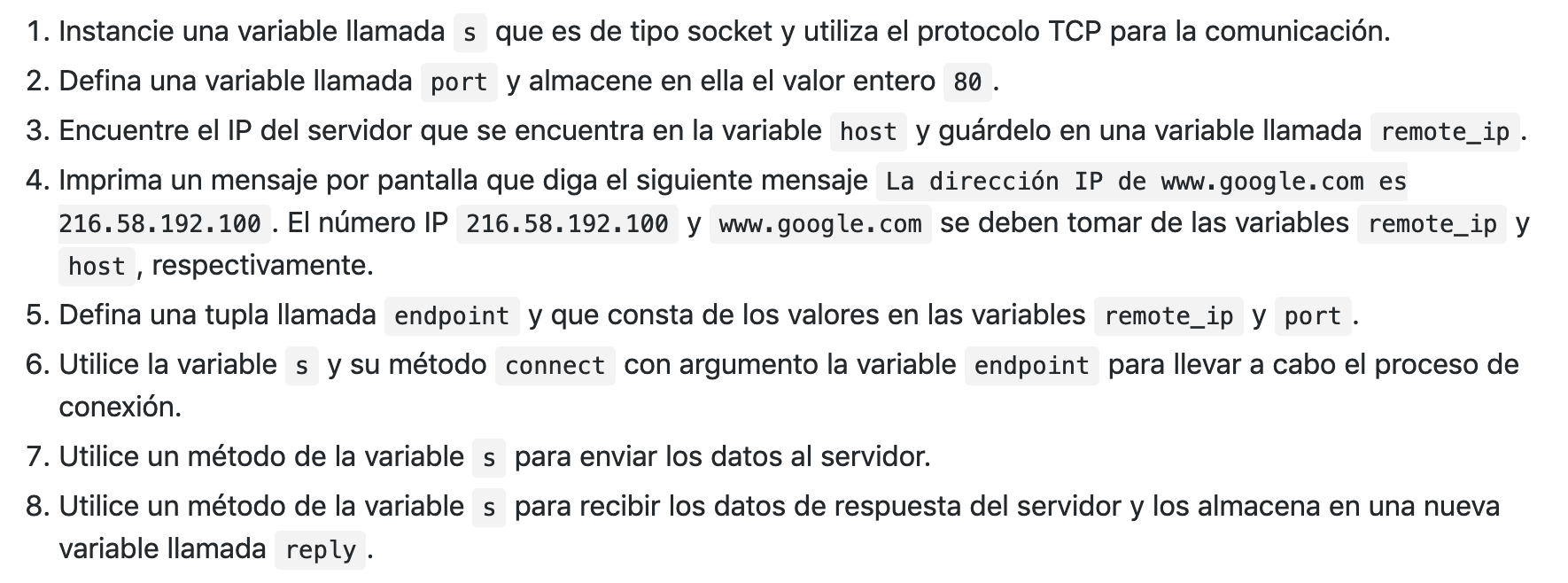
Despliegue de Servicios de Red - Usando Vagrant

John Sanabria - [john.sanabria@correounivalle.edu.co](mailto:john.sanabria@correounivalle.edu.co)

La semana anterior los estudiantes se familiarizaron con Vagrant a través del taller que se encuentra en [este documento](https://docs.google.com/document/d/1Me45PWM_9Muqo7w6VEuCb2bYiM1SZeYT6KyKbPNldyU/edit?usp=sharing).

El día de hoy se plantearán diversos escenarios de sistemas distribuidos, donde los estudiantes deberán hacer el despliegue de dichos escenarios y probar que son funcionales.

* Creación de una máquina virtual que despliegue el servidor Apache2 y se pueda acceder al servicio HTTP desde el *host* usando una aplicación en Python.
  + A la hora de crear la máquina debe especificar, con las instrucciones propias de Vagrant, la redirección del puerto 80.
  + El programa cliente se encuentra en [este enlace](https://github.com/josanabr/computernetworks/blob/master/basics/socket-http-client/socket-05.py). El código de este programa requiere ser completado[[1]](#footnote-0). Los ítems a ser completados están aquí.  
    
  + **IMPORTANTE**:
    - cambiar el valor de la variable host a “localhost”.
    - cambiar el valor de la variable port a 8080
* A continuación se darán los códigos de dos programas escritos en Python, un código es cliente y el otro es servidor. **Estos códigos se comunican a través del protocolo UDP**. Ejecute el programa cliente (que correrá en el *host*) y el servidor que correrá en la máquina virtual creada en el punto anterior.  
  Es importante considerar que tanto el cliente como el servidor a la hora de ser ejecutados se les debe pasar como argumentos “--port *nnn*” donde *nnn* es un número entero por encima de 1024 y por debajo de 65535.   
  Modificar el servidor de modo que escuche por todas las interfaces de la máquina virtual.  
  Ejecutar el servidor en la máquina virtual.   
  Modificar la configuración de la máquina virtual (usando el comando VBoxManage) para hacer la redirección del puerto **UDP** que usted designe para hacer la prueba.   
  A continuación los códigos en Python del cliente y el servidor.  
  **Cliente -** [**code**](https://github.com/josanabr/computernetworks/blob/master/basics/UDP/client-udp.py)

|  |
| --- |
| #!/usr/bin/env python  # Python Network Programming Cookbook, Second Edition -- Chapter - 1  # This program is optimized for Python 2.7.12 and Python 3.5.2.  # It may run on any other version with/without modifications.  import socket  import sys  import argparse  host = 'localhost'  data\_payload = 2048  def echo\_client(port):  """ A simple echo client """  # Create a UDP socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  server\_address = (host, port)  print ("Connecting to %s port %s" % server\_address)  message = 'This is the message. It will be repeated.'  try:  # Send data  message = "Test message. This will be echoed"  print ("Sending %s" % message)  sent = sock.sendto(message.encode('utf-8'), server\_address)  # Receive response  data, server = sock.recvfrom(data\_payload)  print ("received %s" % data)  finally:  print ("Closing connection to the server")  sock.close()  if \_\_name\_\_ == '\_\_main\_\_':  parser = argparse.ArgumentParser(description='Socket Server Example')  parser.add\_argument('--port', action="store", dest="port", type=int, required=True)  given\_args = parser.parse\_args()  port = given\_args.port  echo\_client(port) |

**Servidor -** [**code**](https://github.com/josanabr/computernetworks/blob/master/basics/UDP/server-udp.py)

|  |
| --- |
| #!/usr/bin/env python  # Python Network Programming Cookbook, Second Edition -- Chapter - 1  # This program is optimized for Python 2.7.12 and Python 3.5.2.  # It may run on any other version with/without modifications.  import socket  import sys  import argparse  host = 'localhost'  data\_payload = 2048  def echo\_server(port):  """ A simple echo server """  # Create a UDP socket  sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # Bind the socket to the port  server\_address = (host, port)  print ("Starting up echo server on %s port %s" % server\_address)  sock.bind(server\_address)  while True:  print ("Waiting to receive message from client")  data, address = sock.recvfrom(data\_payload)    print ("received %s bytes from %s" % (len(data), address))  print ("Data: %s" %data)    if data:  sent = sock.sendto(data, address)  print ("sent %s bytes back to %s" % (sent, address))  if \_\_name\_\_ == '\_\_main\_\_':  parser = argparse.ArgumentParser(description='Socket Server Example')  parser.add\_argument('--port', action="store", dest="port", type=int, required=True)  given\_args = parser.parse\_args()  port = given\_args.port  echo\_server(port) |

* NFS - *Network File System* - [slides](https://docs.google.com/presentation/d/1pR3lJkrnsByDimz2BW8_FyRxsPvPwpNboK3l2UWtzpY/edit?usp=sharing). El estudiante deberá crear las recetas que le permitan desplegar un servidor de NFS y dos clientes que acceden al servidor. El estudiante deberá entonces poder crear un archivo o directorio en un directorio compartido por el servidor de NFS y todos los clientes y el servidor evidenciar la creación de dicho archivo. En [este tutorial](https://vitux.com/install-nfs-server-and-client-on-ubuntu/) el estudiante podrá validar como se crea el servidor y el cliente. Hacer la instalación a mano e ir guardando en scripts los pasos que le funcionaron. Una vez los scripts estén listos usarlos en el Vagrantfile.

1. El programa está escrito para Python 2.7. [↑](#footnote-ref-0)