Python: cómo obtener la dirección IP pública

Este asunto tenía tiempo en nuestras mentes hasta que este vuestro sitio web estuvo "caído" durante varias horas debido a que nuestro propio proveedor de alojamiento web decidió instalar los certificados necesarios para poder establecer comunicaciones seguras y encriptadas entre los navegantes y nuestro servidor web (**TLS** o **https**, como quieran verlo). Por cierto que lo recibimos, dicho servicio, con mucho gusto y justificamos el tiempo fuera de línea debido a este evento (todo esto lo especulamos porque aún no hemos recibido un comunicado explícito al respecto).

En todo caso el tema va, y debió haber ido (publicado) hace muchísimo tiempo por estos lares, mejor tarde que nunca, aquí vamos.

Introducción

Para ir al meollo del problema, esto sucede porque ya se agotaron las direcciones IPv4 y aún no hemos establecido la estructura completa para soportar IPv6. Una sencilla búsqueda en cualquier navegador web y en cualquier idioma que elijamos nos arrojará luces sobre el tema: para ir a IPv6 hay que cambiar prácticamente todo el hardware (excepto nuestros ordenadores, que desde hace tiempo que ya están preparados). Ciertos países, como Japón, tienen ya una infraesctrutura sobre IPv6 pero a nosotros ahora es que falta tiempo (y trabajo). Si tuviéramos IPv6 podríamos asignar una dirección IP a cada uno de nuestros aparatos y sobrarían direcciones (si quieren saber más, lean nuestro artículo al respecto).

Es por ello que se inventó el NAT (Network Address Translate) que no es más que la traducción de direcciones de red: esencialmente nuestro proveedor de internet "compra" bloques de direcciones IPv4 a organizaciones como LACNIC (en nuestro caso para Latinoamérica y el Caribe) y nos asigna una de esas direcciones para nuestro hogar u oficina. Hasta acá todo bien si solo tuviéramos un solo dispositivo, como de hecho era en los años 90, pero hoy en día tenemos teléfonos móviles celulares, tabletas, televisores, cámaras de vigilancia ¡y algunos hasta neveras que comunican al internet! Todos estos aparatos ante el internet tienen la misma dirección IP que nos da nuestro ISP (Internet Service Provider), **lo que hacemos es instalar un enrutador, generalmente un aparato que también provee conexión inalámbrica, y ese aparato por medio de DHCP distribuiye las direcciones de una red privada.** Cuando se creó en el siglo pasado el sistema IPv4 se dejó tres clases de redes para direcciones IP privadas.

¿Cuántas direcciones IP necesitamos en nuestra organización? Pues que hay tres clases para uso privado:

- *Clase A*: 10.0.0.0 a 10.255.255.255 (8 bits red, 24 bits anfitrión).
- *Clase B*: 172.16.0.0 a 172.31.255.255 (16 bits red, 16 bits anfitrión)
- *Clase C*: 192.168.0.0 a 192.168.255.255 (24 bits red, 8 bits anfitrión)

Nosotros gustamos mucho de usar la tercera red, *Clase C,* así que esta máquina donde escribimos estas líneas suele tener la misma dirección IP: 192.168.1.47 y todos los programas que corren aquí pues ¿adivinen qué? pues el sistema operativo les entrega esa dirección. Incluso nuestros ordenadores portátiles, que tienen dos interfaces de red (ethernet y "wifi"), devolverán dos direcciones IP de la *Clase C* que dijimos.

Posible solución

Como podrán imaginar, consultar al sistema operativo cuál es nuestra dirección IP, pues siempre nos devolverá la que nos asigna nuestro enrutador en la red de área local, por ello es que desde hace años nosotros lo que hacemos es consultar con nuestro servidor web, donde está alojada nuestra página (máquina que tiene una dirección IP fija de manera pública, es decir, en la Internet como tal) con un sencillo guion que devuelve a cualquier navegador dicha información.

Decimos que es una posible solución porque imaginamos habrán sistemas operativos especializados y de seguro que algún programador en GNU/Linux tenga una mejor propuesta pero ***por ahora*** *ésta es nuestra opción seleccionada.*

En el lado del servidor web

Pues que nuestro servidor web ubicado en el internet, con una dirección IP fija pública, debemos colocar el siguiente fichero **siempre y cuando usemos Apache2, que es con la que lo hemos probado:**

"REMOTE_ADDR" forma parte de las variables de entorno que contiene Apache2 para identificar los visitantes o internautas y que podremos utilizar en nuestros trabajos de programación. Esta variable tiene su contraparte en lenguaje PHP y hemos de decir que es bastante más complejo, claro está. En PHP sería algo como esto:

A dicho archivo lo podremos llamar **ip.shtml** o **ip.php,** ya sea que usemos html o php (o le colocamos el nombre que vosotros queráis, incluso lo podéis colocar solito en un directorio con el nombre de fichero "index.shtml" o "index.php").

En el lado del cliente

Pues en el lado del cliente... ¡un simple navegador web y el enlace web al dominio que tengamos! Este artículo debería terminar aquí... **si nuestro oficio no fuera el de programar y programar.**

El chiste del asunto es hacer un guion que bien podamos ejecutarlo cuando queramos o hacerlo periódicamente de manera automática (y obtener así nuestra dirección IP pública). Esta es una prueba de concepto, en realidad nosotros tenemos algo más complejo que incluye una base de datos que registra todos los eventos derivados (y usamos lenguaje PHP en realidad), pero vamos al asunto que nos interesa, utilizar a Python para ello en el lado del cliente.

Paranoia generalizada

Nosotros no somos los únicos que utilizamos este "invento" como tal: incluso hay páginas web dedicadas exclusivamente al tema, por ejemplo las siguientes:

- http://www.soportweb.com/
- http://www.ifconfig.me/ip

La primera dirección resuelve bastante rápido, la segunda tarda un poco más en retribuir y eso lo tendremos en cuenta en nuestras líneas de código. Pensamos que es más lenta porque si visitamos la página principal www.ifconfig.me devuelve una cantidad brutal de valores *e imaginamos que la consulta de dirección IP igual procesa todas las variables pero devuelve un simple resulrado, cosa no muy recomendable de hacer.* Recordemos el "mantra" de las aplicaciones GNU/Linux: hacer una sola cosa y hacerla bien, de manera puntual, sin desperdiciar ciclos de máquina ni memoria RAM o disco duro sino el estrictamente necesario.

Para estas pruebas ambas direcciones las usaremos en ese orden, aunque de hecho nosotros usamos nuestros propios dominios, ya que atendemos varios clientes a los cuales les hemos hecho páginas web y les copiamos el mismo mecanismo a cada uno de ellos (¡somos ***copycat!*** ja ja ja?).

En este punto queremos resaltar que este "problema" no es de nosotros solamente y que solucionan de forma parecida a nosotros, la necesidad es la madre de las invenciones.

¿Qué hay allí afuera, en la jungla del internet?

Pues una rápida inspección por medio del buscador **DuckDuckGo** nos hizo doler la cabeza porque son soluciones muy elaboradas e implican instalar librerías adicionales pero nosotros nunca nos rendimos. Esencialmente lo que observamos es que nuestro guion debería ser compatible con Python 2 y Python 3 y que no necesite descargar librerías especializadas para nada. Al final del artículo dejaremos las fuentes consultadas por si quieren ver un ejemplo completamente paranóico (ea, que se nos ha pegado la palabrita) un ejemplo digno de estudiar, pero vamos por partes, aquí nosotros lo colocamos sencillito y si ustedes quieren estudiar (y saber

más) pues aprovechan al internet *no solo para jugar y conversar, sino en verdad algo útil que nos produzca beneficio de alguna manera.*

Creando el guion en Python

Librería a utilizar

Nosotros de hecho usamos Python 3 (luego veremos a Python 2) así que usaremos urllib.request, una librería cuyo trabajo es resolver consultas tipo HTTP, en su mayoría. Como siempre en Python todo es complejo, más sin embargo la forma como se escribe dicho lenguaje nos hace comprensible rápidamente, vamos a ver

```
import urllib.request as urllib
```

La anterior línea se traduce como la siguiente orden: "importar **urlib.request** como (colocarle este nombre) **urlib**". Colocar un alias nos evita escribir mucho código (¡vaya flojazos, nos dirán, si lo que hacen es trabajar sentaditos con aire acondicionado!) pero pronto veremos que esto tendrá un beneficio adicional.

Lo próximo que haremos es crear una consulta en memoria, es decir, crear un objeto según la librería que acabamos de importar:

```
consulta = urllib.build_opener()
```

Antes siquiera de hacer la conexión, primero debemos "disfrazar" nuestro querido **bot** (abreviatura de robot) para que se haga pasar ante nuestro servidor (o el servidor web de otro) como un simple e inofensivo navegador web Mozilla Firefox 57 en un ordenador con GNU/Linux de 64 bits, todo esto es para ser corteses y comedidos con nuestros anfitriones:

```
consulta.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x8
6_64; rv:57.0) Gecko/20100101 Firefox/57.0')]
```

Prestos a conectarnos

Ya casi tenemos todo listo, como ven no hemos programado mayor cosa y estamos a punto de resolver, solo queda un detalle en los parámetros de la siguiente instrucción:

```
url = consulta.open('http://www.soportweb.com/', timeout=17)
```

En los parámetros de la orden **.open()** debemos respetar el orden de los parámetros de la función (ver nuestro tutorial sobre funciones en Phyton) la cual es la siguiente:

```
urllib.request.urlopen(url, data=None, [timeout, ]*, cafile=None, capath=
None, cadefault=False, context=None)
```

Como vemos, luego de el enlace web le podremos pasar "data" y en tercer lugar está el "timeout" que es el tiempo que debe esperar hasta que reciba respuesta, nosotros como somos más pacientes le colocamos 17 segundos (recordar el segundo sitio web que tarda un poco en responder). *El punto aquí es que como no vamos a pasar "data", debemos escribir el nombre del parámetro "timeot=17" porque si no lo hacemos le estaremos pasando "17" como "data" y no como "timeout"* ¿si se entiende, verdad? En cuanto los demás parámetros son opcionales y nosotros acá no los y utilizaremos (arriba tienen el enlace si quieren conocer la función a fondo).

Recabando el resultado

Ahora guardaremos en la variable que llamaremos (¡qué originales somos!) "respuesta" el resultado de la consulta:

```
respuesta = url.read()
```

Solo quedaría imprimir por pantalla y cerrar el objeto:

```
url.close()  print('Servidor:'+respuesta)
```

Prueben ustedes y verán que devuelve en un formato un tanto extraño la dirección IP... si es que todo salió bien, porque ¿y si no tenemos internet, qué sucedería? ¿y si el dominio web está vencido en los DNS o si simplemente el servidor está caído? Primero repasemos lo sencillo que es trabajar con Python y luego "fortaleceremos" nuestro guion:

```
import urllib.request as urllib  consulta = urllib.build_opener()  consul
ta.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
```

```
rv:57.0) Gecko/20100101 Firefox/57.0')]  url = consulta.open('http://www.
soportweb.com/', timeout=17)  respuesta = url.read()  url.close()  print(
'Servidor:'+respuesta)
```

Problemas al conectar

Ya tenemos el guion básico, así puro y simple funciona para nosotros, pero tenemos que "pulirlo" más. en el caso de que suceda algún error, que no se cuelgue el programa y devuelva un resultado decente: para ello usaremos **try ~ except**, fíjense:

```
import urllib.request as urllib  consulta = urllib.build_opener()  consul
ta.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:57.0) Gecko/20100101 Firefox/
57.0')]  servidor
 = 'http://www.soportweb.com/'  try
:    url = consulta.open(servidor, timeout=17)    respuesta = url.read()
   url.close()    print('
Servidor:'+respuesta)  except:    print('
Falló la consulta ip a '+servidor)
```

A nuestro guion básico agregamos una variable llamada **servidor** donde está la dirección web del servidor al cual nos queremos conectar para que nos responda cual es nuestra dirección IP pública para poderlo reutilizar en la respuesta del mensaje "de error" (ciertos errores en computación se llaman excepciones en realidad), le colocamos en colores los cambios hechos.

Mejorando UTF-8

Si leyeron nuestro tutorial sobre HTML5, sabrán que somos fieles defensores de la letra eñe, los acentos y diéresis y en Python no es la excepción: vamos a sanear los caracteres y la configuración sobre cómo va a trabajar Python, coloreamos de violeta las nuevas instrucciones:

```
#!/usr/bin/env python  # -*- coding: utf-8 -*-
  import urllib.request as urllib  consulta = urllib.build_opener()  cons
ulta.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64
; rv:57.0) Gecko/20100101 Firefox/57.0')]  servidor = 'http://www.soportw
eb.com/'  try:    url = consulta.open(servidor, timeout=17)    respuesta
= url.read()    url.close()      try:
      respuesta = respuesta.de
code('UTF-8')
```

```
    except UnicodeDecodeError:
      respuesta = respues
ta.decode('ISO-8859-1')
    print('Servidor:'+respue
sta)  except:    print('Falló la consulta ip a '+servidor)
```

Por eso es que en esta sección nos referimos al lenguaje HTML5: algunos servidores responderán con la codificación de caracteres UTF-8 y otros con la norma ISO correspondiente. En el segundo **Try** lo que hacemos es "traducir" dichos caracteres para imprimirlos correctamente por pantalla.

En cuanto a las dos primeras líneas al comienzo del guion, dado el caso de que no le coloquemos la extensión a nuestro fichero .py y/o para que GNU/Linux sepa con cual programa va a correr el guion pues esto sirve como guia (ya que está como comentario, por tener un numeral al principio de la línea, en otros ambientes de sistemas operativo no surte ningún efecto alguno). La segunda línea le indica al propio Python que las instrucciones se las daremos, incluso, con caracteres UNICODE (aunque está marcado como comentario, no importa el sistema operativo que usemos, esto incumbe a Python directamente).

El uso de Try~Except y la etiqueta identificadora es para recordarnos que es en el manejo del juego de caracteres que podríamos tener problemas.

Allanando el camino para Phyton 2

Sí, ya sabemos que hemos advertido constantemente que todo lo debemos hacer en Python 3... pero tenemos clientes que aún tienen Python 2 ***y si no está echado a perder, pues no lo toque*** *(a estos "cochinitos" les llegará su respectivo día que tendrán que montar un servidor nuevo y en ese momento, pues, les colocaremos Python 3).* Aquí volvemos a repetir que utilizaremos nuestra manera minimalista y pragmática de resolver las cosas: importaremos la librería de sistema y veremos cuál es el primer caracter de la versión (2 ó 3) y la guardaremos en una variable (tenemos un tutorial sobre tratamiento de cadenas en Python). Esto lo usaremos para determinar qué librería vamos a cargar ¿recuerdan que usamos un alias para nombrar a la librería? **Pues acá el uso práctico:**

```
import sys  version = sys.version[0]  if version == '2':    import urllib
2 as urllib  else:    import urllib.request as urllib
```

Dicho esto lo procedemos a agregar a nuestro guion, que va creciendo poco a poco (en violeta lo nuevo agregado):

```
#!/usr/bin/env python  #
-*- coding: utf-8 -*-  import sys
version = sys.version[0]  if version == '2':
 import urllib2 as urllib  else:
 import urllib.request as urllib
  consulta = urllib.build_opener()  consulta.addheaders = [('User-agent',
 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox
/57.0')]  servidor = 'http://www.soportweb.com/'  try:   url = consulta.o
pen(servidor, timeout=17)   respuesta = url.read()   url.close()   try:
   respuesta = respuesta.decode('UTF-8')   except UnicodeDecodeError:
 respuesta = respuesta.decode('ISO-8859-1')   print('Servidor:'+respuesta
)  except:   print('Falló la consulta ip a '+servidor)
```

Pero se nos presenta ahora un pequeño inconveniente: Python 2 no sabe nada acerca de UTF-8 así que debemos depurar para que si y solo si estamos en Python 3 hagamos "limpieza" del juego de caracteres en el resultado (en verde colcoamos la solución cmpleta):

```
#!/usr/bin/env python  #
-*- coding: utf-8 -*-  import sys
version = sys.version[0]  if version == '2':
 import urllib2 as urllib  else:
 import urllib.request as urllib
  consulta = urllib.build_opener()  consulta.addheaders = [('User-agent',
 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox
/57.0')]  servidor = 'http://www.soportweb.com/'  try:   url = consulta.o
pen(se
rvidor, time
out=17)   respuesta = ur
l.read()   url.close()   if version == '3':
     try:       respuesta = respuesta.decode('UTF-8')     except UnicodeD
ecodeError:       respuesta = respuesta.decode('ISO-8859-1')   print('Ser
vidor:'+respuesta)  except:   print('Falló la consulta ip a '+servidor)
```

Trabajando con dos servidores web

Pues ya con esto sería más que suficiente sin embargo debemos tener en cuenta que nuestro guion debe hacer una sola cosa y debe hacerlo bien: si el primer servidor no responde a nuestra solicitud pues usaremos el segundo servidor. Para mantener el hilo didáctico nombraremos

nuestras variables con sufijos tales como "1" y "2" en el siguiente guion que crece en complejidad.
NOTA: la manera correcta de trabajar esto es hacerlo con matrices (en Python lo llaman listas).

[code lang="python"]
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import sys
version = sys.version[0]
if version == '2':
import urllib2 as urllib
else:
import urllib.request as urllib
url1 = None
url2 = None
servidor1 = 'http://www.soporteweb.com'
servidor2 = 'http://www.ifconfig.me/ip'
consulta1 = urllib.build_opener()
consulta1.addheaders = [('User-agent', 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0')]
consulta2=consulta1
try:
url1 = consulta1.open(servidor1, timeout=17)
respuesta1 = url1.read()
if version == '3':
try:
respuesta1 = respuesta1.decode('UTF-8')
except UnicodeDecodeError:
respuesta1 = respuesta1.decode('ISO-8859-1')
url1.close()
print('Servidor1:'+respuesta1)
except:
print('Falló la consulta ip a '+servidor1)
try:
url2 = consulta2.open(servidor2, timeout=17)
respuesta2 = url2.read()
if version == '3':
try:
respuesta2 = respuesta2.decode('UTF-8')
except UnicodeDecodeError:
respuesta2 = respuesta2.decode('ISO-8859-1')
url2.close()
print('Servidor2:'+respuesta2)

except:
print('Falló la consulta ip a '+servidor2)
[/code]

En el siguiente esquema de línea numeradas y mejor coloración de "gist" ofrecido por GitHub lo podremos explicar mejor:

Explicado línea por línea:

1. Le indica a GNU/Linux, como comentario, donde se encuentra Python y de maner tácita especifica que ese lenguaje de programación, aunque no le coloquemos la extensión .py al fichero.
2. Le especifica a Python, como comentario, que usaremos caracteres UTF-8 para nuestro lenguaje en castellano.
3. Importamos la librería que permite hacer llamadas por funciones al sistema operativo.
4. Consultamos la versión de Python que tengamos instalada pero solo nos importa el primer caracter que bien comienza por "2" o por "3" y lo almacenamos en la variable **version** (nombre de variable sin acento).
5. Línea vacía.
6. Evaluamos que si la variable **version** es dos, ejecutamos la línea 7, cualquier otro valor ejecutamos la línea 8.
7. Para Python 2 cargamos la librería **urllib2**.
8. Para cualquier otra versión de Python ejecutamos la línea 9.
9. Para cualquier otra versión de Python cargamos la librería **urllib.request**
10. Línea vacía.
11. Definimos la variable **url1** como vacía, sin contenido (a veces debemos "formatear" de esta manera las variables antes de usarlas).
12. Igual a la línea anterior pero para la variable **url2**.
13. Asignamos valor a **url1** (nuestro primer enlace web para obtener dirección ip pública).
14. Igual que la línea anterior pero para la variable **url2**.
15. Línea vacía.
16. Creamos la **consulta1** (¿recuerdan el "preformateo de variables"?)
17. A **consulta1** la vamos a hacer pasar como si fuera un navegador Firefox para que el servidor se sienta cómodo.
18. Creamos **consulta2** simplemente copiando a **consulta1** y nos ahorramos algo de trabajo.
19. Línea vacía.
20. Intentamos conectar.
21. Le damos la orden de conectarse con **servidor1** y si hubiera algún error se ejecutaría la línea 32.
22. Recogemos el resultado de la **consulta1** en una variable llamada **respuesta1**
23. Evaluamos si estamos usando Python 3, de ser así vamos a la línea 24

24. Intentamos limpiar **respuesta1**
25. Asumimos que recibimos en juego de caracteres UTF-8 con la función **decode**
26. Pero si **decode** no la puede procesar entonces asumimos que **respuesta1** está con otro juego de caracteres.
27. Lo que nos queda es probar si el juego de caracteres de **respuesta1** es 'ISO-8859-1'
28. Línea vacía.
29. Cerramos la conexión **url1**
30. IMPRIMIMOS POR PANTALLA NUESTRA DIRECCIÓN IP PÚBLICA (Respuesta N° 1).
31. Línea vacía.
32. Dado el caso que **consulta1** no se pudo realizar, vamos con nuestro segundo enlace web

Lo dejamos hasta la línea 32 porque esencialmente es repetir de nuevo el proceso pero cambiando los sufijos "1" por "2".

Para nuestros lectores y lectoras

Os dejamos a vuestra imaginación cómo podríamos resolver para *ene* direcciones web. Porr ejemplo, como los ramanes del libro de de Arthur C. Clarke "*Rendevouz with Rama":* lo haremos todo por triplicado, es decir, intentaremos tener tres servidores web que nos retribuyan nuestra dirección IP pública.

Este guion lo pueden "bajar" en este enlace, lo van a ver tal cual en vuestro navegador y por medio de los menús deplegables escogen la opción "Guardar como de vuestro navegador para que así tengan una copia en vuestro disco duro y puedan probar las siguientes variables:

- En vez de lograr conectar con un servidor web y mostrar resultado de una vez y salir, *conectar con todas y cada una de las direcciones y mostrar todos y cada uno de los resultados.*
- Guardar los resultados de cada uno de los sitios web para luego compararlos, **si son todos iguales imprimir resultado** de lo contrario mostrar un mensaje "discrepancia de resultado: ambigüedad".
- Escoger aleatoriamente dos, tres o cuatro servidores web de una lista amplia y realizar el mismo procedimiento anterior.

Conclusiones.

No hay tarea pequeña ni trabajo fácil en el mundo de la programación, esperamos haber aprendido algo que les sea útil en vuestro trabajo diario.

.

Fuentes consultadas

En idioma inglés

- «External IP Address Search Using Python Source Code»