Beginner's guide to Celery

Mehdi Ben Jaafar

Code Sheriff @Bitcraft

Nobody likes waiting

...

And that's why you should use Celery!

What is Celery???

...

A task queue based on distributed message passing.

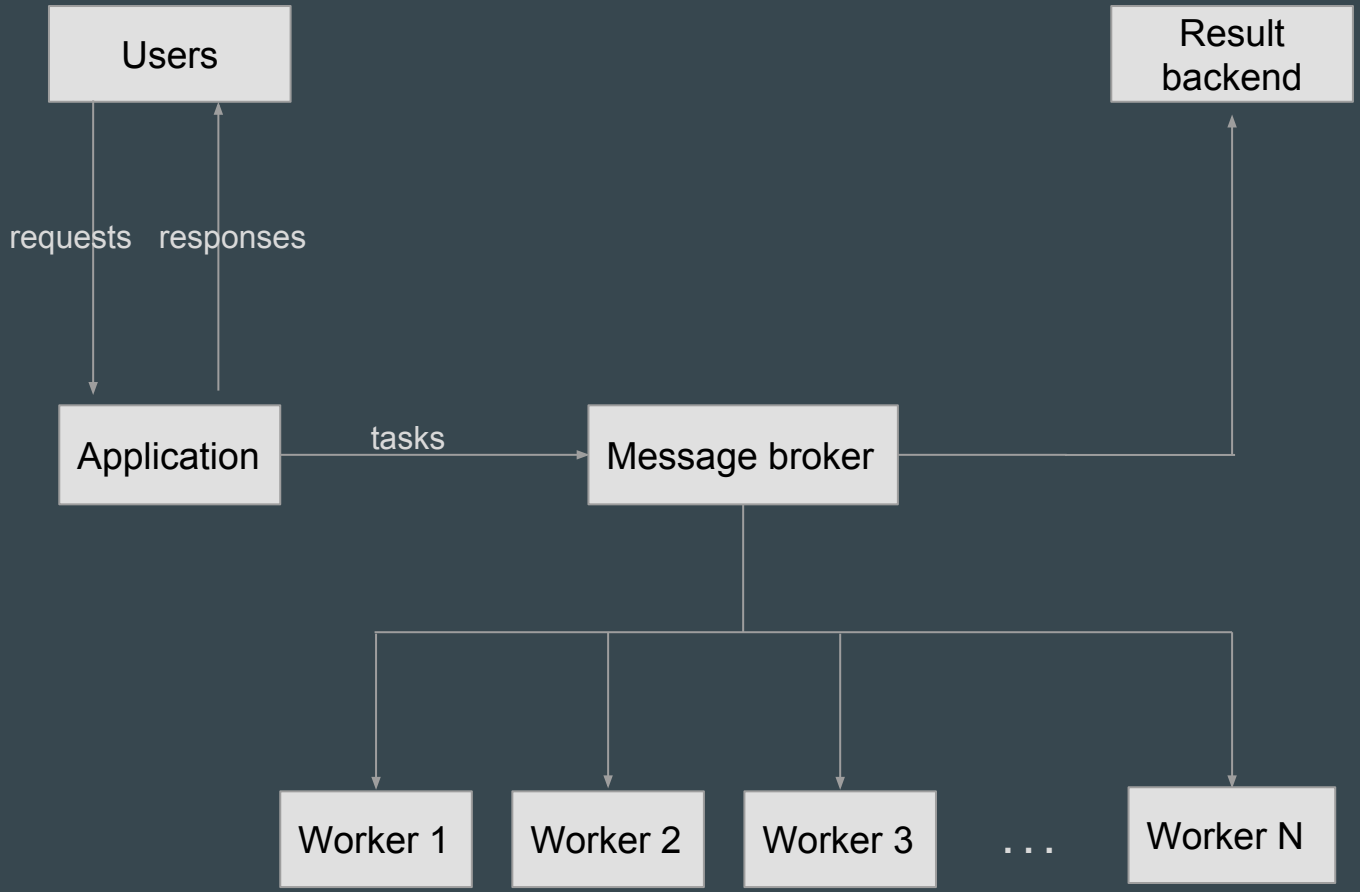
Can you describe it a bit more?

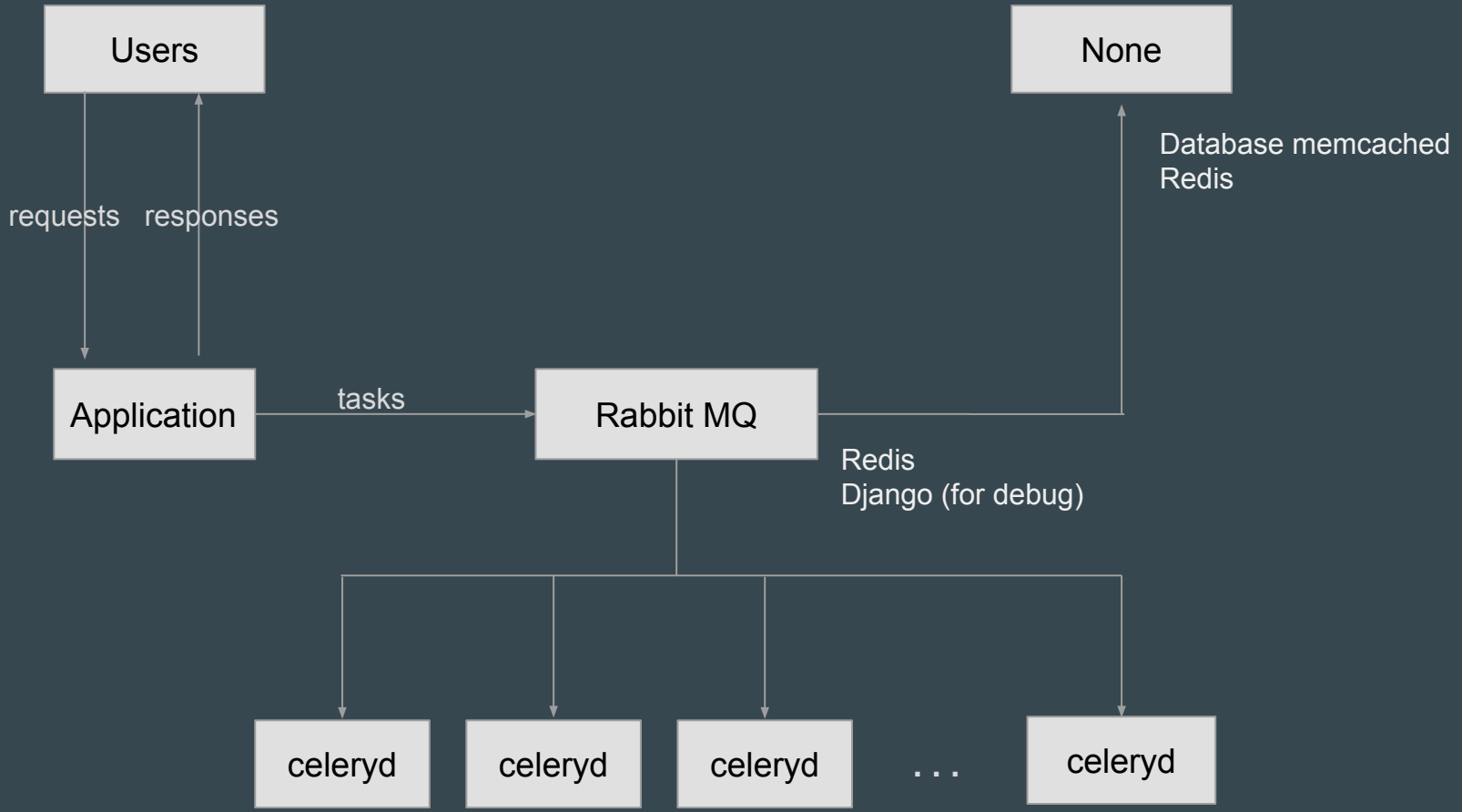
Celery is :

- Asynchronous
- Concurrent
- Distributed
- Scalable

Used for:

- Background tasks
  - Calculating points/badges
  - Running spam filters
- Escape the Request/Response Cycle:
  - Sending emails asynchronously
- Periodic Jobs


Users
Result backend
requests
responses
Application
tasks
Message broker
Worker 1
Worker 2
Worker 3
. . .
Worker N


Users
None
requests
responses
Database memcached
Redis
Application
tasks
Rabbit MQ
Redis
Django (for debug)
celeryd
celeryd
celeryd
. . .
celeryd

Architecture Recap

- A Producer
- A Message broker: Stores and distributes the tasks
- A (lot of) Worker(s): The slave to execute the task
- A Result Backend (optional): stores the task result if any

Install and Configure

1. Install the Message Queue (in our case RabbitMQ)
2. Using pip install **celery** and **django-celery**
3. Add '**djcelery**' to **INSTALLED_APPS** in settings
4. The “tricky part”:
   a. create a celery.py file to define a celery app
   b. import the celery.py in the __init__.py of the django app

   more details: http://docs.celeryproject.org/en/latest/django/first-steps-with-django.html

5. Define your tasks in a tasks.py file
6. Run the tasks for profit.

What is a Task?

“It is a class that can be created out of any **callable**(function).
It defines how it can be called, and executed.”

In short: It is like function but more...

What can you do with tasks that you can’t do with a function?

- **Task can be run by a worker**
- Tasks can be retried

Defining a simple Task

Defining a task:

```
from celery import shared_task

@shared_task
def add(a, b):#a simple function just add the task decorator
    print a+b
```

Running a task:

```
from .tasks import add

add.delay(2, 3) #call a task by using delay
```

Defining a periodic Task

Instead of manually running the task you can schedule it!

```
from celery.decorators import periodic_task
from celery.task.schedules import crontab

@periodic_task(run_every=(crontab(minute="*/5")))
def say_hello():
    print “hello!”
```

Running Celery on Production

We need a solution that:

- Allows us to run Celery on the background
- Make sure Celery is always up even after a server restart

Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

why did we choose Supervisor as our process controller?

- it’s simple: one conf file is all you need to add a process
- it’s centralized: one place to define all processes
- it’s not limited to Celery


Malconfigured Celery
Server Memory

Solving the case of :
Celery and the murder of bitcraft.tests server

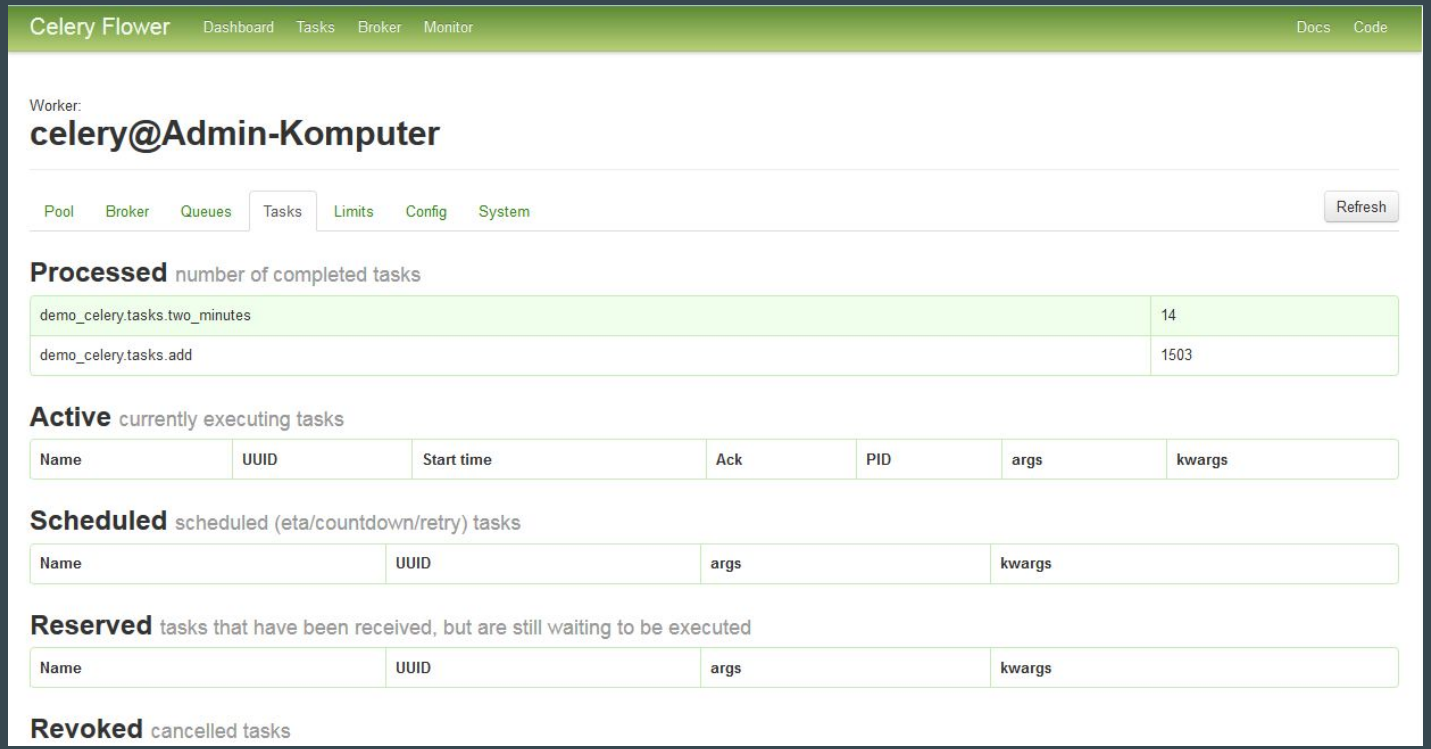
For conducting an investigation you need tools

Flower - Celery: a web based tool for monitoring and administrating Celery clusters

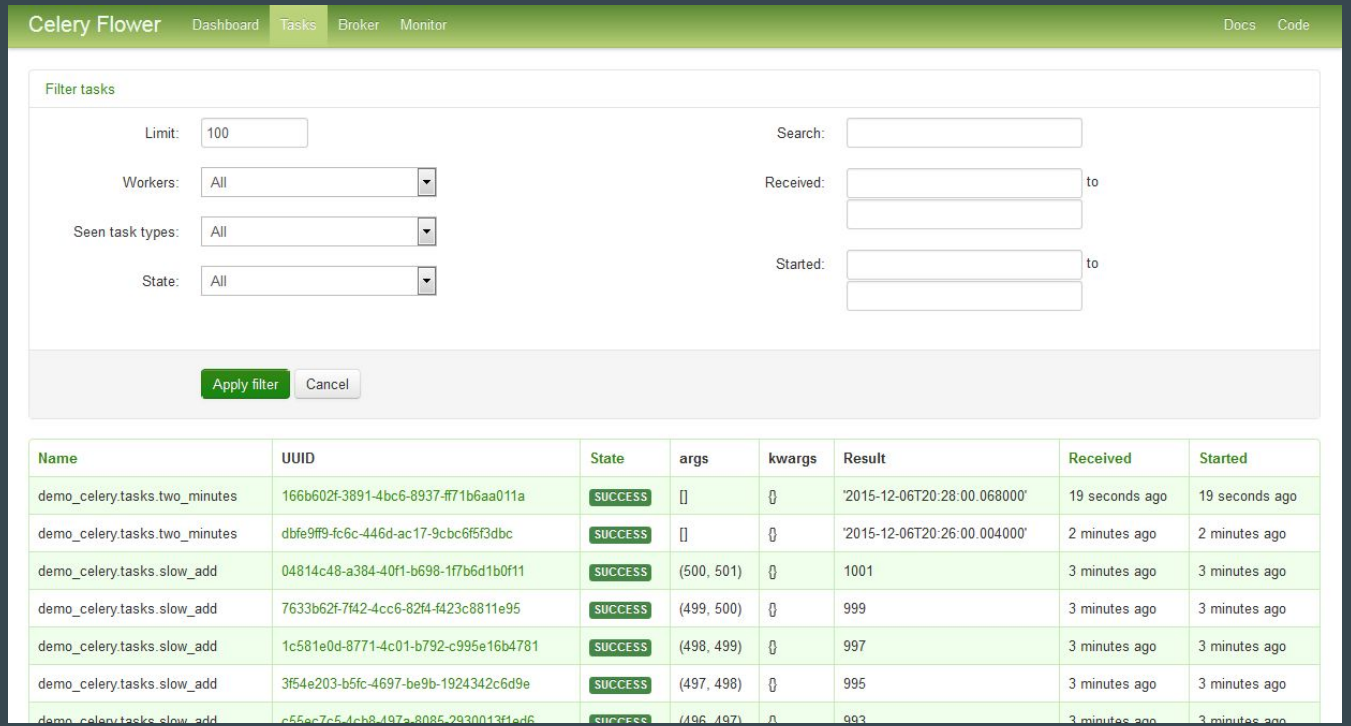
It offers:

- **Real-time monitoring**
- Remote Control
- Broker monitoring
- HTTP API

Queue status



Task History



Monitoring tasks

Periodic Tasks Eternal Hell

Don’t schedule more task than you can execute !!!

Example:

1. Have a Task scheduled every 5 min.
2. The Task takes 10 min to execute
3. execution time > scheduling interval = Dead Server (R.I.P)

Possible Solution:

- Add an expires parameter to task.

Limit the calls to external API's

- Avoid building tasks that rely on multiple external API's.
- Minimise uses of external API's by:
  - limiting the number of executions of a request (e.g. setting a limit of 24h between tries)
  - caching results

Try not to pass entire objects as parameters

Instead of:

```
some_task.delay(user) #BAD
```

Use:

```
some_task.delay(user.pk) #GOOD
```

And define the task as:

```
@shared_task
def some_task(user_id):
  user = User.objects.get(id=user_id)
```

Don’t be afraid to tweak the settings

Celery have a multitude of settings. While for testing it works fine by keeping most of them to default it’s not really optimal.

- **CELERY_IGNORE_RESULT** - Most of the times the task return nothing of interest, save time and space and don’t save the results
- **CELERYD_PREFETCH_MULTIPLIER** - should be high if the queue will have a lot of small tasks and low if the queue have few big tasks
- if you have a **high concurrency** (number of processes on workers) be sure that it will not drain your limit of **database connections**

Want to know more?

Ask me some questions: m.jaafar@bitcraft.com.pl

Or Check the docs:

- Celery documentation: http://docs.celeryproject.org
- Supervisor documentation: http://supervisord.org
- Flower documentation: http://flower.readthedocs.org


& Drink More Coffee
Code More