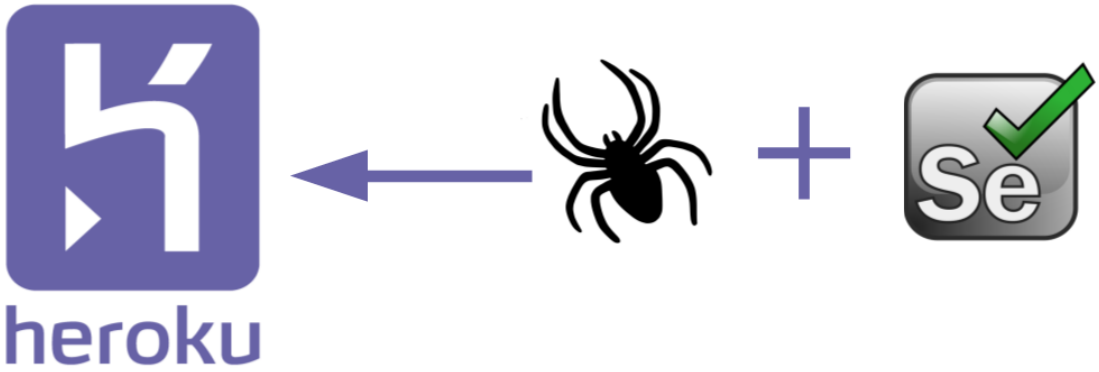
Romik Kelesh

87 Followers About Follow

How to deploy a Python — Web Scraper with Selenium on Heroku

Romik Kelesh Jul 30, 2020 · 6 min read



Anyone who starts with web scraping gets to the point where they don't want to run it on their computer anymore, but on a cloud and if possible for free.

Deploying web scrapers on Heroku is not that difficult.
However, if you want to do this in combination with Selenium, there are a few things to

consider.

In this article, I will explain everything in a bit more detail, because I am aware that some of the readers are beginners.

Web Scraper — Example

For the illustration purposes I created a small web scraper.
The web scraper goes to the Medium page and outputs the source of the page.

Nothing complicated.

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("https://medium.com")

print(driver.page_source)


driver.quit()

print("Finished!")
```

Set up our code for Heroku

Add some arguments to Chrome-Options

At the beginning we have to change our code a little bit, so that our web scraper can run on Heroku.

First we have to import the `os` module.
The `os` module provides a portable way of using operating system dependent functionality.

In our case we need it to access Heroku's environment variables.

For those who don't know: an environment variable is made of a name/value pair, whose value is set outside the program.
It affects the way running processes behave on a computer.

```
from selenium import webdriver
import os
```

Now we have to make some necessary settings for our chrome driver.

```
chrome_options = webdriver.ChromeOptions()

chrome_options.add_argument("--headless")

chrome_options.add_argument("--disable-dev-shm-usage")

chrome_options.add_argument("--no-sandbox")
```

- The `headless` argument doesn't open the browser, when the web scraper is running and so it runs in the background. It is also required by Heroku himself, if it has not changed.

To explain the utility of the arguments `--no-sandbox` and `--disable-dev-shm-usage` in detail, we would need another blog post, but in short:

- `sandbox` is an additional feature from Chrome, which aren't included on the Linux box that Heroku spins up for you. Therefore, we do not want to have a sandbox.
- `/dev/shm` is an implementation of the traditional shared memory concept.
  The shared memory space is typically too small for Chrome and will cause Chrome to crash when rendering large pages.
  In the past, the size of the shared memory had to be increased.
  Since Chrome Version 65, this is no longer necessary. Instead, launch the browser with the `--disable-dev-shm-usage` flag.
  This will write shared memory files into `/tmp` instead of `/dev/shm` .

For those who wants to know more, I added following two links:

- no-sandbox: https://www.google.com/googlebooks/chrome/med_26.html
- disable-dev-shm-usage: https://www.cyberciti.biz/tips/what-is-devshm-and-its-practical-usage.html

Storing the paths in Heroku's — Environment Variables

```
chrome_options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")

driver =
webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH")
, chrome_options=chrome_options)
```

As already mentioned in the subtitle, we store the paths for Heroku's environment variables here, once for google-chrome and the chromedriver.

So the complete code should finally look like this.

```
from selenium import webdriver

import os

chrome_options = webdriver.ChromeOptions()

chrome_options.add_argument("--headless")

chrome_options.add_argument("--disable-dev-shm-usage")

chrome_options.add_argument("--no-sandbox")

chrome_options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")

driver =
webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH")
, chrome_options=chrome_options)
```

```
driver.get("https://medium.com")

print(driver.page_source)

print("Finished!")
```

Preparation for deployment

Virtual environment

Depending on the operating system you have (I use Windows) open your command line and change to your project folder.

Depending on the operating system you have (I use Windows) open your command line and change to your project folder.

```
C:\Users\Romik>cd Desktop

C:\Users\Romi\Desktop>cd myproject

C:\Users\Romi\Desktop\myproject>
```

Then, if you don't already have it, install the Python package virtualenv.

```
C:\Users\Romi\Desktop\myproject>pip install virtualenv
```

Now create your virtual environment and give it a name. Usually people give it the name `env` for environment.

```
C:\Users\Romi\Desktop\myproject>virtualenv env
```

And now you have to activate the virtual environment.

```
C:\Users\Romi\Desktop\myproject>env\Scripts\activate

(env) C:\Users\Romi\Desktop\myproject>
```

Requirements and installation of the modules

Now you have to install all modules you use in your project in this virtual environment.

In our case it is only `selenium`, because `os` is already one of the standard modules of Python.

```
(env) C:\Users\Romi\Desktop\myproject> pip install selenium
```

Heroku also requires two files, one is the `requirements.txt` file which lists all your installed modules and their versions, and the other is the `Procfile` to know how to run your code.

```
(env) C:\Users\Romi\Desktop\myproject> pip freeze > requirements.txt
```

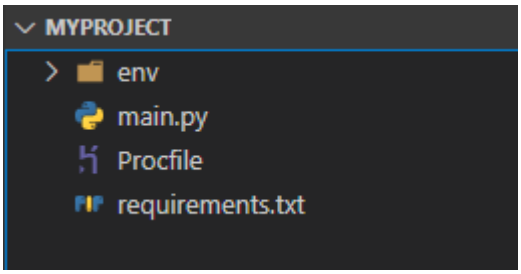
The `requirements.txt` must now be located in your project folder.

```
(env) C:\Users\Romi\Desktop\myproject> echo worker: python main.py >
Procfile
```

Depending on what operating system you have, you will need to use different commands to create a `Procfile` file.
In the end, the `Procfile` file must only contain the following command or content.

You just have to tell the worker in the `Procfile` which command to execute.
In our case, it should execute the Python file with our written code.
Heroku will search for this `Procfile` and then execute the command it contains.

Your folder should look like this.



Deploying to Heroku

First things first

Create an account on Heroku, if you don't already have one.
Don't worry, you won't have to give your credit card details.
It is not necessary for our use case.

After you have logged in, create a new app.

Name the app what you want. The choice of region is not so relevant in our case. So choose for that also whatever you want.

After all these steps, you should have arrived at this view.

Environment variables

Change the tab to Settings.

Do you remeber `"GOOGLE_CHROME_BIN"` and `"CHROMEDRIVER_PATH"` from this code snippet below?

```
chrome_options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")

driver =
webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH")
, chrome_options=chrome_options)
```

We will now use them in our application.

Under the `Config Vars` section, click the `Reveal Config Vars` button and fill out the input fields.

Key: `CHROMEDRIVER_PATH` Value: `/app/.chromedriver/bin/chromedriver`

Key: `GOOGLE_CHROME_BIN` Value: `/app/.apt/usr/bin/google-chrome`

Buildpacks

Buildpacks are scripts that are run when your app is deployed. They are used to install dependencies for your app and configure your environment.

You can also find them under the settings tab, directly below the `Config Vars` section.

These are the three buildpacks we need.

Links for the last two:

- https://github.com/heroku/heroku-buildpack-google-chrome
- https://github.com/heroku/heroku-buildpack-chromedriver

Ready for deployment

Go to the `Deploy` tab, and under the `Deployment method` section, select the `Heroku Git/CLI` option.

Besides the `Heroku Git/CLI` , there are of course the two other methods like `GitHub` and `Container Registry` , but I decided to use the first method in this blog.

Finally, follow the instructions in the `Deploy using Heroku Git` section.
The installation of the Heroku CLI is quite straightforward.

Last step

Now you have to assign a dyno that does the work.
Type `heroku ps:scale worker=1`

In case you are wondering what *dynos* are, in short, they are the heart of Heroku.
You can find more details here: https://www.heroku.com/dynos

Now your code will continue to run until you stop the dyno. To stop the dyno, use the command `heroku ps:scale worker=0`

If you want to check the logs to make sure its working correctly, type `heroku logs --tail`.

Thank you for reading

Web Scraping Heroku Python Selenium