

Threading Vs Celery in Django



Radwan Hijazi · [Follow](#)

5 min read · Mar 25, 2024



Listen



Share

In our Django projects we have a very large and big tasks to implement, this tasks takes a lot of time, And this is very bad for the project performance .

So the key to solve this performance issues is on **background tasks**.

in this article we will talk about threading and celery as a background tasks in Django, the pros and cons of each library and when we use celery or threading.

The main task

if you have an endpoint like “`api/user/send-mail/<int:user_id>/`” this endpoint takes the user id and get the email of this user and takes one parameter in the body which is the `message_content` , Then Send the email with this `message_content` to this user, these process takes a lot of time and we want to implement the task in background

Example of the code :

```
#myapp/mailer.py

#import libs
from django.conf import settings
from django.core.mail import send_mail

def send_email_by_threading(to_user_email,msg_content) :
```

```
# subject of the email msg
subject = "New message from admin"
# the admin email [sender]
from_email = settings.EMAIL_HOST_USER
# this email must goes to [receiver]
recipient_list = [to_user_email]
# send_mail method to implement the logic
send_mail(subject, msg_content, from_email, recipient_list)
```

Let's implement this task with threading and celery .

Threading in Django

threading is a built-in library in python and you don't need to install it , and work in threading in Django is so easily as you imagine .

Example of send the email with threading :

```
# myapp/views.py
from rest_framework import decorators, permissions
# import threading
import threading
from django.contrib.auth.models import User
from rest_framework.response import Response
from .mailer import send_email_by_threading

@decorators.api_view(["POST"])
@decorators.permissions_classes([permissions.IsAdminUser])
def SendMailView (request,user_id) :
    try :
        # catch the email message from client
        msg_content = request.data.get("message_content",None)

        # check the message is inserted by the client
        if msg_content is None :
            return Response({"message":"please insert the message"},status=400)

        # check if there user in the db has this id or not
        try :
```


```
        to_user = User.objects.get(id=user_id)
    except User.DoesNotExist:
        return Response({"message": "user with this id not found"}, status=404)

    # send the email by threading
    to_user_email = to_user.email
    thread = threading.Thread(target=send_email_by_threading, args=(to_user_email,))
    thread.start()

    return Response({"message": "email sent successfully"}, status=200)

except Exception as error:
    return Response({
        "message": f"an error occurred : {error}"
    }, status=500)
```

NOTE : You can implement this logic in custom serializer but for save time i implement the logic in the views

Now u implement the task successfully using threading 

but here we have a problem , if we call the endpoint two or three times there is threads not working and has been killed by the operating system.

but why and how to solve it !

i will tell you why after implement the task with celery

Celery in Django

Celery do exactly what threading do but in a different pattern ,celery run tasks faraway from the server and takes the tasks you want to implement and add it in a Queue

Queue Fast Explanation:

i want you to imagine there is a list, and in this list the tasks that user do in the server , and this tasks is ordered by it's time the latest first and oldest after latest.

Then the tasks in this queue implemented depends on your workers in your

machine.

After created our task we need to add the celery in Django to be able to run the tasks in the background using celery.

configure celery in Django

install celery

```
pip install celery
```

Celery 5.3.x supports Django 2.2 LTS or newer versions. Please use Celery 5.2.x for versions older than Django 2.2 or Celery 4.4.x if your Django version is older than 1.11

follow these steps for right configuration

```
# myproject/celery.py
import os
from celery import Celery

# Set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myproject.settings')

app = Celery('myproject')

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django apps.
app.autodiscover_tasks()

@app.task(bind=True, ignore_result=True)
def debug_task(self):
    print(f'Request: {self.request!r}')
```

then we need to import the celery app in `myproject/__init__.py` file

```
# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app

__all__ = ('celery_app',)
```

final step is to add some options in settings.py file

```
CELERY_BROKER_URL = "redis://localhost:6379/" # you must have redis in your
CELERY_ACCEPT_CONTENT = ["application/json"]
CELERY_TASK_SERIALIZER = "json"
CELERY_TIMEZONE = "Africa/Cairo"
```

Finally celery configured successfully but still not working 😊.

we configure the celery successfully but we didn't create the functions that must perform the background task , and for doing that we go throw the app Dir and create a file called it myapp/tasks.py and write on it :

```
# myapp/tasks.py
from celery import shared_task
from django.conf import settings
from django.core.mail import send_mail

@shared_task
def send_email_by_celery(to_user_email,msg_content) :
    # subject of the email msg
    subject = "New message from admin"
    # the admin email [sender]
    from_email = settings.EMAIL_HOST_USER
    # this email must goes to [reciver]
    recipient_list = [to_user_email]
    # send_mail method to implement the logic
    send_mail(subject, msg_content, from_email, recipient_list)
```

the task is created successfully but we must run the celery server to implement

the tasks , just you need to type this command in the same project directory :

```
celery -A your_project_name worker --loglevel=info --pool=solo
```

Then let's write the logic in the views.py file

```
# myapp/views.py
from rest_framework import decorators, permissions

from django.contrib.auth.models import User
from rest_framework.response import Response
# import our celery method which we created in tasks.py file
from .tasks import send_email_by_celery

@decorators.api_view(["POST"])
@decorators.permissions_classes([permissions.IsAdminUser])
def SendMailView (request,user_id) :
    try :
        # catch the email message from client
        msg_content = request.data.get("message_content",None)

        # check the message is inserted by the client
        if msg_content is None :
            return Response({"message":"please insert the message"},status=400)

        # check if there user in the db has this id or not
        try :
            to_user = User.objects.get(id=user_id)
        except User.DoesNotExist:
            return Response({"message":"user with this id not found"},stauts=404)

        # send the email by celery
        to_user_email = to_user.email
        send_email_by_celery.delay(to_user_email,msg_content)

        return Response({"message":"email sent successfully"},status=200)

    except Exception as error :
        return Response({
            "message" : f"an error accoured : {error}"
        },status=500)
```

we successfully done the task 

You Can now decide the pros and cons of each background task performer , you see that how we use threading and how its configuration is more easy than celery and we also see the threading maybe kill the process because threading didn't manage the tasks like celery .

when i use threading ?

you can use it for create a simple background tasks not heavy tasks, but you can not deploy the project with threading because it maybe cause some issues in the future .

when i use celery ?

if you work in a big systems like CMS , E-commerce , etc., the perfect choice for background tasks is celery , maybe the configuration is so complex than threading but your project will become more and more powerful in the future and didn't cause any problem in the background tasks side.

Conclusion

the main thing that make your projects good, fast and efficient not the programming language or the framework , it depends on you and how you will create the project and what tools you will use and how you will use theses tools.

Follow me in [Github](#) or [Linkedin](#) .

[Django](#)[Celery Django](#)[Threading](#)[Python](#)[Backend](#)

[Follow](#)

Written by Radwan Hijazi

10 Followers · 5 Following

thinking then transform coffee to code.

Responses (2)



What are your thoughts?

[Respond](#)

Itay

Jul 5, 2024



Why not working with the async await keywords? The first solution you have seems extremely hard to maintain. Using asynchronous operations will increase your app performance



1



1 reply

[Reply](#)



Emad Abdel Halim Ahmed

Mar 28, 2024



Good luck brother



1

[Reply](#)

More from Radwan Hijazi



Radwan Hijazi

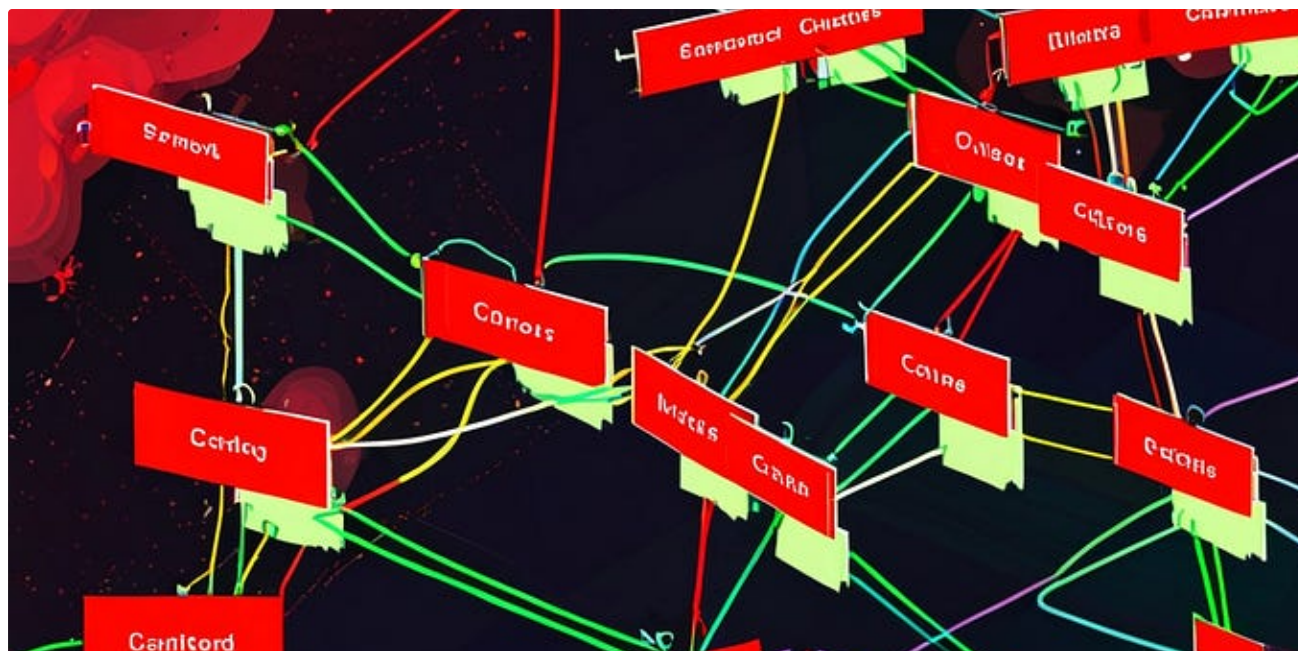
How could i build my own NO-SQL db used for caching from scratch (Using Python)?

we live in the 20th century where there is a very big and large data, and theses data need to be saved on databases, we have types of...

Aug 8, 2024

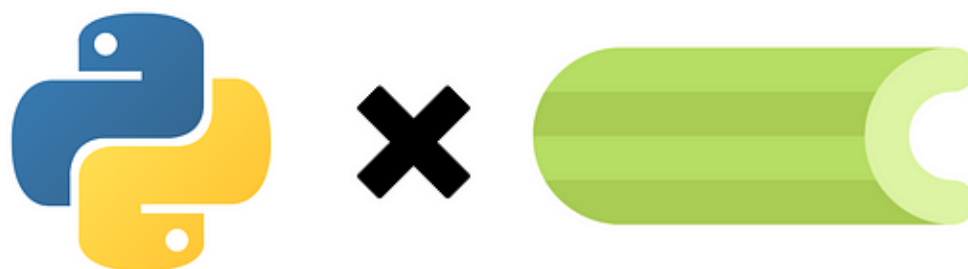
See all from Radwan Hijazi

Recommended from Medium



J.

18

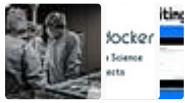


 15

1



Lists



Coding & Development

11 stories · 975 saves



Predictive Modeling w/ Python

20 stories · 1780 saves



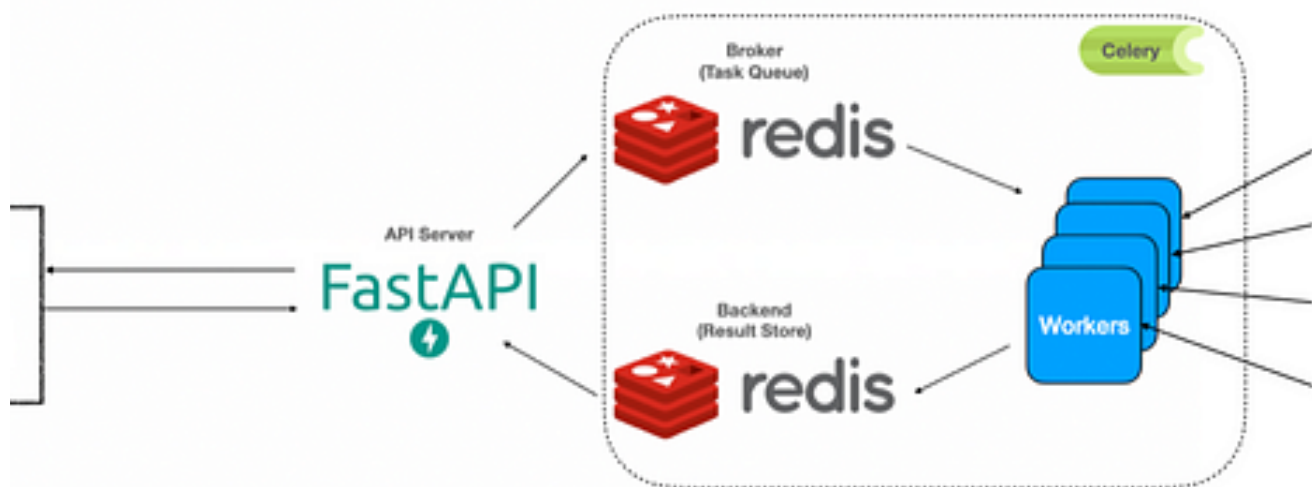
Practical Guides to Machine Learning

10 stories · 2153 saves



ChatGPT

21 stories · 942 saves



In Python in Plain English by Dhruv Ahuja

Mastering Background Job Queues with Celery, Redis, and FastAPI 🚀

💡 Heads Up! Click here to unlock this article for free if you're not a Medium member!



Oct 11, 2024



173



1



```
26 func (l *Listener) listenToQueue(ctx context.Context, topic string, queue MessageQueue) {
51
52 // set the callback function for this event to commit the message to kafka
53 wrapper.SetCallback(func(c context.Context) {
54     commitToKafka(c, queue, msg)
55 })
56
57 l.workerPool.Submit(func() {
58     if err := l.eventHandler.HandleEvent(ctx, wrapper); err != nil {
59         log.Error(ctx, "Error handling event", log.Tags{"topic": topic,
60             "error": err})
61         wrapper.DoCallback() // commit the message to kafka
62     }
63 })
64
65 // Wait for an OS signal to exit
66 sigchan := make(chan os.Signal, 1)
67 signal.Notify(sigchan, syscall.SIGINT, syscall.SIGTERM)
68 <-sigchan
69
70 log.Info(ctx, "Received signal, shutting down...", log.Tags{"topic": topic})
```

I added this code block but copied it blindly from the example given in the segmentio/kafka-go docs. Is this the correct way to handle this? Or is there a better way?


The signal handling code you've added is problematic in its current location. It's inside the `listenToQueue` method which runs as a goroutine for each queue, meaning you'll have multiple signal handlers competing with each other. Additionally, this signal handling should be at a higher level in your application.

Here's how to fix it:

```
func (l *Listener) Run(ctx context.Context) error {
    for topic, queue := range l.queues {
        go l.listenToQueue(ctx, topic, queue)
    }
    <-ctx.Done()
    return nil
}

func (l *Listener) listenToQueue(ctx context.Context, topic string, queue MessageQueue) {
    defer queue.Close()

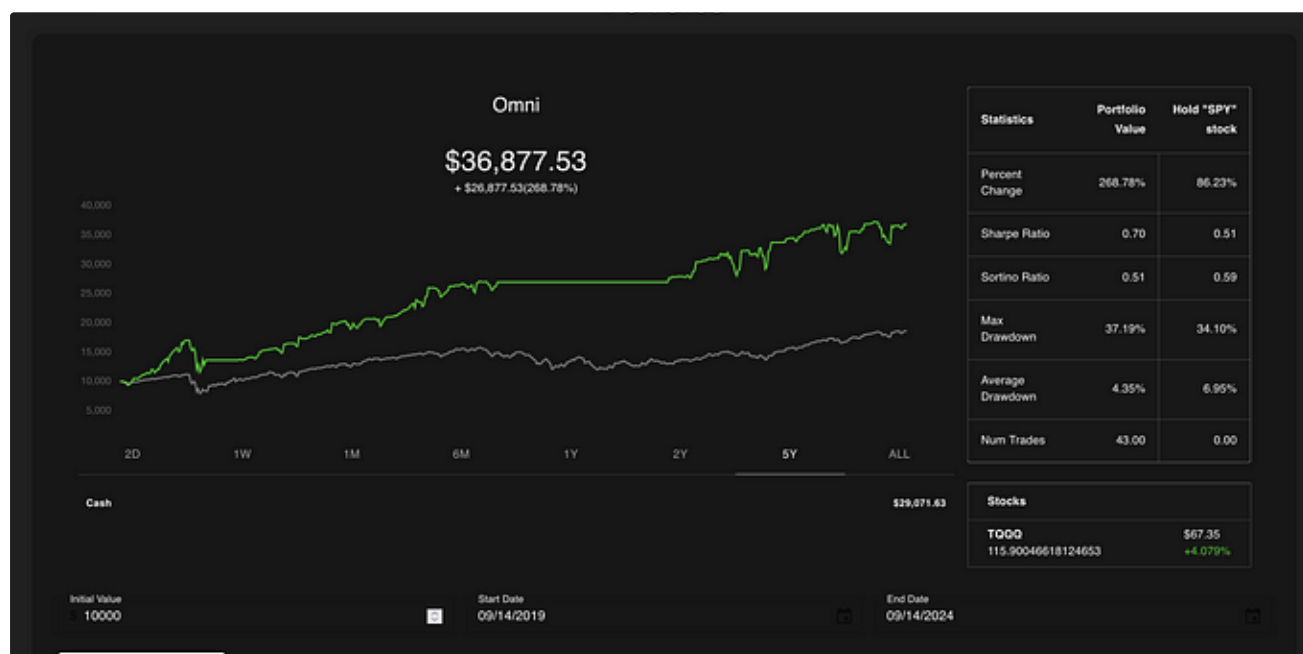
    for {
```


 In Level Up Coding by Jacob Bennett

The 5 paid subscriptions I actually use in 2025 as a Staff Software Engineer

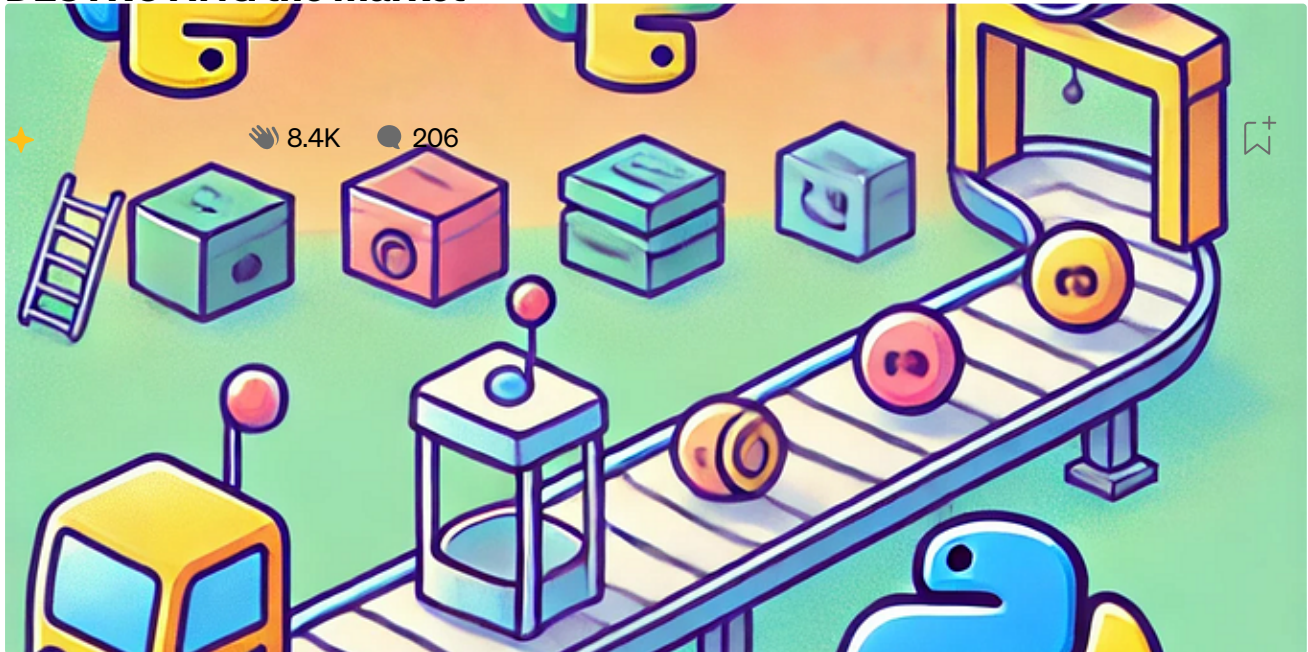
Tools I use that are cheaper than Netflix



★ Jan 7 🖱 3.9K 💬 95



 In DataDrivenInvestor by Austin Starks

I used OpenAI's o1 model to develop a trading strategy. It is DESTROYING the market





 8.4K  206



 Ashfaque Ali

Asynchronous Tasks in Django with Celery

When I first started using Django, I quickly fell in love with its simplicity and “batteries-included” philosophy. For small projects, it...

 Nov 10, 2024  38



See more recommendations