



**Advanced Postgres**

Surendra Panpaliya

# Agenda

Concepts and benefits of partitioning

01

Implementing table partitioning

02

Managing and maintaining partitions

03

Setting up streaming replication

04

Failover and recovery strategies

# Agenda

Introduction to replication tools



```
graph TD; A[Introduction to replication tools] --> B[PgBouncer, Patroni]; B --> C[PostgreSQL Architecture]; C --> D[Migrating from Oracle to PostgreSQL];
```

PgBouncer, Patroni

PostgreSQL Architecture

Migrating from Oracle to PostgreSQL

# Partitioning in PostgreSQL

Partitioning is a database design technique

Divides large tables into smaller

More manageable pieces called partitions

# Partitioning in PostgreSQL



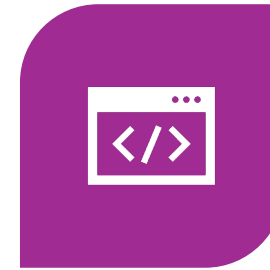
CAN SIGNIFICANTLY  
IMPROVE



PERFORMANCE,



MANAGEABILITY



SCALABILITY.

# Types of Partitioning

01

Range  
Partitioning

02

List  
Partitioning

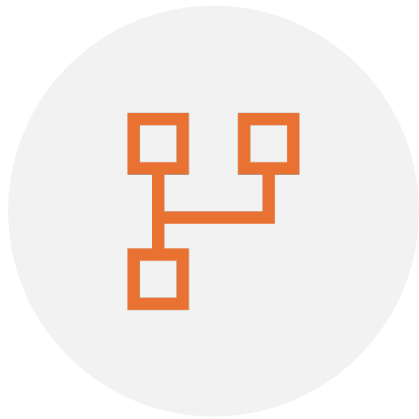
03

Hash  
Partitioning

04

Composite  
Partitioning

# Range Partitioning



DIVIDES DATA



BASED ON



A RANGE OF VALUES

# List Partitioning

Divides  
Data

Based on

List of  
values

# Hash Partitioning

Distributes  
data across  
partitions

Based on

Hash function.

# Composite Partitioning

Combines

Two or  
more

Partitioning  
methods.

# Example: Range Partitioning

**Scenario**

Partition a table

storing patient records

by year of birth.

# Example: Range Partitioning

---

## Step 1: Create the Parent Table

---

```
CREATE TABLE patients (  
    patient_id SERIAL PRIMARY KEY,  
    name TEXT NOT NULL,  
    birth_year INT NOT NULL,  
    medical_record TEXT  
) PARTITION BY RANGE (birth_year);
```

---

# Step 2: Create Partitions



CREATE TABLE patients\_2000\_2009



PARTITION OF patients



FOR VALUES



FROM (2000) TO (2010);

# Step 2: Create Partitions



CREATE TABLE patients\_2010\_2019



PARTITION OF patients



FOR VALUES



FROM (2010) TO (2020);

# Step 2: Create Partitions



CREATE TABLE patients\_2020\_2029



PARTITION OF patients



FOR VALUES

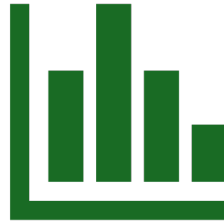


FROM (2020) TO (2030);

# Benefits of Partitioning



Improved Query  
Performance



Efficient Data  
Management



Load Balancing

# Improved Query Performance



Queries can scan

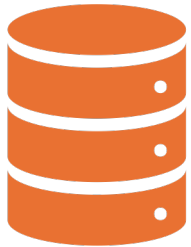


relevant partitions



instead of the entire  
table.

# Efficient Data Management



Easier to manage large  
datasets

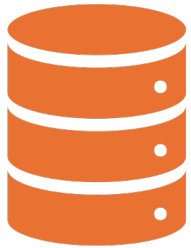


by archiving or

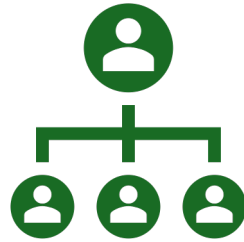


deleting old partitions.

# Load Balancing



Distributes data



across multiple



storage devices

# Example Query Using Partitioned Table

**Retrieve Patients Born Between 2010 and 2019**

```
SELECT * FROM patients
```

```
WHERE birth_year
```

```
BETWEEN 2010 AND 2019;
```

# Replication in PostgreSQL

Replication involves

copying data from

one database server (primary)

to another (standby).

# Replication in PostgreSQL



Enhances

data  
availability

fault  
tolerance

load  
balancing

# Types of Replication



STREAMING  
REPLICATION



LOGICAL  
REPLICATION



SYNCHRONOUS  
REPLICATION



ASYNCHRONOUS  
REPLICATION

# Streaming Replication



CONTINUOUSLY  
STREAMS



WAL (WRITE-AHEAD  
LOG) CHANGES



FROM THE PRIMARY TO  
THE STANDBY SERVER.

# Logical Replication



REPLICATES DATA  
CHANGES



AT THE LOGICAL LEVEL



(E.G., TABLE ROWS)

# Synchronous Replication

Ensures data is  
written

to both primary  
and standby  
servers

before  
committing the  
transaction

# Asynchronous Replication



COMMITTS  
TRANSACTIONS



ON THE PRIMARY  
SERVER



WITHOUT  
WAITING FOR



CONFIRMATION  
FROM



THE STANDBY  
SERVER

# Example: Setting Up Streaming Replication

## Step 1: Configure Primary Server

Update postgresql.conf

```
wal_level = replica
```

```
max_wal_senders = 3
```

```
wal_keep_segments = 64
```

# Example: Setting Up Streaming Replication

**Update pg\_hba.conf**

```
host replication all 192.168.1.0/24 md5
```

**Restart PostgreSQL**

```
sudo systemctl restart postgresql
```

# Step 2: Set Up Standby Server



**Create Base Backup from Primary**



```
pg_basebackup -h primary_host -D /var/lib/postgresql/12/main  
-U replication_user -P -R
```

# Update postgresql.conf on Standby

---

```
primary_conninfo = 'host=primary_host
```

---

```
port=5432
```

---

```
user=replication_user
```

---

```
password=your_password'
```

# Update postgresql.conf on Standby

## Start PostgreSQL on Standby

```
sudo systemctl start postgresql
```

# Monitoring Replication

## Check Replication Status

```
SELECT * FROM pg_stat_replication;
```

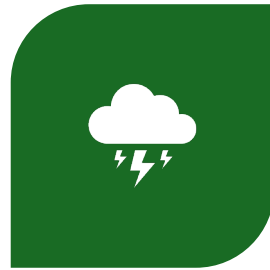
# **Failover and Recovery strategies**

Surendra Panpaliya

# Failover and recovery strategies



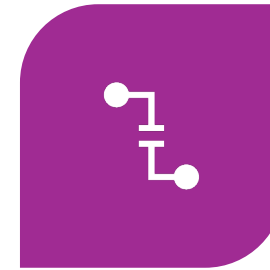
CRUCIAL  
COMPONENTS



OF A DISASTER  
RECOVERY PLAN



TO HANDLE  
UNEXPECTED  
FAILURES,



MAINTENANCE, OR  
OTHER  
DISRUPTIONS.

# Failover Strategies

Surendra Panpaliya

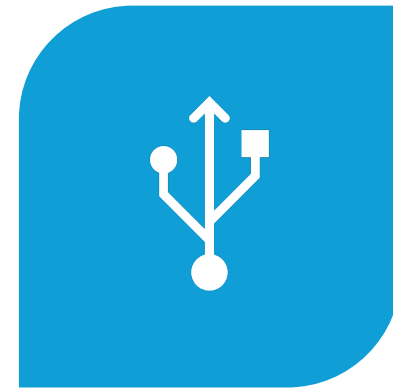
# Automated Failover



INVOLVES USING  
TOOLS



AUTOMATICALLY  
DETECT FAILURES

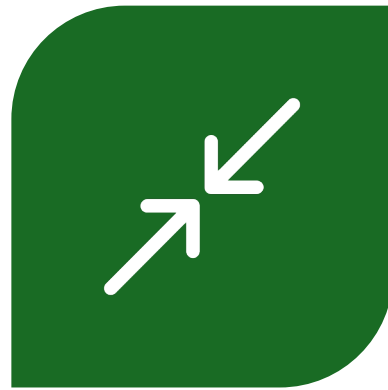


SWITCH TO A  
STANDBY SERVER

# Automated Failover



MINIMIZES  
DOWNTIME



ENSURES



CONTINUOUS  
AVAILABILITY

# Tools for Automated Failover

Patroni

pg\_auto\_failover

pgpool-II

# Patroni



An open-source tool



High availability



Failover management

# pg\_auto\_failover



A PostgreSQL  
extension



for automatic



failover and  
management.

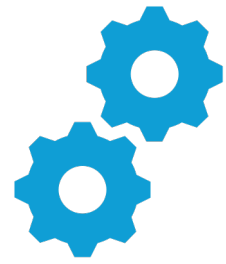
# pgpool-II



Provides load  
balancing



connection pooling



automatic failover

# Manual Failover

Involves  
manually  
switching

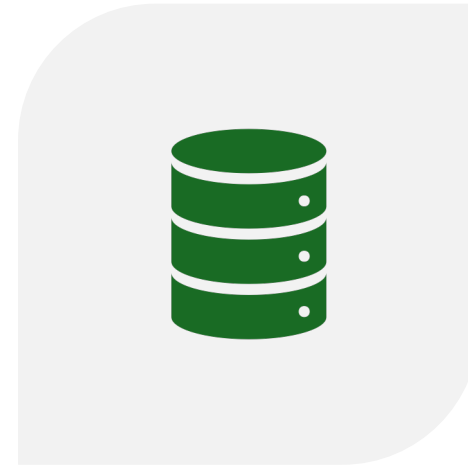
to the  
standby  
server

in case of a  
failure.

# Manual Failover



REQUIRES  
INTERVENTION



FROM A DATABASE  
ADMINISTRATOR.

# Steps for Manual Failover



PROMOTE STANDBY



UPDATE  
DNS/APPLICATIONS



RECONFIGURE OLD  
PRIMARY

# Promote Standby



Use PostgreSQL commands



to promote the standby server to the primary role.



```
pg_ctl promote -D /var/lib/postgresql/12/main
```

# Update DNS/Applications



Redirect applications



to the new primary server.

# Reconfigure Old Primary

Reconfigure the former primary server

as a new standby or

address any issues

before reintegrating it

into the cluster.

# Geo-Replication



FOR GEOGRAPHICALLY  
DISTRIBUTED SYSTEMS



TO REPLICATE DATA  
ACROSS



DIFFERENT DATA  
CENTERS

# Geo-Replication



Ensures that



if one data center fails



Other can take over

# Geo-Replication Implementation

Set up replication between

primary and standby servers

located in different

geographical locations

# Geo-Replication Implementation



Ensure network latency  
and



bandwidth are  
sufficient



to handle data  
synchronization.

# Recovery Strategies

Surendra Panpaliya

---

# Point-in-Time Recovery (PITR)

Allows to restore the database

to a specific point in time.

Useful for recovering from

accidental data loss or corruption.

# Steps for PITR



**Create a Base Backup:**



Regularly take base backups of data directory.



```
pg_basebackup -D /var/lib/postgresql/12/main
```



```
-F tar -z -P -U replicator
```

# Restore Base Backup



Restore the base backup to the target location



```
tar -xzf base_backup.tar.gz -C /var/lib/postgresql/12/main
```

# Apply WAL Files



USE THE WAL FILES



TO ROLL FORWARD  
THE DATABASE



TO THE DESIRED  
POINT IN TIME.

# Example Recovery Configuration



Create a recovery.conf file with the appropriate settings.



```
restore_command = 'cp /var/lib/postgresql/archive/%f %p'
```



```
recovery_target_time = '2024-07-27 12:00:00'
```

# Start PostgreSQL



Start the PostgreSQL server,



which will apply the WAL files and

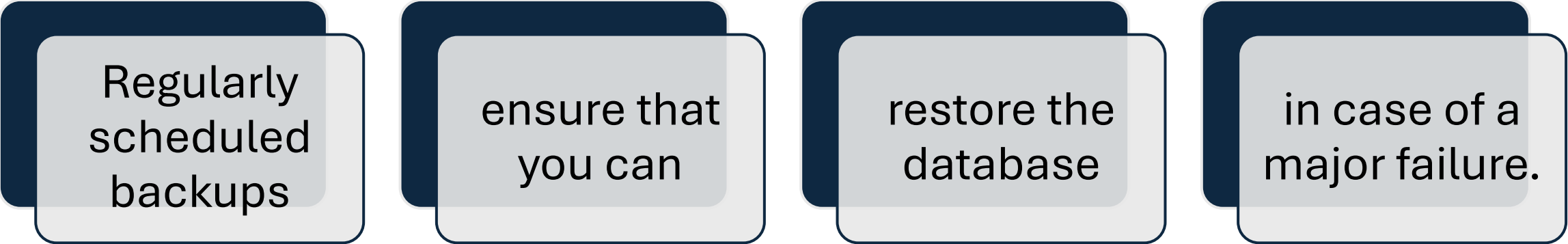


recover the data to the specified time.



```
pg_ctl start -D /var/lib/postgresql/12/main
```

# Backup and Restore



Regularly  
scheduled  
backups

ensure that  
you can

restore the  
database

in case of a  
major failure.

# Backup Command



```
pg_dumpall -U postgres -f /path/to/backup.sql
```

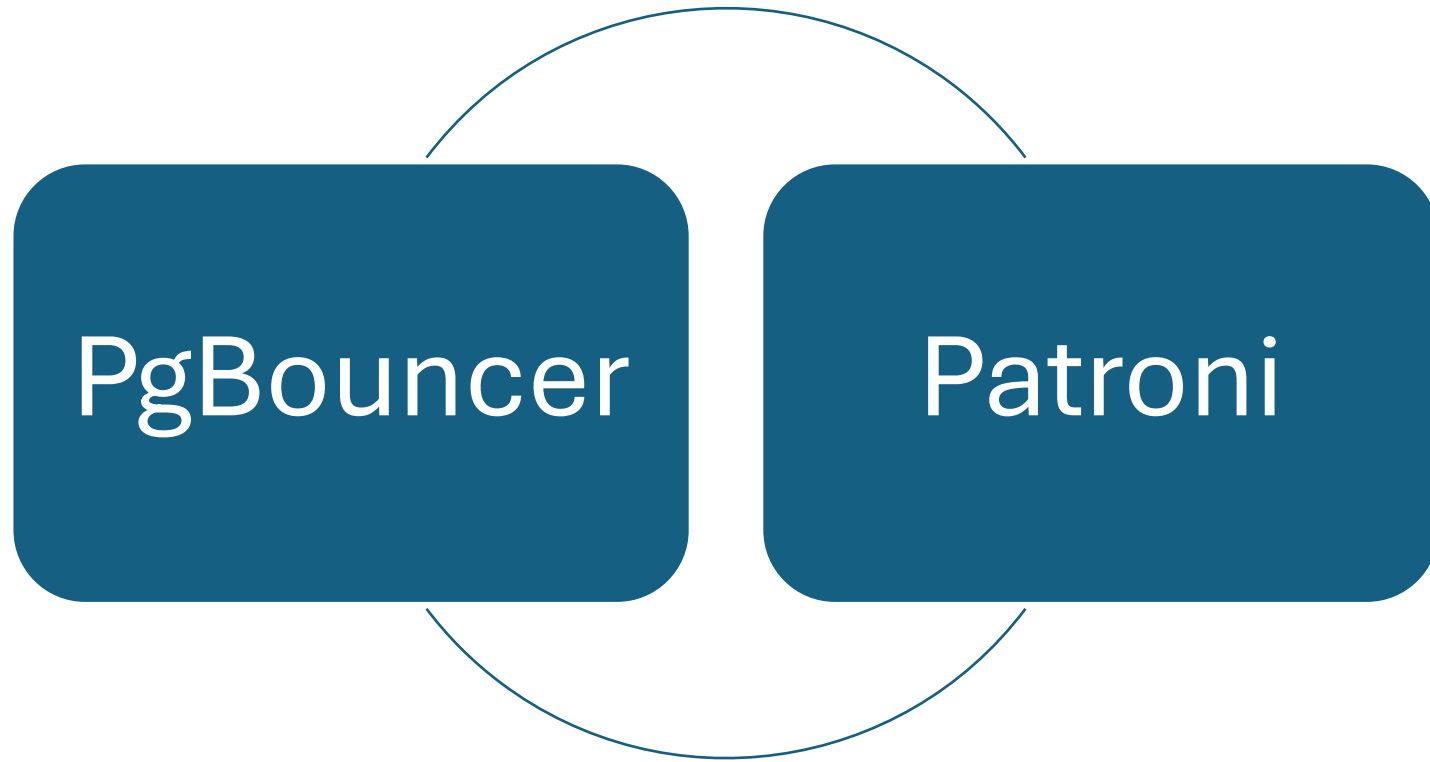


**Restore Command:**



```
psql -U postgres -f /path/to/backup.sql
```

# Introduction to replication tools



# PgBouncer and Patroni



PLAY A CRITICAL ROLE IN  
MANAGING



POSTGRESQL DATABASES,

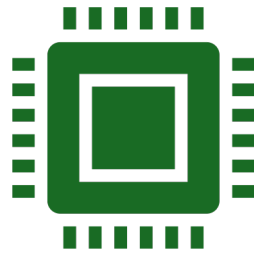


ESPECIALLY IN HIGH-  
DEMAND ENVIRONMENTS.

# PgBouncer and Patroni



Help ensure



high availability,  
scalability



efficient resource  
management.

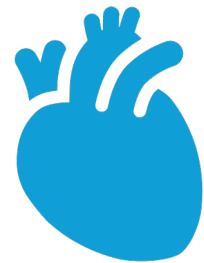
# PgBouncer



Lightweight



connection pooler



for PostgreSQL.

# PgBouncer



Helps manage database  
connections



efficiently by reducing  
the overhead associated



with establishing and  
closing connections.

# PgBouncer

Especially useful

Large numbers of

concurrent connections

might be required

# Key Features

1

Connection  
Pooling

2

Transaction  
Pooling

3

Query Pooling

4

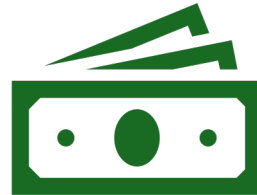
Load  
Balancing

# Connection Pooling

---



Reuses database  
connections,



reducing the  
overhead of



frequent connection  
establishment.

# Transaction Pooling

---



Efficiently manages  
connections



on a per-  
transaction basis,



which is useful for  
applications



with many short-  
lived transactions.

# Query Pooling

---



Allows pooling of  
queries



for better  
performance



in certain scenarios.

# Load Balancing

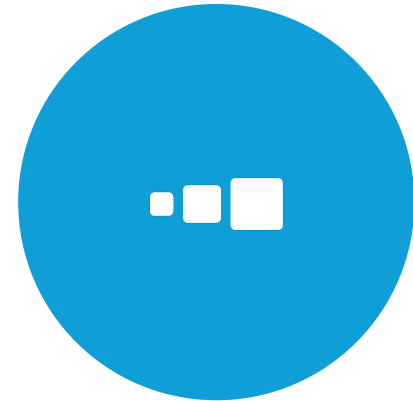
---



DISTRIBUTES  
CONNECTIONS ACROSS



MULTIPLE DATABASE  
INSTANCES



TO BALANCE THE LOAD.

# Benefits in Healthcare

---



IMPROVED  
PERFORMANCE



SCALABILITY



RESOURCE  
EFFICIENCY

# Improved Performance



REDUCES  
CONNECTION  
OVERHEAD AND



IMPROVES RESPONSE  
TIMES,



WHICH IS CRUCIAL  
FOR APPLICATIONS



THAT NEED TO  
DELIVER REAL-TIME  
DATA,



SUCH AS ELECTRONIC  
HEALTH RECORDS  
(EHR) SYSTEMS.

# Scalability

Manages large numbers of

concurrent connections effectively,

supporting the scalability of

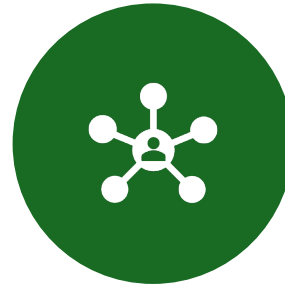
healthcare applications.

# Resource Efficiency

---



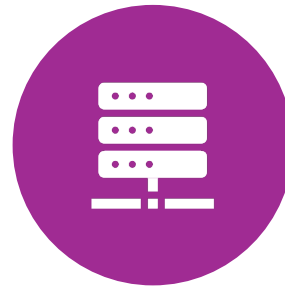
Reduces the load on  
PostgreSQL servers



by pooling connections,



leading to more  
efficient



use of server resources.

# Example Configuration

---

**pgbouncer.ini:**

**[databases]**

**mydatabase = host=localhost dbname=mydatabase**

# Example Configuration



[pgbouncer]



listen\_addr = 127.0.0.1



listen\_port = 6432

# pgbouncer.ini



`auth_type = md5`



`auth_file = /etc/pgbouncer/userlist.txt`



`pool_mode = transaction`

# Starting PgBouncer

```
pgbouncer -d /etc/pgbouncer/pgbouncer.ini
```

# Patroni



Patroni is an open-source tool



designed to manage



high-availability  
PostgreSQL clusters.

# Patroni

---



Automates the failover  
process



Provides a robust  
solution



for maintaining  
database



uptime and continuity

# Key Features

Automatic Failover

Leader Election

Configuration Management

Integration with Etcd/Consul/Zookeeper.

# Automatic Failover



AUTOMATICALLY  
DETECTS AND

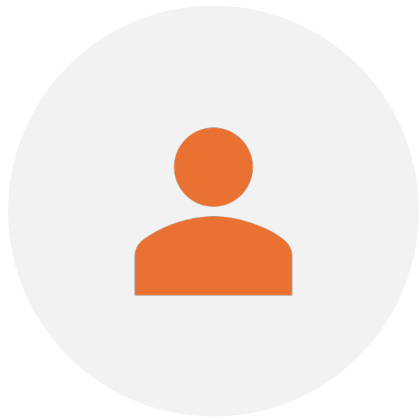


HANDLES FAILURES BY

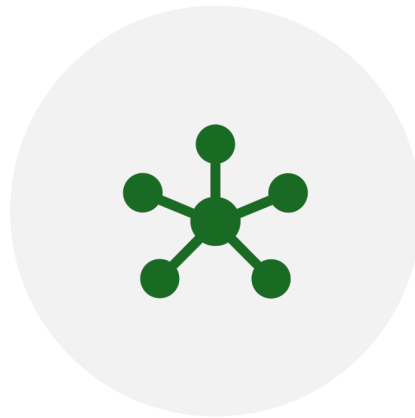


PROMOTING STANDBY  
SERVERS TO PRIMARY.

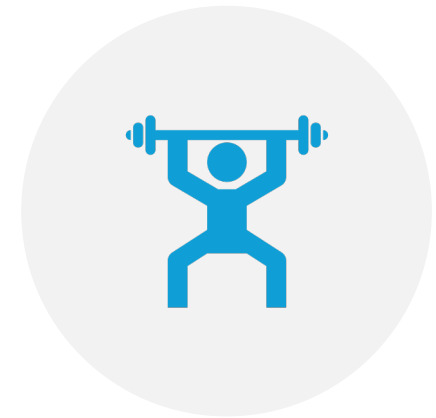
# Leader Election



MANAGES THE LEADER  
ELECTION PROCESS



IN A CLUSTER TO  
ENSURE



THERE IS ALWAYS ONE  
ACTIVE PRIMARY.

# Configuration Management



HANDLES DYNAMIC  
CONFIGURATION



CHANGES AND UPDATES.

# Integration with Etcd/Consul/Zookeeper



USES DISTRIBUTED  
KEY-VALUE STORES



FOR CONSENSUS AND



CONFIGURATION  
MANAGEMENT.



# Benefits in Healthcare



High Availability

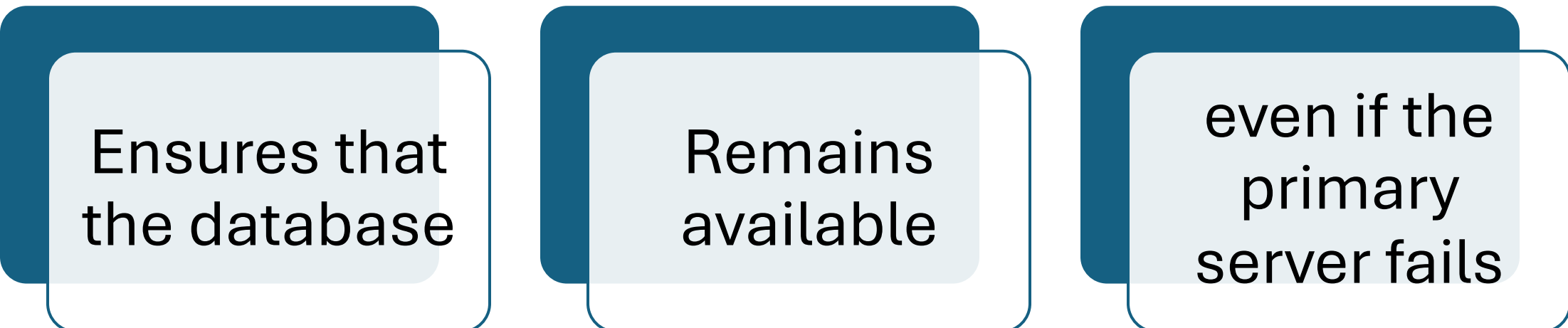


Disaster Recovery



Automatic Management

# High Availability



Ensures that  
the database

Remains  
available

even if the  
primary  
server fails

# Disaster Recovery



FACILITATES QUICK  
RECOVERY



FROM FAILURES BY  
PROMOTING



STANDBY SERVERS TO  
PRIMARY

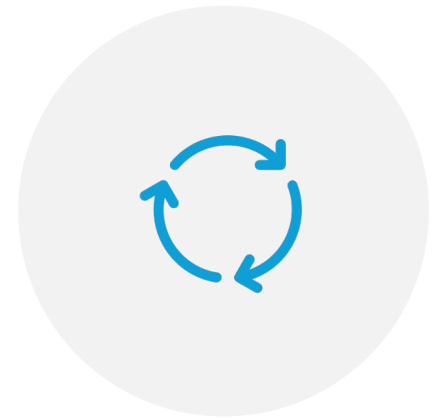
# Automatic Management



REDUCES ADMINISTRATIVE  
OVERHEAD



BY AUTOMATING FAILOVER



FAILBACK PROCESSES.

# Example Configuration

**patroni.yml:**  
**yaml code**

```
scope: my_cluster  
namespace: /db/  
name: pg1
```

```
restapi:  
  listen: 0.0.0.0:8008  
  connect_address: 127.0.0.1:8008
```

# patroni.yml

etcd:

host: 127.0.0.1:2379

postgresql:

listen: 0.0.0.0:5432

connect\_address: 127.0.0.1:5432

data\_dir: /var/lib/postgresql/12/main

bin\_dir: /usr/lib/postgresql/12/bin

# patroni.yml

authentication:

  superuser:

    username: postgres

    password: supersecretpassword

replication:

  username: replicator

  password: supersecretpassword

# patroni.yml

parameters:

wal\_level: replica

archive\_mode: on

archive\_command: 'cp %p /var/lib/postgresql/archive/%f'

max\_wal\_senders: 5

max\_replication\_slots: 5

# Starting Patroni

```
patroni /etc/patroni/patroni.yml
```



**Thank you for  
your support and  
patience**

**Surendra Panpaliya**  
**Founder and CEO**  
**GKTCS Innovations**

<https://www.gktcs.com>