Celery

A Distributed Task Queue

Oskar Hollmann

User Technologies

26.10.2016

Motivation

- time demanding tasks are a pain in web apps
    - HTTP request can easily time out
    - it's not acceptable to block the client for too long
    - client may not care about the result
- either we return an URL to the client who polls it later to get the result
- or we push it through web sockets
- how can we achieve that?

Motivation

Node.js to the rescue!

Motivation

Node.js is cool, but...

- Why use Node.js and struggle with async code when async operations are seldom needed in a web app?
- Async code is not enough – we might need to distribute the tasks, run them periodically, . . .
- Node.js programmers are a rare commodity.
- What we actually need is **an asynchronous task queue**.
- Examples: RabbitMQ, JMS, Celery, . . .

Use Cases of Distributed Task Queues

1. Non-blocking task execution.
2. Task execution with failure recovery.
3. Concurrent task execution for single-threaded apps.
4. Distribute task to other machines.
5. Handle complex task workflows with dependencies.
6. Periodic tasks.

Task Queue in Web App

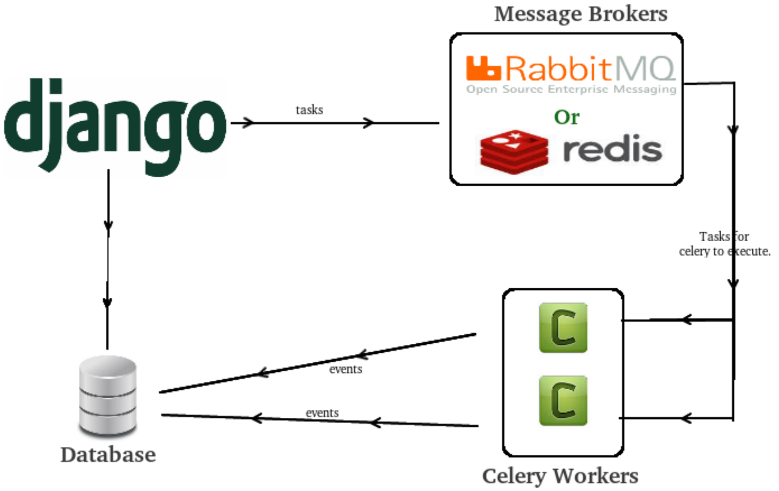

Figure: Source en.proft.me

Celery is

- distributed task queue written in Python
- bindings for: PHP, Ruby, NodeJS and more
- different message broker transports: Redis, RabbitMQ, MongoDB and more
- arbitrary number of queues and workers

Workers and Queues

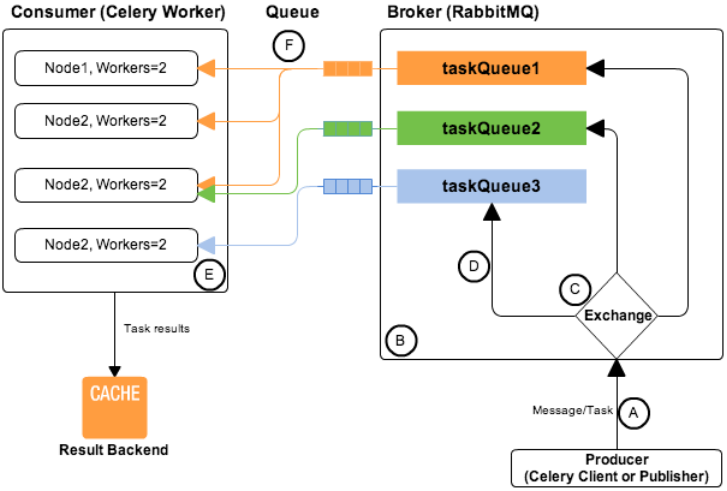

Figure: Source abhishek-tiwari.com

Minimal Example – Create Celery App

```python
from celery import Celery

app = Celery(
    'tasks',
    broker='redis://localhost:6379/1',
    backend='redis://'
)


# Decorator creates a Celery task from a regular function
@app.task
def add(x, y):
    return x + y
```

Minimal Example – Calling Tasks

```python
from celery_example import add

# Fire up the task
add.delay(1, 4)

# Or a more sophisticated way
res = add.apply_async(args=(2, 4), queue='celery')

res.status  # Get status of the task
res.get()  # Wait for the result
```

Problem Specification

- We have an online payment method where each order must go through non-trivial scoring process.
- Problems with synchronous code:
    - Scoring may take up tu a minute.
    - The computation is resource-heavy and must not affect processing of new orders.
    - To increase throughput of the app, different scoring tasks must be run concurrently.
    - Scoring cannot run in parallel for one customer.

Implementation in Celery

Task Orchestration