Implementing failover of logical replication slots in Patroni

PGConf.EU 2022, Berlin
Alexander Kukushkin
2022-10-27
About me

Alexander Kukushkin

Principal Software Engineer @Microsoft

The Patroni guy

akukushkin@microsoft.com

Twitter: @cyberdemn
Streaming replication

**Physical**
- replicates binary changes via WAL files over TCP to standby
- can’t replicate between different major versions or platforms

**Logical**
- replicates only data objects and their changes
- works between different major versions and platforms
- requires the replication slot
Enabling logical replication

- `wal_level = logical`  # default value is replica
- `max_wal_senders = 10`
- `max_replication_slots = 10`

Postgres restart is required!
Replication slots

• Provide guarantees that WAL segments are not removed until consumed
• Provide protection against relevant rows being removed by (auto)vacuum
• Logical replication slots are tightly coupled with particular database.
Creating a logical replication slot

• `SELECT pg_create_logical_replication_slot('
  <slot_name>', '<output_plugin_name>');`

  -- or --

• `CREATE_REPLICATION_SLOT <slot_name>
  LOGICAL <output_plugin_name>`

  • `pg_recvlogical --create-slot --slot=<slot_name> \\`
  --plugin=<plugin_name> --dbname=<database_name>`
Logical Decoding Plugins

• Built-in core
  • test_decoding
  • pgoutput (logical replication)
• 3rd party
  • wal2json - JSON
  • pglogical
  • Debezium decoderbufs - Protobuf
  • decoder_raw - as SQL
  • ...

What is Logical Decoding?

• Extracting data changes in a “simple” format that could be interpreted by an external tool.

• Outside of PostgreSQL it is also known as Change Data Capture (CDC)
Example: creating the slot

```sql
localhost/testdb=# SELECT * FROM pg_create_logical_replication_slot ('my_slot', 'test_decoding');

<table>
<thead>
<tr>
<th>slot_name</th>
<th>lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>0/130694C8</td>
</tr>
</tbody>
</table>

(1 row)
```

```sql
localhost/testdb=# SELECT slot_name, plugin, slot_type, database, confirmed_flush_lsn FROM pg_replication_slots;

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>0/130694C8</td>
</tr>
</tbody>
</table>

(1 row)
```
Example: peek changes without consuming

```sql
CREATE TABLE replicate_me (
    id BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name TEXT);

INSERT INTO replicate_me (name) VALUES ('PGConf.EU');

SELECT * FROM pg_logical_slot_peek_changes('my_slot', NULL, NULL);
```

<table>
<thead>
<tr>
<th>lsn</th>
<th>xid</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/130694F8</td>
<td>830</td>
<td>BEGIN 830</td>
</tr>
<tr>
<td>0/1309A768</td>
<td>830</td>
<td>COMMIT 830</td>
</tr>
<tr>
<td>0/1309A7A0</td>
<td>831</td>
<td>BEGIN 831</td>
</tr>
<tr>
<td>0/1309A808</td>
<td>831</td>
<td>table public.replicate_me: INSERT: id[bigint]:1 name[text]:'PGConf.EU'</td>
</tr>
<tr>
<td>0/1309A960</td>
<td>831</td>
<td>COMMIT 831</td>
</tr>
</tbody>
</table>

(5 rows)
Example: pg_recvlogical

$ pg_recvlogical -h localhost -U postgres -d testdb --slot=my_slot --start -f -

BEGIN 830
COMMIT 830
BEGIN 831
table public.replicate_me: INSERT: id[bigint]:1 name[text]:'PGConf.EU'
COMMIT 831
^Cpg_recvlogical: error: unexpected termination of replication stream:
Logical replication example

```
CREATE PUBLICATION my_publication
FOR TABLE replicate_me WITH (publish = 'insert');

CREATE SUBSCRIPTION my_sub
CONNECTION 'host=172.168.18.50 port=5432 user=repl dbname=testdb'
PUBLICATION my_publication;
```
Logical replication slots and HA

Primary

Logical slot

Logical standby

Standby

physical replication

logical replication
Logical replication slots and HA

Primary

Logical slot

Logical standby

ERROR: no logical slot!

Standby
Logical replication slots and HA

• Replication slots are not replicated!
• Logical decoding doesn’t work on standbys
  • Logical replication slots can’t be created on standbys:
    localhost/testdb=# SELECT * FROM
    pg_create_logical_replication_slot ('my_slot', 'test_decoding');
    ERROR: logical decoding cannot be used while in recovery
• Consumers can’t continue work after failover/switchover.
Naive solution in Patroni

- Don’t allow incoming connections after failover/switchover before logical slots are created:
  - Patroni REST API health-check returns 503
  - Delay callbacks
  - K8s leader Service without endpoints

- Logical events could be silently lost if consumer wasn’t running or was lagging!
Can we create logical slots in the past?

• Yes, with the custom extension: https://github.com/x4m/pg_tm_aux

• **Pros:** events won't be lost

• **Cons:** It’s not always possible to install 3rd party extensions

• **Potential problems:**
  • WAL isn’t accessible
  • May fail to take a “catalog snapshot”
Can we do better?
If we restart Postgres in read-only mode, existing replication slots are still there:

```sql
localhost/testdb=# SELECT pg_is_in_recovery();
pg_is_in_recovery
  ┌───────────────────┐
  │ t │
  └───────────────────┘
  (1 row)

localhost/testdb=# SELECT slot_name, plugin, slot_type, database, confirmed_flush_lsn
               FROM pg_replication_slots;

slot_name      plugin           slot_type database confirmed_flush_lsn
--------------- ----------------------- -------------- -------------------------------
my_slot         test_decoding    logical        testdb                   0/1309AA80
(1 row)
```
Logical replication slots can exist on replicas!
How replication slots are stored?

$ ls -l $PGDATA/pg_replslot/my_slot/
total 4
-rw------- 1 postgres postgres 200 Apr 11 08:15 state
What if we copy the slot file?

• Stop postgres on the replica

• Copy the slot file from the primary

• Start postgres on the replica

• Profit?!
Adding a "secret" sauce

- `pg_replication_slot_advance (slot_name name, upto_lsn pg_lsn)`
  - Introduced in PostgreSQL v11 (released in October 2018)
  - Works with logical slots! (must be connected to the right DB)
  - Also works on replicas!
Example: pg_replication_slot_advance()

localhost/testdb=# SELECT pg_is_in_recovery();
pg_is_in_recovery
_________
t
(1 row)

localhost/testdb=# SELECT * FROM pg_replication_slot_advance('my_slot', '0/1412FA78');

<table>
<thead>
<tr>
<th>slot_name</th>
<th>end_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>0/1412FA78</td>
</tr>
</tbody>
</table>

(1 row)
Is it safe?

- Copy the slot file - Yes
- `pg_replication_slot_advance()` - Yes
- Use replication slot after promote - might be unsafe

```sql
localhost/testdb=# SELECT slot_name, plugin, slot_type, database, catalog_xmin, restart_lsn, confirmed_flush_lsn FROM pg_replication_slots;
```

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>catalog_xmin</th>
<th>restart_lsn</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>831</td>
<td>0/1309A768</td>
<td>0/1309AA80</td>
</tr>
</tbody>
</table>

(1 row)
• Logical decoding uses `pg_catalog` to figure out table structures
  • Needs access to old snapshots
• Race conditions:
  • Slots on replicas are usually behind
  • Autovacuum might cleanup `pg_catalog` tuples required for decoding
How to protect from it?

1. Replicas must use replication slots (primary_slot_name GUC)

2. The hot_standby_feedback parameter must be set to on
hot_standby_feedback = off

localhost/testdb=# SELECT slot_name, plugin, slot_type, database, catalog_xmin, restart_lsn, confirmed_flush_lsn FROM pg_replication_slots;

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>catalog_xmin</th>
<th>restart_lsn</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>864</td>
<td>0/14116228</td>
<td>0/1412E410</td>
</tr>
<tr>
<td>postgresql</td>
<td>physical</td>
<td></td>
<td></td>
<td></td>
<td>0/1412E4C0</td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

Replica:

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>catalog_xmin</th>
<th>restart_lsn</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>863</td>
<td>0/140FE578</td>
<td>0/140FE5B0</td>
</tr>
</tbody>
</table>

(1 row)
```sql
SELECT slot_name, plugin, slot_type, database, catalog_xmin, restart_lsn, confirmed_flush_lsn FROM pg_replication_slots;
```

**Primary:**

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>catalog_xmin</th>
<th>restart_lsn</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>864</td>
<td>0/14116228</td>
<td>0/1412E410</td>
</tr>
<tr>
<td>postgresql1</td>
<td></td>
<td>physical</td>
<td></td>
<td>863</td>
<td>0/1412E4C0</td>
<td></td>
</tr>
</tbody>
</table>

(2 rows)

**Replica:**

<table>
<thead>
<tr>
<th>slot_name</th>
<th>plugin</th>
<th>slot_type</th>
<th>database</th>
<th>catalog_xmin</th>
<th>restart_lsn</th>
<th>confirmed_flush_lsn</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>test_decoding</td>
<td>logical</td>
<td>testdb</td>
<td>863</td>
<td>0/140FE578</td>
<td>0/140FE5B0</td>
</tr>
</tbody>
</table>

(1 row)
The plan

• Copy logical slots to replicas with Postgres restart
  • Fsync files and directories after coping!

• Periodically call `pg_replication_slot_advance()`

• Handle possible errors?
Implementation details in Patroni

• Leader: periodically publish **confirmed_flush_lsn** for all permanents slots to the Distributed Configuration Store (DCS)

• Replicas:
  • Use `pg_read_binary_file()` function to copy the slot file if it is missing on the replica
    • Requires superuser or specially configured `rewind_user`
  • Enable **hot_standby_feedback**
Cascading replication

The node2 doesn’t have logical slots, but must have `hot_standby_feedback` enabled (automatically configured by Patroni)
Possible problems

• Requested WAL segment pg_wal/XXX has already been removed

• The logical slot could be unsafe to use after promote if replicas physical slot didn’t reach the `catalog_xmin` of the logical slot on the old primary (Patroni shows a warning)
$ patronictl edit-config
---
+++@@ -1,6 +1,12 @@
  loop_wait: 10
  maximum_lag_on_failover: 1048576
postgresql:
+  use_slots: true
+  parameters:
+  + wal_level: logical
    use_pg_reshape: true
    retry_timeout: 10
  ttl: 30
+permanent_slots:
+  my_slot:
+    database: testdb
+    plugin: test_decoding

Apply these changes?
Configuration changed
Logical client issues

• Clients must be prepared to receive some events for the second time after failover/switchover

• Logical replication may be ahead of physical replication that is acknowledged by synchronous replicas and see some events that didn’t survive failover.

• Debezium doesn’t correctly handle keepalive messages: DBZ-4055
  • May cause slow (or indefinite) shutdown of the primary because it keeps walsender process alive
  • “Breaks” switchover in the old (before 2.1.2) Patroni
Monitoring

```
localhost/postgres=# SELECT slot_name,
     pg_size_pretty(CASE WHEN pg_is_in_recovery() THEN pg_last_wal_replay_lsn()
     ELSE pg_current_wal_lsn() END - confirmed_flush_lsn)
FROM pg_replication_slots WHERE slot_type = 'logical';

<table>
<thead>
<tr>
<th>slot_name</th>
<th>pg_size_pretty</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_slot</td>
<td>5736 bytes</td>
</tr>
</tbody>
</table>

(1 row)
```
Conclusion

• Failover of logical slots is supported by Patroni starting from 2.1.0 (released in July 2021)
  • Requires PostgreSQL v11 or newer
  • The old “feature” (create logical slots after promote) is disabled.
• Used in production at Zalando for more than 30 databases
• Don’t forget about the monitoring.
• Craig Ringer
Thank you!

Questions?