Продвинутое использование Celery

Александр Кошелев, Яндекс

Python Party, Киев, 4 октября 2014

Содержание

- Общее описание – что такое Celery
- Использование MongoDB и Redis в качестве брокера
- Несколько приложений Celery в одном проекте
- Эксклюзивный доступ к ресурсам
- Логирование ошибок
- Мониторинг Celery

Что такое Celery

Для пользователя

- "Distributed task queue"
- Асинхронные задачи
- Задачи по расписанию

Что такое Celery

Для пользователя

- *Задачи* ставятся в *очереди* клиентами
- Очереди хранит *брокер*
- Очереди разбираются *воркерами*
- Задачи возвращают *результаты*
- Воркеры создают *события*

Что такое Celery

Для разработчика

- Kombu + абстракция над "задачей"
- Kombu
  - "Messaging library"
  - Несколько транспортов для сообщений
  - AMQP или "virtual AMQP"

Что такое Celery

Для админа

- Воркеры (1..N)
  - Мастер-процесс (1)
  - Процессы выполняющие задачи (1..N)
- Воркер слушает очереди (1..N) у брокера
- Особый воркер - celery beat
  - Имеет расписание
  - Ставит задачи в очереди
  - Сохраняет своё состояние
- Брокер - транспорт между клиентом и воркерами

Альтернативные брокеры

- Нет дополнительной сущности
- Лучше подходят под задачу

Альтернативные брокеры

Virtual AMQP

- Эмуляция AMQP
- Есть не все возможности
- Больше кода, который может сломаться
- Потенциально меньшая эффективность

Альтернативные брокеры

Redis

- Работает через virtual AMQP
- Быстрый
- Встроенный LRU механизм

Альтернативные брокеры

MongoDB

- Работает через virtual AMQP
- Failover через Replica Set
- Экспериментальный статус
- Не работают события

Несколько приложений

Что такое "приложение"

- Набор настроек
- Зарегистрированные задачи
- Всегда есть хотя бы одно

Несколько приложений

Разные брокеры

- Разный профиль нагрузки
- Разный тип задач

Несколько приложений

Использование

```
from celery import Celery

app_global = Celery()
app_global.conf.update(BROKER_URL='mongodb://')

app_local = Celery()
app_local.conf.update(BROKER_URL='redis://')

app_global.set_current()
task_a.apply_async()

app_local.set_current()
task_b.apply_async()
```

Блокировки ресурсов

- Ограниченная пропускная способность
- Эксклюзивный доступ
- Запуск одной задачи за раз

Блокировки ресурсов

Интеграция

- Декоратор

```
@task
@locked('lock-name')
def my_task():
    ...
```

- Базовый класс

```
class MyTask(LockedTask):
    def locked_run(self):
        ...
```

- Внутри beat шедулера

Блокировки ресурсов

Инструмент

- ZooKeeper (kazoo)
- etcd
- ...
- MongoDB

Блокировки ресурсов

Semaphore

- Нужно ограничить нагрузку на компонент
- У компонента есть емкость - кол-во тикетов
- Если тикеты кончились - обратно в очередь

Блокировки ресурсов
Semaphore

```
zk = KazooClient()

def locked(name, capacity):
    def _decorator(func):
        def _wrapper(self, *args, **kwargs):
            semaphore = zk.Semaphore('/path', name, capacity)

            if semaphore.acquire(blocking=False):
                try:
                    return func(self, *args, **kwargs)
                finally:
                    semaphore.release()
            else:
                raise self.retry(coundown=5)

        return _wrapper
    return _decorator
```

Блокировки ресурсов

Semaphore

```
@task(bind=True)
@locked('save-to-storage-task', 10)
def save_to_storage(self, data):
    storage.save(data)
```

Блокировки ресурсов
Mutex

- Эксклюзивный доступ
- Если лок у кого-то другого - обратно в очередь

Блокировки ресурсов
Mutex

```
from celery import Task

class LockedTask(Task):
    lock_name = None
    retry_on_lock = True

    def run(self, *args, **kwargs):
        lock = zk.Lock('/path', self.lock_name)

        if lock.acquired(blocking=False):
            try:
                return self.locked_run(*args, **kwargs)
            finally:
                lock.release()
        elif self.retry_on_lock:
            raise self.rerty(countdown=5)

    def locked_run(self):
        raise NotImplementedError
```

Блокировки ресурсов
Mutex

```
class ImportDataTask(LockedTask):
    name = 'import-data'
    retry_on_lock = False

    def locked_run(self, url):
        response = requests.get(url)

        ...
```

Логирование ошибок

Варианты

- Воркер пишет ошибки в консоль или файл
- Настройка логинга в обработчике сигнала
- Настройка логинга в проекте

Логирование ошибок

Настройка

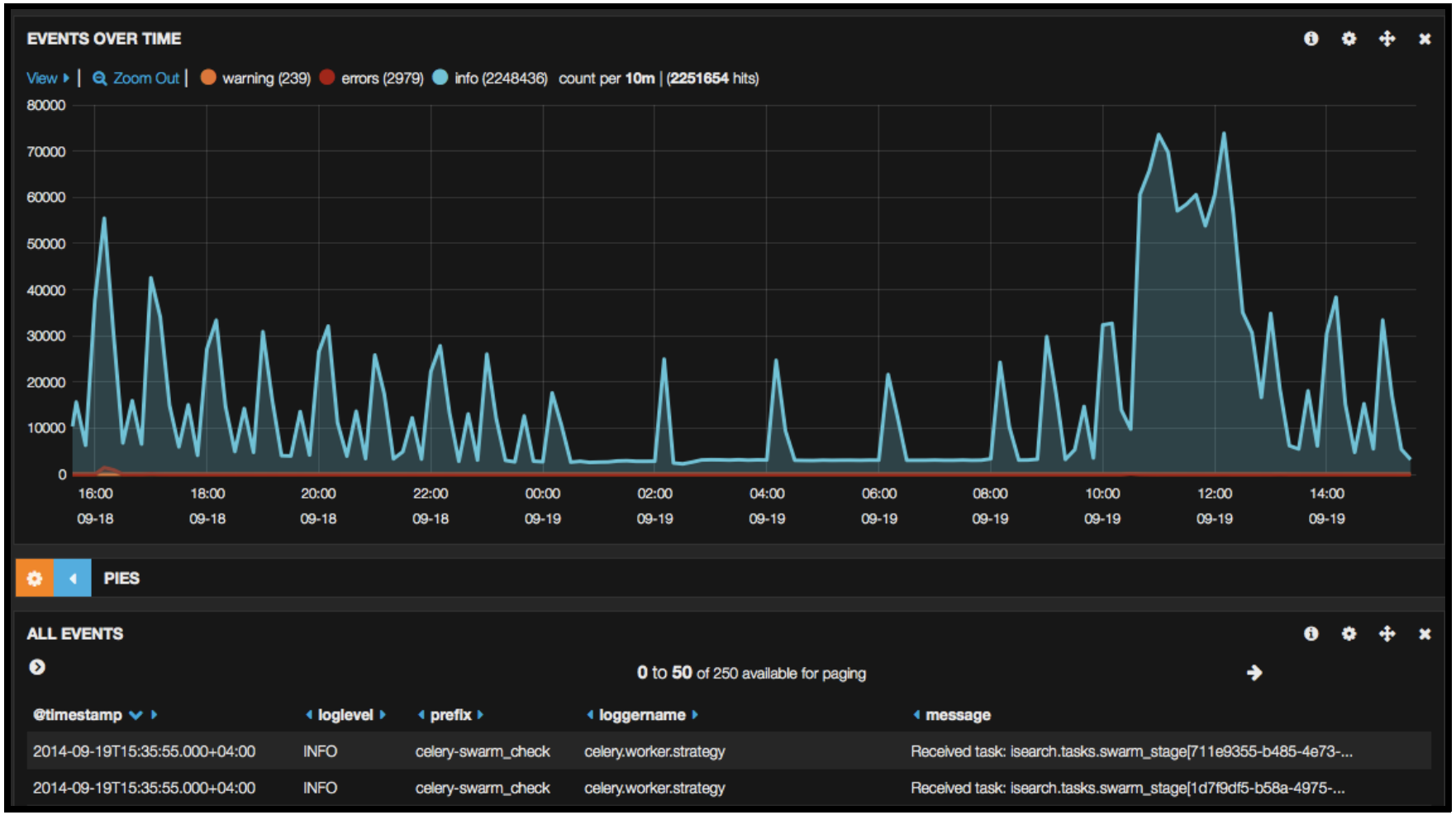
```
import logging

from celery.signals import setup_logging


@setup_logging.connect
def setup_worker_logging(loglevel, logfile, format, colorize):
    root = logging.getLogger()
    root.setLevel(loglevel)
    root.addHandler(logging.handlers.SysLogHandler())
```

Логирование ошибок

Аналитика



Мониторинг

- Получение текущего состояния воркера
  - celery inspect ping
  - celery inspect active
  - celery inspect stats
- Подписка на события воркера
  - celery events
  - celery flower

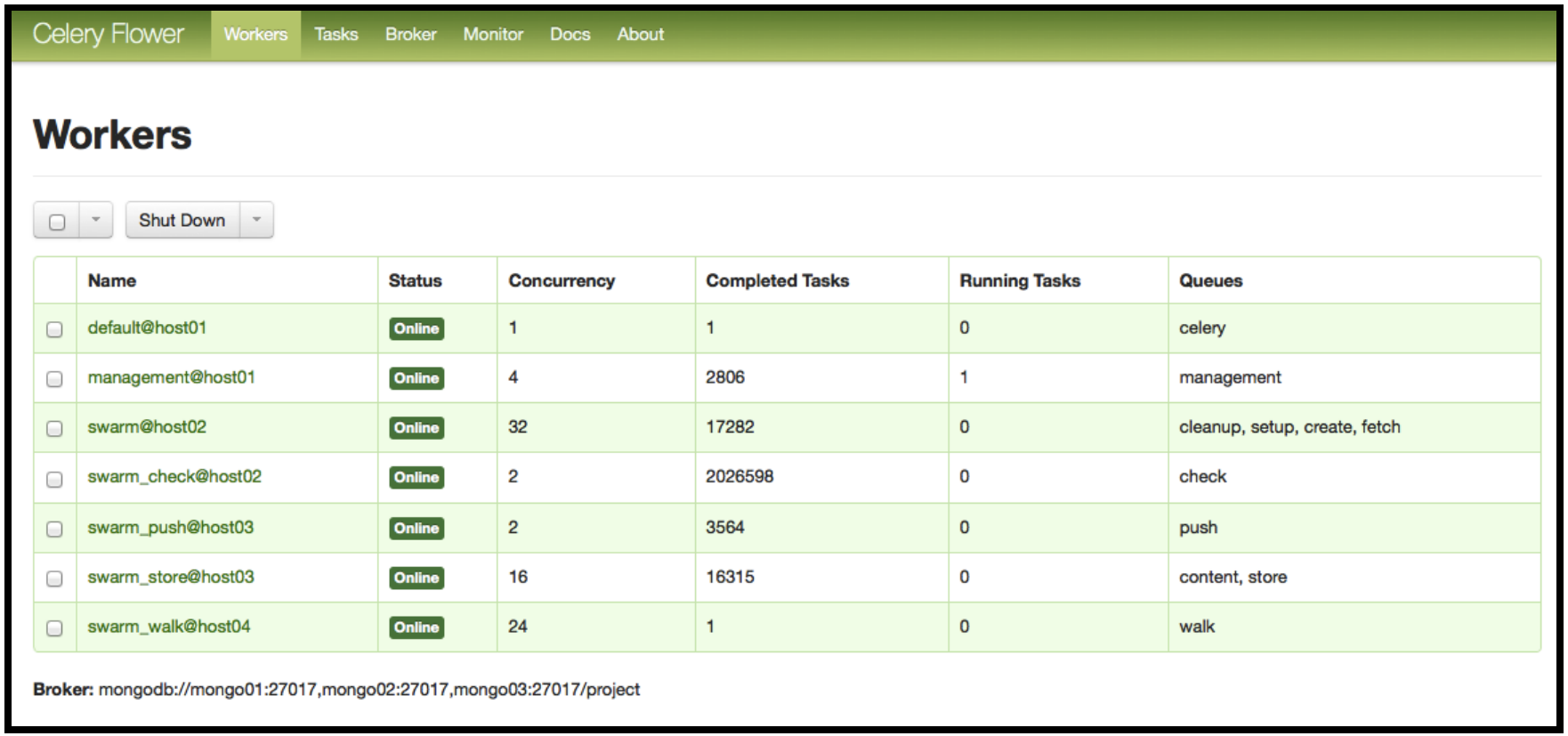
Мониторинг

celery inspect

```
user@host:~$ celery inspect ping
-> default@host01: OK
        pong
-> management@host01: OK
        pong
-> swarm@host02: OK
        pong
-> swarm_check@host02: OK
        pong
-> swarm_push@host03: OK
        pong
-> swarm_store@host03: OK
        pong
-> swarm_walk@host04: OK
        pong
```

Мониторинг

Flower



Мониторинг

Что выбрать

- celery inspect - текущий мониторинг
- celery flower - глубокая отладка и медитация

Вопросы?

daevaorn@yandex.ru
github.com/daevaorn