Scrapy Proxy Guide: How to Integrate & Rotate Proxies With Scrapy

If you are scraping at scale then using proxies is a must to avoid your spiders getting blocked or returning unreliable data.

There are many different proxy types to choose from with different integration methods, so in this guide, we're going to go through step by step:

- What Are Proxies & Why Do We Need Them?
- The 3 Most Popular Proxy Integration Methods
- How To Integrate & Rotate Proxy Lists
- How To Use Rotating/Backconnect Proxies
- How To Use Proxy APIs

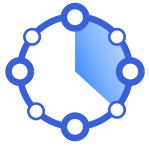
First, let's quickly go over some the very basics.

Need help scraping the web?

Then check out **ScrapeOps**, the complete toolkit for web scraping.

Proxy Manager

Scraper Monitoring



Job Scheduling

What Are Proxies & Why Do We Need Them?

Web scraping proxies are IP addresses that you route your requests through instead of using your own or servers IP address.

We need them when web scraping as they allow us to spread our requests over thousands of proxies so that you can easily scrape a website at scale, without the target website blocking us.

If you doing a very small scraping project or scraping a website without a sophisticated anti-bot countermeasures then you mightn't need them. However, when you start scraping big websites or at larger volumes then proxies quickly become a must as they allow you to:

- Bypass anti-bot countermeasures
- Get country specific data from websites
- Hide your identity from the websites you are scraping

There are many different types of proxies (datacenter proxies, residential proxies, mobile proxies, ISP proxies, SOAX proxies), however, for the purposes of this guide we will focus on how to integrate them into our Scrapy spiders.

The 3 Most Popular Proxy Integration Methods

When it comes to proxies there are 3 main integration methods that are most commonly used:

1. Proxy Lists
2. Rotating/Backconnect Proxies
3. Proxy APIs

All 3 have their pros and cons, and can have an impact on whether you have dedicated proxies or proxies in a shared pool. However, which type you use is really down to your own personal preferences and project requirements (budget, performance, ease of use, etc).

The easiest proxies to use are **smart proxies** that either allow you to send your requests to a single proxy endpoint or to a HTTP API.

These smart proxy providers take care of all the **proxy selection**, **rotation**, **ban detection**, etc. within their proxy, and allow you to easily enable extra functionality like JS rendering, country-level geotargeting, residential proxies, etc. by simply adding some flags to your request.

Examples of smart proxy providers are: ScraperAPI, Scrapingbee, Zyte SmartProxy

---

How To Integrate & Rotate Proxy Lists

The most fundamental way of using proxies, is to insert a list of proxy IPs into your

spider and configure it to select a random proxy every time it makes a request.

```
'proxy1.com:8000',
'proxy2.com:8031',
'proxy3.com:8032',
```

When you sign up to some proxy providers, they will give you a list of proxy IP addresses that you will then need to use in your spider. Most free proxy lists online use this approach and some large providers still offer this method for datacenter IPs or if you want dedicated proxies.

To integrate the a list of proxies with your spider, we can build our own proxy management layer or we can simply install an existing Scrapy middleware that will manage our proxy list for us.

There are a number of free Scrapy middlewares out there that you can choose from (like scrapy-proxies), however, for this guide we're going to use the scrapy-rotating-proxies middleware as it was developed by the some of Scrapy's lead maintainers and has some really cool functionality.

scrapy-rotating-proxies is very easy to setup and is very customisable. To get started simply install the middleware:

```
pip install scrapy-rotating-proxies
```

Then we just need to update our `settings.py` to load in our proxies and enable the **scrapy-rotating-proxies** middleware:

```
## settings.py

## Insert Your List of Proxies Here
ROTATING_PROXY_LIST = [
    'proxy1.com:8000',
    'proxy2.com:8031',
    'proxy3.com:8032',
]

## Enable The Proxy Middleware In Your Downloader Middlewares
DOWNLOADER_MIDDLEWARES = {
    # ...
    'rotating_proxies.middlewares.RotatingProxyMiddleware': 610,
    'rotating_proxies.middlewares.BanDetectionMiddleware': 620,
    # ...
}
```

And that's it. After this all requests your spider will make will be proxied using one of the proxies from the `ROTATING_PROXY_LIST`.

Alternatively, you could give the **scrapy-rotating-proxies** middleware a path to a file that contains the proxy list and your spider will use the proxies from this list when making requests.

```
## settings.py
```

```
ROTATING_PROXY_LIST_PATH = '/my/path/proxies.txt'
```

The very cool thing about the **scrapy-rotating-proxies** middleware is that it will actively monitor the health of each individual proxy and remove any dead proxies from the proxy rotation.

You can also define your own ban detection policies, so you can tell the **scrapy-rotating-proxies** middleware what constitutes a dead proxy so it can remove it from the rotation. For more on this functionality then check out the docs.

How To Use Rotating/Backconnect Proxies

Once upon a time, all proxy providers gave you lists of proxy IPs when you purchased a plan with them.

However, it is far more common for them to provide you a single proxy endpoint that you send your requests too and they handle the selection and rotation of the proxies on their end. Making it much easier for you to integrate a proxy solution into your spider.

Examples of such proxies include BrightData, Oxylabs, NetNut. Their proxy endpoints look something like this:

```
'zproxy.lum-superproxy.io:22225', # BrightData
```

```
'pr.oxylabs.io:7777', # Oxylabs
'gw.ntnt.io:5959', # Netnut
```

Important: When using a single proxy endpoint, you shouldn't use a rotating proxy middleware like the **scrapy-rotating-proxies** middleware as it could interfere with the correct functioning of the proxy.

You have a couple of options on how you integrate one of these proxy endpoints into your spider.

---

1. Via Request Parameters

Simply include the proxy connection details in the meta field of every request within your spider.

```
## your_spider.py

def start_requests(self):
    for url in self.start_urls:
        return Request(url=url, callback=self.parse,
                       meta={"proxy": "http://proxy:8010"})
```

Scrapy's HttpProxyMiddleware, which is enabled by default, will then route the request through the proxy you defined.

2. Create Custom Middleware

A cleaner and more modular approach is to create a custom middleware which you then enable in your `settings.py` file. This will ensure all spiders will use the proxy.

Here is an example custom middleware that you can add to your `middlewares.py` file:

```
## middlewares.py

import base64

class MyProxyMiddleware(object):

    @classmethod
    def from_crawler(cls, crawler):
        return cls(crawler.settings)

    def __init__(self, settings):
        self.user = settings.get('PROXY_USER')
        self.password = settings.get('PROXY_PASSWORD')
        self.endpoint = settings.get('PROXY_ENDPOINT')
        self.port = settings.get('PROXY_PORT')

    def process_request(self, request, spider):
        user_credentials = '{user}:{passw}'.format(user=self.user,
```

```
passw=self.password)
        basic_authentication = 'Basic ' +
base64.b64encode(user_credentials.encode()).decode()
        host = 'http://{endpoint}:
{port}'.format(endpoint=self.endpoint, port=self.port)
        request.meta['proxy'] = host
        request.headers['Proxy-Authorization'] = basic_authentication
```

user_credentials.encode() => returns a byte string (encoding utf-8) <class 'bytes'>
base64.b64encode(bytes) => encodes the byte string to base 64 <class 'bytes'>
bytes.decode() returns a regular string with the same characters as a byte string

Then you just need to enable it in your `settings.py` file, and fill in your proxy connection details:

```
## settings.py

PROXY_USER = 'username'
PROXY_PASSWORD = 'password'
PROXY_ENDPOINT = 'proxy.proxyprovider.com'
PROXY_PORT = '8000'

DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.MyProxyMiddleware': 350,
    'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware': 400,
}
```

Note: For this middleware to work correctly, you will need to put it before the default Scrapy HttpProxyMiddleware by assigin it a lower number.

How To Use Proxy APIs

Over the last few years, a number of smart proxy solution have been launched that take care of all the proxy/user-agent selection, rotation, ban detection, and are easily customisable.

Typically, these smart proxy solutions allow you to make requests via their HTTP endpoint. Some even have dedicated SDKs and traditional proxy endpoints.

Instead, of adding a proxy to your request, you send the url you want to scrape to them via their API and then they return the HTML response to you. Only charging you if the request has been successful.

For this example, we're going to use ScraperAPI.

---

1. Via API Endpoint

To send the pages we want to scrape to **ScraperAPI** we simply just need to forward the urls we want to scrape to their API endpoint. We can do this by creating a simple function:

```
## myspider.py


API_KEY = 'YOUR_API_KEY'
```

```
def get_proxy_url(url):
    payload = {'api_key': API_KEY, 'url': url}
    proxy_url = 'http://api.scraperapi.com/?' + urlencode(payload)
    return proxy_url
```

And use this function in our Scrapy request:

```
## myspider.py

yield scrapy.Request(url=get_proxy_url(url), callback=self.parse)
```

This is how your final code should look.

```
## myspider.py

import scrapy
from urllib.parse import urlencode

API_KEY = 'YOUR_API_KEY'

def get_proxy_url(url):
    payload = {'api_key': API_KEY, 'url': url}
    proxy_url = 'http://api.scraperapi.com/?' + urlencode(payload)
    return proxy_url

class QuotesSpider(scrapy.Spider):
    name = "QuotesSpider"

    def start_requests(self):
```

```
        urls = [
            'http://quotes.toscrape.com/',
        ]
        for url in urls:
            yield scrapy.Request(url=get_proxy_url(url),
callback=self.parse)

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get(),
                'author': quote.css('small.author::text').get(),
                'tags': quote.css('div.tags a.tag::text').getall(),
            }

        next_page_url = response.css("li.next >
a::attr(href)").extract_first()
        if next_page is not None:
           next_page_url = 'http://quotes.toscrape.com/' + next_page
           yield response.follow(get_proxy_url(next_page_url),
callback=self.parse)
```

---

2. Using SDK

Alternatively, a lot of proxy providers now have their own SDKs and custom middlewares that make it even easier to integrate them into your scrapers.

For ScraperAPI, simply install their SDK:

```
pip install scraperapi-sdk
```

Then integrate the SDK into your code by initialising the ScraperAPIClient with your API key and then using the `client.scrapyGet` method to make requests.

```
## myspider.py

import scrapy
from scraper_api import ScraperAPIClient
client = ScraperAPIClient('YOUR_API_KEY')

class QuotesSpider(scrapy.Spider):
    name = "quotes"

    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(client.scrapyGet(url=url),
callback=self.parse)
```

More Scrapy Tutorials

That's it for all the most common ways you can integrate proxies with Scrapy.

If you would like to take things a step further by using multiple types of proxies from different proxy providers, and create your own hierarchical waterfall system then be sure to check out our Proxy Waterfall System guide.

If you would like to learn more about Scrapy in general, then be sure to check out The Scrapy Playbook.