Celery Python task queue

CONCEPTS

The caller only publishes a message; a separate worker process runs the task. No worker, no work.

Your app task.delay()
↓ publish
Broker RabbitMQ or Redis
↓ consume
Worker runs the task
↓ store
Result backend optional, for .get()

Run the worker from the project root so -A proj can import the app module.

Results are only kept if a result backend is configured; without one .get() fails.

WORKER

celery -A proj worker -l INFO
Start a worker

-c *<n>*
Number of pool processes

-Q *<q1>*,*<q2>*
Consume only these queues

-n *<name>*@%h
Set the node name

--autoscale=*<max>*,*<min>*
Grow and shrink the pool

-B
Embed beat in the worker

--time-limit *<s>*
Hard limit per task

DEFINE AND CALL

@app.task
Declare a task

@shared_task
Task without importing the app

@app.task(bind=True)
Get self for retry and request

add.delay(2, 3)
Queue with plain arguments

add.apply_async((2, 3), countdown=10)
Queue with options

apply_async(eta=*<datetime>*)
Run at an absolute time

DEFINE AND CALL (cont.)

apply_async(queue="*<q>*")
Send to a named queue

add.s(2, 3)
Signature: a task call as data

res.get(timeout=*<s>*)
Block for the result

res.ready() / res.state
Check for completion

CANVAS

chain(a.s(), b.s())
Pipe each result into the next

a.s() | b.s() | c.s()
Chain with the pipe operator

group(a.s(i) for i in r)
Run many tasks in parallel

chord(group)(callback.s())
Run a callback after the group

A chain passes each return value as the first argument of the next task; use .si() to ignore it.

RETRIES AND ERRORS

self.retry(exc=e, countdown=*<s>*)
Retry the task later

autoretry_for=(*<Exc>*,)
Retry automatically on these errors

max_retries=*<n>*
Give up after n retries

acks_late=True
Acknowledge after the task finishes

Use bind=True to get self.retry; the call raises Retry, so nothing after it runs.

ROUTING AND CONFIG

task_routes = {"*<task>*": {"queue": "*<q>*"}}**
Route tasks to queues

broker_url = "redis://localhost:6379/0"
Set the broker

result_backend = "redis://..."
Set the result backend

PERIODIC TASKS

celery -A proj beat -l INFO
Start the scheduler

beat_schedule = {...}
Declare the schedule

crontab(hour=7, minute=30)
Cron-style schedule

Run exactly one beat per schedule, or every task fires multiple times.

INSPECT AND CONTROL

celery -A proj status
Workers that answer

celery -A proj inspect active
Tasks running now

celery -A proj inspect reserved
Fetched, not started

celery -A proj inspect registered
Tasks each worker knows

celery -A proj inspect -d *<node>* **-j active**
Target one node, JSON output

celery -A proj control revoke *<id>*
Cancel a task

OPERATE

celery -A proj call *<task>* **-a '[2, 3]'**
Fire a task from the shell

celery -A proj purge -f
Drop all waiting messages, no prompt

kill -TERM *<pid>*
Warm shutdown: finish running tasks

RECIPES

Minimal app

```
from celery import Celery
app = Celery("proj",
broker="redis://localhost",
             backend="redis://
localhost")

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