2ndQuadrant®
PostgreSQL

Robust Replication strategies in PostgreSQL: Repmgr

Amruta Deolasee
Pallavi Sontakke

PostgreSQL
the world's most advanced open source database

Topics

- Replication strategies in PostgreSQL
- Repmgr as a Cluster Management Solution

Write Ahead Logging : WAL

- PostgreSQL inherently uses this for data integrity
- Provides:
  - Atomicity
  - Durability
- For consistency of each transaction:
  - XLOG/WAL is written
  - Then database is written

Write Ahead Logging : WAL

- In the event of crash:
  - We can recover database using:
    - checkpoint or base-backup
    - and WAL log

- WAL logs stay in:
  - *pg_xlog* (older) */ pg_wal* in data directory

Using WAL : Requires non-caching disk

- Sometimes, disk drive can:
  - cache data
  - report successful write to the kernel
- Issue?
  - Data is not yet stored on the disk
- Crash in this state:
  - Can cause data corruption

WAL And Replication

- Write-ahead log records:
  - Can also be used to keep the data in sync between the database servers.
  - This is achieved in two ways:
    - File-Based Log Shipping
    - Streaming WAL Records
      - Synchronous
      - Asynchronous

Replication: File-Based Log Shipping

- WAL log files are shipped
  - from the master to the standby servers

- i.e. Master can
  - directly copy the logs to standby server storage
  - or can share storage with the standby servers.

File-Based Log Shipping - Drawbacks

- WAL file is shipped only after it reaches that threshold (~ 16 MB)
- This causes:
  - a delay in replication
  - increased chance of losing data:
    - if the master crashes
    - and logs are not archived

Replication: Streaming WAL Records

- Master sends/ streams:
  - WAL record chunks to the standby server
  - as they are generated
- Standby server :
  - connects to the master to receive the WAL chunks.

Streaming WAL Records: Pros

- is more granular
- need not wait for the WAL file to be filled
- This allows:
  - Standby server to stay more up-to-date than is possible with file-based log shipping.
- is mostly asynchronous:
  - for performance

Streaming Replication: Synchronous

- Synchronous replication:
  - confirms for the user:
    - all changes made by a transaction have been successfully transferred to standby/s
  - (+) provides greater data protection
    - e.g. financial applications
  - (-) performance penalty due to round trip

Streaming Replication: Asynchronous

Asynchronous replication:

- WAL records are shipped to standby:
- after transaction commit on primary

- (+) has improved performance
- (-) Data loss scenario! :
  - transactions were ongoing on primary
  - and it suffers a crash
  - So there are transactions not yet shipped to standby server !

Streaming Replication: Usual GUC parameters

- *wal_level*
- *hot_standby*
- *archive_mode*

GUC parameter: wal_level

- determines:
  - type of replication that can be setup on the server
  - how much information is written to the WAL.
- *minimal*
  - default
  - writes only the information needed to recover from a crash or immediate shutdown.

GUC parameter: wal_level

- *replica*
  - used for streaming replication
  - adds logging required for WAL archiving as well as information required to run read-only queries on a standby server
- *logical*
  - used for logical replication
  - adds information necessary to support logical decoding.

GUC parameter: wal_level

- Each level includes:
  - the information logged at all lower levels.
- this parameter can only be set at server start.

GUC parameter: hot_standby

- *off*
  - disables
- *on*
  - is specified on standby/slave
  - enables read only connection on the node.
- is ignored
  - when the server is running as master.

Enabling *hot_standby*: Uses

- standby can balance load by serving read queries
- primary can move from recovery through to normal operation
  - while users continue running queries on standby

GUC parameter: archive_mode

- *off*
  - disables archive mode by default
- *on/ always*
  - enables archive_mode
  - completed WAL segments are sent to archive storage by setting *archive_command*

GUC parameter: archive_mode

- *on*
  - archives only the WAL that it generated itself
  - i.e. on primary
- *always*
  - archives every WAL segment it receives
  - i.e. on standby

GUC parameter: archive_mode

- *always*
  - standby receives WAL segments
    - during restoring from the archive
      - i.e. when WAL segments are played back into the Database
    - during streaming replication from primary

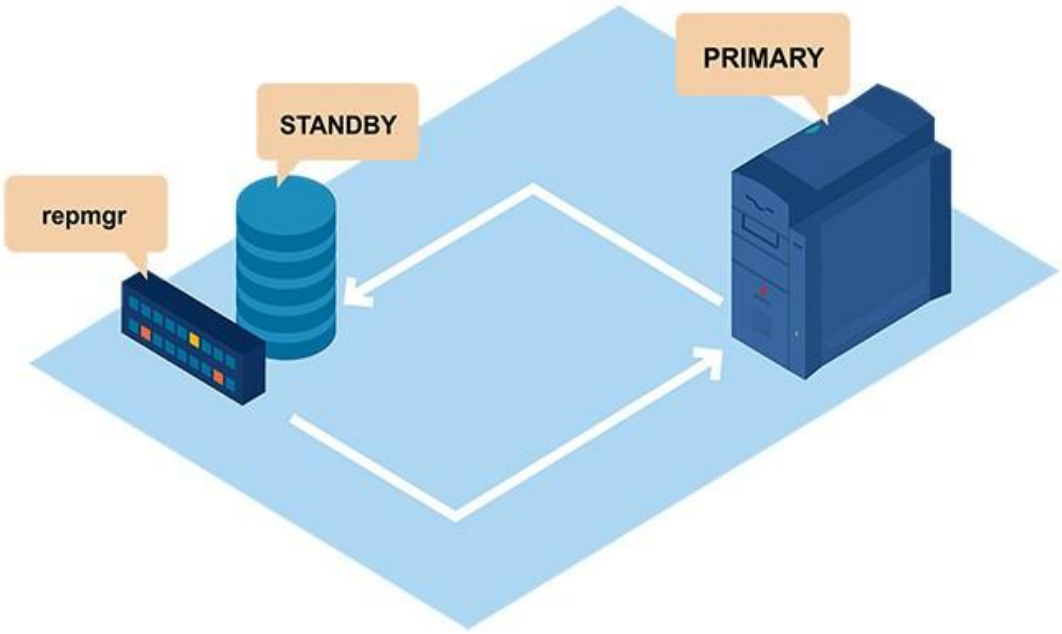
Enabling *archive_mode*: Use

- saves against Data Center or whole continent failure

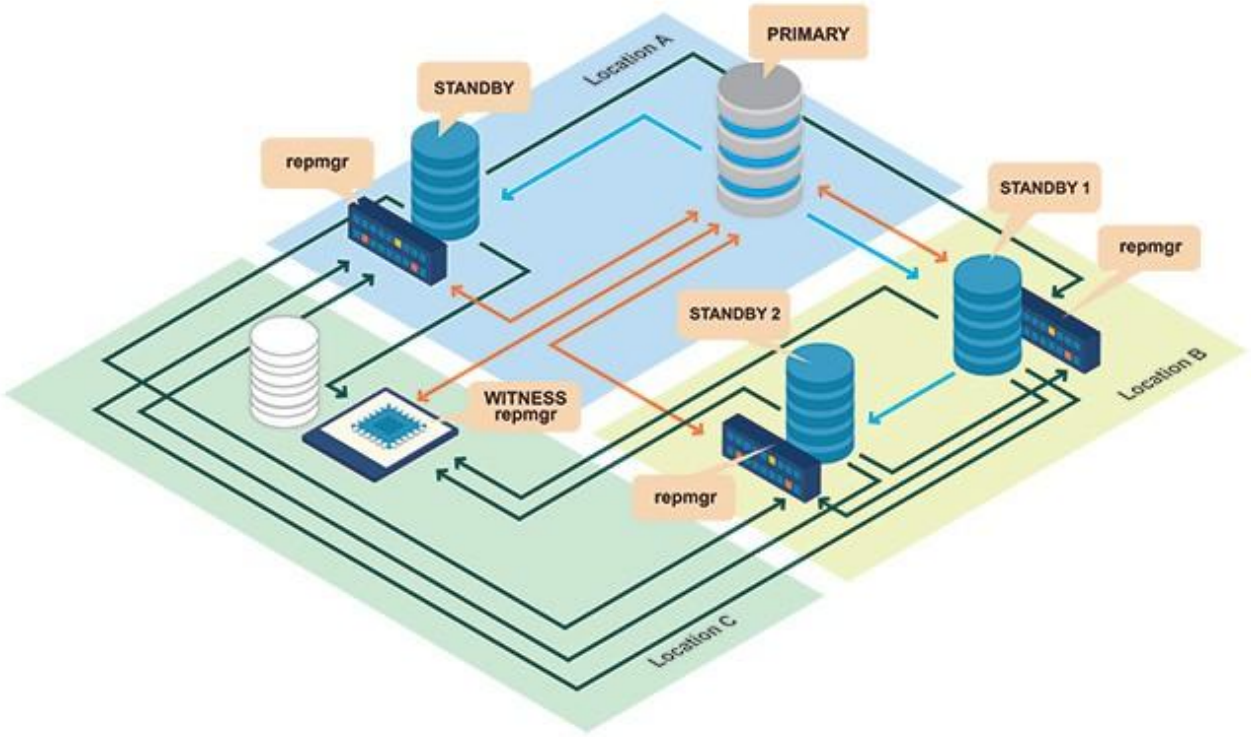
repmgr

- **rep**lication **man**ager
- Cluster Management Solution
- Wraps built-in commands
- Two main roles
  - set up and manage a (streaming) replication cluster
  - manual/automatic failover and monitoring

simple repmgr setup



complex repmgr setup



Set up a replication cluster

- Prerequisites:
  - PostgreSQL and repmgr must be installed on all servers.
  - connections between the PostgreSQL port (default: 5432) must be possible on all nodes
- Setting up RepMgr on primary postgres node
- Adding nodes to the cluster

Setting up RepMgr on primary postgres node

2ndQuadrant® PostgreSQL

- Create repmgr user (superuser or replication privilege)
- Create db for metadata
- Adjust postgresql.conf and pg_hba.conf
- Prepare repmgr.conf

Settings on primary

2ndQuadrant® PostgreSQL

- Adjust postgresql.conf

```
max_wal_senders = 10
wal_level = 'hot_standby'
hot_standby = on
archive_mode = on
archive_command = '/bin/true'
```

Settings on primary

2ndQuadrant® PostgreSQL

- Create repmgr.conf

```
node_id=1
node_name=node1
conninfo='host=node1 user=repmgr dbname=repmgr connect_timeout=2'
data_directory='/var/lib/postgresql/data'
```

Syntax

- General pattern: repmgr [options] <object> <verb>
- object ∈ {primary, standby, node, cluster , witness}
- verb ∈ { register , clone, follow, switchover, check, show, . . .}
- eg: repmgr -f /etc/repmgr.conf primary register

Start a cluster

```
$ repmgr -f /etc/repmgr.conf primary register
```

```
    INFO: connecting to primary database...
NOTICE: attempting to install extension "repmgr"
NOTICE: "repmgr" extension successfully installed
NOTICE: primary node record (id: 1) registered
```

- Installs repmgr extension
- metadata objects
- adds metadata record for primary server

Status and Record

2ndQuadrant® PostgreSQL

```
$ repmgr -f /etc/repmgr.conf cluster show
 ID | Name  | Role    | Status    | Upstream | Connection string
----+-------+---------+-----------+----------+-----------------------------------------------------
 1  | node1 | primary | * running |          | host=node1 dbname=repmgr user=repmgr
connect_timeout=2

repmgr=# SELECT * FROM repmgr.nodes;
-[ RECORD 1 ]----+-----------------------------------------------------
node_id          | 1
upstream_node_id |
active           | t
node_name        | node1
type             | primary
location         | default
priority         | 100
conninfo         | host=node1 dbname=repmgr user=repmgr connect_timeout=2
repluser         | repmgr
slot_name        |
config_file      | /etc/repmgr.conf
```

Preparing the standby

- Create repmgr.conf

```
node_id=2
node_name=node2
conninfo='host=node2 user=repmgr dbname=repmgr
connect_timeout=2'
data_directory='/var/lib/postgresql/data'
```

Clone the Standby

```
$ repmgr -h node1 -U repmgr -d repmgr -f /etc/repmgr.conf standby clone
```

```
    NOTICE: destination directory "/var/lib/postgresql/data" provided
INFO: connecting to source node
DEBUG: upstream_node_id determined as 1
INFO: checking and correcting permissions on existing directory "/var/lib/postgresql/data"
NOTICE: starting backup (using pg_basebackup)...
INFO: executing:
  /usr/lib/postgresql/11/bin/pg_basebackup -l "repmgr base backup"  -D /var/lib/postgresql/data -h node1
-U repmgr -X stream --no-slot
DEBUG: create_recovery_file(): creating "/var/lib/postgresql/data/recovery.conf"...
DEBUG: recovery file is:
standby_mode = 'on'
primary_conninfo = 'host=node1 user=repmgr application_name=node3'
recovery_target_timeline = 'latest'

NOTICE: standby clone (using pg_basebackup) complete
NOTICE: you can now start your PostgreSQL server
```

Verify that replication is functioning

```
repmgr=# SELECT * FROM pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 19111
usesysid         | 16384
usename          | repmgr
application_name | node2
client_addr      | 192.168.1.12
client_hostname  |
client_port      | 50378
backend_start    | 2019-01-28 15:14:19.851581+09
backend_xmin     |
state            | streaming
sent_location    | 0/7000318
write_location   | 0/7000318
flush_location   | 0/7000318
replay_location  | 0/7000318
sync_priority    | 0
sync_state       | async
```

Register standby and view the cluster

$ repmgr -f /etc/repmgr.conf standby register

NOTICE: standby node "node2" (ID: 2) successfully registered

$ repmgr -f /etc/repmgr.conf cluster show

```
 ID | Name  | Role    | Status    | Upstream | Location | Connection string
----+-------+---------+-----------+----------+----------+--------------------------------------
 1  | node1 | primary | * running |          | default  | host=node1 dbname=repmgr user=repmgr
 2  | node2 | standby |   running | node1    | default  | host=node2 dbname=repmgr user=repmgr
```

In short

- **Adjust configuration files**
- **repmgr primary register --------- on primary server**
- **repmgr standby clone --------- on standby server**
- **repmgr standby register --------- on standby server**

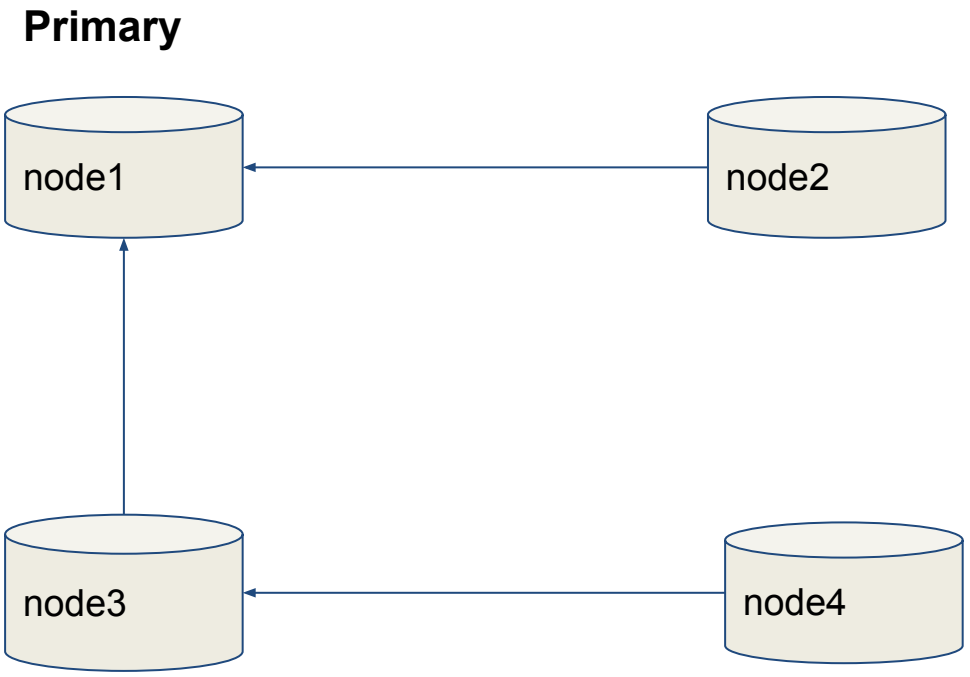
Repmgr High Availability : Use Cases

- Repmgr cluster comes in handy for these use-cases:
  - failure of primary server
    - Failover
  - planned maintenance of primary server
    - Switchover
  - Automatic failover
  - Split brain scenario

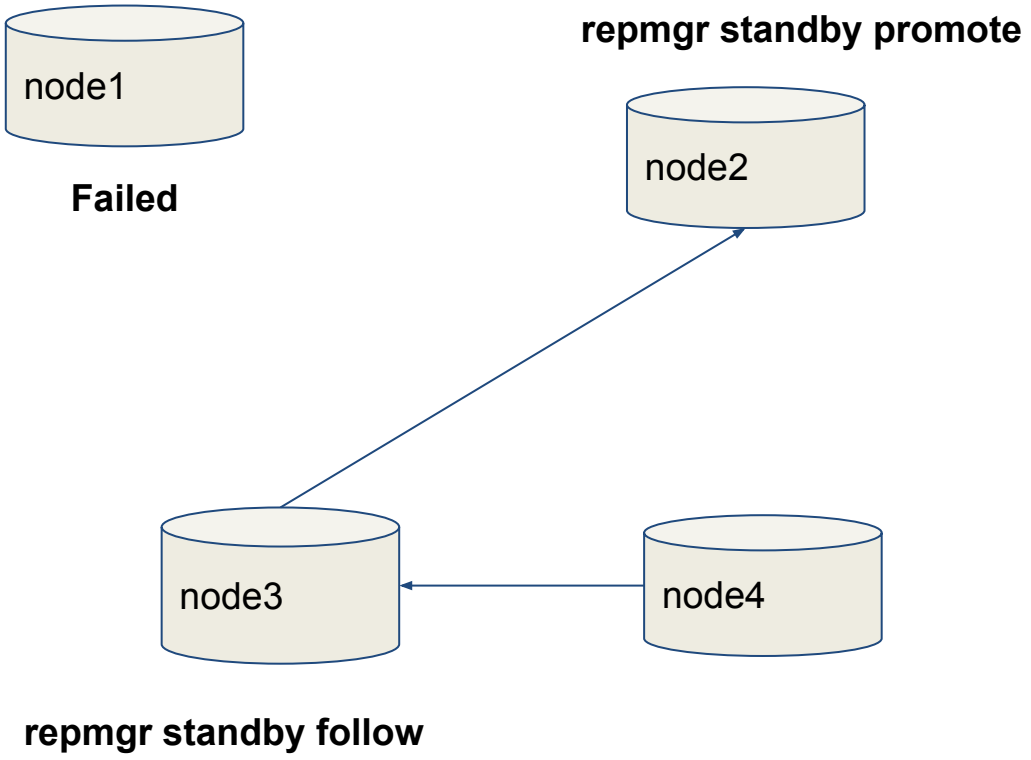
Repmgr HA Use Case: Failover

- Primary upstream node becomes unreachable/ fails
- promote one of the standby servers so that applications can failover to this server

Failover



Failover



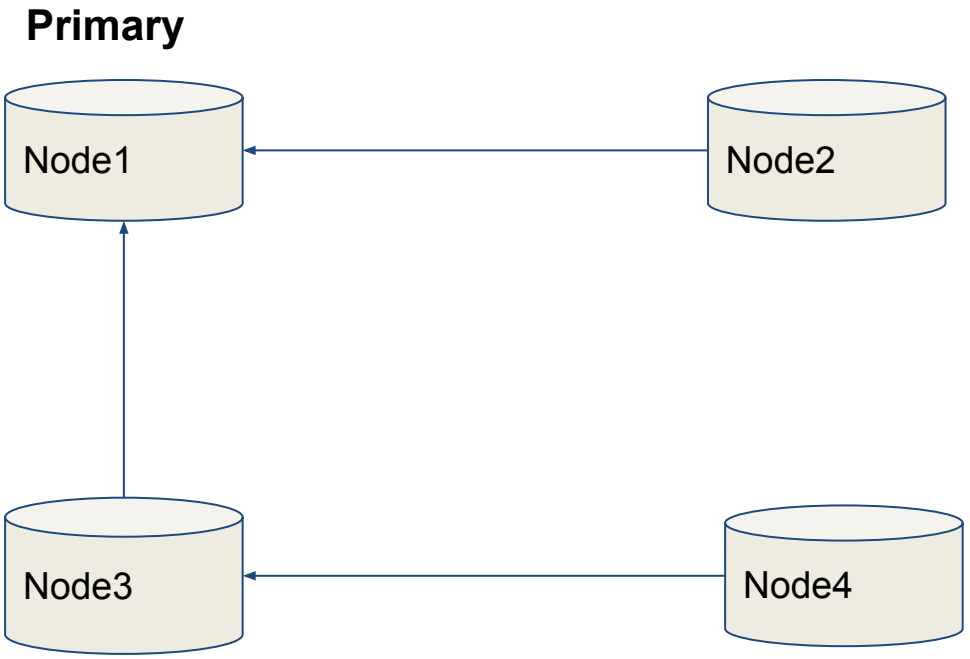
Failover: View cluster status

```
$ repmgr -f /etc/repmgr.conf cluster show
 ID | Name  | Role    | Status    | Upstream | Location | Connection string
----+-------+---------+-----------+----------+----------+--------------------------------------------------
 1  | node1 | primary | - failed  |          | default  | host=node1 dbname=repmgr user=repmgr
 2  | node2 | primary | * running |          | default  | host=node2 dbname=repmgr user=repmgr
 3  | node3 | standby |   running | node2    | default  | host=node3 dbname=repmgr user=repmgr
 4  | node4 | standby |   running | node3    | default  | host=node4 dbname=repmgr user=repmgr
```

Repmgr HA Use Case: Switchover

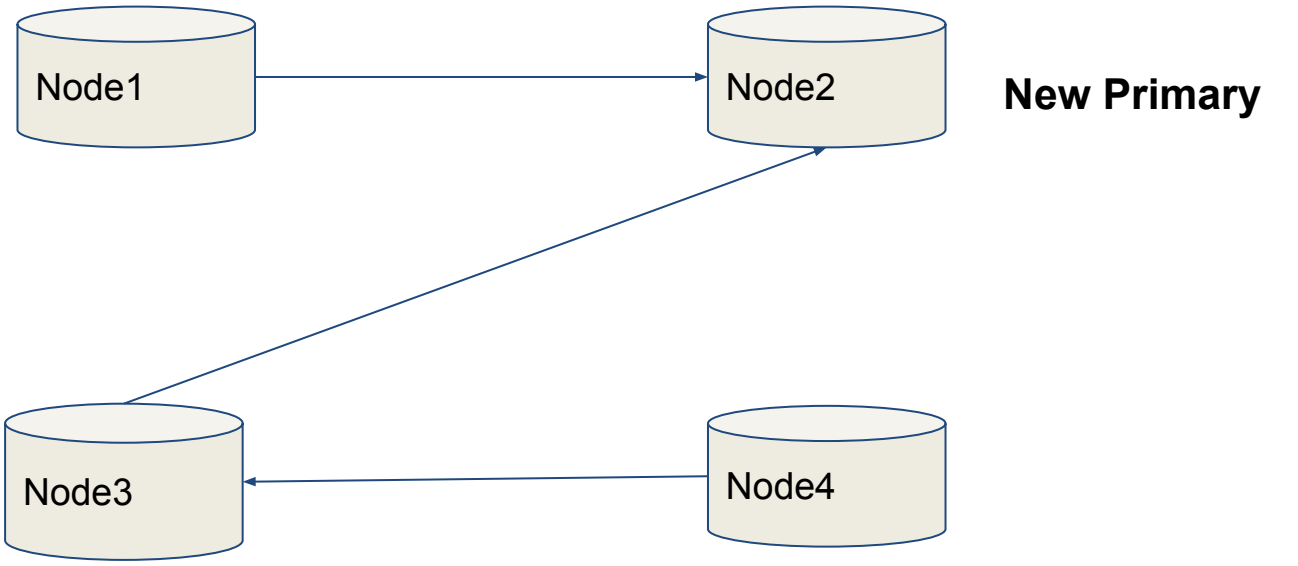
- when primary upstream node needs maintenance-downtime
- we can promote one of the standby servers
- so that applications can switchover to this server

Switchover



Switchover

repmgr standby switchover --siblings-follow



Automatic Failover with repmgrd

- management and monitoring **daemon**
- runs on each node in a replication cluster
- automate actions
  - failover
  - updating standbys to follow the new primary
- monitoring

Configuration set up for repmgrd

- Adjust postgresql.conf

```
shared_preload_libraries = 'repmgr'
```

- Adjust repmgr.conf

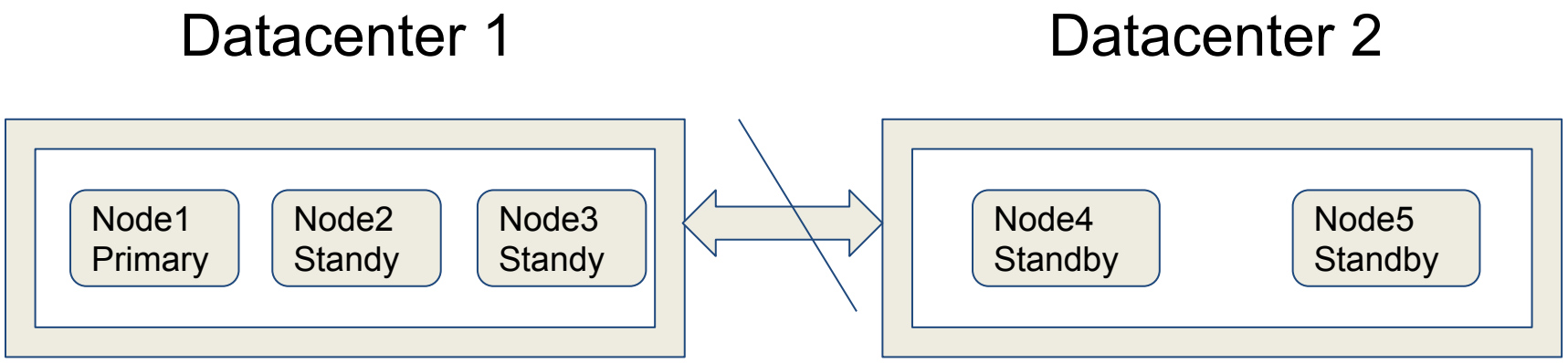
```
failover=automatic
promote_command='/usr/bin/repmgr standby promote -f
/etc/repmgr.conf --log-to-file'
follow_command='/usr/bin/repmgr standby follow -f
/etc/repmgr.conf --log-to-file --upstream-node-id=%n'
```

Automatic failover demonstration

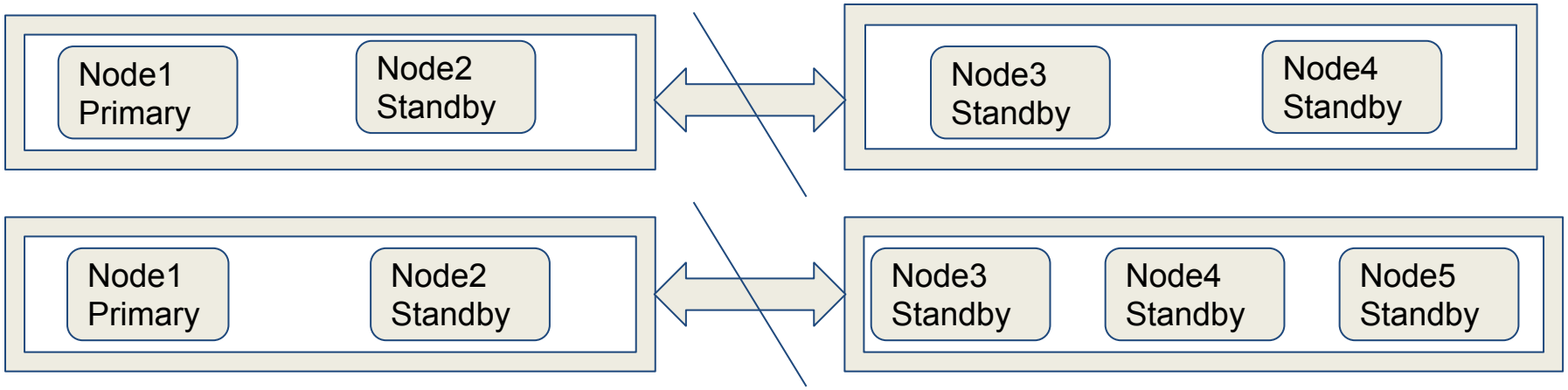
- Primary server fails
- deamon on node2 and node3 gets into action
- No manual action required

```
$ repmgr -f /etc/repmgr.conf cluster show
 ID | Name  | Role    | Status    | Upstream | Location | Connection string
----+-------+---------+-----------+----------+----------+-----------------------------------------------
 1  | node1 | primary | - failed  |          | default  | host=node1 dbname=repmgr user=repmgr
 2  | node2 | primary | * running |          | default  | host=node2 dbname=repmgr user=repmgr
 3  | node3 | standby |   running | node2    | default  | host=node3 dbname=repmgr user=repmgr
 4  | node4 | standby |   running | node3    | default  | host=node4 dbname=repmgr user=repmgr
```

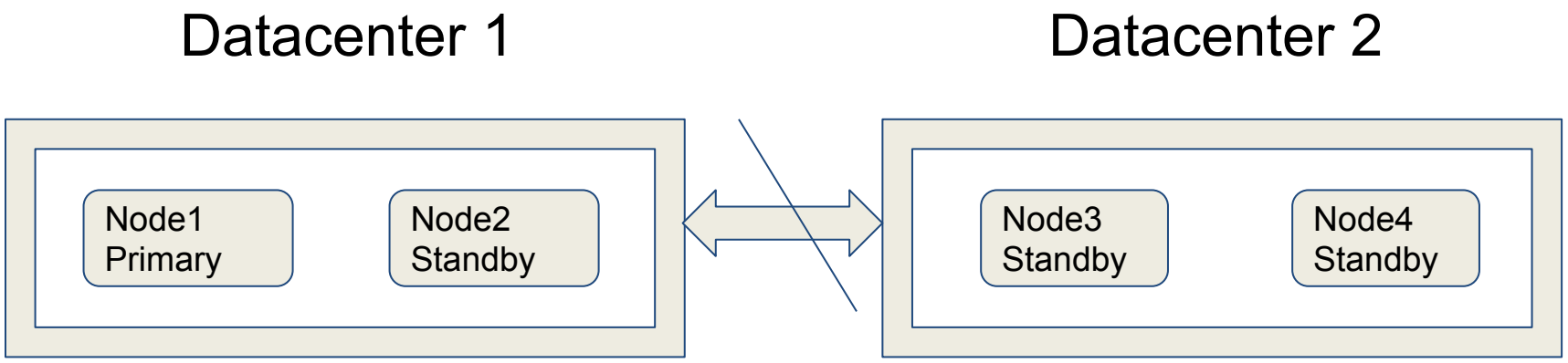
Problem Scenarios

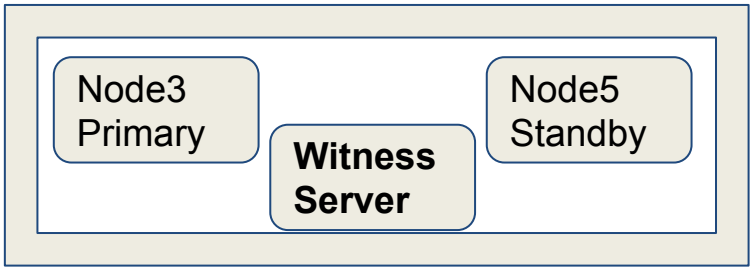


Problem Scenarios



Split Brain Scenario

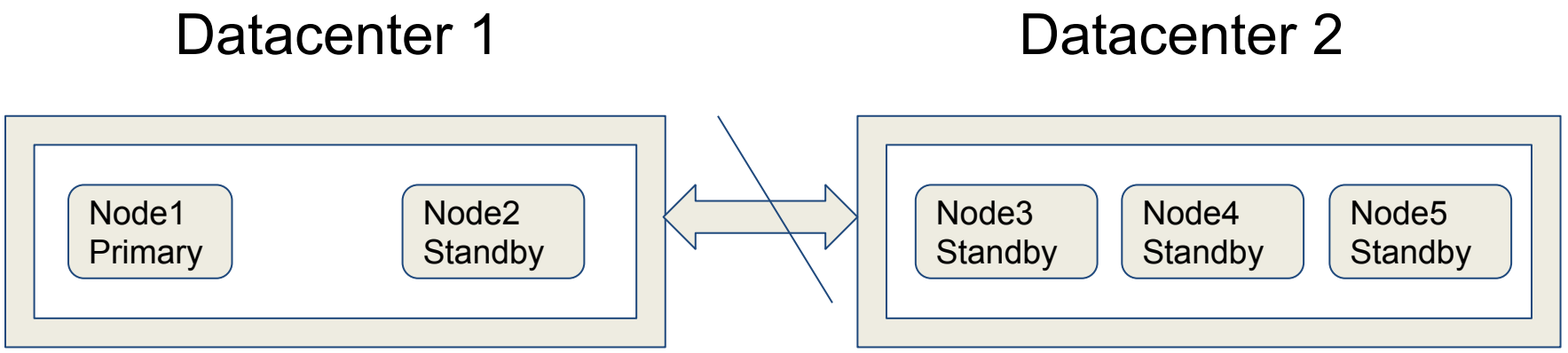




Solution:

- witness server in current primary location
- provides a "casting vote"
- Not a part of replication cluster
- **repmgr witness register**

Split Brain Scenario



Solution

- **'location'** parameter in repmgr.conf

```
node_id=1
node_name=node1
conninfo='host=node1 user=repmgr dbname=repmgr connect_timeout=2'
data_directory='/var/lib/postgresql/data'
location='dc1'
```

- repmgrd will check if any servers in the same location as the current primary node are visible.

We're hiring!

- Contact us:
- *info@2ndquadrant.com*

2ndQuadrant®
PostgreSQL