proxy_py Documentation

Release 2.0

DevAlone

Oct 04, 2018

Contents

Contents:

CHAPTER 1

proxy_py README

proxy_py is a program which collects proxies, saves them in a database and makes periodically checks. It has a server for getting proxies with nice API(see below).

1.1 Where is the documentation?

It's here -> http://proxy-py.readthedocs.io

1.2 How to build?

1 Clone this repository

```
git clone https://github.com/DevAlone/proxy_py.git
```

2 Install requirements

```
cd proxy_py
pip3 install -r requirements.txt
```

3 Create settings file

```
cp config_examples/settings.py proxy_py/settings.py
```

4 Install postgresql and change database configuration in settings.py file

5 (Optional) Configure alembic

6 Run your application

7 Enjoy!

1.3 I'm too lazy. Can I just use it?

TODO: update, old version!

Yes, you can download virtualbox image here -> https://drive.google.com/file/d/1oPf6xwOADRH95oZW0vkPr1Uu_iLDe9jc/view?usp=sharing

After downloading check that port forwarding is still working, you need forwarding of 55555 host port to 55555 guest.

1.4 How to get proxies?

proxy_py has a server, based on aiohttp, which is listening 127.0.0.1:55555 (you can change it in the settings file) and provides proxies. To get proxies you should send the following json request on address *http://127.0.0.1:55555/api/v1/* (or other domain if behind reverse proxy):

```
{
    "model": "proxy",
    "method": "get",
    "order_by": "response_time, uptime"
}
```

Note: order_by makes the result sorted by one or more fields(separated by comma). You can skip it. The required fields are *model* and *method*.

It's gonna return you the json response like this:

```
{
    "count": 1,
    "data": [{
            "address": "http://127.0.0.1:8080",
            "auth_data": "",
            "bad_proxy": false,
            "domain": "127.0.0.1",
            "last_check_time": 1509466165,
            "number_of_bad_checks": 0,
            "port": 8080,
            "protocol": "http",
            "response_time": 461691,
            "uptime": 1509460949
        }
    ],
    "has_more": false,
    "status": "ok",
    "status_code": 200
}
```

Note: All fields except *protocol*, *domain*, *port*, *auth_data*, *checking_period* and *address* CAN be null

Or error if something went wrong:

```
{
    "error_message": "You should specify \"model\"",
    "status": "error",
    "status_code": 400
}
```

Note: status_code is also duplicated in HTTP status code

Example using curl:

```
curl -X POST http://127.0.0.1:55555/api/v1/ -H "Content-Type: application/json" --
↪data '{"model": "proxy", "method": "get"}'
```

Example using httpie:

```
http POST http://127.0.0.1:55555/api/v1/ model=proxy method=get
```

Example using python's *requests* library:

```
import requests
import json


def get_proxies():
    result = []
    json_data = {
        "model": "proxy",
        "method": "get",
    }
    url = "http://127.0.0.1:55555/api/v1/"

    response = requests.post(url, json=json_data)
    if response.status_code == 200:
        response = json.loads(response.text)
        for proxy in response["data"]:
            result.append(proxy["address"])
    else:
        # check error here
        pass

    return result
```

Example using aiohttp library:

```
import aiohttp


async def get_proxies():
    result = []
    json_data = {
        "model": "proxy",
        "method": "get",
    }

    url = "http://127.0.0.1:55555/api/v1/"

    async with aiohttp.ClientSession() as session:
        async with session.post(url, json=json_data) as response:
            if response.status == 200:
                response = json.loads(await response.text())
                for proxy in response["data"]:
                    result.append(proxy["address"])
            else:
                # check error here
```

(continues on next page)

(continued from previous page)

```
                pass

    return result
```

1.5 How to interact with API?

Read more about API here -> https://github.com/DevAlone/proxy_py/tree/master/docs/API.md

1.6 How to contribute?

TODO: write guide about it

1.7 How to test it?

If you've made changes to the code and want to check that you didn't break anything, just run

```
py.test
```

inside virtual environment in proxy_py project directory.

1.8 How to deploy on production using supervisor, nginx and postgresql in 8 steps?

1 Install supervisor, nginx and postgresql

```
root@server:~$ apt install supervisor nginx postgresql
```

2 Create virtual environment and install requirements on it

3 Copy settings.py example:

```
proxy_py@server:~/proxy_py$ cp config_examples/settings.py proxy_py/
```

4 create unprivileged user in postgresql database and change database authentication data in settings.py

```
proxy_py@server:~/proxy_py$ vim proxy_py/settings.py
```

```
DATABASE_CONNECTION_KWARGS = {
    'database': 'YOUR_POSTGRES_DATABASE',
    'user': 'YOUR_POSTGRES_USER',
    'password': 'YOUR_POSTGRES_PASSWORD',
    # number of simultaneous connections
    # 'max_connections': 20,
}
```

5 Copy supervisor config example and change it for your case

```
cp /home/proxy_py/proxy_py/config_examples/proxy_py.supervisor.conf /etc/supervisor/
↪conf.d/proxy_py.conf
vim /etc/supervisor/conf.d/proxy_py.conf
```

6 Copy nginx config example, enable it and change if you need

```
cp /home/proxy_py/proxy_py/config_examples/proxy_py.nginx.conf /etc/nginx/sites-
↪available/proxy_py
ln -s /etc/nginx/sites-available/proxy_py /etc/nginx/sites-enabled/
vim /etc/nginx/sites-available/proxy_py
```

7 Restart supervisor and Nginx

```
supervisorctl reread
supervisorctl update
/etc/init.d/nginx configtest
/etc/init.d/nginx restart
```

8 Enjoy using it on your server!

1.9 What is it depend on?

See *requirements.txt*

CHAPTER 2

proxy_py API

proxy_py expects HTTP POST requests with JSON as a body, so you need to add header `Content-Type: application/json` and send correct JSON document.

Example of correct request:

```
{
  "model": "proxy",
  "method": "get"
}
```

Response is also HTTP with JSON and status code depending on whether error happened or not.

- 200 if there wasn't error
- 400 if you sent bad request
- 500 if there was an error during execution your request or in some other cases

status_code is also duplicated in JSON body.

2.1 Possible keys

- `model` - specifies what you will work with. Now it's only supported to work with `proxy` model.
- `method` - what you're gonna do with it
  - `get` - get model items as json objects. Detailed description is below
  - `count` - count how many items there are. Detailed description is below

2.1.1 get method

`get` method supports following keys:

- `order_by` (string) - specifies ordering fields as comma separated value.

Examples:

"uptime" just sorts proxies by uptime field ascending.

Note: uptime is the timestamp from which proxy is working, NOT proxy's working time

To sort descending use − before field name.

"-response_time" returns proxies with maximum response_time first (in microseconds)

It's possible to sort using multiple fields

"number_of_bad_checks, response_time" returns proxies with minimum number_of_bad_checks first, if there are proxies with the same number_of_bad_checks, sorts them by response_time

- limit (integer) - specifying how many proxies to return
- offset (integer) - specifying how many proxies to skip

Example of get request:

```
{
    "model": "proxy",
    "method": "get",
    "order_by": "number_of_bad_checks, response_time",
    "limit": 100,
    "offset": 200
}
```

Response

```
{
    "count": 6569,
    "data": [
        {
            "address": "socks5://localhost:9999",
            "auth_data": "",
            "bad_proxy": false,
            "domain": "localhost",
            "last_check_time": 1517089048,
            "number_of_bad_checks": 0,
            "port": 9999,
            "protocol": "socks5",
            "response_time": 1819186,
            "uptime": 1517072132
        },

        ...

    ],
    "has_more": true,
    "status": "ok",
    "status_code": 200
}
```

- count (integer) - total number of proxies for that request
- data (array) - list of proxies
- has_more (boolean) - value indicating whether you can increase offset to get more proxies or not
- status (string) - "error" if error happened, "ok" otherwise

Example of error:

Request:

```
{
    "model": "user",
    "method": "get",
    "order_by": "number_of_bad_checks, response_time",
    "limit": 100,
    "offset": 200
}
```

Response:

```
{
    "error_message": "Model \"user\" doesn't exist or isn't allowed",
    "status": "error",
    "status_code": 400
}
```

2.1.2 count method

Same as get, but not returns data

CHAPTER 3

proxy_py Guides

3.1 proxy_py How to create a collector

Collector is a class which is used to parse proxies from web page or another source. All collectors are inherited from *collectors.abstract_collector.AbstractCollector*, also there is *collectors.pages_collector.PagesCollector* which is used for paginated sources. It's always better to learn through the examples.

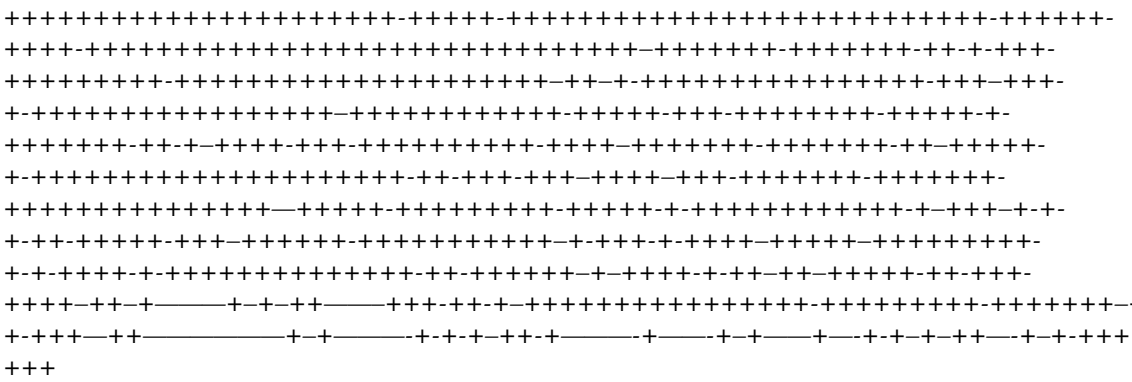
3.1.1 Simple collector

Let's start with the simplest collector we can imagine, it will be collecting from the page http://www.89ip.cn/ti.html as you can see, it sends form as GET request to this url http://www.89ip.cn/tqdl.html?num=9999&address=&kill_address=&port=&kill_port=&isp=

Firstly we can try to check that these proxies are really good. Just copy and paste list of proxies to file say /tmp/proxies and run this command inside virtual environment

```
cat /tmp/proxies | python3 check_from_stdin.py
```

You're gonna get something like this:



"+" means working proxy with at least one protocol, "-" means not working, the result above is perfect, so many good proxies.

Note: working means proxy respond with timeout set in settings, if you increase it, you're likely to get more proxies.

Alright, let's code!

We need to place our collector inside *collectors/web/* directory using reversed domain path, it will be *collectors/web/cn/89ip/collector.py*

To make class be a collector we need to declare a variable *__collector__* and set it to *True*

Note: name of file and name of class don't make sense, you can declare as many files and classes in each file per domain as you want

```
from collectors import AbstractCollector


class Collector(AbstractCollector):
    __collector__ = True
```

We can override default processing period in constructor like this:

```
def __init__(self):
    super(Collector, self).__init__()
    # 30 minutes
    self.processing_period = 30 * 60
    '''
    floating period means proxy_py will be changing
    period to not make extra requests and handle
    new proxies in time, you don't need to disable
    it in most cases
    '''
    # self.floating_processing_period = False
```

The last step is to implement *collect()* method. Import useful things

```
from parsers import RegexParser

import http_client
```

and implement method like this:

```
async def collect(self):
    url = 'http://www.89ip.cn/tqdl.html?num=9999&address=&kill_address=&port=&kill_
↪port=&isp='
    # send a request to get html code of the page
    html = await http_client.get_text(url)
    # and just parse it using regex parser with a default rule to parse
    # proxies like this:
    # 8.8.8.8:8080
    return RegexParser().parse(html)
```

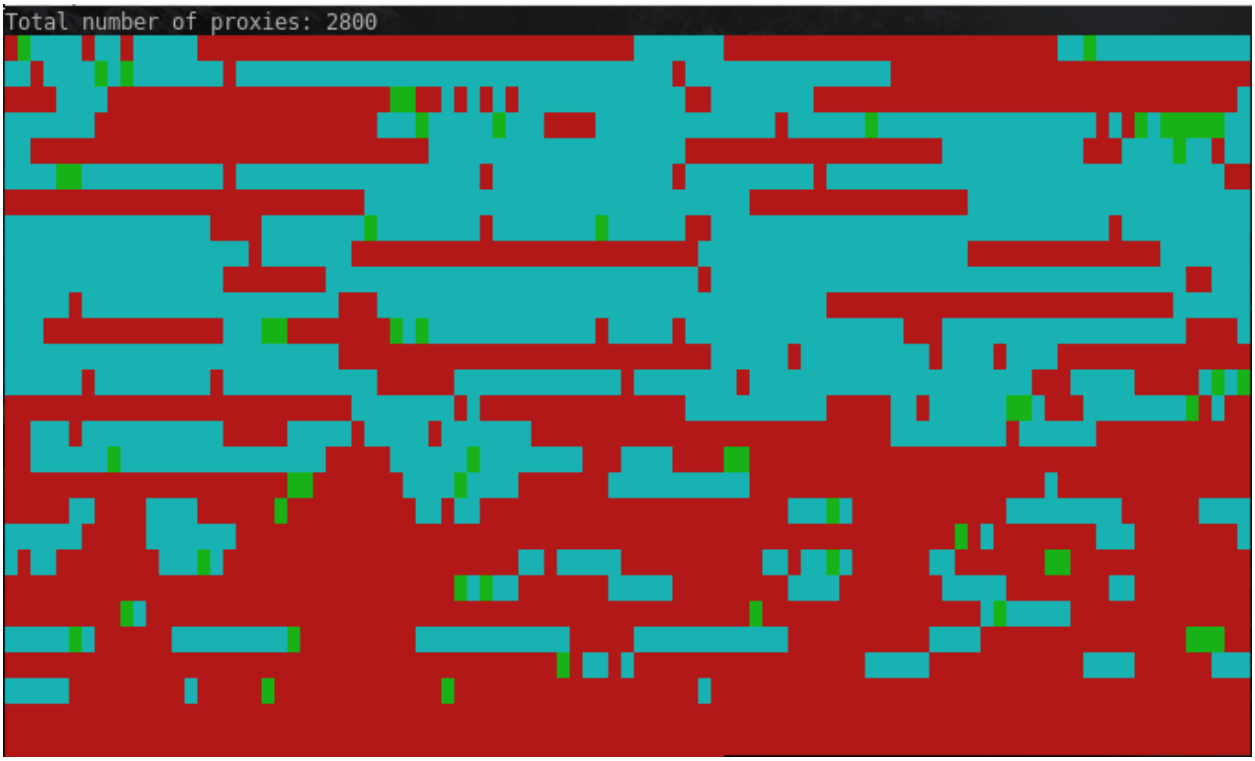
That's all!

Now is time for a little test, to be sure your collector is working you can run proxy_py with *–test-collector* option:

```
python3 main.py --test-collector collectors/web/cn/89ip/collector.py:Collector
```

which means to take class Collector from the file *collectors/web/cn/89ip/collector.py*

It's gonna draw you a pattern like this:



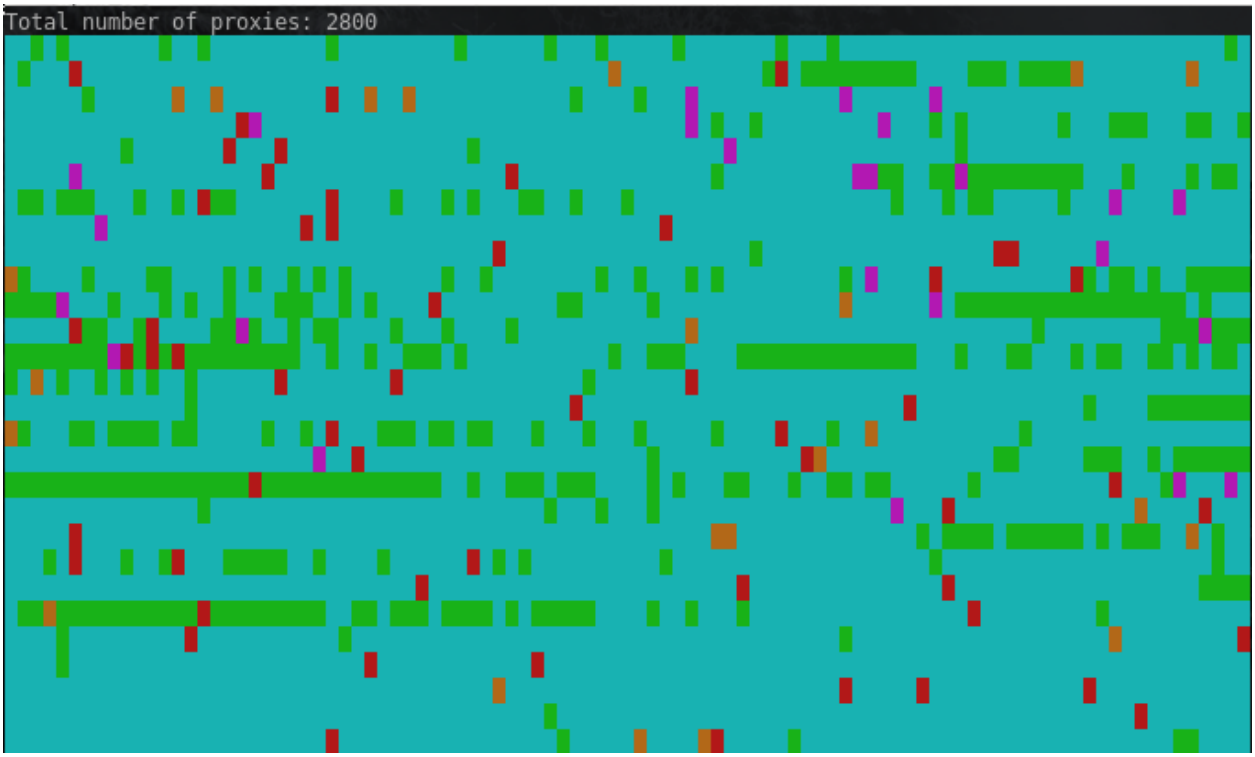
Where red cell means not working proxy

- cyan - respond within a second
- green - slower than 5 seconds
- yellow - up to 10 seconds
- magenta - slower than 10 seconds

Note: don't forget that settings.py limits amount of time for proxy to respond. You can override proxy checking timeout by using *–proxy-checking-timeout* option. For example

```
python3 main.py --test-collector collectors/web/cn/89ip/collector.py:Collector --
↪proxy-checking-timeout 60
```

With 60 seconds timeout it looks better



3.1.2 Paginated collector

Alright, you've done with a simple collector, you're almost a pro, let's now dive a little deeper

TODO: complete this guide

CHAPTER 4

proxy_py Modules

4.1 async_requests module

class async_requests.**Response**(*status*, *text*, *aiohttp_response=None*)

Bases: object

static from_aiohttp_response(*aiohttp_response*)

async_requests.**get**(*url*, ***kwargs*)

async_requests.**get_random_user_agent**()

async_requests.**post**(*url*, *data*, ***kwargs*)

async_requests.**request**(*method*, *url*, ***kwargs*)

4.2 checkers package

4.2.1 Submodules

checkers.base_checker module

checkers.d3d_info_checker module

checkers.ipinfo_io_checker module

4.2.2 Module contents

4.3 collectors package

4.3.1 Subpackages

collectors.checkerproxy_net package

Submodules

collectors.checkerproxy_net.collector_checkerproxy_net module

collectors.checkerproxy_net.collector_checkerproxy_net_today module

Module contents

collectors.free_proxy_list_net package

Submodules

collectors.free_proxy_list_net.base_collector_free_proxy_list_net module

collectors.free_proxy_list_net.collector_free_proxy_list_net module

collectors.free_proxy_list_net.collector_free_proxy_list_net_anonymous_proxy module

collectors.free_proxy_list_net.collector_free_proxy_list_net_uk_proxy module

collectors.free_proxy_list_net.collector_socks_proxy_net module

collectors.free_proxy_list_net.collector_sslproxies_org module

collectors.free_proxy_list_net.collector_us_proxy_org module

Module contents

collectors.freeproxylists_net package

Submodules

collectors.freeproxylists_net.freeproxylists_net module

Module contents

static from_request(*request: server.api_v1.requests_to_models.request.Request*)

class server.api_v1.requests_to_models.request.**FetchRequest**(*class_name*, *fields: list = None*, *order_by: list = None*)
Bases: *server.api_v1.requests_to_models.request.Request*

class server.api_v1.requests_to_models.request.**GetRequest**(*class_name*, *fields: list = None*, *order_by: list = None*)
Bases: *server.api_v1.requests_to_models.request.FetchRequest*

static from_request(*request: server.api_v1.requests_to_models.request.Request*)

class server.api_v1.requests_to_models.request.**Request**(*class_name*)
Bases: object

server.api_v1.requests_to_models.request_executor module

server.api_v1.requests_to_models.request_parser module

exception server.api_v1.requests_to_models.request_parser.**ConfigFormatError**
Bases: Exception

exception server.api_v1.requests_to_models.request_parser.**ParseError**
Bases: Exception

class server.api_v1.requests_to_models.request_parser.**RequestParser**(*config*)
Bases: object

ALLOWED_CHARS = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789/: !=>

ALLOWED_KEYS = {'filter', 'model', 'offset', 'order_by', 'limit', 'fields', 'method'}

COMMA_SEPARATED_KEYS = {'fields', 'order_by'}

MAXIMUM_KEY_LENGTH = 64

MAXIMUM_LIMIT_VALUE = 1024

MAXIMUM_VALUE_LENGTH = 512

MINIMUM_LIMIT_VALUE = 1

comma_separated_field_to_list(*string_field*)

method_count(*req_dict*, *config*, *result_request*)

method_get(*req_dict*, *config*, *result_request*)

parse(*request: dict*)

parse_dict(*req_dict*)

parse_fields(*req_dict*, *config*)

parse_list(*req_dict*, *config*, *request_key*, *config_key*, *default_value*)

parse_order_by_fields(*req_dict*, *config*)

validate_key(*key: str*)

validate_value(*key: str*, *value*)

exception server.api_v1.requests_to_models.request_parser.**ValidationError**
Bases: *server.api_v1.requests_to_models.request_parser.ParseError*

Module contents

Submodules

server.api_v1.api_request_handler module

server.api_v1.app module

Module contents

server.frontend package

Submodules

server.frontend.app module

Module contents

4.12.2 Submodules

server.base_app module

server.proxy_provider_server module

4.12.3 Module contents

4.13 setup module

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

S

V