Postgres Security 101: Connection and Login (5/8)

Oz · Following

This section outlines essential best practices for securing PostgreSQL connections and logins, including the proper configuration of authentication methods, connection limits, and password complexity policies. These practices will help ensure that your database is secure against unauthorized access and mitigate potential attack vectors.

5.1 Do Not Specify Passwords in the Command Line

- When a command is executed on the command line, for example

```
postgresql://postgres:PASSWORD@host
```

the password may be visible in the user's shell/command history or in the process list, thus exposing the password to other entities on the server.

```
# Check the process or task list if the password is visible.
sudo ps -few
# Check the shell or command history if the password is visible.
History
#Remediation: Use the --password or -W terminal parameter without directly spec
psql -u <user> --password
```

5.2 Ensure PostgreSQL is Bound to an IP Address

- By default, listen_addresses is set to localhost which prevents any and all remote TCP connections to the PostgreSQL port. Some Docker images may set listen_addesses to *. * corresponds to all available IP interfaces; thus, the PostgreSQL server then accepts TCP connections on all the server's IPv6 and IPv4 interfaces. (The same is true for a setting of 0.0.0.0.) You can make this configuration more restrictive by setting the listen_addresses configuration option to a specific list of IPv4 or IPv6 address so that the server only accepts TCP connections on those addresses. This parameter can only be set at server start

```
SHOW listen_addresses;
-- If * or 0.0.0.0 is returned, this is a failure.
```

5.3 Ensure Login via "Local" UNIX Domain Socket Is Configured Correctly

- Configure secure login methods for local connections. A remote host login, via SSH, is arguably the most secure means of remotely accessing and administering the PostgreSQL server. Once connected to the PostgreSQL server, using the psql client, via UNIX DOMAIN SOCKETS, while using the peer authentication method is the most secure mechanism available for local database connections. Provided a database user account of the same name of the UNIX account has already been defined in the database, even ordinary user accounts can access the cluster in a similarly highly secure manner

```
-- allow only postgres user logins locally via UNIX socket
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             postgres                                peer

-- allow all local users via UNIX socket
local   all             all                                     peer
-- allow all local users, via UNIX socket, only if they are connecting to a db
local   samerole        all                                     peer
-- allow only local users, via UNIX socket, who are members of the 'rw' role ir
local   all             +rw                                     peer
```

5.4 Ensure Login via "Host" TCP/IP Socket Is Configured Correctly

- A large number of authentication methods are available for hosts connecting using TCP/IP sockets. Methods trust, password, and ident are not to be used for remote logins. Method md5 used to be the most popular and can be used in both encrypted and unencrypted sessions, however, it is vulnerable to packet replay attacks. It is recommended that scram-sha-256 be used instead of md5. Use of the gss, sspi, pam, **ldap**, radius, and cert methods are dependent upon the availability of external authenticating processes/services and thus are not covered here.

```
# The use of the "md5" authentication method is vulnerable to packet replay att
host    all        all             10.6.128.238/32        md5
# This is fail

host    all        all             10.10.80.238/32        scram-sha-256
# This can be pass our exam
```

Please read **ldap** and other authentication method for increase your security.

5.5 Ensure per-account connection limits are used

- Limiting concurrent connections to a PostgreSQL server can be used to reduce the risk of Denial of Service (DoS) attacks.

```
SELECT rolname, rolconnlimit
FROM pg_roles
WHERE rolname NOT LIKE 'pg_%';;
-- Default Value: -1

rolname    | rolconnlimit
-------------+--------------
 repuser     |           -1
 admin       |           -1
 replication |           -1
 admin1      |           -1
 bob         |           -1
 alice       |           -1
 kemal.oz    |           -1
 db_monitor  |           -1
 postgres    |           -1
(9 rows)
ALTER USER "kemal.oz" CONNECTION LIMIT 4; -- please set reasonable concurrent c
SELECT rolname, rolconnlimit
```

```
FROM pg_roles
WHERE rolname NOT LIKE 'pg_%';;
   rolname    | rolconnlimit
-------------+--------------
 repuser      |           -1
 admin        |           -1
 replication  |           -1
 admin1       |           -1
 bob          |           -1
 alice        |           -1
 db_monitor   |           -1
 postgres     |           -1
 kemal.oz     |            4
```

5.6 Ensure Password Complexity Is Configured

- Enforce strong password policies. Password complexity configuration is crucial to restrict unauthorized access to data. By default, PostgreSQL doesn't provide for password complexity. Moreover, many compliance frameworks such as PCI DSS, and HIPPA require both password complexity and length. It is worth stating that the NIST 800–63B Password Guidelines publication is a good reference of authentication management.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
set_user,pgaudit

SHOW dynamic_library_path;
dynamic_library_path
----------------------
$libdir
vi ${PGDATA}/postgresql.conf
shared_preload_libraries = 'pgaudit,$libdir/passwordcheck,somethingelse'
-- This changes need to restart
create user deneme1 password 'test123';
ERROR:  password is too short
-- password required at least 8 charackter
```

5.7 Ensure Authentication Timeout and Delay Are Well Configured

- Set appropriate authentication timeouts and delays to prevent brute-force attacks. You can edit postgresql.conf below parameter

```
connection_timeout = 3s  # example timeout of 3 seconds
```

If you can masquerade your eggs like haproxy you can edit haproxy configuration file. As show below;

```
#/etc/haproxy/haproxy.cfg

timeout connect 3s
```

5.8 Ensure SSL Is Used for Client Connection

- Enable SSL to encrypt client connections. Secure login methods for remote connections. A large number of authentication METHODs are available for hosts connecting using TCP/IP sockets, including: • trust • reject • md5 • scram-sha-256 • password • gss • sspi • ident • pam • "ldap" • radius • cert. You can visit to apply "Securing PostgreSQL with SSL Encryption" article for detail.

```
psql 'host=10.10.80.68 user=postgres sslmode=require'
# output
psql: error: connection to server at "10.5.56.67", port 5432 failed: server doe
# If output like this then this is a failure. Please read article above

psql 'host=10.10.80.68 user=postgres sslmode=require'
# output
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, c
1. Use TYPE hostssl when administrating the database cluster as a superuser.
2. Use TYPE hostnossl for performance purposes and when DML operations are deem
```

Additional Information

1. Use TYPE hostssl when administrating the database cluster as a superuser.
2. Use TYPE hostnossl for performance purposes and when DML operations are deemed safe without SSL connections.

5.9 Ensure Authorized IP Address Ranges Are Not Too Large

- Restrict IP address ranges for connections to minimize exposure. P addressing uses a 32-bit address to identify each host on an IPv4 network. To make addresses easier to read, they are written in dotted decimal notation, each address being four octets in length. For example, address 00001010000000010000000100000001 in binary is written as 10.1.1.1. However, the most important thing is range in terms of subnet that dividing a network into at least two separate networks. ***Please do not forget allowing a too large range of Ip addresses to connect to PostgreSQL cluster multiply the risks unnecessarily***

```
Below is a table providing typical subnets for IPv4.

Prefix size Network mask Usable hosts per subnet
/1           128.0.0.0      2,147,483,646
/2           192.0.0.0      1,073,741,822
/3           224.0.0.0      536,870,910
/4           240.0.0.0      268,435,454
/5           248.0.0.0      134,217,726
/6           252.0.0.0      67,108,862
/7           254.0.0.0      33,554,430
Class A
/8          255.0.0.0       16,777,214
/9          255.128.0.0     8,388,606
/10         255.192.0.0     4,194,302
/11         255.224.0.0     2,097,150
/12         255.240.0.0     1,048,574
/13         255.248.0.0     524,286
/14         255.252.0.0     262,142
/15         255.254.0.0     131,070
Class B
/16         255.255.0.0     65,534
/17         255.255.128.0   32,766
/18         255.255.192.0   16,382
/19         255.255.224.0   8,190
/20         255.255.240.0   4,094
/21         255.255.248.0   2,046
/22         255.255.252.0   1,022
/23         255.255.254.0   510
Class C
/24         255.255.255.0   254
/25         255.255.255.128 126
/26         255.255.255.192 62
/27         255.255.255.224 30
/28         255.255.255.240 14
/29         255.255.255.248 6
/30         255.255.255.252 2
```

```
/31         255.255.255.254 0
/32         255.255.255.255 0
```

5.10 Ensure Specific Database and Users Are Used

- Limit access to specific databases and users. The keyword “all” in the database and user part of the pg_hba.conf rules can allow any user to connect to any database, it is recommended to restrict the connection to specific user and database

```
\h create role
Command:     CREATE ROLE
Description: define a new database role
Syntax:s
CREATE ROLE name [ [ WITH ] option [ ... ] ]
where option can be:
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
...
A good practice when creating roles is to add a role with the CREATEROLE and
CREATEDB attributes but not the SUPERUSER. And use this role to add any other
role and database you need to create.
CREATE ROLE admin WITH
  LOGIN
  NOSUPERUSER
  NOINHERIT
  CREATEDB
  CREATEROLE
  NOREPLICATION
  CONNECTION LIMIT 10
  PASSWORD 'test123'
  VALID UNTIL '2024-04-19 17:00:00+03';
create role "user1" in role "admin";
alter user "user1" login;
"""
The role attributes LOGIN, SUPERUSER, CREATEDB, and CREATEROLE can be thought
of as special privileges,  but they are never inherited as ordinary privileges
on database objects are.  You must actually SET ROLE to a specific role having
one of these attributes in order to make use of the attribute.
"""
SELECT * FROM pg_roles ;
\du
"""
you can manage your pg_hba configuration file your role. Below passed example
admin role can be login to clusterdb. You can create a lot of role your requrim
"""
```

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    all             all             10.80.208.0/24          scram-sha-256
""" Like tihs configurations fail our exam"""
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    clusterdb       admin           10.80.208./21           scram-sha-256
# Like tihs configurations pass our exam
```

5.11 Ensure Superusers Are Not Allowed to Connect Remotely

- Prevent superuser remote connections to enhance security. Allowing a PostgreSQL superuser to connect to a database from a remote host is dangerous, best is to only allow the superuser(s) to connect locally with a peer authentication. If some advanced privileges are required, best is to use the PostgreSQL predefined roles.

```
show listen_addresses;
 listen_addresses
------------------
 10.70.12.23
(1 row)
**10.70.12.23: This is server ip4. Please avoid to use * or 0.0.0.0

/*
If You are admin you can edit conf file. Please do not forget other parameter
user cannot be all. If you give this permission. This is fail.
*/
TYPE    DATABASE   USER                ADDRESS                 METHOD
Host    all        postgres            10.10.120.120/32        scram-sha-256
**10.10.120.120: Admin
```

5.12 Ensure That ‘password_encryption’ Is Correctly Set

- Use strong encryption for stored passwords. PostgreSQL allow to set password encryption, default is now ‘scram-sha-256’ but it can be set to ‘md5’ or ‘trust’ which is insecure. Consider other methods your compan policies; • reject • scram-sha-256 • password • gss • sspi • ident • pam • “ldap” • radius • cert.

```
TYPE    DATABASE   USER                ADDRESS                 METHOD
Host    all        postgres            10.10.120.0/24          trust
Host    all        postgres            10.10.120.0/24          md5
```

```
-- This is failed

TYPE    DATABASE   USER                ADDRESS                 METHOD
Host    all        postgres            10.10.120.0/24          scram-sha-256
-- This is passed
```

Securing a PostgreSQL database is a crucial responsibility for database administrators, as security breaches can lead to data leaks or other serious risks. In this article, we've covered key steps to enhance PostgreSQL security, such as avoiding passwords in command-line connections, restricting server access to specific IP addresses, using appropriate authentication methods, and enabling SSL connections. Additionally, setting user connection limits and enforcing strong password policies are vital measures for securing your database system. Beyond these initial steps, there are many other aspects of database security to consider, including encryption, backup strategies, audit logs, and monitoring. If you're interested in deepening your knowledge on PostgreSQL security, I recommend checking out another one of my articles: "**Postgres Security 101: PostgreSQL Settings (6/8)**". It provides detailed insights into essential PostgreSQL settings that can further safeguard your database system and ensure better security management. 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 | Techonology

Following

Written by Oz

149 Followers · 13 Following

Database Administrator 🐘

No responses yet

Gvadakte

What are your thoughts?

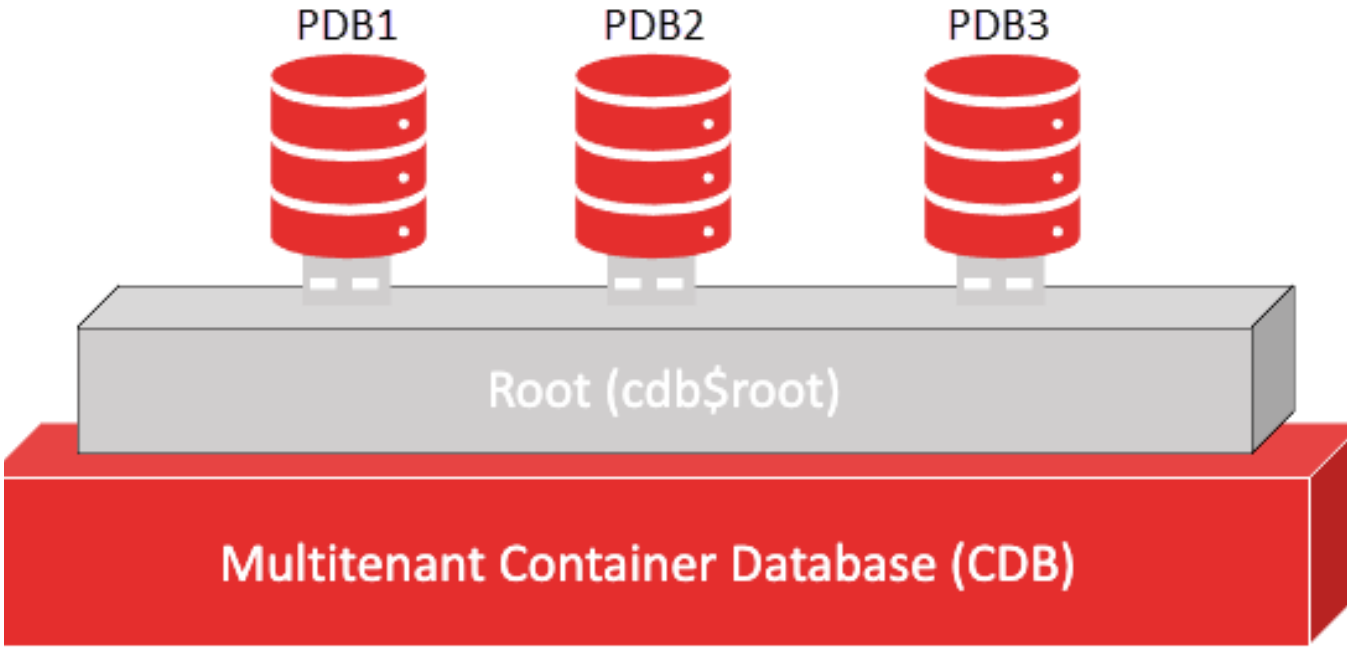
More from Oz

Oz

Installing Percona Monitoring & Management (PMM) with Postgres

Introduction:

Sep 26, 2024 54 1

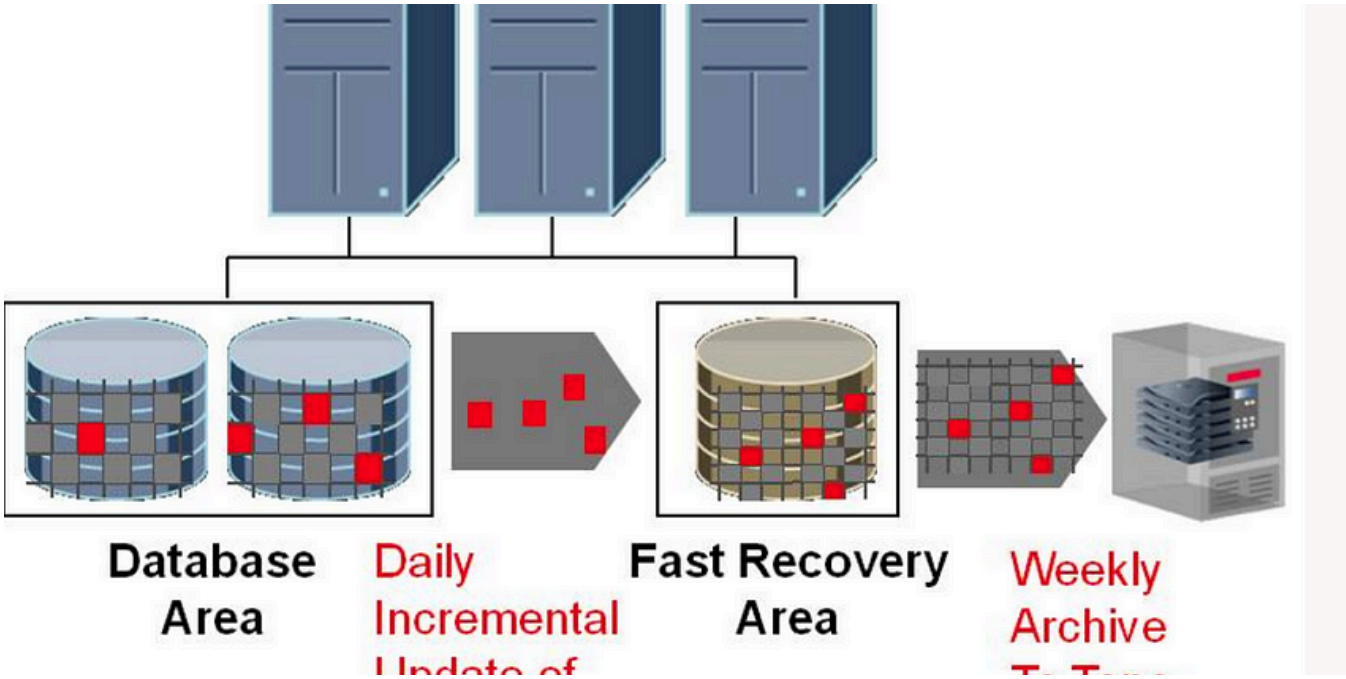


Oz

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

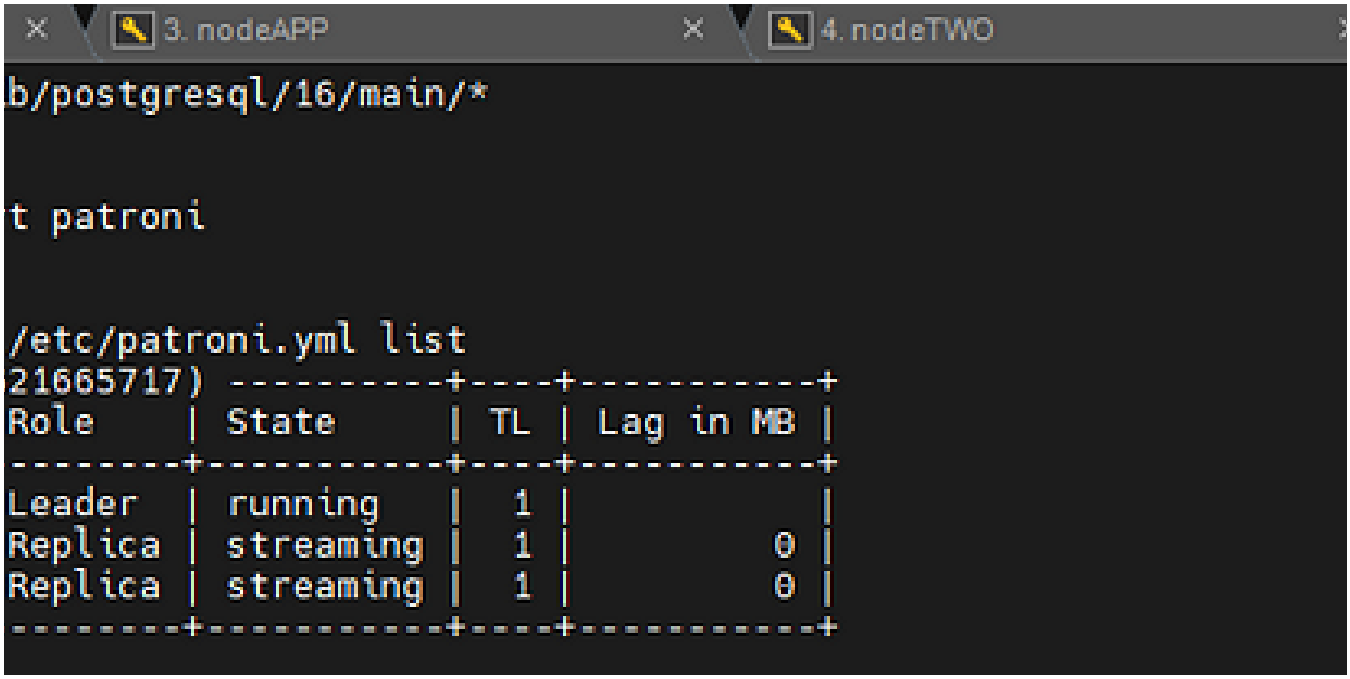
Recommended from Medium

Tihomir Manushev

Vector Search with pgvector in PostgreSQL

Simple AI-powered similarity search

Mar 9

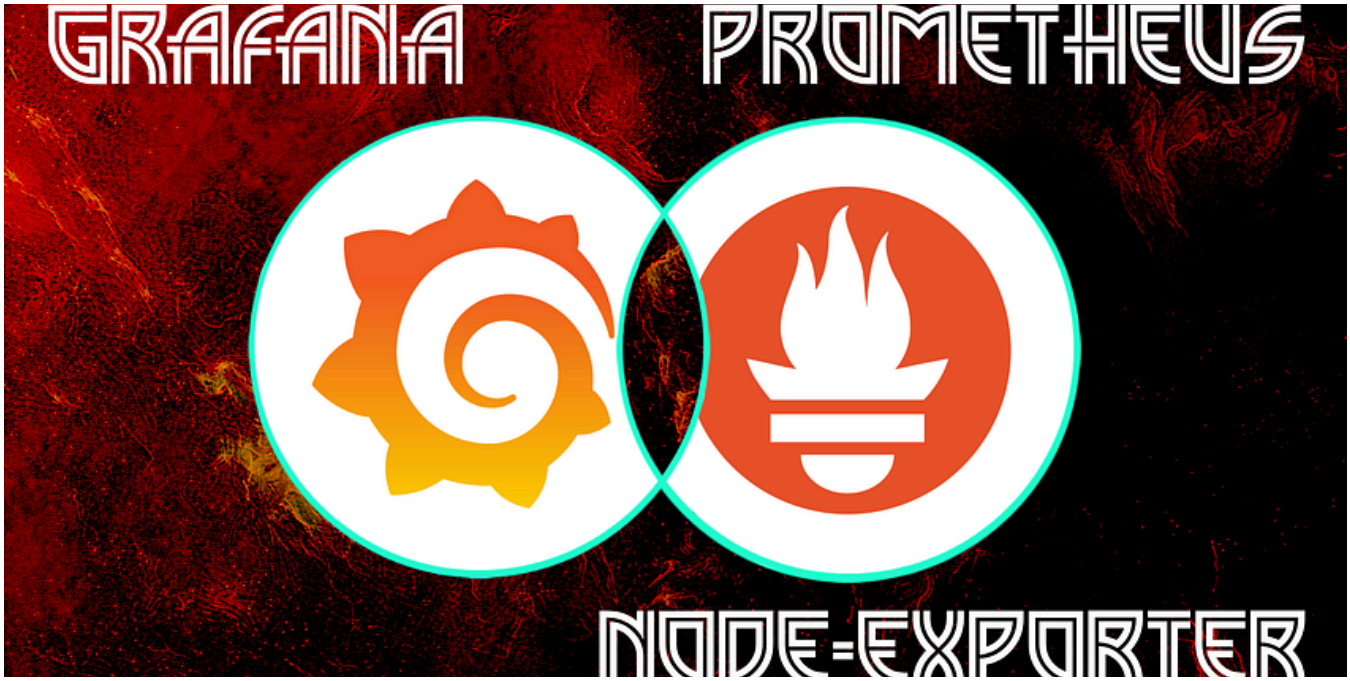


Dickson Gathima

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



crptcpchk

Grafana, Prometheus & Node-Exporter | Setup Guide

Grafana open source software enables you to query, visualize, alert on, and explore your metrics, logs, and traces wherever they are stored...

Oct 30, 2024 8 1

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

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

In Databases by Sergey Egorenkov

Why Uber Moved from Postgres to MySQL

How PostgreSQL's architecture clashed with Uber's scale—and why MySQL offered a better path forward

Mar 29 228 7

See more recommendations