Contents

[Pythonanywhere 1](#_Toc82159508)

[Google Cloud 1](#_Toc82159509)

[AWS 4](#_Toc82159510)

# Pythonanywhere

[Pylivetrader](https://github.com/alpacahq/pylivetrader) is the package that will allow Alpaca to understand Quantopian algos. Pylivetrader will only work with Python 3.6.

you’ll need to create a virtual environment with Python 3.6 and install pylivetrader

Open a bash on PythonAnywhere and type the following commands:

$ python3.6 -m venv venv  
$ source venv/bin/activate  
(venv)$ pip install pylivetrader

All the packages installed inside the virtual environment will only be accessible inside the same, here called ‘venv’. The Python version inside ‘venv’ will also be restricted to 3.6 (in this particular case); however, you can still use other python versions outside of the ‘venv’. In order to do so, simply type deactivate.

(venv)$ deactivate

You can jump right back into the virtual environment by typing:

$ source venv/bin/activate

Go to tasks, scroll all the way down to ‘Always-on tasks’, and type the following command:

source venv/bin/activate && pylivetrader run -f algo.py --backend-config config.yaml

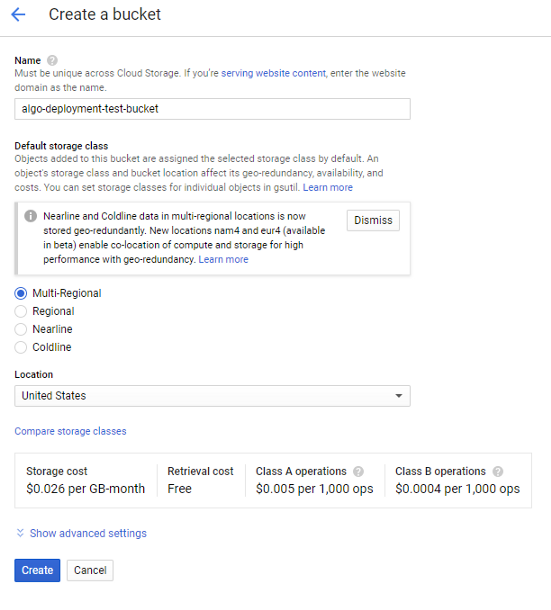
# Google Cloud

Once you’ve signed up for a GCP account, head to the [GCP console](https://console.cloud.google.com/) and create a project.



Hit “New Project”, and you can name it anything you like. I’ve named mine “Algo Deployment Test”.

Once you’ve created a project, go to the sidebar and navigate to Compute Engine > VM Instances. In this panel, you can see any instances you’ve created. That might be blank for now, but let’s go ahead and make one. Hit Create and fill out your configuration like this.



The only things you need to change are the name, which can be whatever you want, the “Boot disk” — we want Ubuntu 18.04 LTS - and the “Machine Type.” For machine type, choose the “micro” option there at the top to get the free tier. You’ll know that it’s free because the panel on the right will say that you have several hundred hours of f1-micro instance usage for free this month.

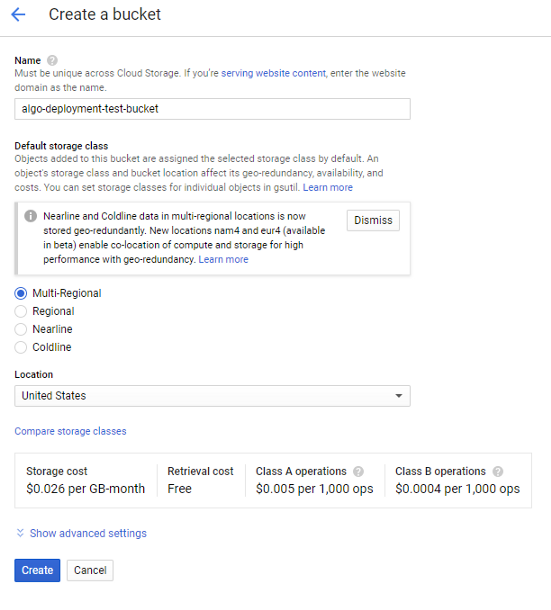
Hit Create at the bottom, and after a short bit of watching a circle spin, you’ll have a machine up and running, visible in the VM Instances panel. It’ll look something like this:



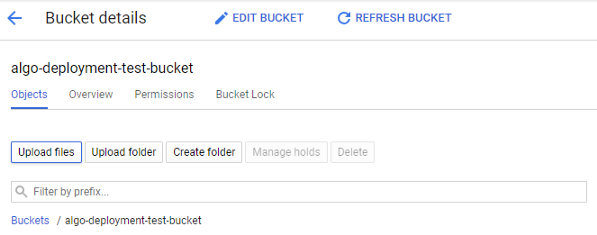
Your new compute instance where we’ll be running the script.

You can click Open in browser window to get a shell to the machine. Go ahead and do that now — we’ll need it in a minute. But before we type anything into the terminal, we need to get our script onto the machine. (You might know of some ways to do that through the terminal — if so, go for it! Or, you can follow along to do it through the GCP console interface.)

In the sidebar, scroll to Storage > Browser. On the Storage page, go ahead and hit Create Bucket. You’ll get a prompt like this:

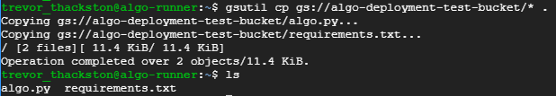


We’re not too concerned about the storage cost, since we’ll be storing much less than 1 GB and only sharing it with our one instance. (And we have a few hundred dollars in credit anyhow, if you’ve signed up for the trial.) Once your bucket is created, it’s easy to put your files into it.



Hit “Upload files” to get files from your local machine into GCP.

Upload the requirements.txt and algo.py files you checked out from the GitHub repository and plugged your Alpaca API credentials into. We’re almost done! All that’s left is to get the files onto our instance. In the browser terminal you’ve opened, you can type gsutil cp gs://<your-bucket-name>/\* . to download your files onto the VM.



The gsutil command pulls down files from the Google Storage system.

Now that the script is on the cloud instance, we just need to run it. Python is already installed, but to make requirement management easy, go ahead and download and install pip. In the VM terminal, do this:

sudo apt update

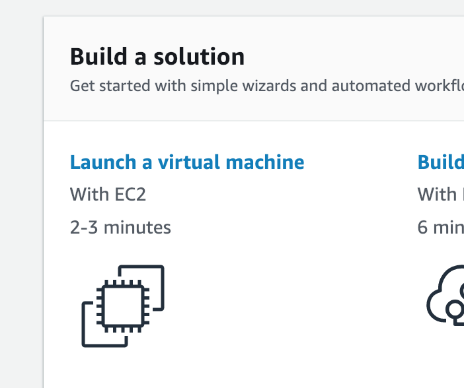
sudo apt install python3-pip

And then install the algorithm’s dependencies:

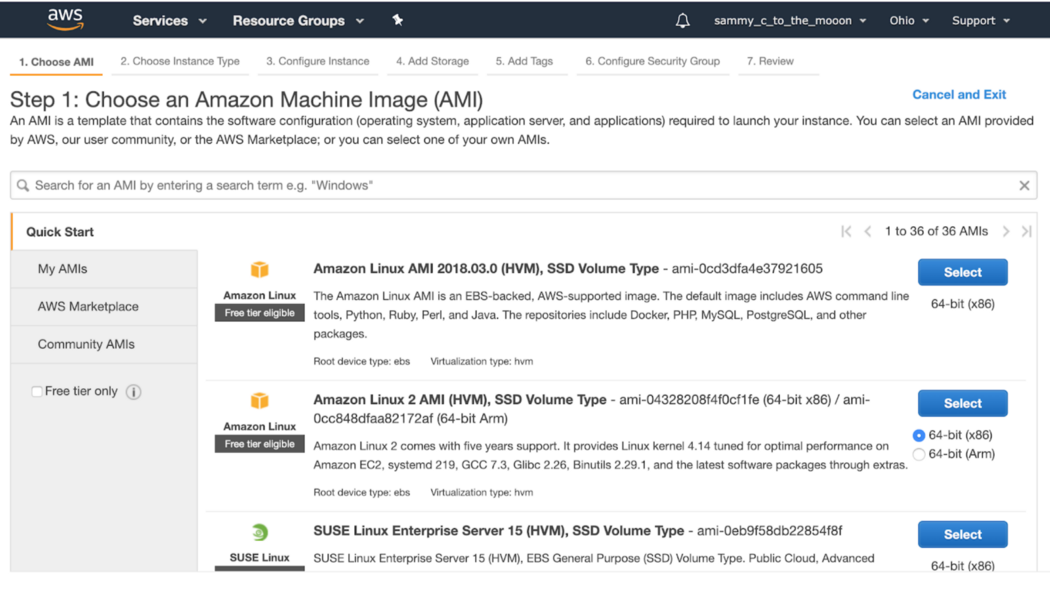
sudo pip3 install -r requirements.txt

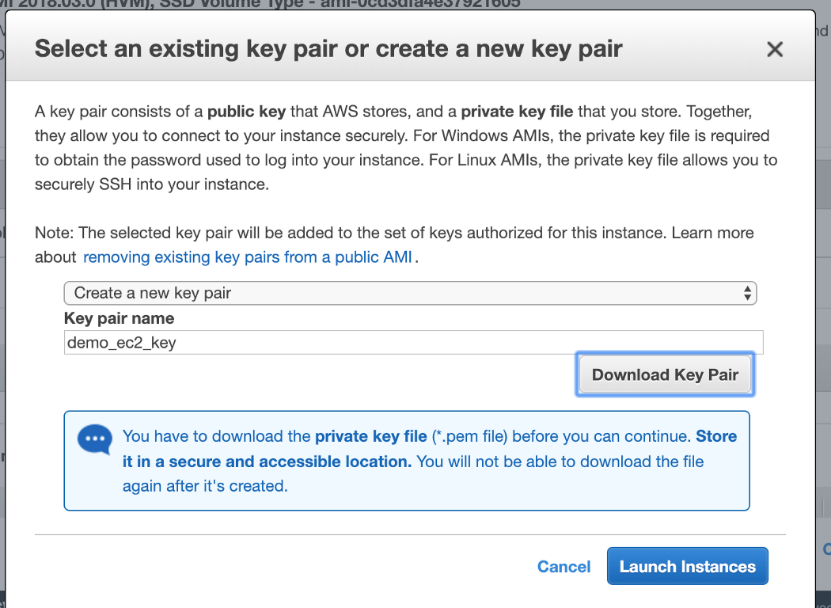
And that’s it! Now, when you’re ready to get the algorithm running, you can just type python3 algo.py into the VM’s terminal and watch it get to work. (At this point, you can also delete the bucket you created, if you’re worried about a few pennies in storage fees.)

# AWS

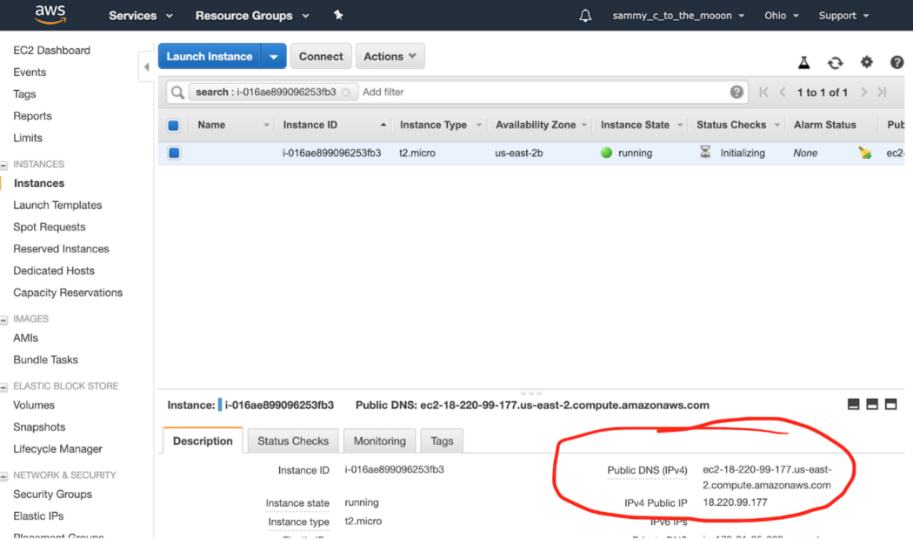
When you sign in, “Launch a virtual machine” with EC2 on your front page. Click that. (Or, search “EC2” and click the “Launch Instance” button.)

Click the “Amazon Linux 2 AMI” free tier eligible option. Use the t2.micro free tier eligible and click “Review and Launch”.



You’ll get notified to set a security group if this is your first time create a new key pair and download the .pem file. **You’ll need the path to this file later.**

Click “Launch Instances”. Click the instance id link on the following page to go to your EC2 instances.



Your instance will take a bit to initialize, while it’s doing that, copy your public DNS.

Open a terminal window (I’m doing this from Mac OSX) and type the following:

ssh -i [path to your key here] ec2-user@[your public DNS here]

My example:

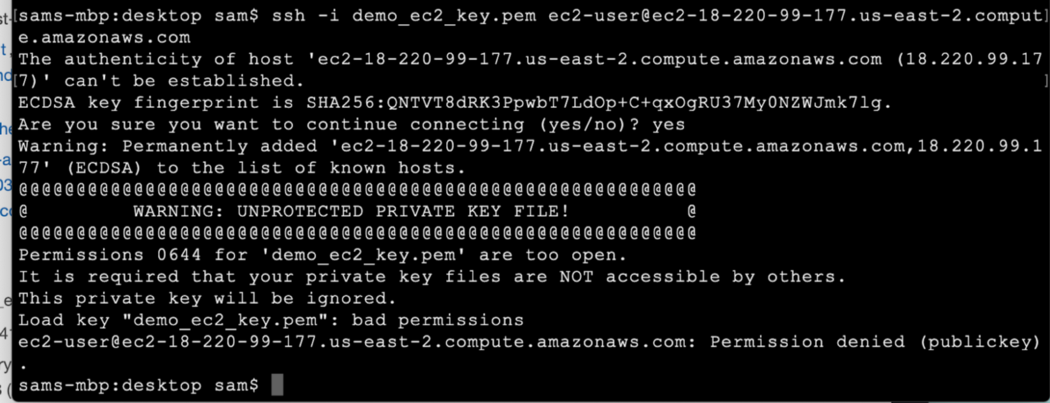
ssh -i demo\_ec2\_key.pem ec2-user@ec2–18–220–99–177.us-east-2.compute.amazonaws.com

(If you ever use a different image, your log-in may be ubuntu@[yourDNShere])

\*Some troubleshooting may be required. In this case, my connection was timing out, so I searched “EC2 timeout” and found the solution. In my security group, I had to specify port 22 as inbound traffic. This was because I originally skipped the security group step above, by default your security group should include port 22 for inbound traffic.

## **How to fix your key permissions**

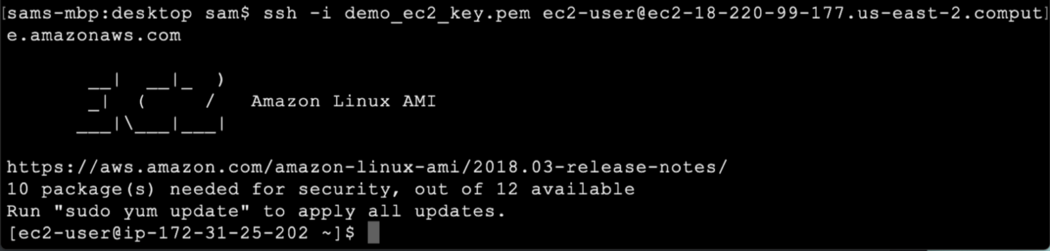
The first time you try to connect, you’ll probably get “bad permissions” as a result:



All you have to do fix this is run:

sudo chmod 600 [path to your key]

Now run the command again. This screen means you did everything correctly:



# Step 4.) Set up Alpaca Trade API.

Enter python -V to see what version of python you’re running. By default, I get 2.7. Run the following to install Python3.7:

sudo yum install python37

pip3 install --user alpaca-trade-api

Now your EC2 instance should be good to go! Keep this window up for later.

Open a new terminal window, and run the following:

scp -i [path to your key] [path to your algo] ec2-user@[your public DNS]:/~

My example with apca\_5min\_ema.py as a strategy file name:

scp -i demo\_ec2\_key.pem apca\_5min\_ema.py ec2-user@ec2–18–220–99–177.us-east-2.compute.amazonaws.com:~/

Go over to your original EC2 window, and run ls to check if the upload worked.

You should see your algorithm listed in the directory. Test it by running python3 apca\_5min\_ema.py.

# Step 7.) Closing your terminal without quitting your algo.

## Start a new instance of screen

In order to keep our algo running without quitting when we disconnect, we can use a handy Linux command, screen. Go ahead and run it.

This will pop up a new, blank terminal. It’s actually another window of your terminal. Now run python3 apca\_algo.py.

Hit **CTRL + A + D** to detach the screen and return to your normal terminal.

Now you can type screen -ls to see your process is still running.

Typing tail apca\_log.log confirms this as well, seeing the results in our log file.

Hit **CTRL + D** to logout of EC2 altogether.

Log into your Alpaca account, and confirm your orders are being placed.

## Reconnecting to your screen

To get back to your algo, log into EC2 with SSH.

Run screen -ls to see what screens are running. You should see something like:

There is a screen on:  
 3634.pts-0.ip-172–31–34–247 (Detached)  
1 Socket in /var/run/screen/S-ec2-user.

That long screen id is what you’ll type in next to reconnect:

screen -r 3634.pts-0.ip-172–31–34–247

Alternatively, if you just want to quit the screen you can use ps aux | grep apca\_5min\_algo.py to see the process ID:

ec2-user 3658 0.1 6.4 427400 65388 pts/1 S+ 22:14 0:00 python3 **apca\_5min\_ema.py**ec2-user 3726 0.0 0.1 119468 1040 pts/0 S+ 22:20 0:00 grep — color=auto **apca\_5min\_ema.py**

Then, you can use screen -XS [process id] quit to quit the screen immediately. My example:

screen -SX 3658 quit

Finally, confirm if you’ve quit your algo with screen -ls .

There you have it! This should give you a good starting point for spinning up new EC2 instances, navigating AWS, and managing your algorithms.