Cronster Documentation

Release 0.1.1

Florian Einfalt

Feb 04, 2018

Contents

Contents:

CHAPTER 1

Installation

To install `cronster`, type:

```
$ pip install cronster
```

CHAPTER 2

Getting Started

2.1 Install Redis

To get started with `cronster`, install `Redis` on your computer (macOS & Homebrew in this example):

```
$ brew install redis
$ brew services start redis
```

Digital Ocean has a great tutorial on how install Redis on Ubuntu.

2.2 Run the crawler

Assuming `Redis` is running on your machine in default configuration and you want to crawl from `~/projects`, run the following command:

```
$ cronster_crawler -r ~/projects
```

The full help output explains the other options of the CLI application. It is possible to change certain parameters should `Redis` on a different host/port or should you want to adjust the interval between crawls:

```
Usage: cronster_crawler [OPTIONS]

Options:
  -r, --root TEXT           Crawling root, default: the current working directory
  -h, --cache-host TEXT     Cache host, default: localhost
  -p, --cache-port INTEGER  Cache port, default: 6379 (Redis default)
  -i, --interval INTEGER    Crawling interval, default: 2 seconds
  --help                    Show this message and exit.
```

2.3 Run the scheduler

Assuming `Redis` is running on your machine in default configuration, run the following command:

```
$ cronster_scheduler
```

The full help output explains the other options of the CLI application. It is possible to change `Redis`-related parameters should `Redis` on a different host or port:

```
Usage: cronster_scheduler [OPTIONS]

Options:
  -h, --cache-host TEXT     Cache host, default: localhost
  -p, --cache-port INTEGER  Cache port, default: 6379 (Redis default)
  --help                    Show this message and exit.
```

2.4 Write a `crontab` file

Write the following `YAML` file to `~/projects/crontab` (or any other location anywhere in the hierarchy under your `root` location):

```
test_job:
    cmd: echo "Hello, World!"
    schedule: "*/5 * * * *"
```

You should see the job being picked up by the crawler on the next crawl and should see the scheduler run the job every five minutes.

CHAPTER 3

API Documentation

3.1 crawler

class cronster.crawler.**CronsterCrawler**(*root*, *cache_host*, *cache_port*, *interval*)

Bases: object

Cronster crawler class. Crawl the file system recursively for `crontab` files, read the contents and store a list of *CronsterJob* in a Redis cache.

__init__(*root*, *cache_host*, *cache_port*, *interval*)

Initialise a *CronsterCrawler*.

Parameters

- **root** (*str*) – File system root to crawl
- **cache_host** (*str*) – Host that serves the Redis cache
- **cache_port** (*int*) – Port on the host that exposes the Redis service
- **interval** (*int*) – Time between crawls in seconds

crawl()

Recursively crawl the file system from `root` in a given `interval`. Add *CronsterJob* from `crontab` files to the cache as a JSON string.

display_crontabs()

Print the current cache content to the console in tabulated form.

get_crontab_data(*crontab*)

Given a `crontab` file path, load and return the *CronsterJob* contained in the file.

Parameters **crontab** (*str*) – Crontab file path

Returns Jobs

Return type list

Example output:

```
[
    {
        "name": "job_name",
        "cmd": "echo $PATH",
        "schedule": "* * * * *",
        "path": "/path/to/crontab/file",
        "hash": "dc8a776c99d9b8ab97550e87c857dc959a857c5b"
    }
]
```

3.2 scheduler

class cronster.scheduler.**CronsterJob**(*job_name*, *job_cmd*, *job_schedule*, *job_path*, *job_hash*)

Bases: object

Cronster job class. Representation of an individual cronster job.

__init__(*job_name*, *job_cmd*, *job_schedule*, *job_path*, *job_hash*)

Initialise a *CronsterJob*.

Parameters

- **job_name** (*str*) – Job name
- **job_cmd** (*str*) – Job command
- **job_schedule** (*str*) – Job schedule in cron format, e.g. `*/5 * * * *`
- **job_path** (*str*) – Job's crontab file path
- **job_hash** (*str*) – Job hash

__lt__(*other*)

_execute_command()

Execute the job's command as a subprocess.

cmd

Returns Job command

Return type str

cron

Returns Job schedule

Return type str

hash

Returns Job hash

Return type str

is_due

Returns Whether or not the job is due to be run.

Return type bool

name

Returns Job name

Return type str

path

Returns Job crontab file path

Return type str

run()
Run the job and schedule the next run.

schedule()
Calculate the next run time and schedule the job to run.

status

Returns Job status

Return type bool

class cronster.scheduler.**CronsterScheduler**(*cache_host*, *cache_port*)
Bases: object

Cronster scheduler class. Load jobs from a Redis cache and run any number of *CronsterJob* based on their schedule.

__init__(*cache_host*, *cache_port*)
Initialise a *CronsterScheduler*.

Parameters

- **cache_host** (*str*) – Host that serves the Redis cache
- **cache_port** (*int*) – Port on the host that exposes the Redis service

clear()
Clear the job queue.

run_pending()
Run all pending jobs.

start()
Start the scheduler. Jobs will be run according to their schedule.

status()
Return the current status of all scheduler jobs.

Returns Job data

Return type tuple

stop()
Stop the scheduler. Jobs will not be running regardless of their schedule.

update()
Load the current cache contents, add jobs or change jobs' statuses.

class cronster.scheduler.**CronsterSchedulerPrompt**(*scheduler*)
Bases: cmd.Cmd

Cronster command prompt class. Implement CLI commands to control the attached *CronsterScheduler*.

__init__(*scheduler*)
Initialise a *CronsterSchedulerPrompt*.

Parameters **scheduler** (*CronsterScheduler*) – Scheduler to control

do_clear(*args*)
Invoke *cronster.scheduler.CronsterScheduler.clear()* to clear the job queue.

do_exit(*args*)
Close the scheduler.

do_start(*args*)
Invoke *cronster.scheduler.CronsterScheduler.start()* to start the scheduler.

do_status(*args*)
Invoke *cronster.scheduler.CronsterScheduler.status()* and print the status information to the console.

do_stop(*args*)
Invoke *cronster.scheduler.CronsterScheduler.stop()* to stop the scheduler.

do_update(*args*)
Invoke *cronster.scheduler.CronsterScheduler.update()* to force a job update form the cache.

cronster.scheduler.**_update_loop**(*scheduler*)
Run an infinite update/run loop.

Parameters **scheduler** (*CronsterScheduler*) – Scheduler to run

cronster.scheduler.**run_scheduler**(*cache_host*, *cache_port*)
Instantiate and run a *CronsterScheduler*. Run its run/update loop in a separate thread.

Parameters

- **cache_host** (*str*) – Host that serves the Redis cache
- **cache_port** (*int*) – Port on the host that exposes the Redis service

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

C

D

G

H

I

N

P

R

S

U