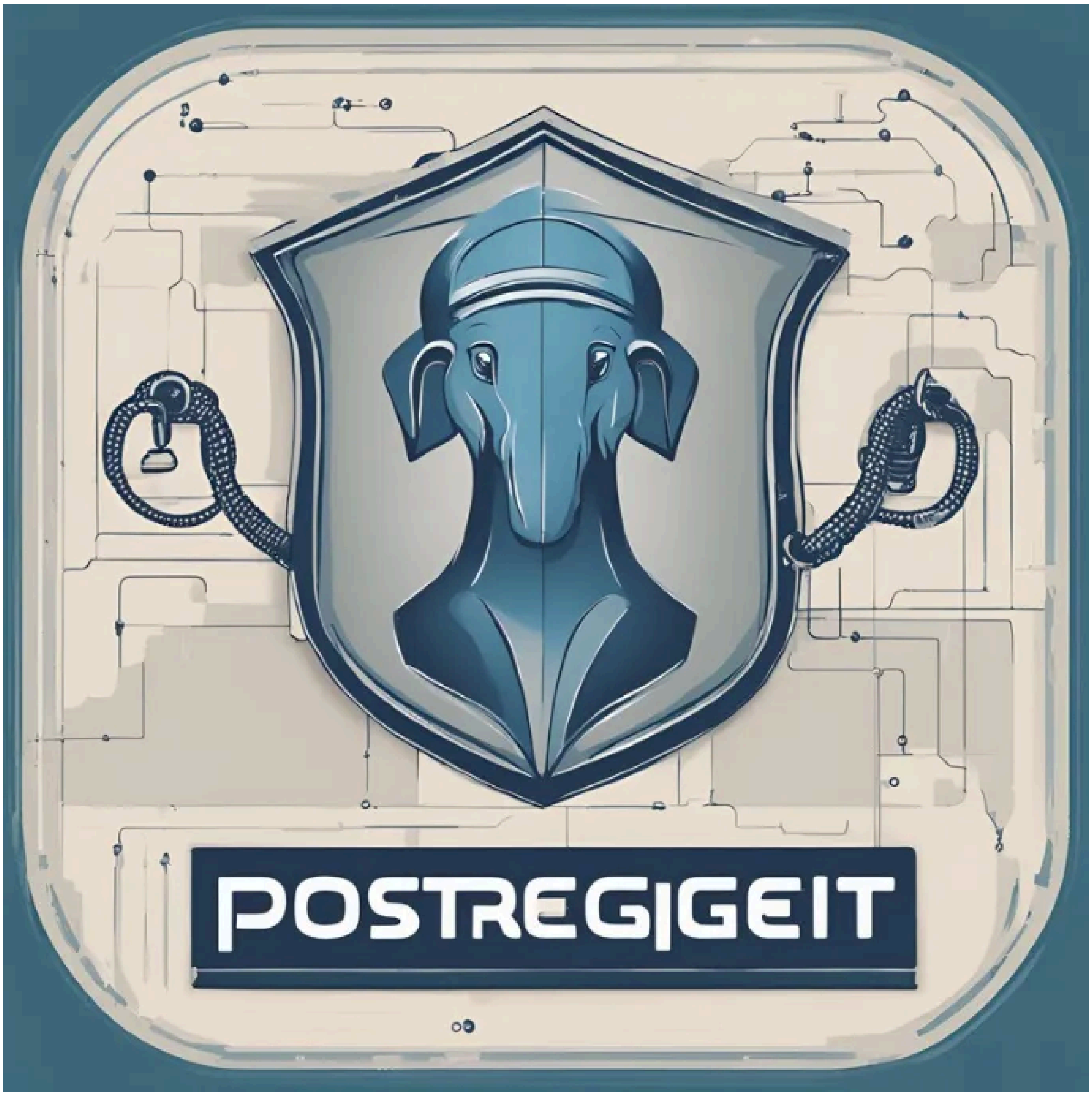
Enhancing Data Security in PostgreSQL: Using pgcrypto and Anonymizer Extensions

Oz · Following

4 min read · Jun 24, 2024

PostgreSQL is a widely-used and robust database management system, renowned for its advanced data handling capabilities. In the current era, where data security is crucial, PostgreSQL offers several extensions designed to enhance data security and privacy. This article delves into the use of the `pgcrypto` and `anon` extensions to implement encryption and data masking, thereby boosting data security.



Setting Up pgcrypto

The `pgcrypto` extension provides cryptographic functions for PostgreSQL, allowing for secure data encryption and hashing. Here's a step-by-step guide to installing and enabling `pgcrypto`:

Check Available Extensions: First, verify if `pgcrypto` is available in your PostgreSQL installation:

```
SELECT * FROM pg_available_extensions WHERE name='pgcrypto';

--You should see an output similar to this:
```

```
name     | default_version | installed_version |          comment
----------+-----------------+-------------------+-------------------------
pgcrypto | 1.3             |                   | cryptographic functions
```

Create pgcrypto Extension: Enable `pgcrypto` in your database:

```
CREATE EXTENSION pgcrypto;
```

Verify Installation: Confirm that `pgcrypto` is installed:

```
SELECT * FROM pg_available_extensions WHERE name='pgcrypto';

-- The output should indicate that pgcrypto is installed:

name     | default_version | installed_version |          comment
----------+-----------------+-------------------+-------------------------
pgcrypto | 1.3             | 1.3               | cryptographic functions
```

Setting Up Anonymizer Extension

The `anon` extension in PostgreSQL helps mask sensitive data, ensuring privacy. Here's how to set it up:

Install Required Dependencies: Install necessary packages before creating the `anon` extension:

```
dnf install ddlx_14-0.27-1PGDG.rhel9.noarch.rpm
dnf install python3-faker-13.3.3-1.el9.noarch.rpm
dnf install postgresql_anonymizer_14-1.1.0-1.rhel9.x86_64.rpm
```

Create and Configure Database: Create a new database and configure it for the `anon` extension:

```
CREATE DATABASE employee;

ALTER DATABASE employee SET session_preload_libraries = 'anon';
```

Enable Anonymizer Extension: Connect to the new database and create the `anon` extension:

```
\c employee;

CREATE EXTENSION anon CASCADE;

SELECT anon.init();
```

Implementing Data Masking

Now, let's create a sample table and apply data masking:

Create Sample Database and Table: Create a new database and a table to hold sample data:

```
\c employee;

CREATE TABLE people (id INT,firstname VARCHAR(10),lastname VARCHAR(10),phone VA

INSERT INTO people (id, firstname, lastname, phone) VALUES (1, 'Kemal', 'Oz', '
```

Create Role and Grant Permissions: Create a role for data access and grant necessary permissions:

```
CREATE ROLE hr LOGIN;

GRANT SELECT ON people TO hr;
```

Apply Data Masking: Enable dynamic masking and set up masking rules:

```
CREATE EXTENSION IF NOT EXISTS anon CASCADE;

SELECT anon.start_dynamic_masking();
```

Description: This command initializes dynamic data masking provided by the `anon` extension.

Purpose: To activate dynamic masking of data, which allows the system to apply masking rules dynamically as queries are executed.

```
SECURITY LABEL FOR anon ON ROLE hr IS 'MASKED';
```

- **Description:** This command assigns a security label to the `hr` role indicating that the role is subject to data masking rules.
- **Purpose:** To specify that the `hr` role should see masked data according to the rules defined in the `anon` extension.

```
SECURITY LABEL FOR anon ON COLUMN people.lastname IS 'MASKED WITH FUNCTION anor
```

- **Description:** This command applies a security label to the `lastname` column of the `people` table, specifying that the data should be masked using the `anon.fake_last_name()` function.
- **Purpose:** To mask the `lastname` column data with a fake last name when accessed by roles with the `MASKED` label, ensuring that sensitive information is not exposed.

```
SECURITY LABEL FOR anon ON COLUMN people.phone IS 'MASKED WITH FUNCTION anon.pa
```

- **Description:** This command applies a security label to the `phone` column of the `people` table, specifying that the data should be masked using the `anon.partial` function. The `anon.partial` function reveals only the first two and last two digits of the phone number, masking the middle digits with asterisks (`******`).
- **Purpose:** To partially mask the `phone` column data, exposing only the first and last two digits, ensuring that the sensitive parts of the phone number are hidden from unauthorized users.

Verify Data Masking: Check the data masking in action:

```
SELECT * FROM people;
```

The output should show masked data:

```
id | firstname |  lastname  |    phone
----+-----------+------------+------------
1  | Kemal     | Wintheiser | 90******78
```

Conclusion

By leveraging the `pgcrypto` and `anon` extensions, PostgreSQL users can significantly enhance their data security measures. `pgcrypto` provides robust cryptographic functions, while `anon` ensures that sensitive data is masked effectively, protecting privacy and meeting compliance requirements. These tools are essential for any organization handling sensitive information, offering peace of mind in an increasingly data-centric world. For more detailed and technical articles like this, keep following our blog on Medium. If you have any questions or need further assistance, feel free to reach out in the comments below and directly.

Technology Database Data Security Crypto Pgcrypto

Following

Written by Oz

149 Followers · 13 Following

Database Administrator 🐘

No responses yet

Gvadakte

What are your thoughts?

More from Oz

Oz

Managing Time Series Data Using TimeScaleDB on Postgres

TimescaleDB is an open-source time-series database optimized for fast ingest and complex queries, built on PostgreSQL. This guide will help...

Jul 18, 2024 125

Queries

time	Service	username	Query	Execution_time
2024-09-26 13:15:00	pmm-server-postgresql	pmm-ma...	SELECT /* agent='pgstatstatements' */ "pg_stat_statements"."userid", "pg_stat_statements"."dbid", "pg_stat_statements"."queryi...	0.0242
2024-09-26 13:19:00	pmm-server-postgresql	pmm-ma...	SELECT current_database() datname, schemaname, relname, heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit, toast_b...	0.0165
2024-09-26 13:10:48	pmm-server-postgresql	pmm-ma...	SELECT * FROM pg_stat_database_conflicts	0.0159
2024-09-26 13:18:00	pmm-server-postgresql	pmm-ma...	SELECT /* agent='pgstatstatements' */ "pg_stat_statements"."userid", "pg_stat_statements"."dbid", "pg_stat_statements"."queryi...	0.0131
2024-09-26 13:19:00	pmm-server-postgresql	pmm-ma...	SELECT /* agent='pgstatstatements' */ "pg_stat_statements"."userid", "pg_stat_statements"."dbid", "pg_stat_statements"."queryi...	0.0131
2024-09-26 13:12:00	pmm-server-postgresql	pmm-ma...	SELECT /* agent='pgstatstatements' */ "pg_stat_statements"."userid", "pg_stat_statements"."dbid", "pg_stat_statements"."queryi...	0.0126
2024-09-26 13:15:00	pmm-server-postgresql	pmm-ma...	SELECT current_database() datname, schemaname, relname, heap_blks_read, heap_blks_hit, idx_blks_read, idx_blks_hit, toast_b...	0.0113

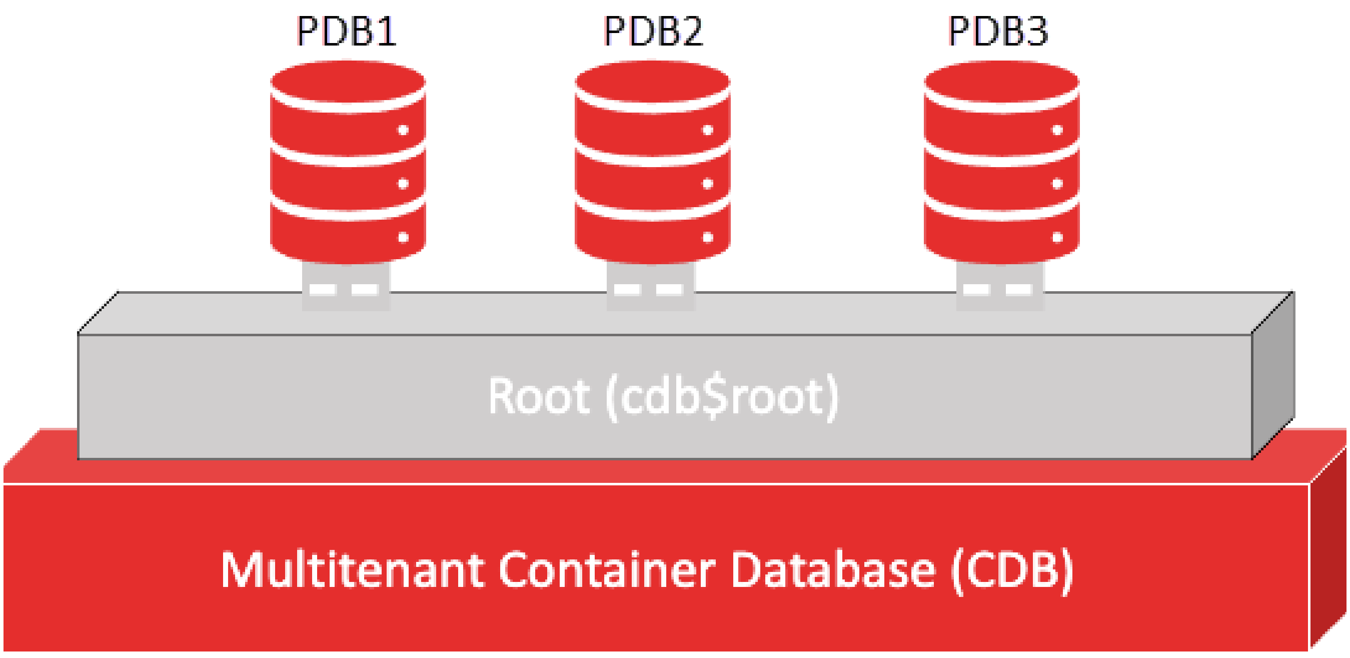
1 - 7 of 538 rows

Oz

Installing Percona Monitoring & Management (PMM) with Postgres

Introduction:

Sep 26, 2024 54 1

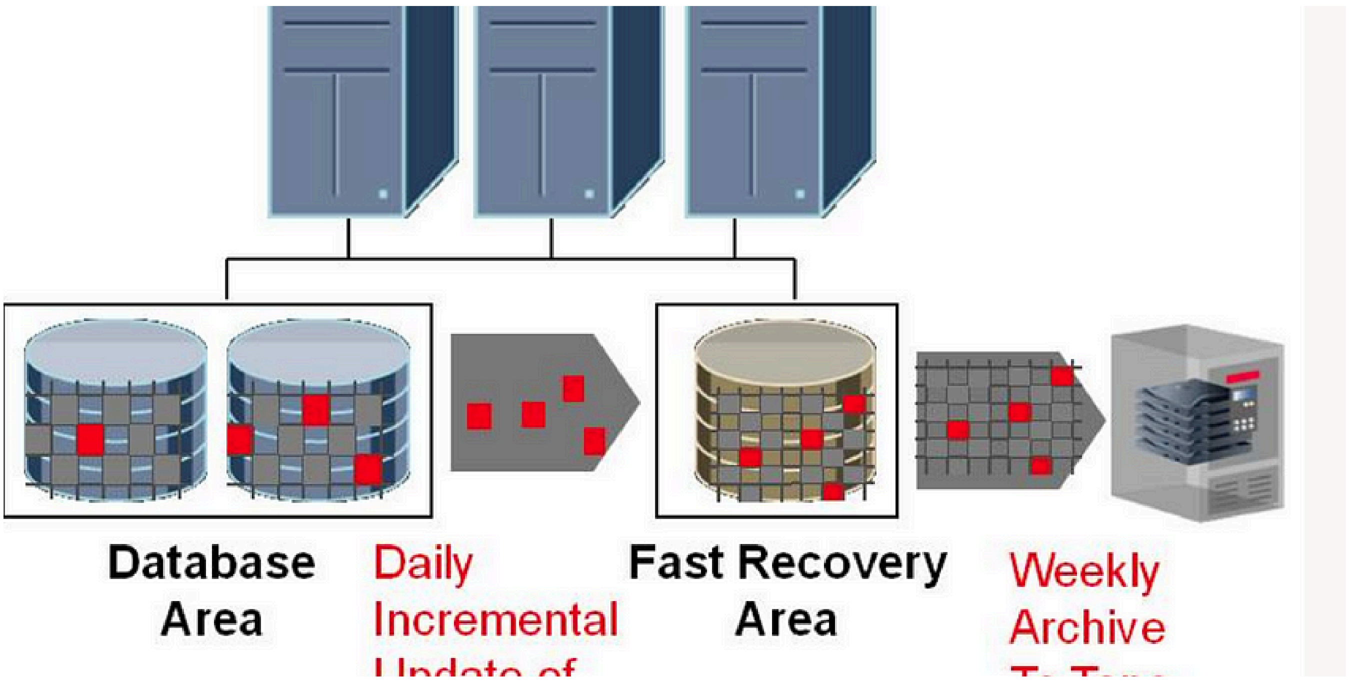


Oz

Pluggable Database Command

—————————— -————————————— - create pluggable database pdb1 admin user root identified by test123; alter pluggable database...

May 12, 2023



Oz

RMAN Backup Basic Commands

rman target / rman target sys/password@YDKTST; backup database; backup database format '/backup/path/%d_%t_%s.rman'; backup tablespace...

May 11, 2023 1

See all from Oz

Recommended from Medium

Ajaymaurya

How Often Should You Reindex Your PostgreSQL Database? A Data-Driven Approach

PostgreSQL is one of the most powerful and flexible relational databases available today. However, like any database system, it requires...

Mar 24

Frank Goortani

OpenSearch vs. Elasticsearch: A Comprehensive Comparison in 2025

The search and analytics landscape has evolved dramatically since Amazon's 2021 fork of Elasticsearch to create OpenSearch. Today, both...

Mar 5

Tihomir Manushev

Vector Search with pgvector in PostgreSQL

Simple AI-powered similarity search

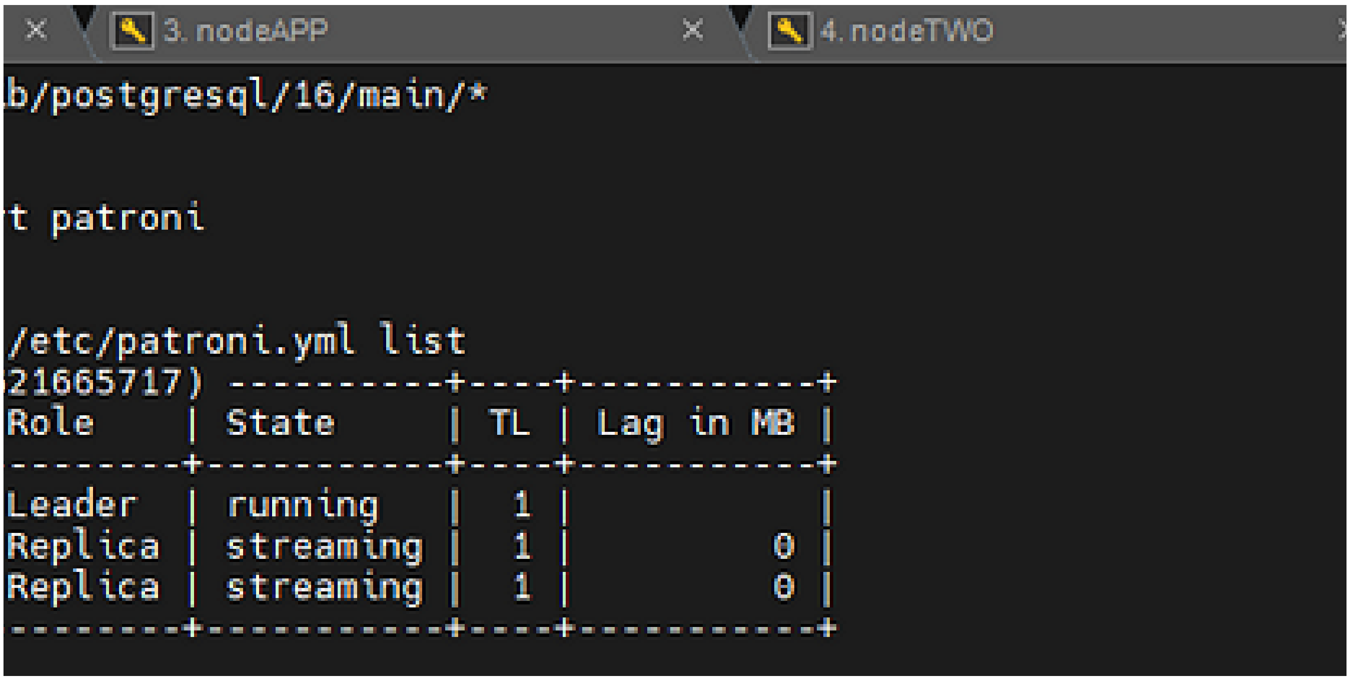
Mar 9

Vedant Jadhav

Understanding User, Roles and Privileges in PostgreSQL

Recently, I found myself in a situation where I had to grant specific database privileges and access to my colleagues based on their job...

Feb 3 1 1



```
3. nodeAPP                         4. nodeTWO
b/postgresql/16/main/*


t patroni


/etc/patroni.yml list
21665717) ----------+----+-----------+
Role     | State     | TL | Lag in MB |
--------+-----------+----+-----------+
Leader  | running   |  1 |           |
Replica | streaming |  1 |         0 |
Replica | streaming |  1 |         0 |
--------+-----------+----+-----------+
```

Dickson Gathima

Building a Highly Available PostgreSQL Cluster with Patroni, etcd, and HAProxy

Achieving high availability in PostgreSQL requires the right combination of tools and architecture.

Mar 14 4

In Towards Dev by Nakul Mitra

PostgreSQL Performance Optimization—Cleaning Dead Tuples & Reindexing

Performance optimization is crucial in PostgreSQL to ensure efficient query execution and minimal resource consumption.

Mar 28 1

See more recommendations