TUTORIAL RADIS QUEUE - Enqueueing Jobs

O Redis é uma estrutura de armazenamento de dados em memória de código aberto (licenciado pela BSD), usado como banco de dados, *cache* e intermediário de mensagens. Ele suporta estruturas de dados como strings, hashes, listas, conjuntos, conjuntos classificados com consultas de intervalo, bitmaps, hiperloglogs, índices geoespaciais com consultas e fluxos de raio.

O RQ (Redis Queue) é uma **biblioteca Python simples para enfileirar tarefas e processá-las em segundo plano** com os *workers*. É apoiado pela Redis e foi projetado para ter uma baixa barreira à entrada. Pode ser facilmente integrado na sua pilha da web .Qualquer chamada de função Python pode ser colocada em uma fila RQ.

O RQ requer Redis> = 3.0.0.

PRÁTICA

❖ Instalação do Redis

execute os comandos:

```
sudo apt update
sudo apt install redis-server
```

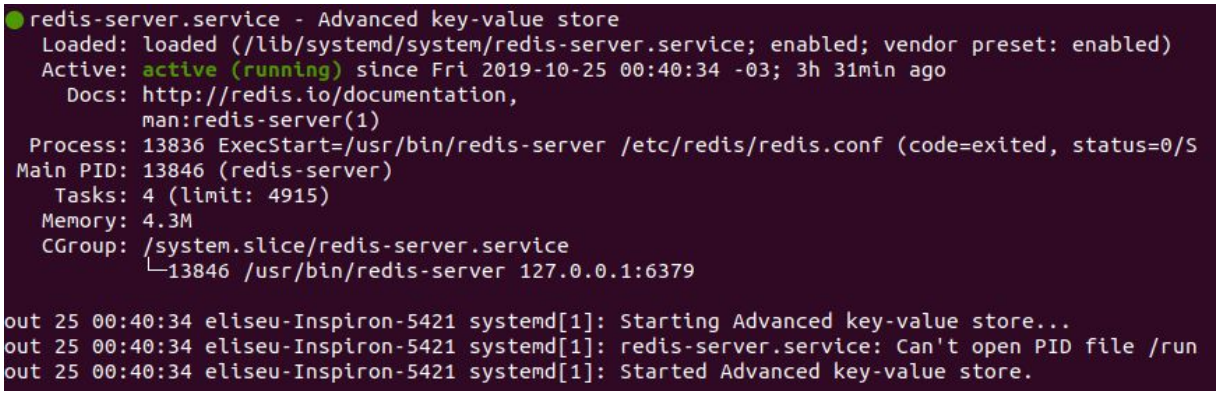
Isso fará o download e instalará o Redis e suas dependências. Após isso, há uma alteração importante na configuração a ser feita no arquivo de configuração Redis, que foi gerado automaticamente durante a instalação.

```
sudo nano /etc/redis/redis.conf
```

Dentro do arquivo, encontre a supervised no e substitua para supervised systemd. Essa diretiva permite que você declare um sistema init para gerenciar o Redis como um serviço, fornecendo a você mais controle sobre sua operação.

Reinicie o redis e depois verifique se a resposta confere com o esperado na imagem seguinte

```
sudo systemctl restart redis.service
sudo systemctl status redis
```



Se ocorreu algum problema na instação ou configuração do redis, veja:
<https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-redis-on-ubuntu-18-04>

❖ Sobre o aplicativo Flask

Flask é um pequeno framework web escrito em Python. Neste exemplo, vamos criar um aplicativo simples, permitindo que um usuário envie uma URL por meio de um formulário.

Para fazer as coisas funcionarem, vamos criar um novo ambiente virtual e instalar as dependências:

- python -m venv env && source env/bin/activate

Nós vamos instalar flask, redis, beautifulsoup4e (biblioteca que facilita a captura de informações de páginas da web) e rq com pip:

- pip install flask redis rq beautifulsoup4

Nós vamos criar um aplicativo simples de arquivo, único em um arquivo chamado app.py

- nano app.py

```
from flask import Flask, request
import redis
from rq import Queue

import time
```

Estamos importando time para simular algum atraso em nossa tarefa em segundo plano.

Em seguida, criaremos uma variável app do flask, ainda no mesmo arquivo app.py, e configuraremos nossa instância do objeto Redis na fila de tarefas:

```
app = Flask(__name__)

r = redis.Redis()
q = Queue(connection=r)
```

Vamos criar uma função muito simples que manipulará uma tarefa. Ele pega um argumento (n) e retorna o tamanho dele com um atraso simulado.

Como você pode ver, é apenas uma função Python normal!

```
def background_task(n):

    """ Function that returns len(n) and simulates a delay """

    delay = 2

    print("Task running")
    print(f"Simulating a {delay} second delay")

    time.sleep(delay)

    print(len(n))
    print("Task complete")

    return len(n)
```

Por fim, criaremos uma rota que procura uma string de consulta n como parâmetro.

```
@app.route("/task")
def index():

    if request.args.get("n"):

        job = q.enqueue(background_task, request.args.get("n"))

        return f"Task ({job.id}) added to queue at {job.enqueued_at}"

    return "No value for count provided"


if __name__ == "__main__":
    app.run()
```

Se n for fornecido na URL, ele adicionará uma tarefa à fila, com o valor para n como argumento da função. Adicionamos uma tarefa a uma fila com: q.enqueue(function_name, args)

Neste caso, temos armazenados isso em uma variável chamada job. Agora temos acesso ao jobo bjeto. Você notará que ligamos job.id e job.enqueued_at que retornam um ID de tarefa exclusivo e a data em que a tarefa foi enfileirada.

❖ Execução

Antes de executar o aplicativo, você precisa iniciar o rq worker.

Abra um novo terminal (no mesmo diretório que app.py) e inicie o *worker* com:

- source env/bin/activate

- rq worker

você verá algo como

```
04:50:29 Worker rq:worker:565a3cb5504c416bb8c22b683b66aa50: started, version 1.1.0
04:50:29 *** Listening on default...
04:50:29 Cleaning registries for queue: default
```

Agora inicie o aplicativo Flask no console inicial (o que está disponível):

- export FLASK_APP=app.py
- export FLASK_ENV=development
- flask run

Vá para o seu navegador e digite :

- http://127.0.0.1:5000/task?n=100

e fique de olho no seu terminal executando o processo rq worker.

No navegador você verá algo como:

```
Task (7dc516bb-7720-446a-acce-cbb272d7f598) added to queue at 2019-03-08
22:03:15.936306
```

No terminal, aparecerá a informação do tipo:

```
22:03:18 default: Job OK (86a8479a-e02d-4aca-8466-6df47fa69efe)
22:03:18 Result is kept for 500 seconds
22:03:18 default: run.background_task('100')
(7dc516bb-7720-446a-acce-cbb272d7f598)
Task running
Simulating a 2 second delay
3
Task complete
22:03:20 default: Job OK (7dc516bb-7720-446a-acce-cbb272d7f598)
22:03:20 Result is kept for 500 seconds
```

❖ Para maiores informações sobre o Tutorial e continuação para Exemplo 2, ver:

Doc Task queues with Flask | Learning Flask Ep. 21 <https://pythonise.com/feed/flask/flask-rq-task-queue>

Suporte de execução:

Video Task queues with Flask | Learning Flask Ep. 21 <https://www.youtube.com/watch?v=3muR5gB8x2o>

RQ easy queue job <https://python-rq.org/> <https://python-rq.org/docs/>