

Supplementary AWS Documentation

The purpose of this document is to detail how the current deployment on AWS Lightsail was created, and how to go about making modifications when necessary.

A detailed video recording of all these steps has already been recorded, and is available here (<https://drive.google.com/file/d/1CqEGa9kHOY9ypZ9VthGmUb6plNC8KYww/view?usp=sharing>) will perhaps share greater practical insight.

Nevertheless, am creating this document as well - could be useful for quickly searching up commands, for instance.

AWS is of course the well known suite of cloud computing services that offers a broad set of storage, database and analytics solutions.

AWS Lightsail is a service within AWS that offers easy to use Virtual Private Server (VPS) instances, among other things. It is designed for developing and developing applications easily without worrying too much about the underlying infrastructure.





So if one navigates to the AWS Lightsail panel (as shown in the video) once can view all the existing instances.





Good morning!

Filter by name, location, tag, or type


Sort by **Date** ▾

Create Instance

	ubuntu-sample 2 GB RAM, 2 vCPUs, 60 GB SSD	 
Running	13.200.127.231 	
2406:da1a:b07:7400:c91b:5918:a769:e31f		
Mumbai, Zone A		

	Ubuntu-Instance-New 2 GB RAM, 2 vCPUs, 60 GB SSD	 
Running	3.109.24.178 	
2406:da1a:b07:7400:38da:916d:8f8c:d35c		
Mumbai, Zone A		

'Ubuntu-Instance-New' is the instance which is actually running the live programs used by the system. 'ubuntu-sample' was just created for testing and demonstration purposes.

	Ubuntu-Instance-New
2 GB RAM, 2 vCPUs, 60 GB SSD	
Ubuntu	
Mumbai, Zone A (ap-south-1a)	

The specifications of the instance server can be viewed, including its memory and RAM capacity. These are the best specs available in the free tier, and are sufficient for the current system.

Now for creating a new similar instance (if required in the future), after hitting the 'Create Instance' button, one must choose their blueprint or template.

The instance used in the system is based on Ubuntu (20.04 precisely) – one of the more widely used templates. The exact specs chosen as the blueprint can be seen below.

Instance location ?

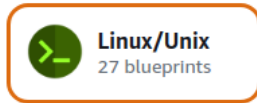


You are creating this instance in **Mumbai, Zone A** (ap-south-1a)

[Change AWS Region and Availability Zone](#)

Pick your instance image ?

Select a platform



Select a blueprint

Apps + OS

OS Only



Amazon Linux
2023
2023.1.20230...



Amazon
Linux 2
2.0.20230822.0



Ubuntu
22.04 LTS



Ubuntu
20.04 LTS

Next, one must choose an instance plan.

The below configuration has the best specification of those that have a free trial, and are sufficient for the system.

Choose your instance plan ?

New! Check out our new 16 GB and 32 GB RAM bundles!

Sort by: **Price per month** Memory Processing Storage Transfer

First 3 months free!

\$3.5

USD

\$3.50 USD

512 MB

2 vCPUs

20 GB SSD

512 GB

First 3 months free!

\$5

USD

\$5 USD

1 GB

2 vCPUs

40 GB SSD

1 TB

First 3 months free!

\$10

USD

\$10 USD

2 GB

2 vCPUs

60 GB SSD

1.5 TB

\$20

USD

\$20 USD

4 GB

2 vCPUs

80 GB SSD

2 TB

\$40

USD

\$40 USD

8 GB

2 vCPUs

160 GB SSD

2.5 TB

Price per month

Memory

Processing

Storage

Transfer

Then give a name to the instance, and hit create. The instance should be ready in a couple of minutes.

Now a couple of steps will have to be performed at the beginning to ensure smooth functioning of the system's code.

The dashboard and trading flask python codes run on certain ports of the system, and must be able to receive requests from Tenderly. Thus, these ports need to be open.






However by default, the instance will only have a couple of ports open - the rest are blocked by a firewall. (in the 'networking' tab).

IPv4 Firewall

Create rules to open ports to the internet, or to a specific IPv4 address or range.

[Learn more about firewall rules](#) 

 Add rule



Application	Protocol	Port or range / Code	Restricted to		
SSH	TCP	22	Any IPv4 address Lightsail browser SSH/RDP 		
HTTP	TCP	80	Any IPv4 address		

To change this, one must add a new rule, choosing the 'All Protocols' option as follows

 Add rule

Specify a port and protocol to open. Specify a port range using a dash, such as 0 - 65535.

Application: Protocol: Port or range: ☐ Restrict to IP address

Cancel  Create 
Duplicate rule for IPv6 ☒

This will open all ports in the system.

Also by default, the ip address of the created instance will not be static - which means that if the instance is rebooted or refreshed, its ip address will change.

This is very inconvenient as for codes running on the instance to have trading permissions, the instance's ip address will have to be whitelisted in the binance trading account.

The solution is to create and attach a static ip address to the system. To do this, go to the 'Networking Section' and choose the 'Create static IP' option.



Create a static IP address

A static IP is a fixed, public IP address that you can assign and reassign to your instances.

Static IP location ?



You are creating this static IP in **Mumbai, all zones** (ap-south-1)

[Change region](#)

Attach to an instance

Attaching a static IP replaces that instance's dynamic IP address.

Static IP addresses can only be attached to instances in the same region.



▼


ubuntu-sample

A static IP address will then be assigned to the instance, and will be visible next to the instance's name.

Static IP: **3.109.24.178** 

Private IP: 172.26.12.216

Public IPv6: 2406:da1a:b07:7400:38da:916d:8f8c:d35c


[Learn more about IPv6](#) 

The assigned static ip address is the address that would require binance permissions for trading.

Now, we would need to be able to push programs (mostly `python` code in our case) and files from our local machine onto the remote server instance, in order to run them there.

In order to get the permissions to do so, we would first need to download the SSH key, a `.pem` file, which helps in SSH authentication.

Use your own SSH client

[Connect using an SSH client](#) 

CONNECT TO

3.109.24.178

IPv6: 2406:da1a:b07:7400:38da:916d:8f8c:d35c

USER NAME

ubuntu

SSH KEY

This instance uses your current **default** SSH key for this region.



[Download default key](#)

Now the command to be run on the terminal to do this would be
'scp -i [perm_file] [local_location] ubuntu@[ip]:[remote location]'

Here, 'scp' is a secure copy utility used to transfer files between locations using the Secure Shell (aforementioned SSH) protocol. The -i flag is just used to specify the file to use for authentication.

[perm_file] refers to the location of the .pem file that was downloaded as the SSH key.

[local_location] refers to the location of the file you want to push, on your local device.

The 'ubuntu' part is common for all instances that use an ubuntu template, as does ours.

[ip] refers to the static ip address of the instance.

[remote location] refers to the location on the remote server instance where we want to push the file.

For example, the command may look like this

```
'sudo scp -i LightsailDefaultKey-ap-south-1.pem  
instance_demo.py ubuntu@13.200.127.231:instance_demo.py'
```

In the video, I have demonstrated using this command to push a sample python code onto the lightsail instance and run it.

As further demonstrated, if the code is run simply as say 'python3 program.py', once the terminal window is closed, the code will stop running.

However this cannot be allowed to happen, as our trading/dashboard codes must keep running in the background. To solve this problem, we make use of screens.

A screen is essentially a virtual terminal that allows one to run multiple terminal screens within a single window, especially used for executing long running tasks.

We can create a screen, run our command there, then detach from the screen and close the terminal. Our command will continue to run.

In the video tutorial, I have demonstrated in detail how to implement and work with screens, using a sample python code that continuously writes to a dashboard.

The basic commands that were required to accomplish this are:

'screen -S <screen_name>' to create a new screen

'ctrl+a+d (all at once)' to detach from the screen

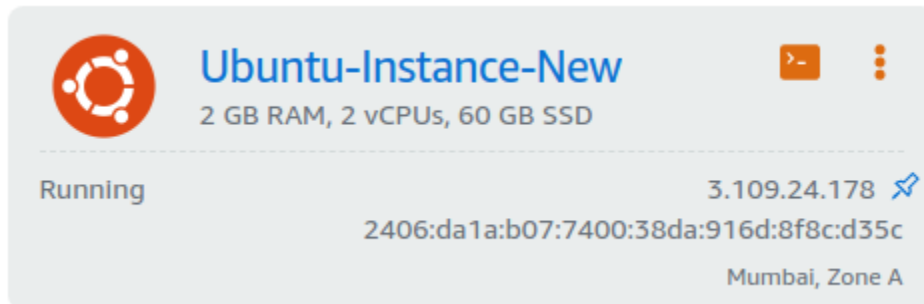
'screen -r <screen_name>' to go back to the detached screen again

'screen -list' to list all the currently running screens, and there screen IDs

'screen -X -S <session-id>' quit to terminate/delete a screen

Alternatively, once can navigate the screen and execute 'exit' to accomplish this.

Now, will be giving a brief overview of the current deployment. The deployment is entirely hosted on the 'Ubuntu-Instance-New'



On connecting to the instance using SSH and running `screen -list`, we can see the following three screens running.

```
ubuntu@ip-172-26-12-216:~$ screen -list
There are screens on:
  21535.live_trading_screen      (08/08/23 17:49:21)      (Detached)
  20663.closing_trades_dashboard_screen (08/08/23 17:22:10)    (Detached)
  2554.open_trades_dashboard_screen (08/08/23 16:20:32)    (Detached)
```

As the names suggest, these screens run the trading screen, the closing trades dashboard handling screen, and the opening trades dashboard handling screen respectively.

A look at the code shows that they run on the ports 80, 20 and 44 respectively.

On entering the live trading screen, we can see that the code has been run using this command

```
ubuntu@ip-172-26-12-216:~/live_trading$ sudo python3 gmx_open_trading.py >> log_output.txt
2>&1
```

This way, both the output and error logs of the file are written to the file 'output.txt'.

One can view the contents of the file using 'cat output.txt', 'less output.txt' or any other editor.

If one goes to any of the dashboard screens, one can see the latest output and logs as they appear

```
xc2ed4ebc97f4a23830763573c3f5487e1b50c527d51c994bc741a77e497499e7
'0xc2ed4ebc97f4a23830763573c3f5487e1b50c527d51c994bc741a77e497499e7', '0x4dee5967b394d6fd
eaaef99ab4c16a517244050', 'SHORT', 'SELL', 'ETH', 0.2802, 1811.82, 507.67, '2023-08-17 03
59:15', 'No', 150.973, 47.61051019796954]
:ffff:34.107.120.110 - - [2023-08-16 22:29:17] "POST / HTTP/1.1" 200 117 2.246652
x658a007f5e690b662a33fcf93bb0f3638d3573a2c8f631ff3af317c51af99ff0
'0x658a007f5e690b662a33fcf93bb0f3638d3573a2c8f631ff3af317c51af99ff0', '0x328fe286b2be1895
59e2601364afe9daa22aba3', 'LONG', 'BUY', 'LINK', 7.4, 6.739, 49.87, '2023-08-17 04:00:55'
'No', 0, 1.0987913949815242]
:ffff:35.246.199.121 - - [2023-08-16 22:30:57] "POST / HTTP/1.1" 200 117 1.787639
```

One can view further previous logs by simply scrolling – however there is an upper limit by default.

To obtain infinite scrolling, one can enter copy mode.

To enter into copy mode, one uses ctrl + A, and then hits the escape key.

```
, 'No', 0, 1.0987913949815242]
::ffff:35.246.199.121 - - [2023-08-16 22:30:57] "POST / HTTP/1.1" 200 117 1.787639
Copy mode - Column 1 Line 34(+1024) (90,34)
```

To exit copy mode, hit escape once again.

— — — —

Thus, the document details how to go about working with AWS, how the current deployment is set up, and how to modify or create new instance deployments in the future.

It is recommended to watch the detailed demonstration video for further practical information on AWS and the deployments.

Again, it can be found at

<https://drive.google.com/file/d/1CqEGa9kHOY9ypZ9VthGmUb6p1NC8KYww/view?usp=sharing>