Logical Replication in PostgreSQL

Tallinn 2016

Petr Jelínek
Whoami

• 2ndQuadrant
  – PostgreSQL developer and consultant
• PostgreSQL contributor for over a decade
  – DO, default privileges, TABLESAMPLE, etc
• Pgbouncer co-maintainer
• Contacts
  – petr@2ndquadrant.com
  – https://github.com/pjmodos
Logical Replication

• Target node is writeable
  – Allows temp tables
  – Allows different indexes
  – Allows different security
  – Allows data transformation

• Selective Replication
  – Can replicate subset of database

• Cross-version
History
Logical Replication History

• Trigger based solutions
  – Londiste (~2007)
• Run outside of the PostgreSQL
• Use table(s) as queue
  – Amplify load on the upstream
  – No sync replication
• Complex code to ensure commit order
Current Development

• BDR
  – Modified PostgreSQL 9.4 + extension
  – 9.6 coming soon (extension only)
  – Multi-master
  – Transparent DDL

• pglogical
  – Extension for 9.4+
  – Mostly for one way replication
  – Replacement for trigger-based solutions
Streaming Replication
Physical Streaming Replication

Master

Apps

Standby

Executor

Heap

WalSender

WAL

Executor

Heap

WalReceiver
pglogical
The image illustrates a diagram related to the PostgreSQL database system, specifically focusing on the PGLogical component. The diagram shows the flow of data and processing between different components such as Provider, Subscriber, and Apps. Key elements include:

- **Provider**: Contains Executor and WalSender components.
- **Subscriber**: Contains Executor and Apply components.
- **Apps**: Central component connecting Provider and Subscriber.
- **WalSender**: Outputs WAL (Write-Ahead Logging) data.
- **WAL**: Write-Ahead Logging data flow.
- **Heap**: Storage area of the database.
- **Output plugin**: Part of the WalSender component.

The diagram emphasizes the logical flow of data and operations in a PostgreSQL database system, highlighting the roles of different components in managing transactions and data integrity.
pglogical

• Selective Replication
• Online Upgrade
• Data Transport
  – Data integration
  – Streaming changes to analytical database
  – Master configuration data management
  – …
• Optionally synchronous apply
pglogical

• Installs as extension
  – Runs as part of PostgreSQL instance
  – All configuration is inside the database
• Uses logical decoding to read WAL
  – Minimal overhead on provider
  – Transactions are sent in commit order
• Executes triggers marked as ENABLE REPLICA on subscriber
Installation

- Extension
  - `CREATE EXTENSION pglogical;`

- Provider
  - `create_node('myprovider', 'dbname=foo host=10.10.1.1')`

- Subscriber
  - `create_node('mysubscriber', 'dbname=foo host=10.10.1.2')`
  - `create_subscription('mysubscription', 'dbname=foo host=10.10.1.1')`
Replication Sets

- Replication is defined in terms of groups (sets) of tables, rather than individual tables
  - Need to be defined on each provider node
- Table is not replicated until added to a set
- Tables may be defined in more than one set, but changes for the table will only be sent once to each subscription
Replication Sets

• By default new replication sets replicate all actions
  – INSERT, UPDATE, DELETE, TRUNCATE
• It's possible to filter actions for given replication set
• Useful for data aggregation, data warehousing etc.
• Predefined sets, “default”, “default_insert_only”, “ddl_sql”
Table replication

• Add table to replication set
  – pglogical.replication_set_add_table(
    set_name := 'default',
    relation := 'public.users',
    synchronize_data := true);

• Full data resynchronization possible at later time
  – pglogical.alter_subscriber_resynchronize_table

• Structure cannot be synchronized automatically yet
Sequences

- Replicated using replication sets just like tables
  - `pglogical.replication_set_add_sequence`
- Replicated periodically in bulk
- Dynamic buffering of last value
  - Subscriber is in front of the provider
  - This is similar to how Londiste replicates sequences
DDL Replication

• Initial schema either fully synchronized or not at all
• The DDL commands are not automatically replicated yet
• `pglogical.replicate_ddl_command(command [, replication_sets])`
  – replication_sets defaults to “ddl_sql”
Example setup
Performance (pgbench)
Caveats

• Big transactions may cause replication to lag
  – This is common problem for transactional replication systems
• Does not play well with physical replication yet
  – Failover
• Currently requires superuser
Future
pglogical 2.0
Column Filtering

- Add table to replication set
  - `pglogical.replication_set_add_table(set_name := 'default',
    relation := 'public.users',
    columns := '{id,name,...}');`

- Array of replicated columns
- REPLICA IDENTITY columns required
- The table on subscriber does not need the extra columns
Row based Filtering

- Add table to replication set
  - `pglogical.replication_set_add_table(
    set_name := 'default',
    relation := 'public.users',
    row_filter := 'expression');`

- Standard SQL expression
- Same limitations as CHECK CONSTRAINT
- Executed during replication
  - Session variables of the replication connection
Thanks!

- info@2ndquadrant.com
- bdr-list@2ndquadrant.com
- https://github.com/2ndQuadrant/pglogical