Pushy Postgres and Python

BrightLink

Help people prove themselves worthy

↓

Candidate/credential &

Performance exam management

- small business
- candidate & credential management
- Python stack

Start with

Why?

You

- *always on*, *social* users
- scale out: throwing hardware at the problem only goes so far
- more visibility w/o hurting performance

Me

- love Python
- love sharing knowledge

tl;dr

Use Postgres?

Get data-driven push notices in your distributed app for free!

❦,

`@drocco007`

- This is the one-slide version of this talk
- add distrib. message processing with
- data-driven push notices
- no additional infrastructure, ops changes, etc.

Outline:

- overview of Observer
- Notify/Listen in PG
- Case study

Demand
DB

- Do username & password match?
- How many people registered for event?
- Find email addresses for all users in OH

Ubiquitous Connectivity

"Don't call us, we'll call you."

New paradigm

SQL?

Just imagine…

```
    while True:
        db.execute('SELECT are_we '
                   'FROM there_yet;')
        sleep(10)
```

(if all the apps on your phone did this…)

You are here

responsiveness ⇅ system health

Are you watching closely?

$$\begin{tikzcd}
 & \text{observer1} \\
\boxed{\text{subject}} \arrow[ur, "\text{notify}"] \arrow[r] \arrow[dr] & \text{observer2} \\
 & \dots
\end{tikzcd}$$

- Observer pattern
- Subject: thing being observed
- Subject: doesn't need to know about observer purpose, #
- Encapsulate action from reaction
- Python's first-class functions makes this easy

UI

$$\begin{tikzcd}
 & \text{A/B usage tracker} \\
 & \text{Analytics collector} \\
\boxed{\text{button}} \arrow[uur, "\text{click}"] \arrow[ur] \arrow[r] \arrow[dr] & \text{Form validator} \\
 & \dots
\end{tikzcd}$$

```
$("#action-btn").on("click", …)
```

Django

auth.user_logged_in etc.

`models`

SQLAlchemy

- session events
- attribute/model updates

Tradeoffs

- decouple action from reaction: flexibility & focus
- widely applicable
- can lead to complex interactions & difficult to trace cascades

* * *

Postgres does this!

Look at how Postgres implements Observer

PostgreSQL: NOTIFY/LISTEN

DB connected clients can

- LISTEN on a message channel
- NOTIFY other listeners of events
- not directly tied to DB objects (relations, tuples, etc.)

Channeling

```
LISTEN anything_good_on_this
```

- channel name is an *identifier*
- 63 char effective limit (w/o recompilation), truncates
- quoted names are case sensitive, unquoted names are not (!)

“Message for you, sir!”

```
NOTIFY arthur_king_of_the_britons, 'ARROWED!'
```

psql NOTIFY/LISTEN

Transaction Interaction

```
BEGIN;
NOTIFY unspeakable_thoughts, 'I''m reluctant to tell you this…'
ROLLBACK;
```

Transaction Interaction

```
LISTEN im_sorry_did_you_say_something;

BEGIN;

-- NOTIFY posted here

SELECT some_data FROM this_monster_view;
UPDATE table_with_a_zillion_indexes…;

COMMIT;

-- message delivered here
```

Typical…

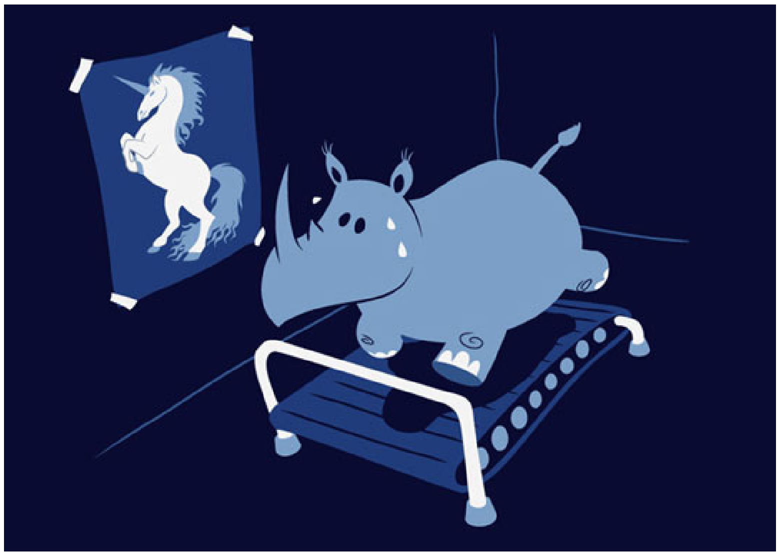
```
CREATE FUNCTION new_user_handler() RETURNS TRIGGER AS $$
BEGIN
    EXECUTE 'NOTIFY new_user;';
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER app_user_trigger
AFTER INSERT ON app_user
      FOR EACH ROW EXECUTE PROCEDURE new_user_handler();
```

Use Case

- async job processing
- separate from app server
- e.g. reports, batch/cleanup jobs, notifications

Goals



- keep UI responsive for tasks initiated by user
- minimize percieved impact on system

Why [! $OTHER_TECH]

DevOps has work to do. Duh.

Leverage existing infrastructure

Data Proximity

Dovetail with TRIGGER

Approach

- user action/automated tasks create jobs
- job triggers notice
- worker farm picks up new job & runs it

Trigger Happy

```
CREATE FUNCTION job_queued_handler() RETURNS TRIGGER AS $$
BEGIN
    EXECUTE 'NOTIFY job_queued, ''' || NEW.queue_name || ''';';
    RETURN NULL;
```

```
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER job_queued_trigger
AFTER INSERT ON job_queue
      FOR EACH ROW EXECUTE PROCEDURE job_queued_handler();
```

NEW is the row that has been inserted

PyListen

```
conn = psycopg2.connect(DSN)
conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
cursor = conn.cursor()

cursor.execute('LISTEN job_queue_control_message;')
cursor.execute('LISTEN job_queued;')

while self.running:
    if select.select([conn], [], [], 10) != ([], [], []):
        conn.poll()
        while conn.notifies:
            notice = conn.notifies.pop()
            print("Got NOTIFY from PID {}: {} {}"
                  .format(notice.pid, notice.channel, notice.payload))
```

Get the Word Out

```
def dblisten(q):
    # connect/loop as above
    q.put(notice)

@websocket.WebSocketWSGI
def broacast_notify(ws):
    "Relay PG NOTIFY to WebSocket"

    q = eventlet.Queue()
    eventlet.spawn(dblisten, q)
    while True:
        notice = q.get()
        ws.send(unicode(notice.payload))
```

or gevent, ws4py, ∅MQ, Redis PUB/SUB, etc.

Ideas

- notify active subscribers
- streaming monitoring/analytics
- cache invalidation
- µservice architecture

- notify active user/post subscribers
- streaming analytics
- monitoring/event notification
- cache invalidation
- µservice architecture
- biz KPIs

Performance Notes

- in production for 1.5+ years, millions of jobs
- many concurrent notices → n^2 notify performance
  - mitigate w/ relay

❦

@drocco007