📓 PostgreSQL High Availability Cluster

Using Patroni, Etcd, and HAProxy with Docker Containers.

This documentation outlines the deployment of a PostgreSQL High Availability (HA) cluster in a testing environment using Docker containers. It integrates Patroni, Etcd, and HAProxy to simulate automated failover and assess cluster scalability. Please note that this setup is intended solely for testing purposes and does not prioritize security configurations.

👨‍💻 **Prepared by:**

Isuru Pavithra Hapuarachchi

📅 **Date:**

April 2025

Contents

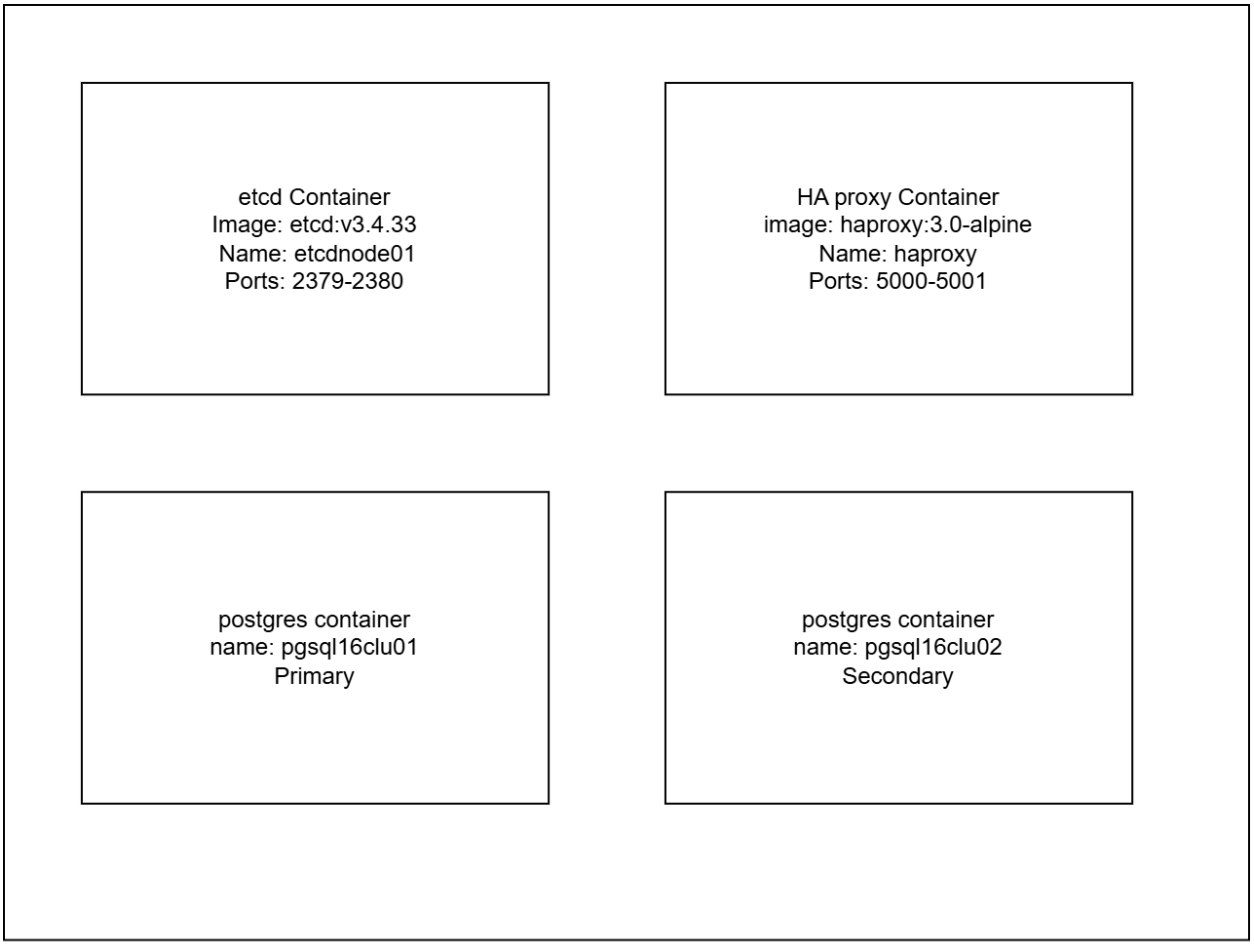
Overview

This documentation covers the implementation of a PostgreSQL High Availability (HA) Cluster using Docker containers running on a single server for testing purposes. The system integrates PostgreSQL, Patroni, etcd, and HAProxy to create a fault-tolerant, self-healing database cluster with automatic failover capabilities.

The architecture features PostgreSQL streaming replication in a hot-standby configuration, providing data redundancy and ensuring minimal downtime during failover events. It uses lightweight PostgreSQL 16.8 Alpine containers, with Patroni handling cluster coordination, etcd providing distributed consensus, and HAProxy directing traffic between primary and replica nodes based on read/write operations.

Although this implementation is deployed on a single physical server using multiple Docker containers for testing, the architecture is designed to scale across multiple physical or virtual machines.

Architecture



Components

1. **PostgreSQL Database**
   - **Version**: PostgreSQL 16.8-Alpine
   - **Configuration**: Master-Slave architecture with streaming replication

(Possible to add more Postgres container nodes to the current architecture)

2. **Patroni**
   - **Purpose**: Handles PostgreSQL high availability through automated failover management
   - **Features**:
     - Manages PostgreSQL configuration
     - Coordinates leader election
     - Handles failover and recovery procedures
     - Integrates with etcd for consensus and state storage
   - **Installation:** Inside Docker container using Dockerfile.
3. **etcd**
   - **Version**: v3.4.33
   - **Purpose**: Distributed key-value store for:
     - Cluster coordination
     - Configuration storage
     - Leader election support
4. **HAProxy**
   - **Version**: 3.0-Alpine
   - **Purpose**: Load balancer that:
     - Routes write operations to the master node.
     - Distributes read operations across available nodes.
     - Performs health checks to ensure availability.
     - Provides a single endpoint for client applications.
5. **Docker & Docker-compose**
     - Containerizes PostgreSQL, Patroni, etcd, and HAProxy.
     - Ensures isolated and consistent environments.
     - Simplifies deployment and testing on a single server.
     - Manages multi-container setup via docker-compose.yml
     - Handles service dependencies and networking
     - Easy one-command startup (docker-compose up).

Configuration

Docker Compose File

The docker-compose.yml file orchestrates all containers in the cluster:

```
services:
  etcdnode01:
    image: quay.io/coreos/etcd:v3.4.33
    container_name: etcdnode01
    restart: unless-stopped
    volumes:
      - etcdnode_data:/etcd-data
    environment:
      ETCD_DATA_DIR: /etcd-data
      ETCD_NAME: etcdnode01
      ETCD_INITIAL_ADVERTISE_PEER_URLS: http://etcdnode01:2380
      ETCD_LISTEN_PEER_URLS: http://0.0.0.0:2380
      ETCD_LISTEN_CLIENT_URLS: http://0.0.0.0:2379
      ETCD_ADVERTISE_CLIENT_URLS: http://etcdnode01:2379
      ETCD_INITIAL_CLUSTER: etcdnode01=http://etcdnode01:2380
      ETCD_INITIAL_CLUSTER_TOKEN: etcd-postgres-cluster
      ETCD_INITIAL_CLUSTER_STATE: new
    ports:
      - "2379:2379"
      - "2380:2380"
    networks:
      - postgres_network

  pgsql16clu01:
    container_name: pgsql16clu01
    build:
      dockerfile: ./patroni.dockerfile
      context: .
      target: postgres-patroni
    restart: unless-stopped
    environment:
      POSTGRES_USER: # Add your PostgreSQL username
```

```
    POSTGRES_PASSWORD: # Add your PostgreSQL password
    POSTGRES_DB: # Add your PostgreSQL database name
    IP_NODO_1: pgsql16clu01
    IP_NODO_2: pgsql16clu02
    NOME_DB_USER: # Add your PostgreSQL username
    DB_PASSWORD: # Add your PostgreSQL password
  volumes:
    - postgres_data_01:/var/lib/postgresql/data
    - ./patroni.yml:/venv/etc/patroni.yml
  ports:
    - "5432:5432"
    - "8008:8008"
  depends_on:
    - etcdnode01
  networks:
    - postgres_network

pgsql16clu02:
  container_name: pgsql16clu02
  build:
    dockerfile: ./patroni2.dockerfile
    context: .
    target: postgres-patroni
  restart: unless-stopped
  environment:
    POSTGRES_USER: # Add your PostgreSQL username
    POSTGRES_PASSWORD: # Add your PostgreSQL password
    POSTGRES_DB: # Add your PostgreSQL database name
    IP_NODO_1: pgsql16clu01
    IP_NODO_2: pgsql16clu02
    NOME_DB_USER: # Add your PostgreSQL username
    DB_PASSWORD: # Add your PostgreSQL password
  volumes:
    - postgres_data_02:/var/lib/postgresql/data
    - ./patroni2.yml:/venv/etc/patroni.yml
  ports:
    - "5433:5432"
    - "8009:8009"
  depends_on:
```

```
      - etcdnode01
    networks:
      - postgres_network

  haproxy:
    image: haproxy:3.0-alpine
    container_name: haproxy
    restart: unless-stopped
    ports:
      - "5000:5000"  # Primary write endpoint
      - "5001:5001"  # Read-only replica endpoint
    volumes:
      - ./haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg:ro
    depends_on:
      - pgsql16clu01
      - pgsql16clu02
    networks:
      - postgres_network

networks:
  postgres_network:
    driver: bridge

volumes:
  postgres_data_01:
  postgres_data_02:
  etcdnode_data:
```

HAProxy Configuration

The haproxy.cfg file defines load balancing rules and health checks:

```
global
    log stdout format raw local0
    maxconn 4096
    daemon
    user haproxy
    group haproxy

defaults
    mode tcp
    log global
    option tcplog
    timeout connect 5s
    timeout client  30s
    timeout server  30s

frontend patroni_cluster_frontend
    bind *:5000
    default_backend patroni_cluster_write

frontend patroni_readonly_frontend
    bind *:5001
    default_backend patroni_cluster_read

# Backend for write traffic - assumes pgsql16clu01 is primary on 5432
backend patroni_cluster_write
    mode tcp
    option tcp-check
    balance first
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-
sessions
    server pgsql16clu01 #IP_masternode:5432 check
    server pgsql16clu02 #IP_Slavenode:5433 check backup

# Backend for read traffic - both nodes OK
backend patroni_cluster_read
    mode tcp
    option tcp-check
```

```
    balance roundrobin
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-
sessions
    server pgsql16clu01 #IP_masternode:5432 check
    server pgsql16clu02 #IP_Slavenode:5433 check
```

Patroni Configuration

The patroni.yml file configures the primary Patroni instance:

```
scope: miopg16cluster
namespace: /mio_pg16_cluster/
name: pgsql16clu01

restapi:
  listen: 0.0.0.0:8008
  connect_address: pgsql16clu01:8008

etcd3:
  hosts:
    - etcdnode01:2379
  name: pgsql16clu01  # Required for identifying this member in etcd

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
        password_encryption: md5
        wal_level: logical
        hot_standby: "on"
        max_connections: 100
        max_worker_processes: 8
        wal_keep_size: 128MB
```

```
        max_wal_senders: 10
        max_replication_slots: 10
        max_prepared_transactions: 0
        max_locks_per_transaction: 64
        wal_log_hints: "on"
        track_commit_timestamp: "off"
        archive_mode: "on"
        archive_timeout: 1800s
        archive_command: mkdir -p ../wal_archive && test ! -f
../wal_archive/%f && cp %p ../wal_archive/%f
      recovery_conf:
        restore_command: cp ../wal_archive/%f %p

  initdb:
    - encoding: UTF8
    - data-checksums

  pg_hba:
    - host replication {Replication_username} 0.0.0.0/0 trust
    - host all all 0.0.0.0/0 trust
  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb

postgresql:
  listen: 0.0.0.0:5432
  connect_address: pgsql16clu01:5432
  data_dir: /var/lib/postgresql/data
  bin_dir: /usr/local/bin
  pgpass: /tmp/pgpass

  authentication:
    replication:
      username: # Add Username for replication
      password: # Add Password for replication
    superuser:
```

```
      username: # Add your PostgreSQL username
      password: # Add your PostgreSQL password

  tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false

watchdog:
  mode: 'off'
  device: /dev/watchdog
  safety_margin: 5
```

The patroni2.yml file configures the Secondary Patroni instance:

```
scope: miopg16cluster
namespace: /mio_pg16_cluster/
name: pgsql16clu02

restapi:
  listen: 0.0.0.0:8009
  connect_address: pgsql16clu02:8009

etcd3:
  hosts:
    - etcdnode01:2379
  name: pgsql16clu02

bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
```

```
        password_encryption: md5
        wal_level: logical
        hot_standby: "on"
        max_connections: 100
        max_worker_processes: 8
        wal_keep_size: 128MB
        max_wal_senders: 10
        max_replication_slots: 10
        max_prepared_transactions: 0
        max_locks_per_transaction: 64
        wal_log_hints: "on"
        track_commit_timestamp: "off"
        archive_mode: "on"
        archive_timeout: 1800s
        archive_command: mkdir -p ../wal_archive && test ! -f
../wal_archive/%f && cp %p ../wal_archive/%f
      recovery_conf:
        restore_command: cp ../wal_archive/%f %p

  initdb:
    - encoding: UTF8
    - data-checksums

  pg_hba:
    - host replication {Replication_username} 0.0.0.0/0 trust
    - host all all 0.0.0.0/0 trust

  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb

postgresql:
  listen: 0.0.0.0:5433
  connect_address: pgsql16clu02:5433
  data_dir: /var/lib/postgresql/data
  bin_dir: /usr/local/bin
```

```
  pgpass: /tmp/pgpass
  authentication:
    replication:
      username: # add your replication username
      password: # add your replication password
    superuser:
      username: # add your PostgreSQL username
      password: # add your PostgreSQL password

  tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
watchdog:
  mode: 'off'
  device: /dev/watchdog
  safety_margin: 5
```

Dockerfiles

The patroni.dockerfile defines how the primary Patroni container is built:

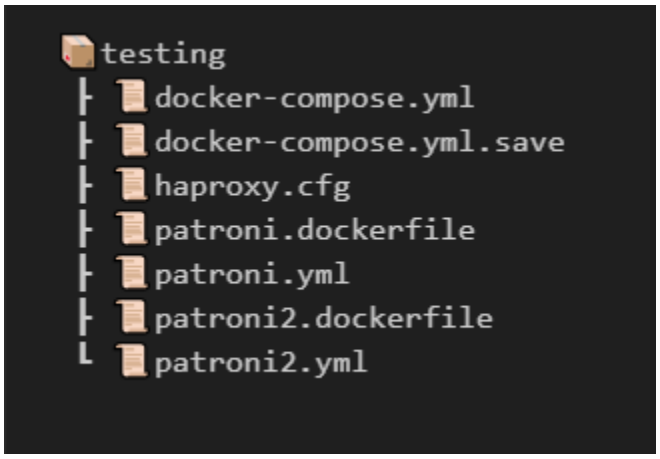
```
FROM postgres:16-alpine as postgres-patroni

# Install Patroni dependencies
RUN apk add --no-cache build-base linux-headers python3-dev py3-pip
py3-virtualenv \
    && python3 -m venv /venv && mkdir /venv/etc

# Install Python dependencies (psycopg2 and Patroni)
RUN /venv/bin/pip install psycopg2 && /venv/bin/pip install
patroni[etcd]

# Copy Patroni configuration file into the container
COPY patroni.yml /venv/etc/patroni.yml

# Set the working directory to the PostgreSQL home directory
WORKDIR /var/lib/postgresql

# Ensure PostgreSQL data directory exists and set permissions
RUN mkdir -p /var/lib/postgresql/data && \
    chmod 0700 /var/lib/postgresql/data && \
    chown postgres:postgres /var/lib/postgresql/data

# Expose PostgreSQL and Patroni's REST API ports
EXPOSE 5432 8008

# Start Patroni (Patroni will initialize the database if needed)
CMD su-exec postgres /venv/bin/patroni /venv/etc/patroni.yml
```

The patroni2.dockerfile defines how the secondary Patroni container is built:

```
FROM postgres:16-alpine as postgres-patroni

# Install Patroni dependencies
RUN apk add --no-cache build-base linux-headers python3-dev py3-pip
py3-virtualenv \
    && python3 -m venv /venv && mkdir /venv/etc

# Install Python dependencies (psycopg2 and Patroni)
RUN /venv/bin/pip install psycopg2 && /venv/bin/pip install
patroni[etcd]

# Copy Patroni configuration file into the container
COPY patroni2.yml /venv/etc/patroni.yml

# Set the working directory to the PostgreSQL home directory
WORKDIR /var/lib/postgresql

# Ensure PostgreSQL data directory exists and set permissions
RUN mkdir -p /var/lib/postgresql/data && \
    chmod 0700 /var/lib/postgresql/data && \
    chown postgres:postgres /var/lib/postgresql/data

# Expose PostgreSQL and Patroni's REST API ports
EXPOSE 5433 8009

# Start Patroni (Patroni will initialize the database if needed)
CMD su-exec postgres /venv/bin/patroni /venv/etc/patroni.yml
```

Deployment Steps

File tree



Review and customize configuration files

- Adjust PostgreSQL parameters in Patroni configuration files
- Modify HAProxy settings if needed
- Update hostnames/IPs if not using Docker's internal DNS

Start the cluster

```
docker-compose up -d
```

Verify deployment

```
docker-compose ps
```

Usage and Testing

Connection Information

- Write operations (master): localhost:5000
- Read operations (any available node): localhost:5000

Create a test table and insert data

```
CREATE TABLE test_table (id SERIAL PRIMARY KEY, name TEXT);
INSERT INTO test_table (name) VALUES ('test_data');
```

Verify replication on the slave node

```
psql -h localhost -p 5001 -U postgres -c "SELECT * FROM test_table;"
```

Test failover

Stop the current master

```
docker stop <master-container-name>
```

Check that the slave becomes the new master

```
docker exec -it <patroni-container> sh
source /venv/bin/activate
patronictl -c /venv/etc/patroni.yml list
```

Verify connectivity to the new master

- psql -h localhost -p 5000 -U postgres

Issues

pg_hba.conf Management

- Always define access rules via pg_hba section in patroni.yml, not manually in postgres – Patroni will manage it.
- Misconfigured pg_hba entries can block replication or client access.

Container Networking & Port Access

- Ensure necessary ports (e.g., 5432, 8008, 2379 for etcd, 5000+ for HAProxy) are open and properly mapped in Docker Compose.
- Confirm containers are in the same Docker network and can resolve each other’s hostnames.

Patroni Installation Location

- Patroni must be installed and run inside the PostgreSQL container environment.
- Use a virtual environment or Alpine-compatible installation for Python dependencies.

Etcd Cluster Discovery

- Check etcd cluster status and connectivity; all Patroni nodes must connect reliably to etcd.
- Clock drift or slow disk I/O can affect etcd performance.

Volume Permissions

- Ensure proper ownership (postgres:postgres) on mounted volumes (e.g., /data, /etc/patroni) to avoid permission errors.

Future Improvements

1. **Ansible Integration**
2. **Environment Variable Configuration**
3. **Configuration Consolidation**
4. **Monitoring Integration**
5. **Automated Testing**
6. **Security Enhancements**