Processamento Distribuído com Celery

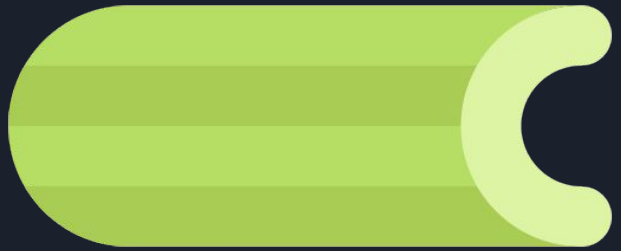
Professor: Adriano Antunes Prates
Aluno: Rafael Lima Guedes

Problema

- Aplicações síncronas bloqueiam requisições longas
- Tarefas pesadas prejudicam performance
- Necessidade de paralelismo e escalabilidade

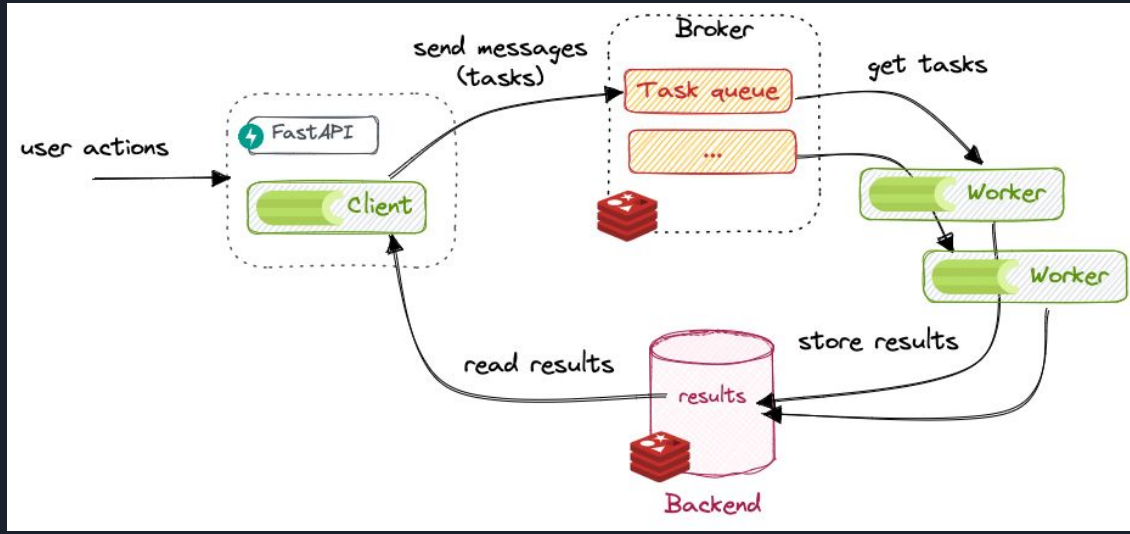
O que é o Celery

- Framework para processamento assíncrono e distribuído
- Executa tarefas em background usando filas (message brokers)



Como funciona?

1. App envia tarefa
2. Broker armazena na fila
3. Worker consome e executa
4. Backend opcional salva resultado
5. App pode consultar status



Cenários de Uso

- Envio de e-mails e notificações
- Processamento pesado (ETL, imagens, IA)
- Execução periódica (cron) com Celery Beat
- Integração com APIs externas sem travar o sistema

Arquitetura e Funcionamento

- App → envia tarefa → Broker (RabbitMQ/Redis)
- Workers → consomem e executam tarefas
- Result Backend → guarda status/resultado
- Escalável: múltiplos workers e filas paralelas

Limitações e Desafios

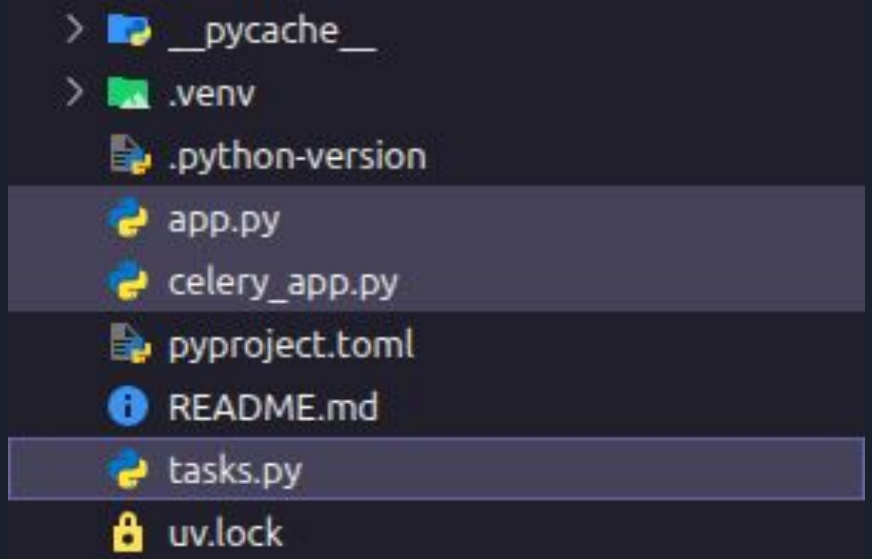
- Exige infraestrutura adicional (broker + backend)
- Debug mais complexo por ser assíncrono
- Configuração inicial pode ser extensa
- Workers precisam ser monitorados

Benefícios

- Respostas rápidas (não bloqueia requisições)
- Escalabilidade horizontal via múltiplos workers
- Tolerância a falhas com retries
- Desacoplamento da aplicação principal

Demonstração (Fluxo Básico)

- Redis como broker
- App cliente
- Celery worker



Demonstração (Fluxo Básico)

- Instalação Python
- Instalação Redis
- Instalação Celery

```
uv python install 3.12
```

```
sudo apt install redis-server -y
```

```
uv venv && source .venv/bin/activate
uv add celery[redis]
```

Demonstração (Fluxo Básico)

- Definição da função Celery
- Conexão com Redis

```python
@celery_app.task
def soma(a, b, task_num):
    print(f"[TASK {task_num}] Iniciando processamento...")
    time.sleep(5)
    print(f"[TASK {task_num}] Finalizando processamento")
    return a + b


celery_app = Celery(
    "celery_demo",
    broker="redis://localhost:6379/0",
    backend="redis://localhost:6379/0"
)
```

Demonstração (Fluxo Básico)

- Comando para iniciar worker
- Worker escutando fila

```
uv run celery -A tasks worker --loglevel=info
```

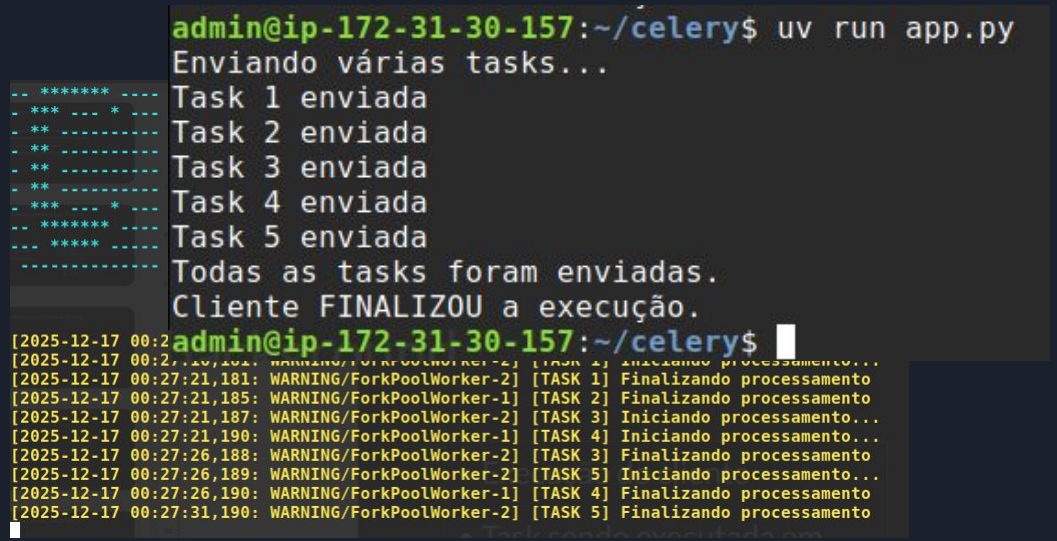
```
 -------------- celery@ip-172-31-30-157 v5.6.0 (recovery)
--- ***** -----
-- ******* ---- Linux-6.12.48+deb13-cloud-amd64-x86_64-with-glibc2.41 2025-12-16 18:06:23
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         celery_demo:0x7f623ae7e7b0
- ** ---------- .> transport:   redis://localhost:6379/0
- ** ---------- .> results:     redis://localhost:6379/0
- *** --- * --- .> concurrency: 2 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.soma

[2025-12-16 18:06:24,061: INFO/MainProcess] Connected to redis://localhost:6379/0
[2025-12-16 18:06:24,065: INFO/MainProcess] mingle: searching for neighbors
[2025-12-16 18:06:25,074: INFO/MainProcess] mingle: all alone
[2025-12-16 18:06:25,087: INFO/MainProcess] celery@ip-172-31-30-157 ready.
```

Demonstração (Fluxo Básico)

- Execução do cliente
- Task sendo executada em background



Conclusão

- Celery é solução madura para tarefas assíncronas
- Permite paralelizar, escalar e otimizar o desempenho
- Exige boas práticas e infraestrutura mínima

FIM