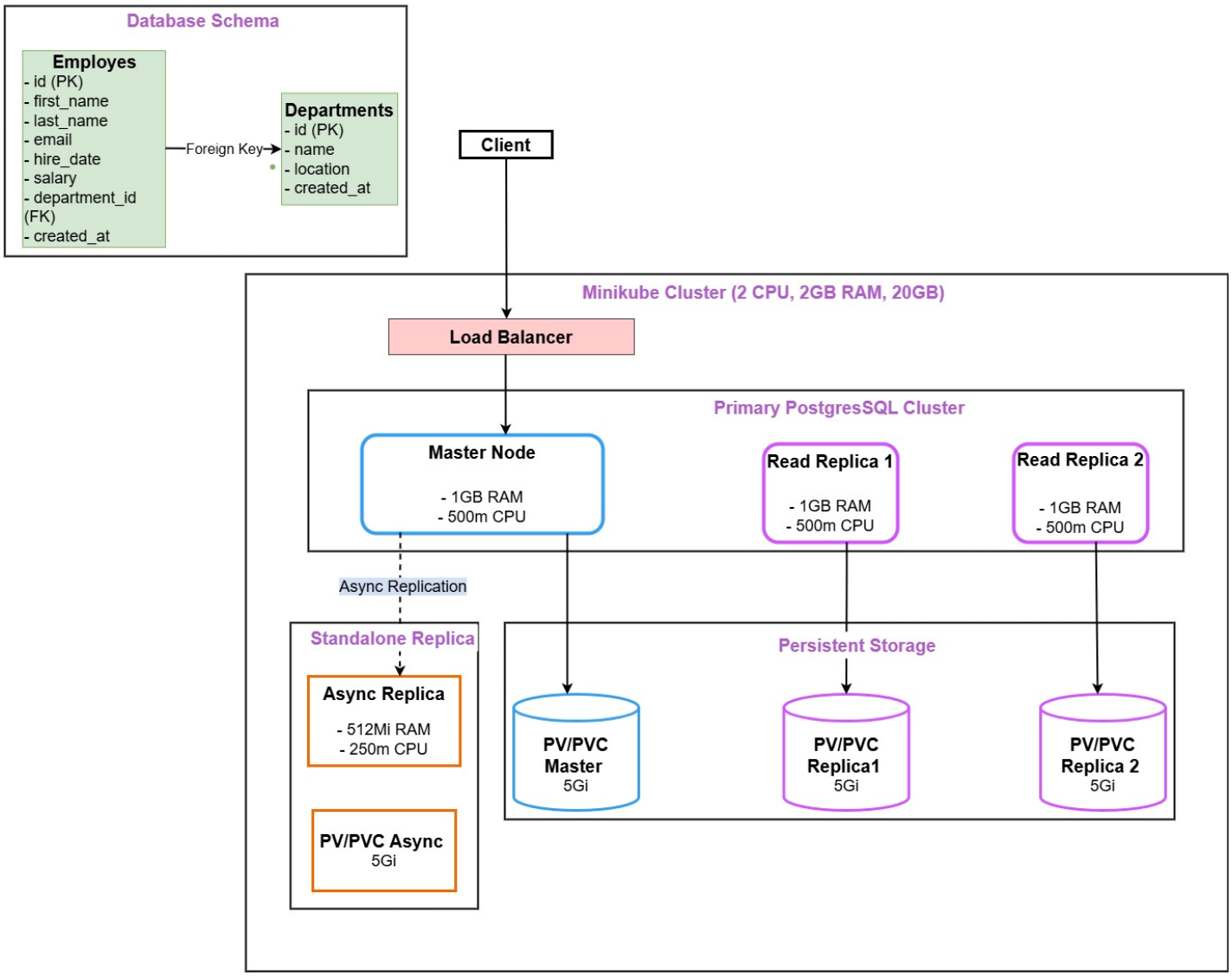
Documentation for PostgreSQL Database Cluster with Replication on Kubernetes

1. Overview of the Solution

This document provides an architectural overview and explanation of the solution implemented to meet the assignment requirements. It includes the deployment of a PostgreSQL database cluster with replication using Minikube and Helm, along with the creation and management of a standalone replica.



2. Architectural Diagram

Diagram Explanation:

The architectural diagram illustrates the following components:

1. **Minikube Kubernetes Cluster:**

- A local (localhost) Kubernetes cluster running on Minikube with resources of **2 CPUs**, **2 GB RAM**, and **20 GB Disk Space**.

2. **Primary (main) PostgreSQL Cluster:**
    - Deployed using Helm Charts.
    - Includes:
        - **1 Master Node** for read/write operations with **1 GB RAM, 500m CPU**.
        - **2 Read Replicas** for load balancing and scalability, each with **1 GB RAM, 500m CPU**.
        - **Persistent Volumes (PV/PVC)** for data storage, each with **5 GiB** capacity.
    - The **Load Balancer** ensures even distribution of traffic between nodes in the main cluster.
3. **Standalone PostgreSQL Replica:**
    - Deployed as a separate PostgreSQL instance using **Helm Charts.**
    - Includes:
        - **1 Async Replica Node** for replicating data from the master node.
        - **Persistent Volume (PV/PVC)** for data storage with **5 GiB** capacity.
    - Asynchronous replication is configured from the main cluster to the standalone replica.
4. **Database Schema:**
    - Two related tables:
        - **departments:** Stores department details.
        - **employees:** Stores employee details with a foreign key reference to the departments table.
    - **100,000 records** generated using the Faker library In Python.

3. Implementation Details

3.1 Kubernetes Cluster Setup

- Minikube was used to create a local Kubernetes cluster with the following configuration:
    - **2 CPUs**
    - **2 GB RAM**
    - **20 GB Disk Space**

3.2 Main PostgreSQL Cluster

- Helm Chart (bitnami/postgresql) was used to deploy the main PostgreSQL cluster.
- Configuration in **postgres-main-values.yaml** includes:
    - **Architecture:** Replication.
    - **2 Read Replicas** for scalability.

- **Load Balancer** for distributing traffic.
- Persistent storage for data durability.

3.3 Standalone Replica

- Another **Helm Chart** was deployed for the standalone PostgreSQL instance.
- Configuration in **standalone-values.yaml** includes:
- **Architecture:** Standalone.
- Asynchronous replication configured using a subscription.

3.4 Data Generation

- A Python script (**generate_and_insert_data.py**) was used to:
    - Create the departments and employees tables.
    - Populate **100,000 records** using the Faker library.

3.5 Replication Setup

- Asynchronous replication was configured with:
- **Publication** created in the main cluster.
- **Subscription** created in the standalone replica using replication-setup.sql.

4. Verification Steps

4.1 Verify the Main PostgreSQL Cluster

I will connect to the main PostgreSQL cluster, check tables, verify row counts, and insert a new row for replication testing.

Step 1: Connect to the Main PostgreSQL Cluster

Run the following command to connect to the main cluster as testuser:

```
kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb
```

- **Password:** test123

```
PS C:\WINDOWS\system32> kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb
Password for user testuser:
psql (17.2)
Type "help" for help.

testdb=>
```

Step 2: List Tables

To check the tables in the main cluster, run:

```
\dt
```

```
testdb=> \dt
           List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+--------
 public | departments | table | testuse
 public | employees   | table | testuse
(2 rows)

testdb=>
```

Step 3: Verify Row Counts

Check the number of rows in each table:

```
SELECT COUNT(*) FROM departments;

SELECT COUNT(*) FROM employees;
```

```
testdb=> SELECT COUNT(*) FROM departments;
 count
-------
    12
(1 row)

testdb=> SELECT COUNT(*) FROM employees;
 count
--------
 100000
(1 row)

testdb=>
```

Step 4: Insert a New Row

Insert a new row into the **departments** table:

```
INSERT INTO departments (name, location) VALUES ('New Department', 'Kigali');
```

```
testdb=> INSERT INTO departments (name, location) VALUES ('New Department', 'Kigali');
INSERT 0 1
testdb=>
```

Step 5: Verify the New Row

Query the departments table for the new row:

```
SELECT * FROM departments WHERE name = 'New Department';
```

- **Expected Output:**

```
testdb=> SELECT * FROM departments WHERE name = 'New Department';
 id |      name      |   location    |          created_at
----+----------------+---------------+----------------------------
 11 | New Department | New York      | 2024-11-25 11:27:15.527335
 12 | New Department | New Hampshire | 2024-11-25 11:36:03.511199
 13 | New Department | Kigali        | 2024-11-25 13:37:42.810789
(3 rows)

testdb=>
```

Step 6: Check he Structure of Both Tables:

run:

```
testdb=> \d departments
                                    Table "public.departments"
   Column   |            Type             | Collation | Nullable |                 Default
------------+-----------------------------+-----------+----------+----------------------------------------
 id         | integer                     |           | not null | nextval('departments_id_seq'::regclass)
 name       | character varying(100)      |           | not null |
 location   | character varying(100)      |           |          |
 created_at | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "departments_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "employees" CONSTRAINT "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)
Publications:
    "main_pub"

testdb=> \d employees
                                     Table "public.employees"
    Column     |            Type             | Collation | Nullable |                 Default
---------------+-----------------------------+-----------+----------+----------------------------------------
 id            | integer                     |           | not null | nextval('employees_id_seq'::regclass)
 first_name    | character varying(50)       |           | not null |
 last_name     | character varying(50)       |           | not null |
 email         | character varying(100)      |           | not null |
 hire_date     | date                        |           | not null |
 salary        | numeric(10,2)               |           | not null |
 department_id | integer                     |           |          |
 created_at    | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "employees_pkey" PRIMARY KEY, btree (id)
    "employees_email_key" UNIQUE CONSTRAINT, btree (email)
Foreign-key constraints:
    "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)
Publications:
    "main_pub"

testdb=>
```

Step 7: Exit the Main Cluster

To disconnect from the main cluster, run:

```
\q
```

4.2 Verify the Standalone PostgreSQL Replica

Now, connect to the standalone replica to verify that the data has been replicated.

Step 1: Connect to the Standalone Replica

Run the following command to connect to the standalone replica as testuser:

```
kubectl exec -it postgres-standalone-postgresql-0 -n default -- psql -U testuser -d testdb
```

- **Password:** test123

```
PS C:\WINDOWS\system32> kubectl exec -it postgres-standalone-postgresql-0 -n default -- psql -U testuser -d testdb
Password for user testuser:
psql (17.2)
Type "help" for help.

testdb=>
```

Step 2: List Tables

To check the tables in the standalone replica, run:

```
\dt
```

- **Expected Output:**

```
testdb=> \dt
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | departments | table | testuser
 public | employees   | table | testuser
(2 rows)

testdb=>
```

Step 3: Verify Row Counts

Check the number of rows in each table to confirm replication:

```
SELECT COUNT(*) FROM departments;

SELECT COUNT(*) FROM employees;
```

```
testdb=> SELECT COUNT(*) FROM departments;
 count
-------
    13
(1 row)

testdb=> SELECT COUNT(*) FROM employees;
 count
--------
 100000
(1 row)

testdb=>
```

Step 4: Verify the New Row

Check if the new row inserted in the main cluster is present in the standalone replica:

```
SELECT * FROM departments WHERE name = 'New Department';
```

```
testdb=> SELECT * FROM departments WHERE name = 'New Department';
 id |      name      |    location    |         created_at
----+----------------+----------------+----------------------------
 11 | New Department | New York       | 2024-11-25 11:27:15.527335
 12 | New Department | New Hampshire | 2024-11-25 11:36:03.511199
 13 | New Department | Kigali         | 2024-11-25 13:37:42.810789
(3 rows)

testdb=>
```

Step 5: Again check structure of both Tables in standalone

Run:

```
\d departments

\d employees
```

```
testdb=> \d departments
                                    Table "public.departments"
   Column   |            Type             | Collation | Nullable |                 Default
------------+-----------------------------+-----------+----------+----------------------------------------
 id         | integer                     |           | not null | nextval('departments_id_seq'::regclass)
 name       | character varying(100)      |           | not null |
 location   | character varying(100)      |           |          |
 created_at | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "departments_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "employees" CONSTRAINT "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)

testdb=> \d employees
                                     Table "public.employees"
    Column     |            Type             | Collation | Nullable |                Default
---------------+-----------------------------+-----------+----------+---------------------------------------
 id            | integer                     |           | not null | nextval('employees_id_seq'::regclass)
 first_name    | character varying(50)       |           | not null |
 last_name     | character varying(50)       |           | not null |
 email         | character varying(100)      |           | not null |
 hire_date     | date                        |           | not null |
 salary        | numeric(10,2)               |           | not null |
 department_id | integer                     |           |          |
 created_at    | timestamp without time zone |           |          | CURRENT_TIMESTAMP
Indexes:
    "employees_pkey" PRIMARY KEY, btree (id)
    "employees_email_key" UNIQUE CONSTRAINT, btree (email)
Foreign-key constraints:
    "employees_department_id_fkey" FOREIGN KEY (department_id) REFERENCES departments(id)

testdb=>
```

Step 6: Exit the Standalone Replica

To disconnect from the standalone replica, run:

```
\q
```

4.3 Additional Verification

Check Replication Status on the Main Cluster

Run the following command on the main cluster to view replication status:

```
kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb
```

Then execute:

sql

Copy code

```
SELECT * FROM pg_stat_replication;
```

```
PS C:\WINDOWS\system32> kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb
Password for user testuser:
psql (17.2)
Type "help" for help.

testdb=> SELECT * FROM pg_stat_replication;
  pid  | usesysid |  usename  | application_name | client_addr | client_hostname | client_port | backend_start | bac
lay_lag | sync_priority | sync_state | reply_time
-------+----------+-----------+------------------+-------------+-----------------+-------------+---------------+----
--------+---------------+------------+------------
 45509 |    16386 | repl_user | my_application   |             |                 |             |               |
       |          |           |
 45493 |       10 | postgres  | main_sub         |             |                 |             |               |
       |          |           |
(2 rows)
```

It is true that no port is showing, and here's why:

In the output of **pg_stat_replication**, the **client_por**t and **client_addr** fields are empty because:

1. Replication Connection Setup:
    - PostgreSQL logical replication may not populate client_addr and client_port if the connection is made over a local socket (e.g., within a Kubernetes pod).
    - In a Kubernetes environment, the replication might be happening internally through the Kubernetes service (ClusterIP), so there's no direct IP or port information to display.
2. Internal Networking:
    - If the subscriber connects using a hostname (e.g., **postgres-main-postgresql-primary.default.svc.cluster.local**), PostgreSQL might not populate these fields because it's using internal Kubernetes networking rather than a direct IP/port connection.

Key Takeaway:

The absence of **client_addr** and **client_port** is expected in such environments and does not indicate any issue with the replication setup. Replication is functioning as evidenced by the rows displayed in the **pg_stat_replication** output.

Check Subscription Status on the Standalone Replica

Run the following command on the standalone replica:

bash

Copy code

```
kubectl exec -it postgres-standalone-postgresql-0 -n default -- psql -U testuser -d testdb
```

Then execute:

sql

Copy code

```
SELECT * FROM pg_stat_subscription;
```

```
PS C:\WINDOWS\system32> kubectl exec -it postgres-standalone-postgresql-0 -n default -- psql -U testuser -d testdb
Password for user testuser:
psql (17.2)
Type "help" for help.

testdb=> SELECT * FROM pg_stat_subscription;
 subid | subname  | worker_type |  pid  | leader_pid | relid | received_lsn |      last_msg_send_time       |     last_msg_receipt_time     | latest_end_lsn |       latest_end_time
-------+----------+-------------+-------+------------+-------+--------------+-------------------------------+-------------------------------+----------------+-------------------------------
 16416 | main_sub | apply       | 56635 |            |       | 0/5318928    | 2024-11-25 14:14:17.740451+00 | 2024-11-25 14:14:17.740653+00 | 0/5318928      | 2024-11-25 14:14:17.740451+00
(1 row)

testdb=>
```

The pg_stat_subscription view on the **standalone replica** confirms that:

1. The subscription **main_sub** is active and working (**worker_type**: apply).
2. The replica is receiving and processing replication data, as indicated by:
    - **last_msg_send_time** and **last_msg_receipt_time** timestamps are recent.
    - **received_lsn** matches the **latest_end_lsn**, showing replication is **up-to-date.**

5. Tools and Technologies

- **Minikube:** Local Kubernetes cluster.
- **Helm Charts:** To deploy PostgreSQL clusters.
- **PostgreSQL 17.2:** Database engine.
- **Python (Faker Library):** For generating test data.
- **YAML Configuration:** For defining Kubernetes and Helm setups.

Here's the additional **verification commands** to include in your documentation, focusing on checking logs, cluster, and pods, along with their purposes and expected outputs.

6. Additional Verification Commands

6.1 Check Pods in the Cluster

Use this command to list all the running pods in the default namespace. This ensures that all required components (primary, replicas, standalone) are running.

```
kubectl get pods -n default
```

- **Purpose:** Verify the status of all pods in the cluster.

```
PS C:\WINDOWS\system32> kubectl get pods -n default
NAME                                READY   STATUS    RESTARTS   AGE
postgres-main-postgresql-primary-0  1/1     Running   0          23h
postgres-main-postgresql-read-0     1/1     Running   0          25h
postgres-standalone-postgresql-0    1/1     Running   0          25h
PS C:\WINDOWS\system32>
```

6.2 Check Services in the Cluster

Use this command to check all services and ensure that the LoadBalancer and ClusterIP are properly set up.

```
kubectl get svc -n default
```

- **Purpose:** Verify service endpoints and their configurations.

```
PS C:\WINDOWS\system32> kubectl get svc -n default
NAME                                 TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes                           ClusterIP      10.96.0.1        <none>        443/TCP          27h
postgres-main-postgresql-primary     LoadBalancer   10.106.42.255    <pending>     5432:30408/TCP   25h
postgres-main-postgresql-primary-hl  ClusterIP      None             <none>        5432/TCP         25h
postgres-main-postgresql-read        ClusterIP      10.111.135.202   <none>        5432/TCP         25h
postgres-main-postgresql-read-hl     ClusterIP      None             <none>        5432/TCP         25h
postgres-standalone-postgresql       ClusterIP      10.110.121.37    <none>        5432/TCP         25h
postgres-standalone-postgresql-hl    ClusterIP      None             <none>        5432/TCP         25h
PS C:\WINDOWS\system32>
```

6.3 Describe the Pods

Use this command to check the details of the main PostgreSQL primary pod.

kubectl describe pod postgres-main-postgresql-primary-0 -n default

- **Purpose:** View detailed information about the pod, such as events, container states, and resource usage.

  Detailed information about the pod, including:

```
PS C:\WINDOWS\system32> kubectl describe pod postgres-main-postgresql-primary-0 -n default
Name:             postgres-main-postgresql-primary-0
Namespace:        default
Priority:         0
Service Account:  postgres-main-postgresql
Node:             minikube/192.168.49.2
Start Time:       Sun, 24 Nov 2024 16:58:01 +0200
Labels:           app.kubernetes.io/component=primary
                  app.kubernetes.io/instance=postgres-main
                  app.kubernetes.io/managed-by=Helm
                  app.kubernetes.io/name=postgresql
                  app.kubernetes.io/version=17.2.0
                  apps.kubernetes.io/pod-index=0
                  controller-revision-hash=postgres-main-postgresql-primary-5b6f98cb8
                  helm.sh/chart=postgresql-16.2.2
                  statefulset.kubernetes.io/pod-name=postgres-main-postgresql-primary-0
Annotations:      kubectl.kubernetes.io/restartedAt: 2024-11-24T15:25:43+02:00
Status:           Running
IP:               10.244.0.16
IPs:
  IP:           10.244.0.16
Controlled By:  StatefulSet/postgres-main-postgresql-primary
Containers:
  postgresql:
    Container ID:    docker://2a9e552daab0c919caefec515448cccb93840db3b7d937ed717fb6e02c19d5f4
    Image:           docker.io/bitnami/postgresql:17.2.0-debian-12-r0
    Image ID:        docker-pullable://bitnami/postgresql@sha256:1dd43b042f79d184b28e6012b72621b0b43438e4695210cdfa4d40a9c48e9354
    Port:            5432/TCP
    Host Port:       0/TCP
    SeccompProfile:  RuntimeDefault
    State:           Running
      Started:       Sun, 24 Nov 2024 16:58:02 +0200
    Ready:           True
    Restart Count:   0
    Limits:
      cpu:     500m
      memory:  1Gi
    Requests:
      cpu:      250m
      memory:   512Mi
    Liveness:   exec [/bin/sh -c exec pg_isready -U "testuser" -d "dbname=testdb" -h 127.0.0.1 -p 5432] delay=30s timeout=5s period=10s #success=1 #failure=6
    Readiness:  exec [/bin/sh -c -e exec pg_isready -U "testuser" -d "dbname=testdb" -h 127.0.0.1 -p 5432
 -f /opt/bitnami/postgresql/tmp/.initialized ] || [ -f /bitnami/postgresql/.initialized ]
 delay=5s timeout=5s period=10s #success=1 #failure=6
    Environment:
      BITNAMI_DEBUG:                      false
      POSTGRESQL_PORT_NUMBER:             5432
      POSTGRESQL_VOLUME_DIR:              /bitnami/postgresql
      PGDATA:                             /bitnami/postgresql/data
```

6.4 Check Logs for PostgreSQL Pods

View logs for the PostgreSQL primary pod to identify replication or startup errors.

kubectl logs postgres-main-postgresql-primary-0 -n default

- **Purpose:** Debug and monitor PostgreSQL pod behavior, especially during replication.

- Logs showing PostgreSQL starting, connections being made, and replication events.

6.5 Port Forwarding for External Access

If you want to access PostgreSQL from outside the cluster, use port forwarding:

```
kubectl port-forward --namespace default svc/postgres-main-postgresql-primary 5432:5432 &
```

- **Purpose:** Allow local access to the database without exposing it externally.

6.6 Check Configurations of Secrets

Ensure secrets such as PostgreSQL passwords are correctly stored:

kubectl get secret postgres-main-postgresql -n default -o yaml

- **Purpose:** Verify that the database credentials are configured correctly.

```
PS C:\WINDOWS\system32> kubectl get secret postgres-main-postgresql -n default -o yaml
apiVersion: v1
data:
  password: dGVzdDEyMw==
  postgres-password: aE1QS1lkNDVsYQ==
  replication-password: cmVwbDEyMw==
kind: Secret
metadata:
  annotations:
    meta.helm.sh/release-name: postgres-main
    meta.helm.sh/release-namespace: default
  creationTimestamp: "2024-11-24T12:55:07Z"
  labels:
    app.kubernetes.io/instance: postgres-main
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: postgresql
    app.kubernetes.io/version: 17.2.0
    helm.sh/chart: postgresql-16.2.2
  name: postgres-main-postgresql
  namespace: default
  resourceVersion: "5027"
  uid: 9c896262-2624-4ab7-b406-fb097e4b257c
type: Opaque
```

6. GitHub Repository

All source files, including Helm charts, scripts, and configurations, are available in the following GitHub repository: https://github.com/callixte12/DBA-Task

7. Conclusion

This solution implements a robust PostgreSQL cluster with replication using Kubernetes and Helm. All requirements of the assignment have been fulfilled:

1. Minikube cluster deployed.
2. PostgreSQL main cluster with a load balancer.
3. Standalone PostgreSQL replica with asynchronous replication.
4. Database schema with 100,000 records.
5. Verification of replication and database operations.

References

- Kubernetes Documentation
- PostgreSQL Documentation
- Bitnami Helm Charts
- Faker Python Library Documentation
- TQDM Python Library Documentation

----- END----

Names: Callixte Muhawenimana

Date: November 24th, 2024