Celery

- Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system.
- It's a task queue with focus on real-time processing, while also supporting task scheduling.
- Celery has a large and diverse community of users and contributors; you should come join us on IRC or our mailing-list.
- Celery is Open Source and licensed under the BSD License.

Celery is a distributed task queue system that can be used with Django to perform asynchronous tasks such as sending emails, processing background jobs, and more. In this guide, I'll walk you through the steps to integrate and use Celery with a Django project.

1. Installation: First, you need to install Celery along with a message broker such as RabbitMQ or Redis. You can install Celery using pip:

   pip install celery

2. Configuration: In your Django project settings, you'll need to configure Celery to use a message broker. This includes specifying the broker URL, which could be something like RabbitMQ or Redis.

3. Creating tasks: Tasks are defined as Python functions decorated with `@celery.task`. These tasks can then be executed asynchronously.

   ```
   # tasks.py
   from celery import shared_task

   @shared_task
   def send_email(email):
    # Code to send email
      Pass
   ```

4. Calling tasks: You can call these tasks from your Django views or other parts of your application. Instead of executing the task directly, you'll use Celery's `delay ()` method to enqueue the task for asynchronous execution.

   ```
   # views.py

   from .tasks import send_email


   def my_view(request):

     # Code logic

     email = "example@example.com"
   ```

send_email.delay(email)

return HttpResponse("Email sent asynchronously.")

5. Running the Celery worker**: Celery requires a worker process to execute the tasks. You'll need to start the Celery worker process using the `celery` command.

   celery -A your_project_name worker -l info

   With Celery set up in your Django project, tasks can be executed asynchronously, allowing your application to handle time-consuming tasks without blocking the main request-response cycle. This is particularly useful for tasks like sending emails, processing large datasets, or interacting with external APIs.