

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

Since the deadline was 48 hrs, and I had some left, I went ahead and created a new project using Django and deployed to AWS so you could see it live

Assumptions

These are some assumptions I made when coding the solution:









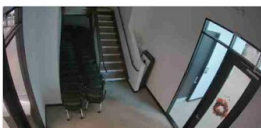
- The worker greenlets are coded as the minimum expression of them, that is, they are a single function that calls an asynchronous request to the API
- Concurrency was managed with a Pool to maximize the throughput (at first I thought of making batches of 5 greenlets or `gevent.spawns` and running those batches sequentially, but a pool seemed like a better option)
- ~~The solution is just really 2 files, since that seemed like an adequate MVP~~
- The above was replaced in this project by a Django webapp that gives a decent ui to consume the API
- `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

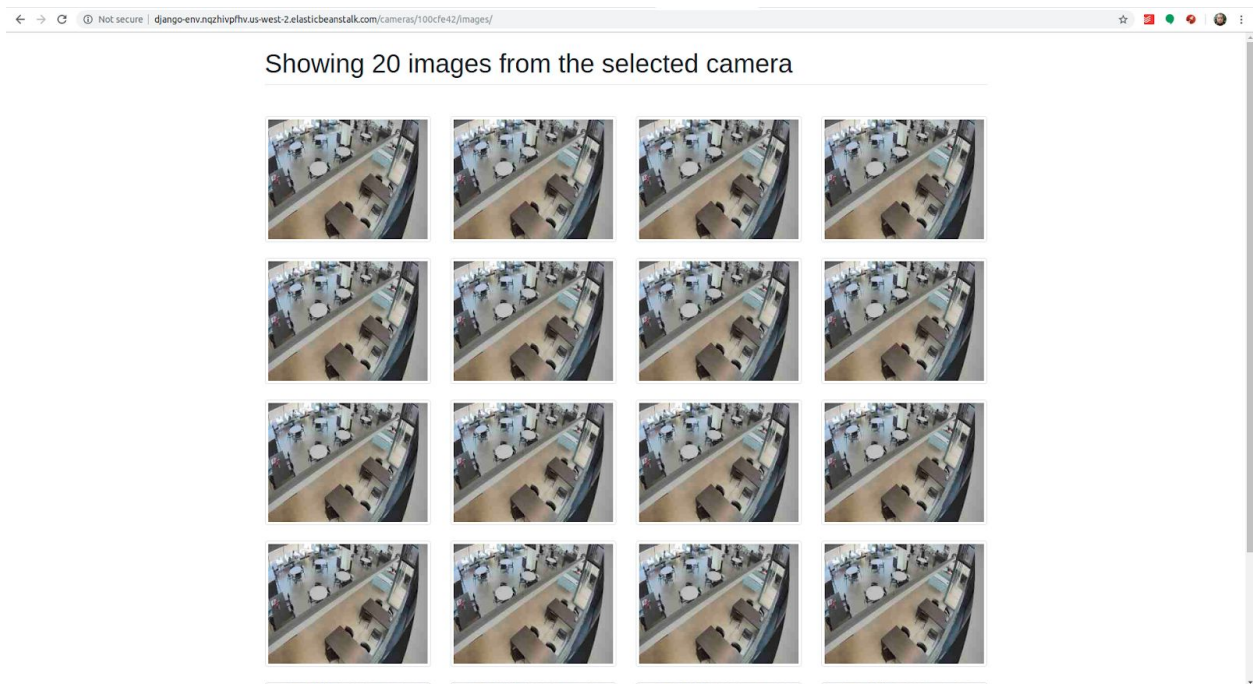
Here are some screenshots of the website running (it is also deployed at <http://django-env.nqzhivpfhv.us-west-2.elasticbeanstalk.com/cameras/>) if you want to see it live:

First page lets you look at 9 random cameras from the demo account (with a preview image):

Grabbing a few random cameras

 <p>CFD1 Common Space ID: 100aa76e View images</p>	 <p>CFA5 Network Closet ID: 10035320 View images</p>	 <p>CFD2 Kitchen Exit ID: 100cfe42 View images</p>	 <p>CFA1 Network Closet 1 ID: 1001870e View images</p>
 <p>CFD1 Prep Room ID: 100bb9cc View images</p>	 <p>CFA1 Stage ID: 1004d2cc View images</p>	 <p>CFA5 Elevator Lobby 1 ID: 10035428 View images</p>	 <p>CFD1 Front Entry ID: 100feea8 View images</p>
 <p>CFD1 Stairwell</p>			

Clicking the button of any of them loads one that has the last 20 images from that camera as thumbnails:



The Requirements

Here are the relevant segments of code to cover the requirements (these didn't really change on this version, just the Django webapp was added):

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)
```

The Repository

The github repo is this: <https://github.com/loucho/django-een>

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~~
- The one above was somehow achieved since it is a web app now
- Add unit tests and probably research on best practices
- There is a pesky error with gevents where it will sometimes not load correctly because of a possible infinite block, refreshing the page fixes it most of the time and then it goes away for a while. I tried to debug it but it only happened while deployed in AWS, so it's kinda hard