Tarefas assíncronas com Django

Eu

Renan do Vale de Moura

@rvmoura96

- Desenvolvedor Python na VERT Capital
- Curto jogos (principalmente Borderlands)

Motivação do tema

O motivo deste tema foi:

- Mostrar que existem mais de uma única ferramenta para execução de tarefas assíncronas com Django.
- Mostrar que algumas vezes ferramentas mais leves e simples de implementação podem ser mais do que o suficiente para suas necessidades.

O que são Tasks assíncronas

Tasks assíncronas são tarefas que são executadas em um momento diferente da sua criação e geralmente não são executadas na mesma ordem de sua criação.

Como são armazenadas estas tasks?

Utilizamos algumas ferramentas (**brokers)** que armazenam em filas **(queues)** as tarefas. Quando falamos de aplicações com Django as ferramentas mais comuns são **RabbitMQ** e o **Redis**.

Como são enviadas para os Brokers?

Através de códigos que enviam as tarefas para a fila que serão executadas posteriormente, estes códigos são chamados de **Producers**.

Como são executadas estas tasks?

Estas tarefas são executadas pelos **Workers**, responsáveis pela coleta da tarefa e execução da mesma.

Preciso de tasks assíncronas?

Isso depende muito. Tasks assíncronas adicionam complexidade a um projeto, porém podem melhorar a experiência do usuário.

Preciso de tasks assíncronas?

Aqui temos alguns critério para saber se devemos usar tasks assíncronas:

O resultado leva um grande tempo para ser processado? Você deveria utilizar um sistema de fila de tarefas.

O usuário deve ver imediatamente o resultado da task? Você não deve utilizar um sistema de fila para as tarefas.

Casos de uso de filas de tasks assíncronas

Tarefa	Utilizar tarefas assíncronas?
Envio de e-mails em massa	Sim
Modificar Arquivos (Incluindo imagens)	Sim
Trazer dados de APIs de terceiros	Sim
Inserir ou Atualizar dados em massa	Sim
Atualizar a foto de perfil de um usuário	Não
Adicionar conteúdo a um blog ou CMS	Não
Executar cálculos com longo tempo para conclusão	Sim

Escolhendo um software para ATQ

Software	Prós	Contras
Celery	Comum utilização com Python e Django, compatível com múltiplos meios de armazenamento, flexível e bom com grande quantidade de tarefas.	Difícil para implantação de setup inicial, alta curva de aprendizado mesmo para tarefas simples.
DjangoChannels	Biblioteca mantida pela Django Software Foundation, fácil utilização e adiciona suporte a websockets ao Django	Não há mecanismo para tentar novamente uma tarefas caso a mesma falhe. Suporte somente ao Redis.

Escolhendo um software para ATQ

Software	Prós	Contras
AWS Lambda	Flexível, escalável, fácil configuração inicial.	Chamada da API pode ser lenta, requer logging de serviços externos, adiciona complexidade ao projeto e requer a criação de APIs REST para notificações.
Redis-Queue (RQ)	Custo de memória inferior comparado ao Celery e relativamente mais fácil para configuração	Não há mecanismo para tentar novamente uma tarefas caso a mesma falhe. Suporte somente ao Redis.

Django-RQ

Django-RQ é uma integração entre RQ, uma biblioteca Python para a filas utilizando Redis.

Django-RQ

Para a instalação do Django-RQ:

```
pip install django-rq
```

Após a instalação da lib, devemos instalá-la em nossa aplicação Django no nosso arquivo settings.py:

```
INSTALLED_APPS = (
    # other apps
    "django_rq",
)
```

Django-RQ

Ainda no arquivo settings.py, precisamos configurar nossas filas:

```
RQ_QUEUES = {
    "default": {
        "URL": config("REDISTOGO_URL", default="redis://localhost:6379/0")
    }
}
```

Django-RQ

Após, configurar o settings.py, devemos incluir as urls referentes a app do django-rq em nosso projeto:

```
# For Django < 2.0
urlpatterns += [
    url(r'^django-rq/', include('django_rq.urls')),
]

# For Django >= 2.0
urlpatterns += [
    path('django-rq/', include('django_rq.urls'))
]
```

Exemplo de aplicação prática

$$F_n = F_{n\text{-}1} + F_{n\text{-}2}$$