary and invoked it with the full path.

C/ “pg_basebackup” Fails Due to Non-Empty Data Directory

- “pg_basebackup” refused to run because the target data directory was not empty, because data of pod postgres-1 generated almost immediately after using command “rm -rf /var/lib/postgresql/data/*” which cause the termination of logging into the postgres-1 pod and preventing the “pg_basebackup” command from running.
- Resolution: I re-edited the “postgres-statefulset.yaml” to include initContainer upon its creation.

D/ Replication Connection Refused

- The replica could not connect to the primary, with errors such as “no pg_hba.conf entry” or “no encryption.”
- Technical Detail: The default “pg_hba.conf” did not allow replication connections from the replica's IP.
- Resolution: I edited “pg_hba.conf” to permit connections from the correct subnet and reloaded the configuration using “pg_ctl”.

III. Replication Testing and Verification

To confirm that replication was functioning as intended, I performed the following tests:

A/ Table Creation and Data Insertion:

On the primary pod, I created a test table and inserted sample data through these SQL commands:

```
“CREATE TABLE replication_test (id SERIAL PRIMARY KEY, message TEXT);”
```

```
“INSERT INTO replication_test (message) VALUES ('Hello from primary');”
```

B/ Replica Data Verification:

On the replica pod, I queried the same table to ensure the data appeared, confirming that changes on the primary were replicated in near real-time.

C/ Recovery Mode Check:

I ran command “SELECT pg_is_in_recovery();” on the replica, which returned “true”, indicating it was operating as a hot standby.

D/ Log and Status Monitoring:

I used “kubectl logs” on postgres-1 to check if the pg_basebackup is loaded with it and PostgreSQL's “pg_stat_replication” on postgres-0 to view for any replication connection, its current status, check for lag, and ensure there were no errors or delays.

IV. Learning Outcomes and Reflections

Completing this activity was both challenging and rewarding, providing deep insight into the practicalities of deploying a replicated PostgreSQL database in Kubernetes.

Not everything went as expected—several technical hurdles arose, such as PersistentVolumeClaim binding issues, path and permissions problems, and the intricacies of PostgreSQL's replication configuration. However, overcoming these obstacles significantly improved my troubleshooting and problem-solving skills.

Deploying database replication in Kubernetes offers clear advantages: high availability, automated recovery, and the ability to scale read workloads easily by adding replicas. However, it also introduces complexity, especially in storage management and the need for precise configuration across multiple layers (Kubernetes and PostgreSQL).

Adapting traditional database replication concepts to the Kubernetes environment was initially difficult, particularly understanding how StatefulSets, PersistentVolumes, and Secrets interact to provide stable storage and identity for each pod.

The key Kubernetes technologies that made this possible were StatefulSets (for stable network identity and storage), PersistentVolumes (for data durability), Secrets (for secure credential management), and init containers (for automating replica initialization).

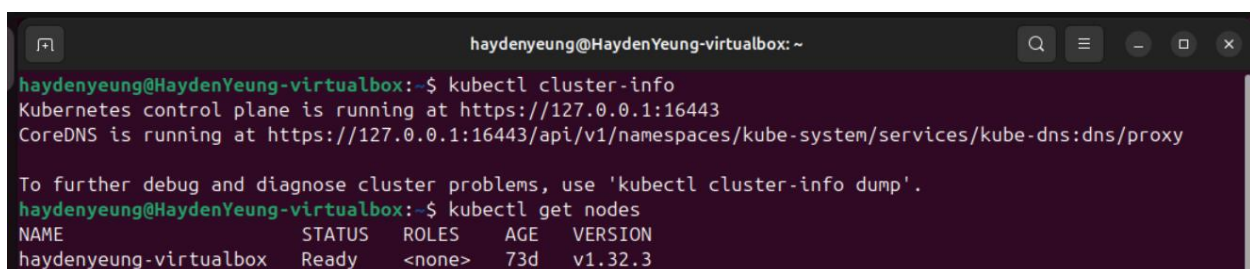
Overall, I found Kubernetes to be a powerful and suitable platform for deploying PostgreSQL with replication, provided one is prepared for the initial learning curve and configuration effort. The experience has given me confidence in both Kubernetes and PostgreSQL, and a strong foundation for future cloud-native database deployments.

V. Appendices

Step-by-step instructions:

1/ Setup the working environment:

- Perform basic checking with “kubectl cluster-info” & “kubectl get nodes” to make sure that the cluster and node is working fine.

A terminal window titled 'haydenyeung@HaydenYeung-virtualbox: ~' with search, menu, and window control icons in the title bar. The terminal shows the following commands and output:

```
haydenyeung@HaydenYeung-virtualbox:~$ kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:16443
CoreDNS is running at https://127.0.0.1:16443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
haydenyeung@HaydenYeung-virtualbox:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
haydenyeung-virtualbox Ready    <none>   73d   v1.32.3
```

2/ Create Kubernetes Resource Files

- I created 4 YAML files: postgres-secret.yaml, postgres-pv-pvc.yaml, postgres-service, and postgres-statefulset.yaml
- “postgres-secret.yaml”

```

GNU nano 7.2 postgres-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
type: Opaque
stringData:
  POSTGRES_USER: admin
  POSTGRES_PASSWORD: adminpass
  REPLICATION_USER: replicator
  REPLICATION_PASSWORD: replicatorpass

```

- “postgres-pv-pvc.yaml”

```

GNU nano 7.2 postgres-pv-pvc.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /mnt/data/postgres
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  volumeName: postgres-pv

```

- “postgres-service.yaml”

```

GNU nano 7.2 postgres-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres-headless
spec:
  clusterIP: None
  selector:
    app: postgres
  ports:
    - port: 5432
      name: postgres

```

- “postgres-statefulset.yaml”

```

GNU nano 7.2 postgres-statefulset.yaml *
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: postgres-headless
  replicas: 2
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:14
          ports:
            - containerPort: 5432
          envFrom:
            - secretRef:
                name: postgres-secret
          volumeMounts:
            - name: postgres-data
              mountPath: /var/lib/postgresql/data
      volumeClaimTemplates:
        - metadata:
            name: postgres-data
          spec:
            accessModes:
              - ReadWriteOnce
            resources:
              requests:
                storage: 1Gi

```

3/ Create a directory for hostPath

- I applied these two Linux command to generated the directory and set it to be owned by “postgres” user

“sudo mkdir -p /mnt/data/postgres”

“sudo chown 999:999 /mnt/data/postgres”

- “999:999” is the default UID/GID for the “postgres” user according to the Official Docker Image

4/ Applied all 4 YAML above

- After applied all four yaml files, I encountered the following error

```

haydenyeung@HaydenYeung-virtualbox: ~/task6.30$ kubectl get all
NAME                READY   STATUS    RESTARTS   AGE
pod/postgres-0      0/1     Pending   0           4m47s

NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP     10.152.183.1 <none>        443/TCP    73d
service/postgres-headless ClusterIP      None         <none>        5432/TCP   25m

NAME                READY   AGE
statefulset.apps/postgres 0/2     4m47s

```

- Applied command “kubectl describe pod postgres-0” to check its status

```

haydenyeung@HaydenYeung-virtualbox:~/task6.30$ kubectl describe pod postgres-0
Name:          postgres-0
Namespace:     default
Priority:       0
Service Account: default
Node:          <none>
Labels:        app=postgres
               apps.kubernetes.io/pod-index=0
               controller-revision-hash=postgres-5d996f8c8b
               statefulset.kubernetes.io/pod-name=postgres-0
Annotations:   <none>
Status:        Pending
IP:            <none>
IPs:           <none>
Controlled By: StatefulSet/postgres
Containers:
  postgres:
    Image:      postgres:14
    Port:       5432/TCP
    Host Port:  0/TCP
    Environment Variables from:
      postgres-secret Secret Optional: false
    Environment:  <none>
    Mounts:
      /var/lib/postgresql/data from postgres-data (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-l77jm (ro)
Conditions:
  Type             Status
  PodScheduled     False
Volumes:
  postgres-data:
    Type:          PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:     postgres-data-postgres-0
    ReadOnly:      false
  kube-api-access-l77jm:
    Type:          Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:  kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:    true
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason             Age   From          Message
  ----     -
  Warning  FailedScheduling   5m9s  default-scheduler  0/1 nodes are available: pod has unbound immediate PersistentVolumeClaims.
  Warning  Preemption         0/1 nodes are available: 1 Preemption is not helpful for scheduling.

```

- The error message indicated that the pod had an unbound immediate PersistentVolumeClaim
- I learned that StatefulSets automatically create a unique PersistentVolumeClaim for each pod, and my original static PersistentVolume did not match the required claim names.
- After researching, I switched to dynamic provisioning by removing the manual PV/PVC and letting Kubernetes handle storage allocation, which resolved the issue. This was simply achieved by removing the “volumeName: postpres-pv” located at the end of “volumeClaimTemplate:” – from “postgres-pv-pvc.yaml”


```

haydenyeung@HaydenYeung-virtualbox: ~/task6.3D$ kubectl get all
NAME                READY   STATUS    RESTARTS   AGE
pod/postgres-0       1/1     Running   0           54s
pod/postgres-1       1/1     Running   0           46s

NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP   10.152.183.1 <none>        443/TCP    73d
service/postgres-headless ClusterIP   None         <none>        5432/TCP   28m

NAME                READY   AGE
statefulset.apps/postgres 2/2     54s

```

- Re-applied the new “postgres-statefulset.yaml” and obtained the following result

```

haydenyeung@HaydenYeung-virtualbox: ~/task6.3D$ kubectl get all
NAME                READY   STATUS    RESTARTS   AGE
pod/postgres-0       1/1     Running   0           54s
pod/postgres-1       1/1     Running   0           46s

NAME                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP   10.152.183.1 <none>        443/TCP    73d
service/postgres-headless ClusterIP   None         <none>        5432/TCP   28m

NAME                READY   AGE
statefulset.apps/postgres 2/2     54s

```

5/ Initialize Database and create a replication user

- I accessed pod “postgres-0” through “kubectl exec -it postgres-0 -- bash”
- Enter “psql -U admin”
- Then create the replication user through the command shown in the following image:

```

haydenyeung@HaydenYeung-virtualbox: ~/task6.3D$ kubectl exec -it postgres-0 -- bash
root@postgres-0:/# psql -U admin
psql (14.18 (Debian 14.18-1.pgdg120+1))
Type "help" for help.

admin=# CREATE ROLE replicator WITH REPLICATION LOGIN ENCRYPTED PASSWORD 'replicatorpass';
CREATE ROLE

```

- Had a check on the existing roles through command “\du” inside the same window

```

admin=# \du

```

Role name	Attributes	Member of
admin	Superuser, Create role, Create DB, Replication, Bypass RLS	{}
replicator	Replication	{}

- Checked on the value of “wal_level”, “max_wal_senders”, and “hot_standby”

```

admin=# SHOW wal_level;
wal_level
-----
replica
(1 row)

admin=# SHOW max_wal_senders;
max_wal_senders
-----
10
(1 row)

admin=# hot_standby;
ERROR: syntax error at or near "hot_standby"
LINE 1: hot_standby;
        ^

admin=# SHOW hot_standby;
hot_standby
-----
on
(1 row)

```

- The meaning of those values are as follows:
 - **wal_level = replica**: Ensures the primary server logs enough data for streaming replication, allowing postgres-1 to stay in sync with postgres-0 using WAL records.
 - **max_wal_senders = 10**: Allows up to 10 replicas to connect, which supports a single replica setup (postgres-1) and provides flexibility for future scaling.
 - **hot_standby = on**: Enables the replica to handle read-only queries, which is consistent with my testing approach (e.g., querying the test table on postgres-1 to verify replication).

6/ Setup for the Replica Pod (postgres-1)

- In the following picture, ones can see that there were errors in this process and how I overcame them are also included in it.

```

haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl exec -it postgres-1 -- bash
root@postgres-1:/# pg_ctl -D "$PGDATA" -m fast stop
pg_ctl: cannot be run as root
Please log in (using, e.g., "su") as the (unprivileged) user that will
own the server process.
root@postgres-1:/# su - postgres
postgres@postgres-1:~$ pg_ctl -D "$PGDATA" -m fast stop
-bash: pg_ctl: command not found
postgres@postgres-1:~$ which pg_ctl
postgres@postgres-1:~$ find / -name pg_ctl 2>/dev/null
/usr/lib/postgresql/14/bin/pg_ctl
postgres@postgres-1:~$ /usr/lib/postgresql/14/bin/pg_ctl -D "$PGDATA" -m fast stop
pg_ctl: directory "" does not exist
postgres@postgres-1:~$ /usr/lib/postgresql/14/bin/pg_ctl -D /var/lib/postgresql/data -m fast stop
waiting for server to shut down....command terminated with exit code 137

```

- The purpose of that step was to stop the PostgreSQL server on pod postgres-1 so that its data can be removed or overwritten safely which setting an empty directory for replicating the entire data directory from primary pod (postgres-0) to replica (postgres-1) through “pg_basebackup” command.
- I tried to access to pod “postgres-1” and attempted to remove all the data stored in it through “rm -rf /var/lib/postgresql/data/*” for the first time, but was not be able to type in the “pg_basebackup” command as pod “postgres-1” restarted upon its data being wiped out – as shown in the following image.

```

haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl exec -it postgres-1 -- bash
root@postgres-1:/# su - postgres
postgres@postgres-1:~$ rm -rf /var/lib/postgresql/data/*
postgres@postgres-1:~$ pg_basebackup -h postgres-0.postgres-headless -D /var/lib/postgrescommand terminated with
exit code 137
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl exec -it postgres-1 -- bash
error: Internal error occurred: unable to upgrade connection: container not found ("postgres")
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/postgres-0                      1/1     Running   1           14h
pod/postgres-1                      1/1     Running   3 (24s ago)  14h

NAME                                TYPE               CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                  ClusterIP         10.152.183.1 <none>        443/TCP     73d
service/postgres-headless           ClusterIP         None         <none>        5432/TCP    14h

NAME                                READY   AGE
statefulset.apps/postgres           2/2     14h
haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl exec -it postgres-1 -- bash
root@postgres-1:/# su - postgres
postgres@postgres-1:~$ pg_basebackup -h postgres-0.postgres-headless -D /var/lib/postgresql/data -U replicator -W
--progress --verbose --wal-method=stream
Password:
pg_basebackup: error: connection to server at "postgres-0.postgres-headless" (10.1.186.33), port 5432 failed: FATAL: no pg_hba.conf entry for replication connection from host "10.1.186.53", user "replicator", no encryption
postgres@postgres-1:~$

```

- The second time, I managed to typed in the “pg_basebackup” command but encountered the error of connection.
- Through the help of DeepSeek and documents, I fixed this error by:

1/ Find the ip of both running pods to figure out the common subnet that can be used to “stored” both of them

```

haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE
INESS GATES
postgres-0                          1/1     Running   1           14h   10.1.186.33    haydenyeung-virtualbox
e>
postgres-1                          1/1     Running   3 (5m53s ago)  14h   10.1.186.53    haydenyeung-virtualbox
e>

```

- From this result, I figured that these pods are in the 10.1.186.0/24 subnet, and also within the broader 10.1.0.0/16 subnet.
- I re-accessed to “postgres-0” and edit its “pg_hba.conf” file by adding

```

haydenyeung@HaydenYeung-virtualbox:~/my-container$ kubectl exec -it postgres-0 -- bash
root@postgres-0:/# su - postgres
postgres@postgres-0:~$ nano /var/lib/postgresql/data/pg_hba.conf
-bash: nano: command not found
postgres@postgres-0:~$ echo "host replication replicator 10.1.0.0/16 md5" >> /var/lib/postgres/data/pg_hba.conf
-bash: /var/lib/postgres/data/pg_hba.conf: No such file or directory
postgres@postgres-0:~$ echo "host replication replicator 10.1.0.0/16 md5" >> /var/lib/postgresql/data/pg_hba.conf

```

- Then I checked its content through “cat /var/lib/postgres/data/pg_hba.conf” to make sure that the added line present in this file

```

# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all trust
# IPv4 local connections:
host all all 127.0.0.1/32 trust
# IPv6 local connections:
host all all ::1/128 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all trust
host replication all 127.0.0.1/32 trust
host replication all ::1/128 trust
host all all all scram-sha-256
host replication replicator 10.1.0.0/16 md5

```

- Then I reloaded the configuration file of PostgreSQL’ Server inside pod “postgres-0” to allow the replication connection from the replica, pod “postgres-1” in this case.

```

postgres@postgres-0:~$ /usr/lib/postgresql/14/bin/pg_ctl -D /var/lib/postgresql/data reload
server signaled

```

- After that, I logged back to pod “postgres-1” to retry to the remove data in it, and yet encountered error “command terminated with exit code 137” for numerous time which prevented me from applying “pg_basebackup” to this pod. With the help DeepSeek, Grok, and Google Gemini, I was suggested to add the initContainer to automate the initialization of the pod “postgres-1” as a streaming replica of the pod “postgres-0” to avoid the error of exit code 137. Furthermore, I learned that by doing this, I will not have to manually stop the server, clear the data directory, and then run “pg_basebackup” command – in order word, I set pod “postgres-1” immediately as the streaming replica directly at the beginning.
- Here are the code lines of initContainer added to the “postgres-statefulset.yaml” (which I included in my github, link is provided)


```

initContainers:
- name: init-replica
  image: postgres:14
  command:
  - bash
  - c
  - |
    if [ "${hostname}" = "postgres-1" ] && [ ! -f /var/lib/postgresql/data/PG_VERSION ]; then
      export PGPASSWORD="$REPLICATION_PASSWORD"
      pg_basebackup -h postgres-0.postgres-headless -D /var/lib/postgresql/data -U $REPLICATION_USER --progress --verbose
      touch /var/lib/postgresql/data/standby.signal
      echo "primary_conninfo = 'host=postgres-0.postgres-headless port=5432 user=$REPLICATION_USER password=$REPLICATION_PASSWORD'"
    fi
envFrom:
- secretRef:
    name: postgres-secret
volumeMounts:
- name: postgres-data
  mountPath: /var/lib/postgresql/data

```

- I re-applied this new “postgres-statefulset.yaml” and check the running pods to make sure they run correctly

```

haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl logs postgres-1 -c init-replica
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/2000028 on timeline 1
pg_basebackup: starting background WAL receiver
pg_basebackup: created temporary replication slot "pg_basebackup_66"
0/34989 kB (0%), 0/1 tablespace (...lib/postgresql/data/backup_label)
34999/34999 kB (100%), 0/1 tablespace (...ostgresql/data/global/pg_control)
34999/34999 kB (100%), 0/1 tablespace (...ostgresql/data/global/pg_control)
34999/34999 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 0/2000100
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: syncing data to disk ...
pg_basebackup: renaming backup_manifest.tmp to backup_manifest
pg_basebackup: base backup completed

```

```

^Chaydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl delete pod postgres-1
pod "postgres-1" deleted
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl apply -f postgres-statefulset.yaml
statefulset.apps/postgres configured
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl logs postgres-1 -c init-replica
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl logs postgres-1 -c init-replica
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl get all

NAME                READY   STATUS    RESTARTS   AGE
pod/postgres-0      1/1     Running   0           32s
pod/postgres-1      1/1     Running   0           36s

NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP     10.152.183.1 <none>        443/TCP    73d
service/postgres-headless  ClusterIP     None         <none>        5432/TCP   15h

NAME                READY   AGE
statefulset.apps/postgres  2/2     15h
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$

```

7/ Testing Replication

- I logged into pod “postgres-0” to create a table and insert to into a value

```

haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl exec -it postgres-0 -- bash
Defaulted container "postgres" out of: postgres, init-replica (init)
root@postgres-0:/# su - postgres
postgres@postgres-0:~$ -U admin
-bash: -U: command not found
postgres@postgres-0:~$ psql -U admin
psql (14.18 (Debian 14.18-1.pgdg120+1))
Type "help" for help.

admin=# CREATE TABLE replication_test (id SERIAL PRIMARY KEY, message TEXT);
CREATE TABLE
admin=# INSERT INTO replication_test (message) VALUES ('Hello from primary');
INSERT 0 1
admin=# SELECT * FROM replication_test;
 id |      message
-----+-----
  1 | Hello from primary
(1 row)

```

- In a new terminal window, I accessed to “postgres-1” to check if those changes in “postgres-0” reflected on this pod

```

haydenyeung@HaydenYeung-virtualbox: ~/task6.3D
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl exec -it postgres -- bash
Error from server (NotFound): pods "postgres" not found
haydenyeung@HaydenYeung-virtualbox:~/task6.3D$ kubectl exec -it postgres-1 -- bash
Defaulted container "postgres" out of: postgres, init-replica (init)
root@postgres-1:/# su - postgres
postgres@postgres-1:~$ psql -U admin
psql (14.18 (Debian 14.18-1.pgdg120+1))
Type "help" for help.

admin=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-----
 t
(1 row)

admin=# SELECT * FROM replication_test;
 id |      message
-----+-----
  1 | Hello from primary
(1 row)

admin=#

```



```

pid | usesysid | username | application_name | client_addr | client_hostname | client_port |      backend_start      | backen
xmin | state | sent_lsn | write_lsn | flush_lsn | replay_lsn | write_lag | flush_lag | replay_lag | sync_priority | sync_state |
reply_time
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
68 | 16385 | replicator | walreceiver | 10.1.186.57 | | 40336 | 2025-05-13 00:35:48.118438+00 |
streaming | 0/3000148 | 0/3000148 | 0/3000148 | 0/3000148 | | | 0 | async |
25-05-13 04:00:04.048794+00
(1 row)

```

- I found the same value is reflected on “postgres-1”, also, I did perform check-up on “pg_is_in_recovery();” in this pod and “pg_stat_replication” on “postgres-0”

- The replica (postgres-1) is in **recovery mode** as value “T” was found, which means it is acting as a **standby/replica** and is continuously applying changes streamed from the primary (postgres-0).
- For “pg_stat_replication”:
 - username: replicator (the replication user)
 - application_name: walreceiver (the process on the replica receiving WAL data)
 - client_addr: 10.1.186.57 (the IP of your replica pod)
 - state: streaming (the replica is actively connected and receiving data)
 - sent_lsn, write_lsn, flush_lsn, replay_lsn: All equal, meaning the replica is fully caught up—no replication lag
 - sync_state: async (asynchronous replication, which is the default)
 - This means that the primary node, “postgres-0”, recognizes the replica, “postgres-1”, as connected and streaming. This is the official confirmation from the primary’s perspective that replication is healthy and active.

References

1. PostgreSQL Global Development Group. (2024). *PostgreSQL 14 Documentation: Streaming Replication*. <https://www.postgresql.org/docs/14/warm-standby.html>
2. The Kubernetes Authors. (2024). *StatefulSets*. <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>
3. The Kubernetes Authors. (2024). *Persistent Volumes*. <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>
4. The Kubernetes Authors. (2024). *Secrets*. <https://kubernetes.io/docs/concepts/configuration/secret/>
5. DigitalOcean. (2022). *How To Set Up Replication in PostgreSQL on Ubuntu 20.04*. <https://www.digitalocean.com/community/tutorials/how-to-set-up-replication-in-postgresql-on-ubuntu-20-04>
7. PostgreSQL Global Development Group. (2024). *pg_basebackup — take a base backup of a PostgreSQL cluster*. <https://www.postgresql.org/docs/14/app-pgbasebackup.html>

8. Grinberg, M. (2021). *Running PostgreSQL on Kubernetes: What You Need to Know*.
<https://severalnines.com/blog/running-postgresql-kubernetes-what-you-need-know/>