Member-only story

Postgres Security 101: User Access and Authorization (4/8)

Oz · Following

12 min read · Oct 4, 2024

Listen Share More

Managing user access and authorization is one of the foundational elements of database security in PostgreSQL. A well-defined user management strategy helps prevent unauthorized access, data breaches, and ensures that only the right users have the necessary permissions. In this article, we'll explore PostgreSQL's user roles, how to create users with different privilege levels, and the importance of adhering to the principle of least privilege. We'll also cover schema and object-level permissions, which allow for fine-grained access control in your PostgreSQL environment.

4.1 Ensure Sudo Is Configured Correctly

When created, the PostgreSQL user may have interactive access to the operating system, which means that the PostgreSQL user could login to the host as any other user would.

```
sudo grep postgres /etc/shadow | cut -d: -f1-2
# If this output is not postgres:!<something> then this is a failure.

# Execute the following command:
sudo passwd -l postgres
```


- Review and configure sudo permissions to limit administrative access. This grants any Operating System user that is a member of the dba group the ability to use sudo -iu postgres to become the postgres user

```
sudo groupadd dba
```

```
useradd -m -G dba kemal

echo '%dba ALL=(postgres) PASSWD: ALL' > /etc/sudoers.d/postgres

chmod 600 /etc/sudoers.d/postgres

su - kemal

sudo -iu postgres
```

4.3 Ensure Excessive Administrative Privileges Are Revoked

- With respect to PostgreSQL administrative SQL commands, only superusers should have elevated privileges. PostgreSQL regular, or application, users should not possess the ability to create roles, create new databases, manage replication, or perform any other action deemed privileged. Typically, regular users should only be granted the minimal set of privileges commensurate with managing the application

```
psql -c "\du+ kemal*"

List of roles
 Role name | Attributes |Member of | Description
-----------+------------+-------------------------+-------------
 kemal.oz  | Superuser  | {"DBA"}  |

psql -c "ALTER ROLE kemal.oz NOSUPERUSER;"

ALTER ROLE "kemal.oz" NOSUPERUSER;

psql -c "\du+ kemal*"
                                    List of roles
 Role name | Attributes |Member of | Description
-----------+------------+-------------------------+-------------
 kemal.oz  |            | {"DBA"}  |

# Lock Out Accounts if Not Currently in Use (Manual)
ALTER ROLE "kemal.oz" NOLOGIN;
```

4.4 Ensure Excessive Function Privileges Are Revoked (Manual)

- Manually review and revoke excessive function privileges. Functions in PostgreSQL can be created with the SECURITY DEFINER option. When SECURITY DEFINER functions are executed by a user, said function is run with the privileges of the user who created it, not the user who is running it.

```
SELECT nspname, proname, proargtypes, prosecdef, rolname, proconfig FROM pg_prc
 nspname |                    proname                    |       proargtypes
---------+-----------------------------------------------+-----------------------
 public  | dblink_connect_u                              | 25 25
 public  | dblink_connect_u                              | 25
 profile | get_diffreport                                | 19 3910 1043 25 16
 profile | get_diffreport                                | 19 1043 3910 25 16
 profile | get_diffreport                                | 19 3910 3910 25 16

/*In the query results, a prosecdef value of 't' on a row indicates that that
function uses privilege elevation. If elevation privileges are utilized which
are not required or are expressly forbidden by organizational guidance,
this is a fail.
*/

psql -c "ALTER FUNCTION dblink_connect_u SECURITY INVOKER;"

SELECT proname, proacl FROM pg_proc WHERE proname ='dblink_connect_u';
REVOKE EXECUTE ON FUNCTION dblink_connect_ (integer,boolean) FROM appreader;
```

4.5 Ensure Excessive DML Privileges Are Revoked

- Manually review and revoke excessive Data Manipulation Language (DML) privileges.

```
select t.schemaname, t.tablename, u.usename,
 has_table_privilege(u.usename, t.tablename, 'select') as select,
 has_table_privilege(u.usename, t.tablename, 'insert') as insert,
 has_table_privilege(u.usename, t.tablename, 'update') as update,
 has_table_privilege(u.usename, t.tablename, 'delete') as delete
from pg_tables t, pg_user u
where t.schemaname not in ('information_schema','pg_catalog');
 schemaname | tablename |   usename    | select | insert | update | delete
------------+-----------+--------------+--------+--------+--------+--------
 public     | a         | repuser      | f      | f      | f      | f
 public     | b         | repuser      | f      | f      | f      | f
 public     | a         | admin        | f      | f      | f      | f
 public     | b         | admin        | f      | f      | f      | f
 public     | a         | replication  | f      | f      | f      | f
 public     | b         | replication  | f      | f      | f      | f
 public     | a         | postgres     | t      | t      | t      | t
 public     | b         | postgres     | t      | t      | t      | t
 public     | a         | kemal.oz     | f      | t      | t      | f
 public     | b         | kemal.oz     | f      | f      | f      | f

select t.tablename, u.usename,
 has_table_privilege(u.usename, t.tablename, 'select') as select,
```

```
 has_table_privilege(u.usename, t.tablename, 'insert') as insert,
 has_table_privilege(u.usename, t.tablename, 'update') as update,
 has_table_privilege(u.usename, t.tablename, 'delete') as delete
from pg_tables t, pg_user u
where t.tablename = 'a'
and u.usename in ('kemal.oz');
 tablename | usename  | select | insert | update | delete
-----------+----------+--------+--------+--------+--------
 a         | kemal.oz | f      | t      | t      | f

REVOKE update, insert  ON TABLE a FROM "kemal.oz";

select t.tablename, u.usename,
 has_table_privilege(u.usename, t.tablename, 'select') as select,
 has_table_privilege(u.usename, t.tablename, 'insert') as insert,
 has_table_privilege(u.usename, t.tablename, 'update') as update,
 has_table_privilege(u.usename, t.tablename, 'delete') as delete
from pg_tables t, pg_user u
where t.tablename = 'a'
and u.usename in ('kemal.oz');
 tablename | usename  | select | insert | update | delete
-----------+----------+--------+--------+--------+--------
 a         | kemal.oz | f      | f      | f      | f

/*
Note: For versions of PostgreSQL prior to version 15, CVE-2018-1058 is applicab
it is recommended that all privileges be revoked from the public schema for all
all databases. If you have upgraded from one of these earlier releases, this CV
fixed for you during an upgrade. You can correct this CVE by issuing:
https://nvd.nist.gov/vuln/detail/CVE-2018-1058
*/
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
```

4.6 Ensure Row Level Security (RLS) Is Configured Correctly (Manual)

- Configure Row Level Security to restrict data access based on user roles.

```
SELECT oid, relname, relrowsecurity FROM pg_class WHERE relrowsecurity IS TRUE;
 oid | relname | relrowsecurity
-----+---------+----------------

CREATE TABLE passwd1 (
 user_name text UNIQUE NOT NULL,
 pwhash text,
 uid int PRIMARY KEY,
 gid int NOT NULL,
 real_name text NOT NULL,
 home_phone text,
```

```
 extra_info text,
 home_dir text NOT NULL,
 shell text NOT NULL
);

SELECT oid, relname, relrowsecurity FROM pg_class WHERE relname = 'passwd';
  oid  | relname | relrowsecurity
-------+---------+----------------
 26300 | passwd  | f

CREATE USER admin;
CREATE USER bob;
CREATE USER alice;
 INSERT INTO passwd VALUES
 ('admin','xxx',0,0,'Admin1','111-222-3333',null,'/root','/bin/dash');
INSERT INTO passwd VALUES
('bob','xxx',1,1,'Bob','123-456-7890',null,'/home/bob','/bin/zsh');
INSERT INTO passwd VALUES
('alice','xxx',2,1,'Alice','098-765-4321',null,'/home/alice','/bin/zsh');
ALTER TABLE passwd ENABLE ROW LEVEL SECURITY;
SELECT oid, relname, relrowsecurity FROM pg_class WHERE relname = 'passwd';
  oid  | relname | relrowsecurity
-------+---------+----------------
 26300 | passwd  | t
CREATE POLICY admin_all ON passwd TO admin USING (true) WITH CHECK (true);
CREATE POLICY user_mod ON passwd FOR UPDATE
 USING (current_user = user_name)
 WITH CHECK (
 current_user = user_name AND
 shell IN ('/bin/bash','/bin/sh','/bin/dash','/bin/zsh','/bin/tcsh')
 );

GRANT SELECT, INSERT, UPDATE, DELETE ON passwd TO admin;
GRANT SELECT
 (user_name, uid, gid, real_name, home_phone, extra_info, home_dir, shell)
 ON passwd TO public;
GRANT UPDATE
 (pwhash, real_name, home_phone, extra_info, shell)
 ON passwd TO public;
set role admin;
table passwd;
 user_name | pwhash | uid | gid | real_name |  home_phone   | extra_info |  home
-----------+--------+-----+-----+-----------+---------------+------------+------
 admin     | xxx    |   0 |   0 | Admin1    | 111-222-3333 |            | /root
 bob       | xxx    |   1 |   1 | Bob       | 123-456-7890 |            | /home
 alice     | xxx    |   2 |   1 | Alice     | 098-765-4321 |            | /home

set role alice;
table passwd;
ERROR:  permission denied for table passwd
select user_name,real_name,home_phone,extra_info,home_dir,shell from passwd;
 user_name | real_name |  home_phone   | extra_info |  home_dir   |   shell
-----------+-----------+---------------+------------+-------------+-----------
```

```
 admin     | Admin1    | 111-222-3333 |            | /root       | /bin/dash
 bob       | Bob       | 123-456-7890 |            | /home/bob   | /bin/zsh
 alice     | Alice     | 098-765-4321 |            | /home/alice | /bin/zsh

update passwd set user_name = 'joe';
ERROR:  permission denied for table passwd
update passwd set real_name = 'Alice Doe';
UPDATE 1
update passwd set real_name = 'John Doe' where user_name = 'admin';
UPDATE 0
select user_name,real_name,home_phone,extra_info,home_dir,shell from passwd;
 user_name | real_name |  home_phone  | extra_info |  home_dir   |   shell
-----------+-----------+--------------+------------+-------------+-----------
 admin     | Admin1    | 111-222-3333 |            | /root       | /bin/dash
 bob       | Bob       | 123-456-7890 |            | /home/bob   | /bin/zsh
 alice     | Alice Doe | 098-765-4321 |            | /home/alice | /bin/zsh
update passwd set real_name = 'John Doe' where user_name = 'admin1';
UPDATE 0
 select user_name,real_name,home_phone,extra_info,home_dir,shell from passwd;
 user_name | real_name |  home_phone  | extra_info |  home_dir   |   shell
-----------+-----------+--------------+------------+-------------+-----------
 admin     | Admin1    | 111-222-3333 |            | /root       | /bin/dash
 bob       | Bob       | 123-456-7890 |            | /home/bob   | /bin/zsh
 alice     | Alice Doe | 098-765-4321 |            | /home/alice | /bin/zsh

update passwd set shell = '/bin/xx';
ERROR:  new row violates row-level security policy for table "passwd"
delete from passwd;
ERROR:  permission denied for table passwd
insert into passwd (user_name) values ('xxx');
ERROR:  permission denied for table passwd
update passwd set pwhash = 'abc';
UPDATE 1
select user_name,real_name,home_phone,extra_info,home_dir,shell from passwd;
 user_name | real_name |  home_phone  | extra_info |  home_dir   |   shell
-----------+-----------+--------------+------------+-------------+-----------
 admin     | Admin1    | 111-222-3333 |            | /root       | /bin/dash
 bob       | Bob       | 123-456-7890 |            | /home/bob   | /bin/zsh
 alice     | Alice Doe | 098-765-4321 |            | /home/alice | /bin/zsh
update passwd set pwhash = 'abc';
select user_name,real_name,home_phone,extra_info,home_dir,shell from passwd;
 user_name | real_name |  home_phone  | extra_info |  home_dir   |   shell
-----------+-----------+--------------+------------+-------------+-----------
 admin     | Admin1    | 111-222-3333 |            | /root       | /bin/dash
 bob       | Bob       | 123-456-7890 |            | /home/bob   | /bin/zsh
 alice     | Alice Doe | 098-765-4321 |            | /home/alice | /bin/zsh
# Authorized user
table passwd;
 user_name | pwhash | uid | gid | real_name |  home_phone  | extra_info |  home
-----------+--------+-----+-----+-----------+--------------+------------+------
 admin     | xxx    |   0 |   0 | Admin1    | 111-222-3333 |            | /root
 bob       | xxx    |   1 |   1 | Bob       | 123-456-7890 |            | /home
 alice     | abc    |   2 |   1 | Alice Doe | 098-765-4321 |            | /home
```

4.7 Ensure the set_user Extension Is Installed (Manual)

- Install the `set_user` extension to safely change user identities. PostgreSQL access to the superuser database role must be controlled and audited to prevent unauthorized access. Prior to performing this audit you must create a roletree view. Here are the procedures to create this view:

```
CREATE OR REPLACE VIEW roletree AS
WITH RECURSIVE
roltree AS (
 SELECT u.rolname AS rolname,
        u.oid AS roloid,
        u.rolcanlogin,
        u.rolsuper,
        '{}'::name[] AS rolparents,
        NULL::oid AS parent_roloid,
        NULL::name AS parent_rolname
 FROM pg_catalog.pg_authid u
 LEFT JOIN pg_catalog.pg_auth_members m ON u.oid = m.member
 LEFT JOIN pg_catalog.pg_authid g ON m.roleid = g.oid
 WHERE g.oid IS NULL
 UNION ALL
 SELECT u.rolname AS rolname,
        u.oid AS roloid,
        u.rolcanlogin,
        u.rolsuper,
        t.rolparents || g.rolname AS rolparents,
        g.oid AS parent_roloid,
        g.rolname AS parent_rolname
 FROM pg_catalog.pg_authid u
 JOIN pg_catalog.pg_auth_members m ON u.oid = m.member
 JOIN pg_catalog.pg_authid g ON m.roleid = g.oid
 JOIN roltree t ON t.roloid = g.oid
)
SELECT rolname,
       roloid,
       rolcanlogin,
       rolsuper,
       rolparents,
       parent_roloid,
       parent_rolname
FROM roltree;

SELECT
 r.rolname,
 r.roloid,
```

```
 r.rolcanlogin,
 r.rolsuper,
 r.rolparents
FROM roletree r
ORDER BY 1;
          rolname          | roloid | rolcanlogin | rolsuper |       rolparents
---------------------------+--------+-------------+----------+-----------------
 admin                     |  16385 | t           | f        | {}
 admin1                    |  26309 | t           | f        | {}
 alice                     |  26311 | t           | f        | {}
 bob                       |  26310 | t           | f        | {}
 kemal.oz                  |  26299 | t           | f        | {}
 pg_database_owner         |   6171 | f           | f        | {}
 pg_execute_server_program |   4571 | f           | f        | {}
 pg_monitor                |   3373 | f           | f        | {pg_stat_scan_ta
 pg_monitor                |   3373 | f           | f        | {pg_read_all_set
 pg_monitor                |   3373 | f           | f        | {pg_read_all_sta
 pg_read_all_data          |   6181 | f           | f        | {}
 pg_read_all_settings      |   3374 | f           | f        | {}
 pg_read_all_stats         |   3375 | f           | f        | {}
 pg_read_server_files      |   4569 | f           | f        | {}
 pg_signal_backend         |   4200 | f           | f        | {}
 pg_stat_scan_tables       |   3377 | f           | f        | {}
 pg_write_all_data         |   6182 | f           | f        | {}
 pg_write_server_files     |   4570 | f           | f        | {}
 postgres                  |     10 | t           | t        | {}
 replication               |  16386 | t           | f        | {}
 repuser                   |  16384 | t           | f        | {}
(21 rows)
SELECT
 ro.rolname,
 ro.roloid,
 ro.rolcanlogin,
 ro.rolsuper,
 ro.rolparents
FROM roletree ro
WHERE (ro.rolcanlogin AND ro.rolsuper)
OR
(
 ro.rolcanlogin AND EXISTS
 (
 SELECT TRUE FROM roletree ri
 WHERE ri.rolname = ANY (ro.rolparents)
 AND ri.rolsuper
 )
);
 rolname  | roloid | rolcanlogin | rolsuper | rolparents
----------+--------+-------------+----------+------------
 postgres |     10 | t           | t        | {}
rpm -ivh /a/set_user_14-4.0.1-2.rhel9.x86_64.rpm
patronictl -c /etc/patroni/patroni.yml edit-config
shared_preload_libraries = 'set_user,pgaudit,somethingelse' --Required restart
select * from pg_available_extensions where name = 'set_user';
```

```
   name    | default_version | installed_version |                  comment
----------+-----------------+-------------------+------------------------------
 set_user | 4.0.1           |                   | similar to SET ROLE but with

create extension set_user;
select * from pg_available_extensions where name = 'set_user';
   name    | default_version | installed_version |                  comment
----------+-----------------+-------------------+------------------------------
 set_user | 4.0.1           | 4.0.1             | similar to SET ROLE but with
/*
Now, we use GRANT to configure each DBA role to allow it to use the set_user fu
In the example below, we will configure my db user kemal.oz. (You would do this
DBA's normal user role.)
*/
grant execute on function set_user(text) to "kemal.oz","alı.oz";
-- "kemal.oz" is an unprivileged user that can run as "alı.oz" through calls to
-- "alı.oz" is an unprivileged user that can run as "kemal.oz" through calls to
grant execute on function set_user_u(text) to "kemal.oz";
-- kemal.oz is the privileged (non-superuser) role, which is able to escalate p
set role "kemal.oz";
select set_user('postgres');
ERROR: switching to superuser not allowed
 select set_user_u('postgres');
 set_user_u
------------
 OK
 select current_user, session_user;
 current_user | session_user
--------------+--------------
 postgres     | postgres
select reset_user();
 reset_user
------------
 OK
select current_user, session_user;
 current_user | session_user
--------------+--------------
 kemal.oz     | kemal.oz

ALTER USER postgres NOLOGIN;
ERROR:  must be superuser to alter superuser roles or change superuser attribut
alter user "kemal.oz" SUPERUSER;
ALTER USER postgres NOLOGIN;
REVOKE name_of_granting_role FROM kemal.oz; -- an example only REVOKE ROLE
```

4.8 Make Use of Predefined Roles (Manual)

- Utilize predefined roles to simplify and secure role management.

```
select rolname from pg_roles where rolsuper is true;

rolname
------------
 kemal.oz
 db_monitor
 postgres


GRANT pg_monitor TO "kemal.oz";

Default Value:
The following predefined roles exist in PostgreSQL 13.x:
• pg_read_all_data
Read all data (tables, views, sequences), as if having SELECT rights on those
objects, and USAGE rights on all schemas, even without having it explicitly. Th
role does not have the role attribute BYPASSRLS set. If RLS is being used, an
administrator may wish to set BYPASSRLS on roles which this role is GRANTed to.
• pg_write_all_data
Write all data (tables, views, sequences), as if having INSERT, UPDATE, and DEL
rights on those objects, and USAGE rights on all schemas, even without having i
explicitly. This role does not have the role attribute BYPASSRLS set. If RLS is
used, an administrator may wish to set BYPASSRLS on roles which this role is
GRANTed to.
• pg_read_all_settings
Read all configuration variables, even those normally visible only to superuser
• pg_read_all_stats
Read all pg_stat_* views and use various statistics related extensions, even
those normally visible only to superusers.
• pg_stat_scan_tables
Execute monitoring functions that may take ACCESS SHARE locks on tables,
potentially for a long time.
• pg_monitor
Read/execute various monitoring views and functions. This role is a member of
pg_read_all_settings, pg_read_all_stats and pg_stat_scan_tables.
• pg_database_owner
None. Membership consists, implicitly, of the current database owner.
• pg_signal_backend
Signal another backend to cancel a query or terminate its session.
• pg_read_server_files
Allow reading files from any location the database can access on the server wit
COPY and other file-access functions.
pg_write_server_files
Allow writing to files in any location the database can access on the server wi
COPY and other file-access functions.
• pg_execute_server_program
Allow executing programs on the database server as the user the database runs
as with COPY and other functions which allow executing a server-side program.
• pg_checkpoint
Allow executing the CHECKPOINT command.
• pg_use_reserved_connections
```

```
Allow use of connection slots reserved via reserved_connections.
• pg_create_subscription
Allow users with CREATE permission on the database to issue CREATE SUBSCRIPTION
```

4.9 Ensure the Public Schema Is Protected

- Restrict access to the public schema to prevent unauthorized changes. you need to manage the permissions on the public schema carefully.

```
/*
Note: For versions of PostgreSQL prior to version 15, CVE-2018-1058 is applicab
it is recommended that all privileges be revoked from the public schema for all
all databases. If you have upgraded from one of these earlier releases, this CV
fixed for you during an upgrade. You can correct this CVE by issuing:
https://nvd.nist.gov/vuln/detail/CVE-2018-1058
*/

REVOKE CREATE ON SCHEMA public FROM PUBLIC;
GRANT CREATE ON SCHEMA public TO "kemal.oz"; -- Grant necessary privileges to s
/*
Periodically, you should review the current permissions on the public schema
to ensure that there are no unauthorized changes.
*/
SELECT grantee, privilege_type
FROM information_schema.role_table_grants
WHERE table_schema = 'public';
SELECT nspname, usename, has_schema_privilege(usename, nspname, 'CREATE') AS cr
FROM pg_namespace, pg_user
WHERE nspname = 'public';
```

By effectively managing user access and authorization, you can significantly strengthen the security of your PostgreSQL database. Following best practices, such as using role-based access control, assigning the minimum required privileges, and regularly reviewing user permissions, will help keep your data safe and secure. Properly defined access levels minimize the risk of internal and external threats, contributing to a more robust database environment. If you're looking to continue learning about PostgreSQL security, be sure to check out my next article: "***Postgres Security 101: Connection and Login (5/8)***". It dives deep into password management strategies and best practices to ensure your database is protected from unauthorized access. 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.

Database Security | Postgres Security | Security | Cybersecurity | Technology

Following

Written by Oz

149 Followers · 13 Following

Database Administrator 🐘

No responses yet

Gvadakte

What are your thoughts?

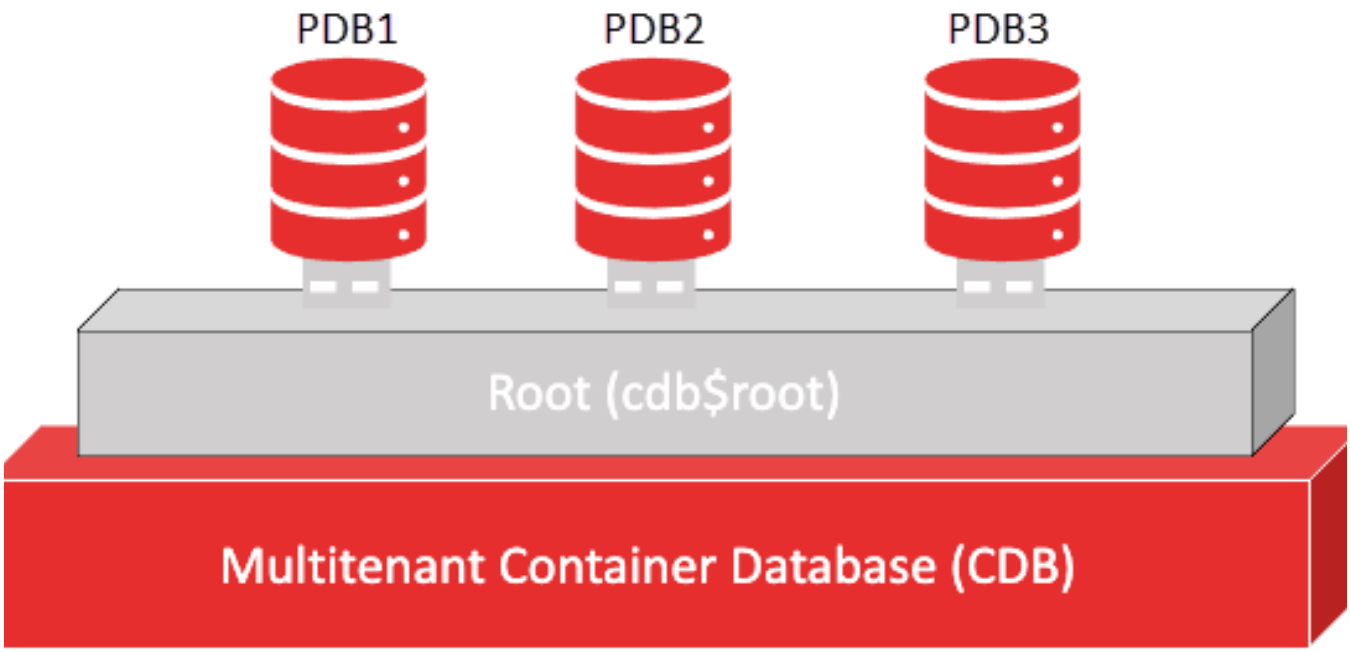
More from Oz

Oz

Installing Percona Monitoring & Management (PMM) with Postgres

Introduction:

Sep 26, 2024 54 1

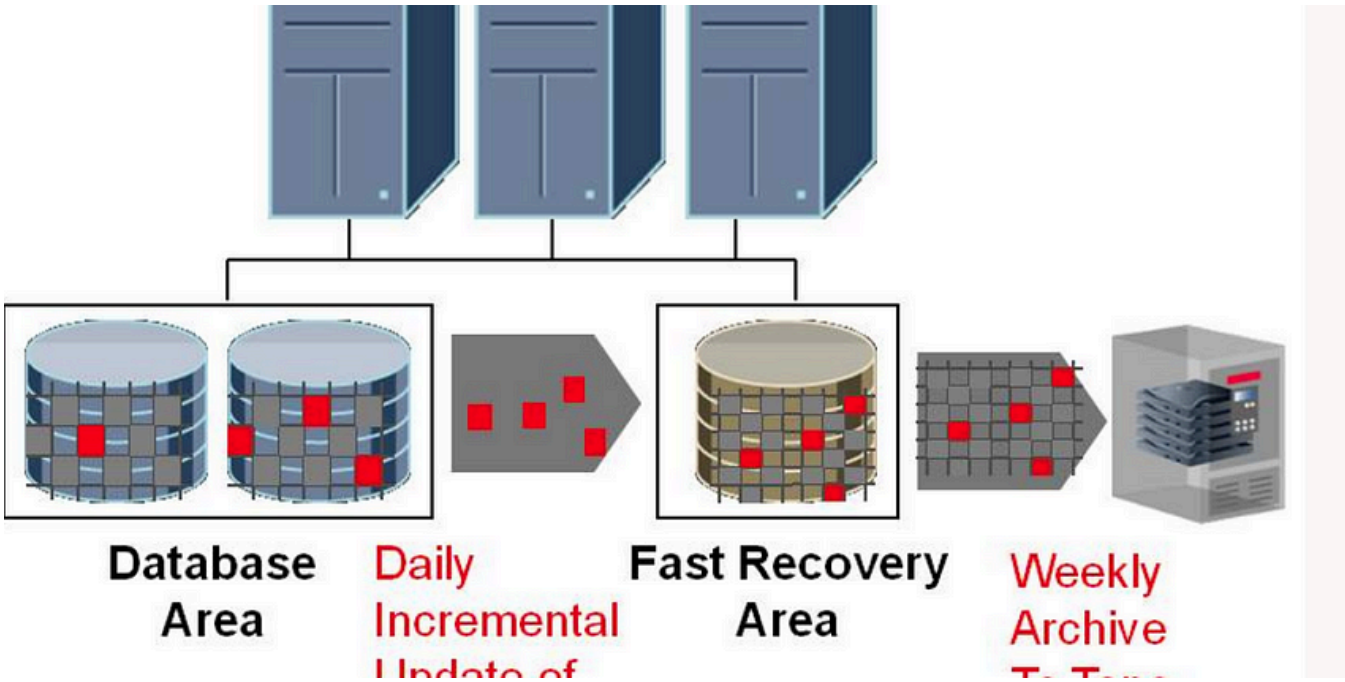


Oz

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

Oz

delete jobs

May 8, 2023

See all from Oz

Recommended from Medium

Tihomir Manushev

Vector Search with pgvector in PostgreSQL

Simple AI-powered similarity search

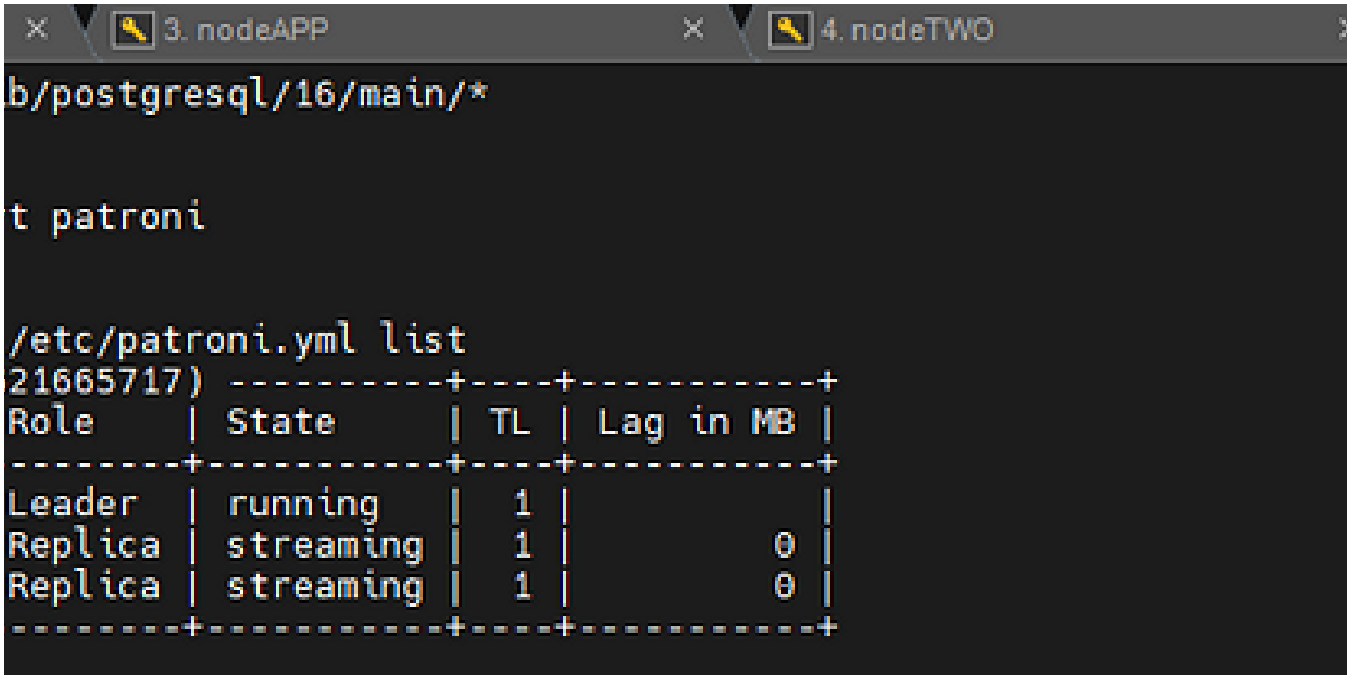
Mar 9

mehmetozanguven

Running PostgreSQL with Podman

Instead of running PostgresSQL locally, we can easily run with Podman. Here are the basic steps you should follow.

Mar 28 2



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 Databases by Sergey Egorenkov

Making SQL query 40x faster for 10 million rows table

Make your SQL query really fast using this approach

Mar 17 8

Anubhav Bhardwaj

PostgreSQL Monitoring Script with Email Alerts

Overview

Jan 14 5

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