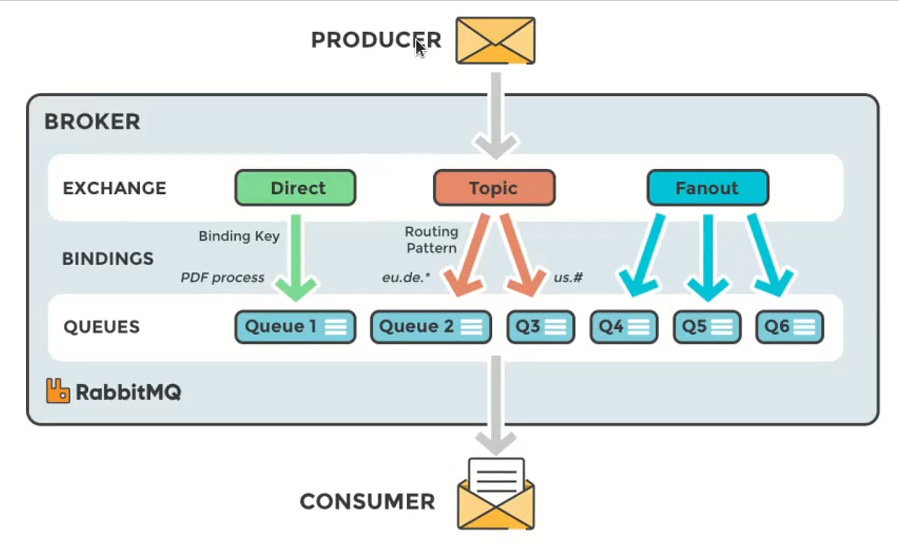
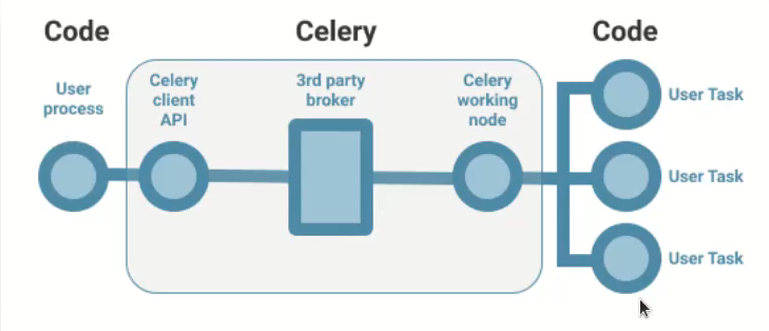
## معمولا زمانی که کاربر درخواستی را به سرور ارسال میکند انتظار پاسخ همزمان را دارد. اما اگر درخواست کاربر نیاز به پردازش سنگینی داشته باشد، امکان پاسخ همزمان مطلوب به کاربر وجود نخواهد داشت. همچنین در صورت افزایش بار کاری سرور ممکن است زمان پاسخ به کاربر طولانی شده و امکان ارسال درخواست های بعدی از کاربر سلب شود. به همین دلیل درخواست های زمانبر را با استفاده از brokerهایی مثل RabbitMq در یک صف ذخیره کرده و به نوبت پردازش شده و پاسخ مطلوب به کاربر ارسال میشود. اما brokerها به دلیل سختی استفاده و تعداد زیاد config معمولا توسط برنامه نویس ها رد میشوند. Celery یک واسط بین کاربر و broker است که وظیفه صف بندی درخواست ها به ساده ترین شکل ممکن را دارند. Celery با استفاده از زبان python نوشته شده است اما امکان استفاده در دیگر زبان ها وجود دارد. Celery برای کارکردن و ذخیره درخواست ها نیاز به یک broker دارد. در این پروژه ما از RabbitMq به عنوان broker استفاده خواهیم کرد

=====================================================================================  
  
celery ابزاریست برای ساده تر کردن ارتباط بین producer و consumer با broker. برای نصب کردن celery میتونید از دستور pip install celery استفاده کنید. برای استفاده از celery حتما باید از یک broker استفاده کنید. پیشنهاد celery استفاده از rabbitmq به عنوان broker میباشد اما میتوانید از redis, couchdb, mongodb یا تمام دیتابیس هایی که از SqlAlchemy و django orm استفاده میکنند، استفاده کنید.

<https://docs.celeryproject.org/en/stable/>





در‌واقع celery میاد و بعد از producer و قبل از consumer قرار میگیرد ( بین producer و broker وconsumer و broker) و broker را مدیریت میکند پس اصلاً کاری با broker نداریم و مدیریت broker برعهده celery میباشد

<https://docs.celeryproject.org/en/stable/index.html>  
=====================================================================================

Launch the app in celery

در این قسنت با application در celery آشنا میشوید. برای شروع کار با celery باید از کلاس Celery استفاده کنید و از این کلاس یک نمونه جدید ایجاد کنید. به هر نمونه ای که از کلاس Celery ایجاد میشود application یا بطور خلاصه app گفته میشود. تمامی appها باید یک اسم داشته باشند که پیشنهاد میشود همنام ماژول باشد. بعد از ایجاد کردن app با استفاده از دکوریتور task میتوانید فانکشن های خود را به صورت async صدا زنید.

تمامی appهای celery ویژگی thread\_safe هستند به این معنی که تمامی این appها به خوبی میتوانند در کنار هم کار کنند

<https://docs.celeryproject.org/en/stable/userguide/application.html>

app = Celery('one', broker="amqp://localhost")

اولین مقداری که کلاس celery میگیرد main میباشد که معمولاً اسم ماژول را به آن میدهند

دمین مقدار آدرس brokerی میباشد که قرار است celery به آن متصل شود (به دلیل اینکه روی کامپیوتر لکال rabbitmq وجود دارد localhost را مینویسید اما اگر روی سروری وجود داشت باید IP اون سرور نوشته شود)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

برای اینکه celery متوجه شود که functionی که قرار است از آن استفاده کند کدام است کافیست در بالای فانکشن اسم instanceی که از کلاس celery ایجاد کرده‌اید همراه با task را به صورت دکوریتور بالای فانکشن بنویسید

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

الان celery متوجه می‌شود که باید حواسش به این فانکشن باشد و کارهارو صف بندی کند

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

با دستور زیر میتوانیم پروژه ای که مربوط به celery میباشد را اجرا کنیم

celery -A one worker

دستور بالا به این معنی میباشد که برای celery یک worker ارسال کن با نام one

توجه شود که one اسم ماژولتان میباشد (one.py)

در‌واقع ماژول one.py همان worker میباشد

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه دستور بالا را اجرا میکنید یکسری پیغام ها نمایش داده می‌شوند در قسمتی از این پیغام ها خط زیر وجود دارد

**app:         one:0x7fa0a9ea4370**

oneی که جلویapp نوشته شده است همان اولین پارامتری است که به کلاس Celery هنگام ساخت instance دادید (main )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

در قسمت دیگری از خروجی عبارت زیر را مشاهده میکنید:

**transport:   amqp://guest:**[**\*\*@localhost**](mailto:**@localhost)**:5672//**

amqp که نشان دهنده rabbitmq میباشد و \*\*:quest نشان دهنده username و پسورد میباشد که در config اولیه rabbitmq یوزرنیم و پسوردی quest و quest میباشد لازم نیست قرار دهید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**concurrency :** میزان کاری است که سلری میتواند بطور همزمان انجام دهد (در واقع تعداد هسته های cpu را نمایش میدهد )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

آپشنی که در worker وجود دار l یا log میباشد با این آپشن به celery میگویید که چه نوع loglevelی را برای من نمایش بده:

celery -A one -l info worker

الان یعنی تمام logهایی که levelشان از info بالاتر هستند را نمایش بده

=====================================================================================

tasks

در این قسمت با نحوه استفاده از workerها به صورت همزمان آشنا میشوید. در celery به طور کلی دو متد apply\_async و delay برای صدا زدن workerها وجود دارد. هر دوی این متدها یک کار را انجام میدهند اما با استفاده از apply\_async شما میتوانید یکسری آپشن های بیشتری را برای صدا زدن wrokerها استفاده کنید.

<https://docs.celeryproject.org/en/stable/userguide/calling.html>

مثلا:

countdown=10 ==> یعنی 10 ثانیه صبر کن بعد دستورات را ارسال کن

expires=120 ==> بعد از 120 ثانیه اگر تسک اجرا نشد آن را از صف حذف کن

و نحوه ی ارسال اطلاعات نیز متفاوت میباشد

مثال:

task.delay(arg1, arg2, kwarg1='x', kwarg2='y')

task.apply\_async(args=[arg1, arg2], kwargs={'kwarg1': 'x', 'kwarg2': 'y'})

خب باید به این صورت رفتار کنید که در یک ترمینال با استفاده از دستور celery -A one -l info worker ورکر (worker) خود را اجرا کنید و در ترمینال دیگر وارد shell پایتون شوید و فانکشن add را import کنید (from one import add) سپس با استفاده از delay یا  
apply\_async به ارسال اطلاعات به پردازید =====> add.apply\_async(args=[5,8]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

هنگامیکه دستور (add.apply\_async(args=[5,8] را بزنید موارد زیر برای شما نمایش داده خواهد شد :

1. بلافاصله بعد از زدن enter خط زیر نمایش داده می‌شود به این معنی که این id uniqe تسک شما میباشد

<AsyncResult: 1a675dbe-d3d2-4403-8d2c-8370b14c4dff>

2. در ترمینالی که celery درحالت اجرا است به خطوط زیر برخواهید خورد :

Received task: one.add[f582d73e-032b-4206-a46e-d03373ba5bd4]

Task one.add[f582d73e-032b-4206-a46e-d03373ba5bd4] succeeded in 15.013456377000693s: 13

به این معنی که یک تسک با شماره id مورد نظر رسید و بعد از 15 ثانیه جواب موفق تسک که ۱۳ میباشد محاسبه می‌شود

حال اگر تعداد apply\_async زیاد بود و یا worker در حالت اجرا نباشد تمامی این تسک ها صف بندی می‌شوند و در زمانی که worker one اجرا شد تسک ها اجرا می‌شوند (به تعداد هسته های cpu تسک هارا همزمان انجام میدهد)

=====================================================================================

flower یک ابزار گرافیکی برای مدیریت کردن اجزای celery مثل taskها و workerها است. flower در مرورگر اجرا میشود.

https://flower.readthedocs.io/en/latest/

https://docs.celeryproject.org/en/stable/userguide/monitoring.html?highlight=flower#flower-real-time-celery-web-monitor

برای نصب flower باید از دستور pip install flower استفاده کنید

برای راه اندازی سرور چندین راه‌کار وجود دارد که در داکیومنت بالا وجود دارد و یکی از آن‌ها مورد زیر میباشد

$ celery flower -A one --broker=amqp://guest:[guest@localhost](mailto:guest@localhost):5672//

بعد از اجرا کردن دستور بالا میتوانیم در localhost روی پورت 5555 به flower دسترسی داشته باشیم

=====================================================================================

management utility

در این قسمت با نحوه مدیریت کردن celery با استفاده از command line آشنا میشوید.محیط های گرافیکی به دلیل مصرف منابع زیاد آنچنان مورد پسند برنامه نویس ها نیستند به همین دلیل امکان این وجود دارد که شما بتوانید اجزای مختلف celery را با استفاده از command line مدیریت کنید

https://docs.celeryproject.org/en/stable/userguide/monitoring.html?highlight=flower

1. اولین دستور shell میباشد که شمارا به shell پایتون میبرد =========> celery shell

2. دومین دستور status میباشد که وضعیت و تعداد worker ها را نشان میدهد(منظور از node همان worker میباشد

celery status

3. دستور بعدی purge میباشد که تمام کارهایی که درون صف ها میباشد را حذف میکند ( توجه شود که هیچ برگشتی برای این دستور وجود ندارد و آپشن f به معنی force میباشد که دیگر از شما سؤال نمیپرسد ) ====> celery purge -f

4. دستور بعدی inspect active میباشد که لیستی از تسک هایی که active هستند را نمایش میدهد

celery inspect active

5. دستور بعدی inspect scheduled میباشد که لیست از تسک هایی که آماده هستند که اجرا شوند را نمایش میدهد

celery inspect scheduled

6. دستور بعدی inspect stats میباشد که درباره هر worker اطلاعات کاملی را نشان میدهد

celery inspect stats

=====================================================================================

keeping results

در این قسمت با rpc در celery آشنا میشوید. rpc روشی است که در آن میتوان پاسخ هر task رو از worker به producer ارسال کرد. به زبانی دیگر بعضی مواقع شما قصد دارید که دنباله ی کاری که به worker فرستاده اید را بگیرید و ببینید که نتیجه‌اش به چه شکلی بود

[https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html#id11](https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html" \l "id11)

در مثال‌هایی که در قسمت‌های قبل انجام دادیم دیدیم که نتیجه ی تسک در همان worker celery نمایش داده میشد و در shell پایتون که تسک ها را ارسال کرده بودیم نمیتوانستیم نتیجه ی تسک را مشاهده کنیم برای اینکار باید در هنگام ساخت instance پارامتر دیگری را به کلاس Celery به نام backend بدهیم :

app = Celery('tasks', backend='rpc://', broker='amqp://localhost')

اگر از دیتابیس redis استفاده میکنید میتوانید به صورت زیر بنویسید

app = Celery('tasks', backend='redis://localhost', broker='amqp://localhost')

حال هنگام فرستادن تسک باید ابتدا آن را در یک متغیر ذخیره کنید ( result = one.delay(4,4)) اکنون در result میتوانید به مقادی مختلفی دسترسی داشته باشید :

result.ready() ====> نشان میدهد که نتیجه آماده شده است یا خیر

result.get()=====> نتیجه را نمایش میدهد اگر هنوز پاسخ آماده نشده باشد آن‌قدر صبر میکند تا پاسخی دریافت کند

فانکشن get پارامتری را به عنوان ورودی دریافت میکند به نام timeout که به این معنی میباشد که اگر در مدت زمانی که به timeout داده شده است (بر حسب ثانیه) پاسخ آمد جواب را نمایش بده و گرنه exeption.timeouterror را بر میگرداند

=====================================================================================

config

در این قسمت با نحوه کانفیگ کردن برنامه های celery آشنا میشوید. celery کانفیگ های زیادی داره که براساس معماری و broker مورد استفاده تفاوت خواهد داشت

https://docs.celeryproject.org/en/stable/userguide/configuration.html

[https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html#id12](https://docs.celeryproject.org/en/stable/getting-started/first-steps-with-celery.html" \l "id12)

سه روش کلی برای ایجاد config هایتان وجود دارد :

1. app.conf.task\_serializer = 'json'

2. app.conf.update(

task\_serializer='json',

accept\_content=['json'], # Ignore other content

result\_serializer='json',

timezone='asia/Tehran',

enable\_utc=True,

)

3. app.config\_from\_object('celeryconfig')

روش اول برای مواردی است که config زیادی وجود ندارد روش دوم برای configهای زیاد میباشد و روش سوم کاربردی‌تر میباشد به این صورت که تمامی config هارا در فایلی دیگر قرار میدهید و این فایل را به app celery میشناسانید

=====================================================================================

signatures

تا الان taskهایی که داشتیم در یک فایل بود حال فرض کنید که تسک هایمان در فایل‌های متفاوت میباشد و یا دارید از میکروسرویس ها استفاده میکنید . میکروسرویس ها کد بیس شون از هم جدا میباشد زمانی که شما در یک میکروسرویس یک تسکی را دارید و میخواهید آن را به یک میکروسرویس دیگر ببرید و از آن استفاده بکنید همین‌جوری نمیتوانید این کار را انجام دهید signatureها میان و کل تسکی که شما در اون میکروسرویس دارید رو جمع میکنند یک جا و به یک بسته آن را تبدیل میکنید و شما میتوانید آن را به جای دیگری انتقال دهید

[https://docs.celeryproject.org/en/stable/userguide/canvas.html#signatures](https://docs.celeryproject.org/en/stable/userguide/canvas.html" \l "signatures)

همان‌طور که در داکیومنت آمده است signature ها میان argumentها و keyword argument ها و اون آپشن های زمان اجرا را برای شما جمع میکند به طوریکه بتوانید آن‌ها را داخل فانکشن ها پاس دهید و یا serialize کنید آن‌ها را و به جای دیگر بفرستید

برای مثال همان کدی که در one.py وجود دارد در آخرش از فانکشن یک signature میسازیم :

result = add.signature((3, 4))

حال میتوانیم از result را در میکروسرویس دیگر به وسیله ی کد زیر اجرا کنیم :

result.delay()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

کاربرد دیگری که وجود دارد این است که شما میخواهید تعداد زیادی تسک را پشت سرهم انجام دهید میتوانید برای هر فانکشن یک signature بنویسید و آنهارا درون لیستی قرار دهید و آن را اجرا کنید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

callbackها به این معنی هستند که شما میتوانید نتیجه ی یک تسک را به یک تسک دیگر ارسال کنید

[https://docs.celeryproject.org/en/stable/userguide/canvas.html#id4](https://docs.celeryproject.org/en/stable/userguide/canvas.html" \l "id4)

result = add.apply\_async((6, 1), link=sub.signature((5,)))

در این مثال جواب فانکشن add که ۷ میباشد به عنوان ورودی اول به فانکشن sub داده می‌شود و ۵ به عنوان پارامتر بعدی در نظر گرفته می‌شود

توجه شود که حتماً باید از apply\_async استفاده کنید و نمیتوانید از delay استفاده کنید

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

signatureها ۳ شکل مختلف دارند :

۱. partial به این معنی که args هایی که فانکشن نیاز دارد تا آن تسک اجرا شود را نصفه‌نیمه به signature بدهیم در این صورت تا زمانی که args ها کامل نشوند تسک انجام نخواهد شد :

partial = add.s(2) # incomplete signature

partial.delay(4) # 2 + 4

2. immutability معمولاً با callback ها استفاده می‌شوند و زمانی استفاده می‌شود که اون نتیجه‌ای که قرار است اجرا شود برای فانکشن دیگر بی‌اهمیت باشد

result = add.apply\_async((6, 1), link=sub.signature((5, 4), immutable=True))

result = add.apply\_async((6, 1), link=sub.si(5, 4))

3. روش سوم همان callback میباشد که در بالا توضیح داده شد

=====================================================================================