CELERY EXPERT

Architecture Distribuée, Interopérabilité Multi-langages et Déploiement Industriel sous Linux.

01. Fondations

L'écosystème global et les prérequis système.

POURQUOI CELERY ?

Latence Réduite

Déchargez les tâches lourdes (Email, PDF) pour répondre instantanément à l'utilisateur.

Scalabilité

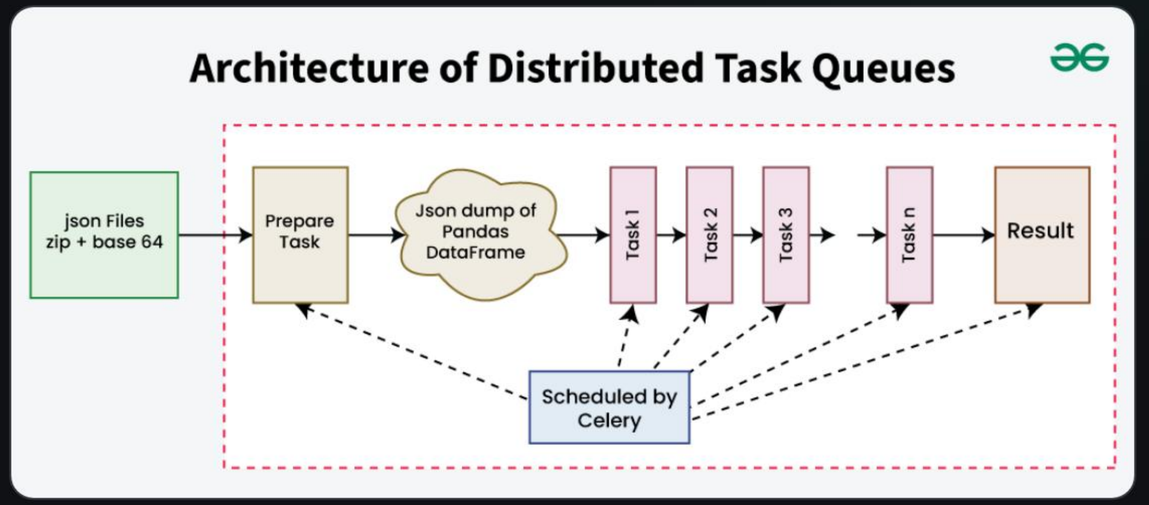
Ajoutez des serveurs (Workers) à la volée pour traiter des millions de tâches en parallèle.

Fiabilité

Mécanisme de retry automatique, priorisation des files et persistence des tâches.

ARCHITECTURE & FLUX

- **Producers :** Django, PHP, Java envoient un message JSON au Broker.
- **Broker :** RabbitMQ ou Redis stocke la file d'attente.
- **Workers :** Nœuds Linux qui exécutent le code métier.
- **Backend :** Stockage des résultats (PostgreSQL, Redis).



INSTALLATION & PRÉREQUIS

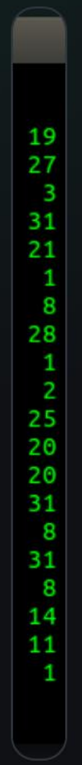
1. Système & Broker

Installation sur Ubuntu/Debian :

```
sudo apt update sudo apt install python3-pip rabbitmq-server sudo systemctl enable rabbitmq-server
```

2. Environnement Python

```
python3 -m venv venv source venv/bin/activate pip install celery[redis,auth]
```



02. Interopérabilité

Comment appeler Celery depuis Django, PHP ou Java.

NATIVE : PYTHON & DJANGO

Configuration Django

```
# settings.py CELERY_BROKER_URL = 'amqp://guest@localhost' CELERY_RESULT_BACKEND = 'rpc://'
```

Définition d'u

```
@shared
```

PHP & JAVA : LES CLIENTS

PHP Client

Utilisez `celery-php` ou envoyez directement via AMQP.

```
$c = new Celery('localhost', 'guest', 'guest'); $c→PostTask('tasks.add', [2, 2]);
```

Java / Spr

Interop via Rabbit

```
rabbit
```

CHOISIR SON BROKER

Serveur Broker	Fiabilité	Vitesse	Cas d'usage recommandé
RabbitMQ	Haute (ACks)	Bonne	Production critique, workflows complexes.
Redis	Moyenne	Excellente	Petites apps, stockage de résultats, cache.
Amazon SQS	Haute	Variable	Infrastructures Full-Cloud (AWS).

DÉPLOIEMENT AVEC SYSTEMD

Pour que vos workers tournent en permanence en arrière-plan sous Linux :

```
# /etc/systemd/system/celery.service [Unit] Description=Celery Service After=network.target [Service] Type=fo
```

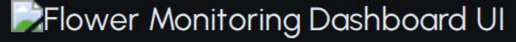
MONITORING : FLOWER

Ne volez pas à l'aveugle. Flower est le tableau de bord temps réel pour Celery.

- État de santé des Workers.
- Historique détaillé des tâches.
- Contrôle des débits (Rate limiting).
- Alerting sur erreurs critiques.

```
pip install flower

celery -A proj flower
```

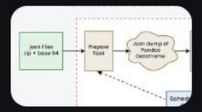


Questions ?

Celery est l'épine dorsale de la scalabilité asynchrone moderne.

github.com/celery/celery docs.celeryq.dev

IMAGE SOURCES



https://media.geeksforgeeks.org/wp-content/uploads/20240912133124/Architecture-of-Distributed-Task-Queues.png

Source: www.geeksforgeeks.org

https://media.geeksforgeeks.org/wp-content/uploads/cli_example.png

Source: www.geeksforgeeks.org

https://www.emergentsoftware.net/media/4nui0wog/body.png

Source: www.emergentsoftware.net

https://docs.celeryq.dev/en/stable/_images/dashboard.png

Source: docs.celeryq.dev