**Report: Jenkins Pipeline for PostgreSQL Table Counts**

**1. Objective**

The goal of this project was to automate the process of connecting to PostgreSQL databases (db1 and db2) and retrieving table counts (schools and students) using a Jenkins pipeline. Security best practices required that the database credentials be stored in Jenkins credentials rather than hardcoding them in scripts.

**2. Environment Setup**

**2.1 Docker Compose**

We used Docker Compose to spin up two PostgreSQL containers. The docker-compose.yml was as follows:

services:

db1:

image: postgres:14

environment:

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- POSTGRES\_DB=${POSTGRES\_DB}

ports:

- "5432:5432"

volumes:

- db1\_data:/var/lib/postgresql/data

db2:

image: postgres:14

environment:

- POSTGRES\_USER=${POSTGRES\_USER}

- POSTGRES\_PASSWORD=${POSTGRES\_PASSWORD}

- POSTGRES\_DB=${POSTGRES\_DB}

ports:

- "5433:5432"

volumes:

- db2\_data:/var/lib/postgresql/data

volumes:

db1\_data:

db2\_data:

* db1 maps host port 5432
* db2 maps host port 5433
* Environment variables POSTGRES\_USER, POSTGRES\_PASSWORD, and POSTGRES\_DB are set in a .env file or passed via shell environment.

**2.2 Database Schema and Data**

We created two tables: schools and students:

**01\_schema.sql**

DROP TABLE IF EXISTS students;

DROP TABLE IF EXISTS schools;

CREATE TABLE schools (

id SERIAL PRIMARY KEY,

school\_name TEXT NOT NULL UNIQUE,

country TEXT NOT NULL,

number\_of\_students INT NOT NULL DEFAULT 0

);

CREATE TABLE students (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

joined\_at TIMESTAMP NOT NULL DEFAULT NOW(),

grade VARCHAR(2) NOT NULL,

school\_name TEXT NOT NULL,

phone\_number TEXT NOT NULL

);

CREATE INDEX idx\_students\_school\_name ON students (school\_name);

**02\_seed.sql**

INSERT INTO schools (school\_name, country, number\_of\_students)

SELECT

'School ' || gs,

(ARRAY['Jordan','Saudi Arabia','UAE','Egypt','Iraq'])[1 + floor(random()\*5)::int],

100 + floor(random()\*1901)::int

FROM generate\_series(1, 200) AS gs;

INSERT INTO students (name, joined\_at, grade, school\_name, phone\_number)

SELECT

'Student ' || gs,

NOW() - (gs || ' days')::interval,

(ARRAY['A','B','C','D','F'])[1 + floor(random()\*5)::int],

'School ' || (1 + floor(random()\*200)::int),

'079' || LPAD((1000000 + gs)::text, 7, '0')

FROM generate\_series(1, 200) AS gs;

Each table contains **200 records**.

**3. Jenkins Pipeline**

We used a **Declarative Jenkins Pipeline** with credentials stored securely in Jenkins. The credential ID is pg-creds (username: ahmad, password: ahmad).

**3.1 Pipeline Code**

pipeline {

agent any

environment {

DB\_NAME = "mydb"

DB\_USER = "ahmad"

DB\_HOST = "localhost"

DB\_PORT = "5432"

}

stages {

stage('Print Table Counts') {

steps {

withCredentials([usernamePassword(credentialsId: 'pg-creds', usernameVariable: 'DB\_USER', passwordVariable: 'DB\_PASS')]) {

sh '''

echo "Schools count:"

PGPASSWORD=$DB\_PASS psql -h $DB\_HOST -p $DB\_PORT -U $DB\_USER -d $DB\_NAME -t -c "SELECT COUNT(\*) FROM schools;"

echo "Students count:"

PGPASSWORD=$DB\_PASS psql -h $DB\_HOST -p $DB\_PORT -U $DB\_USER -d $DB\_NAME -t -c "SELECT COUNT(\*) FROM students;"

'''

}

}

}

}

}

**3.2 Key Notes**

1. **Credentials Usage:**
   * withCredentials ensures the username and password are not hardcoded.
   * DB\_USER and DB\_PASS are injected as environment variables temporarily.
2. **Database Connection:**
   * Host, port, and database name are defined in the environment block.
   * PGPASSWORD is used for authentication securely.
3. **Query Execution:**
   * psql -t -c "SELECT COUNT(\*) FROM table;" returns only the count without headers.

**4. Pipeline Output**

=== Table Counts from mydb ===

Schools count:

200

Students count:

200

This confirms that all records were inserted correctly and the pipeline can access the database securely.

**5. Security Advantages**

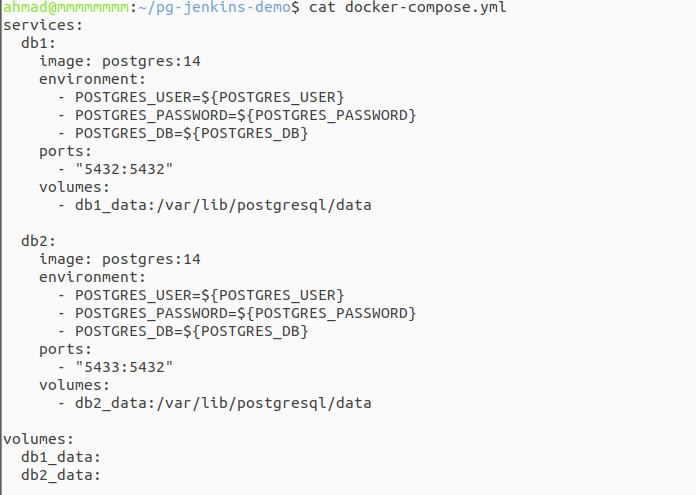
* **No hardcoding:** Credentials are never written directly in code.
* **Credential masking:** Jenkins hides passwords in console output.
* **Environment separation:** Each stage uses environment variables injected at runtime.

**6. Conclusion**

The Jenkins pipeline successfully connects to PostgreSQL, retrieves table counts, and demonstrates best practices for handling sensitive credentials. Using withCredentials improves security and makes pipelines safer for production.

Here’s some screenshots:





Hidden file .env:

