How to Enable and Secure Remote PostgreSQL 17 Connections on Red Hat Linux: Complete Step-by-Step Guide

7 min read · Jun 2, 2025

Jeyaram Ayyalusamy Following

PostgreSQL is one of the most popular open-source relational database management systems (RDBMS) used across industries. It powers mission-critical systems, SaaS platforms, business intelligence tools, and data lakes globally. But while PostgreSQL is extremely flexible, its **default security posture disables remote access** to ensure databases are protected out-of-the-box.

In real-world production, however, you often need to allow external connections:

- Application servers connecting from other machines
- Developers accessing databases remotely
- BI tools querying PostgreSQL over the network
- Multi-tier or microservices architecture where database servers are decoupled

In this comprehensive blog, we'll explore **how to enable remote connections to PostgreSQL 17 running on Red Hat Enterprise Linux (RHEL)** and its derivatives (CentOS, Rocky Linux, AlmaLinux, Oracle Linux), while ensuring best security and performance practices.

Why Remote Access Is Not Enabled by Default?

PostgreSQL disables remote access by default for a very good reason: **security**. Open database ports exposed to the internet are one of the most common ways databases get compromised.

That's why **enabling remote access requires deliberate configuration**, proper network security, and access control to protect your data.

✅ **Good practice:** Open PostgreSQL to trusted internal networks or VPN-connected machines, but avoid exposing directly to public internet unless absolutely necessary (and secured with SSL + proper firewalling).

PostgreSQL Configuration on Red Hat Linux — File Locations

Before diving into the steps, let's review where PostgreSQL stores its configuration files on Red Hat Linux:

File Default Path `postgresql.conf` `/var/lib/pgsql/17/data/postgresql.conf` `pg_hba.conf` `/var/lib/pgsql/17/data/pg_hba.conf` PostgreSQL logs (default) `/var/lib/pgsql/17/data/log/`

🔎 *If unsure where PostgreSQL installed, verify using:*

```
psql -U postgres -c 'SHOW config_file;'
```

```
[postgres@node1~]$ psql -U postgres -c 'SHOW config_file;'
              config_file
----------------------------------------
 /var/lib/pgsql/17/data/postgresql.conf
(1 row)

[postgres@node1~]$
```

> ✅ **Tip:** Always back up these files before making configuration changes.

🔧 Step 1 — Modify PostgreSQL Configuration for Remote Listening

By default, PostgreSQL only listens on `localhost` . We need to change this behavior.

1.1 Edit `postgresql.conf`

```
sudo vi /var/lib/pgsql/17/data/postgresql.conf

[root@node1 ~]# cat /var/lib/pgsql/17/data/postgresql.conf | grep listen_addres
#listen_addresses = 'localhost'         # what IP address(es) to listen on;
[root@node1 ~]#
```

Locate:

```
listen_addresses = 'localhost'

[root@node1 ~]# cat /var/lib/pgsql/17/data/postgresql.conf | grep listen_addres
listen_addresses = '*'          # what IP address(es) to listen on;
[root@node1 ~]#
```

Change to:

```
listen_addresses = '*'
```

- `*` means PostgreSQL will listen on all available network interfaces.
- You may also restrict to specific IPs:

```
listen_addresses = '192.168.100.50'
```

✅ **Security Tip:** If your server has multiple interfaces (private/public), it's often safer to bind PostgreSQL only to private IPs.

Of course! Here's a **shortened, clean, and Medium-ready version** of your blog post:

1.2 Understanding PostgreSQL 17 `port` Parameter in `postgresql.conf`

PostgreSQL 17's `postgresql.conf` file allows administrators to control how the database server operates. One of the key parameters in this file is `port`, which defines where the server listens for client connections.

What is the `port` Parameter?

The `port` parameter specifies the TCP port PostgreSQL uses to accept connections. By default, PostgreSQL listens on **port 5432**:

```
port = 5432  # (change requires restart)
```

If you need PostgreSQL to listen on a different port (for example, to avoid conflicts or run multiple instances), you can change this value.

Why Change the Port?

- Run multiple PostgreSQL instances on the same server.
- Avoid conflicts with other services.
- Customize configurations for different environments.
- As part of network security hardening.

How to Change the Port

1️⃣ Edit `postgresql.conf` (location may vary depending on OS; e.g. `/var/lib/pgsql/17/data/postgresql.conf` on Red Hat):

```
sudo vi /var/lib/pgsql/17/data/postgresql.conf
```

2️⃣ Update the port number:

```
port = 5433
```

3️⃣ Restart PostgreSQL to apply:

```
sudo systemctl restart postgresql-17
```

Important Notes

- Changing the port requires restarting the PostgreSQL service.
- Make sure the new port is open in your firewall and cloud security groups.
- Clients must update connection strings:

```
psql -h <server_ip> -p 5433 -U <username> -d <database>
```

1.3 Edit `pg_hba.conf` — Host-Based Access Control

This file defines *who* can connect, from *where*, and *how* they authenticate.

Open the file:

```
sudo vi /var/lib/pgsql/17/data/pg_hba.conf
```

```
cat /var/lib/pgsql/17/data/pg_hba.conf

# TYPE  DATABASE        USER            ADDRESS                 METHOD
```

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
[root@node1 ~]#
```

Add entries to allow remote clients.

Example 1 — Allow one specific client IP:

```
host    all    all    0.0.0.0/0    md5
```

```
cat /var/lib/pgsql/17/data/pg_hba.conf
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             0.0.0.0/0            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
[root@node1 ~]#
[root@node1 ~]#
```

⚠ Important Note

- While `md5` provides basic password security, exposing your database to `0.0.0.0/0` is **not safe for production** because it allows global internet access.

Best practice:

- Use more restrictive IP ranges (e.g. `192.168.1.0/24` for your internal network).
- Use `scram-sha-256` authentication for stronger security.
- Always secure PostgreSQL with a firewall or VPN.

Example 2 — Allow an entire subnet:

```
host    all    all    192.168.32.131/24    md5
```

Explanation:

- `host` → allows TCP/IP connections.
- `all` → allows all databases.
- `all` → allows all users.
- `IP/subnet` → defines allowed client IP addresses.
- `md5` → requires password authentication.

✅ **Warning:** Avoid using `trust` in production as it allows passwordless access.

🔧 Step 2 — Open Firewall Ports on Red Hat Linux

PostgreSQL listens on TCP port `5432` by default. Red Hat Linux uses **firewalld** as its firewall manager.

2.1 Verify that firewalld is active:

```
sudo systemctl status firewalld
```

2.2 Open port 5432 for PostgreSQL:

```
sudo firewall-cmd --permanent --add-port=5432/tcp
```

2.3 Apply firewall changes:

```
sudo firewall-cmd --reload
```

2.4 Verify:

```
sudo firewall-cmd --list-ports
```

✅ **Best Practice:**

- Only open port 5432 to trusted networks.
- For public clouds, also configure AWS Security Groups, Azure NSGs, or GCP firewall rules appropriately.

🔧 Step 3 — Restart PostgreSQL Service

PostgreSQL must be restarted to apply configuration changes.

```
sudo systemctl restart postgresql-17
```

Verify the service status:

```
sudo systemctl status postgresql-17
```

✅ **Tip:** Always check logs in case PostgreSQL fails to restart:

```
sudo journalctl -xeu postgresql-17
```

🔧 Step 4 — Test Remote Connection

Once configured, test the connection from a remote machine using:

```
psql -h <PostgreSQL_server_IP> -p 5432 -U <username> -d <database>
```

Example:

```
psql -h 192.168.32.131 -p 5432 -U myuser -d mydatabase
```

- You'll be prompted for your PostgreSQL user's password.
- On success, you'll get the psql prompt:

```
[postgres@node1 ~]$ psql -h 192.168.32.131 -p 5432 -U postgres -d postgres
Password for user postgres:
psql (17.5)
Type "help" for help.

postgres=#
postgres=#
```

🛡️ Step 5 — Secure Your Remote Connections (Highly Recommended)

Simply opening PostgreSQL to remote clients can expose you to serious security risks unless properly secured.

Best Practice Description Use SSL/TLS Encrypt all client-server traffic using certificates. Restrict IP Access Allow only known trusted IPs/subnets in both firewalls and `pg_hba.conf` . Use Strong Passwords Enforce complex, unique passwords for all database roles. Enable Logging Enable detailed connection logging for auditability. Use VPN Access PostgreSQL via a private VPN whenever possible. Rotate Credentials Periodically rotate user credentials and revoke unused roles.

✅ PostgreSQL fully supports SSL encryption. You can configure `ssl = on` in `postgresql.conf` and manage certificates using self-signed or enterprise PKI solutions.

🔎 Common Problems and Fixes

Problem Solution `psql: could not connect to server` Verify `listen_addresses` , firewalls, and PostgreSQL status. Authentication failed Ensure correct username/password and matching `pg_hba.conf` entry. Timeout Verify network connectivity and firewall rules. Permission denied on firewall Check cloud security group configurations. Port still blocked Use `ss -plunt

✅ Full Quick Checklist

✓ Update `postgresql.conf` — `listen_addresses`
✓ Update `pg_hba.conf` — allow specific IPs
✓ Open firewall port 5432 via firewalld
✓ Restart PostgreSQL service
✓ Test remote connection
✓ Secure connections with firewall + SSL

🔮 Bonus: Automate Everything with Ansible

For repeatable deployments across multiple servers, you can automate remote PostgreSQL configuration using tools like:

- **Ansible**
- **Terraform + Ansible combo (for cloud-native automation)**
- **Bash provisioning scripts**

> ✅ This allows you to deploy hardened, fully configured PostgreSQL servers at scale with minimal human error.

🔧 Real-World Use Case Scenario

Scenario:
A mid-sized SaaS company is running PostgreSQL 17 on Red Hat Linux virtual machines inside AWS private subnets. Application servers reside in separate private subnets. They:

- Open port 5432 internally using AWS Security Groups.
- Restrict `pg_hba.conf` to only allow connections from application server subnets.
- Use SSL for encrypted traffic.
- Automate PostgreSQL configuration with Ansible.
- Monitor logs for unauthorized connection attempts.

Result:
✅ Secure, scalable, remote PostgreSQL connections with full control over compliance, performance, and auditability.

🚀 Conclusion: PostgreSQL Remote Access on Red Hat — Powerful, Flexible & Secure

With careful configuration, **PostgreSQL 17 can be securely accessed remotely** — enabling modern cloud-native, multi-tier, and microservices architectures to

flourish.

- ✅ Red Hat Linux gives you full control over PostgreSQL configuration.
- ✅ `postgresql.conf` + `pg_hba.conf` provide fine-grained access control.
- ✅ firewalld ensures proper network segmentation.
- ✅ SSL encryption secures data-in-transit.

By following this guide, you not only enable remote access — you do it **safely** and **professionally**.

👉 Follow me on Medium to continue the full PostgreSQL deep dive!

🔗 Let's Connect!

If you enjoyed this post or would like to connect professionally, feel free to reach out to me on LinkedIn:
👉 Jeyaram Ayyalusamy

I regularly share content on **PostgreSQL, database administration, cloud technologies, and data engineering.** Always happy to connect, collaborate, and discuss ideas!

Database Postgresql Sql Linux

Following

Written by Jeyaram Ayyalusamy

76 followers · 2 following

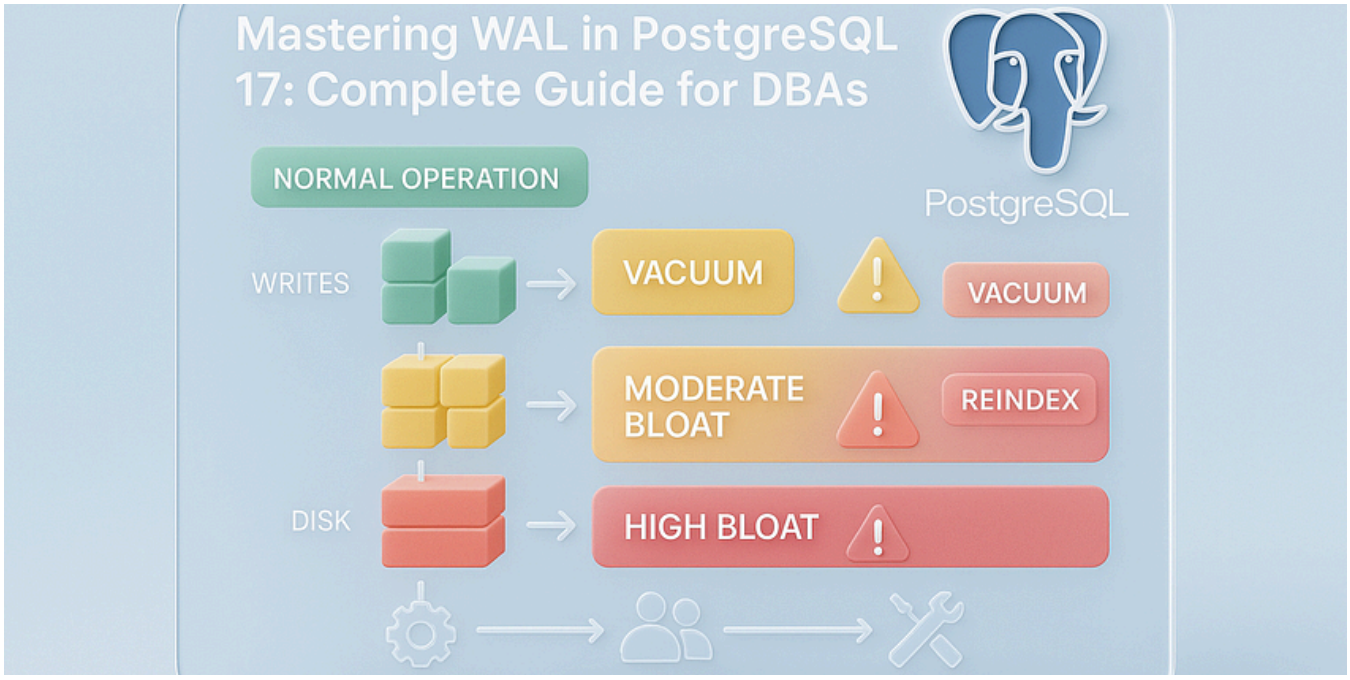
Oracle DBA | AWS Certified | 18+ yrs in DB Admin, RAC, GoldenGate, RDS, MySQL, PostgreSQL | Author of MySQL & RDS books | Cloud & Performance Expert

No responses yet

Gvadakte

What are your thoughts?

More from Jeyaram Ayyalusamy

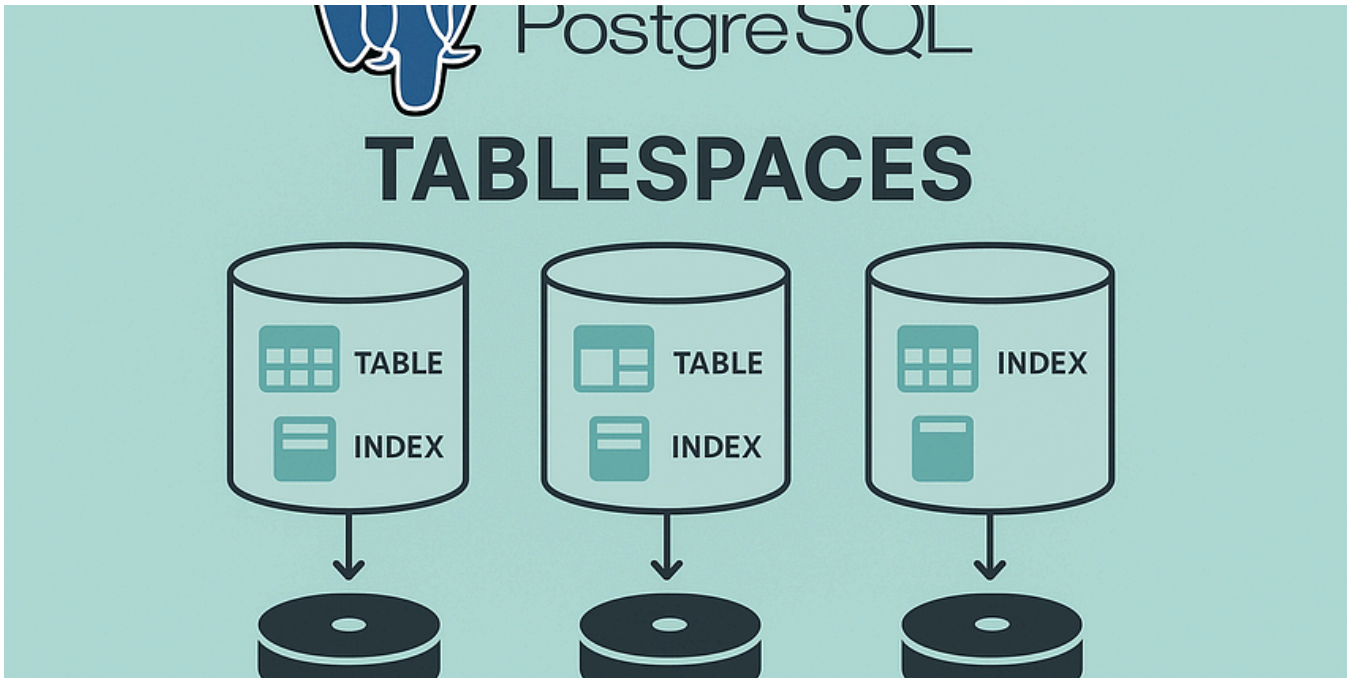


Jeyaram Ayyalusamy

Understanding and Managing Bloating in PostgreSQL: A Complete Guide for DBAs

PostgreSQL is a powerful, reliable, and feature-rich open-source relational database system. It's praised for its extensibility, ACID...

Jun 25 52

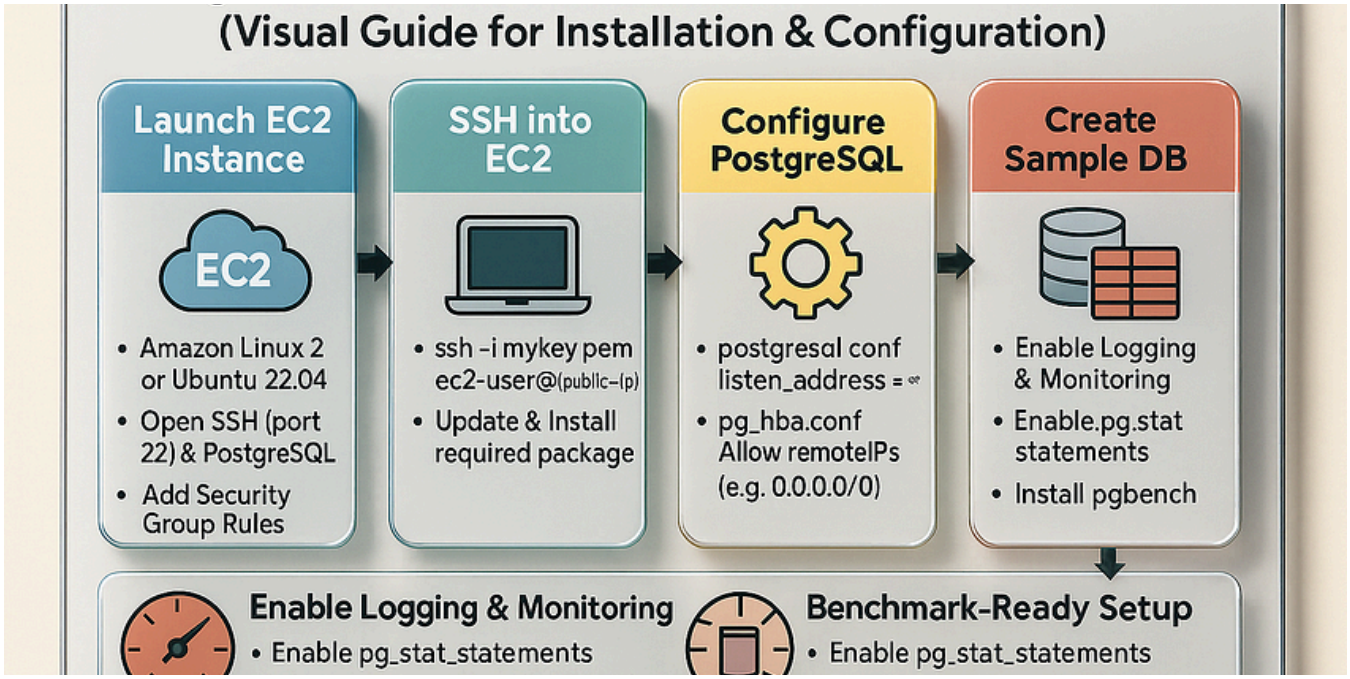


Jeyaram Ayyalusamy

PostgreSQL Tablespaces Explained: Complete Guide for PostgreSQL 17 DBAs

PostgreSQL offers a powerful feature called tablespaces, allowing database administrators to take control of how and where data is...

Jun 12 8



Jeyaram Ayyalusamy

PostgreSQL 17 on AWS EC2—Full Installation & Configuration Walkthrough

Setting up PostgreSQL 17 on AWS EC2 might seem complex at first—but once you break down each component, it becomes an efficient and...

1d ago 50

Jeyaram Ayyalusamy

How to Install PostgreSQL 17 on Red Hat, Rocky, AlmaLinux, and Oracle Linux (Step-by-Step Guide)

PostgreSQL 17 is the latest release of one of the world's most advanced open-source relational databases. If you're using a Red Hat-based...

Jun 3

See all from Jeyaram Ayyalusamy

Recommended from Medium

Azlan Jamal

Stop Using SERIAL in PostgreSQL: Here's What You Should Do Instead

In PostgreSQL, there are several ways to create a PRIMARY KEY for an id column, and they usually involve different ways of generating the...

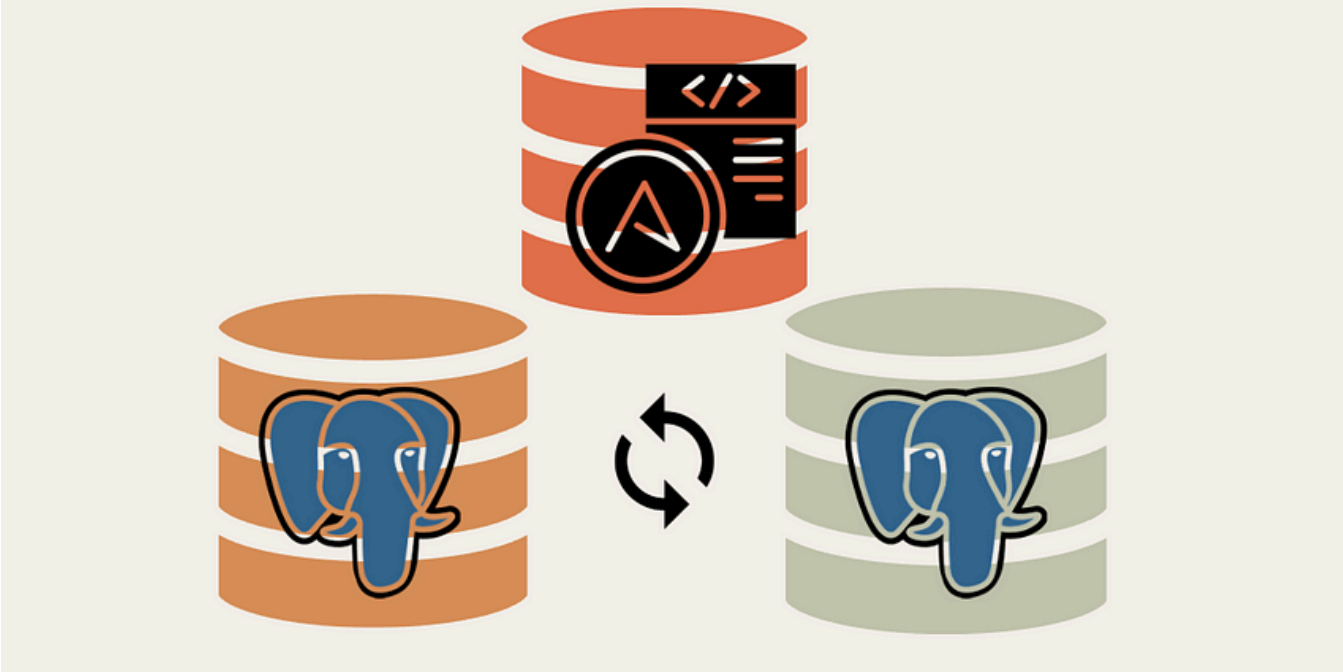
Jul 12 33

Rizqi Mulki

Full-Text Search in PostgreSQL: Better Than You Think

Why developers are abandoning Elasticsearch for PostgreSQL's built-in search—and how it handles 10M records without breaking a sweat

5d ago 2

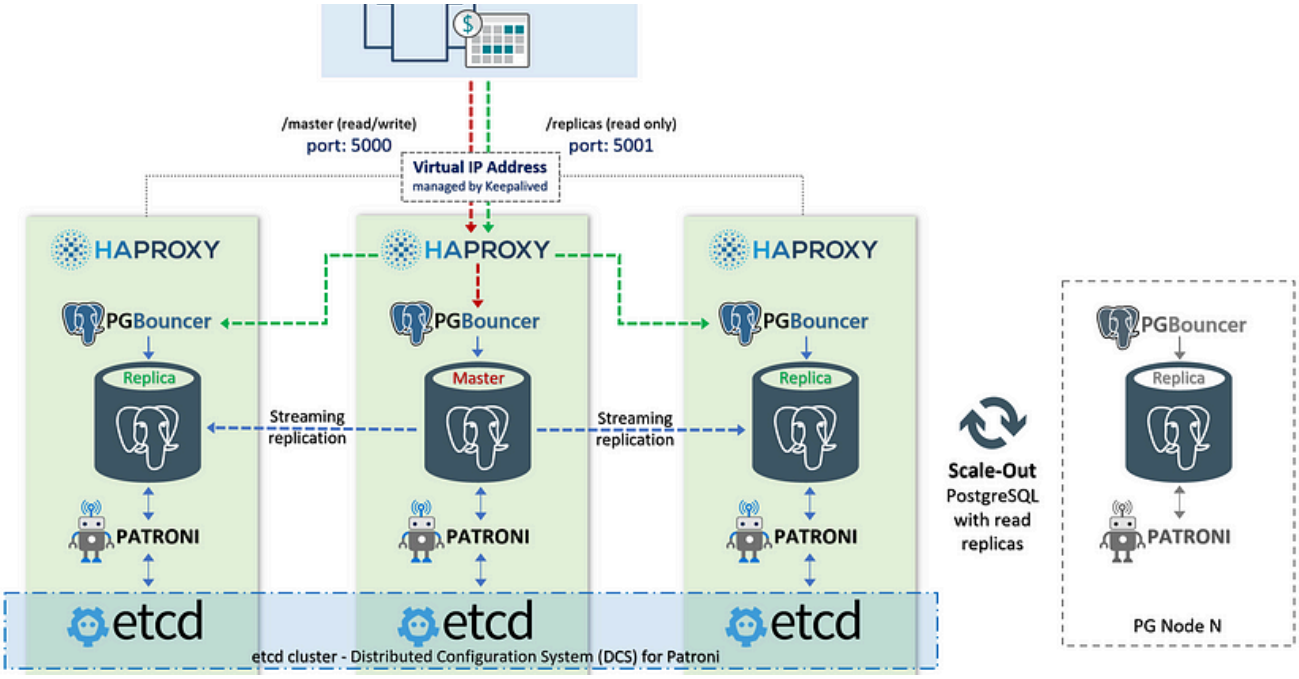


Oz

Automating PostgreSQL Streaming Replication Setup with Ansible

Setting up PostgreSQL streaming replication manually can be a tedious and error-prone task—involving installing PostgreSQL, configuring...

Jul 8 58

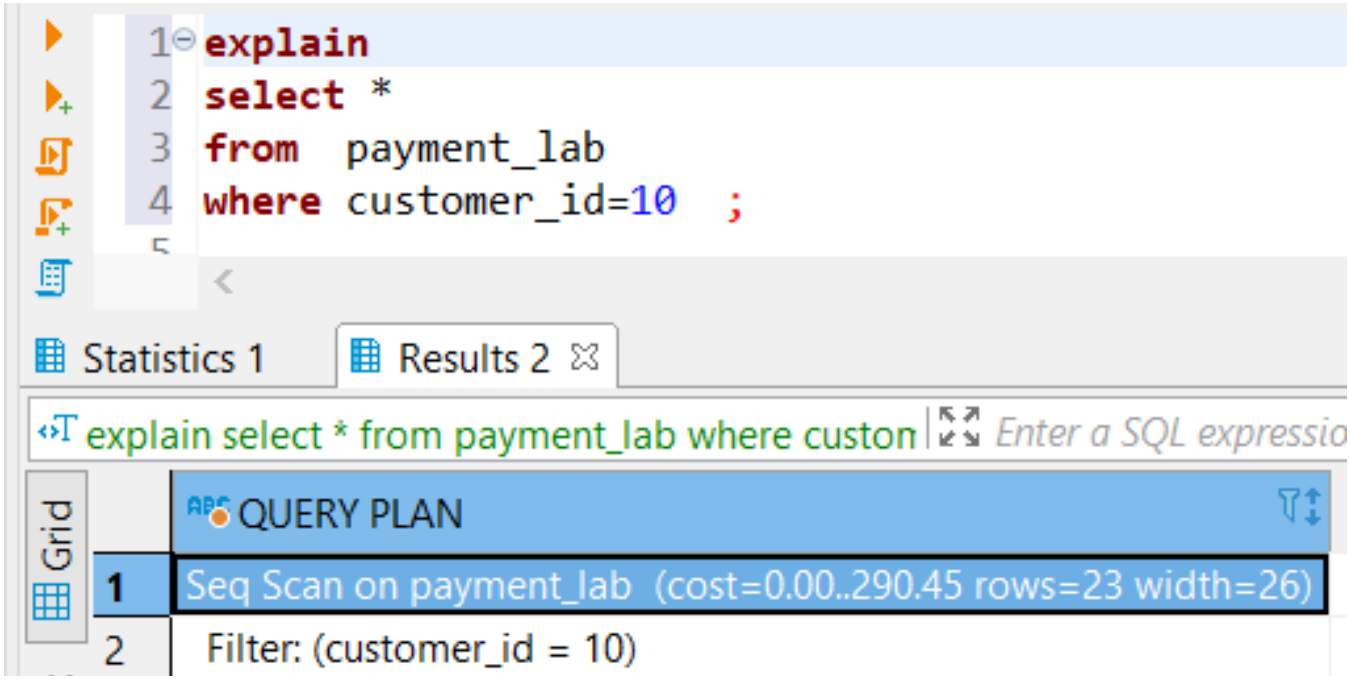


Kanat Akylson

How to Deploy a High-Availability PostgreSQL Cluster in 5 Minutes Using Ansible and Patroni

This tutorial shows how to spin up a production-grade HA PostgreSQL cluster in just 5 minutes using m2y ready-made GitHub repository with...

Jun 9 65 1

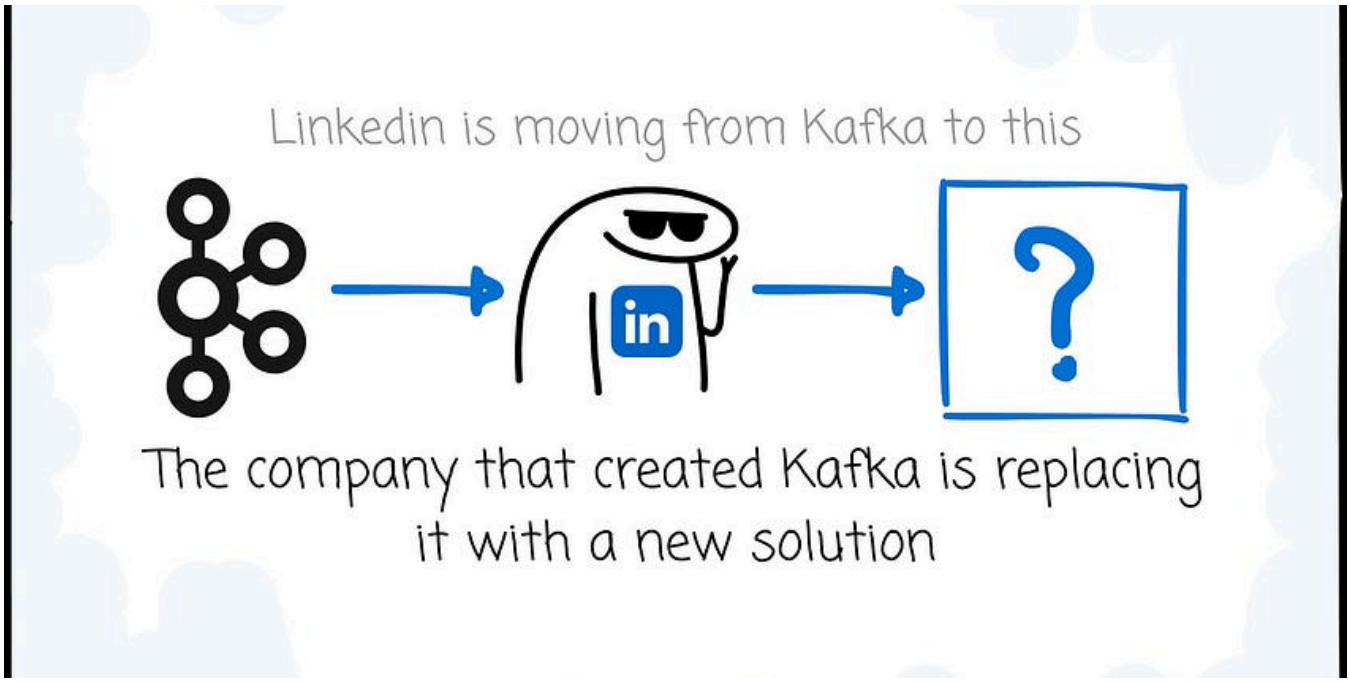


Muhammet Kurtoglu

Postgresql Query Performance Analysis

SQL tuning is critically important for ensuring database performance, reliability, and scalability. In most cases, performance issues in a...

6d ago 10



In Data Engineer Things by Vu Trinh

The company that created Kafka is replacing it with a new solution

How did LinkedIn build Northguard, the new scalable log storage

Jul 17 330 6

See more recommendations