PostgreSQL Streaming Replication

A Practical Guide to Replicating your PostgreSQL Database

Jim McQuillan - mug.org - November 10, 2020

What is Replication?

“**Replication** in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility”

https://en.wikipedia.org/wiki/Replication_(computing)

Why would you want to replicate?

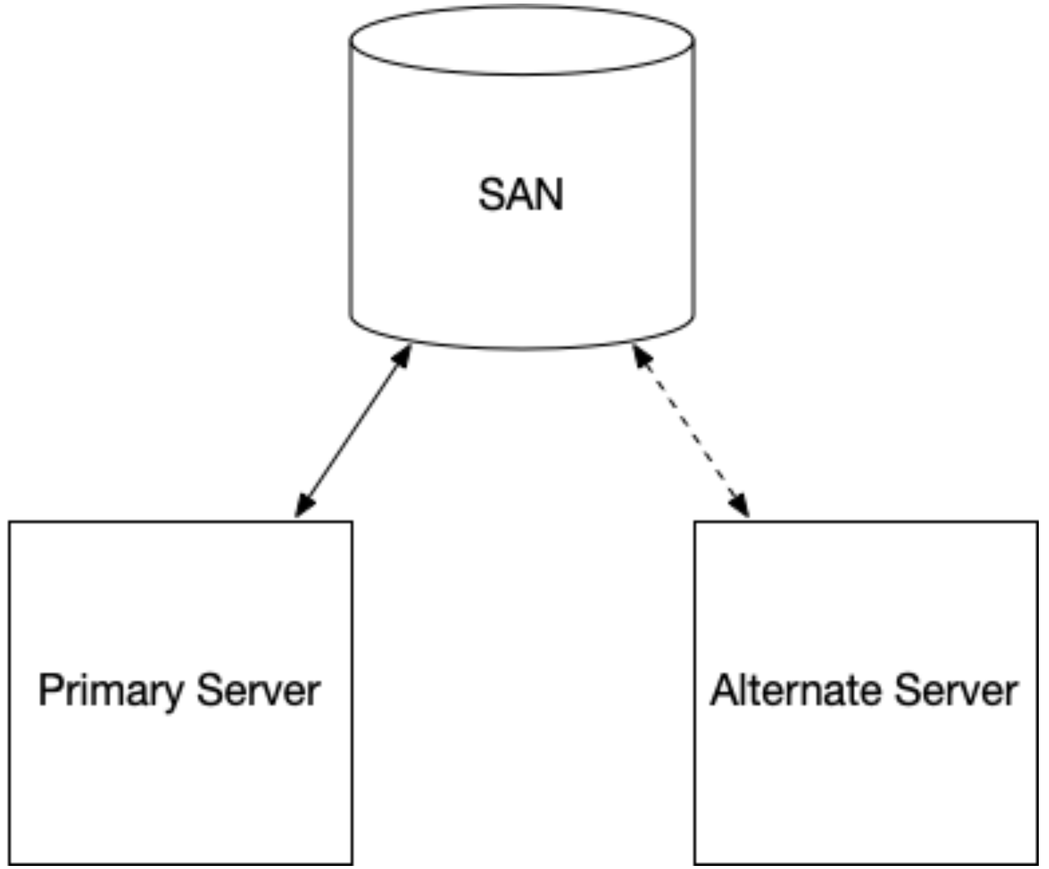
- Reliability - Reduce the possibility of losing valuable data
- Fault Tolerance - Keep running after a failure of one or more components
- Accessibility - The data is always accessible
- Performance - Increase the ability to handle read requests

Types of Replication

- Shared Disk Failover
- File System Replication
- Multi-Master
- Primary-Secondary (Master-Slave)

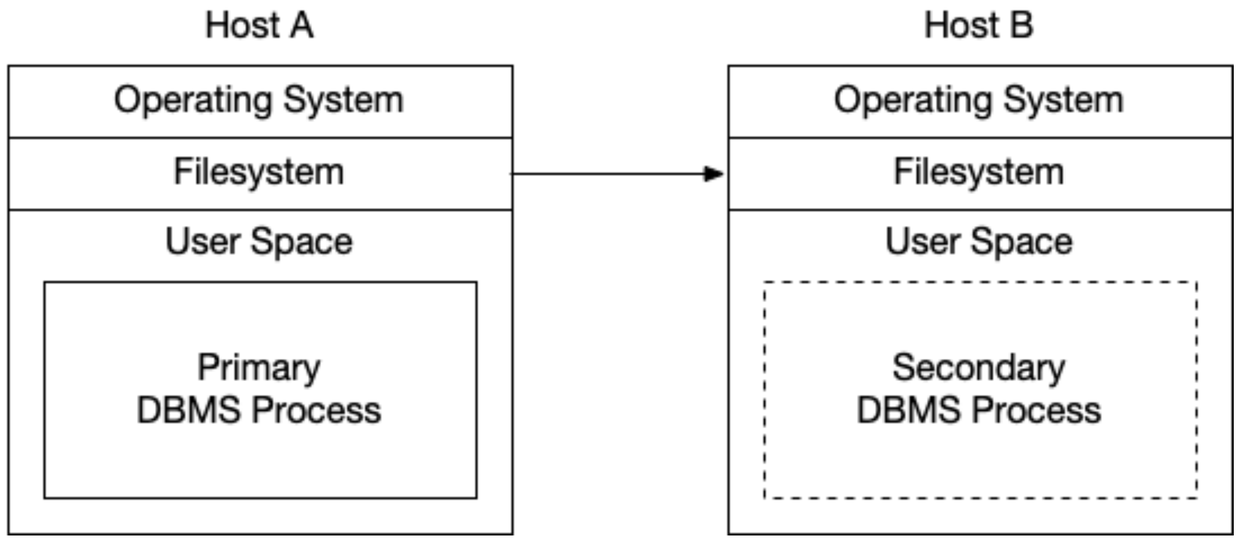
Shared Disk Failover

- Hardware level, usually SAN
- One copy of the database
- Multiple Database servers
- Only one DB server active at a time
- Avoids synchronization overhead
- Failure of SAN renders database unusable



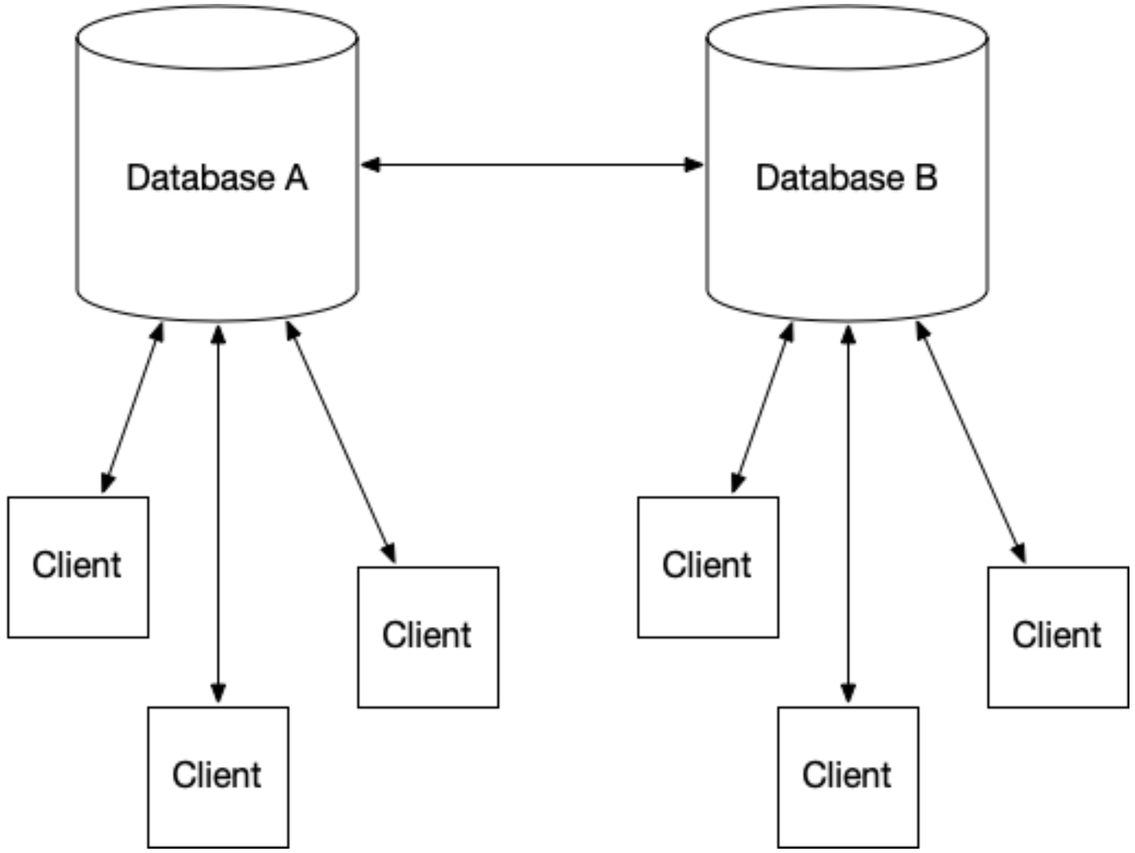
Filesystem Replication

- Operating system level
- All changes written to the filesystem are mirrored to a filesystem residing on another computer. DRBD is an example of a solution for Linux
- Caching in the DBMS can result in corrupt data being mirrored to the secondary server
- Only one DB server can be active at a time



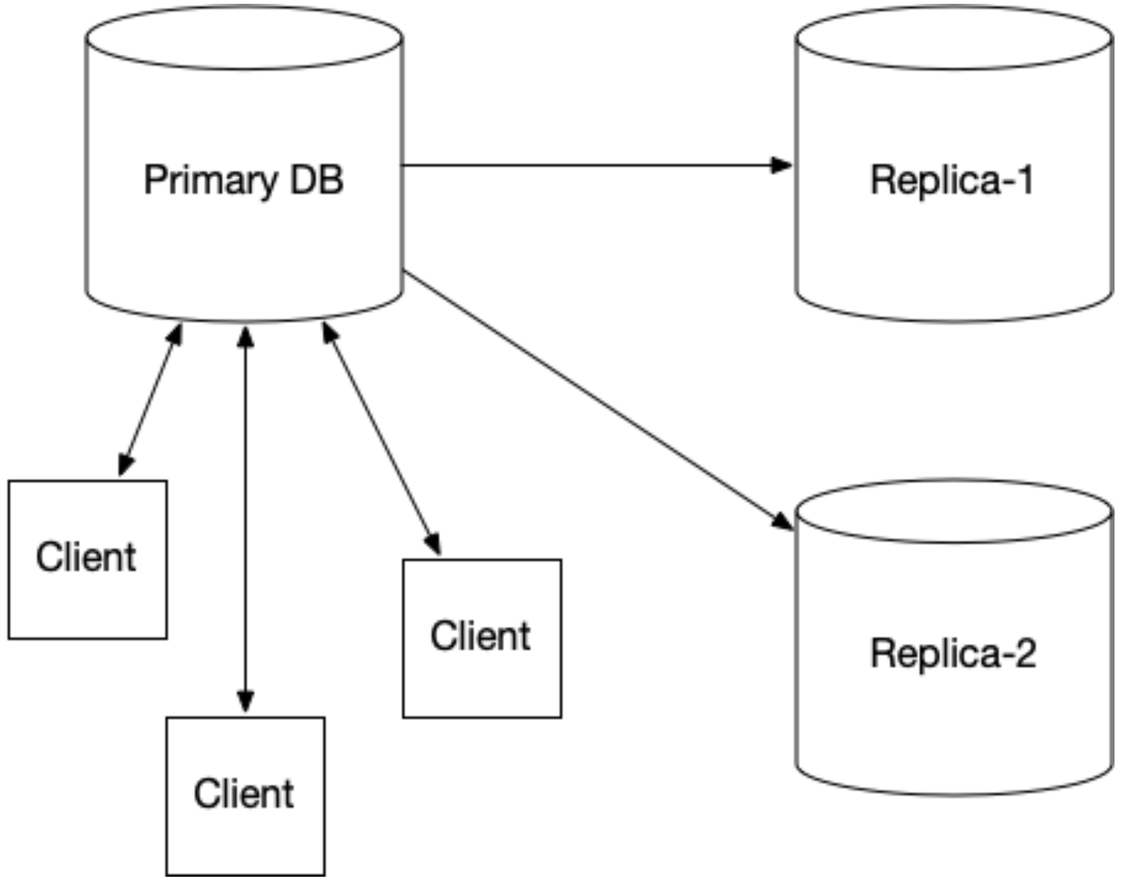
Multi-Master

- No dedicated primary database server
- Updates can happen on any server
- Changes will be propagated to all other servers in the cluster
- Difficult to implement
- Conflict resolution - What happens if the same row is updated by two servers at the same time, which one is correct?



Primary-Secondary (Master-Slave)

- All updates only performed on the primary server
- Updates are pushed to the secondary servers
- Secondary servers can be read from but not written to



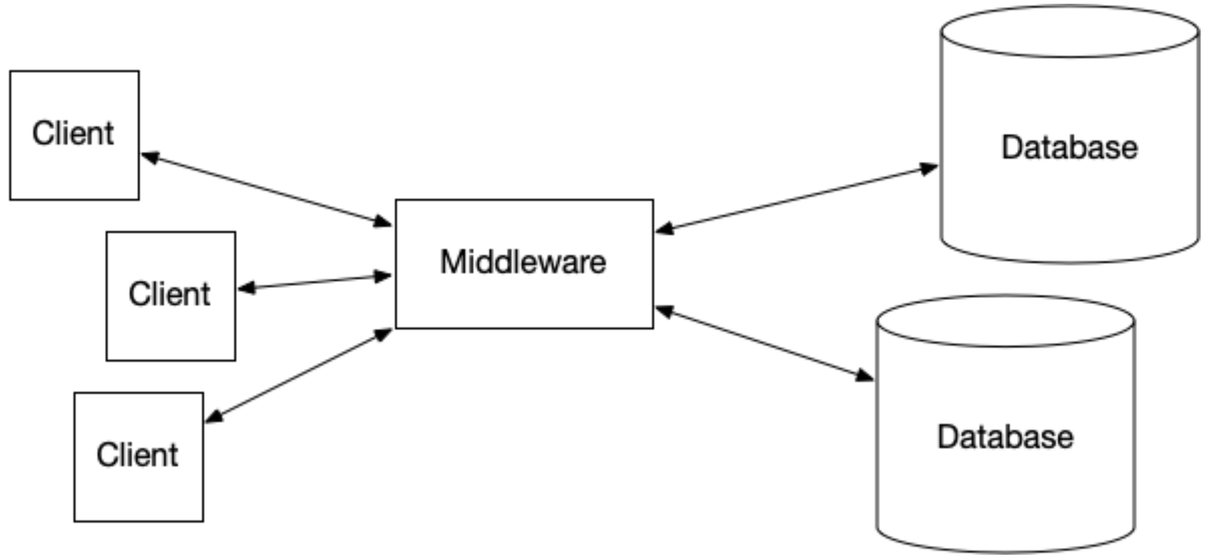
Additional considerations

- Logical Replication
- Write-Ahead Log shipping

- Synchronous Updates
- Asynchronous Updates

Logical Replication

- Individual tables can be replicated
- Statement level or using triggers
- Could be implemented using middleware (Slony,PGPool,Bucardo)
- Need to be careful with functions such as random( ), gen_random_uuid( ) as each database would get a different random value



Write-Ahead Log Shipping

- All or Nothing - Entire database cluster is replicated
- Utilizes the Write-ahead log
- Handles temporary interruption of connection to replicas
- Replicas can be used in read-only mode

```
root@primary:/srv/database/pg_wal# ls -l
total 196612
-rw------- 1 postgres postgres 16777216 Nov  9 17:08 000000010000000000000001
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000002
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000003
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000004
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000005
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000006
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000007
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000008
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 000000010000000000000009
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 00000001000000000000000A
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 00000001000000000000000B
-rw------- 1 postgres postgres 16777216 Nov  9 17:09 00000001000000000000000C
drwx------ 2 postgres postgres     4096 Nov  8 04:36 archive_status
```

PostgreSQL Write-Ahead Log

Synchronous Updates

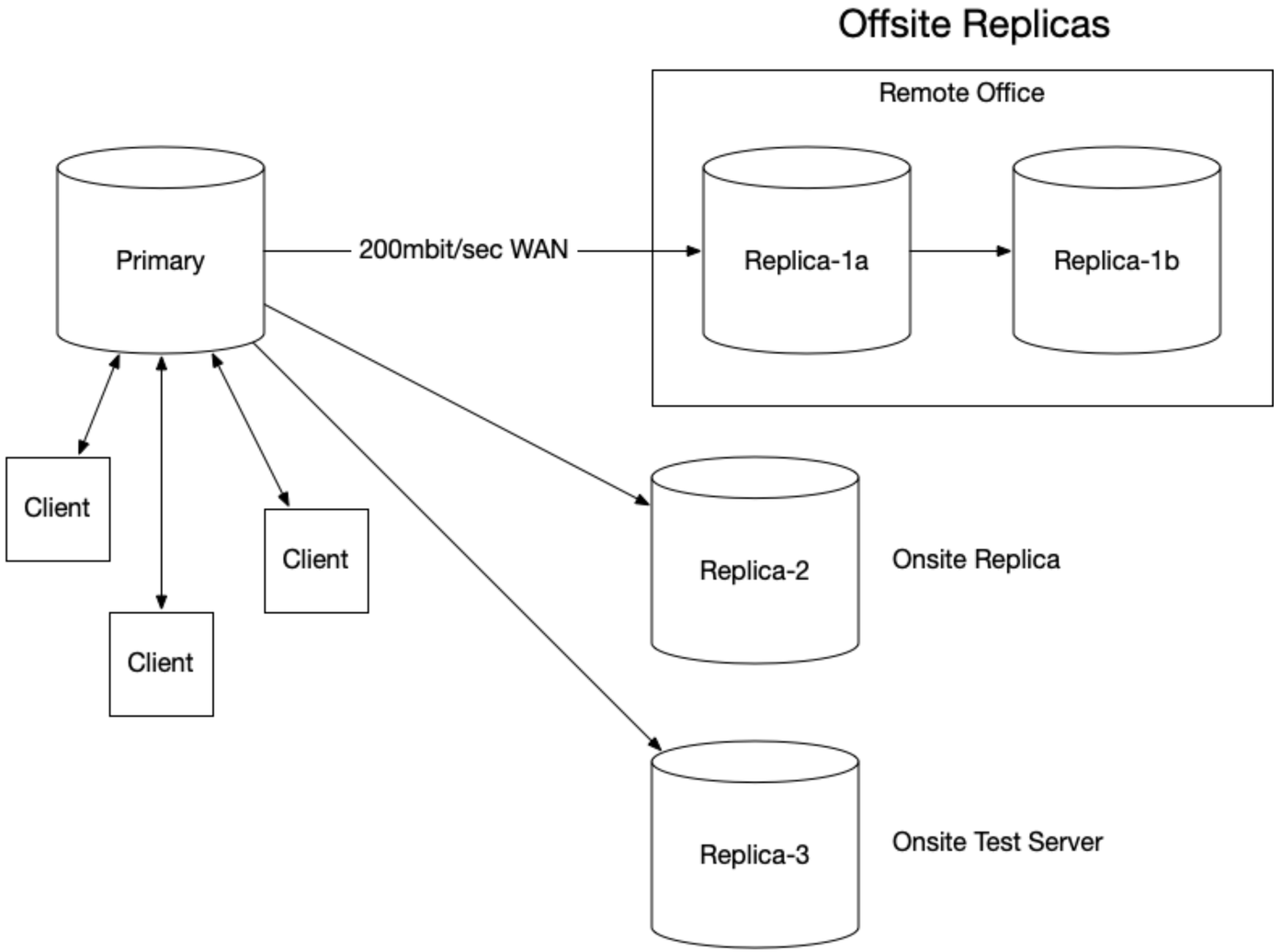
- Every replica acknowledges update before control is returned to the client
- Extremely small chance of losing data
- Well suited to readers, All replicas will have current data
- Poor performance when lots of write activity
- Doesn’t handle replica going offline very well

Asynchronous Updates

- Primary database is updated, Replicas are consistent “eventually”
- Possibility of losing data if the primary crashes before the replicas are updated
- If clients are reading from a replica, they can get stale data if replica isn’t caught up
- If a replica is slow or goes offline it won’t affect the client

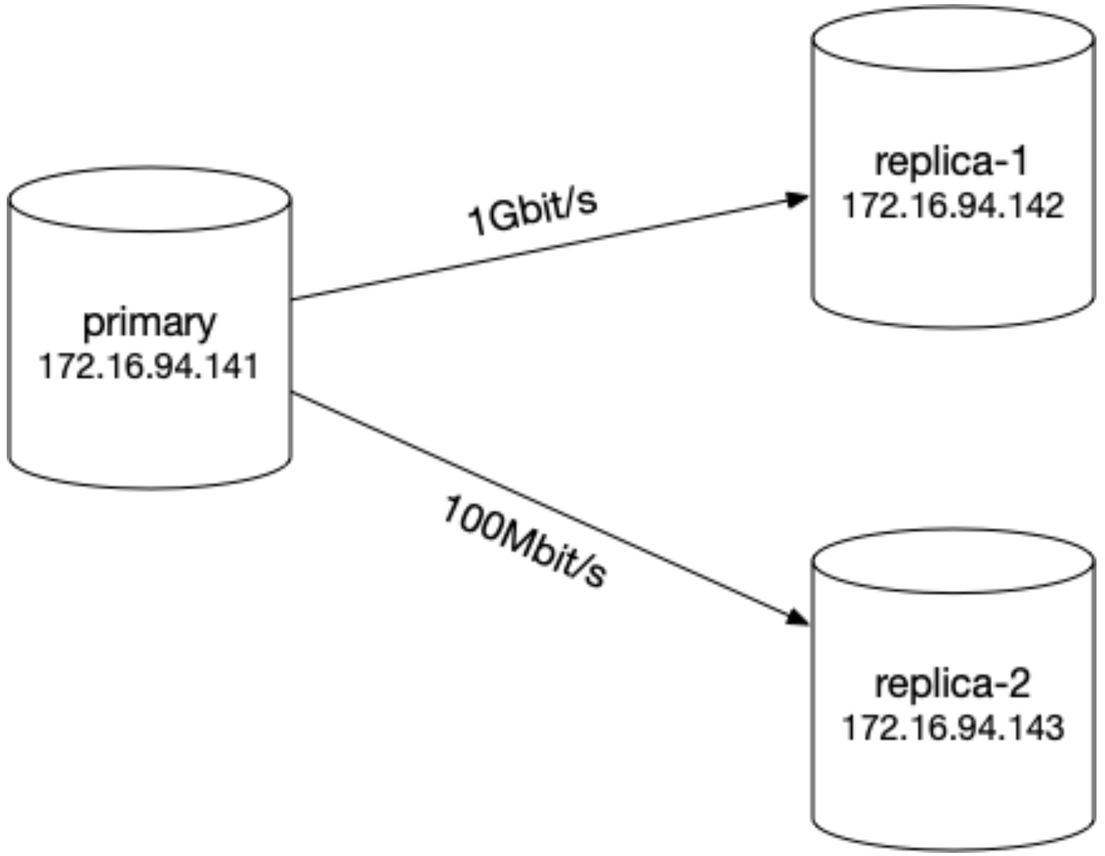
Real World Example

- Write-ahead Log Streaming
- Asynchronous
- Database is over 2TB
- Remote office about 30 miles away
- ~140 clients
- Remote office is a disaster recovery site. Cascading to make it easier to test.



Example Configuration

- Write-Ahead Log shipping
- Asynchronous Updates
- 3 Servers
  - primary
  - replica-1
  - replica-2
- Ubuntu 20.04
- PostgreSQL 12
- Database in `/srv/database`
  instead of `/var/lib/postgresql/12/main`



Primary Server

Fresh install of Ubuntu 20.04 server:

```
$ sudo -i
# timedatectl set-timezone America/Detroit
# apt-get install postgresql-all
# systemctl stop postgresql
# systemctl disable postgresql
# echo 'export PATH=/usr/lib/postgresql/12/bin:$PATH' >>/etc/profile
# mkdir /srv/database
# mkdir /srv/log
# chown postgres:postgres /srv/database  /srv/log
# chmod 0700 /srv/database  /srv/log
# su - postgres
$ pg_ctl -D /srv/database init
```

Primary Server - continued

/srv/database/postgresql.conf

```
listen_addresses  = '*'
logging_collector = on
log_directory     = '/srv/log'
log_filename      = '%Y-%m-%d.log'
```

Start the database cluster and create a user and database

```
$ sudo -i -u postgres pg_ctl -D /srv/database start
$ sudo -i -u postgres createuser jam --superuser
$ createdb mug
```

Secondary Servers

Fresh install of Ubuntu 20.04 server:

```
$ sudo -i

# timedatectl set-timezone America/Detroit

# apt-get install postgresql-all

# systemctl stop postgresql

# systemctl disable postgresql

# echo 'export PATH=/usr/lib/postgresql/12/bin:$PATH' >>/etc/profile
```

Lets Do Some Replicating, Shall we?

Configure primary for replication

Add the following lines to the end of the `/srv/database/pg_hba.conf` file to allow the replication servers to connect

```
host    replication     all             172.16.94.142/32        trust    # replica-1
host    replication     all             172.16.94.143/32        trust    # replica-2
```

Tell postgres to reload the `pg_hba.conf` file

```
$ sudo -i -u postgres pg_ctl -D /srv/database reload
```

Create replication slots

```
$ sudo -i -u postgres psql -c "select pg_create_physical_replication_slot( 'replica_1' )" mug
$ sudo -i -u postgres psql -c "select pg_create_physical_replication_slot( 'replica_2' )" mug
```

Configure replica-1

Create the directory for the database:

```
$ sudo mkdir /srv/database  /srv/log
$ sudo chown postgres:postgres /srv/database /srv/log
$ sudo chmod 0700 /srv/database /srv/log
```

Run **pg_basebackup** to create the replica

```
$ sudo -i -u postgres pg_basebackup -D /srv/database --wal-method=stream
    --slot=replica_1 --write-recovery-conf --progress --host 172.16.94.141
```

Configure replica-1 - continued

Start the database

```
$ sudo -i -u postgres pg_ctl -D /srv/database  start
```

Is It Working?

Look in the log file in /srv/log. You should see something like this:

```
2020-11-10 15:04:54.188 UTC [29514] LOG:  entering standby mode
2020-11-10 15:04:54.190 UTC [29514] LOG:  redo starts at 0/85000028
2020-11-10 15:04:54.190 UTC [29514] LOG:  consistent recovery state reached at 0/85000138
2020-11-10 15:04:54.190 UTC [29512] LOG:  database system is ready to accept read only connections
2020-11-10 15:04:54.196 UTC [29518] LOG:  started streaming WAL from primary at 0/86000000 on timeline 1
```

Query the **pg_replication_slots** view (on the primary)

```
mug=# select * from pg_replication_slots;
 slot_name | plugin | slot_type | datoid | database | temporary | active | active_pid |  xmin  | catalog_xmin | restart_lsn | confirmed_flush_lsn
-----------+--------+-----------+--------+----------+-----------+--------+------------+--------+--------------+-------------+---------------------
 replica_1 | (null) | physical  | (null) | (null)   | f         | t      |      29727 | (null) |       (null) | 0/86000148  | (null)
 replica_2 | (null) | physical  | (null) | (null)   | f         | f      |     (null) | (null) |       (null) | (null)      | (null)
(2 rows)
```

Replication status

The **pg_stat_replication** view shows more status information

```
mug=# \x
Expanded display is on.
mug=# select * from pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 29727
usesysid         | 10
usename          | postgres
application_name | walreceiver
client_addr      | 172.16.94.142
client_hostname  | (null)
client_port      | 54020
backend_start    | 2020-11-10 15:04:54.179158+00
backend_xmin     | (null)
state            | streaming
sent_lsn         | 0/86000148
write_lsn        | 0/86000148
flush_lsn        | 0/86000148
replay_lsn       | 0/86000148
write_lag        | (null)
flush_lag        | (null)
replay_lag       | (null)
sync_priority    | 0
sync_state       | async
reply_time       | 2020-11-10 15:16:05.009385+00

Time: 0.715 ms
mug=#
```

Files of interest in /srv/database

`standby.signal`

The presence of this file indicates the server should startup in hot-standby mode

`postgresql.auto.conf`

This file contains the parameters for connecting to the primary server

Promote A Replica To Primary

There are a couple of ways to promote a replica:

- `pg_ctl -D /srv/database promote`
- `SELECT pg_promote()`
- If you have a line like this: `promote_trigger_file = '/tmp/replica_1'` in `postgresql.conf` you can: `touch /tmp/replica_1`

After promoting, you need to point all of your clients to the new primary

Some Goodies

Remove replication slots

```
SELECT pg_drop_replication_slot( 'replica_1' );
```

Delay replication

Add the following line to the postgresql.conf file

```
recovery_min_apply_delay = 10000 (milliseconds)
```

Monitor replication using Nagios

Add the following line to nrpe.cfg (check_pg_replication script on next slide)

```
command[check_replica_1] = /usr/local/nagios/libexec/check_pg_replication -u nobody -s replica_1
```

Nagios check_pg_replication

```
#!/bin/bash

#
# Setup the environment
#
. /etc/profile

DEBUG=0
DB_NAME=postgres
PG_USER=postgres
PG_PORT=5432
PG_HOST=localhost

function usage () {
  cat <<EOF
Usage : $0 [options]
        -u pg_user
        -p pg_port
        -h pg_host
        -n dbname
        -s slot
EOF
  exit
}

while getopts "u:p:h:n:s:" OPTION
do
  case $OPTION in
    u)    PG_USER=$OPTARG
          ;;
    p)    PG_PORT=$OPTARG
          ;;
    h)    PG_HOST=$OPTARG
          ;;
    n)    DB_NAME=$OPTARG
          ;;
    s)    SLOT=$OPTARG
          ;;
    \?)   echo -n "Unknown option"
          usage
  esac
done
```

```
[ -z "$PG_USER" ] && echo "UNKNOWN : Postgres user not specified"             && exit 3
[ -z "$PG_PORT" ] && echo "UNKNOWN : Postgres port not specified"             && exit 3
[ -z "$PG_HOST" ] && echo "UNKNOWN : Postgres host not specified"             && exit 3
[ -z "$DB_NAME" ] && echo "UNKNOWN : Postgres database name not specified"    && exit 3
[ -z "$SLOT"    ] && echo "UNKNOWN : Postgres replication slot not specified" && exit 3

RESULT=`psql --no-align               \
             --tuples-only            \
             --dbname       $DB_NAME  \
             --port         $PG_PORT  \
             --host         $PG_HOST  \
             --username     $PG_USER  \
             --command      "SELECT active FROM pg_replication_slots WHERE slot_name = '$SLOT'"`

if [ "$RESULT" == "t" ]; then
  echo "OK : Replication slot '$SLOT' active and streaming"
  exit 0
else
  echo "WARNING : Replication slot '$SLOT' is NOT active"
  exit 1
fi
```

In Summary

- Grant permission in **pg_hba.conf**
- Create a **replication_slot** for each replica
- use **pg_basebackup** to initialize the replica
- Monitor **pg_replication_slots** for inactive replicas

Questions?

Jim McQuillan jam@McQuil.com These slides will be available on mug.org