



Postgres Plus Database Administration

Postgres Plus Advanced Server (PPAS) 9.5



Day 3 – High Availability

Postgres Plus Advanced Server (PPAS) 9.5



Day3-1

Backup, Recovery and PITR

Module Objectives

- Backup Types
- SQL Dump Backups
- SQL Dump Options
- Handling Large Databases
- Restoring SQL Dumps
- Cluster Dump
- Offline Backups
- Recovering a Database using Offline Backups
- Online Directory Backups
 - Continuous Archiving
 - Backup with Low level API
 - How to use **pg_basebackup** for Online Backups
- Point-In Time Recovery Concepts
- Recovery Example

Backup

- As with any database, PPAS database should be backed up regularly.
- There are three fundamentally different approaches to backing up PPAS data:
 - SQL dump
 - File system level backup
 - Continuous Archiving

Logical Backup & Physical Backup

- Logical Backup
 - Easy of use
 - Partial backup and recovery
 - Can validate all data files
- Physical Backup
 - Performance
 - Flexibility
 - PITR
 - Consistent

Backup Strategies

- Almost always need to have multiple type of backup.
- Low MTTR = High cost
- Trade-off between your requirements and resources.
- Validation is the most important thing.



Day3-2

Logical Backup & Recovery

Backup - SQL Dump

- Generate a text file with SQL commands
- PPAS provides the utility program **pg_dump** for this purpose.
- **pg_dump does not block readers or writers.**
- **pg_dump** does not operate with special permissions.
- Dumps created by **pg_dump** are internally consistent, that is, the dump represents a snapshot of the database as of the time **pg_dump** begins running.
- Syntax:

```
pg_dump [options] [dbname]
```

pg_dump Options

- **pg_dump** Options
 - a – Data only. Do not dump the data definitions (schema)
 - s – Data definitions (schema) only. Do not dump the data
 - n <schema> - Dump from the specified schema only
 - t <table> - Dump specified table only
 - f <path/file name.backup> - Send dump to specified file
 - Fp – Dump in plain-text SQL script (default)
 - Ft – Dump in tar format
 - Fc – Dump in compressed, custom format
 - Fd – Dump in directory format
 - j njobs -- dump in parallel by dumping njobs tables simultaneously. Only supported with -Fd
 - v – Verbose option
 - o use oids

SQL Dump - Large Databases

- If operating systems have maximum file size limits, it can cause problems when creating large **pg_dump** output files.
- Standard Unix tools can be used to work around this potential problem.
 - You can use your favorite compression program, for example gzip:

```
pg_dump dbname | gzip > filename.gz
```
 - Also the `split` command allows you to split the output into smaller files:

```
pg_dump dbname | split -b 1m - filename
```

Restore - SQL Dump

- The text files created by **pg_dump** are intended to be read in by the psql program. The general command form to restore a dump is

```
psql dbname < infile
```

where infile is what you used as outfile for the **pg_dump** command. The database dbname will not be created by this command, so you must create it yourself

- **pg_restore** is used to restore a database backed up with **pg_dump** that was saved in an archive format – i.e., a non-text format
- Files are portable across architectures
 - Syntax:

```
pg_restore [options...] [filename.backup]
```

pg_restore Options

pg_restore Options

-d <database name> - Connect to the specified database.

Also restores to this database if -C option is omitted

-C – Create the database named in the dump file and restore directly into it

-a – Restore the data only, not the data definitions (schema)

-s – Restore the data definitions (schema) only, not the data

-n <schema> - Restore only objects from specified schema

-t <table> - Restore only specified table

-v – Verbose option

Entire Cluster - SQL Dump

- **pg_dumpall** is used to dump an entire database cluster in plain-text SQL format
- Dumps global objects - users, groups, and associated permissions
- Use psql to restore
- Syntax:

```
pg_dumpall [options...] > filename.backup
```

pg_dumpall Options

pg_dumpall Options

- a Data only. Do not dump schema.
- s Data definitions (schema) only.
- g Dump global objects only - not databases.
- r Dump only roles.
- c Clean (drop) databases before recreating.
- O Skip restoration of object ownership.
- x Do not dump privileges (grant/revoke)
- disable-triggers
disable triggers during data-only restore
- v Verbose option.

Lab



Day3-3

Physical Backup & Recovery

Backup - File System Level Backup (Cold Backup)

- An alternative backup strategy is to directly copy the files that PPAS uses to store the data in the database.
- You can use whatever method you prefer for doing usual file system backups, for example:

```
tar -cf backup.tar /usr/local/pgsql/data
```
- The database server must be shut down in order to get a usable backup.
- File system backups only work for complete backup and restoration of an entire database cluster.
- File system snapshots work for live servers.

Backup - Continuous Archiving

- PPAS maintains WAL files for all transactions in the **pg_xlog** directory
- PPAS automatically maintains the wal logs which are full and switched
- Continuous archiving can be setup to keep a copy of switched WAL Logs which can be later used for recovery
- It also enables online file system backup of a database cluster
- Requirements:
 - wal_level must be set to archive
 - archive_mode must be set to on
 - archive_command must be set in **postgresql.conf** which archives WAL logs and supports PITR

pg_receivexlog

- **pg_receivexlog** streams transaction logs from a running PPAS cluster.
- This tool uses streaming replication protocol to stream transaction logs to a local directory.
- These files can be then used for PITR.
- Transaction logs are streamed in real time thus the **pg_receivexlog** can be used instead of archive command.
- Example:

```
pg_receivexlog -h localhost -D /usr/local/pgsql/archive
```

Continuous Archiving

- **Step 1** - Edit the **postgresql.conf** file and set the archive parameters:

```
wal_level=archive  
archive_mode=on
```

Unix:

```
archive_command= 'cp -i %p /mnt/server/archivedir/%f </dev/null'
```

Windows:

```
archive_command= 'copy "%p" c:\\mnt\\server\\archivedir\\"%f"'
```

%p is the absolute path of WAL otherwise you can define the path

%f is a unique file name which will be created on above path

Base Backup Using Low Level API

- **Step 2** - Make a base backup

Connect using psql and issue the command:

```
SELECT pg_start_backup('any useful label');
```

Use a standard file system backup utility to back up the /data subdirectory

Connect using psql and issue the command:

```
SELECT pg_stop_backup();
```

Continuously archive the WAL segment files

Base Backup Using pg_basebackup Tool

- **pg_basebackup** can take a base backup of a PPAS cluster.
- This backup can be used for PITR or Streaming Replication.
- **pg_basebackup** makes a binary copy of the database cluster files.
- The system is automatically put in and out of backup mode.

Options for pg_basebackup

- Command line options for **pg_basebackup** :
 - D <directory name> - Location of backup
 - F <p or t> - Backup files format. Plain(p) or tar(t)
 - X <f or s> include required transaction log files
 - z - enable gzip compression for files
 - Z - level - Compression level
 - P - Progress Reporting
 - h - host - host on which cluster is running
 - p - port - cluster port
- To create a base backup of the server at localhost and store it in the local directory /usr/local/pgsql/backup:

```
$ pg_basebackup -h localhost -D /usr/backup
```


pg_basebackup - Online Backup

- Steps require to take Base Backup:
 - Modify **pg_hba.conf**
host replication enterprisedb [IPv4 address of client]/32 trust
 - Modify **postgresql.conf**
archive_command = 'cp -i %p /Users/postgres/archive/%f'
archive_mode = on # Require Restart
max_wal_senders = 3 # Maximum 'wal_senders'
wal_keep_segments = 50 # How many WAL segments (=files should be kept on the server
wal_level=archive
- Backup Command:
pg_basebackup -D /usr/backup/testbackup -v -Fp -l
Testbackup -h 127.0.0.1 -U postgres

Point-in-Time Recovery

- Point-in-time recovery (PITR) is the ability to restore a database cluster up to the present or to a specified point of time in the past.
- Uses a full database cluster backup and the write-ahead logs found in the `/pg_xlog` subdirectory.
- Must be configured before it is needed (write-ahead log archiving must be enabled).

Performing Point-in-Time Recovery

- Stop the server, if it's running.
- If you have enough space keep a copy of the data directory and transaction logs.
- Remove all directories and files from the cluster data directory.
- Restore the database files from your file system backup.
- Verify the ownership of restored backup directories (must not be root).
- Remove any files present in `pg_xlog/`.
- If you have any unarchived WAL segment files recovered from crashed cluster, copy them into `pg_xlog/`.
- Create a recovery command file **recovery.conf** in the cluster data directory.
- Start the server.
- Upon completion of the recovery process, the server will rename **recovery.conf** to **recovery.done**.

Point-in-Time Recovery Options

- Settings in the **recovery.conf** file:

`restore_command(string)`

Unix:

```
restore_command = 'cp /mnt/server/archivedir/%f "%p"'
```

Windows:

```
restore_command = 'copy c:\\mnt\\server\\archivedir\\"%f" "%p"\'
```

```
recovery_target_name (string)
```

```
recovery_target_time(timestamp)
```

```
recovery_target_xid(string)
```

```
recovery_target_inclusive(boolean)
```

```
recovery_target_timeline (string)
```

```
pause_at_recovery_target (boolean) < 9.5
```

```
recovery_target_action (pause | promote | shutdown) >= 9.5
```

Module Summary

- Backup Types
- SQL Dump Backups
- SQL Dump Options
- Handling Large Databases
- Restoring SQL Dumps
- Cluster Dump
- Offline Backups
- Recovering a Database using Offline Backups
- Online Directory Backups
 - Continuous Archiving
 - Backup with Low level API
 - How to use **pg_basebackup** for Online Backups
- Point-In Time Recovery Concepts
- Recovery Example
- Backup and Recovery Tool (BART)



Day 3-4.1

High Availability & Replication

Objective

- In this module you will learn:
 - Data Replication
 - Data Replication in PPAS
 - Sync or Async
 - Log-Shipping Standby Servers
 - Log-Shipping Architecture
 - Streaming Replication
 - Hot Streaming Architecture
 - Cascading Replication

Data Replication

- Replication is process of copying data and changes to a secondary location for data safety and availability
- Data loss can occur due to several reasons
- Replication is aimed towards availability of the data when a primary source goes offline
- Data can be recovered from backup but downtimes are costly
- Replication aims towards lowering downtime
- Failovers can be configured to such a level where application may not notice the primary source is offline



Data Replication in PPAS

- Data Replication in PPAS can be achieved by implementing:
 - Log-Shipping Standby Servers
 - Streaming Replication
 - xDB Replication
- Failovers can be automated using:
 - EDB Failover Manager for streaming replicated servers
 - pgpool-II



Sync or Async

- Log shipping is only asynchronous
- PPAS streaming replication is asynchronous by default but can be configured as synchronous
- Asynchronous
 - Disconnected architecture
 - Transaction is committed on primary and flushed to WAL segment
 - Later transaction is transmitted to standby server(s) using stream
 - In case of log shipping, replication happens when the WAL segment is archived
 - Some data loss is possible

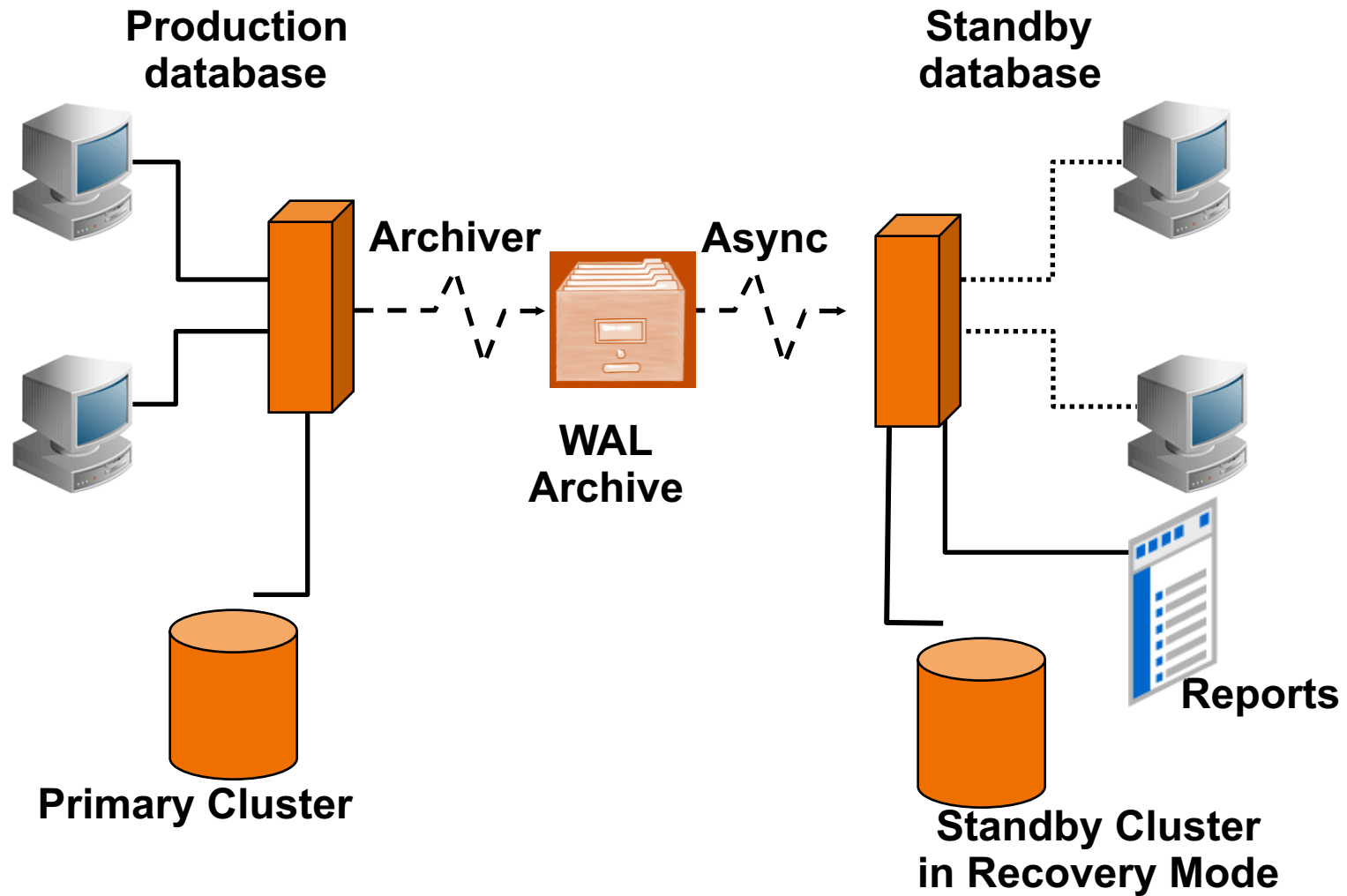
Sync or Async

- Synchronous
 - Only streaming replication supports synchronous replication method
 - Its 2-safe replication
 - Two-phase commit actions
 - Transaction is committed on primary and flushed to WAL segment of primary and standby server
 - User gets a commit message after confirmation from both primary and standby
 - This will introduce a delay in committing transactions

Log-Shipping Standby Servers

- WAL archiving method for high availability (HA) cluster
- Primary server is in WAL archive mode
- Standby server is in continuous recovery mode
- In continuous recovery mode standby server is reading and playing WAL logs from WAL archive area of primary server
- Low performance impact on primary
- Low administration overhead
- Supports warm and hot standby

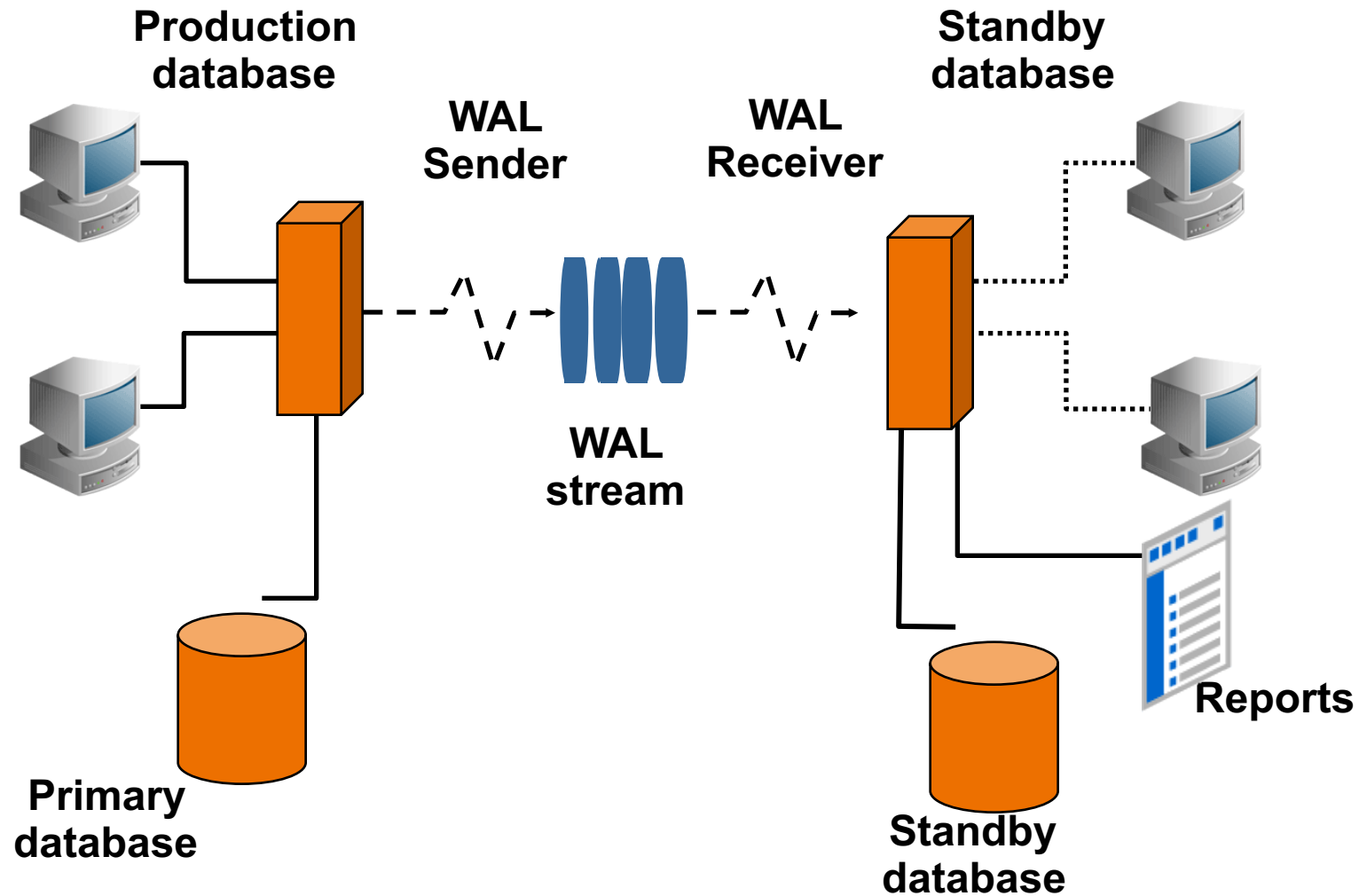
Log-Shipping Architecture



Streaming Replication

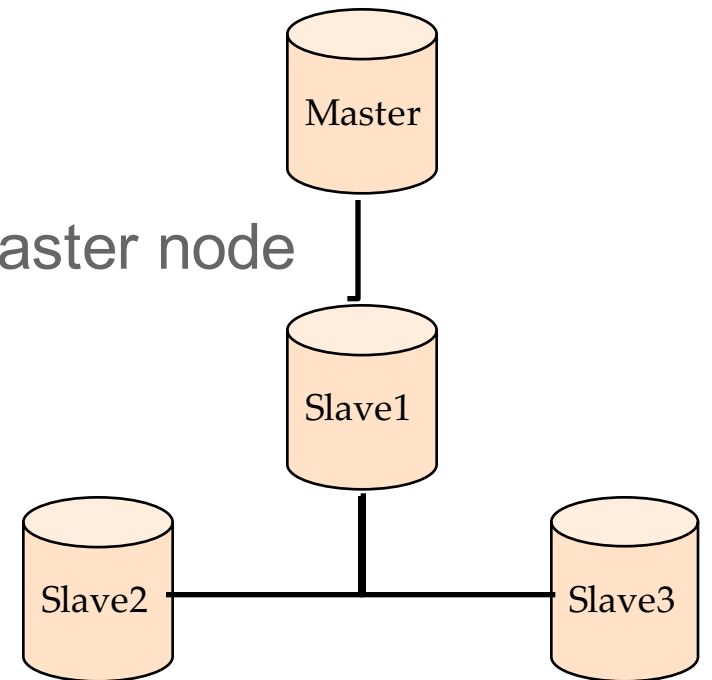
- Streaming Replication (Hot Standby) is a major feature of PPAS
- Standby connects to the primary node using `REPLICATION` protocol
- WAL segments are streamed to standby server
- No log shipping delays, waits for transaction to commit and stream it across to standby immediately
- Synchronous/Asynchronous options available
- Supports cascading replication, so the standby can also send replication changes sharing the overhead of a master

Hot Streaming Architecture



Cascading Replication

- Streaming replication support single master node
- The ability of a standby server to send stream or archive changes to other standby servers is cascading replication
- Such standby servers are called cascading standby
- Helps minimize inter-site bandwidth overheads on master node
- Asynchronous only



Module Summary

- In this module we learned:
 - Data Replication
 - Data Replication in PPAS
 - Sync or Async
 - Log-Shipping Standby Servers
 - Log-Shipping Architecture
 - Streaming Replication
 - Hot Streaming Architecture
 - Cascading Replication



Day 3-4.2

Setup Log-Shipping Standby

Objectives

In this module we will cover:

- Setup Replication Using Archive
 - Setting Up Replication Using Archive
 - Example: Replication Using Archive

Setting Up Replication Using Archive

1. WAL Archive must be configured on primary server
2. Archive location must be accessible to standby server
3. Take a full base backup of primary database cluster
4. Copy the backup to the standby server
5. If using database tablespaces, directory structure must be same on master and standby
6. Create a recovery.conf file inside data directory of standby database cluster
7. Turn on `standby_mode` and set `restore_command` to copy files from the WAL archive location

Example: Replication Using Archive

- Configure WAL archiving:

```
$ sudo mkdir /pg_arch
```

```
$ sudo chown postgres:postgres /pg_arch
```

```
$ vi /opt/PostgreSQL/9.4/data/postgresql.conf
```

```
wal_level = hot_standby
```

```
archive_mode = on
```

```
archive_command = 'cp %p /pg_arch/%f'
```

```
$ sudo service postgresql-9.4 restart
```

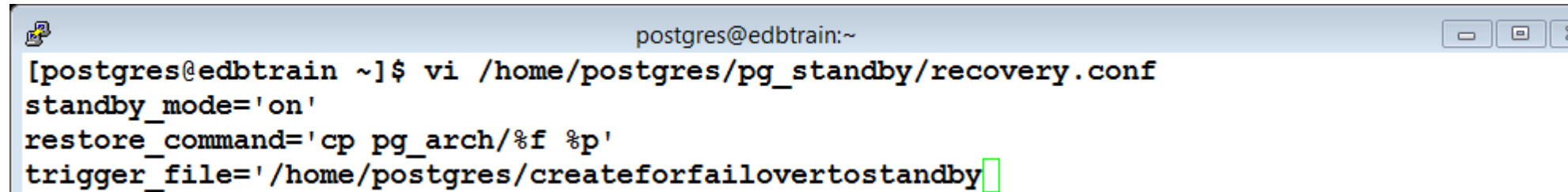
Example: Replication Using Archive

- Take a base backup of primary and copy it to standby

```
postgres@edbtrain:~  
[training@edbtrain ~]$ su - postgres  
Password:  
[postgres@edbtrain ~]$ psql -c "SELECT pg_start_backup('For_Standby');" postgres postgres  
Password for user postgres:  
pg_start_backup  
-----  
0/2000028  
(1 row)  
  
[postgres@edbtrain ~]$ cp -rp /opt/PostgreSQL/9.4/data/ /home/postgres/pg_standby  
[postgres@edbtrain ~]$ psql -c "SELECT pg_stop_backup();" postgres postgres  
Password for user postgres:  
NOTICE: pg_stop_backup complete, all required WAL segments have been archived  
pg_stop_backup  
-----  
0/2000160  
(1 row)  
  
[postgres@edbtrain ~]$ rm -rf /home/postgres/pg_standby/postmaster.pid  
[postgres@edbtrain ~]$
```

Example: Replication Using Archive

- Create recovery.conf

A terminal window titled 'postgres@edbtrain:~' showing the command 'vi /home/postgres/pg_standby/recovery.conf' being executed. The file is opened in vi editor, and the following content is entered:

```
standby_mode='on'  
restore_command='cp pg_arch/%f %p'  
trigger_file='/home/postgres/createforfailovertostandby'
```

```
[postgres@edbtrain ~]$ vi /home/postgres/pg_standby/recovery.conf  
standby_mode='on'  
restore_command='cp pg_arch/%f %p'  
trigger_file='/home/postgres/createforfailovertostandby'
```

- Open postgresql.conf and turn on hot standby mode:
 - \$ vi /home/postgres/pg_standby/postgresql.conf
 - hot_standby = on
 - If testing on single machine – comment archive parameters and change port:
 - port = 5435
- Save and start the standby:
 - \$ pg_ctl -D /home/postgres/pg_standby -l startlog start

Example: Replication Using Archive

Connect to primary and add data

```
enterprisedb@localhost:~  
psql.bin (9.4.0.1)  
Type "help" for help.  
  
edb=# create table x(xid number);  
CREATE TABLE  
edb=# insert into x values  
edb-# (generate_series(1,10000000));  
INSERT 0 10000000  
edb=# select pg_switch_xlog();  
pg_switch_xlog  
-----  
0/2BDAD934  
(1 row)  
  
edb=#
```

Verify data is replicated on standby

```
psql.bin (9.4.0.1)  
Type "help" for help.  
  
edb=# select count(*) from x;  
count  
-----  
10000000  
(1 row)  
  
edb=#
```


Summary

In this module we covered:

- Setup Replication Using Archive
 - Setting Up Replication Using Archive
 - Example: Replication Using Archive



Day 3-4.3

Setup Streaming Replication

Objectives

In this module we will cover:

- Setup Streaming Replication
 - Prepare the Primary Server
 - Synchronous Streaming Replication Setup
 - Configure Authentication
 - Take a Full Backup of Primary Server
 - Setting up the Standby Server
 - Adding Cascading Replicated Standby Server
 - Monitoring Hot Standby
 - Recovery Control Functions

Prepare the Primary Server

- Change WAL Content parameter:
 - `wal_level=hot_standby`
- Sets only the minimum number of segments retained in `pg_xlog`;
 - `wal_keep_segments = 50`
- Two options to allow streaming connection:
 - `max_wal_senders`
 - Set maximum number of concurrent connections from standby servers or stream clients. This enable the ability to stream WAL to the standby
 - `max_replication_slots`
 - maximum number of replication slots that the server can support
- `wal_sender_timeout`
 - Specify time in `ms` to terminate inactive replication connections. Default `60 sec`

Synchronous Streaming Replication Setup

- Default level of Streaming Replication is Asynchronous
- Synchronous level can also be configured to provide 2-safe replication
- Additional parameters need to be configured:
 - `synchronous_commit=on`
 - `synchronous_standby_names`
 - Specifies a comma-separated list of standby names that can support synchronous replication
- Transactions can be configured not to wait for replication by setting the `synchronous_commit` parameter to `local` or `off`
- During synchronous setup `pg_start_backup()` and `pg_stop_backup()` are run in a session with `synchronous_commit = off`

Configure Authentication

- Authentication setting on the primary server must allow replication connections from the standby server(s);
- Provide a suitable entry or entries in `pg_hba.conf` with the database field set to `replication`.
- Open `pg_hba.conf` of primary server:

```
host          replication          all          127.0.0.1/32          trust
```

Note: You will need to reload primary server

Take a full backup of Primary Server

- Connect to the database as a superuser and issue the command:
 - `SELECT pg_start_backup('label');`
- Perform the backup, using any convenient file-system-backup tool such as tar
 - `cp -rp /opt/PostgreSQL/9.4/data /backup/data1`
- Again connect to the database as a superuser, and issue the command:
 - `SELECT pg_stop_backup();`
- Copy the backup directory on to the standby server
- You might change port to 5435 for standby server in case you configure standby on same machine

Setting up Standby Server

- Remove postmaster.pid file
 - `rm /backup/data/postmaster.pid`
- Standby server config parameters:
 - `hot_standby`
 - `max_standby_archive_delay`
 - `max_standby_streaming_delay`
 - `wal_receiver_status_interval`
 - `hot_standby_feedback`
 - `wal_receiver_timeout`

Note: `hot_standby` must be set to `on` for read only transaction support on standby

Setting up Standby Server

- Create `recovery.conf` file inside data directory of standby server
- Setup `recovery.conf` parameters
 - `standby_mode`
 - `primary_conninfo`
 - `primary_slot_name`
 - `recovery_min_apply_delay`
 - `trigger_file`
- Last step: start the standby

Adding Cascading Replicated Standby Server

- Backup the Primary Server using `pg_basebackup`:

```
$ pg_basebackup -h localhost -U replication_user -p 5432 -D /backup/data2
```
- A new cluster will be created with data directory “data2”
- Change the `port` if testing on same machine
- Create `recovery.conf` file in new standby
 - `standby_mode = on`
 - `primary_conninfo = 'host=localhost port=5435 user=replication_user password=secret'`
Note: `primary_conninfo` will connect to standby instead of master
- Change `pg_hba.conf` file of standby to allow connection from `data2` cluster
- Start the cluster

Monitoring Hot Standby

- `pg_stat_replication`
 - Show slaves connected to the master and other useful information
- Streaming Replication can also be monitored by:
 - Size of not yet applied WAL records
 - Lag time between WAL apply
- Recovery information functions:
 - `pg_is_in_recovery()`
 - `pg_current_xlog_location`
 - `pg_last_xlog_receive_location`
 - `pg_last_xact_replay_timestamp()`

Recovery Control Functions

Name	Return Type	Description
<code>pg_is_xlog_replay_paused()</code>	bool	True if recovery is paused.
<code>pg_xlog_replay_pause()</code>	void	Pauses recovery immediately.
<code>pg_xlog_replay_resume()</code>	void	Restarts recovery if it was paused.

Streaming Replication Example

- In this example we will setup async streaming replication on local machine where default cluster is up and running.
- Verify PostgreSQL Cluster is running on 5432 port and data directory is located at /opt/PostgreSQL/9.4/data before proceeding

Streaming Replication Example

- Steps on primary
 1. Login as postgres OS user: `su - postgres`
 2. Open postgresql.conf file using vi editor
 3. Change following parameters:
 - `wal_level = hot_standby`
 - `max_wal_senders = 2`
 - `wal_keep_segments = 32`
 4. Save and close postgresql.conf file
 5. Open pg_hba.conf file using vi editor and add following entry:
 - `host replication all 127.0.0.1/32 trust`
 6. Restart primary server
 - `sudo service postgresql-9.4 restart`

Streaming Replication Example

- Make a base backup by copying the primary server's data directory to the standby server.
 - `pg_basebackup -h localhost -U postgres -D /home/postgres/data`
- Next steps can be performed on separate standby box if available. In that case don't forget to change `pg_hba.conf` entry added in previous slides and copy the backup to standby server

Streaming Replication Example

- Steps on hot standby
 1. Login as postgres OS user: `su - postgres`
 2. Open postgresql.conf file using vi editor
 - `Port=5433` --This is not required if standby has to run on different server
 - `hot_standby=on`
 - Comment `wal_level`, `archive_mode` and `archive_command`
 3. Create recovery.conf file inside data directory of standby server
 - `vi /home/postgres/data/recovery.conf`
 - `standby_mode = 'on'`
 - `primary_conninfo = 'host=localhost port=5432'`
 - `trigger_file = '/home/postgres/trigger_hot_standby_failover'`
 4. Start the server using pg_ctl command
 - `pg_ctl -D /home/postgres/data -l /home/postgres/data/logfile start`
 5. Verify the real time changes been transferred though streaming replication by running transaction on primary

Example: Monitoring Streaming Replication

- Execute:
 - `select * from pg_stat_replication;`
- Find lag(bytes):
 - `Select pg_xlog_location_diff(sent_location, replay_location)
from pg_stat_replication;`
- Find lag(second):
 - `SELECT CASE WHEN pg_last_xlog_receive_location() =
pg_last_xlog_replay_location()
THEN 0 ELSE
EXTRACT (EPOCH FROM now() - pg_last_xact_replay_timestamp())
END AS stream_delay;`

Summary

In this module we covered:

- Setup Streaming Replication
 - Prepare the Primary Server
 - Synchronous Streaming Replication Setup
 - Configure Authentication
 - Take a Full Backup of Primary Server
 - Setting up the Standby Server
 - Adding Cascading Replicated Standby Server
 - Monitoring Hot Standby
 - Recovery Control Functions



Day 3-5

Connection Pooling

Objectives

In this module we will cover:

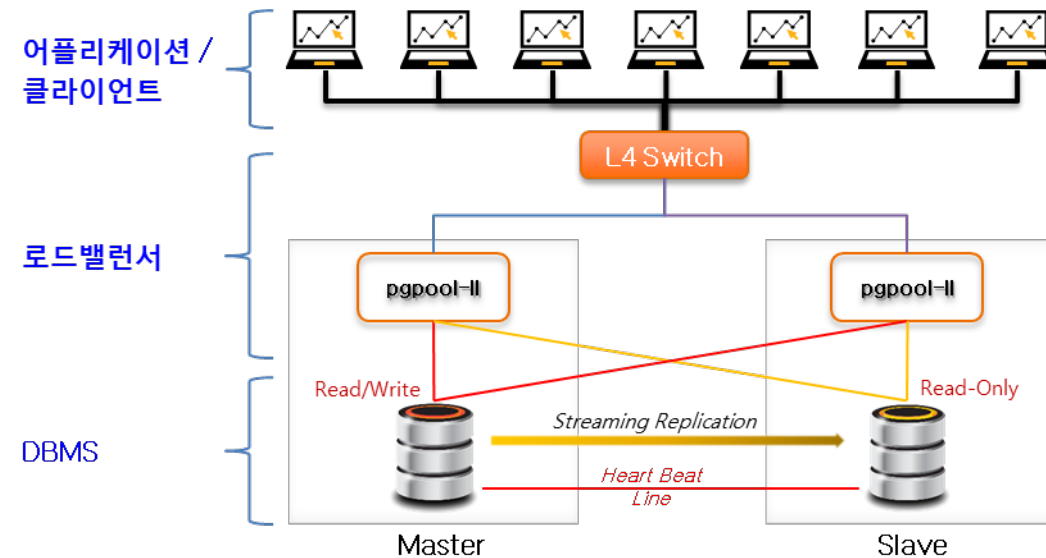
- Connection Pooling Overview
- pgpool-II - Features
 - Connection pooling
 - Replication
 - Load balance
 - Limiting exceeding connections
 - Automatic failover
- pgpool II - Installation and Configuration
- Configuring **pcp.conf**, **pgpool.conf**
- Setting up pool_hba.conf for Client Authentication (HBA)
- Starting/Stopping pgpool-II
- **pgbouncer** –
 - Features, setup, and use
 - “Bouncer is always right”
 - Types of connections
 - How connections are established
 - Managing pgbouncer

Connection Pooling

- Each database client acquires a database connection from the database
- The Postmaster is responsible for listening and providing database connections
- Connection processes are destroyed once the clients log off
- Connection pooling:
 - Provides a mechanism to reuse the existing connection processes once the clients log off
 - Lowers connection overhead on the postmaster
 - Improves application database access performance
 - Spreads connection cost over repeated clients

pgpool-II

- Open source
- Middleware between PPAS servers and a PPAS database client
- pgpool-II talks to PPAS backend and frontend protocol, and relays a connection between them
- Features:
 - Connection Pooling
 - Replication
 - Load Balancing
 - Automatic Failover
 - Parallel Query



Connection Pooling

- pgpool-II saves connections to the PPAS servers
- It reuses them whenever a new connection with the same properties (i.e. username, database, protocol version) comes in
- It reduces connection overhead, and improves system's overall throughput
- Provides cluster-based connection pooling

Replication

- pgpool-II can manage multiple PPAS servers.
- Using the replication function enables creating a real-time backup on 2 or more physical disks, so that the service can continue without stopping servers in case of a disk failure.

Load Balance

- If a database is replicated, executing a SELECT query on any server will return the same result
- pgpool-II automatically detects SELECT and can load balance reads among multiple servers, improving the system's overall throughput
- Load balance works best in a situation where there are a lot of users executing many queries at the same time

Limiting Exceeding Connections

- There is a limit on the maximum number of concurrent connections with PPAS, and connections are rejected after this limit has been reached
- Setting the maximum number of connections, however, increases resource consumption and affects system performance
- pgpool-II also has a limit on the maximum number of connections, but extra connections will be queued instead of returning an error immediately

Automatic Failover

- pgpool-II automatically detects a failed master and can take action to promote a slave to be the new master
- After failover it automatically redirects all write load to the new master and forgets the failed master

pgpool-II Installation

- pgpool-II works on Linux, Solaris, FreeBSD, and most of the UNIX-like architectures
- pgpool-II can be downloaded from <http://pgpool.net/mediawiki/index.php/Downloads>
- Installing pgpool-II requires gcc 2.9 or higher, GNU make and the libpq library
- After extracting the source tar ball, execute the configure script
 - `./configure`
 - `make`
 - `make install` will install pgpool-II.

Configuring pgpool-II

- Configuration files for pgpool-II are `/usr/local/etc/pgpool.conf` and `/usr/local/etc/pcp.conf` by default
- There are several operation modes in pgpool-II
- Each mode has associated functions which can be enabled or disabled, and specific configuration parameters to control their behaviors

Configuring pcp.conf

- pgpool-II provides a control interface for its administration
- pcp.conf is the user/password file for authentication with this interface
- After installing pgpool-II, \$prefix/etc/pcp.conf.sample is created

```
$ cp $prefix/etc/pcp.conf.sample $prefix/etc/pcp.conf
  username:[password encrypted in md5]
```
- Password can be produced with the \$prefix/bin/pg_md5 command:

```
$ pg_md5 -p password: <your password>
```

Configuring pgpool.conf

- Each operation mode has specific configuration parameters in **pgpool.conf**
- After installing pgpool-II, \$prefix/etc/pgpool.conf.sample is created

```
cp $prefix/etc/pgpool.conf.sample $prefix/etc/pgpool.conf
```
- There are additional sample `pgpool.conf` for each mode

`pgpool.conf.sample-replication`

`pgpool.conf.sample-master-slave`

`pgpool.conf.sample-stream`

pgpool.conf parameters

Connections

- listen_addresses, port, pcp_port

Pools

- num_init_children, child_life_time, child_max_connections, client_idle_limit, enable_pool_hba, pool_passwd

Logs

- log_destination, log_connections, log_statement

File Location

- pid_file_name, logdir

Failovers

- failover_command, failback_command, follow_master_command

Load Balancing

- replication_mode, master_slave_sub, load_balance_mode, replicate_select

Backends

- backend_hostname, backend_port, backend_weight, backend_data_directory, backend_flag

Setting up pool_hba.conf for client authentication (HBA)

- Just like `pg_hba.conf` with PPAS, pgpool supports a similar client authentication
- This is done using `pool_hba.conf`
- By default, `pool_hba` authentication is disabled
- Change `enable_pool_hba` to on to enable it
- Copy `pool_hba.conf.sample` as `pool_hba.conf` and edit it if necessary
- Format and usage is similar to `pg_hba.conf`
- Only "trust", "reject", "md5" and "pam" for METHOD field are supported
- To use md5 authentication, you must register username and password in `pool_passwd`
- Users can connect directly to the database and in such case `pg_hba.conf` file settings are used for authentication

Starting/Stopping pgpool-II

- All the backends and the System DB (if necessary) must be started before starting pgpool-II
- If pgpool is installed using rpm packages, use the init scripts from `/etc/init.d/`
- Syntax for `pgpool` command to start pgpool-II:
 - `pgpool [-c][-f config_file][-a hba_file][-F pcp_config_file][-n][-D][-d]`
- There are two ways to stop pgpool-II
- One is via a PCP command in interface and second is a `pgpool` command
- Syntax for `pgpool` command:
 - `pgpool [-f config_file][-F pcp_config_file] [-m {s[mart]|f[ast]|i[mmediate]}} stop`

pgpool-II Example

- In this example we will learn how to install pgpool-II
- The following steps can be performed using root user or sudo:
 1. Download pgpool-II
 - <http://www.pgpool.net/download.php?f=installer-pg93-3.3.4.tar.gz>
 2. Unzip the tar file
 - `tar -zxvf pgpool-II-3.3.4.tar.gz`
 - `cd pgpool-II-3.3.4`
 3. You can view the parameters that can be configured before compiling pgpool-II sources
 - `./configure --help`
 4. Compile and install the sources
 - `./configure --with-pgsql=/opt/PostgresPlus/9.4AS/bin --with-pgsql-includedir=/opt/PostgresPlus/9.4AS/include --with-pgsql-libdir=/opt/PostgresPlus/9.4AS/lib --prefix=/opt/PostgresPlus/9.4`
 - `make`
 - `make install`

pgpool-II Example - Continued

5. After installation lets copy the sample config files

- `cd /opt/PostgresPlus/9.4`
- `cp etc/pcp.conf.sample etc/pcp.conf`
- `cp etc/pgpool.conf.sample etc/pgpool.conf`

6. Generate password:

- `bin/pg_md5 -p edb`

7. Add a user entry in pcp.conf file

- `vi etc/pcp.conf`
- `enterprisedb:e8a48653851e28c69d0506508fb27fc5`

8. Save and close

pgpool-II Example - Continued

9. Open pgpool.conf file and configure pgpool for providing connection pooling for our default PPAS Cluster running on port 5444

- `vi etc/pgpool.conf`

10. Edit following parameters

- `listen_addresses = '*'`
- `port = 9999`
- `backend_hostname0 = 'localhost'`
- `backend_port0 = 5444`
- `backend_weight0 = 1`
- `backend_data_directory0 = '/opt/PostgresPlus/9.4AS/data'`

11. Save and close

pgpool-II Example – Final Steps

13. Start pgpool-II

- `/opt/PostgresPlus/9.4AS/bin/pgpool -f
/opt/PostgresPlus/9.4AS/etc/pgpool.conf -F
/opt/PostgresPlus/9.4AS/etc/pcp.conf -n`

14. Connect to PostgresPlus Cluster using pgpool

- `/opt/PostgresPlus/9.4AS/bin/edb-psql -p 9999 postgres postgres`

15. Exit from the psql terminal and stop pgpool

- `/opt/PostgresPlus/9.4AS/bin/pgpool -mf stop`

pgbouncer - "Bouncer is always right"

- Lightweight connection pooler for PPAS
- Any application can connect to **pgbouncer** as it connects with PPAS
- **pgbouncer** can help to reduce the connection impact on the PPAS Server
- pgbouncer provides connection pooling thus reuse of the existing connections

pgbouncer Features

- **pgbouncer** has low memory requirements (2k per connection by default)
- **pgbouncer** is a database-based pooler thus can connect to a database from different clusters
- Not tied to one backend server
- Supports online restart/upgrade without dropping client connections
- Supports Windows and Linux platforms
- Bundled in PPAS GUI installer

Types of Connections

pgbouncer supports several types of pooling when rotating connections:

- **Session pooling**
 - A server connection is assigned to the client application for the life of the client connection
 - This is the default
- **Transaction pooling**
 - A server connection is assigned to the client application for the duration of a transaction
- **Statement pooling**
 - A server connection is assigned to the client application for each statement

How Connections are Established

- An application connects to pgbouncer as if it were a PPAS database
- **pgbouncer** then creates a connection to the actual database server, or it reuses one of the existing connections from the pool:
 - Step 1: The client application attempts to connect to PPAS on the port where pgbouncer is running
 - Step 2: The database name supplied by the client application must match with the list in pgbouncer.ini
 - Step 3: The user name and password supplied must match with the list in users.txt
 - Step 4: If a connection with the same settings is available in the pool it will be assigned to the client otherwise a new connection object will be created
 - Step 5: Once the client logs off the connection object returns back to the pool

Setting Up pgbouncer

- Create a pgbouncer.ini file:
 - `[databases] db1 = host=127.0.0.1 port=5444 dbname=db1`
 - `[pgbouncer] listen_port = 6543 listen_addr = 127.0.0.1 auth_type = md5 auth_file = users.txt logfile = pgbouncer.log pidfile = pgbouncer.pid admin_users = someuser`
- Create a users.txt file:
 - `"someuser" "same_password_as_in_server"`
- Launch **pgbouncer**:
 - `$ pgbouncer -d pgbouncer.ini`
- Have your application (or the psql client) connect to pgbouncer instead of directly to the PPAS server:
 - `$ psql -p 6543 -U someuser template1`

Manage pgbouncer

- SHOW STATS, SERVERS, CLIENTS, POOLS, LISTS, DATABASES, FDS commands can be used.
- Manage **pgbouncer** by connecting to the special administration database pgbouncer and issuing show help;
 - ```
$ psql -p 6543 -U someuser pgbouncer
pgbouncer=# show help;
NOTICE: Console usage
DETAIL: SHOW
[HELP|CONFIG|DATABASES|FDS|POOLS|CLIENTS|SERVERS|SOCKETS|LISTS|VERSION
]
```

# pgbouncer Example

- In this example we will install and configure pgbouncer to provide connection pooling for Postgres Plus Standard Server Cluster running on port 5444
- Steps:
  1. Open Application Stack Builder Plus from application menu
  2. Select default cluster running on port 5444 and click next
  3. Select pgbouncer from add ons and click next
  4. Select port (default 6432) for pgbouncer and click next to install
  5. pgbouncer will automatically start and can be stopped, restarted using the pgbouncer service that will register automatically
  6. If pgbouncer is not running you can start it using:
    - `pgbouncer -d /opt/PostgresPlus/9.4AS/pgbouncer/share/pgbouncer.ini`

# pgbouncer Example

- Lets configure pgbouncer to provide connection pooling for edbstore database to postgres user
- Steps:
  7. Verify pgbouncer is running:
    - `ps aux | grep pgbouncer`
  8. Open pgbouncer.ini file and add the entry for edbstore
    - `vi /opt/PostgresPlus/9.4AS/pgbouncer/share/pgbouncer.ini`
    - `edbstore = host=127.0.0.1 port=5444 user=postgres password=postgres dbname=edbstore`
    - Verify following entries:
      - `listen_addr = *`
      - `listen_port = 6432`
      - `auth_file = /opt/PostgresPlus/9.4AS/pgbouncer/etc/userlist.txt`
      - `pool_mode = session`

# pgbouncer Example

## 9. Add the postgres user in the user authentication file

- `vi /opt/PostgresPlus/9.4AS/pgbouncer/etc/userlist.txt`
- `"postgres" "postgres"`

## 10. Finally restart pgbouncer to reflect the changes

- `/etc/init.d/pgbouncer restart`

## 11. Connect to the edbstore database using pgbouncer

- `psql -p 6432 edbstore postgres`

# Summary

In this module we covered:

- Connection Pooling Overview
- pgpool-II - Features
  - Connection pooling
  - Replication
  - Load balance
  - Limiting exceeding connections
  - Automatic failover
- pgpool II - Installation and Configuration
- Configuring **pcp.conf**, **pgpool.conf**
- Setting up **pool\_hba.conf** for Client Authentication (HBA)
- Starting/Stopping pgpool-II
- **pgbouncer** –
  - Features, setup, and use
  - “Bouncer is always right”
  - Types of connections
  - How connections are established
  - Managing **pgbouncer**





## Day 3 - 6

# EnterpriseDB Failover Manager

# Create fault tolerant database clusters to minimize downtime with Failover Manager

- EFM aids in the creation of highly available configurations of Postgres
- EFM monitors the health of a Postgres HA configuration
- EFM automates the failover process in the event of a failure
- EFM is used in conjunction with Streaming Replication



# EDB Failover Manager Features

- Automatic & manual failover from master to prioritized and 'furthest ahead' replica node
  - Email notifications when cluster status changes
  - User configurable wait times
- Witness node provides protection against 'split brain' scenarios
- Configurable fencing operation
  - By default uses VIP
  - Parameter to specify alternative operations via fencing scripts  
Ex: reconfigure a load balancer
- Built on PPCD/Jgroups technology

# Release History and What's New in 2.0

- 1.0 – December, 2013
  - Address a gap in available and reliable solutions to **monitor the health of a streaming replication cluster and initiate failover** if a master database fails.
  - Provides a **lightweight and non single-point-of-failure technology** that is **easy to install, configure and use.**
- 1.1 – June, 2014
  - Improved Admin Experience with Agents managed as long running OS Services
  - Added Security with JGroups
  - Improved Cluster and Promote Status Information
- 2.0 – June, 2015
  - **Support for multiple standby replica nodes (> 2 node HA cluster)**
  - Use of Hostname in addition to IP address when defining agent or witness nodes
  - Enhanced fencing to support post-promotion scripts
  - **Support for RHEL / CentOS / OEL 7**

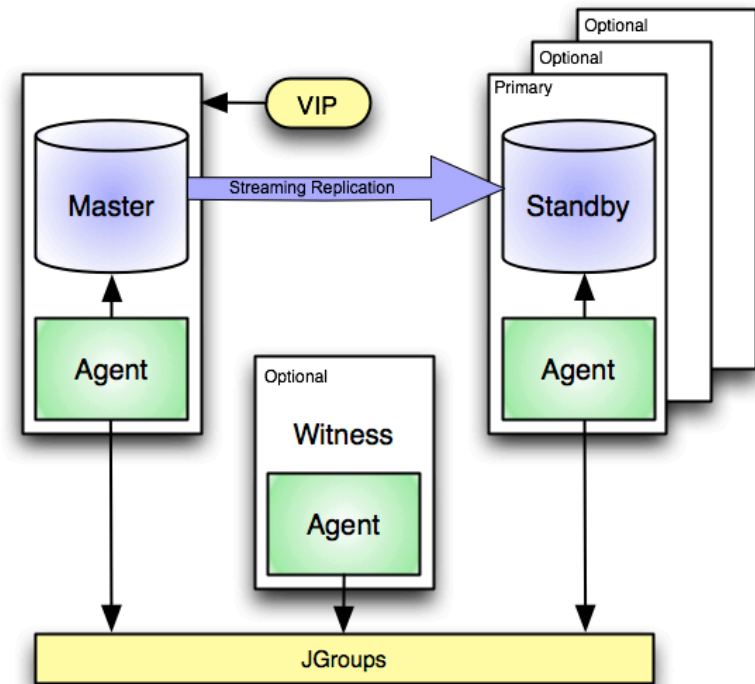
# EFM Architecture and Components

- Agents

- Run on the master and standby nodes
- Monitors the health of those databases
- Communicates with each other and witness to determine cluster health
- Notifies user if an issue is detected
- Initiates failover if needed

- Witness node

- Confirms assertions of either master or standbys
- Breaks tie in the event of conflicting assertions



Example Failover Manager cluster that employs a virtual IP address. You can use a load balancer in place of a virtual IP address if you provide your own fencing script to re-configure the load balancer in the event of a failure.

# Failure Detection and Failover Execution

- Failover Manager supports a very specific set of failover scenarios. Failover can occur:
  - if the Master database crashes or is shutdown.
  - if the node hosting the Master database crashes, reboots, or becomes unreachable due to network connectivity issues.
- If agents cannot confirm that the Master database or node has failed, Failover Manager will not perform any failover actions.
- Failover Manager supports a manual failover mode for situations where you want to monitor and detect failures, but not perform an automatic failover to a Standby.
  - A notification is sent to the administrator when failover conditions are met.
- Failover Manager will alert an administrator to situations that require administrator intervention, but that do not merit promoting a Standby database to Master.

# Configure EFM – Basic Properties

## efm.properties

- `auto.failover=true`
  - EFM will invoke failover in addition to sending email notifications.
- `auto.reconfigure=true`
  - After Failover, EFM will attempt to update remaining standbys to use new master.
- `db.user= .password.encrypted= .port= .database=`
  - Database connection information; use `./efm encrypt` command for password.
- `db.recovery.conf.dir=`
  - Location of the `recovery.conf` file. Rewritten with new master after failover.
- `db.service.owner=`
  - User that owns `$PGDATA`, has write permission to `recovery.conf.dir`.
- `user.email=`
  - Email address to send notifications to. Same across agents.

# Configure EFM – Failure Detection Controls

## efm.properties

Local and remote props apply to connection(s) EFM uses to monitor the databases.

- `local.period=10`
  - Seconds btwn attempts to connect to DB
- `local.timeout=60`
  - Seconds to wait for a response from the db server
- `local.timeout.final=10`
  - If a response is not received from db within # of seconds specified, the database is assumed to have failed.
- `db.reuse.connection.count=`
  - Number of times EFM will reuse db connection to check db health. Default is 0, indicating fresh connection each time.

- `remote.timeout=`
  - how long a standby waits for a # DB ping request from itself and the other nodes before performing failover.

jgroups properties. Number of milliseconds before connection attempts timeout & number of retries. Same across agents

- `jgroups.max.tries=8`
- `jgroups.timeout=5000`

Pingserver properties - Used to test the reachability of external server to determine networking issues

- `PingServerCommand=`
- `PingServerIP=`



# Configure EFM – Virtual IP or Load Balancer

## efm.properties

- EFMs support for Virtual IP Processing - IP and netmask that will be remapped during failover. Same across agents
  - `virtualIp=`
  - `virtualIp.interface=`
    - value must contain the secondary virtual ip # id (ie `:1`", etc).
  - `virtualIp.netmask=`
- EFMs support for Fencing Scripts – Scripts to be called before and after promoting standby
  - `script.fence=`
    - Path to fencing script to be run during promotion
  - `script.post.promotion`
    - Path to script to be run after promotion

# Configure EFM – Manage Nodes and Priorities

## efm.nodes

- efm.nodes file - Add IP addresses and port numbers of other servers that are members of the cluster. Each node must be listed in an address:port format, separated by white space.
  - 10.0.1.8:7800 10.0.1.9:7800
- Influence promotion priority with efm add-node command.
  - ie - instruct EFM that the acctg cluster member that is monitoring 10.0.1.9:7800 is the primary Standby (1):
  - `efm add-node acctg 10.0.1.9:7800 1`
- In the event of a failover, EFM will identify which Standby node has the most recent data. If two Standby nodes contain equally up-to-date data, the node with a higher priority will be promoted.

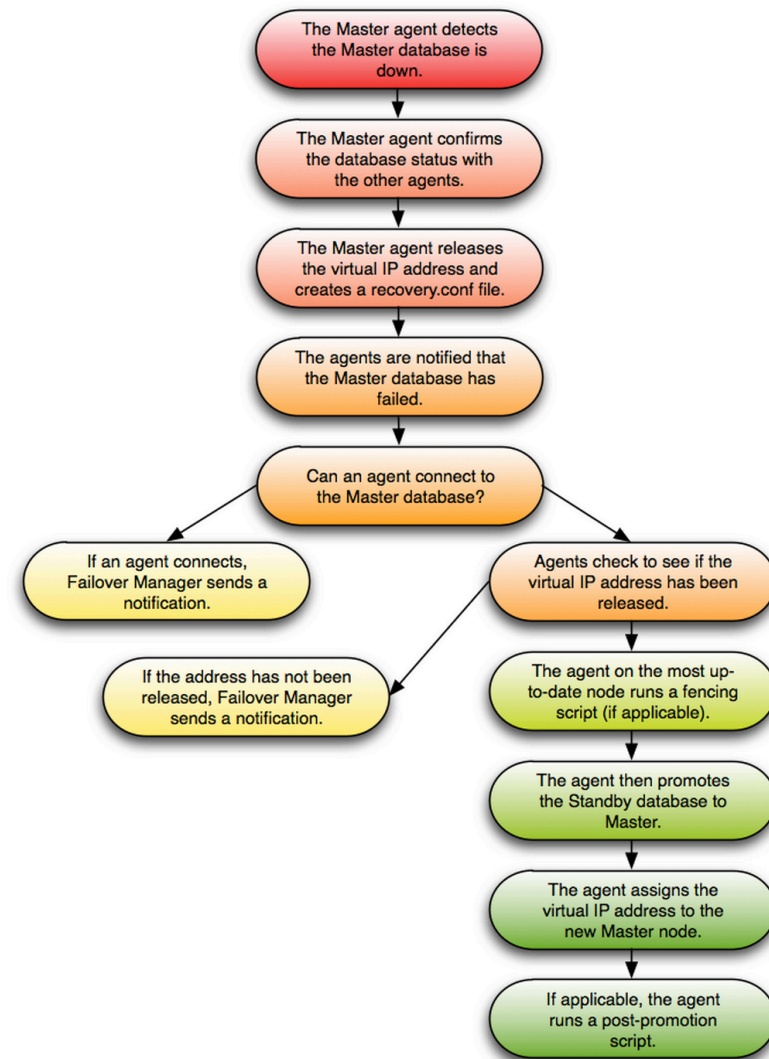
# EnterpriseDB Failover Manager Summary

- Automated failover solution for Highly Available Postgres configurations
- Customizable to meet varying customer needs
- Built on proven/tested technology
- Available now. Download and try today:
- <http://www.enterprisedb.com/download-failover-manager>

# Example Failover Scenario – Master DB Down

Agent on Master node detects that Master DB has failed

1. All agents attempt to connect directly to the Master database.
  1. If any agent can connect, EFM sends an email notification about the state of the Master node.
  2. If no agent can connect, the Master agent declares database failure and releases the VIP (if applicable).
2. The Standby agent on the most prioritized & up-to-date node
  1. Runs a fencing script (if applicable),
  2. Promotes the Standby database to Master database,
  3. Assigns the virtual IP address to the Standby node.
  4. Runs a post-promotion script (if applicable)



# Example Failover Scenario – Master Node Fails

Standby or Witness Agent detects that Master agent has left

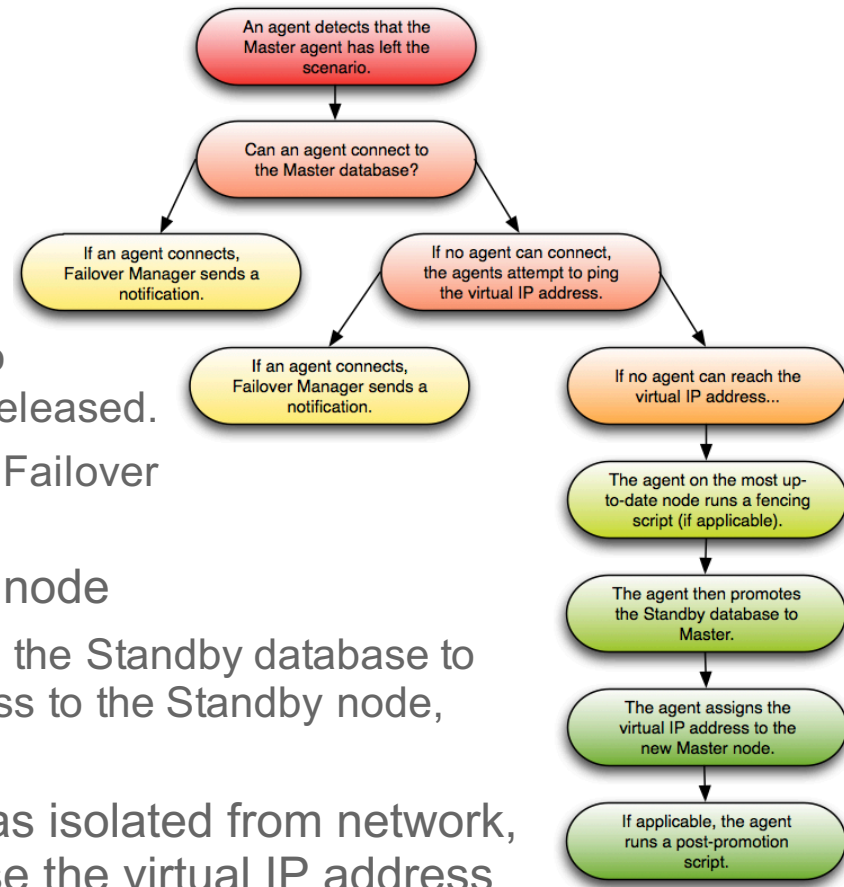
## 1. All agents attempt to connect directly to Master DB.

1. If any agent connects to DB, EFM sends email notification about the failure of the Master agent.
2. If no agent can connect, the agents attempt to ping the virtual IP to determine if it has been released.
3. If no agent can reach the virtual IP or the DB, Failover Manager starts the failover process.

## 2. The Standby agent on the most up-to-date node

1. Runs a fencing script (if applicable), promotes the Standby database to Master database, assigns the virtual IP address to the Standby node, runs a post-promotion script (if applicable)

If this scenario occurred because the master was isolated from network, the Master agent will detect the isolation, release the virtual IP address and create the recovery.conf file. Failover Manager will perform the previously listed steps on the remaining nodes of the cluster.



# EFM Demo