Lousanje Ramírez Navarro

Eagle Eye Networks Skill Test

This document contains a quick rundown of the process followed to get to the solution for the skill test

Assumptions

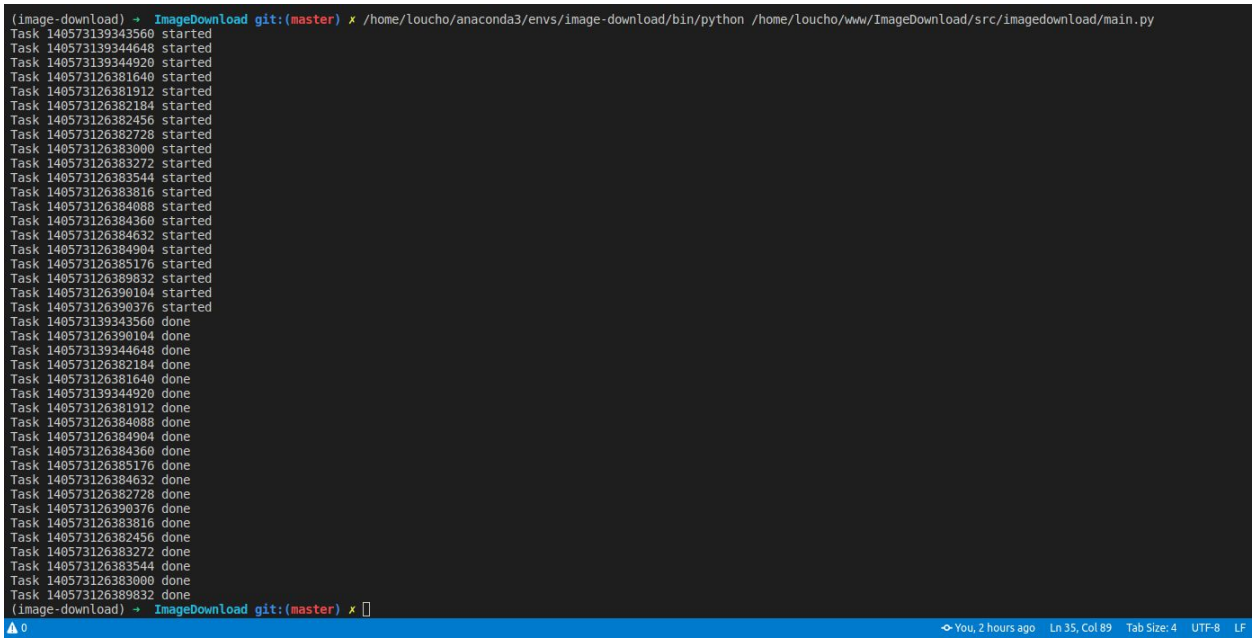
These are some assumptions I made when coding the solution:

- The worker greenlets are coded as the minimum expression of them, that is, they are a single function that calls an asynchronous request to the API
- Concurrency was managed with a Pool to maximize the throughput (at first I thought of making batches of 5 greenlets or gevent.spawns and running those batches sequentially, but a pool seemed like a better option)
- The solution is just really 2 files, since that seemed like an adequate MVP
- monkey.patch_all() was used to make ‘requests’ play nice with gevent
- The een.py file is in great part not my code, I found it on github after a google search and decided to use it as it was a public repo

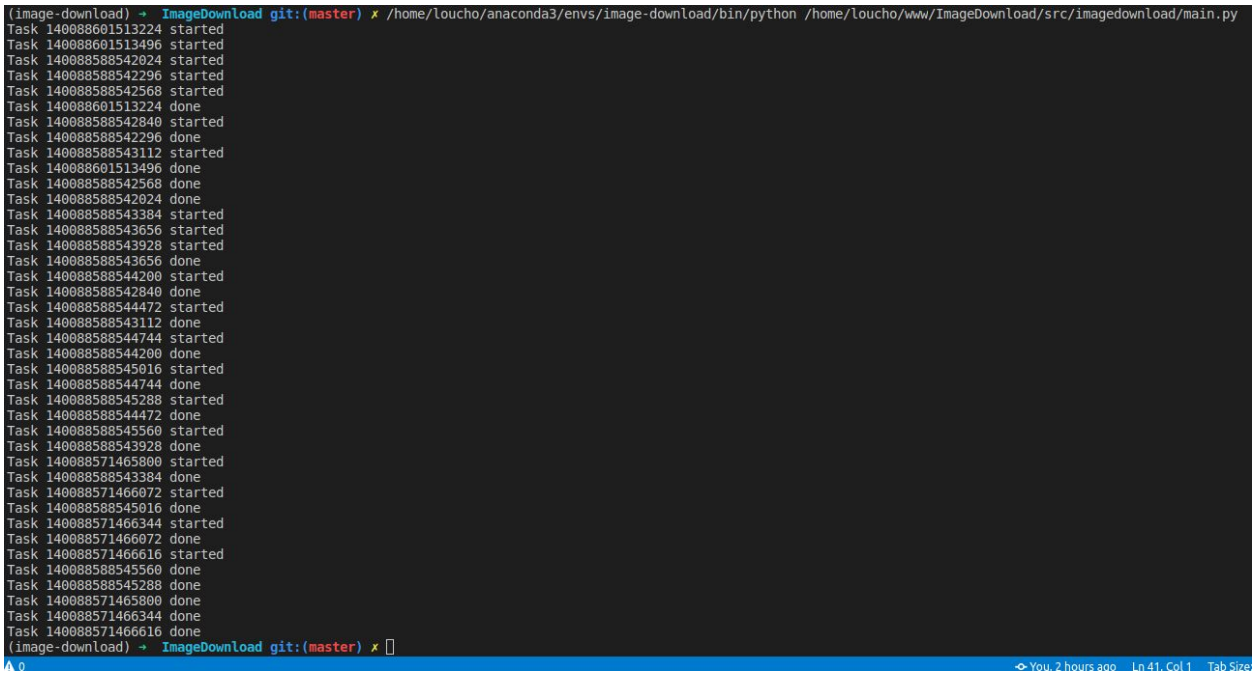
Output

Here are some samples of the script running:

First some proof that it is working asynchronously, all 20 images in 1 shot, we can see all of them start at once and finish in different order:



Now running in a pool of 5, we can see new ones start as soon as a worker is done:



The Requirements

Here are the relevant segments of code to cover the requirements:

Login to the API, grab a random camera:

```
context = een.login("demo@een.com", "bettercloud")
camera_list = context.camera_list()
img_list = []
#seems sometimes there are no images on the camera, let's ensure we get one with at least 20
while len(img_list) < 20:
    #choose a random camera to pull images from (index 1 contains the camera id)
    camera = random.choice(camera_list)[1]
```

Get 20 images (we pass end_timestamp as now and get the latest 20):

```
img_list = context.image_list(esn=camera, asset_type="preview", time=een.timestamp("now"), count=-20)
```

Download them with 5 greenlet workers using gevent (in this case gevent.pool as Pool):

```
#fetch an image and save it the "downloads" folder, include camera name to identify them
def fetch_image(image):
    print('Task %s started' % id(getcurrent())) #get the thread id so we can see the concurrence at work
    img = context.fetch_image(esn=camera, time=image['s'], asset_type="preview")
    fh  = open("downloads/" + camera + "-" + image['s'] + ".jpeg", "wb")
    fh.write(img)
    fh.close()
    print('Task %s done' % id(getcurrent()))

#fetch them all!
#threads = [gevent.spawn(fetch_image, val, index) for index, val in enumerate(img_list)]
#threads = [gevent.spawn(fetch_image, image) for image in img_list]
#gevent.joinall(threads)

#fetch them using a pool of 5 greenlets
pool = Pool(5)
pool.map(fetch_image, img_list)      You, 2 hours ago • added actual code
```

The Repository

The github repo is this: https://github.com/loucho/image-download

Things that I tried or would like to have tried

- Give this a nice UI, even as a console app that could be invoked with some parameters (number of images to download, number of concurrent workers, user and password)
- Encapsulate some of the API responses and some of the behavior in classes (especially since the list of cameras is returning arrays of arrays, so accessing the data is done by the index of the array instead of a named property), this would probably be fun to do, but a bit overkill for the current requirements
- Maybe making it an API endpoint that a webapp can consume and display the results, or something
- Add unit tests and probably research on best practices