**What is Celery?**

“[Celery](http://www.celeryproject.org/) is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.” For this post, we will focus on the scheduling feature to periodically run a job/task.

**Why is this useful?**

* Think of all the times you have had to run a certain task in the future. Perhaps [you needed to access an API](https://realpython.com/api-integration-in-python/) every hour. Or maybe you needed to [send a batch of emails](https://realpython.com/python-send-email/) at the end of the day. Large or small, Celery makes scheduling such periodic tasks easy.
* You never want end users to have to wait unnecessarily for pages to load or actions to complete. If a long process is part of your application’s workflow, you can use Celery to execute that process in the background, as resources become available, so that your application can continue to respond to client requests. This keeps the task out of the application’s context.

From <<https://realpython.com/asynchronous-tasks-with-django-and-celery/>>

It needs message broker (rabbitmq or something else) to send and receive messages. So rabbitmq service needs to be running. Celery also needs to run so that it polls rabbitmq

Workers and clients will automatically retry in the event of connection loss or failure

Machine generated alternative text:
Its called client 
because it 
sends 
request(task to 
execute) 
(Client) 
Microservice 1 
run supervisor.delay 
will add to queue 
Message broker 
polls 
run supervisor 
fetches 
Microservice2 
Celery worker 
Here tasks will be present 
which can be called by 
others. For eg: 
run supervisor 
runs aync 
Celery worker is assigned to a 
module by celery worker 
command. It will poll 
continuously on the message 
broker(rabbitmq) 
Hence all the requests by 
client microservice is run 
asynchronously by celery 

---------------------------------------------------------------

SIMPLE EXAMPLE:

Let’s create the file tasks.py:

**from celery import** Celery

app = Celery('tasks', broker='pyamqp://guest@localhost//')

**@app**.task  
**def add**(x, y):  
 **return** x + y

From <<https://docs.celeryproject.org/en/latest/getting-started/first-steps-with-celery.html>>

Here instance creation(Celery()) and tasks is present in same file. But for larger projects we can separate this in this way.

Here Message broker is directly given, but you can do config\_from\_object and load the configurations from another .py file or django settings file.

Auto discover tasks looks for tasks in all tha tasks.py files in the INSTALLED\_APPS.

Machine generated alternative text:
3 Cetery(•—.•) 
cmfig_ • . cmf : ) 

------------------------------------------------------------------

USUALLY USED IN THIS WAY:

<https://docs.celeryproject.org/en/latest/getting-started/next-steps.html#project-layout>

Our Project

Project layout:

proj/**\_\_init\_\_**.py  
 /celery.py  
 /tasks.py

proj/celery.py (Instance creation)

**from celery import** Celery

app = Celery('proj',  
 broker='amqp://',  
 backend='rpc://',  
 include=['proj.tasks'])

# Optional configuration, see the application user guide.  
app.conf.update(  
 result\_expires=**3600**,  
)

**if** \_\_name\_\_ == '\_\_main\_\_':  
 app.start()

In this module you created our [**Celery**](https://docs.celeryproject.org/en/latest/reference/celery.html#celery.Celery) instance (sometimes referred to as the *app*).

To use Celery within your project you simply import this instance. In this way:-

proj/tasks.py (define tasks by importing instance from celery.py)

**from .celery import** app

**@app**.task  
**def add**(x, y):  
 **return** x + y

**@app**.task  
**def mul**(x, y):  
 **return** x \* y

**@app**.task  
**def xsum**(numbers):  
 **return** sum(numbers)

From <<https://docs.celeryproject.org/en/latest/getting-started/next-steps.html#project-layout>>

--------------------------------------------------------------------------------------

Starting the worker

The **celery** program can be used to start the worker (you need to run the worker in the directory above proj):

**$** celery -A proj worker -l INFO

-----------------------------------------------------

[Calling Tasks](https://docs.celeryproject.org/en/latest/getting-started/next-steps.html#id4)

You can call a task using the **delay()** method:

**>>> from proj.tasks import** add

**>>>** add.delay(**2**, **2**)

From <<https://docs.celeryproject.org/en/latest/getting-started/next-steps.html#project-layout>>

-------------------------------------------------------------------------------------

**ARGUMENTS:**

beat

Start the beat periodic task scheduler.

celery beat [OPTIONS]

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

call

Call a task by name.

celery call [OPTIONS] NAME

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

multi

Start multiple worker instances.

celery multi [OPTIONS]

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

purge

Erase all messages from all known task queues.

Warning:

There’s no undo operation for this command.

celery purge [OPTIONS]

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

result

Print the return value for a given task id.

celery result [OPTIONS] TASK\_ID

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

status

Show list of workers that are online.

celery status [OPTIONS]

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

worker

Start worker instance.

Examples

$ celery –A=proj worker -l INFO $ celery -A proj worker -l INFO -Q hipri,lopri $ celery -A proj worker –concurrency=4 $ celery -A proj worker –concurrency=1000 -P eventlet $ celery worker –autoscale=10,0

celery worker [OPTIONS]

From <<https://docs.celeryproject.org/en/stable/reference/cli.html>>

-----------------------------------------------------------------------------------------------

**TASK AND SHARED TASK**

**TASK:**

app1 = Celery()   
@app1.task   
def test():   
 pass

app2 = Celery()

But, the name ‘test’ will always refer to the instance **bound** to the ‘app1’ app, so it will be configured using app1’s configuration:

assert test.app is app1 \*\*

**SHARED TASK:**

app1 = Celery()

@shared\_task   
def test():   
 pass   
assert test.app is app1 \*\*

app2 = Celery()   
assert test.app is app2 \*\*

Here apps can actually have their own instance and configuration. Its useful in libraries and reusable apps where tasks are reused.

\*\*\*\*\*\*INTERVIEW QUESTIONS\*\*\*\*\*\*\*\*\*

<https://www.interviewqueries.com/celery-interview-questions/> -- Here you will already know the answers.. But check once

<https://realpython.com/asynchronous-tasks-with-django-and-celery/> This link is good. You can go through.

Incase they ask u to explain celery, mainly u can start explaining the diagram that I put I think.