Building a High Availability PostgreSQL Cluster with Patroni: Step-by-Step Guide

8 min read · May 29, 2025

Jeyaram Ayyalusamy Following

This will follow the standard instructional structure with step-by-step configuration based on your nodes: `node1`, `node2`, `etcdnode`, and `haproxynode`.

Introduction

High availability (HA) in PostgreSQL is essential for ensuring uninterrupted database services. Patroni is a popular solution that leverages etcd or Consul for distributed consensus and HA failover, managing PostgreSQL clusters with automatic leader election and replication management.

In this guide, we'll walk through setting up an HA PostgreSQL cluster using Patroni , etcd, and HAProxy on a four-node architecture:

- `**node1**` and `**node2**` — PostgreSQL database nodes
- `**etcdnode**` — etcd cluster node
- `**haproxynode**` — HAProxy load balancer node

cat /etc/hosts

```
#127.0.0.1 localhost
#127.0.1.1 ubuntu2204.localdomain
#127.0.0.1 ubuntu2204.localdomain
#127.0.2.1 node1 node1
```

```
192.168.32.130 node1
192.168.32.131 node2
192.168.32.140 etcdnode
192.168.32.135 haproxynode
```

1. Configure the etcd Cluster

- Node: etcdnode

Step-by-Step Instructions:

1. **Update the system and install dependencies:**

```
sudo apt update -y
sudo apt install net-tools -y
```

2. Install PostgreSQL tools (needed for Patroni):

```
sudo apt install postgresql postgresql-server-dev-14 -y
```

3. Stop any existing PostgreSQL service:

```
sudo systemctl stop postgresql
```

4. Set hostname for clarity (optional but recommended):

```
sudo hostnamectl set-hostname etcdnode
```

5. Install etcd:

```
sudo apt install -y etcd
```

6. Configure etcd by editing `/etc/default/etcd`:

Ensure the file contains:

```
ETCD_LISTEN_PEER_URLS="http://192.168.32.140:2380"
ETCD_LISTEN_CLIENT_URLS="http://localhost:2379,http://192.168.32.140:2379"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://192.168.32.140:2380"
ETCD_INITIAL_CLUSTER="default=http://192.168.32.140:2380,"
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.32.140:2379"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

7. Restart and validate etcd:

```
sudo systemctl restart etcd
sudo systemctl status etcd
curl http://192.168.32.140:2380/members
```

```
root@etcdnode:~#  sudo systemctl restart etcd
root@etcdnode:~# sudo systemctl status etcd
● etcd.service - etcd - highly-available key value store
     Loaded: loaded (/lib/systemd/system/etcd.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2025-05-29 00:39:20 UTC; 5s ago
       Docs: https://etcd.io/docs
             man:etcd
   Main PID: 2463 (etcd)
      Tasks: 8 (limit: 4558)
     Memory: 47.5M
        CPU: 981ms
     CGroup: /system.slice/etcd.service
             └─2463 /usr/bin/etcd

May 29 00:39:20 etcdnode etcd[2463]: 8e9e05c52164694d became candidate at term 5
May 29 00:39:20 etcdnode etcd[2463]: 8e9e05c52164694d received MsgVoteResp from 8e9e05c52164694d at term 5
May 29 00:39:20 etcdnode etcd[2463]: 8e9e05c52164694d became leader at term 5
May 29 00:39:20 etcdnode etcd[2463]: raft.node: 8e9e05c52164694d elected leader 8e9e05c52164694d at term 5
May 29 00:39:20 etcdnode etcd[2463]: published {Name:etcdnode ClientURLs:[http://192.168.32.140:2379]} to cluster cdf818194e3a8c32
May 29 00:39:20 etcdnode etcd[2463]: ready to serve client requests
May 29 00:39:20 etcdnode etcd[2463]: ready to serve client requests
May 29 00:39:20 etcdnode etcd[2463]: serving insecure client requests on 192.168.32.140:2379, this is strongly discouraged!
May 29 00:39:20 etcdnode etcd[2463]: serving insecure client requests on 127.0.0.1:2379, this is strongly discouraged!
May 29 00:39:20 etcdnode systemd[1]: Started etcd - highly-available key value store.
root@etcdnode:~#
root@etcdnode:~#
```

```
root@etcdnode:~#
root@etcdnode:~# curl http://192.168.32.140:2380/members
[{"id":10276657743932975437,"peerURLs":["http://localhost:2380"],"name":"etcdnode","clientURLs":["http://192.168.32.140:2379"]}]
root@etcdnode:~#
```

2. Set Up Patroni on PostgreSQL Node (`node1`)

- Node: `node1`

This node will run one instance of PostgreSQL managed by Patroni, which will coordinate with etcd for cluster state and participate in leader election.

Step-by-Step Instructions:

1. **Update the system and install base tools:**

```
sudo apt update -y
 sudo apt install net-tools
```

2. Install PostgreSQL and development libraries:

```
sudo apt install postgresql postgresql-server-dev-14
```

3. Stop the default PostgreSQL service:

```
sudo systemctl stop postgresql
```

4. Link PostgreSQL binaries into the system path:

```
sudo ln -s /usr/lib/postgresql/14/bin/* /usr/sbin/
```

5. Install Python and required libraries:

```
sudo apt -y install python3 python3-pip
 sudo apt install python3-testresources
```

```
sudo pip3 install --upgrade setuptools
sudo pip3 install psycopg2 patroni python-etcd
```

6. Create and configure Patroni data directory:

```
sudo mkdir -p /data/patroni
sudo chown postgres:postgres /data/patroni
sudo chmod 700 /data/patroni
```

7. Create the Patroni configuration file `/etc/patroni.yml`:

```
scope: postgres
namespace: /db/
name: node1
restapi:
    listen: 192.168.32.130:8008
    connect_address: 192.168.32.130:8008
etcd:
    host: 192.168.32.140:2379
bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
  initdb:
  - encoding: UTF8
  - data-checksums
  pg_hba:
  - host replication replicator 127.0.0.1/32 md5
  - host replication replicator 192.168.32.130/0 md5
  - host replication replicator 192.168.32.131/0 md5
  - host all all 0.0.0.0/0 md5
  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb
postgresql:
```

```
  listen: 192.168.32.130:5432
  connect_address: 192.168.32.130:5432
  data_dir: /data/patroni
  pgpass: /tmp/pgpass
  authentication:
    replication:
      username: replicator
      password: admin@123
    superuser:
      username: postgres
      password: admin@123
  parameters:
      unix_socket_directories: '.'
tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
```

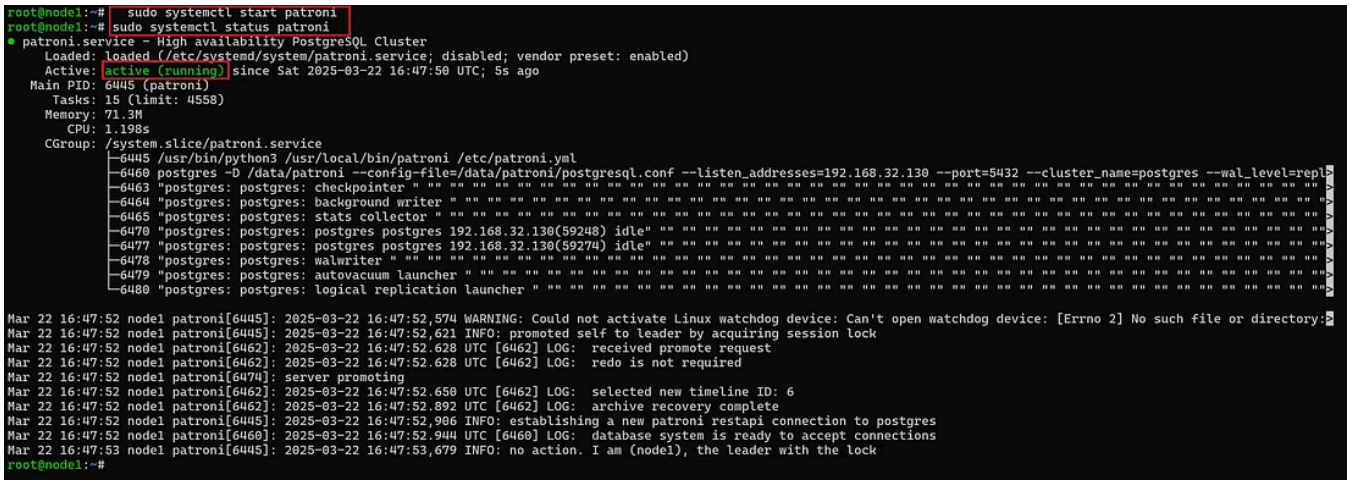
8. Create a systemd service file for Patroni:

sudo vi /etc/systemd/system/patroni.service

```
[Unit]
Description=High availability PostgreSQL Cluster
After=syslog.target network.target
[Service]
Type=simple
User=postgres
Group=postgres
ExecStart=/usr/local/bin/patroni /etc/patroni.yml
KillMode=process
TimeoutSec=30
Restart=no
[Install]
WantedBy=multi-user.targ
```

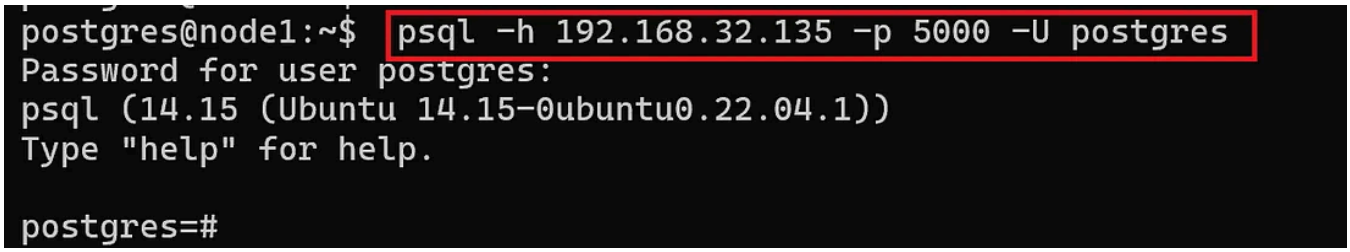
9. Start and verify Patroni:

```
sudo systemctl start patroni
sudo systemctl status patroni
```



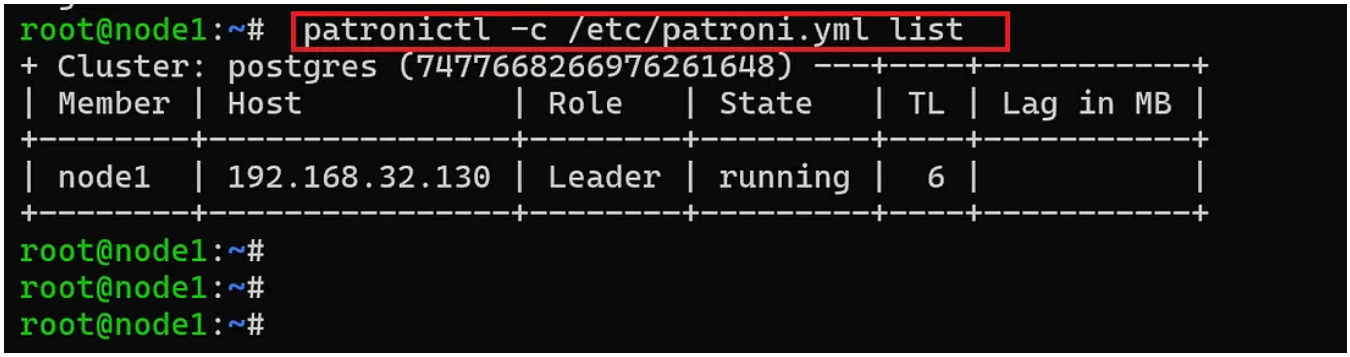
10. Test PostgreSQL access via HAProxy (optional at this stage):

```
psql -h 192.168.32.135 -p 5000 -U postgres
```



11. Check cluster status:

```
patronictl -c /etc/patroni.yml list
```



3. Configure PostgreSQL Node (`node2`) with Patroni

- Node: `node2`

This second PostgreSQL node will join the Patroni-managed cluster and act as a replica. It shares nearly identical steps to `node1` but must be uniquely identified and correctly networked.

Step-by-Step Instructions:

1. **Update the system and install base tools:**

```
sudo apt update -y
sudo apt install net-tools -y
```

2. Install PostgreSQL and development headers:

```
sudo apt install postgresql postgresql-server-dev-14 -y
```

3. Stop the default PostgreSQL service:

```
sudo systemctl stop postgresql
```

4. Link PostgreSQL binaries into the system path:

```
sudo ln -s /usr/lib/postgresql/14/bin/* /usr/sbin/
```

5. Install Python and required libraries:

```
sudo apt -y install python3 python3-pip
sudo apt install python3-testresources
sudo pip3 install --upgrade setuptools
sudo pip3 install psycopg2 patroni python-etcd
```

6. Create and configure Patroni data directory:

```
sudo mkdir -p /data/patroni
sudo chown postgres:postgres /data/patroni
sudo chmod 700 /data/patroni
```

7. Create the Patroni configuration file /etc/patroni.yml :

Ensure the following parameters are set (customize with your IPs and cluster details):

```
scope: postgres
namespace: /db/
name: node2
restapi:
    listen: 192.168.32.131:8008
    connect_address: 192.168.32.131:8008
etcd:
    host: 192.168.32.140:2379
bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
  initdb:
  - encoding: UTF8
  - data-checksums
  pg_hba:
  - host replication replicator 127.0.0.1/32 md5
  - host replication replicator 192.168.32.130/0 md5
  - host replication replicator 192.168.32.131/0 md5
  - host all all 0.0.0.0/0 md5
  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb
postgresql:
```

```
  listen: 192.168.32.131:5432
  connect_address: 192.168.32.131:5432
  data_dir: /data/patroni
  pgpass: /tmp/pgpass
  authentication:
    replication:
      username: replicator
      password: admin@123
    superuser:
      username: postgres
      password: admin@123
  parameters:
      unix_socket_directories: '.'
tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
```

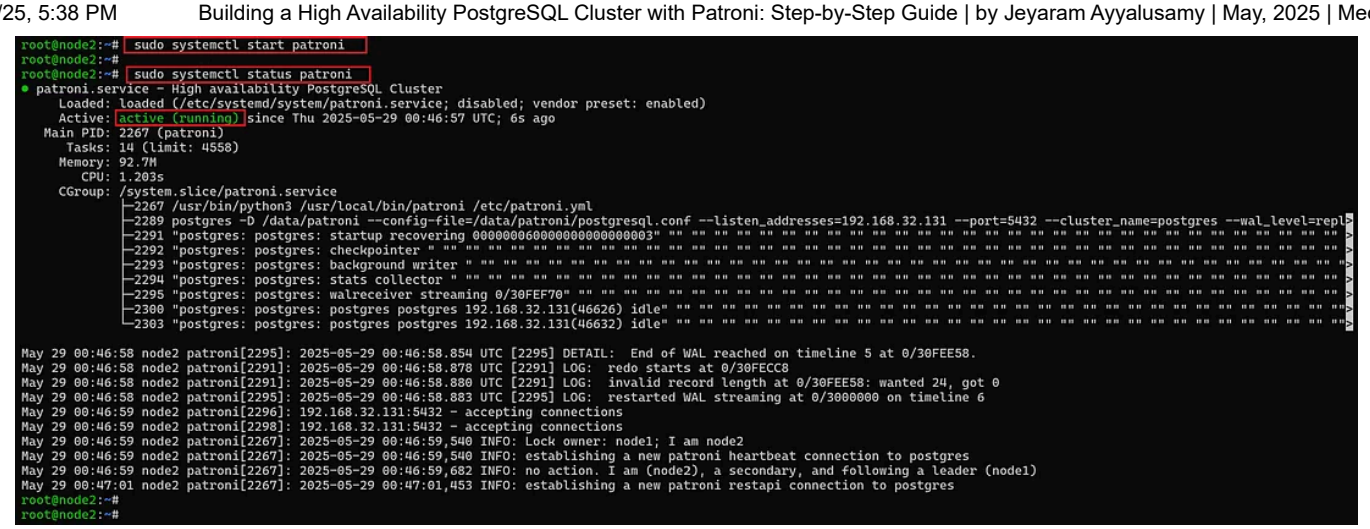
8. Create a systemd service file for Patroni:

sudo vi /etc/systemd/system/patroni.service

```
[Unit]
Description=High availability PostgreSQL Cluster
After=syslog.target network.target
[Service]
Type=simple
User=postgres
Group=postgres
ExecStart=/usr/local/bin/patroni /etc/patroni.yml
KillMode=process
TimeoutSec=30
Restart=no
[Install]
WantedBy=multi-user.targ
```

9. Start and verify Patroni:

```
sudo systemctl start patroni
sudo systemctl status patroni
```



10. Test PostgreSQL access via HAProxy (optional at this stage):

```
psql -h 192.168.32.135 -p 5000 -U postgres
```

11. Check cluster status:

```
patronictl -c /etc/patroni.yml list
```

4. Configure HAProxy for PostgreSQL Cluster

Node: `haproxynode`

HAProxy serves as a TCP load balancer that routes client connections to the current Patroni leader, providing seamless failover and high availability.

Step-by-Step Instructions:

1. **Update the system and install HAProxy:**

```
sudo apt update
sudo apt install net-tools
sudo apt -y install haproxy
```

2. Edit the HAProxy configuration file:

Open /etc/haproxy/haproxy.cfg and append the following configuration:

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd lis
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-version=2.
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
        ssl-default-bind-ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SH
        ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

global
        maxconn 100
        log     127.0.0.1 local2
defaults
        log global
        mode tcp
        retries 2
        timeout client 30m
        timeout connect 4s
        timeout server 30m
        timeout check 5s
listen stats
    mode http
```

```
    bind *:7000
    stats enable
    stats uri /
listen postgres
    bind *:5000
    option httpchk
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server node1 192.168.32.130:5432 maxconn 100 check port 8008
    server node2 192.168.32.131:5432 maxconn 100 check port 8008
```

3. Restart and verify HAProxy:

```
sudo systemctl restart haproxy
sudo systemctl status haproxy
```

4. Test client connection through HAProxy:

From any client or cluster node:

```
psql -h 192.168.32.135 -p 5000 -U postgres
```

Replace 192.168.32.135 with the IP of haproxynode .

5. Access Patroni Cluster Status via REST API

Patroni provides a built-in REST API endpoint on each node that can be queried for health and role information. HAProxy can use this endpoint to determine the leader and ensure traffic is routed only to the primary.

URL: http://192.168.32.135:7000/

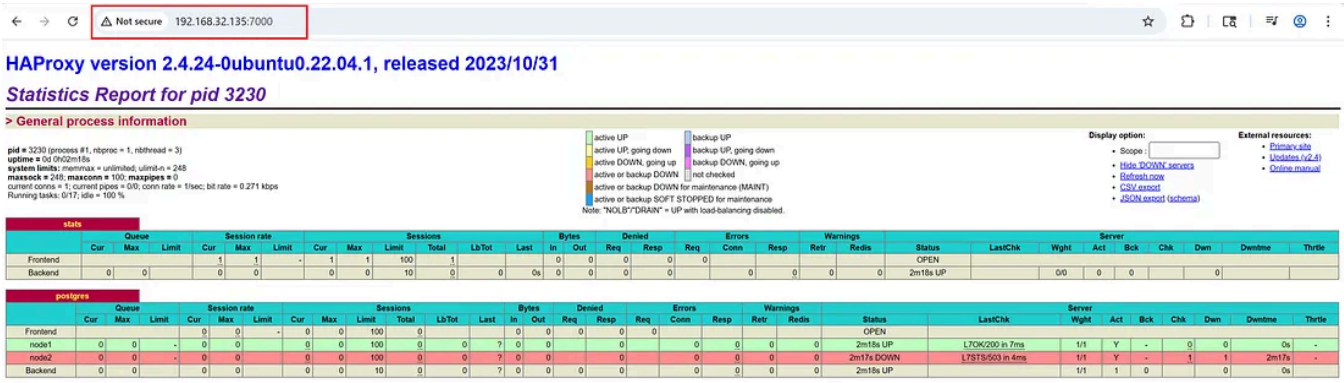
(Replace 192.168.32.135 with the actual IP of the node running the REST API on port 7000)

Use Cases:

- **Monitoring:** You can visit this URL in a browser or use tools like curl to check the current status of a node (e.g., whether it's the leader or replica).
- **HAProxy Health Checks:** This URL is often used as a health-check target to identify the current leader and reroute connections accordingly.

Example Command:

```
curl http://192.168.32.135:7000/
```



🛠️ Patroni's default REST API port is 8008 , but it can be changed in patroni.yml . If you've configured it to run on 7000 , ensure HAProxy and monitoring tools reflect this.

6. Validate PostgreSQL High Availability Access

Once all components (etcd, Patroni, and HAProxy) are configured and running, you can validate the system by connecting to PostgreSQL using both the HAProxy frontend and directly to individual nodes.

Connect via HAProxy (Cluster-Aware Access)

To connect through the HAProxy endpoint, which routes traffic to the current Patroni leader:

```
psql -h 192.168.32.135 -p 5000 -U postgres
```

- **192.168.32.135** is the IP address of the `haproxynode`.
- **5000** is the frontend port configured in HAProxy.
- This connection will always point to the current cluster leader, even if failover occurs.

Connect Directly to Node1

For comparison or diagnostics, connect directly to the PostgreSQL instance on `node1`:

```
psql -h 192.168.32.130 -p 5432 -U postgres
```

- Use this to check node state or for local access when HAProxy isn't involved.

Expected Behavior

- The HAProxy connection will connect only to the current **leader** node.
- If the leader fails, HAProxy will reroute traffic to the **new** leader without requiring client reconfiguration.
- Direct connections (e.g., to `node1`) will fail if that node is not the current leader and cannot accept writes.

```
root@node1:~#
root@node1:~# psql -h 192.168.32.135 -p 5000 -U postgres
Password for user postgres:
psql (14.15 (Ubuntu 14.15-0ubuntu0.22.04.1))
Type "help" for help.

postgres=#
postgres=#
postgres=# \q
root@node1:~#
root@node1:~#
root@node1:~# psql -h 192.168.32.130 -p 5432 -U postgres
Password for user postgres:
psql (14.15 (Ubuntu 14.15-0ubuntu0.22.04.1))
Type "help" for help.

postgres=#
postgres=#
postgres=# \q
root@node1:~#
root@node1:~#
```

7. Test Patroni Cluster Behavior and Failover

Validating Patroni's cluster management ensures that failover and leader election are functioning correctly. These tests use the patronictl command-line utility to inspect cluster state and simulate node failure.

Step-by-Step Cluster Test

1. **Check current cluster status:**

Use patronictl to see the status of all nodes:

```
patronictl -c /etc/patroni.yml list
```

Look for the node with the Leader role.

2. Stop Patroni on the leader node:

```
sudo systemctl stop patroni
```

This simulates a failure on the primary node.

3. Re-check cluster status:

```
patronictl -c /etc/patroni.yml list
```

A new leader should be elected among the remaining available nodes.

4. Restart Patroni on the stopped node:

```
sudo systemctl start patroni
```

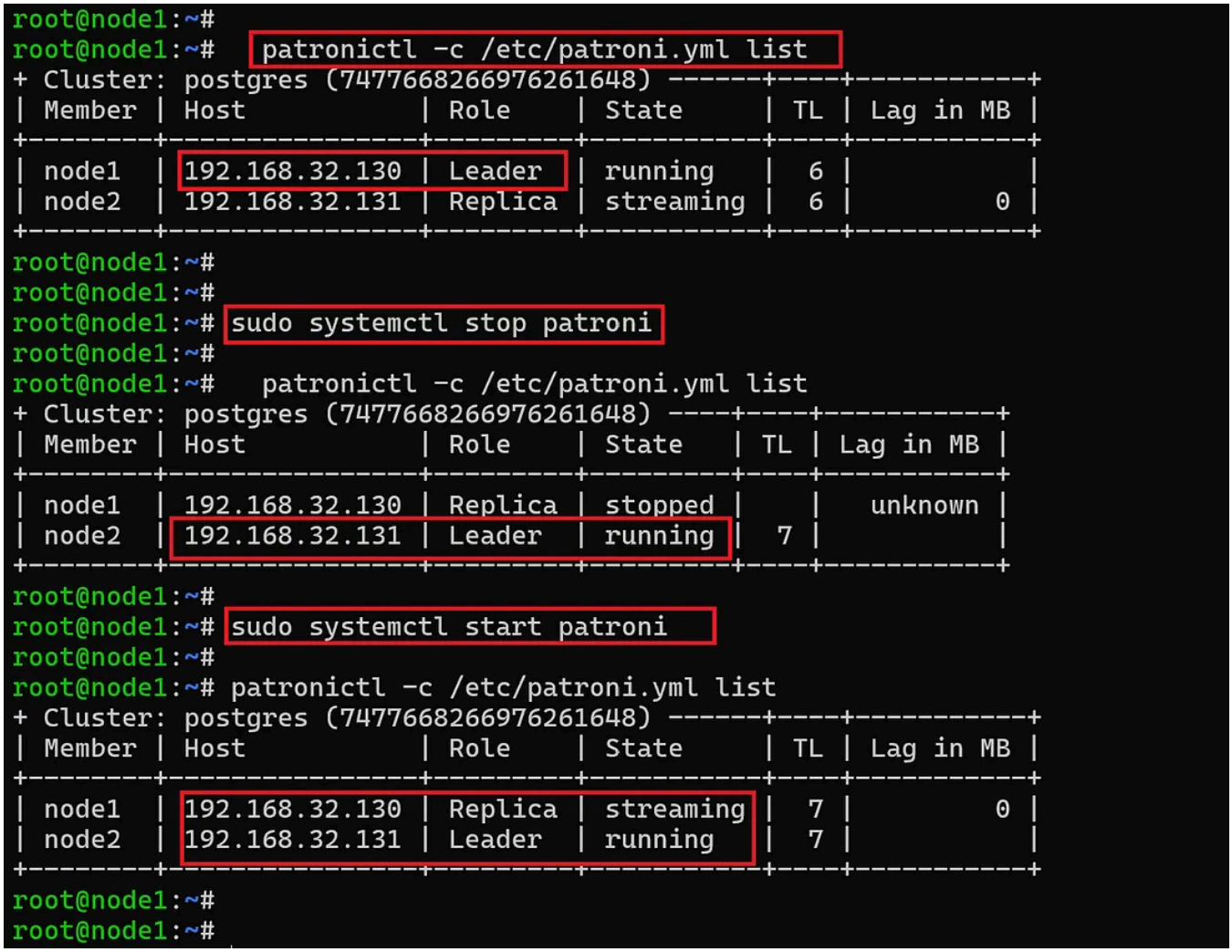
5. Check the cluster status again:

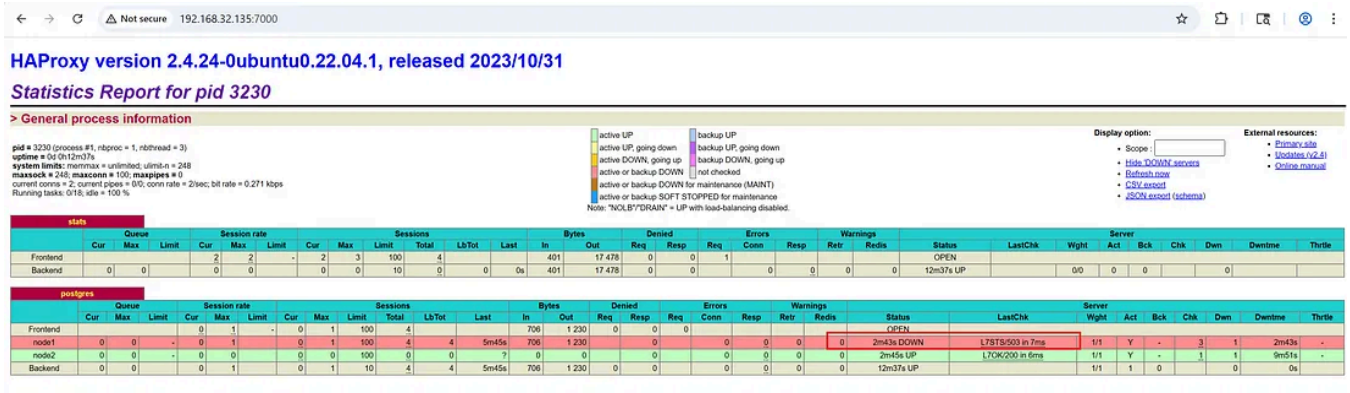
```
patronictl -c /etc/patroni.yml list
```

The restarted node should rejoin the cluster as a replica.

Expected Behavior

- Patroni should automatically promote a replica to leader if the current leader stops.
- When the original leader returns, it should not reclaim leadership unless explicitly configured.
- HAProxy should continue routing to the current leader without client disruption.





Postgresql Database Replication Patroni Haproxy

J

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

Responses (1)

Gvadakte

What are your thoughts?

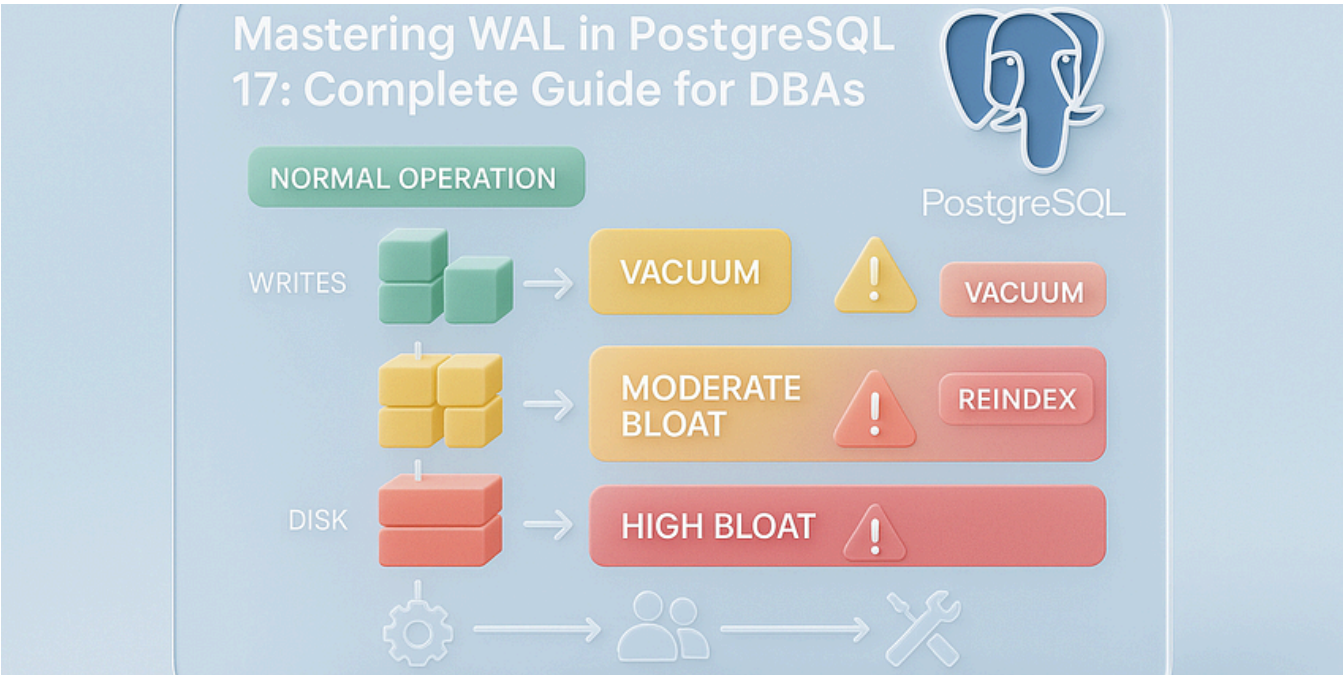
Sarsank
4 days ago

Hi Jeyaram

Thanks for the sharing this document. Could you please share in which OS and version was installed

1 reply Reply

More from Jeyaram Ayyalusamy

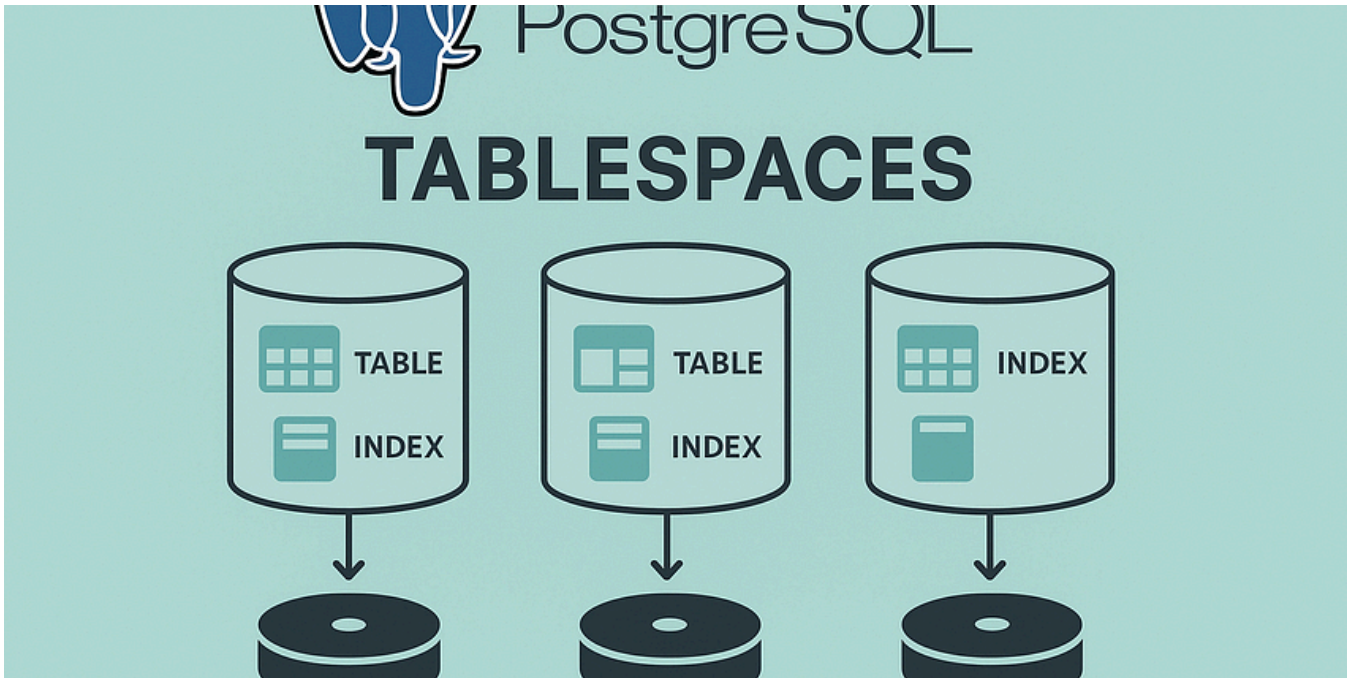


Jeyaram Ayyalusamy

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

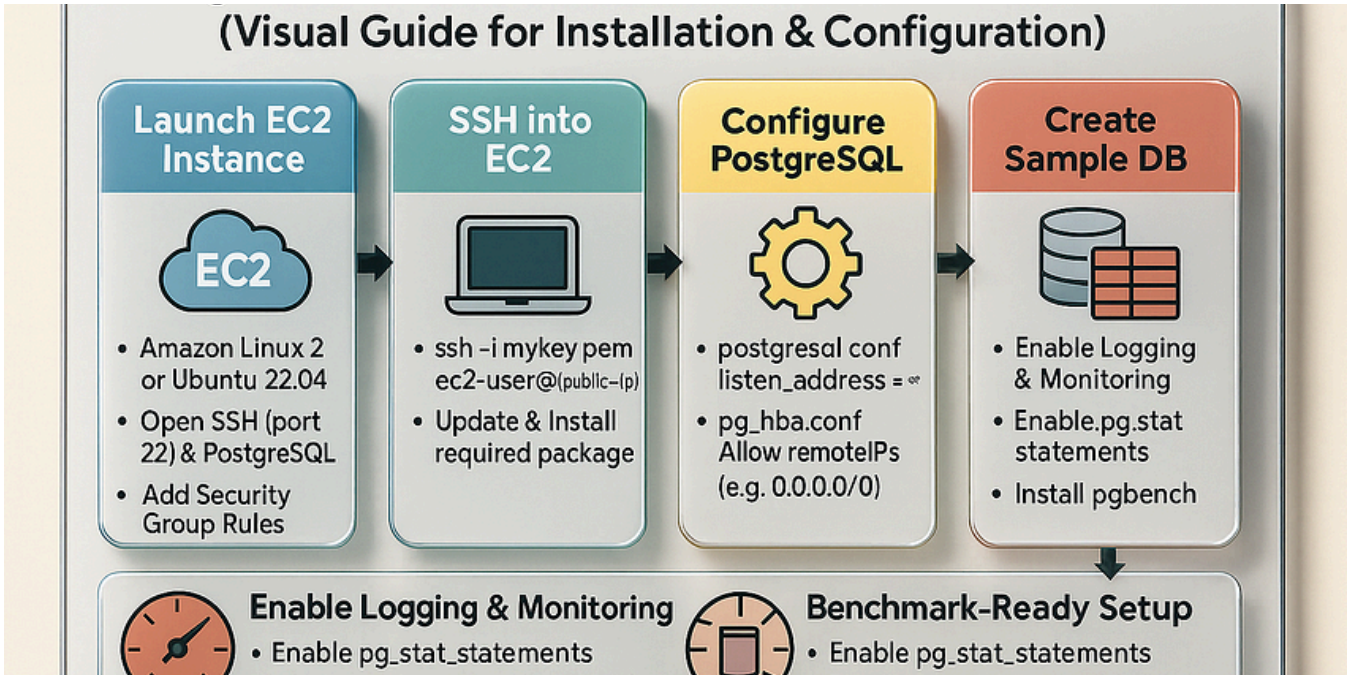


Jeyaram Ayyalusamy

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

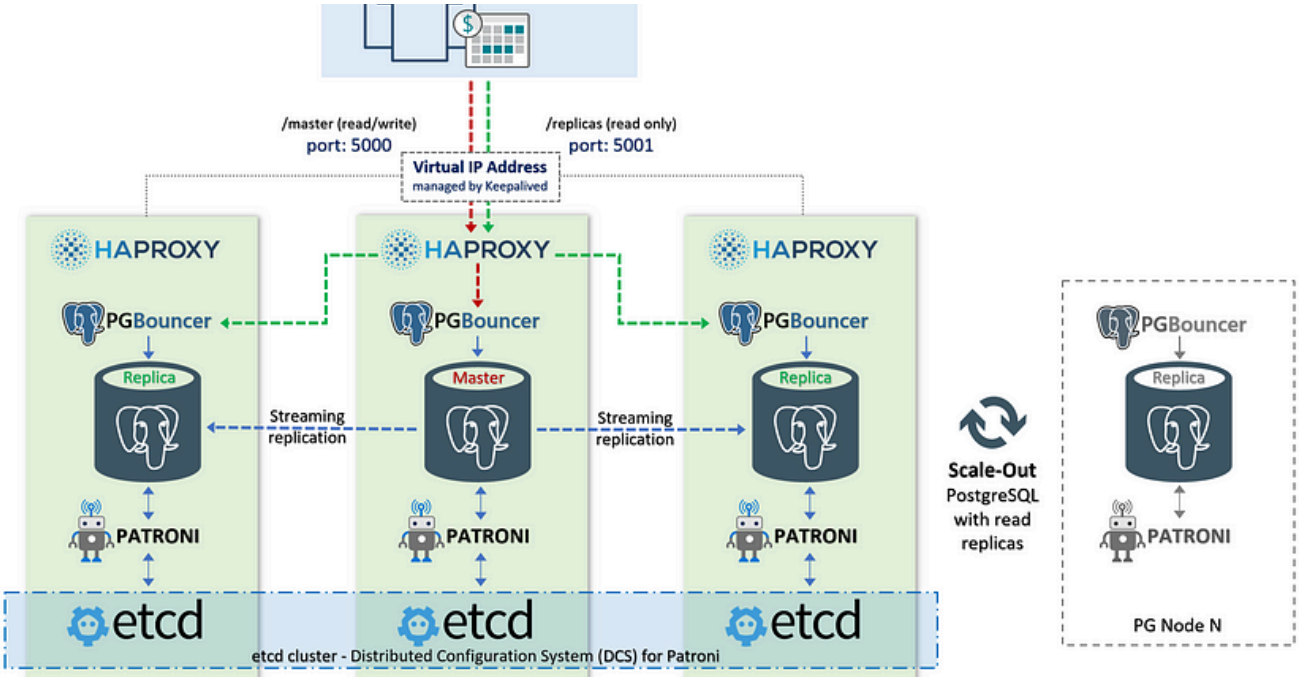
How to Install PostgreSQL 17 on Red Hat, Rocky, AlmaLinux, and Oracle Linux (Step-by-Step Guide)

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



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...

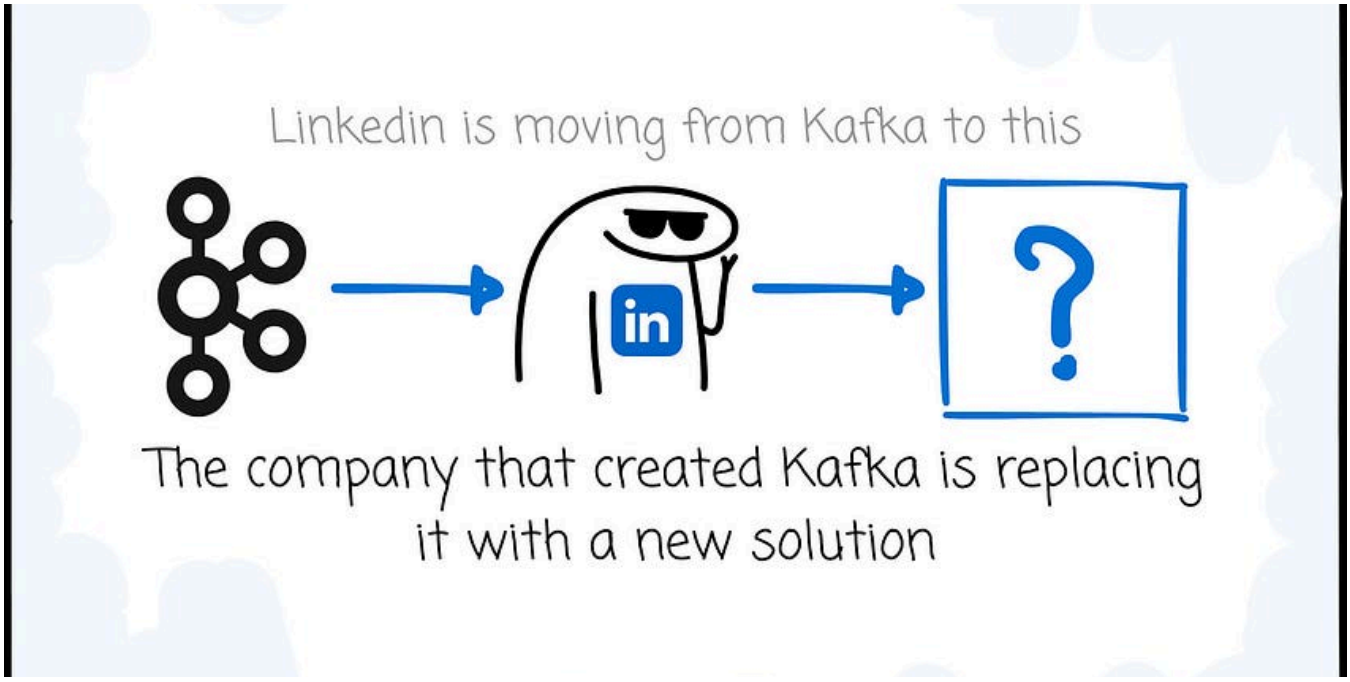
Jun 9 65 1

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

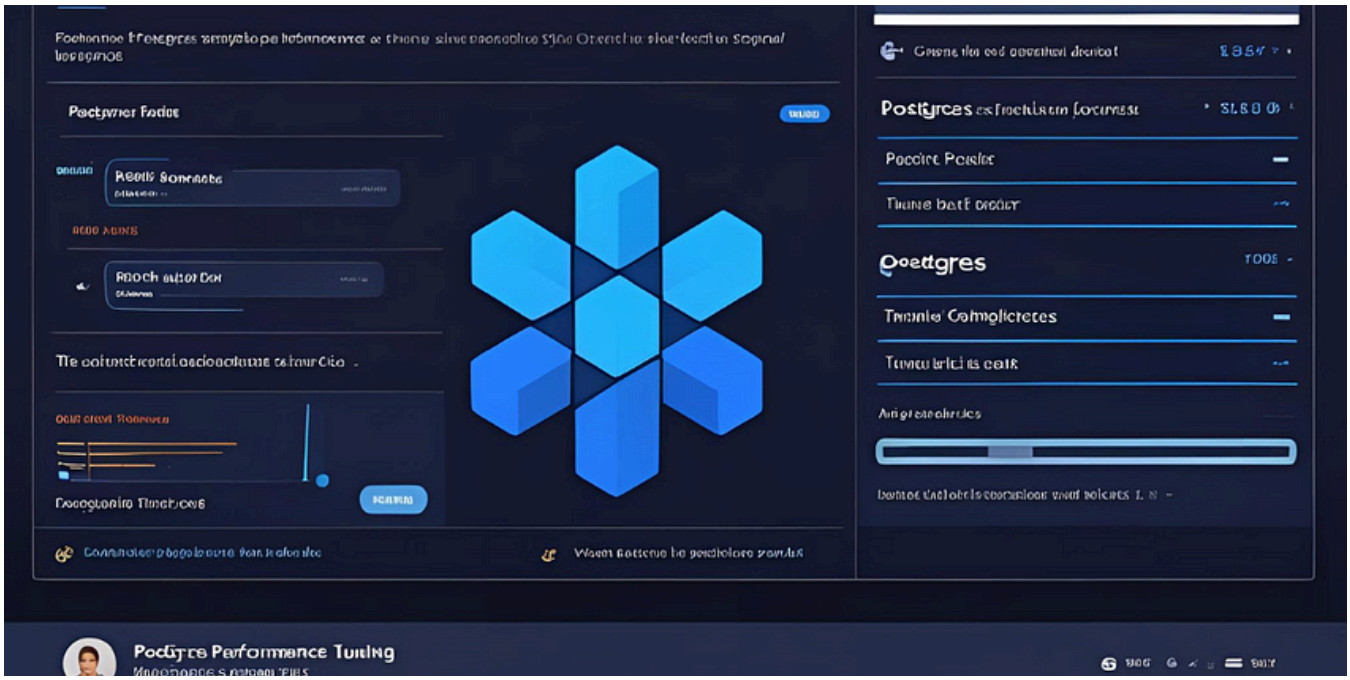


In Data Engineer Things by Vu Trinh

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

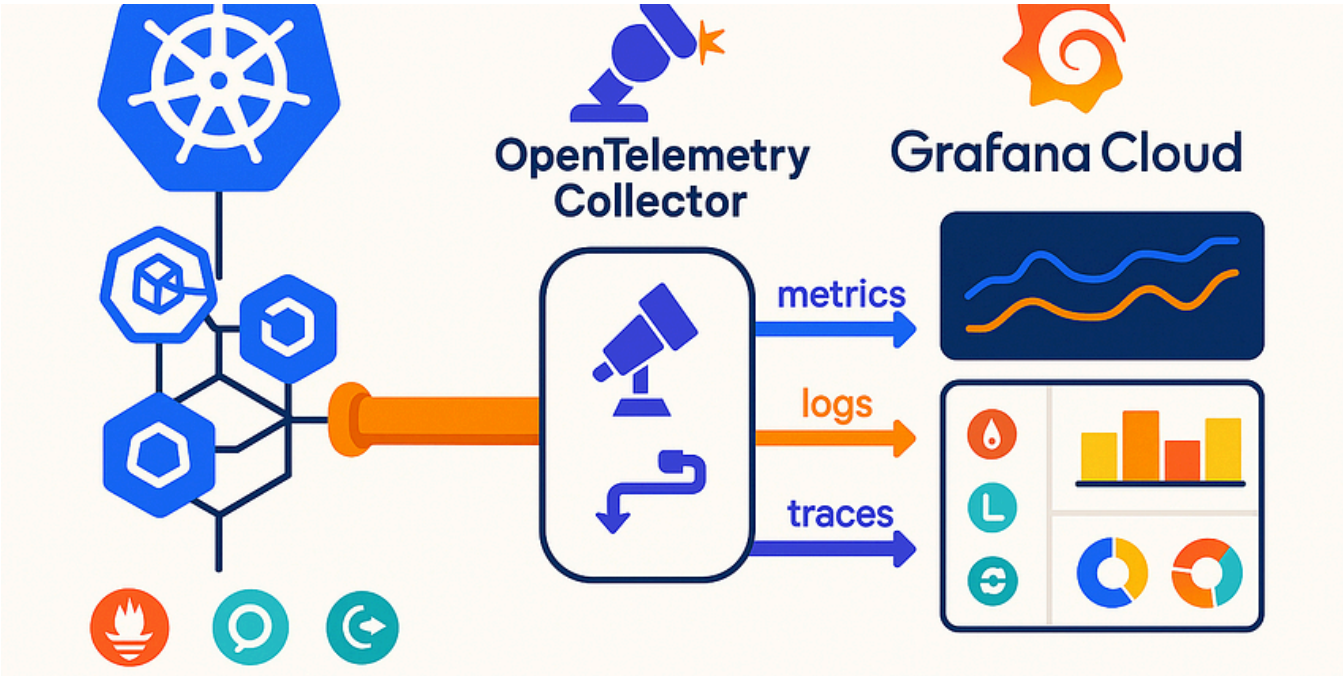


Rizqi Mulki

Postgres Performance Tuning Like a Pro

The advanced techniques that separate database experts from everyone else

6d ago 55

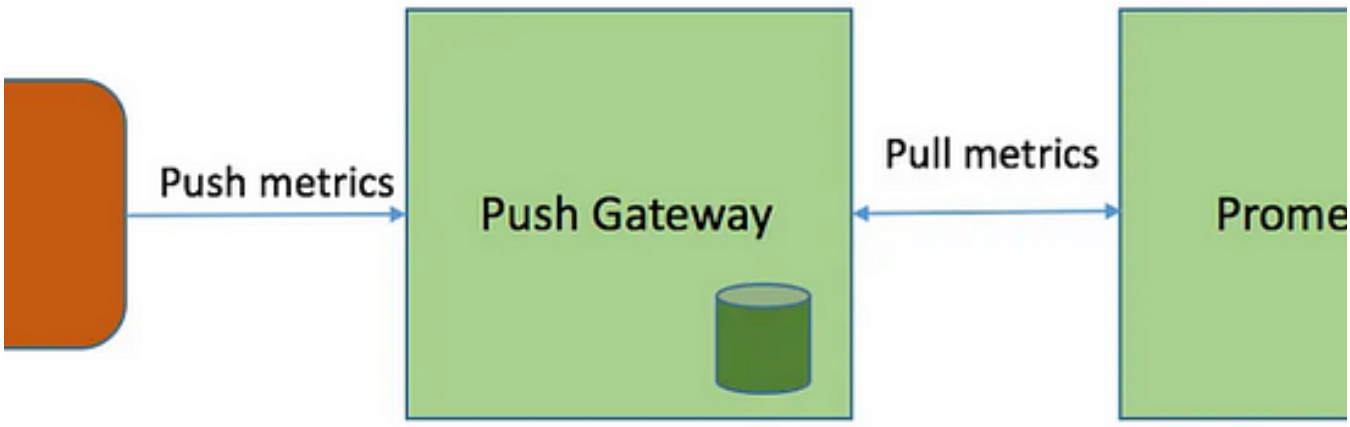


In Dev Genius by using System;

Monitor Your Kube Cluster with Grafana Cloud & OpenTelemetry (with Terraform, of course!)

🚀 Intro

5d ago 4



Think, Write, Repeat

How to send and query metrics to/from Grafana Mimir

Two Ways to Send Metrics to Grafana Mimir

Apr 17 1

See more recommendations