PGQ, Pretty Darn Quick

Dimitri Fontaine, Title by Selena Deckelmann

May 22, 2009

Table of contents

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while( true ) {
  // what a nice daemon!
}
```

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while( true ) {
  // what a nice daemon!
}
```

Of course you want them

- reliable, easy to monitor and control (logs)

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while( true ) {
  // what a nice daemon!
}
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while( true ) {
  // what a nice daemon!
}
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session
- easy to stop & restart

Database processing oriented batches

If you're managing an *OLTP* system, you probably have out of line processing to get done, and probably are using cron batches and home made *daemons*.

Example

```
while( true ) {
  // what a nice daemon!
}
```

Of course you want them

- reliable, easy to monitor and control (logs)
- out of a developer `screen` session
- easy to stop & restart
- to reuse existing models

Reusing Open-Source code?

PGQ is a queuing solution implemented as a PostgreSQL extension module, providing an `SQL` and a `python` API. It offers the producer multi consumers subscribtion model and is the transport layer of the `londiste` replication solution.

Reusing Open-Source code?

PGQ is a queuing solution implemented as a PostgreSQL extension module, providing an `SQL` and a `python` API. It offers the producer multi consumers subscribtion model and is the transport layer of the `londiste` replication solution.
PGQ is

- performant (maintaining 3 tables, using `TRUNCATE`)

Reusing Open-Source code?

PGQ is a queuing solution implemented as a PostgreSQL extension module, providing an `SQL` and a `python` API. It offers the producer multi consumers subscribtion model and is the transport layer of the `londiste` replication solution.
PGQ is

- performant (maintaining 3 tables, using `TRUNCATE`)
- easy to install and monitor

Reusing Open-Source code?

PGQ is a queuing solution implemented as a PostgreSQL extension module, providing an `SQL` and a `python` API. It offers the producer multi consumers subscribtion model and is the transport layer of the `londiste` replication solution.
PGQ is

- performant (maintaining 3 tables, using `TRUNCATE`)
- easy to install and monitor
- robust

Implementing a daemon atop PGQ

Skytools comes with two *middleware* for you to abuse to make daemons with, the `python` *DBScript* facility and the `PHP` *libphp-pgq* one.

Implementing a daemon atop PGQ

Skytools comes with two *middleware* for you to abuse to make daemons with, the `python` *DBScript* facility and the `PHP` *libphp-pgq* one.

For the `python` version see resources on the internet. Here we're dealing with the `PHP` one. `PHP` is inferior a language but we sometime have to support it, to leverage existing model classes.

libphp-pgq

You subclass a `PGQConsumer` superclass and implement two methods:

- `config`
- `process_event`

libphp-pgq

You subclass a `PGQConsumer` superclass and implement two methods:

- `config`
- `process_event`

The transaction `BEGIN` and `COMMIT` are managed by the class you inherited code from, just process data and return one of

```
PGQ_EVENT_OK
PGQ_EVENT_FAILED
PGQ_EVENT_RETRY
PGQ_ABORT_BATCH
```

libphp-pgq abstract classes

Different main classes are available for you to subclass, depending on what you want to do. Some documentation is available online at http://pgsql.tapoueh.org/pgq/pgq-php/

libphp-pgq abstract classes

Different main classes are available for you to subclass, depending on what you want to do. Some documentation is available online at http://pgsql.tapoueh.org/pgq/pgq-php/

- `PGQConsumer`

libphp-pgq abstract classes

Different main classes are available for you to subclass, depending on what you want to do. Some documentation is available online at http://pgsql.tapoueh.org/pgq/pgq-php/

- `PGQConsumer`
- `PGQRemoteConsumer`

libphp-pgq abstract classes

Different main classes are available for you to subclass, depending on what you want to do. Some documentation is available online at http://pgsql.tapoueh.org/pgq/pgq-php/

- `PGQConsumer`
- `PGQRemoteConsumer`
- `PGQEventRemoteConsumer`

Gone live!

We have several `PGQ` daemons in our live environments, monitored with `nagios` and `munin`, and we're able to easily control them. It's much better than previously.

Gone live!

We have several PGQ daemons in our live environments, monitored with nagios and munin, and we're able to easily control them. It's much better than previously.

Example

```
# mydaemon.php start
# mydaemon.php status
# mydaemon.php logmore

# mydaemon.php kill
```