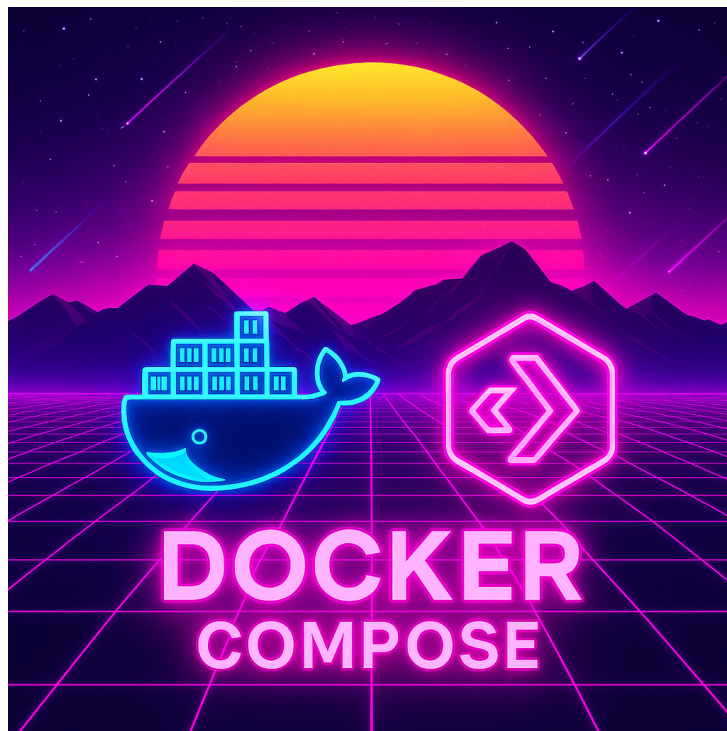


BICA Backup Solution

DevOps Junior Technical Exercise – Premium Minds



João Contramestre

August 4, 2025

Contents

1	Introduction	3
1.1	Problem Approach	3
2	Project Structure	4
2.1	Requirements	4
3	Backup Bash Script: backup.sh	5
3.1	Workflow Steps	5
3.2	Script Code	5
3.3	Environment Variables	7
3.4	Backup Naming Convention	7
4	Cron Scheduling: crontab.txt	8
4.1	Explanation	8
4.2	Key Points	8
5	Docker Compose Setup: docker-compose.yml	9
5.1	Services Overview	9
5.2	docker-compose.yml File	9
5.3	Explanation of Each Step	10
5.4	Benefits of Using Docker Compose	11
6	Dockerfile	12
6.1	Dockerfile Explanation	12
6.2	Explanation of Each Step	12
6.3	Key Advantages	14
6.4	Deployment	15
7	Entrypoint Script: entrypoint.sh	16
7.1	Script Code	16
7.2	Explanation of Each Step	16
7.3	Benefits of This Approach	16
8	Verification and Results	17
8.1	Creating the Containers	17
8.2	Backup without Encryption	18
8.3	Automatic Cleanup Test	18
8.4	Backup Content Verification	19
8.5	Backup with Encryption and Decryption Test	21
9	Tested Workflow	24
9.1	1. Insert Sample Data	24
9.2	2. Trigger a Manual Backup	24
9.3	3. Verify Backup Files	24
9.4	4. Decrypt a Backup (if encrypted)	24
9.5	5. Extract and Restore	24
9.6	6. Verify Data After Restore	25
9.7	Workflow Summary	25

10	Deployment Scenarios	26
10.1	Scenario 1: Local Linux Server with PostgreSQL	26
10.1.1	Preparation	26
10.1.2	Start the Backup Container	27
10.2	Scenario 2: Deployment via Docker Hub	27
10.2.1	Steps	27
10.2.2	Advantages	27
10.3	Summary of Integration	28
11	Important Commands	29
11.1	Docker and Docker Compose	29
11.2	Logs and Monitoring	29
11.3	Backup Decryption	29
11.4	Database Testing Commands	30
11.5	Maintenance and Cleanup	30
12	Future Improvements & Scalability	31
12.1	Logging and Monitoring	31
12.2	Security Enhancements	31
12.3	Portability and Scalability	31
12.4	Documentation and Testing	31
12.5	CI/CD Integration (Optional)	31
13	Conclusion	32

1 Introduction

The **BICA Backup Solution** is a Docker-based system designed to automate PostgreSQL database backups for the **BICA application**. Key features include:

- Automatic PostgreSQL backups twice per day
- Backups are **compressed** and optionally **encrypted**
- Configurable **retention period** for old backups
- Runs entirely inside a **Docker container** and resumes after server reboots
- Logs all backup operations to `/var/log/backup.log`

1.1 Problem Approach

When analyzing the problem, the main challenge was the need to:

- Create a reliable backup process that could survive server reboots
- Make the solution easily **portable** to any Linux server, as described in the statement
- Provide an environment for **local testing** before production deployment

To address these requirements, I designed a **Docker Compose environment** with two services:

1. **postgres-db** – A PostgreSQL 15 container to simulate the production database
2. **bica-backup** – A backup container running **cron** to perform scheduled backups

This approach allows:

- Local testing of the entire backup flow, including encryption and retention
- Easy deployment to a Linux server with minimal configuration changes
- Clear separation between the database and backup logic, following good DevOps practices

2 Project Structure

The repository is organized as follows:

```
backup.sh          # Main backup script
crontab.txt        # Cron schedule for automated backups
docker-compose.yml # Defines PostgreSQL and backup services
Dockerfile         # Builds the backup container image
entrypoint.sh      # Initializes cron and exports env variables
README.md          # Documentation and usage instructions
backups/           # Local folder where backups are stored
pgdata/            # PostgreSQL data directory (persistent volume)
```

This structure ensures:

- Separation of application data (pgdata/) and backups (backups/)
- Easy portability and disaster recovery
- Clean Docker workflow for building, running, and maintaining the solution

2.1 Requirements

- Docker 20+ and Docker Compose
- Host machine with at least:
 - 1 GB RAM (minimum)
 - 500 MB free disk space for backups
- Optional: GPG/OpenSSL knowledge for manual decryption

3 Backup Bash Script: backup.sh

The backup.sh script automates the database backup workflow for the BICA application. It performs all necessary steps to safely dump, compress, and optionally encrypt PostgreSQL backups, while also handling retention policies.

3.1 Workflow Steps

1. **Wait for PostgreSQL readiness** Uses pg_isready to ensure that the database container is online before attempting a backup.
2. **Create a database dump** Runs pg_dump to export the full content of the database to a temporary SQL file.
3. **Compress the backup** Packages the SQL dump into a timestamped .tar.gz archive for storage efficiency.
4. **Optional AES-256-CBC encryption** If ENCRYPT=true and ENCRYPT_PASS is set, the archive is encrypted using openssl with PBKDF2 key derivation for secure backups.
5. **Retention policy cleanup** At 03:00 UTC, the script removes any backups older than RETENTION_DAYS, freeing disk space.

3.2 Script Code

```
1  #!/bin/bash
2  set -e
3
4  # Environment Variables
5  DB_HOST=${DB_HOST:-postgres-db}
6  DB_PORT=${DB_PORT:-5432}
7  DB_USER=${DB_USER:?DB_USER not set}
8  DB_PASSWORD=${DB_PASSWORD:?DB_PASSWORD not set}
9  DB_NAME=${DB_NAME:?DB_NAME not set}
10 BACKUP_DIR=${BACKUP_DIR:-/mnt/backups}
11 RETENTION_DAYS=${RETENTION_DAYS:-7}
12 ENCRYPT=${ENCRYPT:-false}
13 ENCRYPT_PASS=${ENCRYPT_PASS:-""}
14
15 export PGPASSWORD="$DB_PASSWORD"
16
17 TIMESTAMP=$(date +"%Y-%m-%d_%H%M")
18 CURRENT_TIME=$(date +%H%M)
19 BACKUP_FILE="$BACKUP_DIR/bica-backup-${TIMESTAMP}.tar.gz"
20
21 echo "Now is $(date +"%H:%M"), creating backup..."
22
23 # Clean old backups at 03:00
24 if [[ "$CURRENT_TIME" == "0300" ]]; then
25     echo "[${TIMESTAMP}] Cleaning backups older than $RETENTION_DAYS days..."
26     find "$BACKUP_DIR" -type f -mtime +$RETENTION_DAYS -exec rm {} \;
27 fi
28
```

```

29 # Wait for PostgreSQL
30 until pg_isready -h "$DB_HOST" -p "$DB_PORT" -U "$DB_USER"; do
31     echo "Waiting for PostgreSQL in $DB_HOST:$DB_PORT..."
32     sleep 3
33 done
34
35 # Create dump
36 pg_dump -h "$DB_HOST" -p "$DB_PORT" -U "$DB_USER" "$DB_NAME" > /tmp/db_backup.sql
37
38 # Compress
39 tar -czf "$BACKUP_FILE" -C /tmp db_backup.sql
40 rm /tmp/db_backup.sql
41
42 # Encrypt (optional)
43 if [ "$ENCRYPT" = "true" ] && [ -n "$ENCRYPT_PASS" ]; then
44     openssl enc -aes-256-cbc -pbkdf2 -salt \
45         -in "$BACKUP_FILE" -out "${BACKUP_FILE}.enc" -k "$ENCRYPT_PASS"
46     rm "$BACKUP_FILE"
47     BACKUP_FILE="${BACKUP_FILE}.enc"
48 fi
49
50 echo "Backup created: $BACKUP_FILE"

```

3.3 Environment Variables

Variable	Default	Description
DB_HOST	postgres-db	Hostname of the PostgreSQL container
DB_PORT	5432	PostgreSQL listening port
DB_USER	Required	Username for database access
DB_PASSWORD	Required	Password for database access
DB_NAME	Required	Name of the database to back up
BACKUP_DIR	/mnt/backups	Directory where backups are stored
RETENTION_DAYS	7	Days to retain backups before automatic deletion
ENCRYPT	false	Enables backup encryption (true/false)
ENCRYPT_PASS	""	Encryption key for openssl if encryption is enabled

3.4 Backup Naming Convention

```
1 bica-backup-YYYY-MM-DD_HHMM.tar.gz
2 bica-backup-YYYY-MM-DD_HHMM.tar.gz.enc # if encrypted
```

This format guarantees:

- Chronological ordering of backups.
- Easy identification of encrypted vs non-encrypted backups.
- Simplicity when automating retention or restore scripts.

4 Cron Scheduling: crontab.txt

Automated backups are executed via **cron** inside the backup container. The **crontab.txt** defines two schedules:

```
1 # Daily backups
2 0 3 * * * . /etc/environment; /backup.sh >> /var/log/backup.log 2>&1
3 0 14 * * * . /etc/environment; /backup.sh >> /var/log/backup.log 2>&1
```

4.1 Explanation

1. **03:00 UTC** → Performs a full backup and triggers the retention cleanup.
2. **14:00 UTC** → Performs a secondary daily backup without deleting old files.

4.2 Key Points

- `. /etc/environment` → Loads all environment variables for cron jobs, since cron does not inherit Docker container variables by default.
- `» /var/log/backup.log 2 &1` → Redirects both standard output and errors to the backup log for easy troubleshooting.
- Logs can be viewed live with:

```
1 docker exec -it bica-backup tail -f /var/log/backup.log
```

- This approach ensures that the container runs autonomously, without manual intervention, and keeps an auditable log of each operation.

5 Docker Compose Setup: `docker-compose.yml`

The `docker-compose.yml` file defines and orchestrates the services required for the BICA backup solution. It ensures that the PostgreSQL database and the backup container run together with proper networking, environment variables, and volume persistence.

5.1 Services Overview

1. **postgres-db** → Runs the PostgreSQL 15 database instance.
2. **bica-backup** → Runs the backup container that performs automated backups via cron, with optional encryption and retention management.

5.2 `docker-compose.yml` File

```
1  services:
2    postgres-db:
3      image: postgres:15
4      container_name: postgres-db
5      environment:
6        POSTGRES_USER: myuser
7        POSTGRES_PASSWORD: mypass
8        POSTGRES_DB: mydatabase
9      ports:
10       - "5432:5432"
11      volumes:
12       - ./pgdata:/var/lib/postgresql/data
13      networks:
14       - bica-net
15
16    bica-backup:
17      build: .
18      container_name: bica-backup
19      restart: unless-stopped
20      environment:
21        DB_HOST: postgres-db
22        DB_PORT: 5432
23        DB_USER: myuser
24        DB_PASSWORD: mypass
25        DB_NAME: mydatabase
26        BACKUP_DIR: /mnt/backups
27        RETENTION_DAYS: 7
28        ENCRYPT: "true"
29        ENCRYPT_PASS: "MySecretKey"
30      volumes:
31       - ./backups:/mnt/backups
32      depends_on:
33       - postgres-db
34      networks:
35       - bica-net
36
37  networks:
38    bica-net:
39    driver: bridge
```

5.3 Explanation of Each Step

1. Service: postgres-db

- **image:** `postgres:15` → Official PostgreSQL 15 image from Docker Hub.
- **container_name:** `postgres-db` → Assigns a fixed name for easier linking.
- **Environment Variables:**
 - `POSTGRES_USER` → Database admin username.
 - `POSTGRES_PASSWORD` → Secure password for the database.
 - `POSTGRES_DB` → Name of the initial database created on startup.
- **Ports:**
 - `"5432:5432"` exposes PostgreSQL to the host for local access and for the backup container.
- **Volumes:**
 - `./pgdata:/var/lib/postgresql/data` persists database files on the host machine, ensuring data survives container restarts or rebuilds.
- **Networks:**
 - `bica-net` isolates services in a private bridge network for security and easy communication.

2. Service: bica-backup

- **build:** `.` → Builds the Docker image using the `Dockerfile` in the current directory.
- **container_name:** `bica-backup` → Assigns a fixed name for easy management.
- **restart:** `unless-stopped` → Ensures the backup container restarts automatically after reboots or failures.
- **Environment Variables** (used by `backup.sh` and `cron`):
 - `DB_HOST` → Name of the database container (resolves via Docker network DNS).
 - `DB_PORT` → PostgreSQL port (default 5432).
 - `DB_USER` / `DB_PASSWORD` → Credentials for database connection.
 - `DB_NAME` → Database to back up.
 - `BACKUP_DIR` → Directory inside the container where backups are stored.
 - `RETENTION_DAYS` → Number of days before old backups are automatically deleted.
 - `ENCRYPT` → Enables optional AES-256 encryption if set to `"true"`.
 - `ENCRYPT_PASS` → Encryption key used by `openssl` for secure backups.
- **Volumes:**
 - `./backups:/mnt/backups` → Maps a local directory to persist backup files. This allows easy access to backups outside the container.

- **Dependencies:**

- `depends_on: postgres-db` → Ensures the database container is started before the backup container attempts a connection.

- **Networks:**

- Joins `bica-net`, allowing communication with the database via its service name.

3. Network Definition: `bica-net`

- `driver: bridge` → Creates an isolated bridge network for inter-container communication.
- Provides DNS-based service discovery so `bica-backup` can connect to `postgres-db` using `DB_HOST=postgres-db`.

5.4 Benefits of Using Docker Compose

- Simplifies multi-container orchestration with a single command (`docker compose up -d`).
- Ensures network isolation and environment consistency between services.
- Enables persistence of both database and backup data on the host machine.
- Facilitates testing, deployment, and maintenance with minimal manual setup.

6 Dockerfile

The Dockerfile defines the backup container, which installs PostgreSQL client tools, configures cron, and runs the `backup.sh` script automatically.

```
1 FROM debian:bookworm
2
3 # Avoid interactive prompts from apt
4 ENV DEBIAN_FRONTEND=noninteractive
5
6 # Set timezone (default: UTC)
7 ENV TZ=UTC
8     #Europe/Lisbon
9 # Install required packages
10 RUN apt-get update && apt-get install -y \
11     tzdata \
12     postgresql-client-15 \
13     cron \
14     bash \
15     coreutils \
16     tar \
17     gzip \
18     openssl \
19     && ln -snf /usr/share/zoneinfo/$TZ /etc/localtime \
20     && echo $TZ > /etc/timezone \
21     && rm -rf /var/lib/apt/lists/*
22
23 # Copy scripts and cron configuration
24 COPY backup.sh /backup.sh
25 COPY entrypoint.sh /entrypoint.sh
26 COPY crontab.txt /etc/cron.d/backup-cron
27
28 # Set correct permissions
29 RUN chmod +x /backup.sh /entrypoint.sh \
30     && chmod 0644 /etc/cron.d/backup-cron \
31     && crontab /etc/cron.d/backup-cron
32
33 # Run cron in foreground to keep the container alive
34 ENTRYPOINT ["/entrypoint.sh"]
35
```

6.1 Dockerfile Explanation

The Dockerfile is responsible for building the Docker image that automates the PostgreSQL backup process. It installs all required packages, configures cron, and ensures that scheduled backups are executed properly inside the container.

6.2 Explanation of Each Step

1. Base Image Selection

- FROM `debian:bookworm`
- A lightweight, stable Debian-based image is used as the base for compatibility with PostgreSQL client tools and Linux cron.

2. Non-interactive Package Installation

- `ENV DEBIAN_FRONTEND=noninteractive`
- Prevents interactive prompts during `apt` installations, which is necessary for automated Docker builds.

3. Timezone Configuration

- `ENV TZ=UTC`
- Ensures all cron jobs and log timestamps follow the intended timezone.

4. Installing Required Packages

- `postgresql-client-15`: Required for the `pg_dump` utility to export database backups.
- `cron`: Manages scheduled tasks to trigger automatic backups.
- `bash`, `tar`, `gzip`, `coreutils`, `openssl`: Essential for scripting, file compression, and optional encryption.
- After installation, `rm -rf /var/lib/apt/lists/*` cleans the package cache to keep the image size minimal.

5. Copying Scripts and Cron Configuration

- `COPY backup.sh /backup.sh`
- `COPY entrypoint.sh /entrypoint.sh`
- `COPY crontab.txt /etc/cron.d/backup-cron`
- These files contain the main backup logic and the cron schedule definition.

6. Permissions and Cron Registration

- `chmod +x /backup.sh /entrypoint.sh` makes the scripts executable.
- `chmod 0644 /etc/cron.d/backup-cron` ensures proper cron permissions.
- `crontab /etc/cron.d/backup-cron` registers the cron job in the container.

7. Container Entrypoint

- `ENTRYPOINT ["/entrypoint.sh"]`
- When the container starts, `entrypoint.sh` runs automatically.
- This script:
 - (a) Exports environment variables for cron jobs.
 - (b) Creates the log file if missing.
 - (c) Starts `cron` in the foreground to keep the container alive.

6.3 Key Advantages

- Fully automated PostgreSQL backups with optional compression and encryption.
- Lightweight and portable container, ready for any server with Docker installed.
- Cron-based scheduling with proper environment variable handling.
- Log file persistence for easier debugging and auditing of backup operations.

6.4 Deployment

Build and start containers:

```
1 docker compose up -d --build
```

Backups will appear in `./backups`. Stop containers without deleting volumes:

```
1 docker compose down
```


7 Entrypoint Script: entrypoint.sh

The `entrypoint.sh` script initializes the container environment and ensures that the `cron` service runs in the foreground. This guarantees that the container remains active and automatically executes the scheduled backups.

7.1 Script Code

```
1  #!/bin/bash
2
3  # Export container environment variables for cron
4  printenv | grep -E
   ↪  '^(DB_HOST|DB_PORT|DB_USER|DB_PASSWORD|DB_NAME|BACKUP_DIR|RETENTION_DAYS|ENCRYPT|ENCRYPT_PASS) '
   ↪  > /etc/environment
5
6  # Create log file if it does not exist
7  touch /var/log/backup.log
8
9  # Start cron in foreground to keep the container running
10 exec cron -f
```

7.2 Explanation of Each Step

1. Export container environment variables for cron

- Cron jobs do not automatically inherit Docker environment variables.
- This command uses `printenv` to list all environment variables and filters only the ones required for the backup solution: `DB_HOST`, `DB_PORT`, `DB_USER`, `DB_PASSWORD`, `DB_NAME`, `BACKUP_DIR`, `RETENTION_DAYS`, `ENCRYPT`, and `ENCRYPT_PASS`.
- These variables are written to `/etc/environment` so they are available to all cron executions.

2. Create log file if it does not exist

- Ensures that `/var/log/backup.log` exists before starting `cron`.
- All backup operations and errors will be logged in this file.

3. Start cron in foreground

- By default, `cron` runs as a background daemon.
- Docker containers exit when the main process finishes.
- `exec cron -f` keeps `cron` in the foreground, preventing the container from stopping and allowing scheduled backups to run continuously.

7.3 Benefits of This Approach

- Environment variables are consistently available to all scheduled cron jobs.
- The container maintains a single log file for debugging and monitoring.
- Running `cron` in foreground ensures the container remains alive for continuous operation.

8 Verification and Results

This section demonstrates the testing process of the **BICA Backup Solution**, showing backup operations, encryption and decryption, log verification, and automatic cleanup.

8.1 Creating the Containers

Figure 1 shows the process of building and running the Docker containers for the **BICA Backup Solution**. The logs confirm that the containers were successfully created and are running as expected.

```
C:\WINDOWS\system32>cd C:\Users\gast\Desktop
C:\Users\gast\Desktop>cd bica-backup
C:\Users\gast\Desktop\bica-backup>docker compose up -d --build
[*] Building 4/4 (14/14) FINISHED
=> [internal] load local bake definitions                                0.0s
=> => reading from stdin 3978                                          0.0s
=> [internal] load build definition from Dockerfile                    0.0s
=> => transferring dockerfile: 919B                                     0.0s
=> [internal] load metadata for docker.io/library/debian:bookworm      1.1s
=> [auth] library/debian:pull token for registry-1.docker.io          0.0s
=> [internal] load .dockerignore                                       0.0s
=> => transferring context: 2B                                          0.0s
=> [1/6] FROM docker.io/library/debian:bookworm@sha256:b6507e340c43553136f5078284c8c68d86ec8262b1724dde73c325e8d 0.0s
=> => resolve docker.io/library/debian:bookworm@sha256:b6507e340c43553136f5078284c8c68d86ec8262b1724dde73c325e8d 0.0s
=> [internal] load build context                                       0.0s
=> => transferring context: 2.17kB                                      0.0s
=> CACHED [2/6] RUN apt-get update && apt-get install -y tzdata postgresql-client-15 cron bash 0.0s
=> [3/6] COPY backup.sh /backup.sh                                     0.0s
=> [4/6] COPY entrypoint.sh /entrypoint.sh                             0.1s
=> [5/6] COPY crontab.txt /etc/cron.d/backup-cron                      0.1s
=> [6/6] RUN chmod +x /backup.sh /entrypoint.sh && chmod 0644 /etc/cron.d/backup-cron && crontab /etc/cr 0.5s
=> exporting to image                                                 2.1s
=> => exporting layers                                                  0.3s
=> => exporting manifest sha256:39ef12ab0827f45b64962aa7a2ec7888f065bd76dbed1e47e462027a2ff598dc 0.0s
=> => exporting config sha256:fe49bedd0d49418823a27e9d8d2ff3e01ae43a7707a18c0a9126011901f1024e 0.0s
=> => exporting attestation manifest sha256:e05efd0a12cf2d0c80ec311e2d7cf48bf95d3b750159cdd95ec6070e08730db 0.0s
=> => exporting manifest list sha256:2fa01356336cbe8d3d130a86989b19ed88239f8ba5dc6d3bc95b24f646fabf56 0.0s
=> => naming to docker.io/library/bica-backup-bica-backup:latest      0.0s
=> => unpacking to docker.io/library/bica-backup-bica-backup:latest    1.7s
=> resolving provenance for metadata file                             0.0s
[*] Running 4/4
bica-backup Built 0.0s
Network bica-backup_bica-net Created 0.1s
Container postgres-db Started 1.1s
Container bica-backup Started 1.1s
C:\Users\gast\Desktop\bica-backup>docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
b2cf519833089  bica-backup-bica-backup             "/entrypoint.sh"        4 seconds ago Up 3 seconds  0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp  bica-backup
639a75991399  postgres:15                         "docker-entrypoint.s..." 4 seconds ago Up 4 seconds  5432/tcp  postgres-db
```

Figure 1: Docker containers created and running successfully

8.2 Backup without Encryption

Figure 2 shows the result of a backup operation performed without encryption. The output log confirms that the backup was successfully created and stored in the backup directory.

```
C:\Users\gast\Desktop\bica-backup>docker exec -it bica-backup tail -f /var/log/backup.log
Now is 19:50, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1950.tar.gz
Now is 19:51, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1951.tar.gz
```

Figure 2: Successful backup without encryption (log output)

Figure 3 shows the backup file created in the target folder, confirming that the compressed archive was correctly generated.

```

  BICA-BACKUP
  └─ backups
      ├── bica-backup-2025-08-01_1954.tar.gz
      ├── bica-backup-2025-08-01_1955.tar.gz
      └─ backup.sh
  └─ crontab.txt
  └─ docker-compose.yml
  └─ Dockerfile
  └─ entrypoint.sh
  └─ README.md

=> => unpacking to docker.io/library/bica-backup-bica-backup:latest
=> resolving provenance for metadata file
[+] Running 4/4
   0 bica-backup          Built
   1 Network bica-backup_bica-net Created
   2 Container postgres-db Started
   3 Container bica-backup Started

C:\Users\gast\Desktop\bica-backup>docker exec -it bica-backup tail -f /var/log/backup.log
Now is 19:55, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1955.tar.gz
```

Figure 3: Backup file visible in the backup directory

8.3 Automatic Cleanup Test

The automatic retention policy was validated by setting `RETENTION_DAYS=0`. As shown in Figure 4, old backups were automatically removed by the cron job.

```

EXPLORER
└─ BICA-BACKUP
    └─ backups
        ├── bica-backup-2025-08-01_2000.tar.gz
        ├── backup.sh
        ├── crontab.txt
        ├── docker-compose.yml
        ├── Dockerfile
        ├── entrypoint.sh
        └─ README.md

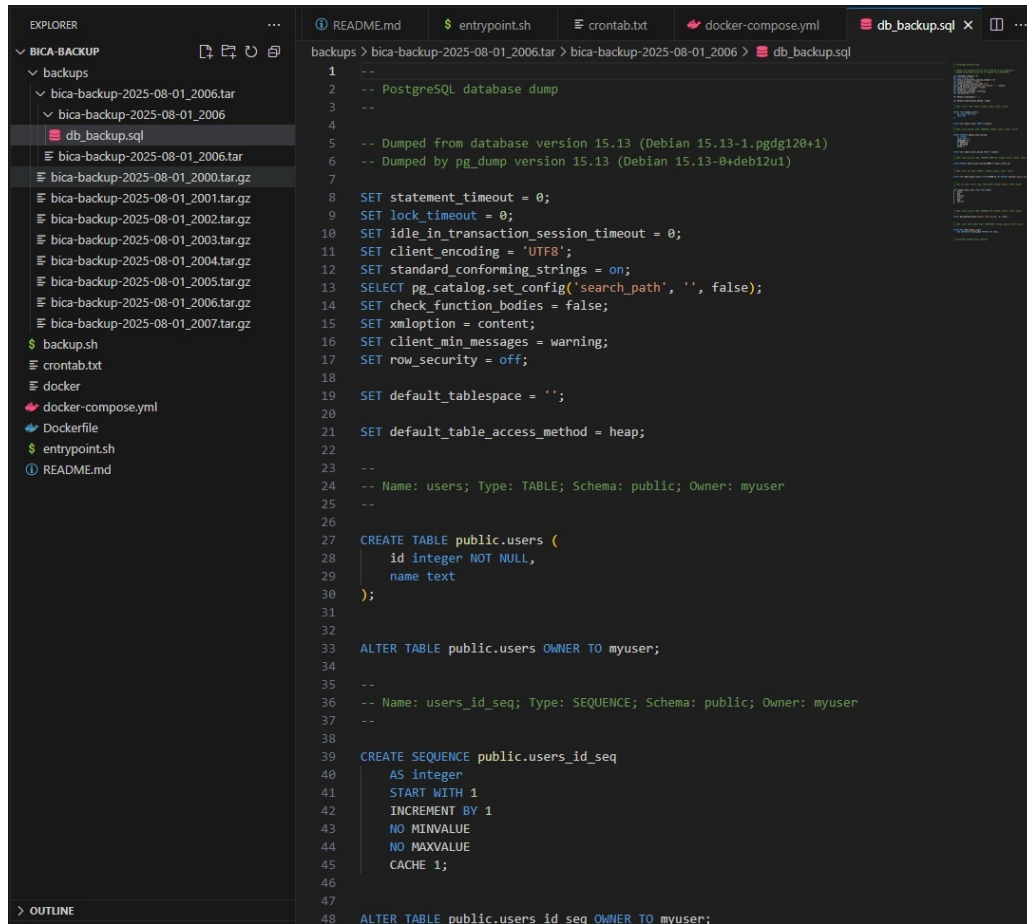
Administrator: Linha de comandos - docker exec -it bica-backup tail -f /var/log/backup.log
C:\Users\gast\Desktop\bica-backup>docker exec -it bica-backup tail
Now is 19:55, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1955.tar.gz
Now is 19:56, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1956.tar.gz
Now is 19:57, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1957.tar.gz
Now is 19:58, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1958.tar.gz
Now is 19:59, creating backup...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_1959.tar.gz
Now is 20:00, creating backup...
[2025-08-01_2000] Clean backups with more than 0 days...
postgres-db:5432 - accepting connections
Backup created: /mnt/backups/bica-backup-2025-08-01_2000.tar.gz
```

Figure 4: Automatic cleanup of old backups

8.4 Backup Content Verification

To confirm that the backup archive contains the expected PostgreSQL dump file, the backup was extracted and inspected.

including the `pg_dump` file.



```
1  --
2  -- PostgreSQL database dump
3  --
4
5  -- Dumped from database version 15.13 (Debian 15.13-1.pgdg120+1)
6  -- Dumped by pg_dump version 15.13 (Debian 15.13-0+deb12u1)
7
8  SET statement_timeout = 0;
9  SET lock_timeout = 0;
10 SET idle_in_transaction_session_timeout = 0;
11 SET client_encoding = 'UTF8';
12 SET standard_conforming_strings = on;
13 SELECT pg_catalog.set_config('search_path', '', false);
14 SET check_function_bodies = false;
15 SET xmloption = content;
16 SET client_min_messages = warning;
17 SET row_security = off;
18
19 SET default_tablespace = '';
20
21 SET default_table_access_method = heap;
22
23 --
24 -- Name: users; Type: TABLE; Schema: public; Owner: myuser
25 --
26
27 CREATE TABLE public.users (
28     id integer NOT NULL,
29     name text
30 );
31
32
33 ALTER TABLE public.users OWNER TO myuser;
34
35 --
36 -- Name: users_id_seq; Type: SEQUENCE; Schema: public; Owner: myuser
37 --
38
39 CREATE SEQUENCE public.users_id_seq
40     AS integer
41     START WITH 1
42     INCREMENT BY 1
43     NO MINVALUE
44     NO MAXVALUE
45     CACHE 1;
46
47
48 ALTER TABLE public.users id seq OWNER TO myuser;
```

Figure 5: Contents of the backup archive (showing `pg_dump`) part1

```
46
47
48 ALTER TABLE public.users_id_seq OWNER TO myuser;
49
50 --
51 -- Name: users_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: myuser
52 --
53
54 ALTER SEQUENCE public.users_id_seq OWNED BY public.users.id;
55
56 --
57 -- Name: users id; Type: DEFAULT; Schema: public; Owner: myuser
58 --
59 --
60
61 ALTER TABLE ONLY public.users ALTER COLUMN id SET DEFAULT nextval('public.users_
62
63
64 --
65 -- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: myuser
66 --
67
68 COPY public.users (id, name) FROM stdin;
69 1 Alice
70 2 Bob
71 3 Charlie
72 4 Alice
73 5 Bob
74 6 Charlie
75 \.
76
77
78 --
79 -- Name: users_id_seq; Type: SEQUENCE SET; Schema: public; Owner: myuser
80 --
81
82 SELECT pg_catalog.setval('public.users_id_seq', 6, true);
83
84 --
85 -- Name: users users_pkey; Type: CONSTRAINT; Schema: public; Owner: myuser
86 --
87 --
88
89 ALTER TABLE ONLY public.users
90 | ADD CONSTRAINT users_pkey PRIMARY KEY (id);
91
92
93 --
```

Figure 6: Contents of the backup archive (showing pg_dump) part2

8.5 Backup with Encryption and Decryption Test

When the `ENCRYPT=true` option is enabled, backups are automatically encrypted using **AES-256** before being stored. The resulting files have the `.tar.gz.enc` extension.

Encryption Process The container logs in Figure 7 show that encrypted backups were generated successfully every minute. The last step confirms the creation of `decrypted_backup.tar.gz`, which is obtained after decryption.

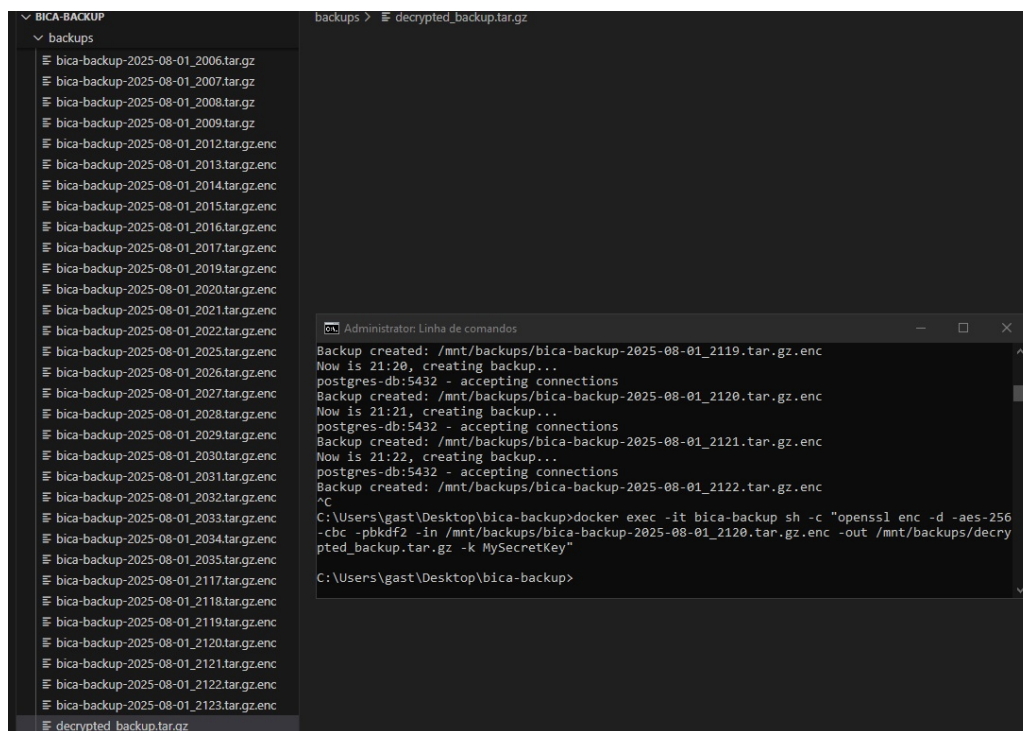


Figure 7: Encrypted backup creation and logs showing successful decryption

Decryption Process To verify the encryption, the backup was decrypted using the following command:

```
1 docker exec -it bica-backup sh -c \  
2 "openssl enc -d -aes-256-cbc -pbkdf2 \  
3 -in /mnt/backups/bica-backup-2025-08-01_2120.tar.gz.enc \  
4 -out /mnt/backups/decrypted_backup.tar.gz \  
5 -k MySecretKey"
```

The decryption consists of two steps:

1. Decrypt the `.enc` file to produce `decrypted_backup.tar.gz`.
2. Extract the `.tar.gz` file to confirm that the PostgreSQL dump is present.

Figure 8 shows the 2nd step, where the decrypted file is being extracted.

```
C:\WINDOWS\system32>cd C:\Users\gast\Desktop\bica-backup\backups
C:\Users\gast\Desktop\bica-backup\backups>rm -r tmp
'rm' is not recognized as an internal or external command,
operable program or batch file.
C:\Users\gast\Desktop\bica-backup\backups>mkdir tmp
C:\Users\gast\Desktop\bica-backup\backups>tar -xzf decrypted_backup.tar.gz -C tmp
C:\Users\gast\Desktop\bica-backup\backups>type tmp\db_backup.sql
--
-- PostgreSQL database dump
--
-- Dumped from database version 15.13 (Debian 15.13-1.pgdg120+1)
-- Dumped by pg_dump version 15.13 (Debian 15.13-0+deb12u1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: users; Type: TABLE; Schema: public; Owner: myuser
--

CREATE TABLE public.users (
    id integer NOT NULL,
    name text
);

ALTER TABLE public.users OWNER TO myuser;

--
-- Name: users_id_seq; Type: SEQUENCE; Schema: public; Owner: myuser
--

CREATE SEQUENCE public.users_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE public.users_id_seq OWNER TO myuser;

--
-- Name: users_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: myuser
```

Figure 8: Extraction of the decrypted file and part 1 of the dump generated by pg_dump

Figure 9 shows the part2 of the PostgreSQL dump generated by pg_dump.

```
ALTER TABLE public.users_id_seq OWNER TO myuser;

--
-- Name: users_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: myuser
--

ALTER SEQUENCE public.users_id_seq OWNED BY public.users.id;

--
-- Name: users id; Type: DEFAULT; Schema: public; Owner: myuser
--

ALTER TABLE ONLY public.users ALTER COLUMN id SET DEFAULT nextval('public.users_id_seq'::regclass);

--
-- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: myuser
--

COPY public.users (id, name) FROM stdin;
1      Alice
2      Bob
3      Charlie
4      Alice
5      Bob
6      Charlie
\.

--
-- Name: users_id_seq; Type: SEQUENCE SET; Schema: public; Owner: myuser
--

SELECT pg_catalog.setval('public.users_id_seq', 6, true);

--
-- Name: users users_pkey; Type: CONSTRAINT; Schema: public; Owner: myuser
--

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_pkey PRIMARY KEY (id);

--
-- PostgreSQL database dump complete
--

C:\Users\gast\Desktop\bica-backup\backups>
```

Figure 9: part2 of the dump generated by pg_dumpmd

9 Tested Workflow

To validate the **BICA Backup Solution**, the following end-to-end workflow was tested:

9.1 1. Insert Sample Data

Before testing backups, sample data was inserted into the PostgreSQL database:

```
1 docker exec -it postgres-db psql -U myuser -d mydatabase -c \  
2 "CREATE TABLE users (id SERIAL PRIMARY KEY, name TEXT, email TEXT);"\  
3 \  
4 docker exec -it postgres-db psql -U myuser -d mydatabase -c \  
5 "INSERT INTO users (name, email) VALUES\  
6 ('Alice', 'alice@example.com'),\  
7 ('Bob', 'bob@example.com');"
```

9.2 2. Trigger a Manual Backup

Run the backup script inside the container manually:

```
1 docker exec -it bica-backup /backup.sh
```

Result:

Now is 14:05, creating backup...

Backup created: /mnt/backups/bica-backup-2025-08-01_1405.tar.gz.enc

9.3 3. Verify Backup Files

The backups are stored in the mounted directory:

```
1 ls -lh backups/  
2 bica-backup-2025-08-01_0300.tar.gz.enc  
3 bica-backup-2025-08-01_1405.tar.gz.enc
```

9.4 4. Decrypt a Backup (if encrypted)

To restore or inspect a backup, first decrypt it:

```
1 docker exec -it bica-backup sh -c \  
2 "openssl enc -d -aes-256-cbc -pbkdf2 \  
3 -in /mnt/backups/bica-backup-2025-08-01_1405.tar.gz.enc \  
4 -out /mnt/backups/decrypted_backup.tar.gz \  
5 -k MySecretKey"
```

9.5 5. Extract and Restore

```

1 # Extract the SQL dump
2 tar -xzf backups/decrypted_backup.tar.gz -C /tmp
3
4 # Restore into PostgreSQL (can be a new DB)
5 docker exec -i postgres-db psql -U myuser -d mydatabase < /tmp/db_backup.sql

```

9.6 6. Verify Data After Restore

```

1 docker exec -it postgres-db psql -U myuser -d mydatabase -c "SELECT * FROM users;"
2   id | name  | email
3 -----+-----+-----
4   1 | Alice | alice@example.com
5   2 | Bob   | bob@example.com
6 (2 rows)

```

9.7 Workflow Summary

1. Data was successfully inserted into the PostgreSQL database
2. Backups were created automatically and manually
3. Encryption and decryption worked with AES-256-CBC
4. Backups were restored and verified successfully

This demonstrates the solution is fully functional, portable, and reliable.

10 Deployment Scenarios

The **BICA Backup Solution** was designed to be portable and easy to integrate into different environments. It supports two main deployment scenarios:

1. Local Linux server with an existing PostgreSQL instance
2. Docker Hub deployment, using the pre-built `bica-backup` image

10.1 Scenario 1: Local Linux Server with PostgreSQL

In this scenario, the company server already has PostgreSQL running natively. The backup container only needs network access to the database.

10.1.1 Preparation

1. Install Docker and Docker Compose:

```
1 sudo apt update && sudo apt install -y docker.io docker-compose
2 sudo systemctl enable docker --now
```

2. Copy or clone the project folder to the server:

```
1 scp -r bica-backup-solution user@server:/opt/bica-backup
2 cd /opt/bica-backup
```

3. Edit `docker-compose.yml` with the real PostgreSQL connection:

- `DB_HOST` → Server IP or hostname
- `DB_PORT` → Usually 5432
- `DB_USER` → User with `pg_dump` permissions
- `DB_PASSWORD` → Database password
- `DB_NAME` → Target database name

Example:

```
1 environment:
2   DB_HOST: 192.168.1.50
3   DB_PORT: 5432
4   DB_USER: bica_user
5   DB_PASSWORD: StrongPassword123
6   DB_NAME: bica_production
7   RETENTION_DAYS: 7
8   ENCRYPT: "true"
9   ENCRYPT_PASS: "MySecretKey"
```

10.1.2 Start the Backup Container

```
1 docker compose up -d
```

The container will:

1. Connect to the existing PostgreSQL database
 2. Perform automatic backups at 03:00 and 14:00 UTC
 3. Store backups in the `./backups` folder (or mounted volume)
 4. Encrypt backups if `ENCRYPT=true`
-

10.2 Scenario 2: Deployment via Docker Hub

If the image `bica-backup` is published to Docker Hub (e.g. `gast0xc/bica-backup`), the company can integrate it into any environment without transferring the project files.

10.2.1 Steps

1. Create a working directory and a minimal `docker-compose.yml`:

```
1 services:
2   bica-backup:
3     image: gast0xc/bica-backup:latest
4     environment:
5       DB_HOST: 192.168.1.50
6       DB_PORT: 5432
7       DB_USER: bica_user
8       DB_PASSWORD: StrongPassword123
9       DB_NAME: bica_production
10      RETENTION_DAYS: 7
11      ENCRYPT: "true"
12      ENCRYPT_PASS: "MySecretKey"
13    volumes:
14      - ./backups:/mnt/backups
15      - ./backup.log:/var/log/backup.log
16    restart: always
```

2. Start the service:

```
1 docker compose up -d
```

10.2.2 Advantages

- No need to copy project files
 - Quick and reproducible deployment
 - Only requires correct environment variables and mounted volumes
-

10.3 Summary of Integration

In both scenarios:

1. Define the PostgreSQL connection variables
2. Mount a folder for backups (`./backups:/mnt/backups`)
3. Run `docker compose up -d`

The backup solution is now:

- **Self-contained** – Includes cron, `pg_dump`, retention logic, and optional encryption
- **Resilient** – Survives server or container restarts
- **Portable** – Works both locally and from Docker Hub

11 Important Commands

This section summarizes the key commands used to build, run, test, and maintain the **BICA Backup Solution**. Each command is presented with its description for clarity.

11.1 Docker and Docker Compose

Build the Docker image:

```
1 docker compose build
```

Start all services in detached mode:

```
1 docker compose up -d
```

Stop and remove all services:

```
1 docker compose down
```

11.2 Logs and Monitoring

View live logs from the backup container:

```
1 docker exec -it bica-backup tail -f /var/log/backup.log
```

List all existing backups:

```
1 docker exec -it bica-backup ls /mnt/backups
```

Open an interactive shell in the backup container:

```
1 docker exec -it bica-backup sh
```

11.3 Backup Decryption

Decrypt an encrypted backup (.tar.gz.enc) with AES-256-CBC:

```
1 docker exec -it bica-backup sh -c "openssl enc -d -aes-256-cbc -pbkdf2 \  
2 -in /mnt/backups/bica-backup-2025-08-01_2120.tar.gz.enc \  
3 -out /mnt/backups/decrypted_backup.tar.gz -k MySecretKey"
```

11.4 Database Testing Commands

Open PostgreSQL client in the database container:

```
1 docker exec -it postgres-db psql -U myuser -d mydatabase
```

Create a test table:

```
1 CREATE TABLE test(id SERIAL PRIMARY KEY, name TEXT);
```

Insert sample data for validation:

```
1 INSERT INTO test(name) VALUES ('Backup_Check');
```

Verify the inserted data:

```
1 SELECT * FROM test;
```

11.5 Maintenance and Cleanup

List Docker volumes (check persistent data):

```
1 docker volume ls
```

Clean up unused Docker containers, images, and networks:

```
1 docker system prune -af
```

12 Future Improvements & Scalability

While the current **BICA Backup Solution** fully meets the requirements of the exercise, there are several opportunities to enhance its robustness, security, and scalability for a production environment:

12.1 Logging and Monitoring

- **Log rotation:** Prevent `/var/log/backup.log` from growing indefinitely.
- **Backup notifications:** Send alerts via email, Slack, or Teams when backups succeed or fail.
- **Health checks:** Add Docker `HEALTHCHECK` to verify that recent backups are available.

12.2 Security Enhancements

- Store `ENCRYPT_PASS` in a `.env` file or **Docker Secrets** instead of hardcoding in `docker-compose.yml`.
- Run the container with a **non-root user** for better security isolation.
- Generate **SHA256 checksums** for each backup to ensure file integrity.

12.3 Portability and Scalability

- Support **multiple databases** by iterating over a list of `DB_NAME`.
- Make the **backup schedule configurable** via an environment variable (e.g., `CRON_SCHEDULE`).
- Optionally **sync backups to cloud storage** (S3, GCP, Azure) for disaster recovery using `rclone` or `aws cli`.

12.4 Documentation and Testing

- Add a detailed **Restore Guide** with a visual flow from Backup → Decrypt → Restore.
- Provide a **test script** that creates dummy data, performs a backup, and validates a full restore.

12.5 CI/CD Integration (Optional)

- Automate **image builds and updates** via GitHub Actions or GitLab CI.
- Run **scheduled tests** to validate backup and restore integrity.
- Publish verified Docker images to a private or public **Docker Hub repository**.

These improvements would allow the solution to evolve into a production-grade backup system that is secure, easily maintainable, and scalable for multiple environments.

13 Conclusion

The **BICA Backup Solution** successfully addresses the challenge of automating PostgreSQL database backups in a reliable, portable, and self-contained way using Docker.

Key Achievements

- **Automation:** Backups are scheduled twice per day with optional AES-256 encryption
- **Resilience:** Cron jobs and retention policies automatically resume after container or server restarts
- **Portability:** The entire solution runs in a single Docker container, deployable on any Linux server

Scalability and Future Vision

This project is designed with scalability in mind:

- **Multi-database Support:** The solution can be easily extended to back up multiple PostgreSQL instances by deploying additional containers with different environment configurations.
- **Cloud Integration:** Backups can be uploaded to cloud storage (AWS S3, GCP, or Azure) for disaster recovery and long-term retention.
- **CI/CD Integration:** By publishing the container to a private or public Docker registry, companies can seamlessly integrate this solution into their DevOps pipelines.
- **Monitoring and Alerts:** Future versions could add health checks, Prometheus metrics, and Slack/email notifications for real-time monitoring.

Final Remarks

The solution not only fulfills the requirements of the technical exercise but also demonstrates how a clean and modular design allows easy deployment, scaling, and future evolution towards a fully production-ready DevOps backup strategy.