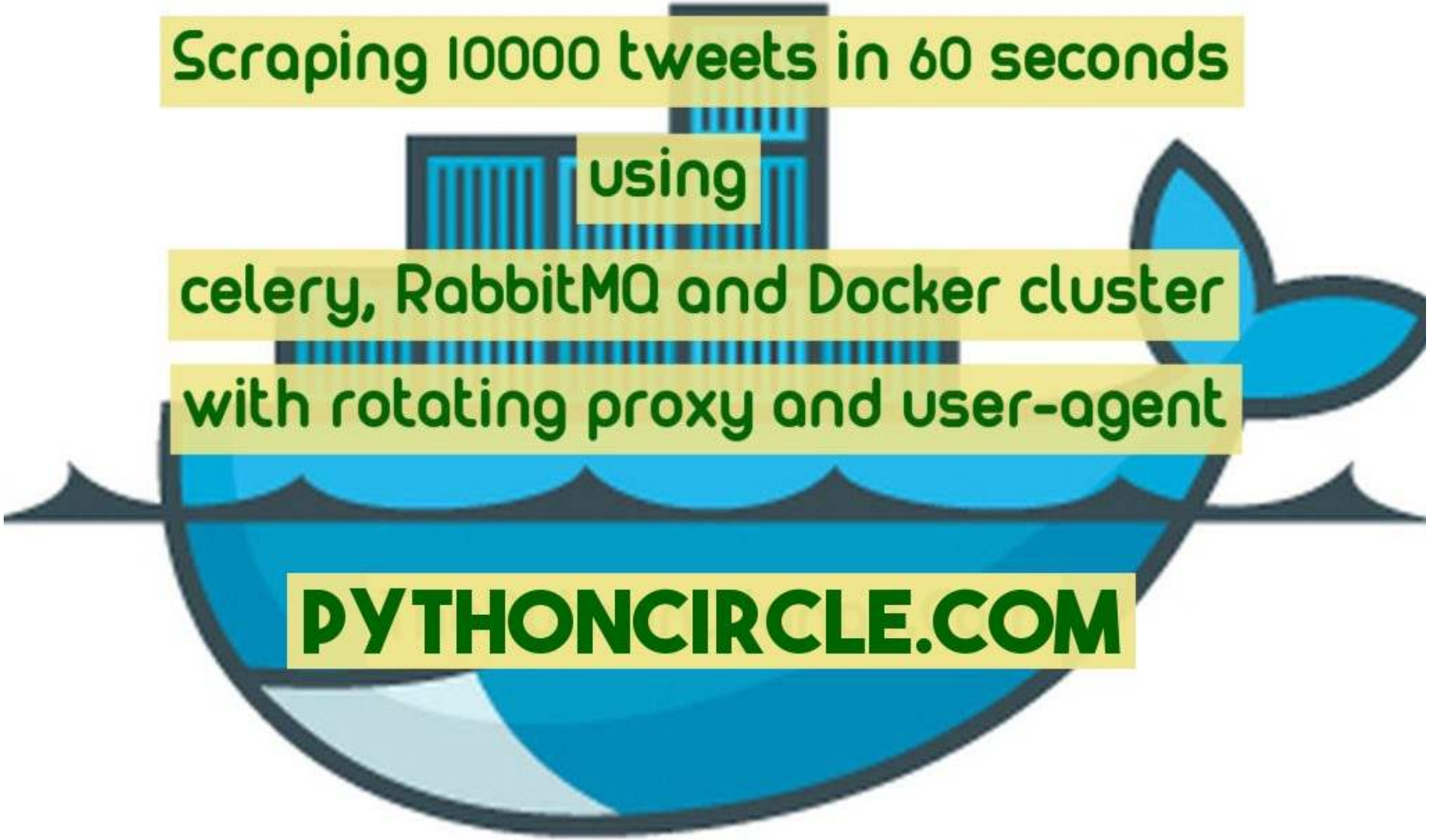
Scraping 10000 tweets in 60 seconds using celery, RabbitMQ and Docker cluster with rotating proxy (/post/518/scraping-10000-tweets-in-60-seconds-using-celery-rabbitmq-and-docker-cluster-with-rotating-proxy/)

beautifulsoup (/tag/beautifulsoup/) scraping (/tag/scraping/) proxy (/tag/proxy/) tor (/tag/tor/) docker (/tag/docker/) rabbitmq (/tag/rabbitmq/) 0 7301


Scraping 10000 tweets in 60 seconds
using
celery, RabbitMQ and Docker cluster
with rotating proxy and user-agent
PYTHONCIRCLE.COM

In previous articles we used requests and BeautifulSoup to scrape the data. Scraping data this way is slow (Using selenium is even slower).

Sometimes we need data quickly. But if we try to speed up the process of scraping data using multi-threading or any other technique, we will start getting http status 429 i.e. too may requests. We might get banned from the site as well.

Purpose of this article is to scrape lots of data quickly without getting banned and we will do this by using docker cluster of celery and RabbitMQ along with Tor.

For this to achieve we will follow below steps:

1. Install docker and docker-compose
2. Download the boilerplate code to setup docker cluster in 5 minutes
3. Understanding the code
4. Experiment with docker cluster
5. Update the code to download tweets
6. Using Tor to avoid getting banned.
7. Speeding up the process by increasing workers and concurrency

Note: This article is for educational purpose only. Do not send too many requests to any server. Respect the robot.txt file. Use API if possible.

Let's start.

Installing docker and docker-compose:

- We will be using docker version Docker version 17.12.0-ce, build c97c6d6 . Install docker by following the instructions on docker's official page (https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce-1).
- Docker-compose version being used is docker-compose version 1.8.0, build unknown. Install docker compose by following instructions on their page (https://docs.docker.com/compose/install/#install-compose).

Boilerplate code:

Clone the code from this Github repository (https://github.com/anuragrana/celery_rabbitmq_docker_cluster). README file is added to help you start with cluster.

Dockerfile used to build the worker image is using python:3 docker image.

Directory structure of code:

```
.
--- celery_main
|   --- celery.py
|   --- __init__.py
|   --- task_receiver.py
|   --- task_submitter.py
--- docker-compose.yml
--- dockerfile
--- README.md
--- requirements.txt
1 directory, 8 files
```

Run the below command to start the docker cluster:

```
sudo docker-compose up
```

This will run one container for each worker and RabbitMQ. Once you see something like

```
worker_1  | [2018-03-01 10:46:30,013: INFO/MainProcess] celery@5af881b83b97 ready.
```

on your screen at the end of output, you are good to go.

Now you can submit the tasks. But before going any further lets try to understand the code while it is simple and small.

Understanding the code:

celery.py:

```
from celery import Celery

app = Celery(
    'celery_main',
    broker='amqp://myuser:mypassword@rabbit:5672',
    backend='rpc://',
    include=['celery_main.task_receiver']
)
```

The first argument to Celery is the name of the current module. This is only needed so that names can be automatically generated when the tasks are defined in the `__main__` module.

The second argument is the broker keyword argument, specifying the URL of the message broker you want to use. Here using RabbitMQ (also the default option).

The third argument is backend. A backend in Celery is used for storing the task results.

task_submitter.py:

```
from .task_receiver import do_work


if __name__ == '__main__':
    for i in range(10):
        result = do_work.delay(i)
        print('task submitted' + str(i))
```

This code will submit the tasks to workers. We need to call `do_work` method with delay so that it can be executed in async manner.

Flow returns immediately without waiting for result. If you try to print the result without waiting, it will print `None` .

task_receiver.py:

```
from celery_main.celery import app
import time
import random


@app.task(bind=True,default_retry_delay=10)
def do_work(self, item):
    print('Task received ' + str(item))
    # sleep for random seconds to simulate a really long task
    time.sleep(random.randint(1, 3))

    result = item + item
    return result
```

We can easily create a task from any callable by using the `task()` decorator. This is what we are doing here.

`bind=True` means the first argument to the task will always be the task instance (self). Bound tasks are needed for retries, for accessing information about the current task request.

Experimenting with Docker cluster:

Run the containers by using command `sudo docker-compose up`.
We will not be running containers in detached mode (-d) as we need to see the output. By default it will create one worker.

In another terminal go inside the worker container using command `sudo docker exec -it [container-name] bash`. It will start the bash session in working directory defined by `WORKDIR` in dockerfile.

Run the task submitter by using command `python -m celery_main.task_submitter`. Task submitter will submit the tasks to workers and exit without waiting for results.

You can see the output (info, debug and warnings) in previous terminal. Find out how much seconds cluster took to complete 10 tasks. Now stop all containers, remove them and restart them. But this time keep the worker count to 10. Use command `sudo docker-compose up --scale worker=10`.

Repeat the process and find the time taken to complete the tasks. Repeat above step by changing the worker count and concurrency value in dockerfile to find the best value for your machine where it took least time.

Increasing concurrency value beyond a limit will no longer improve the performance as workers will keep switching the context instead of doing actual job. Similarly increasing the worker count beyond a limit will make your machine go unresponsive. Keep a tab on CPU and memory consumed by running top command in another terminal.

Let's start downloading tweets:

Now lets start extending the boilerplate code. All the code we are going to write below is available on Github (https://github.com/anuragrana/scraping_tweets_celery_rabbitmq_docker_cluster.git). You can download and run the code to scrape the tweets.
All the twitter handles are in `handles.txt` file placed in root directory of code.

Update the `task_submitter.py` file to read the handles and submit them to to the task receiver. Task Receiver will get the response from twitter and parse the response to extract the tweets available on first page. For simplicity we are not going to the second page.

Code to extract the tweets is as below:

```
@app.task(bind=True,default_retry_delay=10)
def do_work(self, handle):
    print('handle received ' + handle)
    url = "https://twitter.com/" + handle
    session = requests.Session()
    response = session.get(url, timeout=5)
    print("-- STATUS " + str(response.status_code) + " -- " + url)
    if response.status_code == 200:
        parse_tweets(response, handle)


def parse_tweets(response, handle):
    soup = BeautifulSoup(response.text, 'lxml')
    tweets_list = list()
    tweets = soup.find_all("li", {"data-item-type": "tweet"})
    for tweet in tweets:
        tweets_list.append(get_tweet_text(tweet))

    print(str(len(tweets_list)) + " tweets found.")
    # save to DB or flat files.


def get_tweet_text(tweet):
    try:
        tweet_text_box = tweet.find("p", {"class": "TweetTextSize TweetTextSize--normal js-tweet-text tw
        images_in_tweet_tag = tweet_text_box.find_all("a", {"class": "twitter-timeline-link u-hidden"})
        tweet_text = tweet_text_box.text
        for image_in_tweet_tag in images_in_tweet_tag:
            tweet_text = tweet_text.replace(image_in_tweet_tag.text, '')
        return tweet_text
    except Exception as e:
        return None
```

Now if you run this code, it will start throwing too many requests i.e. HTTP status 429 error after few hits. To avoid this we need to use tor network to send the requests from different IPs and we will also use different user agent in each request.

Adding Rotating Proxy:

- Clone this git repository (https://github.com/mattes/rotating-proxy).

```
git clone https://github.com/mattes/rotating-proxy
```

- Change anything in the code as per you requirement.
- Build the image and use the same name in docker-compose file.

```
    rproxy:
        hostname: rproxy
        image: anuragrana/rotating-proxy
        environment:
            - tors=25
        ports:
            - "5566:5566"
            - "4444:4444"
```

- You may skip above steps as docker image with tag used in docker-compose is already present in docker hub.

- Create a file `proxy.py` and write the below code in it.

```
import requests
import user_agents
import random


def get_session():
    session = requests.session()
    session.proxies = {'http': 'rproxy:5566',
                       'https': 'rproxy:5566'}
    session.headers = get_headers()
    return session


def get_headers():
    headers = {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
        "accept-language": "en-GB,en-US;q=0.9,en;q=0.8",
        "User-Agent": random.choice(user_agents.useragents)
    }

    return headers
```

- Create a new file `user_agents.py` . This will contain the list of user agents and we will use one of these, selected randomly, in each request.

If you will run the container now, IP will be changed after every few requests and user agent will be changed on each hit, resulting in almost zero 429 status responses.

Performance:

I was able to download approx 10000 tweets in 60 seconds using 15 workers and 5 concurrency in each worker. Increasing the worker count beyond 15 started making machine unresponsive. You may try with different numbers and find out the configuration which will help scrape the maximum tweets in minimum time.

Source code:

Source code is available on Github (https://github.com/anuragrana/scraping_tweets_celery_rabbitmq_docker_cluster.git).

References:

- https://docs.docker.com/
- https://medium.com/@tonywangcn/how-to-build-docker-cluster-with-celery-and-rabbitmq-in-10-minutes-13fc74d21730
- http://docs.celeryproject.org/en/latest/getting-started/
- https://github.com/mattes/rotating-proxy

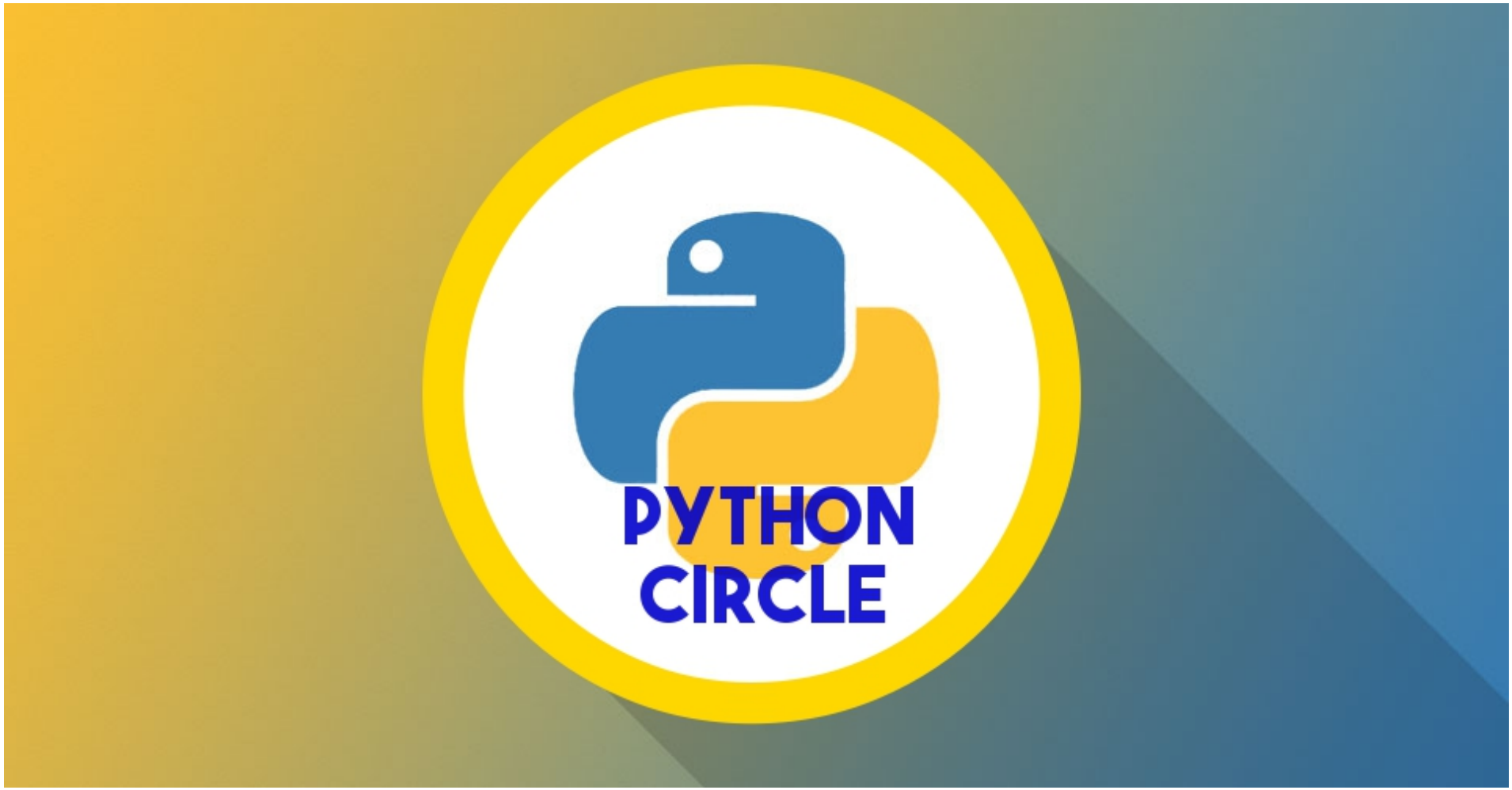
beautifulsoup (/tag/beautifulsoup/) scraping (/tag/scraping/) proxy (/tag/proxy/) tor (/tag/tor/) docker (/tag/docker/) rabbitmq (/tag/rabbitmq/) 0 7301

Related Articles:

Python Script 14: Scraping news headlines using python beautifulsoup (/post/678/python-script-14-scraping-news-headlines-using-python-beautifulsoup/)

Scraping news headlines using python beautifulsoup, web scraping using python, python script to scrape news, web scraping using beautifulsoup, news headlines scraping using python, python programm to get news headlines from web...

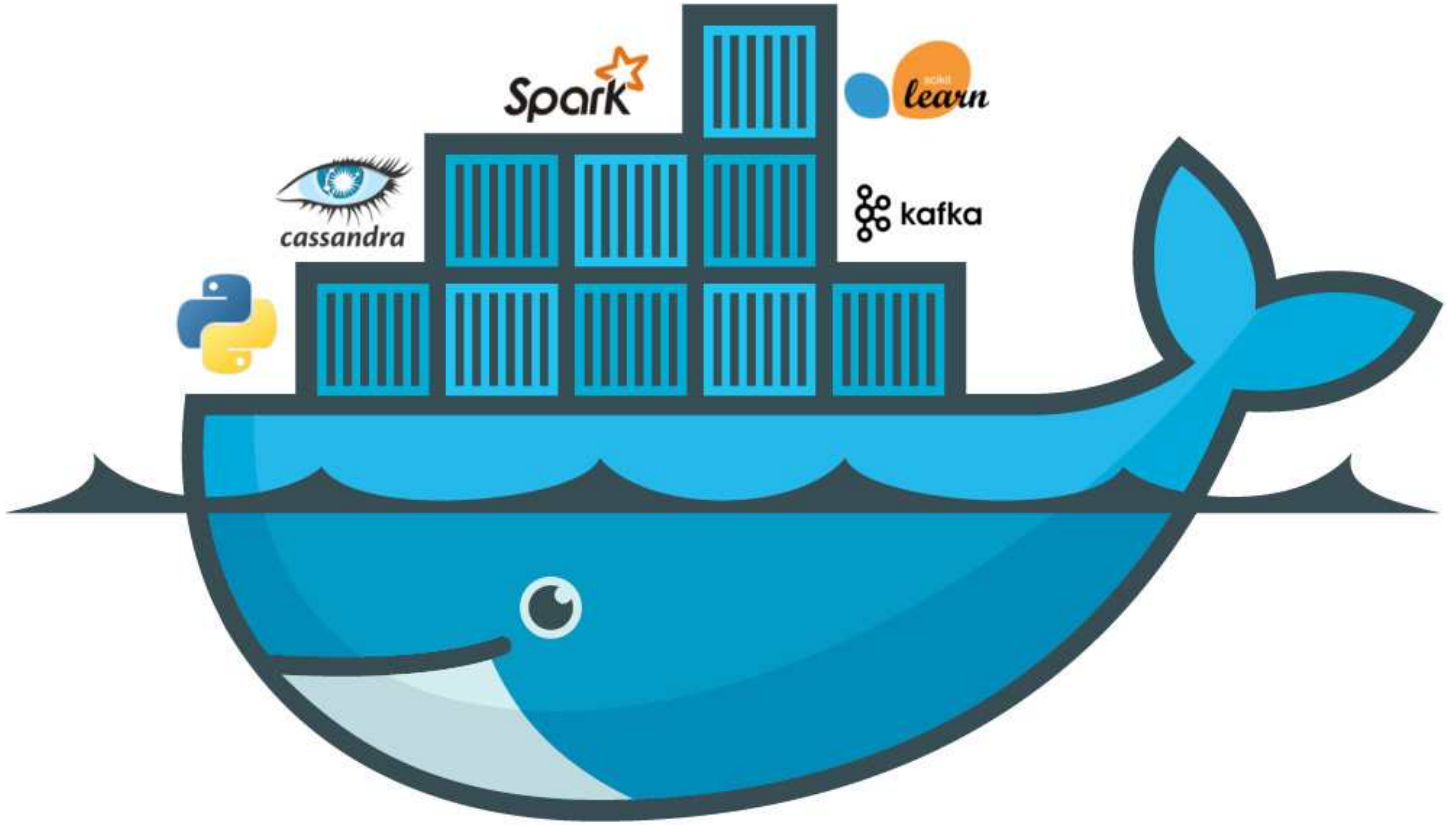
Read Full Article (/post/678/python-script-14-scraping-news-headlines-using-python-beautifulsoup/)



Python Script 2 : Crawling all emails from a website (/post/217/python-script-2-crawling-all-emails-from-a-website/)

Website crawling for email address, web scraping for emails, data scraping and fetching email adress, python code to scrape all emails froma websites, automating the email id scraping using python script, collect emails using python script...

Read Full Article (/post/217/python-script-2-crawling-all-emails-from-a-website/)



Using Docker instead of Virtual Environment for Django app development (/post/488/using-docker-instead-of-virtual-environment-for-django-app-development/)

Using docker instead of virtual environment for Django app development, starting with Dockerization of your Django application. Django with Docker....

Read Full Article (/post/488/using-docker-instead-of-virtual-environment-for-django-app-development/)

How to create completely automated telegram channel with python (/post/265/how-to-create-completely-automated-telegram-channel-with-python/)

Creating a completely automated telegram channel to generate and post content using python code on regular basis. Automating the Telegram channel using python script...

Read Full Article (/post/265/how-to-create-completely-automated-telegram-channel-with-python/)

---

0 thoughts on 'Scraping 10000 Tweets In 60 Seconds Using Celery, Rabbitmq And Docker Cluster With Rotating Proxy'

Leave a comment:

Type your comment here. For code use pastebin link. Limit 500 chars.

Your email please.

Your Name

Submit

*All Fields are mandatory. **Email Id will not be published publicly.

SUBSCRIBE

Please subscribe to get the latest articles in your mailbox.

Your Email ID Please

Subscribe

Python Circle
1,850 likes

Recent Posts:

- 5 lesser used Django template tags (/post/694/5-lesser-used-django-template-tags/)
- Solving Python Error- KeyError: 'key_name' (/post/693/solving-python-error-keyerror-key_name/)
- Intellectual property Law and Coding (/post/692/intellectual-property-law-and-coding/)
- Python Script 17: Setting bing image of the day as desktop wallpaper (/post/691/python-script-17-setting-bing-image-of-the-day-as-desktop-wallpaper/)
- Django Template Fiddle Launched !!!! (/post/690/django-template-fiddle-launched/)
- Python Script 16: Generating word cloud image of a text using python (/post/689/python-script-16-generating-word-cloud-image-of-a-text-using-python/)
- Preventing cross-site scripting attack on your Django website (/post/688/preventing-cross-site-scripting-attack-on-your-django-website/)
- How to generate ATOM/RSS feed for Django website (/post/687/how-to-generate-atomrss-feed-for-django-website/)
- How to create sitemap of Django website (/post/686/how-to-create-sitemap-of-django-website/)
- For loop in Django template (/post/685/for-loop-in-django-template/)
- How to add Favicon to Django websites (/post/684/how-to-add-favicon-to-django-websites/)
- Scraping data of 2019 Indian General Election using Python Request and BeautifulSoup and analyzing it (/post/683/scraping-data-of-2019-indian-general-election-using-python-request-and-beautifulsoup-and-analyzing-it/)
- How to display PDF in browser in Django instead of downloading it (/post/682/how-to-display-pdf-in-browser-in-django-instead-of-downloading-it/)
- Solving Python Error - UnboundLocalError: local variable 'x' referenced before assignment (/post/680/solving-python-error-unboundlocalerror-local-variable-x-referenced-before-assignment/)
- Python Script 15: Creating a port scanner in 8 lines of python (/post/679/python-script-15-creating-a-port-scanner-in-8-lines-of-python/)
- Python Script 14: Scraping news headlines using python beautifulsoup (/post/678/python-script-14-scraping-news-headlines-using-python-beautifulsoup/)
- How to download large csv files in Django (/post/677/how-to-download-large-csv-files-in-django/)
- Text based snake and ladder game in python (/post/676/text-based-snake-and-ladder-game-in-python/)

- Logging databases changes in Django Application (/post/675/logging-databases-changes-in-django-application/)
- Python Script 13: Generating ascii code from Image (/post/674/python-script-13-generating-ascii-code-from-image/)
- Python Frequently Asked Question 1: What can I do in Python? (/post/672/python-frequently-asked-question-1-what-can-i-do-in-python/)
- How to start with Python Programming - A beginner's guide (/post/671/how-to-start-with-python-programming-a-beginners-guide/)
- Python program to find whether a given year is leap year or not (/post/670/python-program-to-find-whether-a-given-year-is-leap-year-or-not/)
- Python Django Project Idea for beginners (/post/669/python-django-project-idea-for-beginners/)
- Uploading a file to FTP server using Python (/post/668/uploading-a-file-to-ftp-server-using-python/)
- Print statement in Python vs other programming languages (/post/667/print-statement-in-python-vs-other-programming-languages/)
- Automating Facebook page posts using python script (/post/666/automating-facebook-page-posts-using-python-script/)
- try .. except .. else .. in python with example (/post/664/try-except-else-in-python-with-example/)
- Python Script 12: Drawing Indian National Flag Tricolor using Python Turtle (/post/662/python-script-12-drawing-indian-national-flag-tricolor-using-python-turtle/)
- Python Script 11: Drawing Flag of United States of America using Python Turtle (/post/661/python-script-11-drawing-flag-of-united-states-of-america-using-python-turtle/)
- Solving Django Error: TemplateDoesNotExist at /app_name/ (/post/660/solving-django-error-templatedoesnotexist-at-app_name/)
- Adding Email Subscription Feature in Django Application (/post/657/adding-email-subscription-feature-in-django-application/)
- Using PostgreSQL Database with Python (/post/656/using-postgresql-database-with-python/)
- Programming On Raspberry Pi With Python: Activate LED and Buzzer on Motion Detection (/post/654/programming-on-raspberry-pi-with-python-activate-led-and-buzzer-on-motion-detection/)
- Programming on Raspberry Pi with Python: Controlling LED (/post/652/programming-on-raspberry-pi-with-python-controlling-led/)
- Programming on Raspberry Pi with Python: Sending IP address on Telegram channel on Raspberry Pi reboot (/post/651/programming-on-raspberry-pi-with-python-sending-ip-address-on-telegram-channel-on-raspberry-pi-reboot/)

- Programming on Raspberry Pi with Python: WIFI and SSH configuration (/post/650/programming-on-raspberry-pi-with-python-wifi-and-ssh-configuration/)
- Programming on Raspberry Pi with Python: Raspberry Pi Setup (/post/649/programming-on-raspberry-pi-with-python-raspberry-pi-setup/)
- Iterator and Generators in Python: Explained with example (/post/648/iterator-and-generators-in-python-explained-with-example/)
- Top 5 Python Books (/post/646/top-5-python-books/)
- Encryption-Decryption in Python Django (/post/641/encryption-decryption-in-python-django/)
- Sending Emails Using Python and Gmail (/post/628/sending-emails-using-python-and-gmail/)
- How to Track Email Opens Sent From Django App (/post/626/how-to-track-email-opens-sent-from-django-app/)
- Python Tip 1: Accessing localhost Django webserver over the Internet (/post/618/python-tip-1-accessing-localhost-django-webserver-over-the-internet/)
- 5 common mistakes made by beginner python programmers (/post/602/5-common-mistakes-made-by-beginner-python-programmers/)
- How to upload and process the Excel file in Django (/post/591/how-to-upload-and-process-the-excel-file-in-django/)
- Creating sitemap of Dynamic URLs in your Django Application (/post/584/creating-sitemap-of-dynamic-urls-in-your-django-application/)
- Adding Robots.txt file to Django Application (/post/578/adding-robotstxt-file-to-django-application/)
- Python Script 3: Validate, format and Beautify JSON string Using Python (/post/576/python-script-3-validate-format-and-beautify-json-string-using-python/)
- Displaying custom 404 error (page not found) page in Django 2.0 (/post/564/displaying-custom-404-error-page-not-found-page-in-django-20/)

Tags:

404 (/tag/404/) 500 (/tag/500/) AJAX (/tag/ajax/) API (/tag/api/) AUTHENTICATION (/tag/authentication/)

AUTOMATION (/tag/automation/) AWS (/tag/aws/) BACKUP (/tag/backup/) BEAUTIFULSOUP (/tag/beautifulsoup/) BOOK (/tag/book/)

BREADBOARD (/tag/breadboard/) CAPTCHA (/tag/captcha/) CELERY (/tag/celery/) COMMANDS (/tag/commands/) CSV (/tag/csv/)

DATA (/tag/data/) DATABASE (/tag/database/) DECRYPTION (/tag/decryption/) DJANGO (/tag/django/)

DJANGO APP (/tag/django%20app/) DJANGO ERROR (/tag/django%20error/) DOCKER (/tag/docker/) DOWNLOAD (/tag/download/)

EC2 (/tag/ec2/) ELASTIC SEARCH (/tag/elastic%20search/) EMAIL (/tag/email/) ENCRYPTION (/tag/encryption/) ERROR (/tag/error/)

ERROR PAGE (/tag/error%20page/) EXCEL (/tag/excel/) EXCEPTION (/tag/exception/) FACEBOOK (/tag/facebook/) FAQ (/tag/faq/)

FEEDS (/tag/feeds/) FOR LOOP (/tag/for%20loop/) FORM (/tag/form/) FREE EMAILS (/tag/free%20emails/) FTP (/tag/ftp/)

GAME (/tag/game/) GENERATOR (/tag/generator/) GMAIL (/tag/gmail/) GPIO (/tag/gpio/) GRAPH API (/tag/graph%20api/)

HARDWARE (/tag/hardware/) HOSTING (/tag/hosting/) IMAGE (/tag/image/) INSTAGRAM (/tag/instagram/)

IP ADDRESS (/tag/ip%20address/) IP LAW (/tag/ip%20law/) ITERATOR (/tag/iterator/) JSON (/tag/json/) KIBANA (/tag/kibana/)

LED (/tag/led/) LOGGING (/tag/logging/) MISTAKE (/tag/mistake/) MONGODB (/tag/mongodb/) PACKAGE (/tag/package/)

PAYMENT GATWAY (/tag/payment%20gatway/) PDF (/tag/pdf/) PITFALLS (/tag/pitfalls/) PROGRAM (/tag/program/)

PROJECT (/tag/project/) PROXY (/tag/proxy/) PYTHON BITES (/tag/python%20bites/) PYTHON TIPS (/tag/python%20tips/)

PYTHONANYWHERE (/tag/pythonanywhere/) RABBITMQ (/tag/rabbitmq/) RASPBERRYPI (/tag/raspberrypi/) REQUESTS (/tag/requests/)

RESPONSE (/tag/response/) SCRAPING (/tag/scraping/) SCRAPY (/tag/scrapy/) SCRIPT (/tag/script/) SECURITY (/tag/security/)

SENSOR (/tag/sensor/) SEO (/tag/seo/) SERVER (/tag/server/) SETUP (/tag/setup/) SIGNALS (/tag/signals/)

SITEMAP (/tag/sitemap/) SOCKET (/tag/socket/) SSH (/tag/ssh/) TAG (/tag/tag/) TELEGRAM (/tag/telegram/)

TEMPLATES (/tag/templates/) TIPS (/tag/tips/) TKINTER (/tag/tkinter/) TOR (/tag/tor/) TRACKING (/tag/tracking/)

TRY CATCH (/tag/try%20catch/) TURTLE (/tag/turtle/) UPLOAD (/tag/upload/) VIRTUAL ENV (/tag/virtual%20env/) WIFI (/tag/wifi/)

WORD CLOUD (/tag/word%20cloud/)