EUROPYTHON 2011
FLORENCE, JUNE 20–26

Playing tasks with Django & Celery

Mauro Rocco
@fireantology

jamendo
open your ears

About me

- I'm a Web Developer
- Python, Javascript, PHP, Java/Android
- celery contributor (just one of the hundreds)

About Jamendo

- Jamendo is a community of free, legal and unlimited music published under Creative Commons licenses
- Free Music for users
- Popularity and earnings for artists
- Music licensing and background music at competitive prices for companies

Jamendo needs

- Multi-format music encoding
- Statistics (downloads, listens, reviews, stars, fb likes) on different units
- Music analysis trough external services
- Music qualification trough several sources
- Integration with third part services
- Common jobs (contract generations, certifications, bills, search index update)

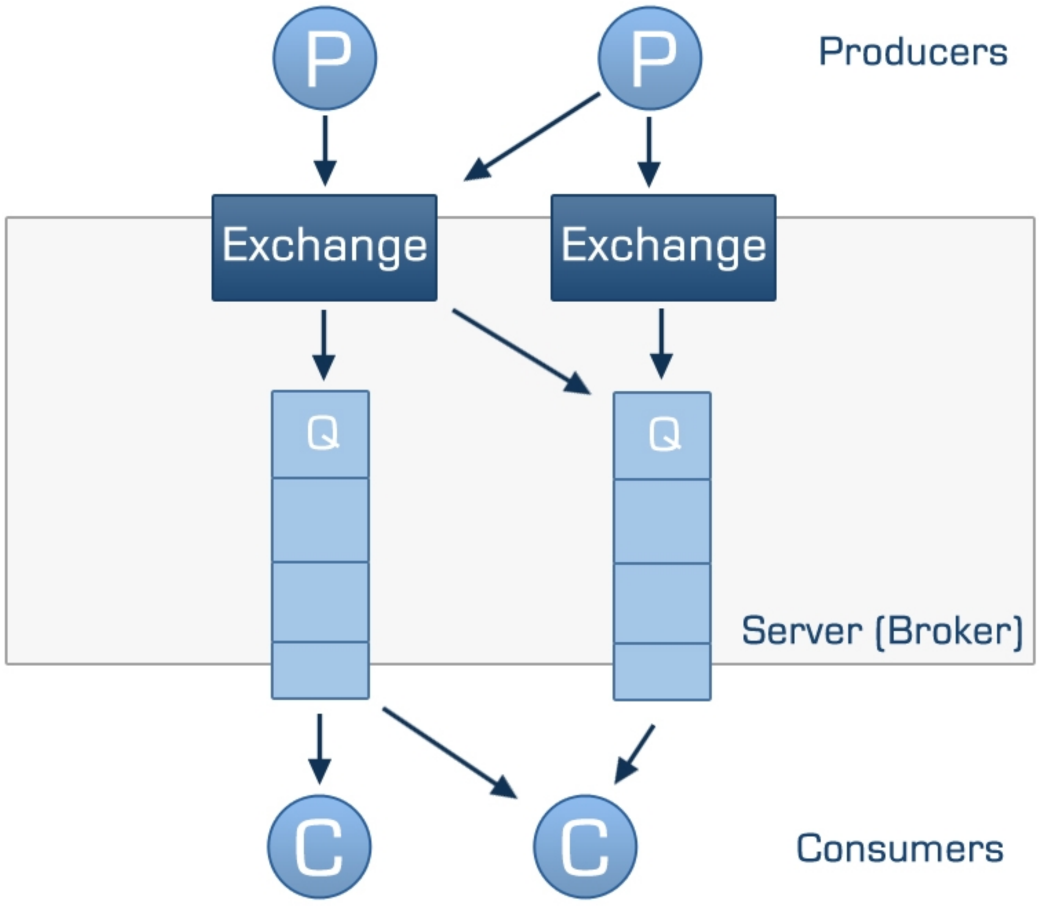
Celery

"Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well"

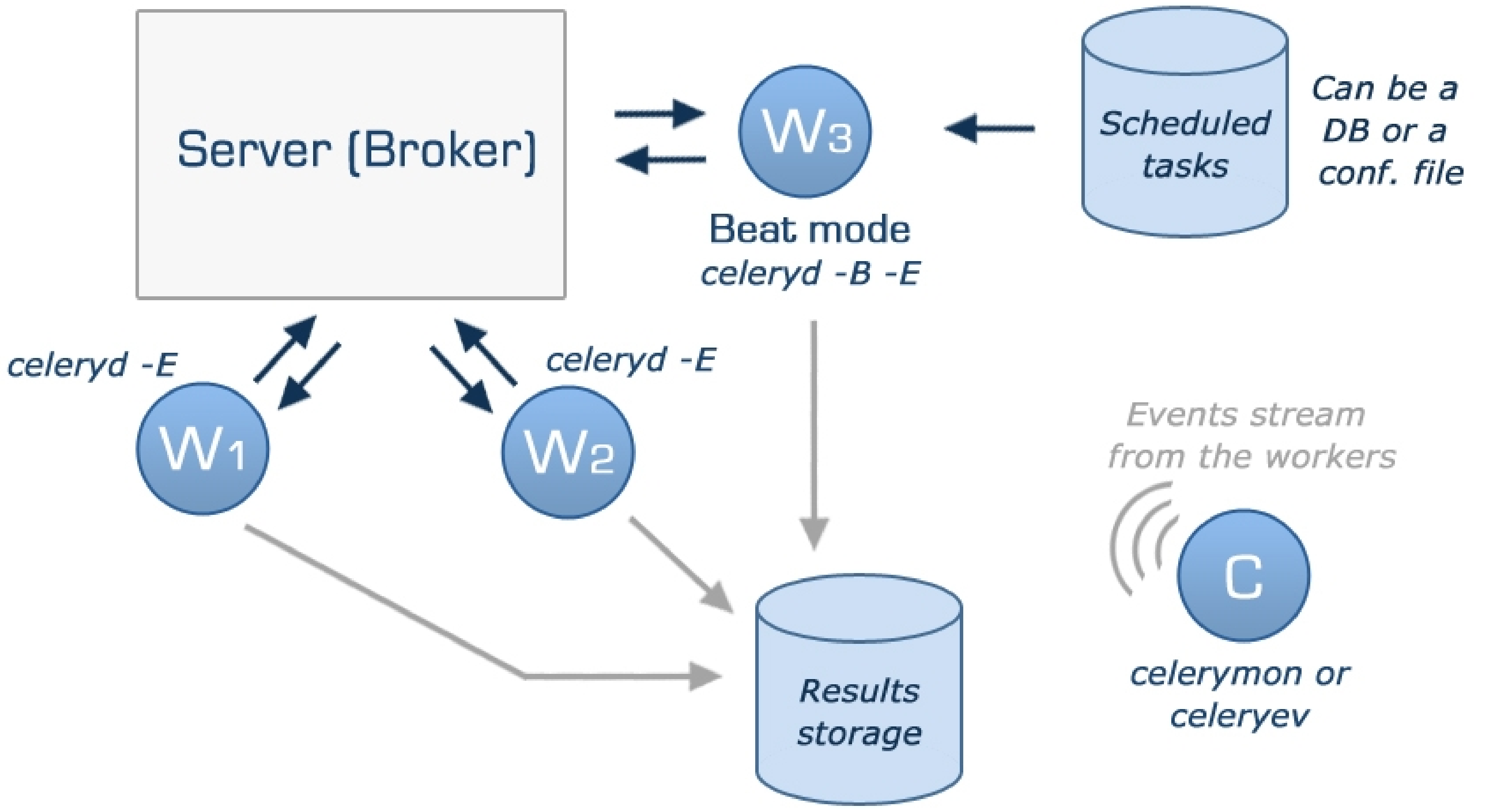
- Async & Sync processes
- Concurrency within a box
- Distributed (across machines)
- Scheduling (interval, cron, ...)
- Fault tolerant
- Subtask, Set of tasks
- Web monitoring (django-celery and others)

AMPQ

The Advanced Message Queuing Protocol (AMQP) is an open standard application layer protocol for Message Oriented Middleware.



Celery schema

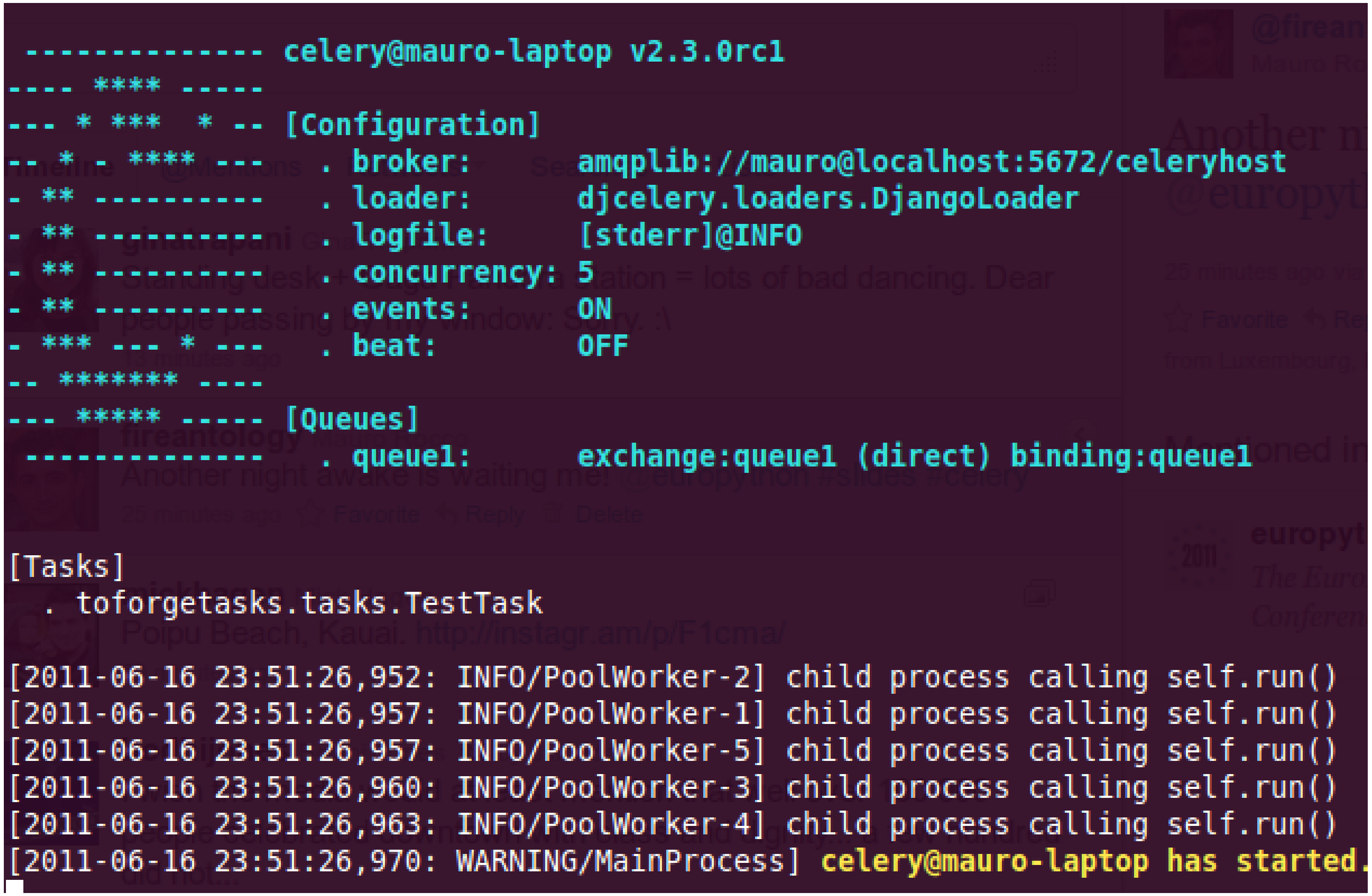


Celery worker

- Is the celery process that execute the tasks
- Can serve one or multiple queues
- Have a max number of tasks that can be executed at the same time
- Can be remotely controlled
- Have a great configuration option called MAX_TASK_PER_CHILD

```
$ celeryd -l INFO -c 5 -Q queue1 -E
```

Celery worker



```
 -------------- celery@mauro-laptop v2.3.0rc1
---- **** -----
--- * ***  * -- [Configuration]
-- * - **** ---   . broker:      amqplib://mauro@localhost:5672/celeryhost
- ** ----------   . loader:      djcelery.loaders.DjangoLoader
- ** ----------   . logfile:     [stderr]@INFO
- ** ----------   . concurrency: 5
- ** ----------   . events:      ON
- *** --- * ---   . beat:        OFF
-- ******* ----
--- ***** ----- [Queues]
 --------------   . queue1:      exchange:queue1 (direct) binding:queue1


[Tasks]
  . toforgetasks.tasks.TestTask

[2011-06-16 23:51:26,952: INFO/PoolWorker-2] child process calling self.run()
[2011-06-16 23:51:26,957: INFO/PoolWorker-1] child process calling self.run()
[2011-06-16 23:51:26,957: INFO/PoolWorker-5] child process calling self.run()
[2011-06-16 23:51:26,960: INFO/PoolWorker-3] child process calling self.run()
[2011-06-16 23:51:26,963: INFO/PoolWorker-4] child process calling self.run()
[2011-06-16 23:51:26,970: WARNING/MainProcess] celery@mauro-laptop has started.
```

Defining a simple task

```
from celery.decorators import task

@task
def make_money(how_much):
    logger = make_money.get_logger()
    logger.info("Congratulation, you earned %s$" % how_much)
    if how_much>1000000:
        return "Bora Bora"
    return "Keep working"
```

```
>>> result = make_money.delay(200)
>>> result.get()
“Keep working”
```

Retrying a task if something fails

```
from celery.decorators import task

@task
def make_money_real_life(how_much, wife=True):
    try:
        logger = make_money.get_logger()
        if wife:
            raise Exception("Incompatibility exception")
        logger.info("Congratulation, you earned %s$" % how_much)
        if how_much>1000000:
            return "Bora Bora"
        return "Keep working"
    except Exception,exc:
        make_money_real_life.retry(exc=exc,
                                   countdown=60,
                                   args=[how_much,False])
```

Task set example

Extract from a jamendo task that upload track metadata in xml format to an ftp server for music analysis

```
def run(self, setid=None, subtasks=None, **kwargs):
    …
    if not setid or not subtasks:
        …
        tasks = []
        for slice in slices:
            tasks.append(uploadTrackSlice.subtask((slice,folder_name)))

        job = TaskSet(tasks=tasks)
        task_set_result = job.apply_async()
        setid = task_set_result.taskset_id
        subtasks = [result.task_id for result in task_set_result.subtasks]
        self.incrementalRetry("Result not ready", args=[setid,subtasks])

    #Is a retry than we just have to check the results
    tasks_result = TaskSetResult(setid, map(AsyncResult,subtasks))
    if not tasks_result.ready():
        self.incrementalRetry("Result not ready", args=[setid,subtasks])
    else:
        if tasks_result.successful():
            return tasks_result.join()
        else:
            raise Exception("Some of the tasks was failing")
```

The Jamendo Task class

The way for define common behaviour to all your tasks is to override __call__ and after_return methods of the celery Task class

```
class JamTask(Task):

    def __call__(self, *args, **kwargs):
        """This method is in charge of call the run method of the task"""
        self.max_retries = 30
        self.sandbox = SandBox(self.name, self.request.id,
                               settings.PATH_SANDBOX, settings.DEBUG)
        self.taskLogger = TaskLogger(args, kwargs)
        self.taskLogger.__enter__()
        .
        .
        return self.run(*args, **kwargs)
    .
    .
    def after_return(self, status, retval, task_id, args, kwargs, einfo):
        """This method is called when the tasks end,
        on whatever return state"""
        self.taskLogger.__exit__(status, retval, args, kwargs, einfo)
        self.cleanTaskSandBox(status,kwargs)
        self.closeAllConnections()
```

Web Monitoring tools

- django-celery
  https://github.com/ask/django-celery/
- celery-pylons
  http://pypi.python.org/pypi/celery-pylons
- flask-celery
  https://github.com/ask/flask-celery/

django-celery

Task scheduling and monitoring trough the Django admin interface

- The celeryconf.py file is replaced by the django settings
- The CELERY_IMPORTS conf var is replaced by the Django INSTALLED_APPS

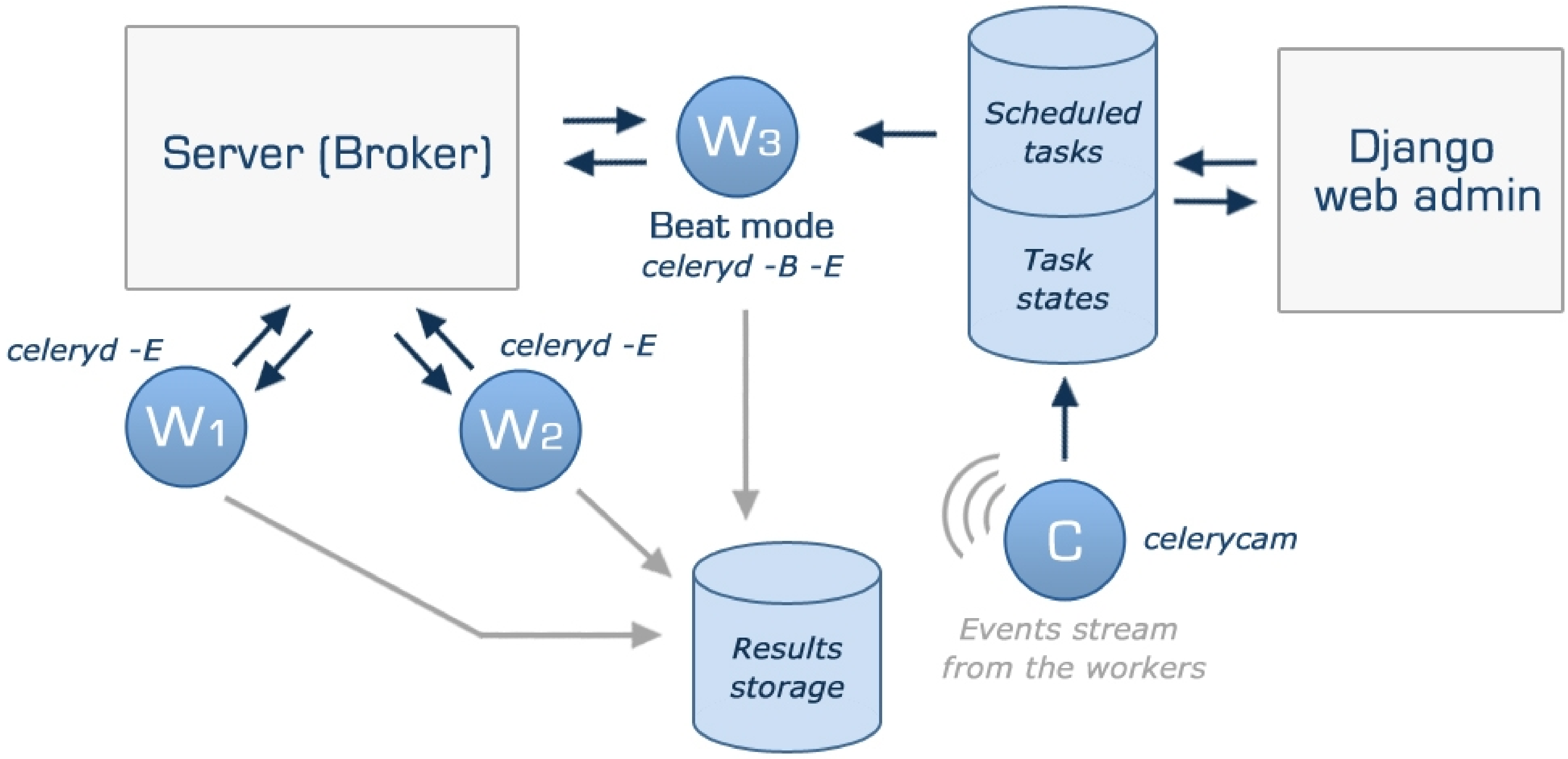
You run celery trough the manage.py of your project

```
$ python manage.py celeryd -l INFO -E
```

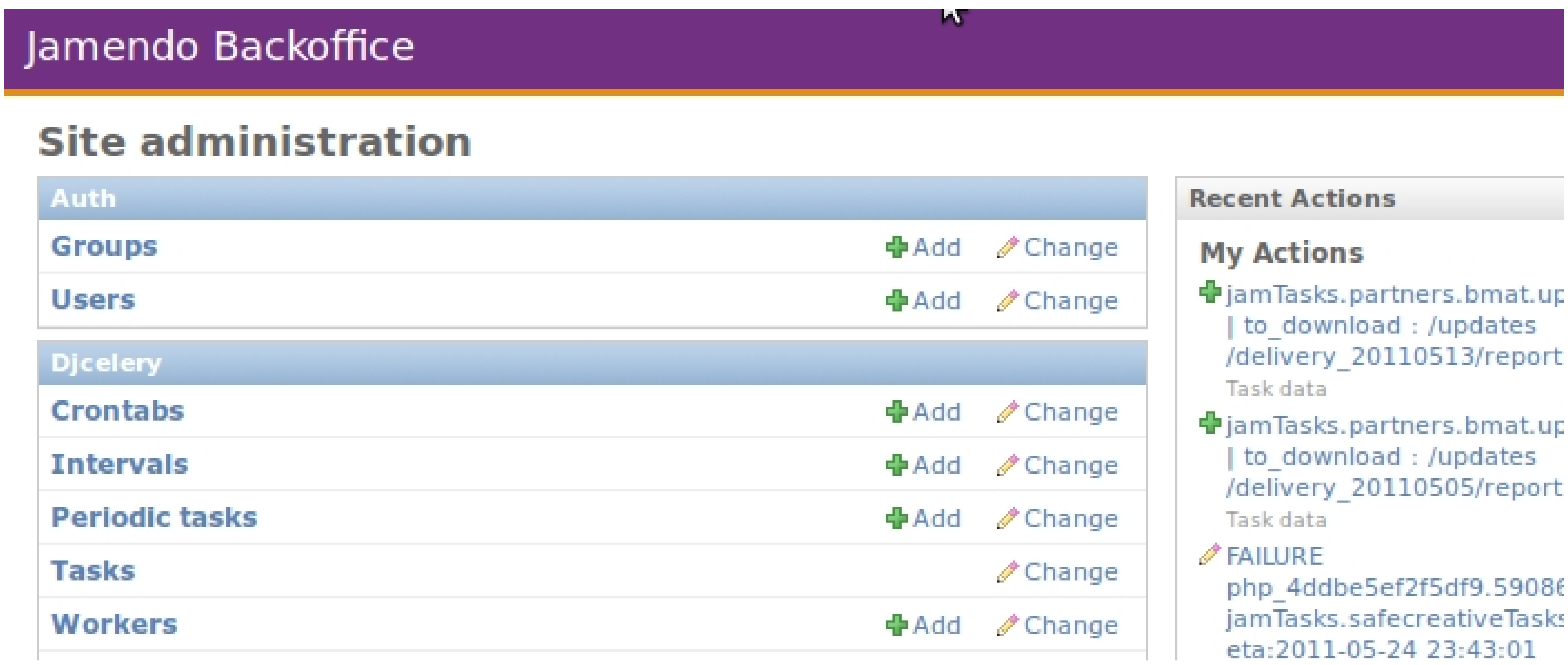
django-celery settings.py

```
INSTALLED_APPS += ("djcelery", )
.
.
import djcelery
djcelery.setup_loader()
.
.
CELERYBEAT_SCHEDULER = "djcelery.schedulers.DatabaseScheduler"
.
.
#standard celery conf vars (Broker settings, concurrency ,...)
```

django-celery schema



django-celery



Jamendo Backoffice

Site administration

Auth		
Groups	Add	Change
Users	Add	Change

Djcelery		
Crontabs	Add	Change
Intervals	Add	Change
Periodic tasks	Add	Change
Tasks		Change
Workers	Add	Change

Recent Actions

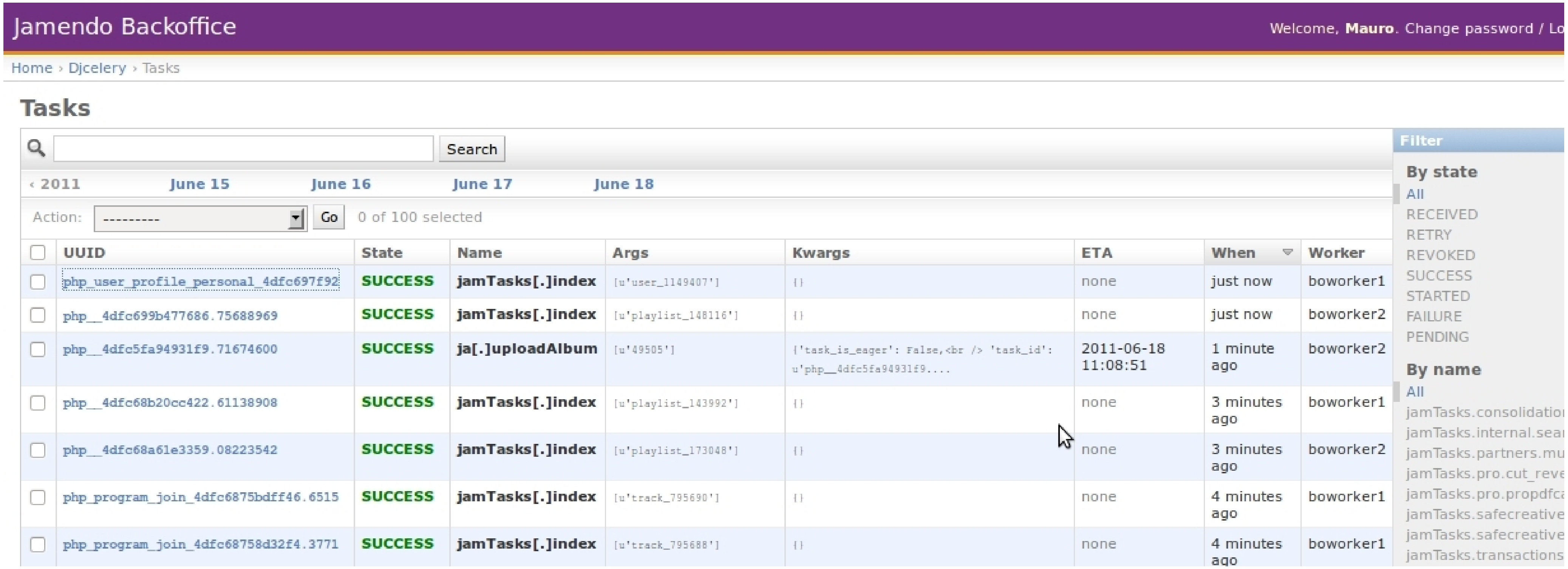
My Actions

jamTasks.partners.bmat.up
| to_download : /updates
/delivery_20110513/report
Task data

jamTasks.partners.bmat.up
| to_download : /updates
/delivery_20110505/report
Task data

FAILURE
php_4ddbe5ef2f5df9.5908
jamTasks.safecreativeTask
eta:2011-05-24 23:43:01

django-celery



Jamendo Backoffice

Welcome, **Mauro**. Change password / Lo

Home › Djcelery › Tasks

Tasks

Search

‹ 2011 June 15 June 16 June 17 June 18

Action: --------- Go 0 of 100 selected

	UUID	State	Name	Args	Kwargs	ETA	When	Worker
☐	php_user_profile_personal_4dfc697f92	SUCCESS	jamTasks[.]index	[u'user_1149407']	{}	none	just now	boworker1
☐	php__4dfc699b477686.75688969	SUCCESS	jamTasks[.]index	[u'playlist_148116']	{}	none	just now	boworker2
☐	php__4dfc5fa94931f9.71674600	SUCCESS	ja[.]uploadAlbum	[u'49505']	{'task_is_eager': False,<br /> 'task_id': u'php__4dfc5fa94931f9....	2011-06-18 11:08:51	1 minute ago	boworker2
☐	php__4dfc68b20cc422.61138908	SUCCESS	jamTasks[.]index	[u'playlist_143992']	{}	none	3 minutes ago	boworker1
☐	php__4dfc68a61e3359.08223542	SUCCESS	jamTasks[.]index	[u'playlist_173048']	{}	none	3 minutes ago	boworker2
☐	php_program_join_4dfc6875bdff46.6515	SUCCESS	jamTasks[.]index	[u'track_795690']	{}	none	4 minutes ago	boworker1
☐	php_program_join_4dfc68758d32f4.3771	SUCCESS	jamTasks[.]index	[u'track_795688']	{}	none	4 minutes ago	boworker1

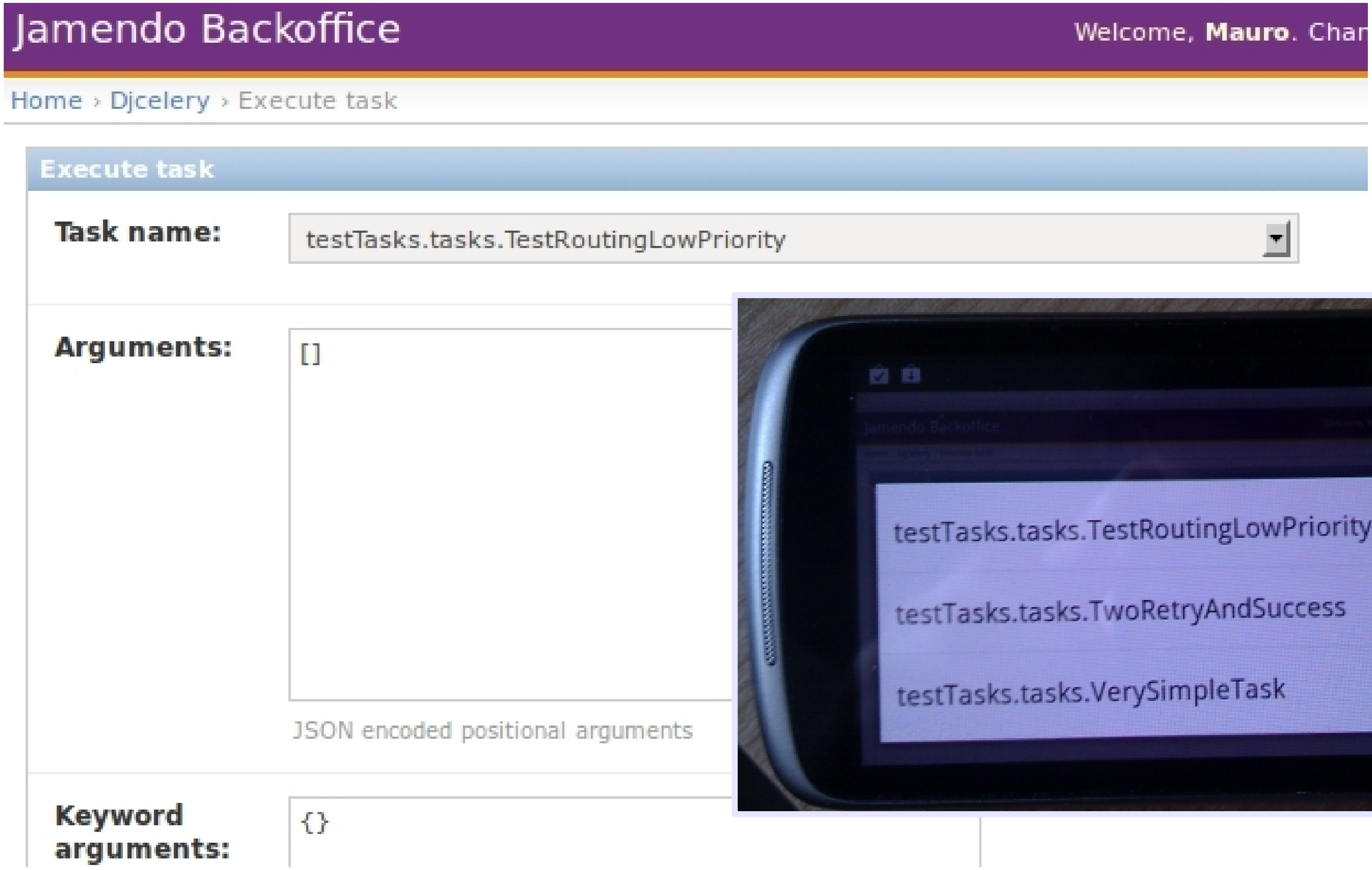
Filter

By state
- All
- RECEIVED
- RETRY
- REVOKED
- SUCCESS
- STARTED
- FAILURE
- PENDING

By name
- All
- jamTasks.consolidatio
- jamTasks.internal.sear
- jamTasks.partners.mu
- jamTasks.pro.cut_reve
- jamTasks.pro.propdfca
- jamTasks.safecreative
- jamTasks.safecreative
- jamTasks.transactions

Some little nice extensions

Execute tasks directly from the django admin interface



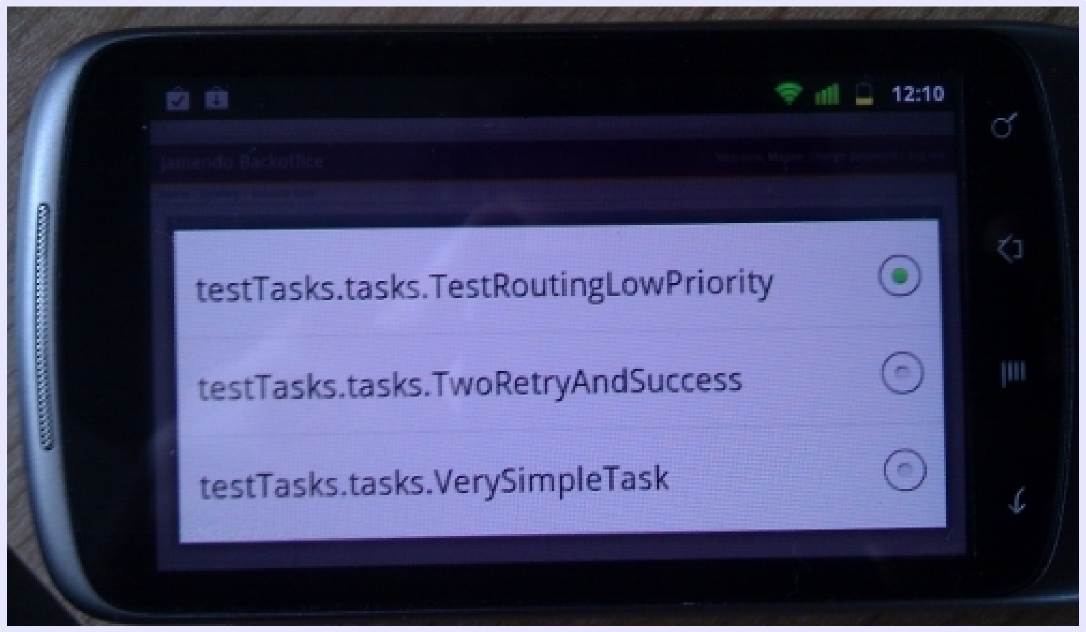
Jamendo Backoffice

Welcome, **Mauro**. Char

Home › Djcelery › Execute task

Execute task

Task name:	testTasks.tasks.TestRoutingLowPriority
Arguments:	[] JSON encoded positional arguments
Keyword arguments:	{}

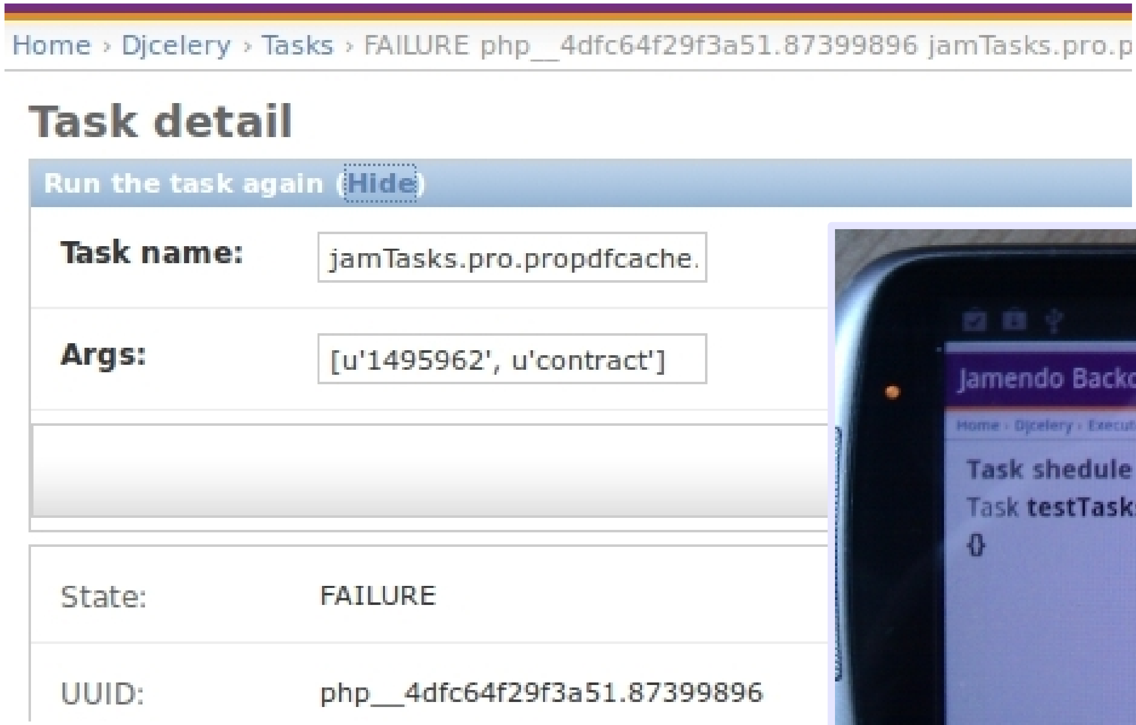


testTasks.tasks.TestRoutingLowPriority [✓]

testTasks.tasks.TwoRetryAndSuccess

testTasks.tasks.VerySimpleTask

Some little nice extensions

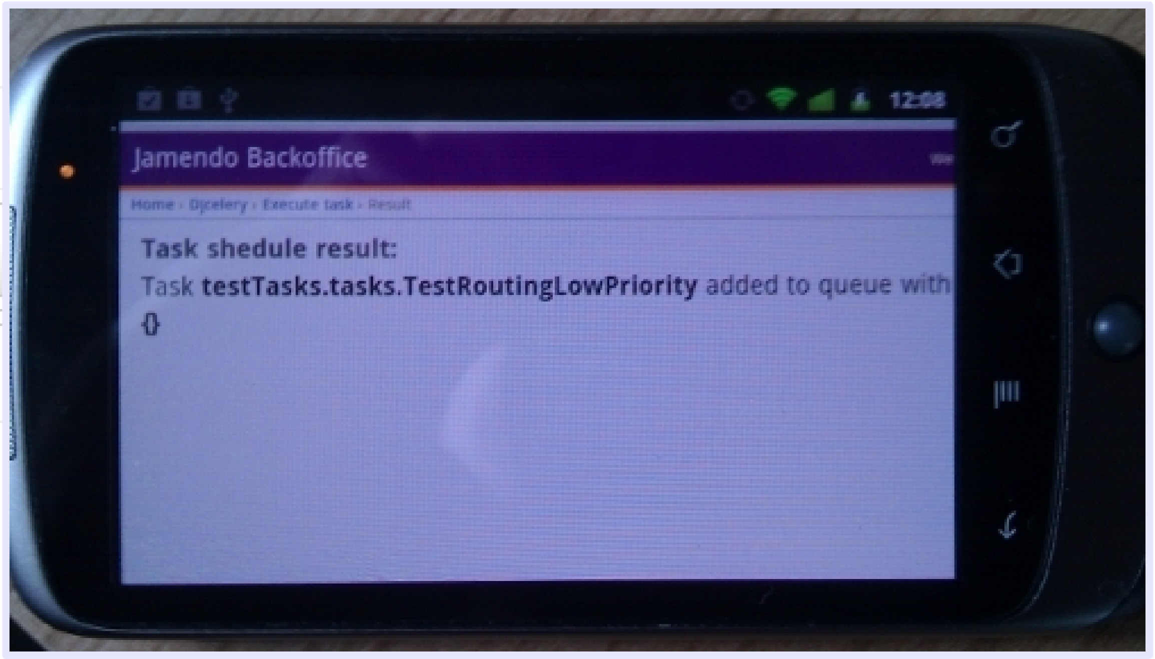

Home › Djcelery › Tasks › FAILURE php__4dfc64f29f3a51.87399896 jamTasks.pro.p

Task detail

Run the task again (Hide)

Task name:	jamTasks.pro.propdfcache.
Args:	[u'1495962', u'contract']

State:	FAILURE
UUID:	php__4dfc64f29f3a51.87399896



Task shedule result:

Task **testTasks.tasks.TestRoutingLowPriority** added to queue with {}

Jamendo needs UNIQUE tasks

A task is unique when can run only one instance of it at the same time in the whole cloud

- Rational utilization of shared resources
- Atomic access to sensitive resources

Our idea:

- Define a list of UNIQUE tasks in settings.py
- If a lock is found define the behaviour **retry** or **fail**
- Allow the possibility of define a task UNIQUE on arguments (same task type with different arguments can run)
- Our solution : **mongodb** for write and release locks.
- Best solution: cache, virtual file system ?

Unique tasks

```
UNIQUE_TASKS = {
    "searchengines.solr.index": { "retry_on_lock": False, "lock_on_type": True, },
    "stats.album.rebuild": { "retry_on_lock": True, "lock_on_type": False, },
}
```

On task start (method __call__)

```
self.taskConcurrency = None
if kwargs["task_name"] in settings.UNIQUE_TASKS:
    self.taskConcurrency = TaskConcurrency(kwargs,
                                           args,
                                           settings.UNIQUE_TASKS\
                                           [kwargs["task_name"]])
    if not self.taskConcurrency.canRun():
        if self.taskConcurrency.retry:
            self.incrementalRetry(Exception("Concurrency Exception"))
        else:
            raise Exception("Concurrency Exception")
```

On task end (method after_return)

```
if self.taskConcurrency:
    self.taskConcurrency.__exit__()
```

Celery logs

- The logger object is not unique, the same handler is added to different logs object
- Main Process logger, PoolWorker logger, TaskLogger
- The command logging.getLogger(“Celery”) give you back only the Main Process logger
- Extend logging features was a bit tricky until the last version

Centralized logging

- We give a very little contribute to celery by adding the signal **after_setup_logger and after_setup_task_logger** (the name are self explanatory)
- **after_setup_logger** is triggered after the build of the Main Process logger and after the build of each PoolWorker logger
- The signals give you back a log object, in this way you can add additional handler for implement a centralized logging
- In our specific case we are sending the logs of all workers to a syslog server that store log lines in a separated file.

Centralized logging

```
import logging
from celery.signals import after_setup_logger, after_setup_task_logger

def after_setup_logger_handler(sender=None, logger=None,
                               loglevel=None, logfile=None,
                               format=None, colorize=None,
                               **kwds):
    handler = logging.handlers.SysLogHandler(address=('syslogserver',
                                                      514))
    handler.setFormatter(logging.Formatter(format))
    handler.setLevel(logging.INFO)
    logger.addHandler(handler)

after_setup_logger.connect(after_setup_logger_handler)
after_setup_task_logger.connect(after_setup_logger_handler)
```

Thank you

http://www.celeryproject.org

QA