

Documentación proyecto 1

1. Cree 4 instancias de aws y la regla para que escuchen por el puerto 8080

Recientes

Inicio rápido

Ubuntu

Windows

Red Hat

SUSE Linux

Debian

Buscar más AMI

Incluidas las AMI de AWS, Marketplace y la comunidad

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-08c40ec9ead489470 (64 bits (x86)) / ami-0f69dd1d0d03ad669 (64 bits (Arm))
Virtualización: hvm Habilitado para ENA: true Tipo de dispositivo raíz: ebs

Apto para la capa gratuita

Descripción

Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2022-09-12

Arquitectura

64 bits (x86)

ID de AMI

ami-08c40ec9ead489470

Proveedor verificado

▼ Tipo de instancia

Información

Tipo de instancia

t2.micro
Familia: t2 1 vCPU 1 GiB Memoria
Bajo demanda Linux precios: 0.0116 USD por hora
Bajo demanda Windows precios: 0.0162 USD por hora

Apto para la capa gratuita

Comparar tipos de instancias

Instancias (4)

Información

Find instancia by attribute or tag (case-sensitive)

Conectar

Estado de la instancia

Acciones

	Name	ID de la instancia	Estado de la instancia	Tipo de instancia	Comprobación de estado	Estado de la instancia	Zona de disponibilidad	DNS de IPv4 pública	Dirección
<input type="checkbox"/>	patico3	i-08d72f1a7de3874d3	En ejecución	t2.micro	Inicializando	Sin alarmas	us-east-1a	ec2-3-94-132-166.com...	3.94.132.
<input type="checkbox"/>	patico2	i-0f04dc0ec2e969843	En ejecución	t2.micro	Inicializando	Sin alarmas	us-east-1a	ec2-54-197-5-88.comp...	54.197.5.
<input type="checkbox"/>	Patomiao	i-030aacecf8a649a73	En ejecución	t2.micro	Inicializando	Sin alarmas	us-east-1d	ec2-3-86-204-223.com...	3.86.204.
<input type="checkbox"/>	patico1	i-054017910e28fb7ce	En ejecución	t2.micro	Inicializando	Sin alarmas	us-east-1d	ec2-3-86-184-12.comp...	3.86.184.

2. Una vez creadas las instancias y nombradas para diferenciarlas se procede a crear el código del proxy utilizando python
 - a. En el habrán 4 archivos primero constantes.py el cual esta encargado de mantener las constantes

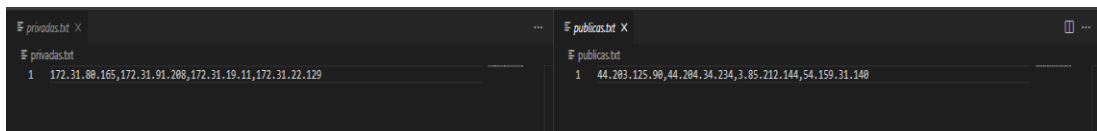
```
constants.py X
constants.py
1
2 privadas = open('/home/ubuntu/pibl/proxy/privadas.txt')
3 priv= privadas.read().split(",")
4 publicas = open('/home/ubuntu/pibl/proxy/publicas.txt')
5 publ= publicas.read().split(",")
6
7 IP_SERVER=str(priv[0])
8
9 IP_SERVERS=[publ[1],publ[2],publ[3]]
10
11
12 PORT = 8080
13 ENCONDING_FORMAT = "utf-8"
14 RECV_BUFFER_SIZE = 2048
15 QUIT = 'QUIT'
16 print(IP_SERVERS)
17 print(IP_SERVER)
```

b. Pb.py encargado de hacer el proxy y el balanceador de carga

```
pb.py X
pb.py
1 import socket
2 import constants
3 import logging
4 import time
5 host , port = constants.IP_SERVER, constants.PORT
6 serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 logging.basicConfig(filename = 'Register.log', format="%(asctime)s %(levelname)s %(message)s", level = 'INFO')
8 def main():
9     serversocket.setsockopt(socket.SOL_SOCKET , socket.SO_REUSEADDR , 1)
10    serversocket.bind((host , port))
11    serversocket.listen(1)
12    print('servidor en el puerto',port)
13    ip_index = 0
14    while True:
15        connection , address = serversocket.accept()
16        request = connection.recv(1024).decode('utf-8')
17        print(address)
18        print(request)
19        logging.info('Consulta: %r\n recibida en el puerto: %r\n',request, port)
20        data_received = send(request, ip_index)
21        print(data_received)
22        response = data_received
23        response = str(data_received.decode(constants.ENCONDING_FORMAT))
24        connection.send(response.encode(constants.ENCONDING_FORMAT))
25        print(response)
26        logging.info('respuesta: %r\n enviada por el puerto: %r\n', response, port)
27        if ip_index <= len(constants.IP_SERVERS)-1:
28            ip_index+=1
29        else:
30            ip_index = 0
31    def send(command_to_send, ip_index):
32        clientsocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33        clientsocket.connect((constants.IP_SERVERS[ip_index], constants.PORT))
34        if command_to_send == '':
35            print('Please input a valid command...')
36        else:
37            clientsocket.send(bytes(command_to_send, constants.ENCONDING_FORMAT))
38            data_received = clientsocket.recv(constants.RECV_BUFFER_SIZE)
39            return data_received
40    if __name__ == '__main__':
41        constants
42        main()
```

c. Y por ultimo dos archivos txt llamados privadas.txt y publicas.txt

Siendo el publicas el único que se modifica ya que las ips privadas nunca cambian



3. Procedemos a crear los archivos que irán en cada uno de los servidores
 - a. Constantes.py para manejar las que siempre quedan concretas

```
PORT = 8080
ENCODING_FORMAT = "utf-8"
RECV_BUFFER_SIZE = 2048
privadas = open('/home/ubuntu/patico2/privadas.txt')
priv= privadas.read().split(",")

IP_SERVER=str(priv[1])
```

- b. Pato.py que será el server el cual correrá el framework en este caso un HTML

```
1 import socket
2 import constantes
3 import os
4
5 host = constantes.IP_SERVER
6 port = constantes.PORT
7
8
9 def listen():
10     serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
11     serversocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
12     serversocket.bind((host, port))
13     serversocket.listen(1)
14     print('servidor en el puerto',port)
15     return serversocket
16
17
18 def temp(serversocket):
19     while True:
20
21         connection, address = serversocket.accept()
22         request = connection.recv(1024).decode(constantes.ENCODING_FORMAT)
23         print("-----request-----")
24         string_list = request.split(' ')
25
26         if request == "":
27             temp(serversocket)
28
29         requesting_file = string_list[1]
30
31         print('Client request',requesting_file)
32
33         myfile = requesting_file.split('?')[0]
34         myfile = myfile.lstrip('/')
35
36         if(myfile == ''):
37             myfile = 'index.html'
38         if(myfile == 'favicon.ico'):
39             temp(serversocket)
40
41         try:
42             file = open(myfile, 'rb')
43             response = file.read()
44             file.close()
45
46             header = 'HTTP/1.1 200 OK\r\n'
47
48             if(myfile.endswith('.jpg')):
49                 mimetype = 'image/jpeg'
50             elif(myfile.endswith('.ico')):
51                 mimetype = 'image/x-ico'
52             else:
53                 mimetype = 'text/html'
54
55             header += 'Content-Type: '+mimetype+'\r\n\r\n'
56
57             final_response = header.encode(constantes.ENCODING_FORMAT)
58             final_response += response
59             send(connection,final_response)
60
61         except Exception as e:
62             print("-")
63             header = 'HTTP/1.1 404 Not Found\r\n'
64             response = '<html><body>Error 404: File not found</body></html>'.encode(constantes.ENCODING_FORMAT)
65             send(connection,response)
66
67
68 def send(connection,final_response):
69     connection.send(final_response)
70     connection.close()
71
72
73
74 if __name__ == '__main__':
75     soc = listen()
76     temp(soc)
```

- c. El index.html el cual sera el framework utilizado para la mostrar la pagina

```
index.html X
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Servidor2</title>
8  </head>
9  <body>
10     <center><h1>El servidor #2 esta funcionando</h1></center>
11 </body>
12 </html>
```

- d. Y por último privadas.txt el cual se encargará de almacenar todas las ips privadas

```
privadas.txt X
privadas.txt
1  172.31.80.165,172.31.91.208,172.31.19.11,172.31.22.129
```