Celery Python task queue

START HERE

`celery -A proj worker -l INFO`
Run a worker

`celery -A proj purge`
Drop waiting messages

`celery -A proj beat -l INFO`
Run the scheduler

`celery -A proj call <task>`
Fire a task by name

`add.delay(2, 3)`
Queue a task

`res.get(timeout=10)`
Wait for the result

`celery -A proj status`
Which workers are up?

`celery -A proj inspect active`
What is running now?

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

CONCEPTS

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

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
Needed when several workers run on one host.

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

`-B`
Embed beat in the worker
Single-instance setups only; never run two schedulers.

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

WORKER (cont.)

`--prefetch-multiplier <n>`
Messages reserved per process
Use 1 for long tasks so one process does not hoard them.

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

`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

`sig.set(queue="<q>")`
Change options on a signature

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

A chord needs a result backend, and a failing header task fails the callback.

RETRIES AND ERRORS

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

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

`retry_backoff=True`
Exponential delay between retries

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

RETRIES AND ERRORS (cont.)

`acks_late=True`
Acknowledge after the task finishes
The task may run twice if the worker dies, so make it idempotent.

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

`task_always_eager = True`
Run tasks inline in tests

`app.autodiscover_tasks()`
Find tasks.py in installed apps

PERIODIC TASKS

`celery -A proj beat -l INFO`
Start the scheduler

`beat_schedule = {...}`
Declare the schedule

PERIODIC TASKS (cont.)

`"schedule": 30.0`
Every 30 seconds

`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

OPERATE (cont.)

`celery -A proj purge -f`
Drop all waiting messages, no prompt
No undo; it affects every queue the app knows.

`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
```

Run and call

```
celery -A proj worker -l INFO
# in a Python shell
from proj import add
r = add.delay(2, 3)
r.get(timeout=10)   # 5
```

Retry with backoff

```
@app.task(bind=True,
autoretry_for=(IOError,),
          retry_backoff=True,
max_retries=5)
def fetch(self, url):
    ...
```

Fan out, then combine

```
from celery import chord, group
chord(group(fetch.s(u) for u in
urls))(merge.s())
```