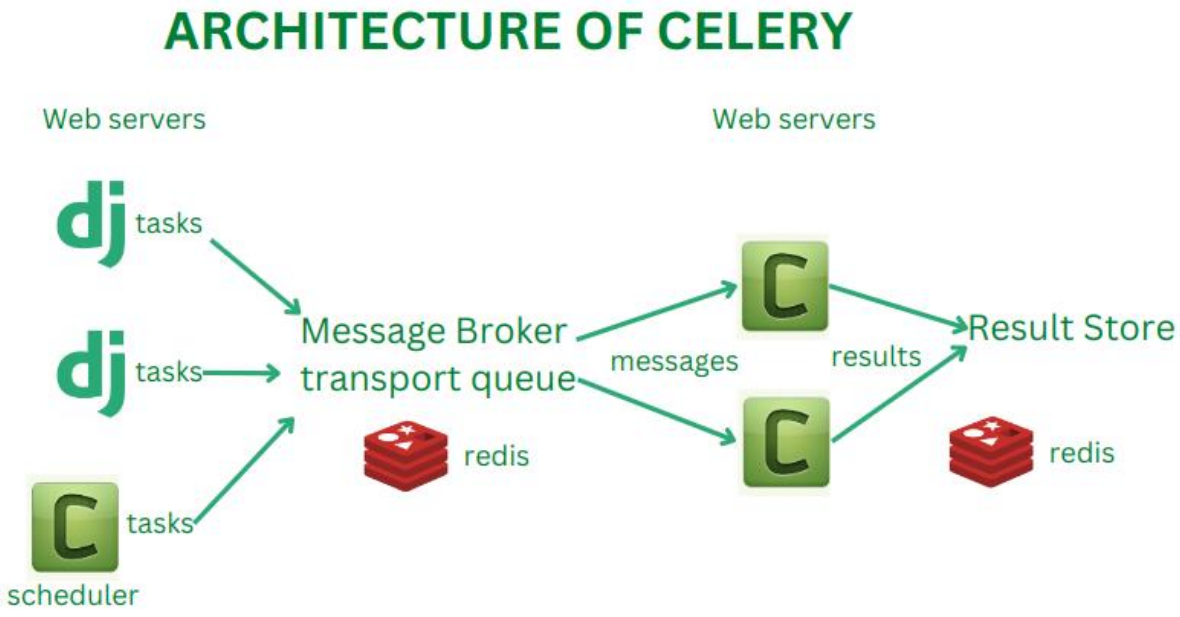
Django Celery

1. **What is celery?**

Celery is a task queue for executing work outside a python web application HTTP request-response cycle. A task queue's input is a unit of work called task. Dedicated workers constantly monitor task queue for new work to perform.

Celery is basically written in python, but the protocol can be implemented in any language.



- **Celery required a message transport to send and receive messages**
- **Supported brokers: RabbitMQ, Redis, AWS SQS**

- **Result store: Redis, Django ORM, Mongo DB, AWS S3, File system**
- **Serialization: Pickle, Json, Yaml, msgpack , Zlib, bzip2 compression**

Basic usage of Celery:

- **So first understand the usage of celery.**
- **Celery is mainly used for task scheduling.**
- **Task scheduling means if there is a piece of code which we want to run periodically.**
- **Second usage is to increase the distributed Processing, it means multiple processes can run at a time.**
- **Task chaining and workflows: Celery supports distributed processing, allowing you to distribute tasks across multiple worker instances or even on different machines. This is beneficial for handling large-scale and resource-intensive tasks.**
- **Asynchronous Task Execution: Celery allows you to offload time-consuming and resource-intensive tasks to be executed asynchronously in the background. This helps in keeping your web application responsive, as tasks are executed outside the normal request-response cycle.**
- **Scalability: By moving tasks to Celery, you can scale your application more effectively. Tasks can be distributed across multiple worker processes or even on different machines, enabling parallel processing of tasks.**
- **Error Handling: Celery provides mechanisms for handling task failures and retries. You can configure retry policies, set maximum retry limits, and define custom error handling strategies**

Some terminologies:

Task producer (Application): Produces or creates the task for celery.

Broker (Task queue): It queues the task. Generally, we use the Redis or RabbitMQ

Worker: In Celery, a distributed task queue framework for Python, a worker is a process or set of processes that execute tasks. Celery allows you to distribute the execution of tasks across multiple worker processes or even across multiple machines.

Pool: Decides who will perform the task – thread, child process. It is basically a group of worker processes.

Concurrency: It will decide the size of the pool.

Autoscale: Used to dynamically resize the pool based on load.

Note: When we start a celery worker, we can specify the pool, auotscale and concurrency etc.

Example: celery –A project_name worker –pool=prefork –concurrency=5 –atoscale=10 –l info

Types of Pool in celery:

- **Prefork**
- **Eventlet**
- **Gevent**
- **Solo**

 # The Prefork pool is better suited for CPU-bound tasks while the eventlet pool works better if you're I/O bound.

**** Issue while running celery worker:**

What makes Celery 4 incompatible with Windows is just the default prefork concurrency pool implementation. In other words, if your Celery-job-to-be-done copes well with eventlet, gevent or solo (solo is a blocking single-threaded execution pool), you can run Celery 4 on Windows with any of these execution pools.

Eventlet is basically superior in some cases.

- **Solo pool is for testing and debugging**

Basic flow of Celery:

- **First a task is triggered and queued using either "delay" method or "apply_async".**
- **Syntax: " Task_name.delay() "or "Task_name.apply_async(args= [])"**
- **When task is queued then the celery worker is started**
- **Command: "celery –A project_name worker –l info –P eventlet"**

- **Note: Here we have used eventlet pool because sometimes default pool of celery does not work properly with windows.**
- **Prefork pool is default pool which is more compatible with Linux.**

Django celery setup and basic project:

Celery documentation: [Django celery]

First create a virtual environment:

“py -m venv env”

Now start the virtual environment:

“env\scripts\activate”

Then create a project (My project name is “Project”):

“Django-admin startproject project”

Now install celery:

“pip install celery”

Now go to settings.py and add these to configure celery:

```
CELERY_BROKER_URL = 'redis://127.0.0.1:6379'
CELERY_RESULT_BACKEND = 'redis://127.0.0.1:6379'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_TASK_SERIALIZER = 'json'
CELERY_RESULT_SERIALIZER = 'json'
```

Note: Before configuring this make sure that you have installed Redis. You can also download its alternative like RabbitMQ also.

Now create a new file in your main directory in which settings.py file is located. Name the file as “celery.py”.

Now paste the content in that file:

```
import os
from datetime import timedelta
from celery import Celery
from celery.schedules import crontab
# Set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'project.settings')


app = Celery('project')


# 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}')
```

You can also find this in documentation of celery.

Now create an app:

“Python manage.py startapp home”

Add this app in settings.py file.

Now migrate the project and run the server.

Now the configurations are done.

Now go to your views.py in home app and create a function:

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.
import time
from .task import task_manage
from django.shortcuts import render
from celery.result import AsyncResult


def Home(request):
    # time.sleep(10)
    r=task_manage.delay(20,30)
    print("Result",r)

    return render(request,'index.html',{'status':r.status,'id':r})

def Task(request,id):
    result=AsyncResult(id)
```

```
    return render(request,'Home.html',{'result':result,'id':id})
```

Let us understand the code:

- **In home function we have called a method from task.py file. In this file we will define the tasks which we want to perform.**
- **When Home function is called then task manager function is called which takes 10 seconds to execute.**
- **So, we have used delay() to enqueue this task. So, when the Home function is called this task will simply enqueued and without taking 10 seconds for task_manage function it will execute its further code.**
- **So, let's create the Task.py file**

Create a task.py file in your app and paste the following code:

```
from celery import shared_task
import time


@shared_task
def task_manage(a,b):
    time.sleep(10)
    print(a+b)
    return a+b
```

Here the task_manage function will execute in 10 seconds and will return the result.

```
r=task_manage.delay(20,30)
```

In this line the ID of process is stored so by using this ID we can simply track the status and result of the process.

Now let's create two templates to see the result.

Create templates folder in app and add templates path in settings.py file:

Home.html

```
<html>
    <head>

        <title></title>

    </head>
    <body>

  <h1>Celery Result:</h1>
  <h2> ID: {{id}}</h2>
  <h2> state: {{result.state}}
    <h2>Status: {{result.status}}</h2>
    <p>Result: {{result.result}}</p>

    </body>
</html>
```

Index.html:

```
<html>
<head>
    <title>Celery Task Result</title>
</head>
<body>
    <h1>Celery Task Result</h1>
    <p><a href="task/{{id.id}}">Result: {{ id.id }}</a></p>
    <p>Status: {{status}}</p>
    <p>Result: {{status.result}}</p>
</body>
```

```
</html>
```

**First index.html will be rendered and it will show the status pending then you must click on ID, and you will be redirected to Home.html

When the task is completed then you must refresh Home.html page.

Now at the last setup urls:

Go to project folder where settings.py file is located

Create your urls like this:

```
from home.views import Home,Task


urlpatterns = [
    path('admin/', admin.site.urls),
    path('',Home),
    path('task/<str:id>',Task)
]
```

Now start the Django server:

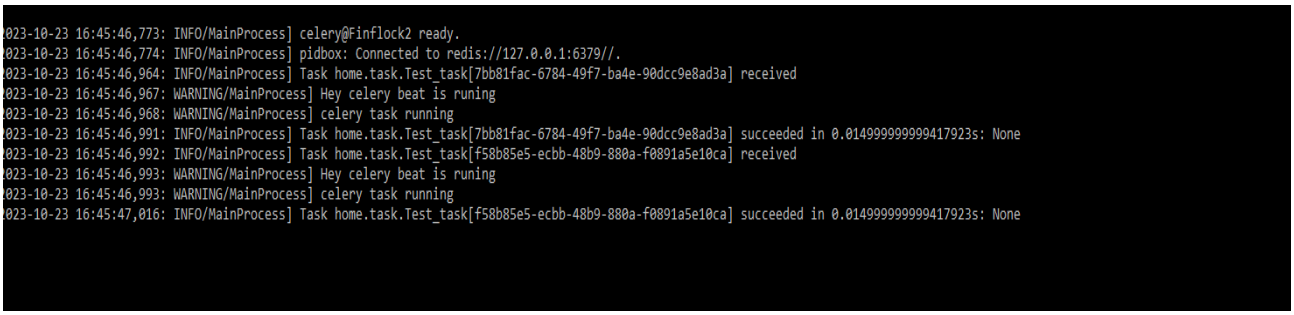
“python manage.py runserver”

Now start the celery workers:

“celery -A project worker –l info –P eventlet”

Note: we are using eventlet pool that is why we are adding “-P eventlet”.

Now go to 127.0.0.1 and refresh it again and again. Here you will see logs like that.



Now celery workers are working properly

This is how you can run multiple processes at a time.

Now let us assume that we want to run a piece of code at a particular time. Like we want to schedule a code then we have to use celery beat.

Celery beat will schedule the task and it will assign the task to worker at the assigned time.

Then the worker will execute the task and will return the result.

Now let's schedule the task:

So first create a task, so go to task.py and create a task:

```
@shared_task
def Test_task(id):
    print("Hey celery beat is runing")
    print(id)
```

Now go to celery.py and schedule the task like this:

```
app.conf.beat_schedule={
'Test-task-celery' :
      {
          'task':'home.task.Test_task',
          'schedule': 5,
          'args': ('celery task running',)
      }
}
```

**** Test-task-celery is the name of task. You can give it as you want.**

In schedule the time is given as second

Now restart the server and celery worker. Also, open and new cmd and start celery beat:

"celery -A project beat –l info"

Now you will see that the task is automatically assigned to the worker and executed by the worker.

Now come to advance scheduling:

We can use crontab for advance scheduling.

Crontab documentation: click here

Example:

```
app.conf.beat_schedule= {


'Test-task-celery' :
      {
          'task':'home.task.Test_task',
          'schedule': crontab(minute='*/1'),
          'args': ('celery task running',)
      }
}
```

****This is how you can schedule a code using celery.**

Celery docs: Click here

Article link: Click here

Configuration docs: Click here