How to Achieve Near-Zero Database Downtime

Gianni Ciolli

VP, Field CTO

- High Availability
- PostgreSQL contributor (Hot Standby)
- Author of the PostgreSQL Administration Cookbook

Postgres Replication Options

Glossary I

Log Shipping Replication

(PostgreSQL feature, 2006)

Replication to one or more Standby nodes by shipping WAL files

Physical Streaming Replication

(PostgreSQL feature, 2010)

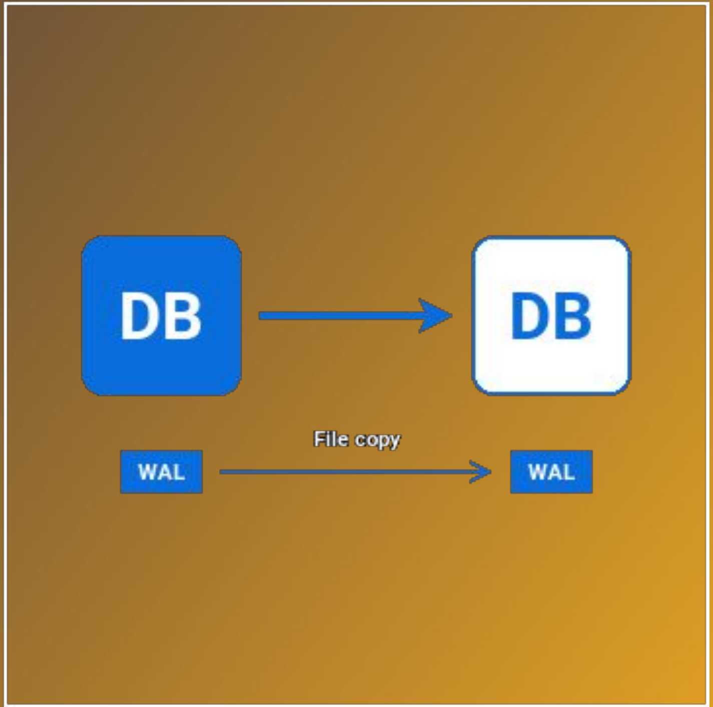
Replication to one or more Standby nodes by streaming WAL records

Glossary I

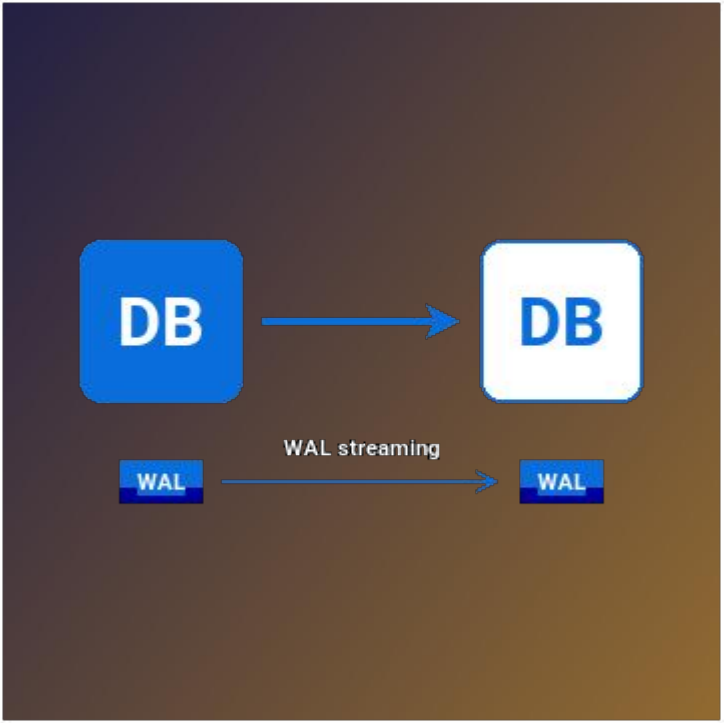
Log Shipping Replication

(PostgreSQL feature, 2006)

Replication to one or more Standby nodes by shipping WAL files



Glossary I



Physical Streaming Replication

(PostgreSQL feature, 2010)

Replication to one or more Standby nodes by streaming WAL records

Glossary II

Logical Decoding

(PostgreSQL feature, 2014)

Extraction of DML changes from WAL

pglogical

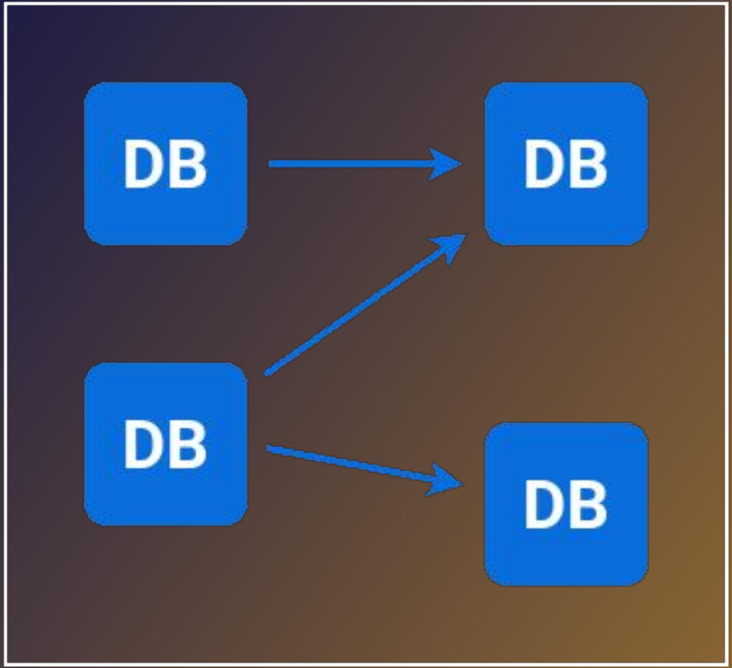
(tool using Logical Decoding, 2015)

(both) Replication to one or more subscriber nodes, by streaming DML changes

Logical Streaming Replication

(PostgreSQL feature, 2017)

Glossary II



pglogical

(tool using Logical Decoding, 2015)

(both) Replication to one or more subscriber nodes, by streaming DML changes

Logical Streaming Replication

(PostgreSQL feature, 2017)

Physical Replication: All and More

Transmits all changes to Postgres data files, including:

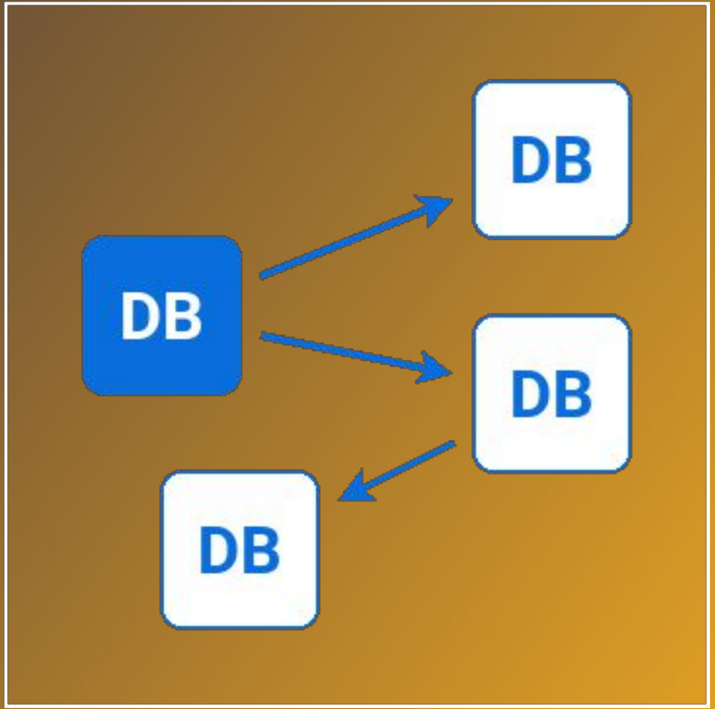
- **changes applied by DML**
- changes applied by DDL
  - **Index Creation**
  - Catalog Table Updates
  - **VACUUM**
- other changes
  - **Autovacuum**

Even in case of **transaction rollback!**

Glossary III

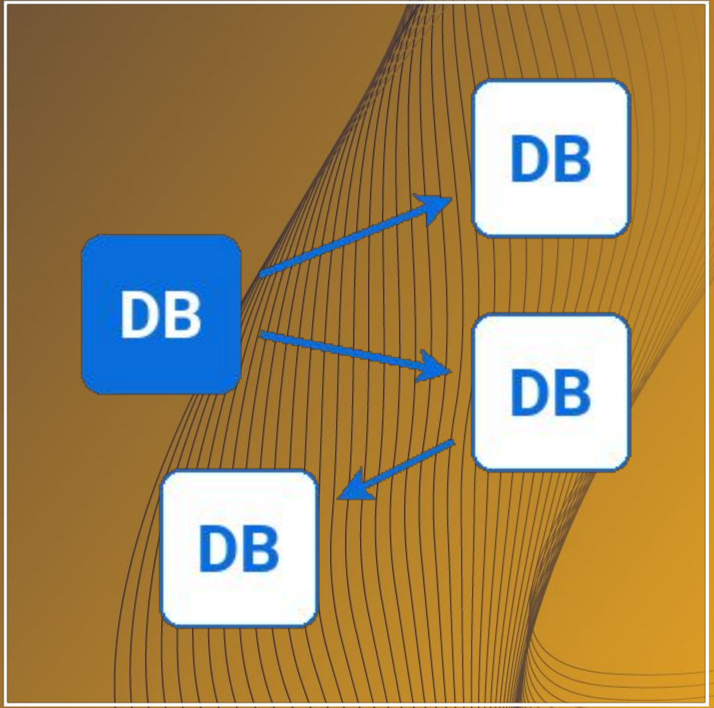
Physical Replication has

- one **Primary** node
- multiple **Standby** nodes

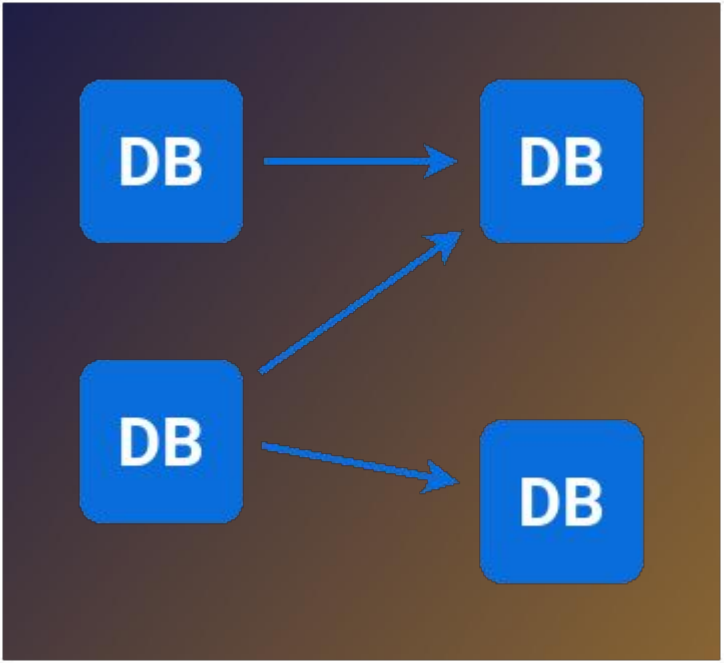


Physical Replication

- Transmits everything
- Cannot write on Standbys
- Replication of all tables/columns/rows and to all nodes
- All nodes must run the same major Postgres version
- Suitable for High Availability



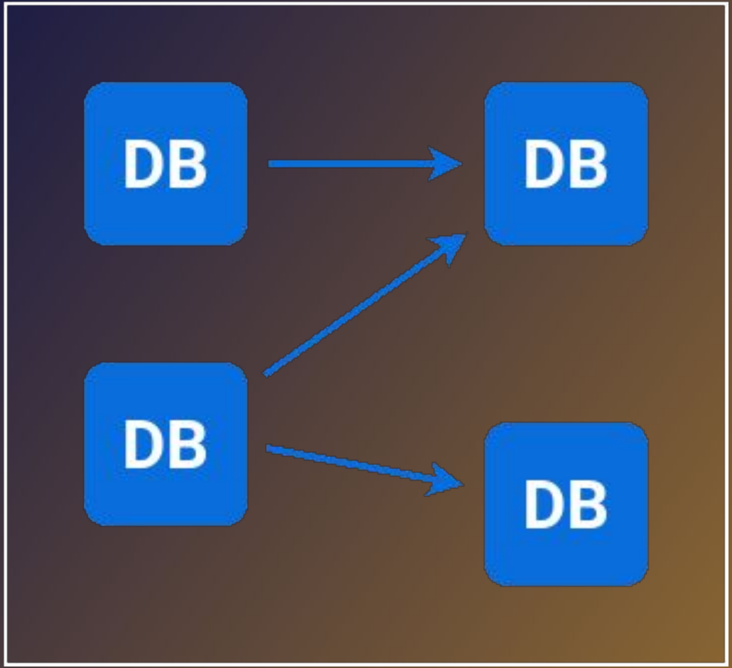
Glossary IV



Logical Replication has

- **Publisher** nodes
- **Subscriber** nodes

Logical Replication



- Transmits only DML
- Can write on Subscribers
- Replication of selected tables/columns/rows and to selected nodes
- Works across different major Postgres versions
- Not suitable for High Availability

Taking the best from both

Physical Replication

- Transmits everything
- Cannot write on Standbys
- Replication of all tables/columns/rows and to all nodes
- Hence all nodes must run the same major Postgres version

✓ **Suitable for High Availability**

Logical Replication

✓ **Transmits only DML**

✓ **Can write on Subscribers**

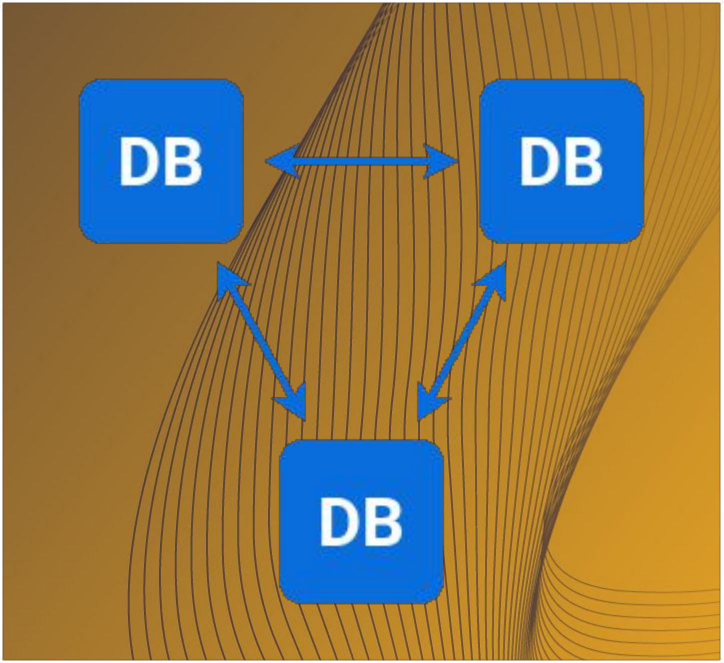
✓ **Replication of selected tables/columns/rows and to selected nodes**

✓ **Works across different major Postgres versions**

- Not suitable for High Availability

EDB Postgres Distributed

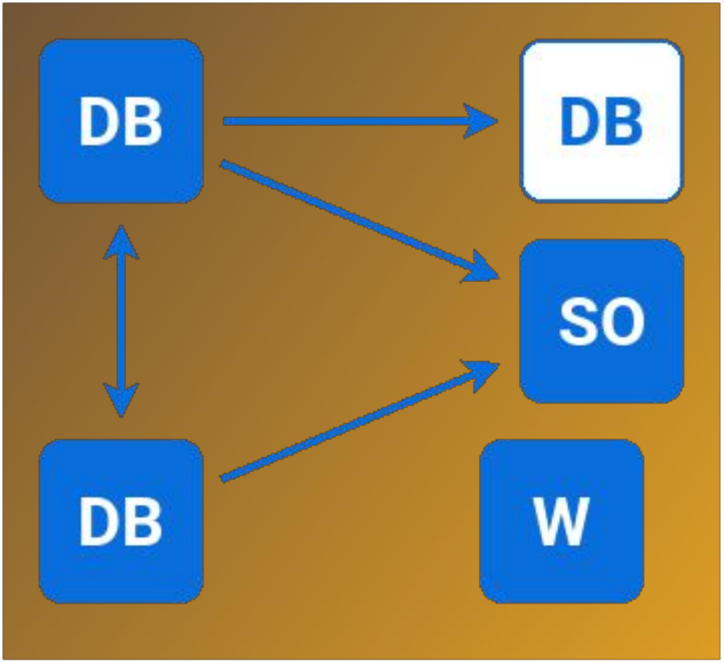
- ✓ Transmits **DML** and some **DDL**
- ✓ Can write on all nodes
- ✓ **Replication of selected tables/columns/rows and to selected nodes**
- ✓ **Works across different major Postgres versions**
- ✓ **Suitable for High Availability**



Glossary V

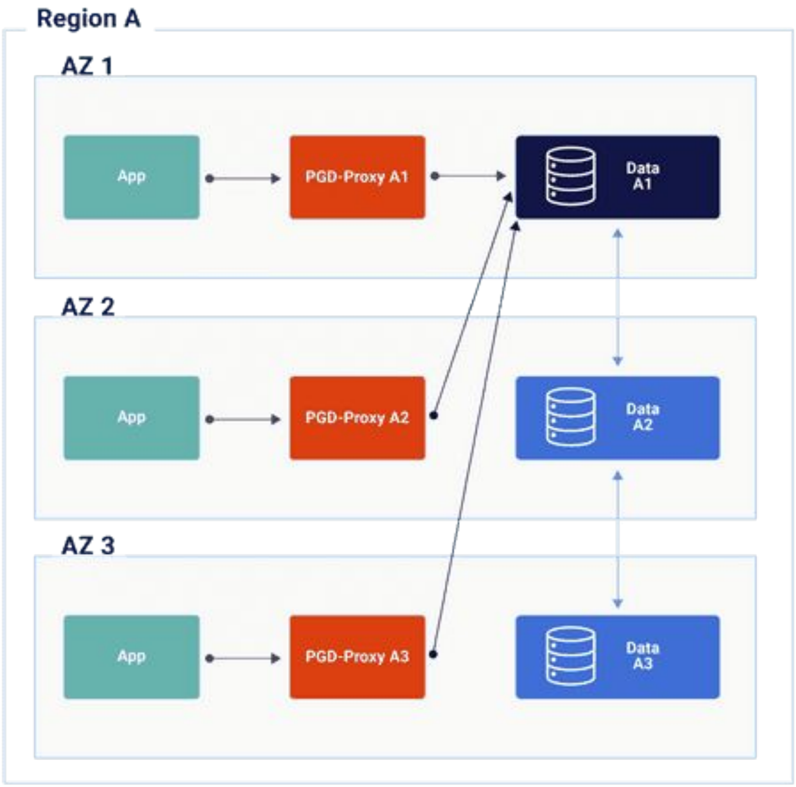
EDB Postgres Distributed has **Data** nodes, plus four other optional database node types:

- **Logical Standby**
- **Physical Standby**
- **Subscriber-Only**
- **Witness**

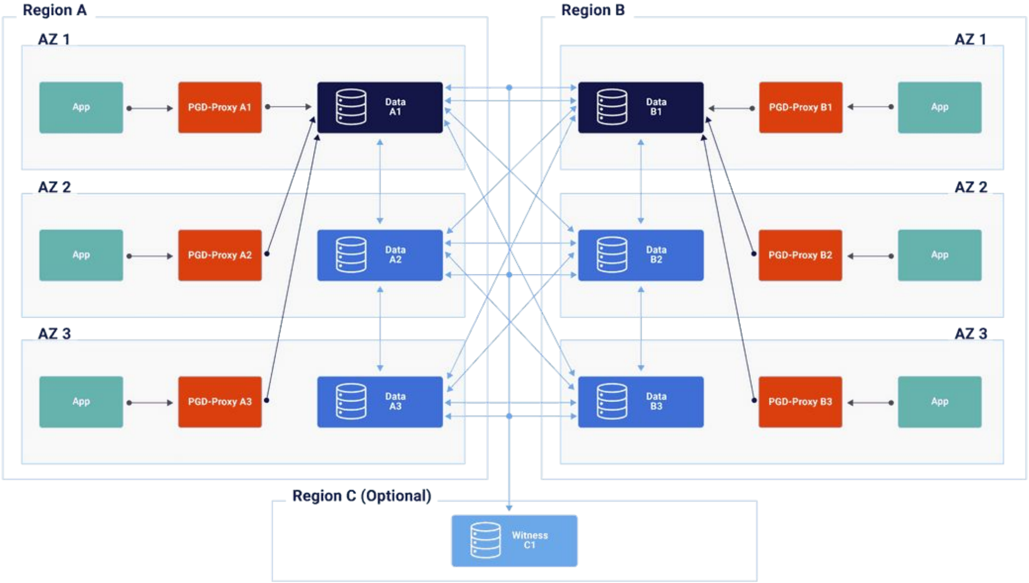


Five Nines Architecture Examples

Always On Single Location



Always On Two Locations



Preventing Data Loss with Distributed Workloads

More than one RPO / RTO

- High Availability requirements **depend** on the **failure scenario**
- E.g.
  “On single node failure we require maximum 5 seconds downtime and zero data loss, but if the entire datacenter fails recovery in 5 minutes is OK”

Only one Speed of Light

(the limit you cannot avoid)

- Durability requirements need to be mindful of distance in distributed databases:
  - Data does **not** travel instantly
  - At long distance, **latency is bigger**
  - Committed data that is waiting to be replicated **increases the RPO**

EDB Postgres Distributed

Three different kinds of durability requirements:

- **Group Commit**
- **Commit At Most Once** (CAMO)
- **Lag Control**

Commit Scopes

- Language for specifying **durability requirements**
- **Declarative** and flexible
- Dynamically defined
- Each transaction can **choose** its **commit scope**
- COMMIT returns when the **requirements are met**

EDB™

Example: Commit Scopes

```
SELECT bdr.add_commit_scope (
 commit_scope_name := ‘app1’ ,
 origin_node_group := ‘dc1’ ,
 rule := ‘ALL (dc1)
   GROUP COMMIT ON RECEIVED’);
```

- Creates a new commit scope called app1
- Specifies the rule for transactions originated inside datacenter 1

Example: Commit Scopes

```
BEGIN;
SET LOCAL bdr.commit_scope='app1';
...
COMMIT;
```

- Uses commit scope **app1**
- COMMIT returns when the transaction reaches the durability specified by **app1**

Group Commit

- Default kind of commit scope
- Protects transactions by requiring they are on multiple nodes
- Choose trade off between Latency and Consistency
- Can choose different modes for **commit decision** and **conflict resolution**

Example: Group Commit

```
ALL (dc1)
GROUP COMMIT ON visible
  AND
ANY 1 (dc2)
GROUP COMMIT ON received
```

- The transaction will be reported as committed if: *“it is visible on all nodes in datacenter 1, and moreover it has been received by at least one node in datacenter 2”*

Lag Control

- A different kind of commit scope
- Replication
- When the replication lag is too big, add a delay to each commit
- `max_lag_size` or `max_lag_time`
- `max_commit_delay`

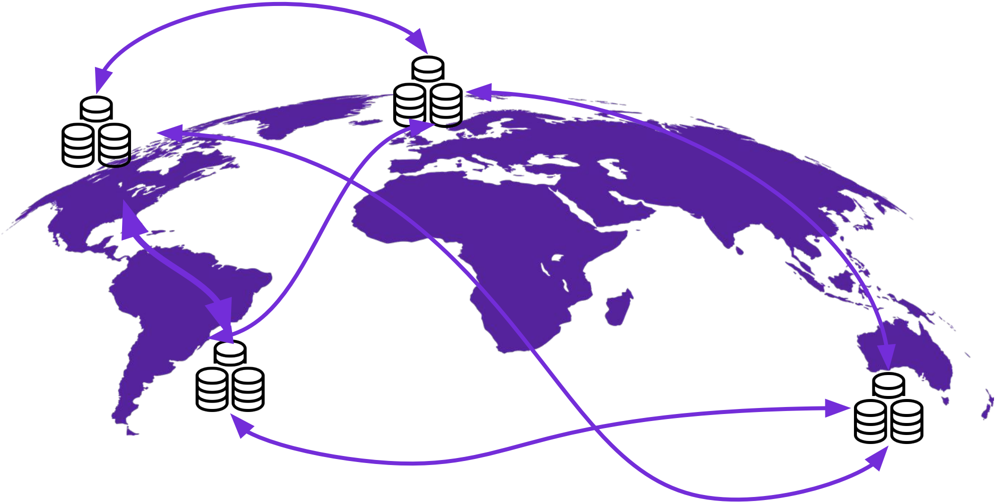
Example: Lag Control

```
LAG CONTROL (
  max_lag_size=1MB,
  max_commit_delay=100ms)
```

“When the replication lag gets higher than 1MB, we slow down each commit by adding a delay up to 100ms”

Try EDB Postgres Distributed Today

- ✓ Best-in-Class High Availability
- ✓ Geo-Distributed (Active/Active) Architectures
- ✓ Evolve Maintenance and Release With Confidence



Join us for the webinar

Register Today

Geo-Distributed Databases and Disaster Recovery for Postgres in the Cloud with BigAnimal

Focus: EDB Postgres Distributed on BigAnimal

Sept. 7, 2023 at 11 a.m. EST

Presenters:
Natalia Wojcik and Aaron Sonntag

END