Celery Redbeat Documentation

Marc Sibson

Jun 13, 2019

Contents:

1 Introduction **1**
- 1.1 Why RedBeat? . . . 1
- 1.2 Getting Started . . . 1
- 1.3 Development . . . 2

2 Configuration **3**
- 2.1 `redbeat_redis_url` . . . 3
- 2.2 `redbeat_redis_use_ssl` . . . 3
- 2.3 `redbeat_key_prefix` . . . 3
- 2.4 `redbeat_lock_key` . . . 3
- 2.5 `redbeat_lock_timeout` . . . 3
- 2.6 Celery 3.x config names . . . 4
- 2.7 Sentinel support . . . 4

3 Creating Tasks **7**
- 3.1 Interval . . . 7
- 3.2 Crontab . . . 8

4 Design **9**
- 4.1 Scheduling . . . 9
- 4.2 Metadata . . . 9

5 Indices and tables **11**

CHAPTER 1

Introduction

alt Circle CI

RedBeat is a Celery Beat Scheduler that stores the scheduled tasks and runtime metadata in Redis.

1.1 Why RedBeat?

1. Dynamic live task creation and modification, without lengthy downtime
2. Externally manage tasks from any language with Redis bindings
3. Shared data store; Beat isn't tied to a single drive or machine
4. Fast startup even with a large task count
5. Prevent accidentally running multiple Beat servers

1.2 Getting Started

Install with pip:

```
pip install celery-redbeat
```

Configure RedBeat settings in your Celery configuration file:

```
redbeat_redis_url = "redis://localhost:6379/1"
```

Then specify the scheduler when running Celery Beat:

```
celery beat -S redbeat.RedBeatScheduler
```

RedBeat uses a distributed lock to prevent multiple instances running. To disable this feature, set:

```
redbeat_lock_key = None
```

1.3 Development

RedBeat is available on GitHub

Once you have the source you can run the tests with the following commands:

```
pip install -r requirements.dev.txt
py.test tests
```

You can also quickly fire up a sample Beat instance with:

```
celery beat --config exampleconf
```

CHAPTER 2

Configuration

You can add any of the following parameters to your Celery configuration (see Celery 3.x compatible configuration value names in below).

2.1 redbeat_redis_url

URL to redis server used to store the schedule, defaults to value of broker_url.

2.2 redbeat_redis_use_ssl

Additional SSL options used when using the `rediss` scheme in `redbeat_redis_url`, defaults to the values of broker_use_ssl.

2.3 redbeat_key_prefix

A prefix for all keys created by RedBeat, defaults to `'redbeat'`.

2.4 redbeat_lock_key

Key used to ensure only a single beat instance runs at a time, defaults to `'<redbeat_key_prefix>:lock'`.

2.5 redbeat_lock_timeout

Unless refreshed the lock will expire after this time, in seconds.

Defaults to five times of the default scheduler's loop interval (`300` seconds), so `1500` seconds (`25` minutes).

See the beat_max_loop_interval Celery docs about for more information.

2.6 Celery 3.x config names

Here are the old names of the configuration values for use with Celery 3.x.

Celery 4.x	Celery 3.x
`redbeat_redis_url`	`REDBEAT_REDIS_URL`
`redbeat_redis_use_ssl`	`REDBEAT_REDIS_USE_SSL`
`redbeat_key_prefix`	`REDBEAT_KEY_PREFIX`
`redbeat_lock_key`	`REDBEAT_LOCK_KEY`
`redbeat_lock_timeout`	`REDBEAT_LOCK_TIMEOUT`

2.7 Sentinel support

The redis connexion can use a Redis/Sentinel cluster. The configuration syntax is inspired from celery-redis-sentinel

```
# celeryconfig.py
BROKER_URL = 'redis-sentinel://redis-sentinel:26379/0'
BROKER_TRANSPORT_OPTIONS = {
    'sentinels': [('192.168.1.1', 26379),
                  ('192.168.1.2', 26379),
                  ('192.168.1.3', 26379)],
    'password': '123',
    'db': 0,
    'service_name': 'master',
    'socket_timeout': 0.1,
}

CELERY_RESULT_BACKEND = 'redis-sentinel://redis-sentinel:26379/1'
CELERY_RESULT_BACKEND_TRANSPORT_OPTIONS = BROKER_TRANSPORT_OPTIONS
```

Some notes about the configuration:

- note the use of `redis-sentinel` schema within the URL for broker and results backend.
- hostname and port are ignored within the actual URL. Sentinel uses transport options `sentinels` setting to create a `Sentinel()` instead of configuration URL.
- `password` is going to be used for Celery queue backend as well.
- `db` is optional and defaults to `0`.

If other backend is configured for Celery queue use `REDBEAT_REDIS_URL` instead of `BROKER_URL` and `REDBEAT_REDIS_OPTIONS` instead of `BROKER_TRANSPORT_OPTIONS`. to avoid conflicting options. Here follows the example::

```
# celeryconfig.py
REDBEAT_REDIS_URL = 'redis-sentinel://redis-sentinel:26379/0'
REDBEAT_REDIS_OPTIONS = {
    'sentinels': [('192.168.1.1', 26379),
                  ('192.168.1.2', 26379),
```

(continues on next page)

(continued from previous page)

```
                  ('192.168.1.3', 26379)],
    'password': '123',
    'service_name': 'master',
    'socket_timeout': 0.1,
    'retry_period': 60,
}
```

If `retry_period` is given, retry connection for `retry_period` seconds. If not set, retrying mechanism is not triggered. If set to `-1` retry infinitely.

CHAPTER 3

Creating Tasks

You can use Celery's usual way to define static tasks or you can insert tasks directly into Redis. The config options is called beat_schedule, e.g.:

```
app.conf.beat_schedule = {
    'add-every-30-seconds': {
        'task': 'tasks.add',
        'schedule': 30.0,
        'args': (16, 16)
    },
}
```

On Celery 3.x the config option was called CELERYBEAT_SCHEDULE.

The easiest way to insert tasks from Python is it use `RedBeatSchedulerEntry()`:

```
interval = celery.schedules.schedule(run_every=60)  # seconds
entry = RedBeatSchedulerEntry('task-name', 'tasks.some_task', interval, args=['arg1',␣
↪2])
entry.save()
```

Alternatively, you can insert directly into Redis by creating a new hash with a key of `<redbeat_key_prefix>:task-name`. It should contain a single key `definition` which is a JSON blob with the task details.

3.1 Interval

An interval task is defined with the JSON like:

```
{
    "name" : "interval example",
    "task" : "tasks.every_5_seconds",
    "schedule": {
```

(continues on next page)

(continued from previous page)

```
        "__type__": "interval",
        "every" : 5, # seconds
        "relative": false, # optional
    },
    "args" : [  # optional
        "param1",
        "param2"
    ],
    "kwargs" : {  # optional
        "max_targets" : 100
    },
    "enabled" : true,  # optional
}
```

3.2 Crontab

An crontab task is defined with the JSON like:

```
{
    "name" : "crontab example",
    "task" : "tasks.daily",
    "schedule": {
        "__type__": "crontab",
        "minute" : "5", # optional, defaults to *
        "hour" : "*", # optional, defaults to *
        "day_of_week" : "monday", # optional, defaults to *
        "day_of_month" : "*/7", # optional, defaults to *
        "month_of_year" : "[1-12]", # optional, defaults to *
    },
    "args" : [  # optional
        "param1",
        "param2"
    ],
    "kwargs" : {  # optional
        "max_targets" : 100
    },
    "enabled" : true,  # optional
}
```

CHAPTER 4

Design

At its core RedBeat uses a Sorted Set to store the schedule as a priority queue. It stores task details using a hash key with the task definition and metadata.

The schedule set contains the task keys sorted by the next scheduled run time.

For each tick of Beat

1. get list of due keys and due next tick
2. retrieve definitions and metadata for all keys from previous step
3. update task metadata and reschedule with next run time of task
4. call due tasks using async_apply
5. calculate time to sleep until start of next tick using remaining tasks

4.1 Scheduling

Assuming your *redbeat_key_prefix* config values is set to *'redbeat:'* (default) you will also need to insert the new task into the schedule with:

```
zadd redbeat::schedule 0 new-task-name
```

The score is the next time the task should run formatted as a UNIX timestamp.

4.2 Metadata

Applications may also want to manipulate the task metadata to have more control over when a task runs. The meta key contains a JSON blob as follows:

```
{
    'last_run_at': {
        '__type__': 'datetime',
        'year': 2015,
        'month': 12,
        'day': 29,
        'hour': 16,
        'minute': 45,
        'microsecond': 231
    },
    'total_run_count'; 23
}
```

For instance by default `last_run_at` corresponds to when Beat dispatched the task, but depending on queue latency it might not run immediately, but the application could update the metadata with the actual run time, allowing intervals to be relative to last execution rather than last dispatch.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search