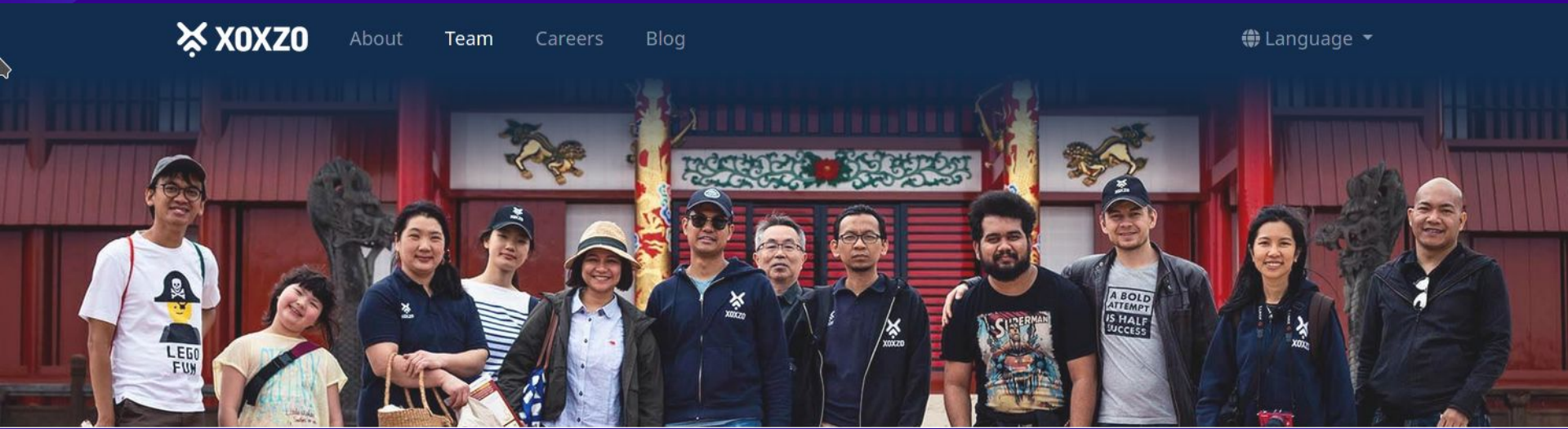
Build multi-tenant task queue using PostgreSQL and Python

Kamal Mustafa @ LaLoka Labs (kamal@lalokalabs.co)

LaLoka Labs

Who am I?

- CTO at LaLoka Labs / Xoxzo Inc
  - Build otp.dev - OTP API for web application
  - At Xoxzo, we build telephony APIs, (SMS, Voice)
- Web developer - Python/Django
- From Malaysia
- Full bio at https://kamal.koditi.my/


XOXZO
About
Team
Careers
Blog
Language
LEGO FUN
A BOLD ATTEMPT IS HALF SUCCESS

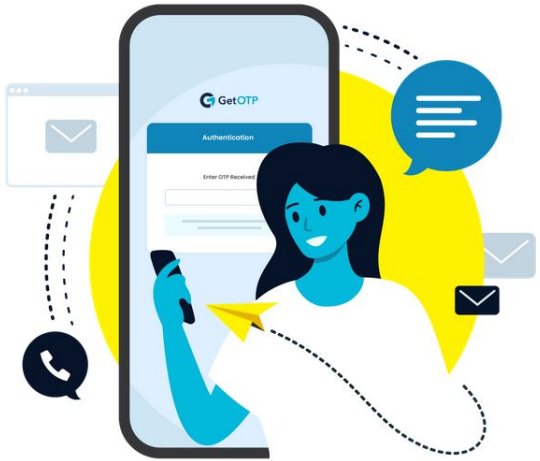
GetOTP Home Pricing Docs Blog Demo Members Area

Fast and Simple OTP Flow

Build a complete, multichannel One-Time Password flow with almost no code. Enjoy our ready-made UI, and save time on resources for your development.

Try For Free →

Try First, Subscribe Later



LaLoka Labs

We ❤ Python

https://blog.xoxzo.com/en/tag/python/

LaLoka Labs

What's in this talk?

- Why build custom task queue?
- Why using PostgreSQL?
- What is multi-tenant task queue?
- Basic implementation in Python
- A demo?
- Questions

LaLoka Labs

Why build custom task queue?

LaLoka Labs

Hard to find task queue with multi-tenant support and low maintenance overhead (in 2010)

LaLoka Labs

Why using PostgreSQL as broker?

PostgreSQL as broker

- We probably already have it, so less thing to manage
- Queue inspection using common and established tools - SQL!
- Persistency for free
- Atomic processing with respect to other database work
- Custom requirement - multi-tenant support?

LaLoka Labs

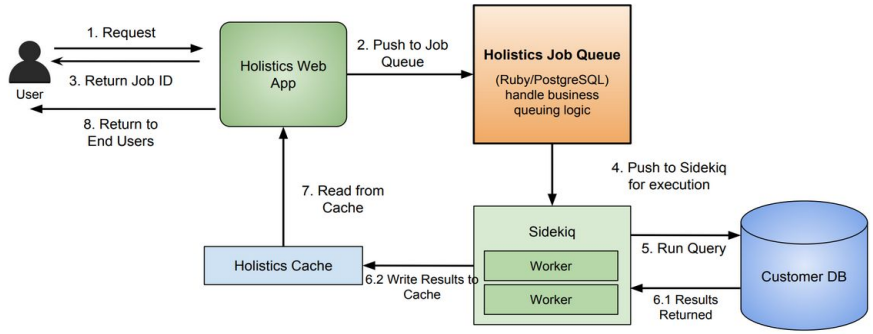
Multi-tenant job queue with PostgreSQL

Holistics Blog Start Here Business Intelligence Data Modeling Using Holistics Our Newsletter Visit Holistics

24 JANUARY 2018 / 8 MIN READ / ENGINEERING, INNER JOIN

How We Built A Job Queue System with PostgreSQL & Ruby For Our B2B SaaS Application

by Huy Nguyen



holistics.io

https://www.holistics.io/blog/how-we-built-a-multi-tenant-job-queue-system-with-postgresql-ruby/

LaLoka Labs

Multi-tenant job queue with PostgreSQL

- SKIP LOCKED to the rescue

Implementation Approach

- Listen/Notify
- Endless loop

Libraries

- Peewee (ORM)
- Pebble (multiprocessing)
- Click
- logzero

Python Implementation

- Task Model
- add_task()
- get_next_task()
- update_task()
- do_task()
- process_task()
- task_done()
- workers()

Task Model

```
class Task(BaseModel):
    id = AutoField()
    name = CharField()
    username = CharField()
    func = CharField()
    status = CharField()
    args = BinaryJSONField(default="{}")
    kwargs = BinaryJSONField(default="{}")
    start_time = DateTimeField(null=True)
    end_time = DateTimeField(null=True)
    retry_time = DateTimeField(null=True)
    created_at = DateTimeField()
    result = CharField(default="")
```

TaskSlot model

```
class TaskSlot(BaseModel):
    username = CharField()
    slots = IntegerField()
```

Get next task – SQL

LaLoka Labs

```sql
-- finds out how many jobs are running per queue, so that we know if it's full
  WITH running_jobs_per_queue AS (
    SELECT
      username,
      count(1) AS running_jobs from task
    WHERE (status = 'running' OR status = 'queued') -- running or queued
    AND created_at > NOW() - INTERVAL '6 HOURS' -- ignore jobs running past 6 hours ago
    group by 1
  ),
  -- find out queues that are full
  full_queues AS (
    select
      R.username
    from running_jobs_per_queue R
    left join taskslot Q ON R.username = Q.username
    where R.running_jobs >= CASE WHEN Q.slots IS NOT NULL THEN Q.slots ELSE 3 END
  )
  select *
  from jobs
  where status IN ('created', 'failed')
    and username NOT IN ( select username from full_queues )
    and retry_time <= now()
  order by id asc
  for update skip locked
  limit 1;
```

Get next task

```
def get_next_task():
    running_jobs_per_queue = (Task
            .select(Task.username, fn.Count(1).alias("running_jobs"))
            .where(Task.status.in_(["running", "queued"]))
            .where(Task.created_at > datetime.datetime.now() - datetime.timedelta(hours=6))
            .group_by(Task.username)
            .cte('running_jobs_per_queue', columns=("username", "running_jobs")))

    full_queues = (running_jobs_per_queue
            .select_from([running_jobs_per_queue.c.username])
            .join(TaskSlot, JOIN.LEFT_OUTER, on=(running_jobs_per_queue.c.username == TaskSlot.username))
            .where(running_jobs_per_queue.c.running_jobs >= Case(None, [((TaskSlot.slots != None), TaskSlot.slots
)], 6))
            .cte("full_queues", columns=("username",)))

    query = (Task
            .select()
            .where(Task.status.in_(["created", "failed"]))
            .where(Task.username.not_in(full_queues.select(full_queues.c.username)))
            .where(Task.retry_time <= fn.Now())
            .order_by(Task.id)
            .for_update("for update skip locked")
            .limit(1)
            .with_cte(running_jobs_per_queue, full_queues))
    return query.get()
```

LaLoka Labs

Add task

```
def add_task(username, func, *args, **kwargs):
    name = kwargs.pop("name", generate_id())
    task = Task(name=name, username=username, func=func)
    task.args = args
    task.kwargs = kwargs
    now = datetime.datetime.utcnow()
    task.created_at = now
    task.retry_time = now
    task.status = kwargs.pop("status", "created")
    task.save()
    return task
```

Worker

```python
@cli.command()
@click.option("--num", default=1, help="Number of workers")
def workers(num=1):
    with ProcessPool(max_workers=num, max_tasks=10) as pool:
        try:
            logger.info("Running")
            process_task(pool)
        except KeyboardInterrupt:
            logger.info("Exiting ...")
            pool.close()
            pool.join()
        except Exception as e:
            logger.error("ERROR:", e)
            time.sleep(5)
            sys.exit(1)
```

LaLoka Labs

Process task

```
def process_task(pool):
    while True:
        with db.atomic() as transaction:
            try:
                task = get_next_task()
            except Exception as e:
                #print("ERROR: ", e)
                time.sleep(3)
                continue
            else:
                res = pool.schedule(do_task_runner, (task,), timeout=20)
                print(task)
                affected = update_task(task, current_status=task.status, status="running")
                logger.info(f"Task {task.name} scheduled {affected}")
                def _task_done_wrapper(future, task=task):
                    task_done(future, task)
                res.add_done_callback(_task_done_wrapper)
```

LaLoka Labs

Stats

```
SELECT username,
    sum(case when status = 'created' then 1 else 0 end) as created,
    sum(case when status = 'running' then 1 else 0 end) as running,
    sum(case when status = 'success' then 1 else 0 end) as success,
    sum(case when status = 'failed' then 1 else 0 end) as failed,
    count(*) as total,
    max(age(start_time, created_at)) as max_delay
    from task
    where created_at > now() - interval '2 day'
    group by username;
```

LaLoka Labs

Full code

https://github.com/lalokalabs/pgsq/

LaLoka Labs

Demo?

LaLoka Labs

Questions?

What's next?

LaLoka Labs

A peek into Oban ...

- Batch support
- Queue control
- Periodic job
- Dashboard?

LaLoka Labs

Thank you!

kamal@lalokalabs.co
twitter.com/k4ml

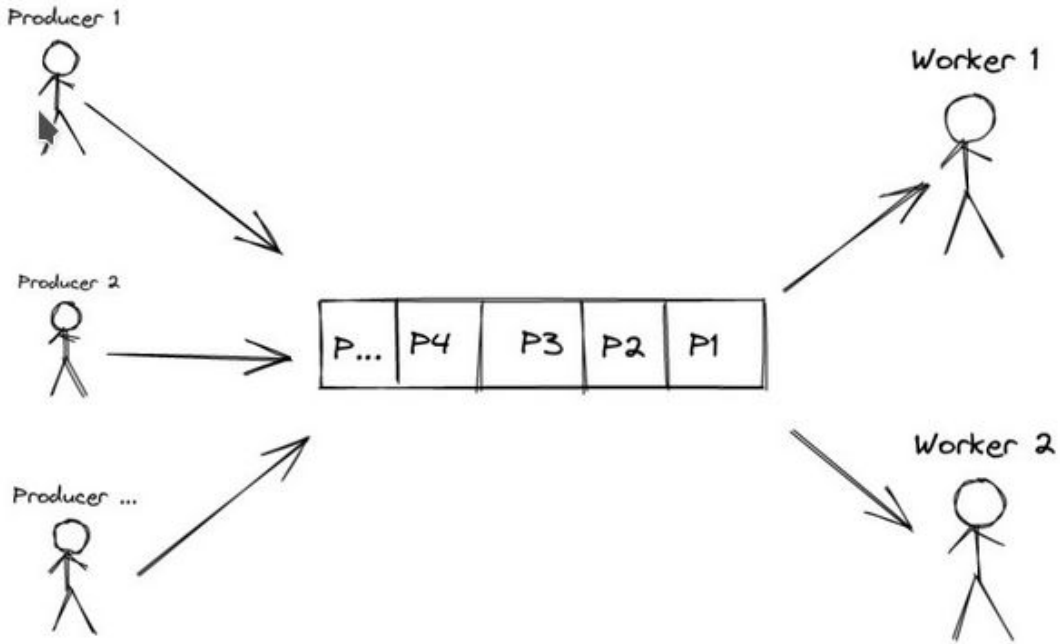
LaLoka Labs

What is task queue?

A way to store list of tasks to be done and execute it later.

LaLoka Labs

Basic Queue

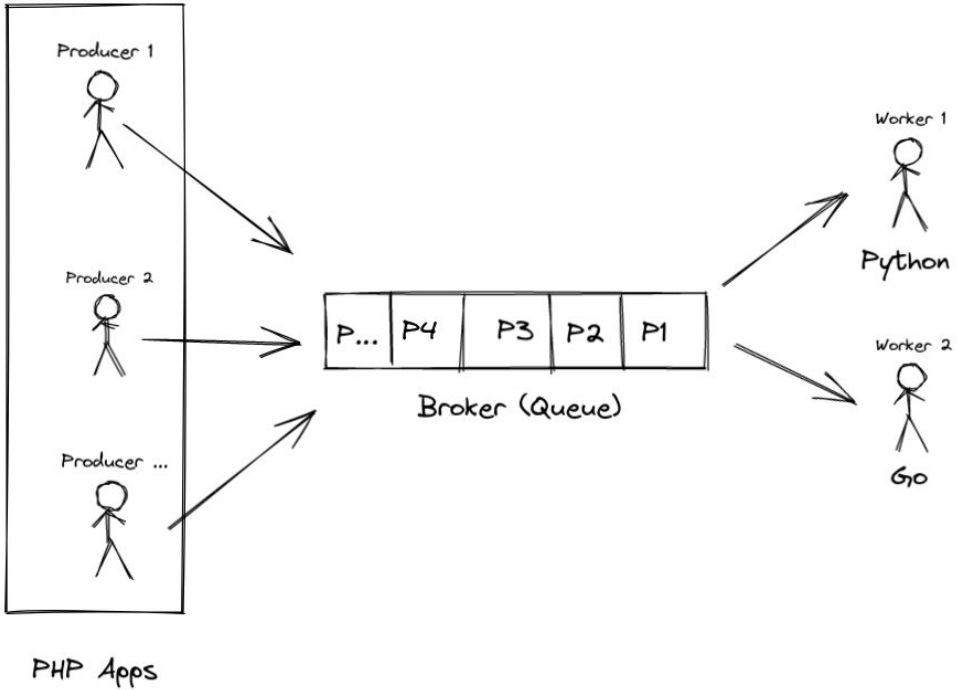


Why we might want to use a task queue?

- Work that can be done asynchronously
- Distributed systems context where work should be done “somewhere else”
- Gives you architectural flexibility

LaLoka Labs

Architectural flexibility



LaLoka Labs

Example use cases

- Sending email
- Processing videos after upload
- Generating backup - Google Takeout, Facebook backup etc
- Replacing cron jobs
- Sending notifications

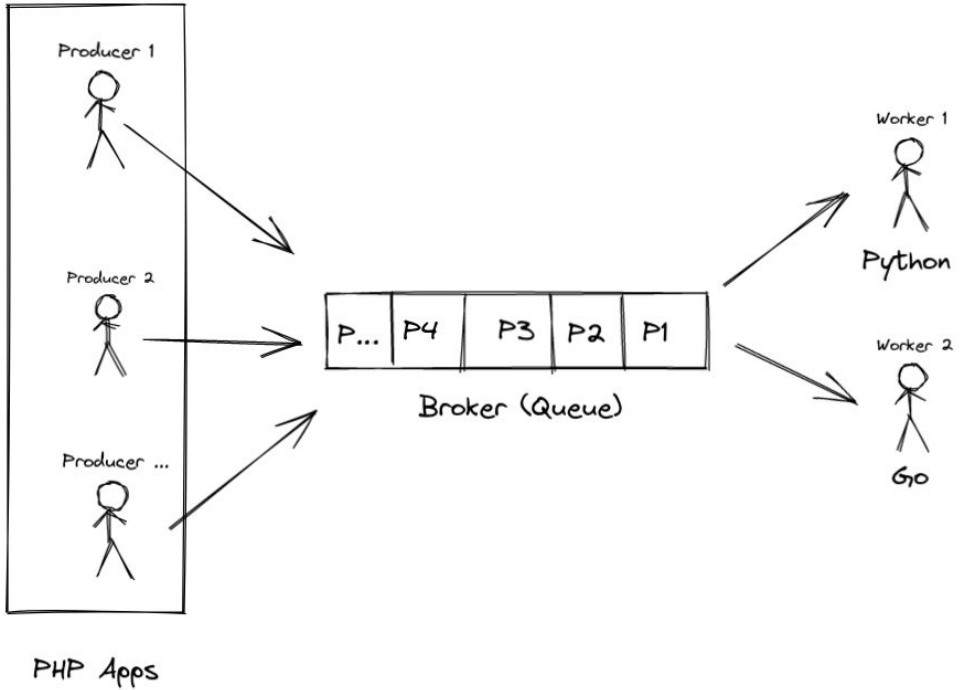
Common open source task queue

- Celery (python)
- django-q
- RQ (redis queue) (python)
- Dramatic (python)
- Resque (ruby)
- Sidekiq (ruby)
- Laravel Queue (PHP)
- Gearman (C/C++)
- Oban (Elixir)

Broker

A message broker (also known as an integration broker or interface engine) is an intermediary computer program module that translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver.

Broker


Producer 1
Producer 2
Producer ...
PHP Apps
P...
P4
P3
P2
P1
Broker (Queue)
Worker 1
Python
Worker 2
Go

Broker

HTTP / AMQP / MQTT

Broker

- RabbitMQ
- ActiveMQ
- QPID
- SQS
- ZeroMQ
- Redis
- NSQ
- PostgreSQL?