A guide to running GRPC based image manipulation for Neuralink Corp's interview
Author: Saurabh Nair

System Requirements:

- **Ubuntu 18.02** or **18.04** (Tested on these two)
- Sudo access (This is crucial, otherwise the package doesn't get installed)

FYI - The setup script creates an environment of its own, so no need to worry about it manipulating other packages / environment on the machine

Repo: https://github.com/saunair/image_manipulation_poetry
(This is a private repo, I could grant access to an interviewer who is curious)

Changes from the assignment's requirements -

1. No build script provided. The setup script in the project root does that.
2. Need to run `sudo ./setup` from the root directory with every fresh environment.
    - ***PATH_TO_PROJECT is the root directory of the project***
3. server and client are commands that can be run within the poetry shell that's created with the aliased commands `server` and `client` and not `./server`
    - No need to be at root level), you can run the command `server` or `client` from anywhere in the project.
4. Added support to choose the number of worker threads and worker cores to use on the server side.
5. Added a mode `--timeit` on the client to run a batch of images from a folder.
    - This was mainly done for testing and timing purposes. Try only if curious
6. A few more options are provided with server and client, you may check them with
    a. `server --help`
    b. `client --help`

FYI - The protobuf files are directly imported in this package. Do not re-generate them.

A guide to running GRPC based image manipulation for Neuralink Corp's interview

Author: Saurabh Nair

Steps to run the project (copied from my git README.md):

Installation

To run the installation, 1. run `sudo ./setup` in the project's root directory.

> *sudo is optional, but recommended. Some permission issues could occur causing installs to fail*
>
> *To verify if the installation worked fine, you may run the unit tests.*

2. run `pytest tests/` from the root directory.

   If all tests pass, the installation worked.

Whenever opening a new terminal to run the server or client, run `./start_terminal` from the root directory to set the correct python environment.

Running the Server

1. Start a new terminal
2. cd into the root directory of this package
3. run: `./start_terminal` *This should make a new poetry shell*
4. Enter the command: `server --host-name MY_HOST --port MY_PORT`

   run `server --help` to know all the options

Running the Client

1. Start a new terminal
2. cd into the root directory of this package
3. run: `./start_terminal` *This should make a new poetry shell*
4. Enter the command: `client --host-name MY_HOST --port MY_PORT --input MY_IMAGE_PATH --output --MY_IMAGE_OUTPUT_PATH`

   run `client --help` to know all the command line options

Example After Installation

> ***Terminal 1:***
> command1: `` `./start_terminal` ``
>
> command2: `` `server` ``
>
> ***Terminal 2:***
> command1: `` `./start_terminal` ``
>
> command2: `` `client --mean --input ./tests/testing_data/image.png --output data/my_output.png` ``

A guide to running GRPC based image manipulation for Neuralink Corp's interview
Author: Saurabh Nair

Features of the implementation:

- The project is written in Python3
- Used Poetry as my package management tool.
- Used Numba to speedup the convolution function
    - Checkout src/image_manipulation/image_utils.py
- Added support for multiprocessor execution of the server (Python's Multithreading is always process-locked)
    - *This sped up the batch execution by 3 times (21 seconds to 7 seconds)*
- Increased the limit of the message size of GRPC's max length to support big images.
    - Had to do this for images that are larger than 4 MB.
- Used gzip compression to compress the byte-string through the client and server
    - This had a minor impact on the speedup.
- Wrote unit(pytest) tests under PATH_TO_PROJECT/tests to check if the build and install got executed correctly
    - The server shouldn't be raising *Any* exceptions. All failed queries are returned with a Null Image.

Possible improvements for production / speed-ups:

1. **Security**
    a. Each connection needs to be validated. Grpc has support for an application level authentication - https://grpc.io/docs/languages/python/alts/
2. **Load balancing on the server side**
    a. A great guide: https://grpc.io/blog/grpc-load-balancing/
    b. Generally any microservice would need to be going through a load balancer. Especially if the system needs to be scaled. Modern load balancers can be setup with ease where a new node can be added without hitting the current operations
3. **Caching on the server's side**
    a. We can use a hash (MD5?) technique at the load-balancer to see if the same request was passed before. If yes, it can use the value in a cache(probably a database of images) and pass it along as the result.
    b. With that method, the only bottleneck would be the memory management of old files and file I/O time.
    c. The cache hit is what'll provide any advantage here. If we expect to send different requests each time, this feature won't be useful.
4. **Network Latency for scale:**
    a. The architect must choose a server close to the client in geography to reduce hops as possible.
5. **Logging**
    a. Currently if there is an error, the server returns a Null image with the exception, which isn't very informative.
    b. Data logging must be an option on the server to log bad queries that caused the code to raise those type of exceptions

A guide to running GRPC based image manipulation for Neuralink Corp's interview
Author: Saurabh Nair

6. **Scalability**
    a. Publish this as a micro-service and run it on AWS.
    b. AWS does the homework of load-balancing + new node addition pretty well.
7. **Implement this in GO and not Python**
    a. With the hindsight of implementing this, I realize why cloud software engineers prefer GO over Python or C++.
    b. The solution will just scale seamlessly with GO, where with my implementation we have to mention the number of threads, processors etc

Conclusion: Thanks to the designer of this question. I had worked with Protobufs before, but not at an RPC / networking level. I also got to explore some new packaging / multi-process frameworks with python. **Time taken: 3.5 days**