Tasks:
you gotta know how to run them,
you gotta know how to safe them.

Filipe Ximenes
@xima

@xima

vinta.com.br/playbook

FLOSS

Django React boilerplate

https://github.com/vintasoftware/django-react-boilerplate

Django Role Permissions

https://github.com/vintasoftware/django-role-permissions

Tapioca

https://github.com/vintasoftware/tapioca-wrapper

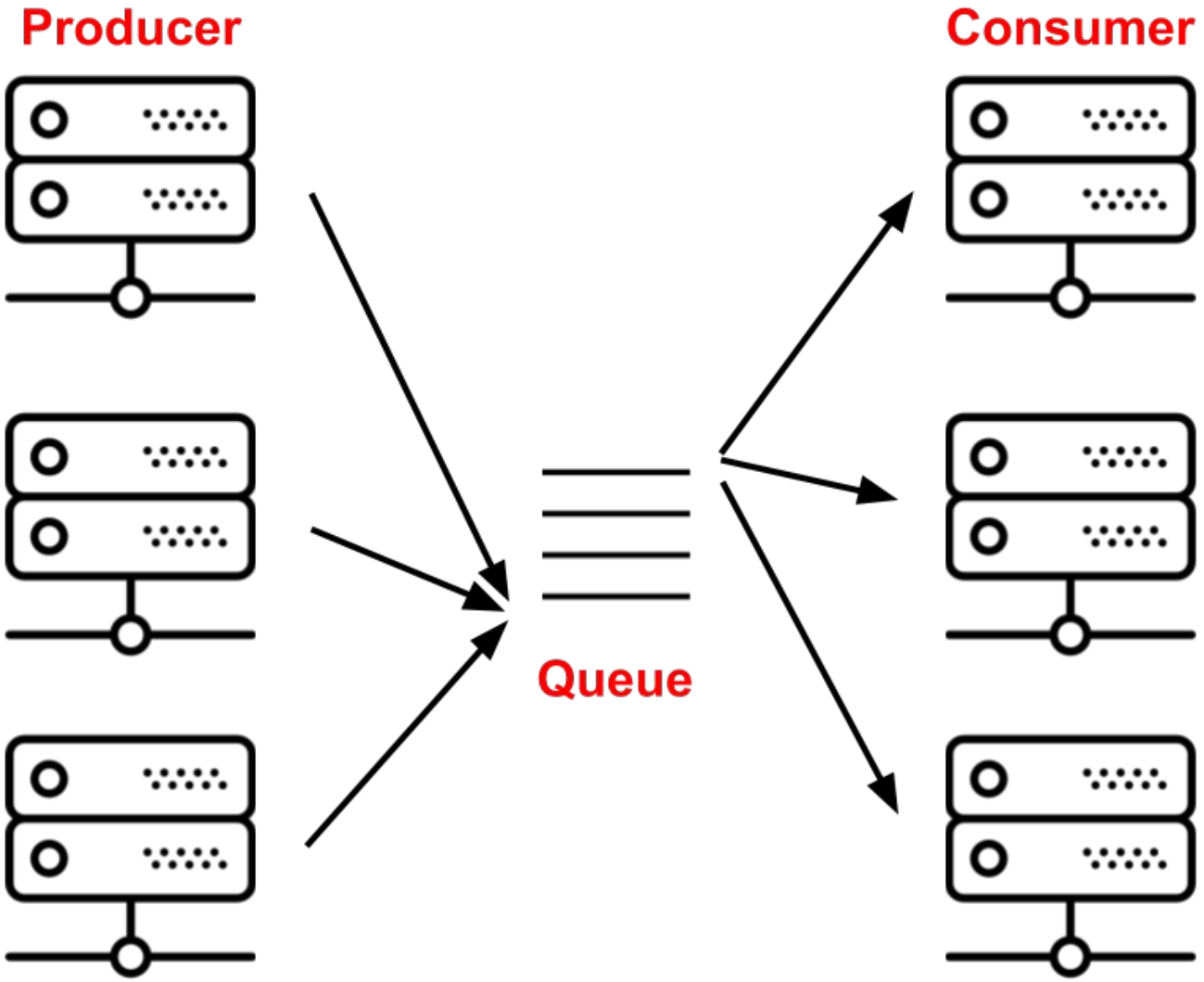
Context

we want to answer quickly to the user

What async tasks can be used for?

- Delegate long lasting jobs;
- Execute remote API calls;
- Prepare and cache values;
- Spread bulk database insertions over time;
- Execute recurrent jobs;

Producer & Consumer



dcramer / taskmaster

Django Q

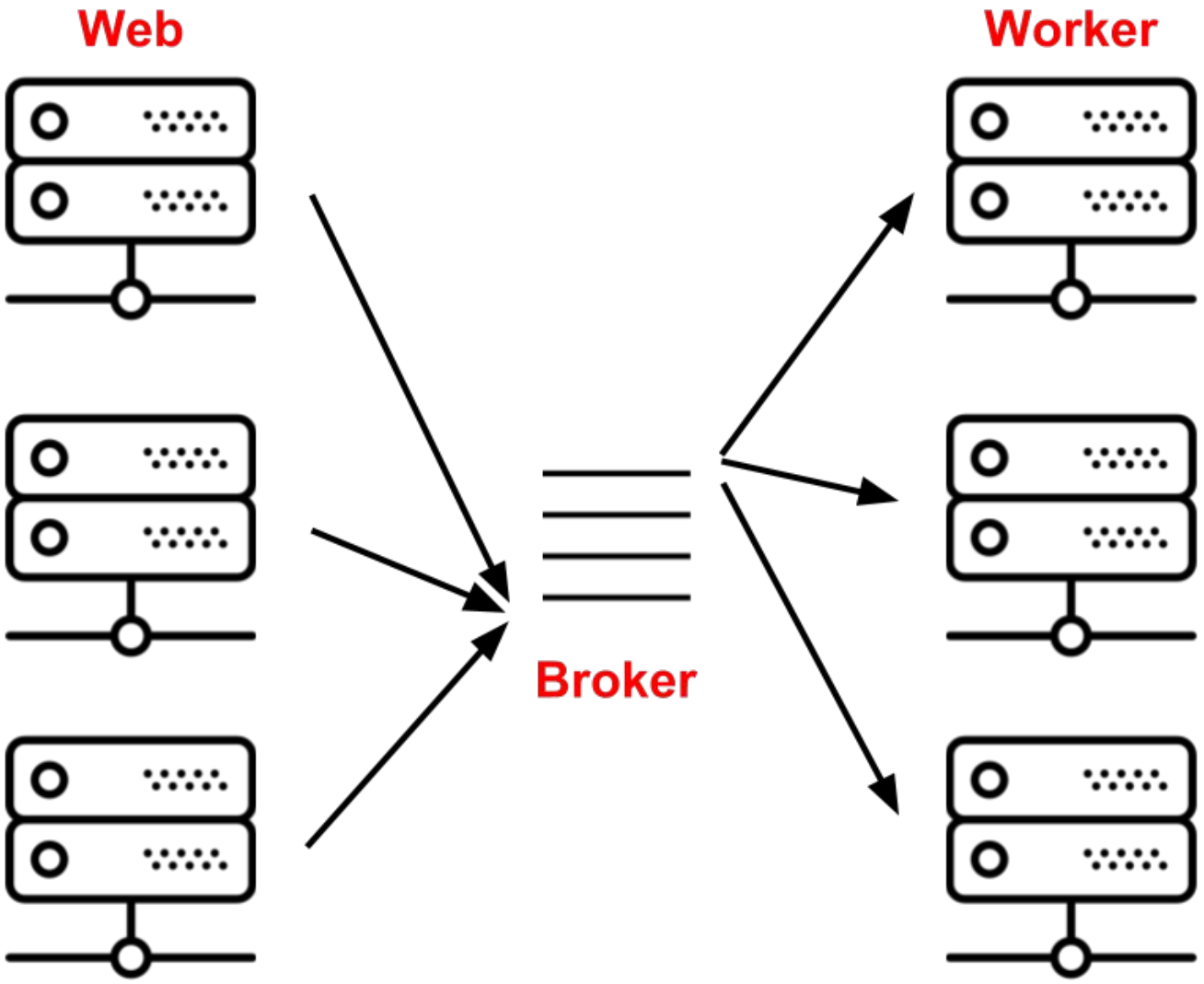
ui / django-rq

huey

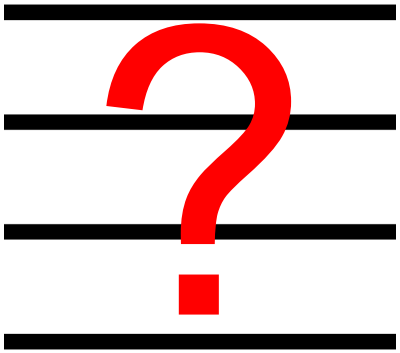
pricingassistant / mrq

Celery

Naming components



Broker




PostgreSQL


MySQL


Cassandra

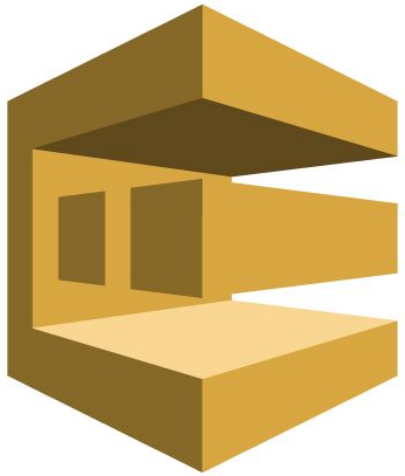

mongoDB

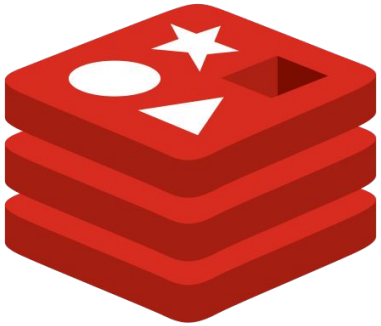

RabbitMQ


redis

Celery Compatibility

RabbitMQ





redis

How it works

```
from celery import
Celery

app = Celery(...)

@app.task
def add(a, b):
    return a + b
```

```
from tasks import add

r = add.delay(4,
5).get()
print(r)  # 9
```

With Django

```
@app.task
def update_attendees(event, n):
    event.attendees_number = n
    event.save()

-----------------------------------------

event = Event.objects.get(name='DjangoCon')
update_attendees.delay(event, 9001)
```

Don't pass complex objects as task parameter

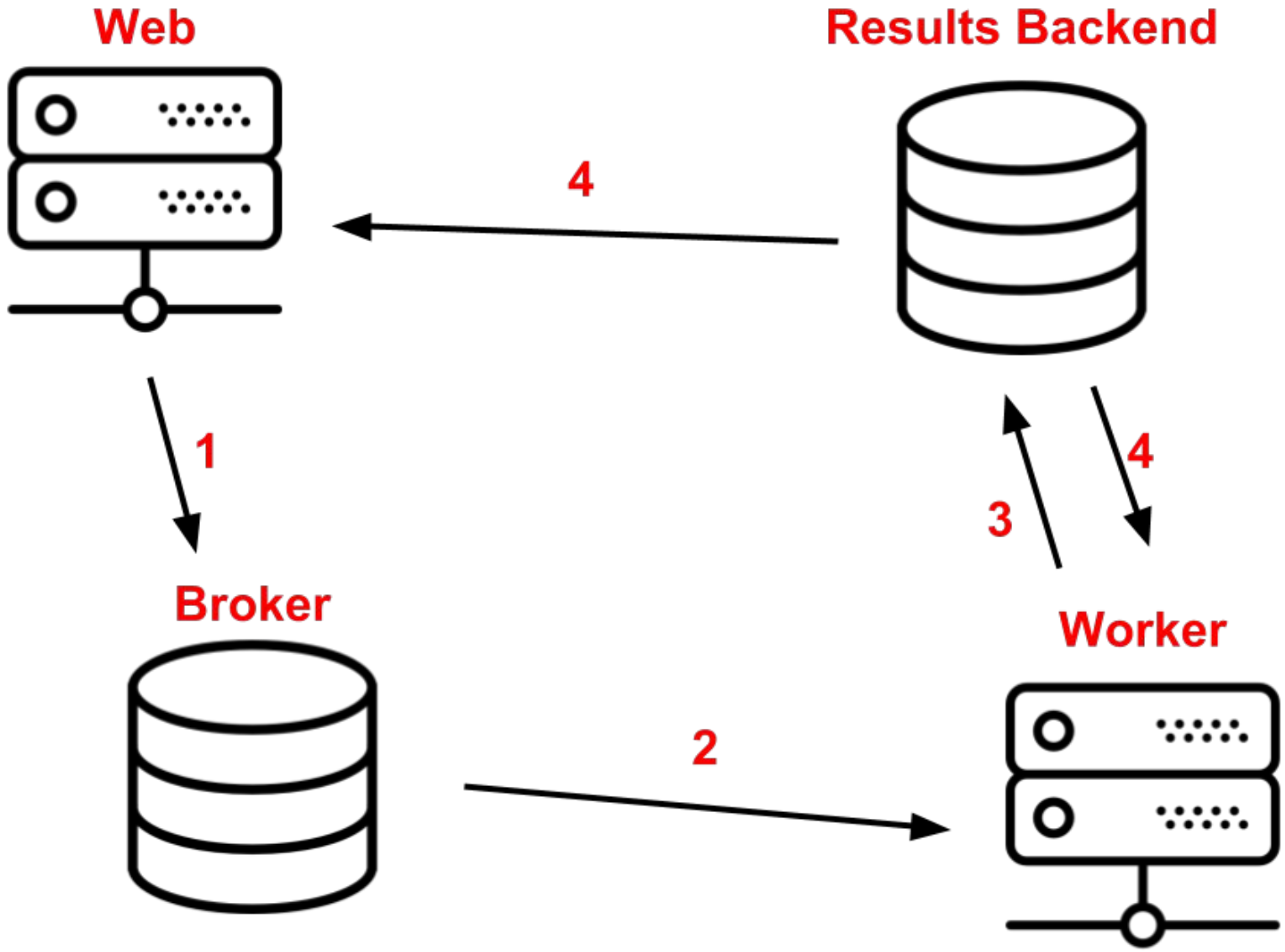
With Django

```
@app.task
def update_attendees(event_id, n):
    e = Event.objects.get(id=id_event)
    e.attendees_number = n
    e.save()

----------------------------------------------

e = Event.objects.get(name='DjangoCon')
update_attendees.delay(e.id, 9001)
```

We missed something

```
result = add.delay(4, 5).get()
```

How do we 'get' the result?



Error Handling

Idempotency

"... is the property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application."

- ***wikipedia***

Idempotent Operations

$$3 * 0 = 0 * 0 = 0 * 0 = 0 \ldots.$$

$$3 * 1 = 3 * 1 = 3 * 1 = 3 \ldots.$$

Idempotent Operations

```
GET /events/12/
```

```
POST /events/
{
    "name": "PyCon"
}
```

```
PUT /events/12/
{
    "name": "PyCon"
}
```

```
DELETE /events/12/
```

Idempotent Operations

GET /events/12/

POST /events/

```
{
    "name": "PyCon"
}
```

PUT /events/12/

```
{
    "name": "PyCon"
}
```

DELETE /events/12/

Atomicity

"... is an indivisible and irreducible series of database operations such that either all occur, or nothing occurs."

- ***wikipedia***

How to be Atomic?

VS.

```
@app.task
def update_data():
    user.status = 'updated'
    user.save()
    r = facebook_request()
    user.name = r.name
    user.save()
```

```
@app.task
def update_data():
    r = facebook_request()
    if r.status != 200:
        return
    user.name = r.name
    user.status = 'updated'
    user.save()
```

How to be Atomic?

VS.

```python
@app.task
def newsletter():
    users = all_users()
    for u in users:
        send(email, 'newsletter')
```

```python
@app.task
def send_newsletter(email):
    send(email, 'newsletter')

@app.task
def newsletter():
    users = all_users()
    for u in users:
        send_newsletter.delay(u.email)
```

Failed? Try again!

```
from tapioca.exceptions import TapiocaException
from tapioca_facebook import Facebook

@app.task(bind=True, retry_limit=4
          default_retry_delay=10)
def likes_do_facebook(self):
    api = Facebook(access_token=ACCESS_TOKEN)
    try:
        api.user_likes(id='me').get()
    except TapiocaException as e:
        self.retry(exc=e)
```

Backoff!

```
def exponential_backoff(task_self):
    minutes = task_self.default_retry_delay / 60
    rand = random.uniform(minutes, minutes * 1.3)
    return int(rand ** task_self.request.retries) * 60

self.retry(exc=e,
           countdown=exponential_backoff(self))
```

Auto Retry

```
from tapioca.exceptions import TapiocaException
from tapioca_facebook import Facebook

@app.task(autoretry_for=(TapiocaException,))
def likes_do_facebook(self):
    api = Facebook(access_token=ACCESS_TOKEN)
    api.user_likes(id='me').get()
```

Add options for exponential backoff with task autoretry #4101

Merged **georgepsarakis** merged 16 commits into `celery:master` from `singingwolfboy:exponential-backoff` 20 days

Conversation 26 Commits 16 Files changed 5

singingwolfboy commented on Jun 21 • edited Contributor

Celery's task autoretry support is very handy, but it's missing an important feature: exponential backoff. When you're writing a large Python application that has the potential to overwhelm another service it depends on, this is an intelligent, respectful way to allow the service to process requests as fast as it can.

I've added documentation for this code, but I'm not sure what is the best way to write automated tests. I have one test that is failing, and I don't understand why -- I don't even know if it's the right sort of test to write. Can someone help me out with writing tests for this change?

Task time limit

```
from celery.exceptions import SoftTimeLimitExceeded

@app.task(task_time_limit=60, task_soft_time_limit=50)
def mytask():
    try:
        possibly_long_task()
    except SoftTimeLimitExceeded:
        recover()
```

acks_late

Notify & Execute

vs.

Execute & Notify

Monitoring

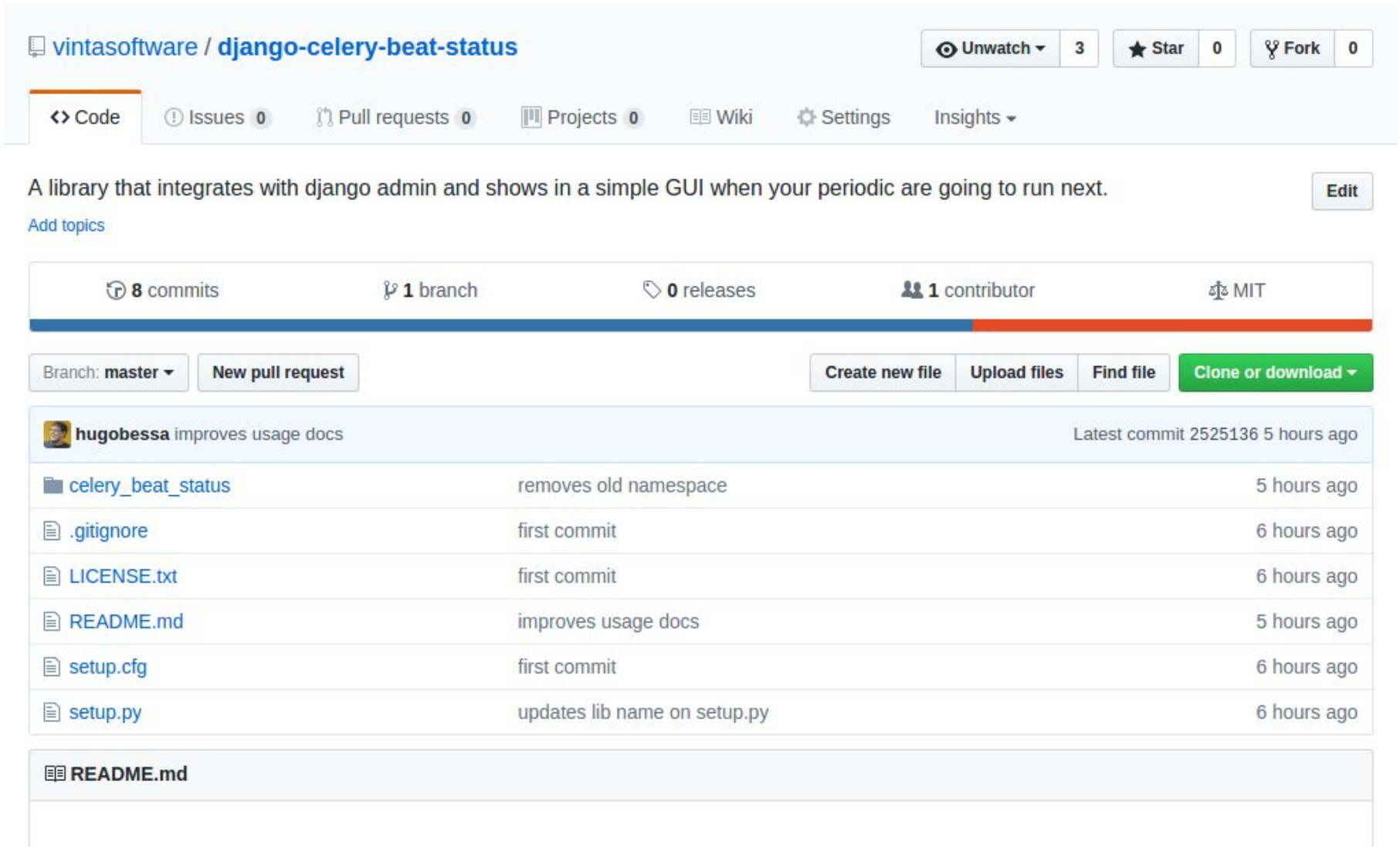
Logging

```
from celery.utils.log import get_task_logger

logger = get_task_logger(__name__)

@app.task
def add(a, b):
    logger.info('Adds {0} + {1}'.format(a, b))
    return a + b
```

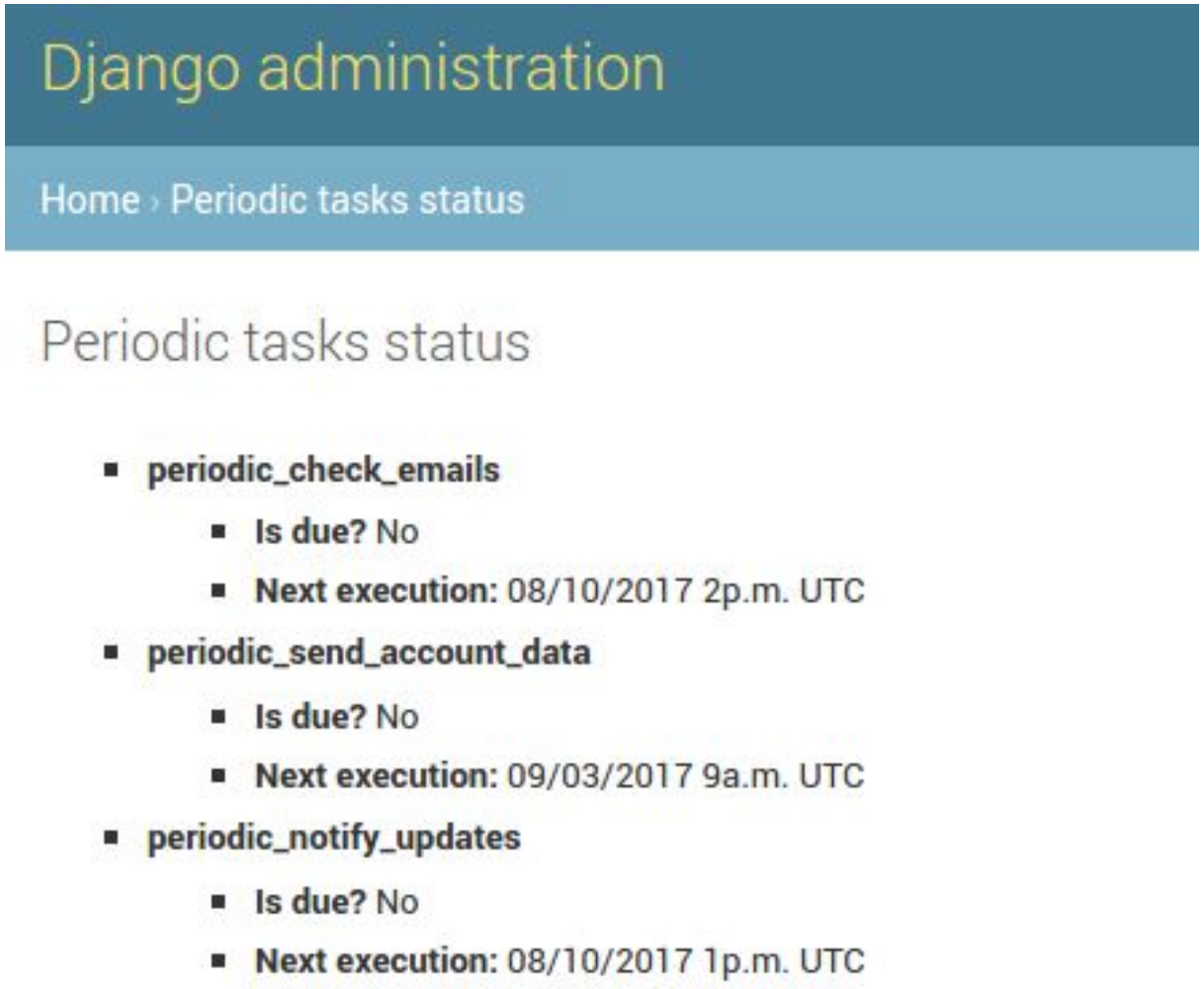
Scream if it fails

Celery Flower

vintasoftware/django-celery-beat-status



vintasoftware/django-celery-beat-status



Tests and Debugging

task_always_eager

```
class Task(object):
    # ...

    def apply_assync():
        # ...
        if app.conf.task_always_eager:
            return self.apply(...)
        # ...

    # ...
```

Rdb: pdb to Celery

```python
from celery.contrib import rdb

@app.task
def add_debug(a, b):
    rdb.set_trace()
    return a + b
```

```
Remote Debugger:6903: Ready to connect: telnet 127.0.0.1 6903

Type `exit` in session to continue.

Remote Debugger:6903: Waiting for client...

Remote Debugger:6903: Now in session with 127.0.0.1:65136.
```

```
filipeximenes@macpro 05_2017_tasks_assincronas $ telnet localhost 6903
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
> /Users/filipeximenes/Dropbox/documents/palestras/05_2017_tasks_assincronas/tasks.py(43)soma_debug()
-> return a + b
(Pdb) a
a = 1
b = 2
(Pdb)
```

Review

- Don't use complex objects in task parameters
- Write idempotent and atomic tasks
- Backoff when you retry
- Make extensive use of monitoring tools
- Get notified if something fails
- Use `task_always_eager` for testing and debugging

celerytaskschecklist.com

Obrigado!

http://bit.ly/vinta2017

celerytaskschecklist.com

Newsletter:
vinta.com.br/blog/

twitter.com/@xima

ximenes@vinta.com.br

github.com/filipeximenes

Extending the base class

```python
from celery import Task

class CustomTask(Task):

    def after_return(self, status, retval, task_id,
                     args, kwargs, einfo):
        # ...

    def on_failure(self, exc, task_id, args, kwargs, einfo):
        # ...

    def on_retry(self, exc, task_id, args, kwargs, einfo):
        # ...

    def on_success(self, retval, task_id, args, kwargs):
        # ...
```

```python
@app.task(base=CustomTask)
def my_task():
    # ...
```

```python
app.Task = CustomTask
```

Canvas

Signatures

```
from celery import signature

signature('tasks.add', args=(2, 2), countdown=10)
add.signature((2, 2), countdown=10)
add.s(2, 2)

add.s(2, 2)()
add.delay(2, 2)

add.signature((2, 2), countdown=1).apply_async()

partial = add.s(2)
partial.delay(4)
```

Callbacks

```
r = add.apply_async((1, 2), link=add.s(3))

r.get()  # 3
r.children[0].result  # 6
```

Chains, Groups & Chords

```
c = chain(add.s(4, 4), mul.s(8), mul.s(10))
res = c().get()  # 640

g = group(add.s(2, 2), add.s(4, 4))
res = g().get()  # [4, 8]

callback = tsum.s()
header = [add.s(i, i) for i in range(100)]
result = chord(header)(callback)
result.get() # 9900
```

Map, Startmap & Chunks

```
x = xsum.map([range(10), range(100)]).delay()
x.get()  # [45, 4950]

p = zip(range(10), range(10))
p  # [(0, 0), (1, 1), (2, 2), ...]
s = add.starmap(p).delay()
s.get()  # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

p2 = zip(range(100), range(100))
c = add.chunks(p2, 10)
c().get()
# [[0, 2, 4, 6, 8, 10, 12, 14, 16, 18],
#  [20, 22, 24, 26, 28, 30, 32, 34, 36, 38],
#  ...
```

Multiple Queues

```
app.conf.task_routes = {
    'app.tasks.add: {'queue': 'add'},
    'app.tasks.subtract: {'queue': 'subtract'}
}
```

```
$ celery -A proj worker -Q add, celery
$ celery -A proj worker -Q subtract
```

```
sub.apply_async((2, 1), queue='subtract')
```

Task dependency

Calling a task from another

```
@app.task
def add(a, b):
    return a + b

@app.task
def add_all(*args):
    result = 0
    for i in args:
        result = add.delay(result, i).get()
    return result

add_all.delay(1, 2, 3, 4)
```

Never wait inside a task

Decouple as much as possible