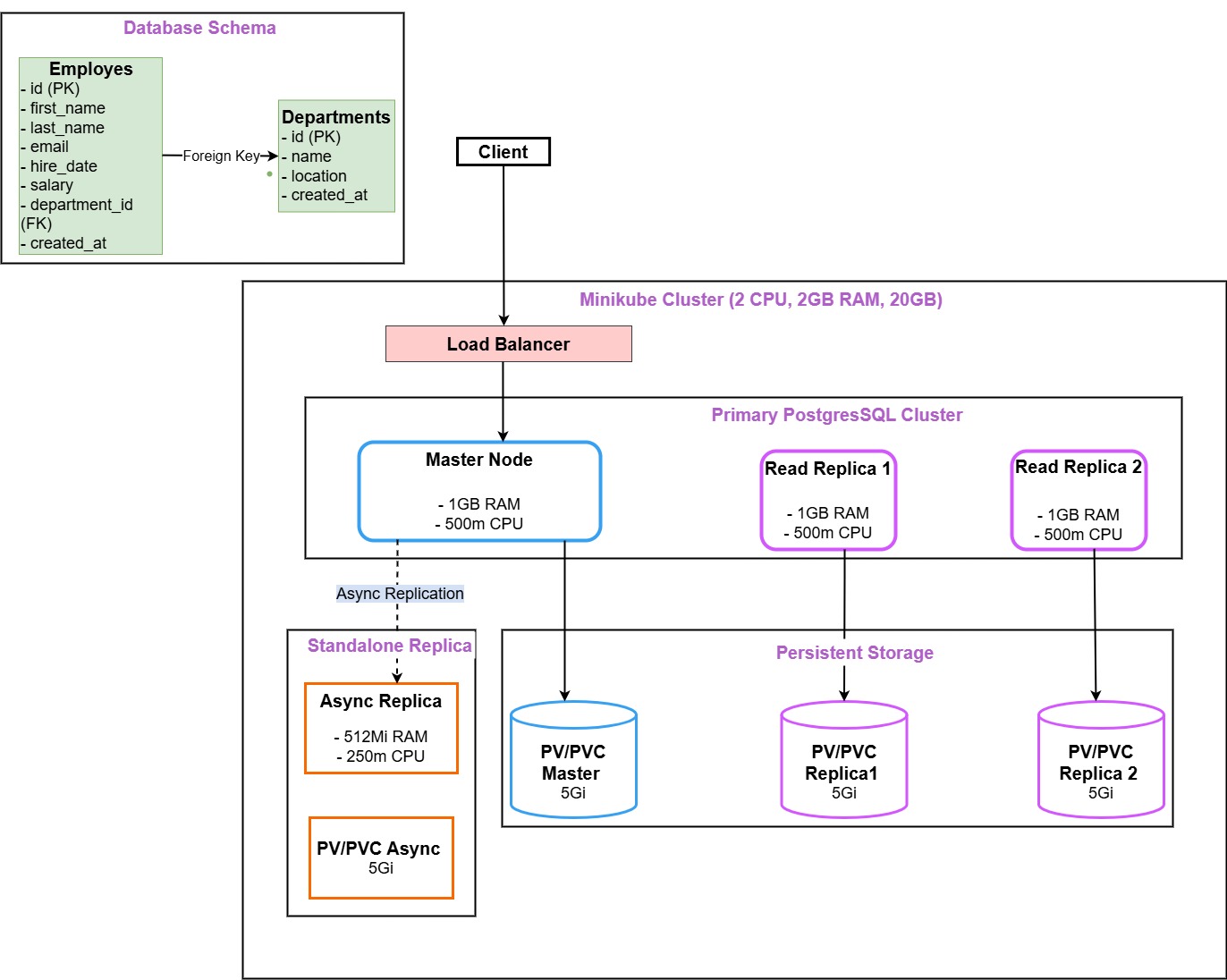
**Documentation for PostgreSQL Database Cluster with Replication on Kubernetes**

**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**

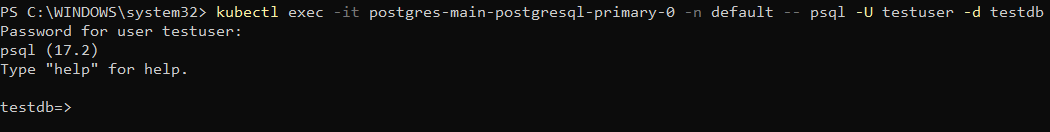
I will connect to the main PostgreSQL cluster, check tables, verify row counts, and insert a new row for replication testing.

**Step 1: Connect to the Main PostgreSQL Cluster**

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

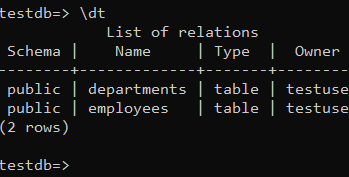
**kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb**

* **Password:** test123

**Step 2: List Tables**

To check the tables in the main cluster, run:

**\dt**

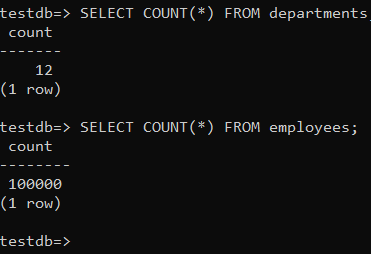


**Step 3: Verify Row Counts**

Check the number of rows in each table:

**SELECT COUNT(\*) FROM departments;**

**SELECT COUNT(\*) FROM employees;**

****

**Step 4: Insert a New Row**

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

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

****

**Step 5: Verify the New Row**

Query the departments table for the new row:

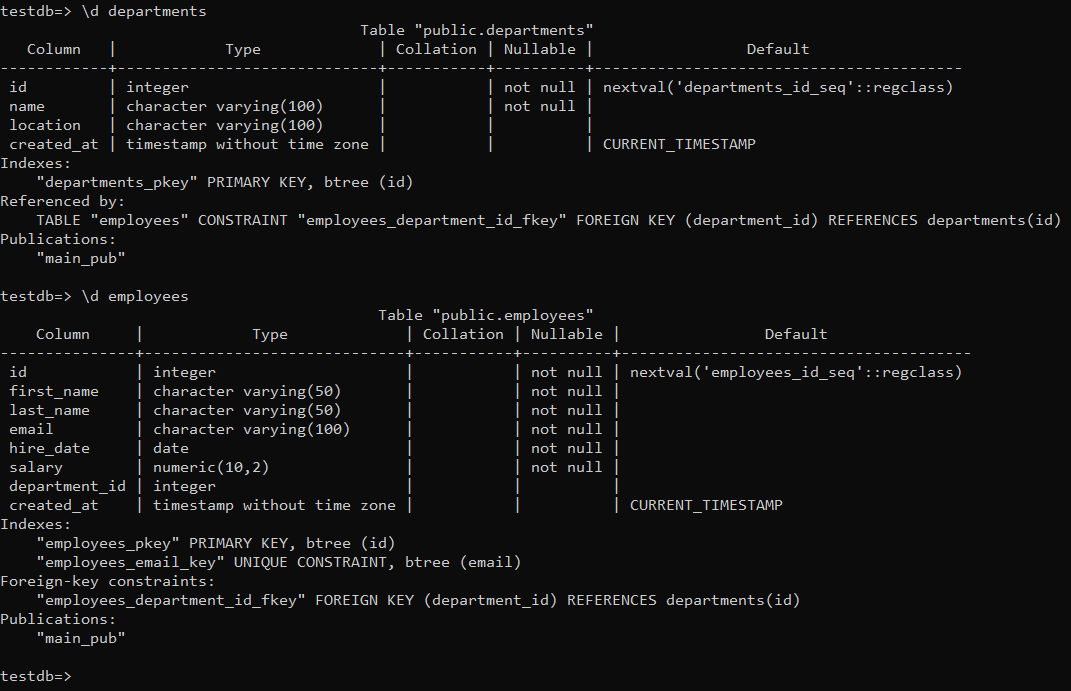
**SELECT \* FROM departments WHERE name = 'New Department';**

* **Expected Output:**

**A computer screen with white text

Description automatically generated**

**Step 6: Check he Structure of Both Tables:**run:



**Step 7: Exit the Main Cluster**

To disconnect from the main cluster, run:

**\q**

**4.2 Verify the Standalone PostgreSQL Replica**

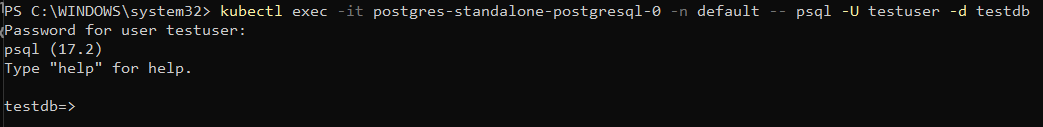
Now, connect to the standalone replica to verify that the data has been replicated.

**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



**Step 2: List Tables**

To check the tables in the standalone replica, run:

**\dt**

* **Expected Output:**

**A computer screen shot of a black background

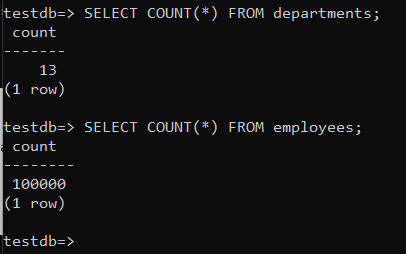
Description automatically generated**

**Step 3: Verify Row Counts**

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

**SELECT COUNT(\*) FROM departments;**

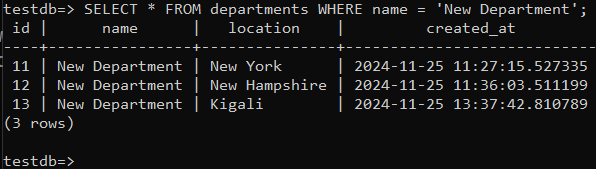
**SELECT COUNT(\*) FROM employees;**

****

**Step 4: Verify the New Row**

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

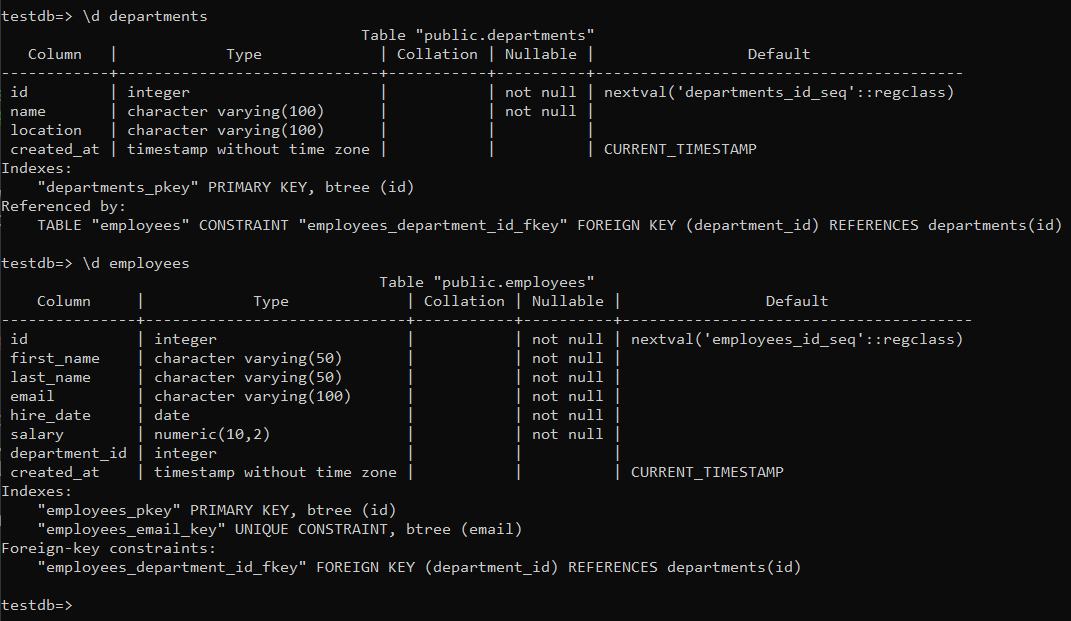
**SELECT \* FROM departments WHERE name = 'New Department';**

****

**Step 5: Again check structure of both Tables in standalone**

Run:

**\d departments**

**\d employees**  


**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:

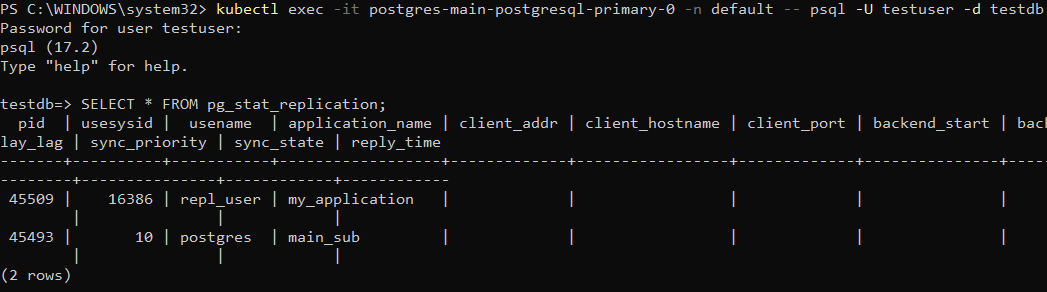
**kubectl exec -it postgres-main-postgresql-primary-0 -n default -- psql -U testuser -d testdb**

Then execute:

**sql**

Copy code

**SELECT \* FROM pg\_stat\_replication;**

****

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;**

**A black background with text

Description automatically generated**

**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.

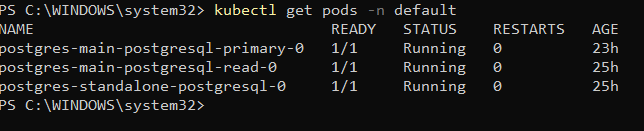
**6. Additional Verification Commands**

**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.

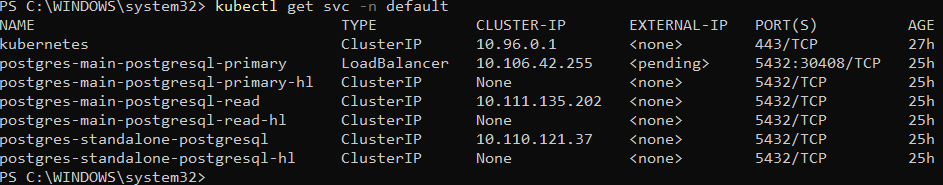
****

**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.

****

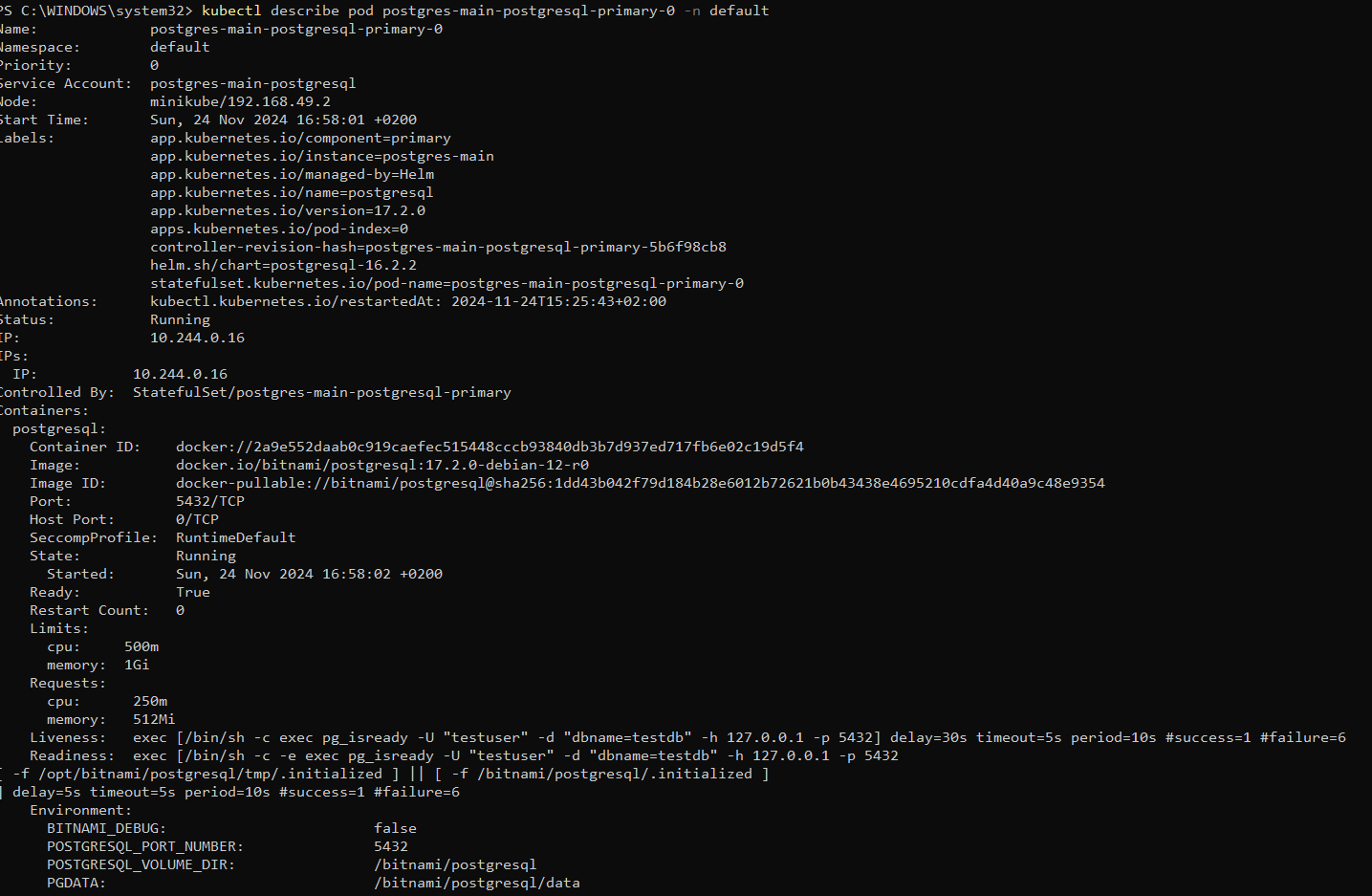
**6.3 Describe the Pods**

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:



**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 &**

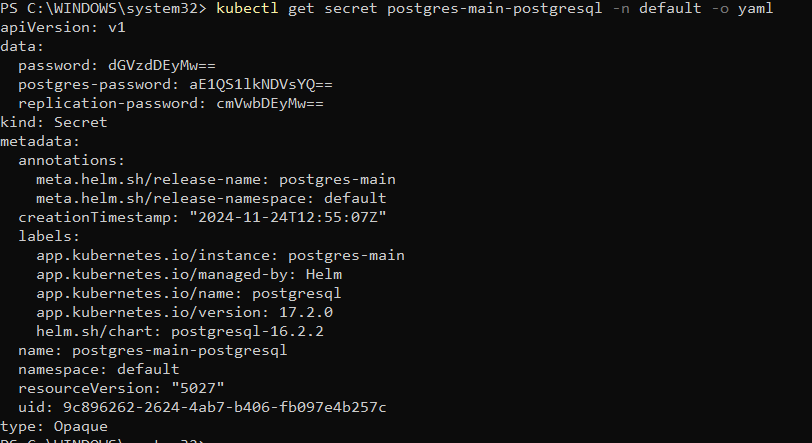
* **Purpose:** Allow local access to the database without exposing it externally.

**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.



**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](https://kubernetes.io/docs/home/)
* [PostgreSQL Documentation](https://www.postgresql.org/docs/)
* [Bitnami Helm Charts](https://github.com/bitnami/charts)
* [Faker Python Library Documentation](https://faker.readthedocs.io/en/master/)
* [TQDM Python Library Documentation](https://tqdm.github.io/)

----- END----

*Names: Callixte Muhawenimana*

*Date: November 24th, 2024*