Celery

Celery - návod na používanie

Celery je nástroj pre (a)synchrónne zaraďovanie a vykonávanie taskov, ktorých riadenie je založené na posielaní správ. Je by default integrovateľná s Flaskom a má podporu pre Django.

1. Inštalácia:
   - pip install Celery

2. Setup:
   - pre používanie Celery je nutné nainštalovať message transport tool - broker. Na výber sú 4 možnosti, pričom najlepšie vlastnosti majú RabbitMQ a Redis. Najodporúčanejšie je používať práve RabbitMQ, ktorý sa inštaluje:
   - sudo apt-get install rabbitmq-server pre Linux
   - brew install rabbitmq pre Mac OS
   - stiahnutim instalatora pre Windows - http://www.rabbitmq.com/download.html

   Po nainstalovani bezi broker na pozadi a je ready prijimat spravy.
   V python programe je dostupny na adrese: 'amqp://localhost'

   - pre uchovavanie navratovych vysledkov taskov a informacii o jednotlivych taskoch je nutne definovat tzv. Celery backend. Rovnako je tu na výber viacero možností (ktoré sú built-in, ale je možné vytvoriť aj vlastný): SQLAlchemy/Django ORM, Memcached, Redis, RPC (RabbitMQ/AMQP) . Pri výbere backendu je potrebné sa dohodnúť, v akom formáte budú požadované návratové hodnoty volaní - RPC ich posiela ako správy, DB backendy ukladajú do DB. Jedná sa však o vec, ktorú iba stačí jednoducho definovať v programe (viď príklad appky nižšie, prípadne ďalšie info na tomto odkaze).

1. Vytvorenie Celery app
   1. Vytvoriť hlavnú riadiacu Celery aplikáciu app/celery.py:
   Tá obsahuje vytvorenie pripojenia na broker, načítanie modúl programu, ktoré používajú Celery pre riadenie a komunikaciu a prípadné nastavenie configu.
   Môže to vyzerat nejak takto (trochu prípomína Flask):

```
from celery import Celery

app = Celery('app',
       broker='amqp://',
       backend='amqp://',
       include=['app.tasks']

    )

if __name__ == '__main__':
  app.start()
```

Pre zmenu configu je možné pridať (do tohto istého súboru):

```
app.conf.update(
  result_expires=3600,
)
```

Vo všetkých skriptoch, kde budem chcieť využívať Celery, je potrebné importovať app (vytvorenú pomocou celery).

Na rovnakej úrovni si vytvoríme ďalší skript, tasks.py (alebo aj iný názov - tento skript bude obsahovať úlohy, ktoré sa majú vykonávať):

```
from .celery import app


@app.task
def add(x, y):
  return x + y


@app.task
def mul(x, y):
  return x * y
```

Úloha vykonateľná prostredníctvom Celery volania sa označuje dekorátorom @app.task. Úlohy sú volateľné prostredníctvom ich unikátnych mien v správach, ktoré sa posielajú v rámci Celery (viac podrobností tu). V našom prípade môžeme v rámci jedného tasku napríklad volať ľubovoľný crawler s parametrom začiatočných url.

4. Používanie Celery

V momente, ako máme zadefinované všetky tasky a vytvorenú Celery appku, môžeme appku spustiť.

```
celery -A priecinok_s_appkou worker -l info --app celery.py
```

Potrebné je byť v roote projektu, resp. v priečinku nad priečinkom, ktorý obsahuje python skripty s appkou a taskami.

Volanie samotných úloh v programoch vyzerá takto:

add.delay(2, 2),
Kde add je názov tasku, delay zabezpečí, že sa má asynchrónne task vykonať.
Alternatívou je volanie add.apply_async((2, 2)), ktoré v podstate vykoná to isté, avšak vďaka ďalších možných argumentom umožňuje nastaviť napríklad frontu, do ktorej sa zaradí task. Task je možné volať aj klasicky, keďže je to konceptuálne vždy metóda obalená dekoráotorom - v tom prípade sa ale Celery nepodiela na riadení (add(2, 2)).
Ďalšie info o volaní taskov je tu.

Volanie mi vráti informácie o tasku, vďaka ktorému viem sledovať stav jobu.

```
res = add.delay(2, 2)
res.state
```

Rovnako sa viem spýtať, či je už výsledok volania pripravený: res.ready(). V momente, kedy je výsledok ready, je zapísaný do definovaného backendu (a môžeme k nemu v rámci programu pristúpiť).
Najvýhodnejšou možnosťou je posielanie výsledkov do databázy (a teda backend ako databáza).