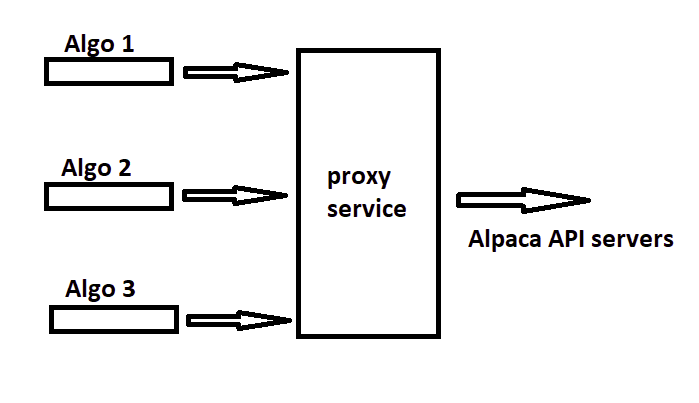
<https://alpaca.markets/learn/alpaca-proxy-agent-01/>

<https://alpaca.markets/learn/alpaca-proxy-agent-02/>

Alpaca only supports one websocket connection at a time. That means you cannot simultaneously get a data stream for more than one algorithm instance. One solution would be to open more accounts, the other is to keep reading ;)

For that purpose I have created the [Alpaca Proxy Agent](https://github.com/shlomikushchi/alpaca-proxy-agent).

Alpaca allows a user to open one websocket, but has much more flexible limits on the data streamed on top of that websocket. In other words, you can get data for multiple algorithms using the same websocket connection. You run the proxy server locally, and the algorithms are using the local network interface to communicate with it.

To execute the alpaca proxy server locally you should:

1. Pull the latest **docker** image from dockerhub (you should always do this to make sure you have the latest changes): docker pull shlomik/alpaca-proxy-agent
2. Choose the data stream source you work with (Alpaca supports 2 data sources: its own, and polygon for funded accounts). Run this to work with the Alpaca data stream: docker run -p 8765:8765 -e USE\_POLYGON=false shlomik/alpaca-proxy-agent

* -p: which local port you expose to the algorithm. You could basically choose whatever you want. E.g: -p 12345:8765 or -p XXXX:8765
* -e USE\_POLYGON as explained, you specify your selected data source here.

Now, once it’s running locally you need to configure your algorithm to connect its websocket to the proxy agent and not directly to the Alpaca servers.

First figure out how docker executes your containers. Is it on localhost (127.0.0.1)? Are you using a docker machine? Is it inside a virtual box (192.168.99.100)? Do a quick google search to figure that out if you don’t know (something like “docker - what is my container URL?” will do the trick).

Each SDK has its own way to connect to the API and it’s documented in the SDKs docs. In the python SDK you could pass the data URL to the websocket instance, or you could define it in an environment variable and the SDK will read it from there:

conn = StreamConn(ALPACA\_API\_KEY,ALPACA\_SECRET\_KEY,

base\_url=URL('https://paper-api.alpaca.markets'),

data\_url=URL('http://192.168.99.100:8765')) # docker url

By doing that, you told the StreamConn instance to connect to the local docker container that runs the proxy server.

The next you need is to set an environment variable called `DATA\_PROXY\_WS` used by internal parts of the sdks. Doing something like this, will do:

set DATA\_PROXY\_WS="ws://192.168.99.100:8765"

or

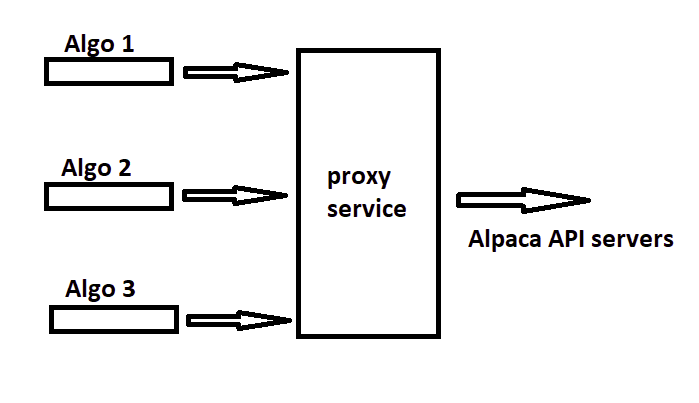
set DATA\_PROXY\_WS="ws://127.0.0.1:8765"

==> decided by your container url

From this point on, you work exactly as you worked before. It’s as simple as that.

## Proxy-Agent Architecture

In its core this project contains a Websocket server and a Websocket client:

* **The websocket server** allows your multiple algorithms to connect to it, and register for specific channels (stock updates, account updates, wild cards, etc..). Each client is stored in a map so when a message is received, it is routed to the right client.
* **The websocket client** connects to the Alpaca API service, and subscribes for all the channels requested by the users' algorithms.
  + When a new user connects, we unsubscribe from the old channels, and subscribes to the new channels (which are the combined channels, the old and the new). => when a new algorithm connects, the other algorithms experience a websocket disconnection. That is not a problem if we do this at the beginning of the trading day (which makes sense) but we must make sure that the clients try to reconnect (which they should do anyway).

The Alpaca python SDK's websocket object, also known as `StreamConn` is based on the [websockets](https://github.com/aaugustin/websockets) python package which is built on top of Python's standard asynchronous I/O framework (asyncio). So this project is built on top of asyncio as well.

Working with asyncio means understanding the eventloop mechanism, and how to work correctly with two threads. To understand asyncio and the event loop see [here](https://realpython.com/async-io-python/) but try to think of it as a multitasking manager that could switch contexts to do many things at once. *For instance, when you request something from an online service. While you wait for the response, there's no reason to hold the execution loop for yourself - give the cpu power to functions that need it. When the response is received, you will get the context of the thread back and you will be able to process it, while a different part of the code may wait for you to complete the processing.*

The eventloop that creates the websocket needs to handle its messages. That seems trivial, but once I introduced another thread (websocket server, and websocket client right?) I faced some difficulties trying to pass the messages from the API back to the algorithm clients thread. This was solved by using a Queue, passing the information between threads.

How to execute 2 "while" loops on the same event loop? So let's first define the tasks that need to be executed:

1. Server listener - waiting for new clients, subscribing to new channels.
2. Websockets client - connecting to the API servers, waiting for new messages to arrive.
3. Sending the received messages to the right algorithm client.

So we run the websocket client on one thread but as I mentioned, the thread that handles the incoming clients, needs to be the one sending them the received messages (same eventloop). so we have here 2 "while" loops:

* one that listens to incoming clients (algorithms)
* one that handles incoming messages( quotes/trades/etc..) and routes them to the right client.

Both should run on the same thread, using the same event loop. For that purpose I use an asyncio method called `asyncio.gather()` which allows us to run this async infinite "while" loops on the same thread.

## Execution/Debugging

As described in the first part, the recommended way to execute the proxy-agent is by using docker. But, you may want to run it locally in an IDE to debug it, or to enhance it. You can. Just do this:

1. clone: git clone https://github.com/shlomikushchi/alpaca-proxy-agent.git
2. cd into the folder
3. create a virtualenv (recommended)
4. install it and requirements: pip install -e .
5. open main.py in the IDE and fire away.

Security: The communication between your algorithm and the proxy-agent does not use ssl. The communication between the proxy-agent and the Alpaca server does. So make sure you do not expose the proxy-agent's port to the outside network (especially if you run on the cloud)

## Project structure

* main.py -- accept new clients + subscribe to new channels + send receive messages(quotes/trades/...) back to clients
* shared\_memory\_obj.py -- just a python module trick, define objects (dictionaries, queues) in a shared module - they will be created once. you can call it a poor man's Singleton.
* server\_message\_handler.py -- handles received messages from the api servers
* defs.py -- shared constants and Enums
* docker files
  + Dockerfile used to build the production docker image
  + Dockerfile-dev used to build a debugging docker image (could be used to run in an IDE)
  + docker-compose.yml - docker compose file to execute the production image locally.
  + dev.yml - docker compose file used to run the container locally in debug mode