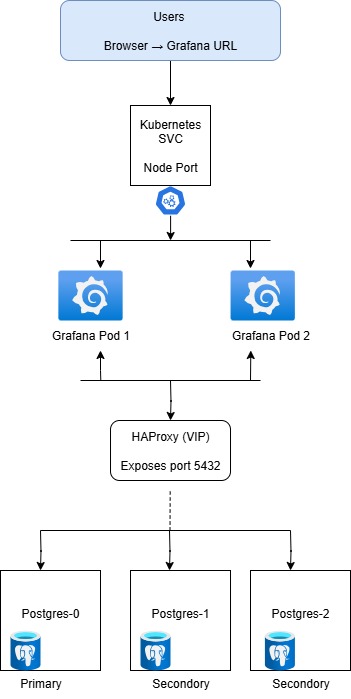
# Postgres HA with Patroni, etcd, HAProxy + Keepalived, and Grafana HA

**Purpose:** Step-by-step operational guide to build a production-like PostgreSQL HA stack using Patroni (Postgres clustering), etcd (DCS), HAProxy + keepalived (VIP) for a stable DB endpoint, and Grafana running in Kubernetes using the HA Postgres DB. This document captures the exact setup, configuration files, verification steps, and troubleshooting tips so your team can reproduce the setup for production.

## Architecture (quick view)



* VIP 10.67.74.137 is the single DB endpoint apps use.
* HAProxy decides which backend (Patroni node) is primary by asking Patroni REST API (/master or /health).
* Patroni + etcd manage cluster state and automatic failover.
* Grafana is configured to use Postgres DB at the VIP; pods are stateless and use DB for persistence.

## Inventory & IPs used in this document (adjust for your environment)

* postgres-1: 10.67.74.134
* postgres-2: 10.67.74.135
* postgres-3: 10.67.74.136
* VIP (keepalived): 10.67.74.137
* HAProxy will listen on 10.67.74.137:5432 for DB traffic and 10.67.74.137:7000 for stats UI.
* Kubernetes Grafana namespace: postgres-test (Grafana installed by Helm release grafana).

**Security note:** passwords in examples (like postgres or AdminPass@123) are for demo only — replace with secure random secrets in production.

# Step-by-step implementation

The document below is split into independent sections. Implement each section on the indicated node(s).

## A. Prepare prerequisites on all Postgres nodes

All Patroni nodes should have: - Ubuntu (or similar), with sudo access - postgresql package (the version you choose, e.g., 14) - python3, pip3 - git, curl, jq, build-essential (for compiling if necessary) - keepalived, haproxy installed on the HAProxy/keepalived host(s)

Example command (run on each VM):

sudo apt update -y  
sudo apt install -y postgresql-14 postgresql-client-14 python3-pip python3-venv \  
 etcd keepalived haproxy jq curl build-essential libpq-dev  
# Patroni (uses pip)  
sudo pip3 install "patroni[etcd]" psycopg2-binary

If your distribution doesn’t have postgresql-14 packages you want, install whichever version you standardize on and update patroni.yml’s bin\_dir accordingly.

## B. Install and configure etcd (DCS)

Patroni needs a distributed configuration store. We use etcd on the same 3 nodes. You can run a 3-node etcd cluster.

**1. Download etcd binaries (run on each node)**

ETCD\_VER=v3.5.15  
curl -L https://github.com/etcd-io/etcd/releases/download/${ETCD\_VER}/etcd-${ETCD\_VER}-linux-amd64.tar.gz -o /tmp/etcd-${ETCD\_VER}.tar.gz  
tar xvf /tmp/etcd-${ETCD\_VER}.tar.gz -C /tmp  
sudo mv /tmp/etcd-${ETCD\_VER}-linux-amd64/etcd /usr/local/bin/  
sudo mv /tmp/etcd-${ETCD\_VER}-linux-amd64/etcdctl /usr/local/bin/

If your VM cannot reach GitHub due to SSL/proxy issues, fetch the tarball through your corporate proxy or copy the binaries to each host.

**2. Create environment file /etc/default/etcd on each node (adjust names/IPs)**

Example for **postgres-1 (10.67.74.134)**:

ETCD\_NAME=postgres-1  
ETCD\_DATA\_DIR="/var/lib/etcd"  
ETCD\_LISTEN\_PEER\_URLS="http://10.67.74.134:2380"  
ETCD\_LISTEN\_CLIENT\_URLS="http://10.67.74.134:2379,http://127.0.0.1:2379"  
ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://10.67.74.134:2380"  
ETCD\_ADVERTISE\_CLIENT\_URLS="http://10.67.74.134:2379"  
ETCD\_INITIAL\_CLUSTER="postgres-1=http://10.67.74.134:2380,postgres-2=http://10.67.74.135:2380,postgres-3=http://10.67.74.136:2380"  
ETCD\_INITIAL\_CLUSTER\_STATE="new"  
ETCD\_INITIAL\_CLUSTER\_TOKEN="pg-etcd-cluster"

Create equivalent files on postgres-2 and postgres-3 replacing names & IPs.

**3. Create systemd unit /etc/systemd/system/etcd.service**

[Unit]  
Description=etcd key-value store  
After=network.target  
  
[Service]  
Type=notify  
EnvironmentFile=/etc/default/etcd  
ExecStart=/usr/local/bin/etcd --config-file /etc/etcd/etcd.yml  
Restart=always  
RestartSec=5s  
LimitNOFILE=40000  
  
[Install]  
WantedBy=multi-user.target

**4. Create a minimal /etc/etcd/etcd.yml (optional)**

A simple config is optional if environment vars are sufficient. A small example:

name: postgres-1  
data-dir: /var/lib/etcd  
listen-peer-urls: http://10.67.74.134:2380  
listen-client-urls: http://10.67.74.134:2379,http://127.0.0.1:2379  
initial-advertise-peer-urls: http://10.67.74.134:2380  
advertise-client-urls: http://10.67.74.134:2379  
initial-cluster: postgres-1=http://10.67.74.134:2380,postgres-2=http://10.67.74.135:2380,postgres-3=http://10.67.74.136:2380  
initial-cluster-state: new  
initial-cluster-token: pg-etcd-cluster

**5. Start etcd**

sudo systemctl daemon-reload  
sudo systemctl enable --now etcd  
sudo systemctl status etcd

**6. Verify etcd cluster**

export ETCDCTL\_API=3  
etcdctl --endpoints="http://10.67.74.134:2379,http://10.67.74.135:2379,http://10.67.74.136:2379" endpoint health  
etcdctl --endpoints="http://10.67.74.134:2379" member list

If all endpoints report healthy, etcd cluster is ready.

## C. Install PostgreSQL and Patroni

You should have PostgreSQL installed (server) and patroni installed (via pip) on each node. Patroni manages Postgres lifecycle and uses the DCS (etcd) for coordination.

**1. Example directory & user setup** (Assuming Ubuntu packages installed):

sudo mkdir -p /var/lib/postgresql/data  
sudo chown -R postgres:postgres /var/lib/postgresql

**2. Sample patroni.yml**

Place /etc/patroni.yml on each node. Only name, restapi.listen/connect\_address, etcd.host and postgresql.listen/connect\_address and data\_dir/bin\_dir vary per node.

Common fragment (edit per node):

scope: postgres-ha  
namespace: /db/  
name: postgres-1 # change to postgres-2 / postgres-3 on other nodes  
  
restapi:  
 listen: 10.67.74.134:8008  
 connect\_address: 10.67.74.134:8008  
  
etcd:  
 host: 10.67.74.134:2379,10.67.74.135:2379,10.67.74.136:2379  
  
bootstrap:  
 dcs:  
 ttl: 30  
 loop\_wait: 10  
 retry\_timeout: 10  
 maximum\_lag\_on\_failover: 1048576  
 postgresql:  
 parameters:  
 max\_connections: 100  
 shared\_buffers: 256MB  
 wal\_level: replica  
 hot\_standby: "on"  
 initdb:  
 - encoding: UTF8  
 - data-checksums  
  
postgresql:  
 listen: 10.67.74.134:5432  
 connect\_address: 10.67.74.134:5432  
 data\_dir: /var/lib/postgresql/data  
 bin\_dir: /usr/lib/postgresql/14/bin  
 authentication:  
 superuser:  
 username: postgres  
 password: postgres  
 replication:  
 username: replicator  
 password: replpass

* Copy appropriate file to postgres-2/3 and change name, IPs, and restapi/listen addresses.

**3. Systemd unit for Patroni** (on each node)

/etc/systemd/system/patroni.service

[Unit]  
Description=Patroni PostgreSQL HA Cluster  
After=network.target  
  
[Service]  
Type=simple  
User=root  
ExecStart=/usr/bin/python3 /usr/local/bin/patroni /etc/patroni.yml  
Restart=on-failure  
RestartSec=5  
  
[Install]  
WantedBy=multi-user.target

**4. Start Patroni on each node**

sudo systemctl daemon-reload  
sudo systemctl enable --now patroni  
sudo systemctl status patroni

**5. Verify Patroni cluster**

From any node’s REST API:

curl http://10.67.74.134:8008/cluster  
curl http://10.67.74.135:8008/cluster  
curl http://10.67.74.136:8008/cluster

You should see one member with role: leader and the others as replica.

**6. Create application DB + user (Grafana example)**

Connect to current primary (use VIP once HAProxy is configured; otherwise test against current leader REST output):

psql -U postgres -h 10.67.74.134 -d postgres  
# inside psql  
CREATE DATABASE grafanadb;  
CREATE USER grafana WITH PASSWORD 'postgres';  
GRANT ALL PRIVILEGES ON DATABASE grafanadb TO grafana;

For production: use strong passwords and consider a dedicated schema or roles.

## D. HAProxy + keepalived (VIP) – single stable endpoint for applications

You can run HAProxy and keepalived on a dedicated VM or on one of the Patroni nodes. For higher resilience, run HAProxy+keepalived on multiple hosts and use the VIP. This example shows one HAProxy instance bound to VIP 10.67.74.137 and health-checking Patroni :8008/master.

**1. keepalived config** /etc/keepalived/keepalived.conf (run on the node that will hold VIP; set state MASTER on primary keepalived node)

vrrp\_instance VI\_1 {  
 state MASTER  
 interface ens18  
 virtual\_router\_id 51  
 priority 150  
 advert\_int 1  
 authentication {  
 auth\_type PASS  
 auth\_pass mysecret  
 }  
 virtual\_ipaddress {  
 10.67.74.137/24  
 }  
}

Start keepalived:

sudo systemctl enable --now keepalived  
sudo systemctl status keepalived  
ip a show ens18 # confirm VIP appears as secondary IP

**2. HAProxy config** /etc/haproxy/haproxy.cfg

global  
 log /dev/log local0  
 maxconn 4096  
 user haproxy  
 group haproxy  
 daemon  
  
defaults  
 log global  
 mode tcp  
 option tcplog  
 option dontlognull  
 retries 3  
 timeout connect 10s  
 timeout client 1m  
 timeout server 1m  
  
frontend postgres\_write  
 bind 10.67.74.137:5432  
 mode tcp  
 default\_backend patroni\_postgres  
  
backend patroni\_postgres  
 option httpchk GET /master  
 http-check expect status 200  
 default-server inter 2s fall 3 rise 2 on-marked-down shutdown-sessions  
 server postgres-1 10.67.74.134:5432 check port 8008  
 server postgres-2 10.67.74.135:5432 check port 8008  
 server postgres-3 10.67.74.136:5432 check port 8008  
  
listen stats  
 bind 10.67.74.137:7000  
 mode http  
 stats enable  
 stats uri /  
 stats refresh 5s  
 stats auth admin:admin123

**3. Start HAProxy**

sudo systemctl enable --now haproxy  
sudo systemctl restart haproxy  
sudo systemctl status haproxy

**4. Verify HAProxy health & leader routing**

* Visit http://10.67.74.137:7000/ → HAProxy stats UI.
* From any client machine:

psql -h 10.67.74.137 -U grafana -d grafanadb -c "SELECT 1;" -W

* On HAProxy, check backends: one server should be UP (leader) and others DOWN (for write backend), because we used /master health check.

## E. Integrate Grafana running on Kubernetes

We configured Grafana Helm to use the Postgres HA VIP as its DB endpoint and set Grafana to be stateless (no PVC).

**1. Create the Kubernetes secrets** (in postgres-test namespace):

# Grafana DB secret (grafana user password)  
kubectl create secret generic grafana-db-secret -n postgres-test --from-literal=GF\_DATABASE\_PASSWORD='postgres'  
  
# Grafana admin secret (UI login)  
kubectl create secret generic grafana-admin-secret -n postgres-test \  
 --from-literal=admin-user=admin --from-literal=admin-password='AdminPass@123'

**2. Example values-grafana.yaml (working)**

replicaCount: 2  
  
persistence:  
 enabled: false  
  
admin:  
 existingSecret: grafana-admin-secret  
 userKey: admin-user  
 passwordKey: admin-password  
  
# inject DB via env + secret  
env:  
 GF\_DATABASE\_TYPE: postgres  
 GF\_DATABASE\_HOST: 10.67.74.137:5432  
 GF\_DATABASE\_NAME: grafanadb  
 GF\_DATABASE\_USER: grafana  
 GF\_DATABASE\_SSL\_MODE: disable  
  
envValueFrom:  
 GF\_DATABASE\_PASSWORD:  
 secretKeyRef:  
 name: grafana-db-secret  
 key: GF\_DATABASE\_PASSWORD  
  
service:  
 type: ClusterIP

**3. Deploy/Upgrade Grafana Helm chart**

helm upgrade --install grafana grafana/grafana -n postgres-test -f values-grafana.yaml

**4. Verification**

* Check pods: kubectl get pods -n postgres-test -l app.kubernetes.io/name=grafana (expect 2 pods)
* Inside pod, verify env vars: kubectl exec -it -n postgres-test <pod> -- env | grep GF\_DATABASE
* Logs should indicate successful DB connection: kubectl logs -n postgres-test <pod> | grep "Connecting to DB"
* UI access: kubectl port-forward -n postgres-test svc/grafana 3000:80 → http://localhost:3000 → login with admin/admin-password from secret.

## F. Tests & Failover validation (demo checklist)

1. **Dashboard persistence test**
   * Create folder Demo-Folder and a dashboard Demo-Dashboard in Grafana UI.
   * In DB: SELECT id, title, is\_folder FROM dashboard WHERE title ILIKE '%Demo%';
   * Delete all Grafana pods: kubectl delete pods -n postgres-test -l app.kubernetes.io/name=grafana
   * Reconnect → confirm Demo-Dashboard present.
2. **User persistence test**
   * Create a new Grafana user (e.g. demo\_user).
   * Query DB SELECT id, login, is\_admin FROM "user" WHERE login = 'demo\_user';
   * Delete pods → login with demo\_user.
3. **Patroni failover test**
   * On current primary node: sudo systemctl stop patroni (or pg\_ctl to simulate crash) or use patronictl for graceful failover.
   * Observe curl http://<other-node>:8008/cluster to confirm new leader.
   * Grafana should continue to work via VIP.
4. **HAProxy node failure** (if using single HAProxy):
   * If HAProxy VM fails, Grafana will lose DB connectivity. For production, deploy HAProxy on at least 2 hosts and use keepalived for VIP across them.

## G. Common issues & troubleshooting

* **etcd certificate / SSL errors when downloading**: use a trusted proxy or copy binaries.
* **etcd.service not found**: ensure etcd binary is at /usr/local/bin/etcd and the systemd unit is created and reloaded.
* **HAProxy shows all backends DOWN**: ensure HAProxy httpchk endpoint is /master and Patroni REST API is reachable at :8008 from HAProxy host. Use curl http://10.67.74.134:8008/master to test.
* **Grafana starts but uses SQLite**: the chart may not render database: into grafana.ini directly. Use env + envValueFrom to inject GF\_DATABASE\_\* values and password from secret.
* **Grafana CrashLoopBackOff**: usually missing GF\_DATABASE\_PASSWORD or incorrect envValueFrom key -> check kubectl describe pod to inspect environment variables.
* **Invalid credentials on Grafana UI**: admin credentials come from grafana-admin-secret. Get current password: kubectl get secret grafana-admin-secret -n postgres-test -o jsonpath="{.data.admin-password}" | base64 --decode.

## H. Appendix — full example files (copy & edit)

### /etc/default/etcd (postgres-1)

ETCD\_NAME=postgres-1  
ETCD\_DATA\_DIR="/var/lib/etcd"  
ETCD\_LISTEN\_PEER\_URLS="http://10.67.74.134:2380"  
ETCD\_LISTEN\_CLIENT\_URLS="http://10.67.74.134:2379,http://127.0.0.1:2379"  
ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS="http://10.67.74.134:2380"  
ETCD\_ADVERTISE\_CLIENT\_URLS="http://10.67.74.134:2379"  
ETCD\_INITIAL\_CLUSTER="postgres-1=http://10.67.74.134:2380,postgres-2=http://10.67.74.135:2380,postgres-3=http://10.67.74.136:2380"  
ETCD\_INITIAL\_CLUSTER\_STATE="new"  
ETCD\_INITIAL\_CLUSTER\_TOKEN="pg-etcd-cluster"

### /etc/etcd/etcd.yml (postgres-1)

name: postgres-1  
data-dir: /var/lib/etcd  
listen-peer-urls: http://10.67.74.134:2380  
listen-client-urls: http://10.67.74.134:2379,http://127.0.0.1:2379  
initial-advertise-peer-urls: http://10.67.74.134:2380  
advertise-client-urls: http://10.67.74.134:2379  
initial-cluster: postgres-1=http://10.67.74.134:2380,postgres-2=http://10.67.74.135:2380,postgres-3=http://10.67.74.136:2380  
initial-cluster-state: new  
initial-cluster-token: pg-etcd-cluster

### /etc/systemd/system/etcd.service

[Unit]  
Description=etcd key-value store  
After=network.target  
  
[Service]  
Type=notify  
EnvironmentFile=/etc/default/etcd  
ExecStart=/usr/local/bin/etcd --config-file /etc/etcd/etcd.yml  
Restart=always  
RestartSec=5s  
LimitNOFILE=40000  
  
[Install]  
WantedBy=multi-user.target

### /etc/patroni.yml (postgres-1)

scope: postgres-ha  
namespace: /db/  
name: postgres-1  
  
restapi:  
 listen: 10.67.74.134:8008  
 connect\_address: 10.67.74.134:8008  
  
etcd:  
 host: 10.67.74.134:2379,10.67.74.135:2379,10.67.74.136:2379  
  
bootstrap:  
 dcs:  
 ttl: 30  
 loop\_wait: 10  
 retry\_timeout: 10  
 maximum\_lag\_on\_failover: 1048576  
 postgresql:  
 parameters:  
 max\_connections: 100  
 shared\_buffers: 256MB  
 wal\_level: replica  
 hot\_standby: "on"  
 initdb:  
 - encoding: UTF8  
 - data-checksums  
  
postgresql:  
 listen: 10.67.74.134:5432  
 connect\_address: 10.67.74.134:5432  
 data\_dir: /var/lib/postgresql/data  
 bin\_dir: /usr/lib/postgresql/14/bin  
 authentication:  
 superuser:  
 username: postgres  
 password: postgres  
 replication:  
 username: replicator  
 password: replpass

### /etc/systemd/system/patroni.service

[Unit]  
Description=Patroni PostgreSQL HA Cluster  
After=network.target  
  
[Service]  
Type=simple  
User=root  
ExecStart=/usr/bin/python3 /usr/local/bin/patroni /etc/patroni.yml  
Restart=on-failure  
RestartSec=5  
  
[Install]  
WantedBy=multi-user.target

### /etc/keepalived/keepalived.conf (example)

vrrp\_instance VI\_1 {  
 state MASTER  
 interface ens18  
 virtual\_router\_id 51  
 priority 150  
 advert\_int 1  
 authentication {  
 auth\_type PASS  
 auth\_pass mysecret  
 }  
 virtual\_ipaddress {  
 10.67.74.137/24  
 }  
}

### /etc/haproxy/haproxy.cfg (example)

global  
 log /dev/log local0  
 maxconn 4096  
 user haproxy  
 group haproxy  
 daemon  
  
defaults  
 log global  
 mode tcp  
 option tcplog  
 option dontlognull  
 retries 3  
 timeout connect 10s  
 timeout client 1m  
 timeout server 1m  
  
frontend postgres\_write  
 bind 10.67.74.137:5432  
 mode tcp  
 default\_backend patroni\_postgres  
  
backend patroni\_postgres  
 option httpchk GET /master  
 http-check expect status 200  
 default-server inter 2s fall 3 rise 2 on-marked-down shutdown-sessions  
 server postgres-1 10.67.74.134:5432 check port 8008  
 server postgres-2 10.67.74.135:5432 check port 8008  
 server postgres-3 10.67.74.136:5432 check port 8008  
  
listen stats  
 bind 10.67.74.137:7000  
 mode http  
 stats enable  
 stats uri /  
 stats refresh 5s  
 stats auth admin:admin123

### values-grafana.yaml (final working)

replicaCount: 2  
  
persistence:  
 enabled: false  
  
admin:  
 existingSecret: grafana-admin-secret  
 userKey: admin-user  
 passwordKey: admin-password  
  
env:  
 GF\_DATABASE\_TYPE: postgres  
 GF\_DATABASE\_HOST: 10.67.74.137:5432  
 GF\_DATABASE\_NAME: grafanadb  
 GF\_DATABASE\_USER: grafana  
 GF\_DATABASE\_SSL\_MODE: disable  
  
envValueFrom:  
 GF\_DATABASE\_PASSWORD:  
 secretKeyRef:  
 name: grafana-db-secret  
 key: GF\_DATABASE\_PASSWORD  
  
service:  
 type: ClusterIP

## I. Final notes & production considerations

* **Backups:** Ensure you implement regular backups (pg\_basebackup, WAL archiving, or logical backups) to a safe off-cluster storage.
* **Security:** Use TLS for etcd, TLS for Patroni/Postgres replication, and secure admin passwords. Do not leave postgres/postgres in production.
* **Monitoring:** Monitor patroni logs, etcd health, HAProxy metrics, and Postgres metrics (pg\_stat\_replication, pg\_stat\_activity).
* **Scaling:** Grafana stateless pods can be scaled horizontally using helm/ReplicaSet. For HAProxy, use at least 2 instances + keepalived for VIP failover.