zalando

Adding replication protocol support for `psycopg2`

Oleksandr Shulgin, Valentin Gogichashvili

zalando

- Why Logical Replication?
- Why Python?

zalando

- https://github.com/zalando/bottledwater-pg

```
import bottledwater

def consume(msg):
    print(msg.payload)

bottledwater.export(consume, 'dbname=test',
                    slot_name='bwtest',
                    format='JSON')
```

zalando

What you will need:

- Python >= 2.5 or 3.x
- https://github.com/psycopg/psycopg2/pull/322
- PostgreSQL 9.4
- A PostgreSQL Logical Decoding Output Plugin

- Compile `psycopg2`!
- `pg_hba.conf`: enable replication connections
  - `local  replication  testrepl  trust`
- `postgresql.conf`:
  - `wal_level = logical`
  - `max_wal_senders > 0`
  - `max_replication_slots > 0`

zalando

Or you can just use our Docker image to try it out!

```
# the server
$ docker run -d --name pg zalando/postgres-bw:0.1
```

zalando

Or you can just use our Docker image to try it out!

```
# the server
$ docker run -d --name pg zalando/postgres-bw:0.1

# the client
$ docker run -it --rm --link pg:pg zalando/postgres-bw:0.1 \
    sh -c 'PGHOST="$PG_PORT_5432_TCP_ADDR" \
           PGPORT="$PG_PORT_5432_TCP_PORT" PGUSER=postgres \
           exec python /root/bwtest.py'
# that's some docker magic ;-)
```

This is our test consumer terminal

```
$ docker run -it --rm --link pg:pg zalando/postgres-bw:0.1 ...

Replication slot doesn't exist.
Creating logical replication slot bwtest for bottledwater...
Streaming changes on the replication slot: bwtest 0/16B6BE8
```

Meanwhile, on another terminal: launch psql

zalando

```
$ docker run -it --rm --link pg:pg zalando/postgres-bw:0.1 \
    sh -c 'exec psql -h "$PG_PORT_5432_TCP_ADDR" \
                     -p "$PG_PORT_5432_TCP_PORT" -U postgres'

psql (9.4.5)
Type "help" for help.

postgres=# CREATE TABLE demo AS
postgres-# SELECT * FROM generate_series(1, 5) AS id;
SELECT 5
postgres=#
```

Back to the consumer terminal

```
$ docker run -it --rm --link pg:pg zalando/postgres-bw:0.1 ...
...
{ "command": "BEGIN",  "xid": 688, "dbname": "postgres" }
{ "command": "INSERT", "xid": 688, "wal_pos": "0/16C38A8", "dbname": "postgres",
"relname": "demo", "relnamespace": "public", "newtuple":{"id":1} }
{ "command": "INSERT", "xid": 688, "wal_pos": "0/16C38E8", "dbname": "postgres",
"relname": "demo", "relnamespace": "public", "newtuple":{"id":2} }
{ "command": "INSERT", "xid": 688, "wal_pos": "0/16C3928", "dbname": "postgres",
"relname": "demo", "relnamespace": "public", "newtuple":{"id":3} }
{ "command": "INSERT", "xid": 688, "wal_pos": "0/16C3968", "dbname": "postgres",
"relname": "demo", "relnamespace": "public", "newtuple":{"id":4} }
{ "command": "INSERT", "xid": 688, "wal_pos": "0/16C39A8", "dbname": "postgres",
"relname": "demo", "relnamespace": "public", "newtuple":{"id":5} }
{ "command": "COMMIT", "xid": 688, "dbname": "postgres" }
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
# dbname=psycopg2test user=testrepl replication=database

cur = conn.cursor()
# psycopg2.extras.ReplicationCursor
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
cur = conn.cursor()

cur.create_replication_slot('pytest', output_plugin='test_decoding')
# CREATE_REPLICATION_SLOT "pytest" LOGICAL "test_decoding"
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
cur = conn.cursor()

cur.create_replication_slot('pytest', output_plugin='test_decoding')
cur.start_replication(slot_name='pytest', decode=True)
# START_REPLICATION SLOT "pytest" LOGICAL 0/0
#
# decode=True:
#   test_decoding produces textual output: ask for unicode conversion
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
cur = conn.cursor()

cur.create_replication_slot('pytest', output_plugin='test_decoding')
cur.start_replication(slot_name='pytest', decode=True)

def consume(msg):
    print(msg.payload)

cur.consume_stream(consume)
# ^^^ endless loop: stop with Control-C
# The replication slot is still there, drop it when you don't need it.
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
cur = conn.cursor()

cur.create_replication_slot('pytest', output_plugin='bottledwater')
cur.start_replication(slot_name='pytest', options={ 'format': 'JSON' },
    decode=True)

# ...
# When the slot is no longer needed, you can drop it. Make new connection!
cur = new_conn.cursor()
cur.drop_replication_slot('pytest')
# DROP_REPLICATION_SLOT "pytest"
```

zalando

```
import psycopg2
from psycopg2.extras import LogicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection)
cur = conn.cursor()

cur.start_replication(slot_name='pytest') # if you didn't drop that slot…

class DemoConsumer(object):
    def __call__(self, msg):
        print(msg.payload)
        msg.cursor.send_feedback(flush_lsn=msg.data_start)

consumer = DemoConsumer()
cur.consume_stream(consumer)
```

Stopping Replication Demo

zalando

```
from psycopg2.extras import StopReplication

cur.start_replication(slot_name='pytest', decode=True)

class DemoConsumer(object):
    def __call__(self, msg):
        print(msg.payload)
        msg.cursor.send_feedback(flush_lsn=msg.data_start)
        if 'stop_repl' in msg.payload:          # a very synthetic example!
            raise StopReplication()

consumer = DemoConsumer()
try:
    cur.consume_stream(consumer)
except StopReplication:
    print('stopping replication')
```

There's also… Physical Replication!

zalando

```
import psycopg2
from psycopg2.extras import PhysicalReplicationConnection

conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=PhysicalReplicationConnection)
cur = conn.cursor()

# Physical Replication doesn't require a named slot, but it can use one.
# If slot is not used, you need to provide the start LSN!
cur.start_replication(start_lsn=…)          # decode=False, binary data

def consume(msg):
    print('got replication message of size %d' % msg.data_size)
    msg.cursor.send_feedback(flush_lsn=msg.data_start)

cur.consume_stream(consume)
```

Asynchronous Connections Support!

zalando

```
conn = psycopg2.connect('dbname=psycopg2test user=testrepl',
    connection_factory=LogicalReplicationConnection, async=True)
cur = conn.cursor()

cur.start_replication(slot_name='pytest')

while True:
    msg = cur.read_message()
    if msg:
        consume(msg)
    else:
        now = datetime.now()
        timeout = 10.0 - (now - cur.io_timestamp).total_seconds()
        sel = select.select([cur], [], [], max(0, timeout))
        if not any(sel):
            cur.send_feedback()  # timed out, send a keepalive message
```

zalando

- https://github.com/confluentinc/bottledwater-pg
- https://github.com/zalando/bottledwater-pg
- https://github.com/psycopg/psycopg2/pull/322
- https://hub.docker.com/r/zalando/postgres-bw/

Thank you!

Questions?