

Servidores

Eugenia Damonte, Ariel Fideleff y Martín Goñi

Índice

1. puTTY	1
1.1. Que es puTTY	1
1.2. Conexión inicial	2
1.3. Conexión usando llaves	4
2. Servidor web	8
3. Comandos usados	12

1. puTTY

1.1. Que es puTTY

puTTY es una serie de herramientas de código abierto que permite la transferencia de archivos mediante la red, así como también el acceso a una consola serial, entre otras cosas. Cuando se habla de puTTY de manera general en realidad se está hablando de una serie de programas o componentes, desarrollados y mantenidos por el programador británico Simon Tatham. Estos son:

- puTTY - Aplicación para utilizar Telnet¹, Rlogin² y un cliente SSH³, también permite la conexión a puertos seriales.
- PSCP - Cliente que permite realizar *command-line secure file copy*, es decir copiar archivos de manera segura desde un terminal. Puede además hacer transferencias SFTP.
- PSFTP - Cliente que permite utilizar SFTP⁴ para transferir archivos..
- puTTYtel - Un cliente específico para Telnet.
- Plink - Una interfaz de consola que permite acceder a el *back end* de puTTY. Normalmente usado para manejar túneles SSH⁵.
- Pageant - Un agente de autenticación para puTTY, PSCP y Plink.
- puTTYgen - Una aplicación que permite generar llaves de encriptación RSA, DSA, ECDSA y EdDSA.

¹Telnet o **Teletype Network** es un protocolo de red que permite acceder a la terminal de otra máquina de manera remota. Es además el nombre del programa que usa el cliente.

²Rlogin o **Remote Login** es una aplicación TCP/IP que inicia una sesión de terminal remota en el host especificado.

³Un cliente SSH es un programa que permite establecer conexiones seguras a servidores SSH.

⁴SFTP o **SSH File Transfer Protocol** es un protocolo seguro de transferencia de archivos, hoy en día ha reemplazado casi completamente a FTP, su predecesor.

⁵Un tunel SSH es un método para transportar información en la red de manera segura usando una conexión SSH encriptada.

En nuestro caso estamos interesados solamente en **puTTY** y **puTTYgen**, dado que son los necesarios para acceder de manera segura a una máquina remota usando SSH.

1.2. Conexión inicial

Nuestro objetivo con **puTTY** era usarlo para poder acceder de manera remota a una máquina⁶ utilizando el protocolo SSH.

Lo primero que hicimos fue crear un puerto por el cual pudiésemos acceder a la VM, para hacer esto fuimos a la configuración de la misma en **VirtualBox** y en el menú **Network** abrimos las opciones avanzadas, seleccionando **port forwarding**. En el clickeamos el botón con el signo mas para crear una nueva regla de redirección de puertos. Le pusimos **SSH** de nombre, dejando el protocolo en **TCP**. **Host IP** y **Guest IP** los dejamos vacíos para que se asignen automáticamente al momento de uso, dado que las direcciones IP no son estáticas. Finalmente completamos los campos correspondientes con los puertos. Para el del **Host**, es decir el de Windows, utilizamos el 5999 dado que es un puerto raro, haciendo poco probable que este ocupado. Para el puerto del **Guest**, es decir la VM, usamos el 22, el puerto estándar usado por SSH.

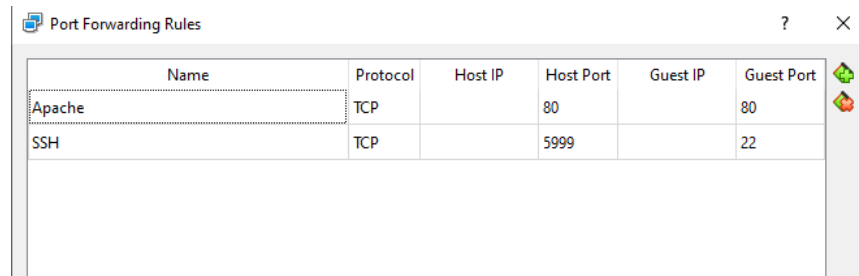


Figura 1.1: El menú de **port forwarding** de nuestra máquina virtual.

Una vez configurados los puertos nos dirigimos a la VM para verificar que los servicios SSH estuviesen funcionando de manera correcta. Para hacer esto usamos dos comandos, el primero `ps ax | grep 'ssh'` busca procesos con la palabra “ssh” en la lista de procesos activos. Al ejecutarlo encontró dos, indicando que los servicios estaban funcionando. Luego, para estar se-

⁶En nuestro caso utilizamos nuestra propia máquina virtual con **Debian 7**.

guros utilizamos otro comando `/etc/init.d/ssh status`, al ejecutarlo nos informó, de nuevo, que los servicios SSH estaban funcionando correctamente.

```
martin@DebianPC:/etc/init.d$ ps ax | grep "ssh"
2456 ?        Ss        0:00 /usr/sbin/sshd
2890 tty1     S+        0:00 grep ssh
martin@DebianPC:/etc/init.d$ /etc/init.d/ssh status
[ ok ] sshd is running.
```

Figura 1.2: Los comandos usados para verificar el funcionamiento de los servicios SSH.

Sabiendo que los servicios SSH estaban funcionando procedimos a realizar el primer intento de conectarnos de manera remota a la VM. Para hacer esto abrimos **puTTY**, y en el menú **Session** pusimos **localhost** en **Host Name** y **5999** en **Port**. El resto de las opciones las dejamos con sus valores predeterminados. Lo que significan estos valores es dentro de la computadora misma(**localhost**), conectarse al puerto **5999**, que es el que especificamos en la configuración de la VM, usando el protocolo SSH.

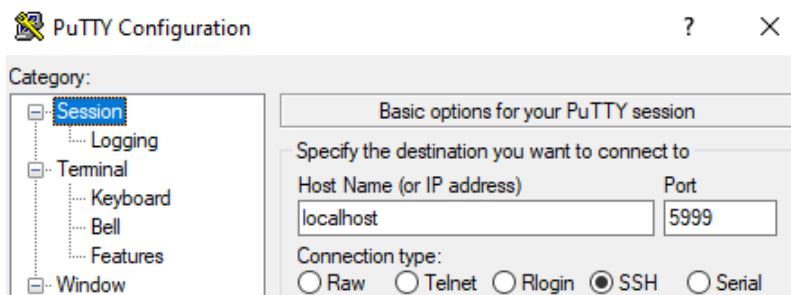


Figura 1.3: La configuración para la primera conexión a la VM.

Habiendo ingresado toda la información clickeamos el botón **Open** para iniciar la conexión con la VM. Al hacerlo apareció una consola pidiendo que ingresemos nuestro nombre de usuario, y luego contraseña. Al ingresarlos, se nos concedió acceso, pudiendo usar la consola de **puTTY** como si fuese la consola de la VM.

```
login as: martin
martin@localhost's password:
Linux DebianPC 3.2.0-4-686-pae #1 SMP Debian 3.2.78-1 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Thu Sep 10 18:20:29 2020 from 10.0.2.2
martin@DebianPC:~$
```

Figura 1.4: La primera conexión a la VM, hecha usando el protocolo SSH y puTTY.

1.3. Conexión usando llaves

Si bien este método funciona, sería muy peligroso usarlo para un servidor real. Esto es porque cualquiera podría conectarse a el mismo y obtener acceso al terminal de la máquina. Para solucionar este problema usamos una opción que tiene el protocolo SSH que permite validar conexiones mediante el uso de un par de llaves de encriptación, una pública y una privada. La pública se encuentra en la VM, y la privada en la computadora desde la cual se realiza la conexión, siendo usada por puTTY.

Lo primero que hay que hacer para usar la autenticación por llaves es generarlas, para esto usamos el programa **puTTYgen**. Una vez abierto bajo la sección **Actions** clickeamos el botón **Generate** sin cambiar ninguna de las opciones. Al terminar guardamos la llave privada como **id_rca.ppk** y la pública como **public.key**.

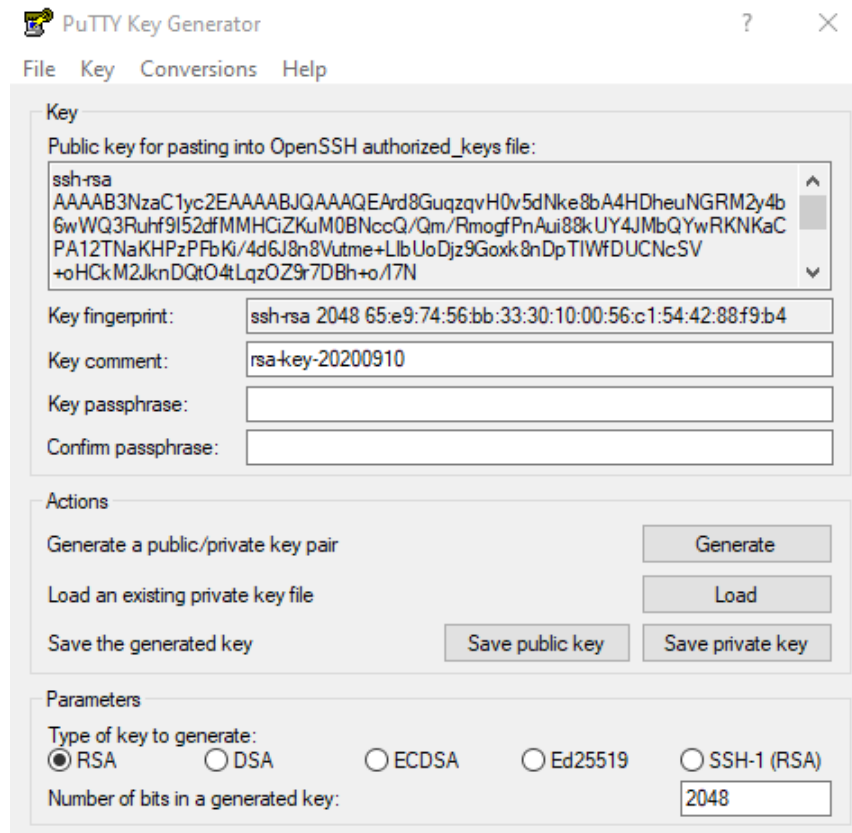


Figura 1.5: Configuración de puTTYgen usada para generar las llaves.

Una vez generadas las llaves teníamos que llevar la pública a la VM, así como también cambiar la configuración de SSH para que solo permitiese conexiones con llaves. Para hacer esto volvimos a conectarnos a la VM usando puTTY. Una vez allí lo primero que hicimos fue ir al directorio escondido `.ssh` en nuestro directorio propio. Allí creamos un archivo llamado `authorized.keys`, en el copiamos la llave pública generada por puTTYgen.

```

---- BEGIN SSH2 PUBLIC KEY ----
Comment: "rsa-key-20200831"
AAAAB3NzaClyc2EAAAABJQAAAQEAhkS24rsmFKN63BDW+BpZ2Vkc1z64xRfa2dPe
AdJVp6wJzo23oEizBxKs2/OIOE2/2uQ22sThblGi5jrRvZQRFwAtiRPygw1Ed0pz
cFortg+G9x98iZwYnA3l7Hh8i1lJrNyZamEs2NzchpAwXYlaXI92jY3ABsC5HGDG
NMK3rS63hsgArgpKjCZS5+IftXJLAxjhgSXSS0bbf5bJ3KkBtjghRmKCibm6b/zb
BrjVNlmslHLBbofNbtgPmXjLlMtXB6wWfmz4epySU9lRY4Qqbr/zuW+GrbpdRo8T
40xls8S06pqfro8om7jubGoCBO/t2rEGgniaNihUbs3VOQ74TQ==
---- END SSH2 PUBLIC KEY ----

```

Figura 1.6: Llave pública copiada a la VM usando la terminal de puTTY.

Con la llave pública en la VM era hora de configurar SSH para que solo sea posible la autenticación mediante llaves, no permitiendo usar contraseñas. Para esto abrimos el archivo `/etc/ssh/sshd_config` que es el archivo de configuración de SSH. En el cambiamos cuatro cosas:

- Cambiamos `PubkeyAuthentication` de no a yes.
- Descomentamos `AuthorizedKeyFile` `.ssh/authorized_keys`.
- Cambiamos `PasswordAuthentication` de yes a no.
- Cambiamos `ChallengeResponseAuthentication` de yes a no.

Para que estos cambios tomen efecto tuvimos que reiniciar los servicios SSH, usando el comando `sudo /etc/init.d/ssh restart`. Al ejecutarlo nos dijo que los servicios se habían reiniciado correctamente.

```

martin@DebianPC:~/.ssh$ sudo /etc/init.d/ssh restart
[sudo] password for martin:
[ ok ] Restarting OpenBSD Secure Shell server: sshd.

```

Figura 1.7: El comando usado para reiniciar los servicios SSH.

Para verificar si esto había funcionado primero era necesario configurar puTTY para utilizar la llave privada. Para hacerlo cerramos la terminal y volvimos a iniciar puTTY. En el menú `Session` volvimos a introducir la misma información que antes. Luego fuimos al menú `Data` bajo `Connection` donde en `Auto-login username` pusimos el nombre de usuario en la VM. Finalmente fuimos a el submenu `Auth`, bajo el menú `SSH`, que también esta en `Connection` y especificamos la ubicación de la llave privada. Para hacer más

rápido el conectarse a la VM con puTTY guardamos todas las configuraciones en una sesión que llamamos **Debian 7 2**.

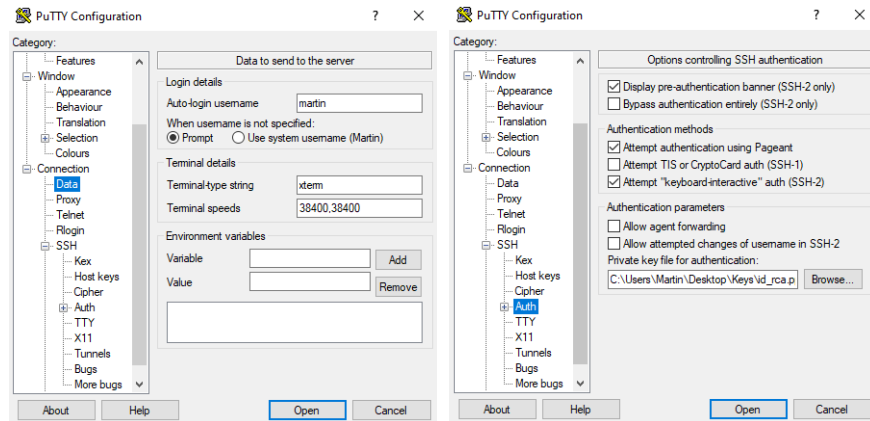


Figura 1.8: Configuración de los menús Data y Auth.

Cuando terminamos de ingresar toda la información apretamos el botón **Open** para conectarnos con la VM. Mientras se establecía la conexión apareció una ventana de puTTY diciendo que había ocurrido un error. Este decía *No supported authentication methods available*, que se traduce como “No hay metodos de autenticación soportados disponibles”. Esto nos parecio extraño ya que parecía que habíamos configurado todo correctamente y no había mucha información sobre cual era la causa del error.

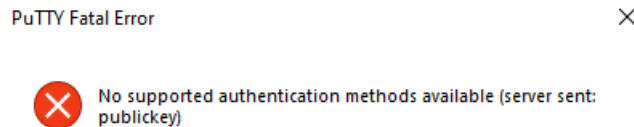


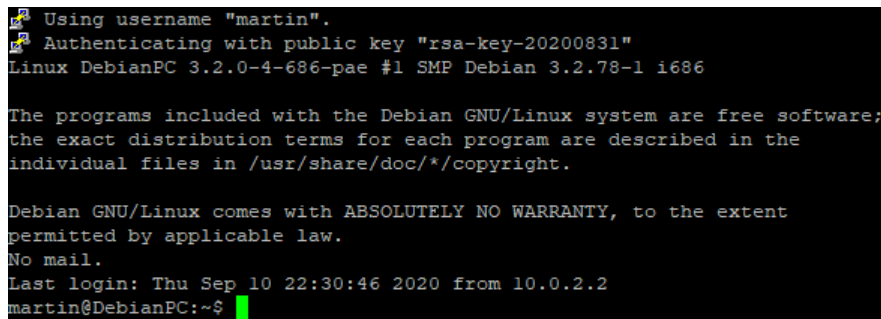
Figura 1.9: El error que nos dió puTTY al intentar conectarnos.

Luego de investigar un poco descubrimos que la causa del error era como habíamos ingresado la llave pública. El formato correcto es **ssh-rsa** (llave), estando todo en una misma línea.

```
ssh-rsa AAAAB3NzaC1yc2EAAAQEAHkS24rsmFKN63BDW+BpZ2Vkc1z64xRfa2dPeAdJvp6wJzo23oE1zBxks2/010E2/2uQ22sThb1G15JrRVZQRfWtIRPc
gwEdopzcFortg+G9x9b1ZwYnA317Hh8111JrNyZamES2NzchpAUXY1aX192JY3ABSc5HGdGNMk3rS63hsgArgpkJC2S5+IftXJLAXJhgSXSS0bbf5bJ3KkbtJghRmkC
Ibm6b/zbBrJWN1ms1HLBbofNbtgPmXJL1MTXB6uWfnc4epySU91RY4Qqbr/zuW+GrbpdRo8T40x1s8S06pgfr08om7Jub6oCB0/t2rEGgn1aNIhubs3V0Q74TQ==
```

Figura 1.10

Luego de solucionar ese problema intentamos conectarnos nuevamente a la VM, cosa que esta vez fue exitosa. Dado que habíamos podido conectarnos a nuestra VM desde **puTTY** utilizando las llaves de manera exitosa decidimos que ya estábamos listos para seguir con el próximo paso, el servidor con Apache.

A terminal window showing the process of connecting to a Debian virtual machine via SSH. The output includes the username 'martin', the public key 'rsa-key-20200831', and the system version 'Linux DebianPC 3.2.0-4-686-pae #1 SMP Debian 3.2.78-1 i686'. It also displays the Debian GNU/Linux license text and the last login time: 'Thu Sep 10 22:30:46 2020 from 10.0.2.2'. The prompt 'martin@DebianPC:~\$' is visible at the bottom.

```
🔓 Using username "martin".
🔓 Authenticating with public key "rsa-key-20200831"
Linux DebianPC 3.2.0-4-686-pae #1 SMP Debian 3.2.78-1 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Thu Sep 10 22:30:46 2020 from 10.0.2.2
martin@DebianPC:~$
```

Figura 1.11: Nos conectamos de manera exitosa a la VM.

2. Servidor web

En esta instancia, decidimos instalar y configurar un servidor web: **Apache HTTP Server**. Un servidor web es un software que permite que un usuario pueda ver el contenido de una página web. A grandes rasgos, lo que sucede al buscar una dirección en el navegador es que este busca en qué servidor o host se encuentra guardada la página y le “pide” el contenido. El servidor web (que está instalado en el host) es el encargado de entregárselo.

Lo primero que hicimos para lograr nuestro objetivo fue instalar Apache. Para eso, usamos el comando `sudo apt-get install apache2`. Verificamos que el servicio anduviera analizando la salida del comando `ps ax | grep 'apache'`.

```
euge@ciruela:~$ ps ax | grep "apache"
2006 ?      Ss      0:00 /usr/sbin/apache2 -k start
2008 ?      S       0:00 /usr/sbin/apache2 -k start
2012 ?      Sl      0:00 /usr/sbin/apache2 -k start
2013 ?      Sl      0:00 /usr/sbin/apache2 -k start
2565 pts/0   S+      0:00 grep apache
euge@ciruela:~$
```

Figura 2.1: `ps ax | grep ‘‘apache’’`

Antes de proceder, debimos asegurarnos de no tener un servidor web corriendo en Windows. Si hubiéramos tenido uno, habríamos tenido un problema: los protocolos HTTP usan, por defecto, el puerto TCP 80; si el puerto está en uso (si tuviéramos otro servidor web), tendríamos que redireccionar los puertos y eso llevaría más trabajo.

Una forma de asegurarnos de que el puerto 80 está libre es correr en el sistema `telnet localhost 80`. Telnet, como contamos previamente, es un programa que nos permite conectarnos a una computadora remota. La sintaxis de este comando es así: `telnet <servidor> <puerto>`. Es decir, al escribir `telnet localhost 80`, le estamos pidiendo a la computadora que se conecte a su puerto 80. Este paso debe realizarse desde la consola de Windows, no en la máquina virtual.

Algo a resaltar es que es probable que el servicio `telnet` esté desactivado en Windows. Para activarlo, simplemente buscamos entre las aplicaciones “Activar o desactivar características de Windows” y tildamos el casillero que dice “Telnet Client”.

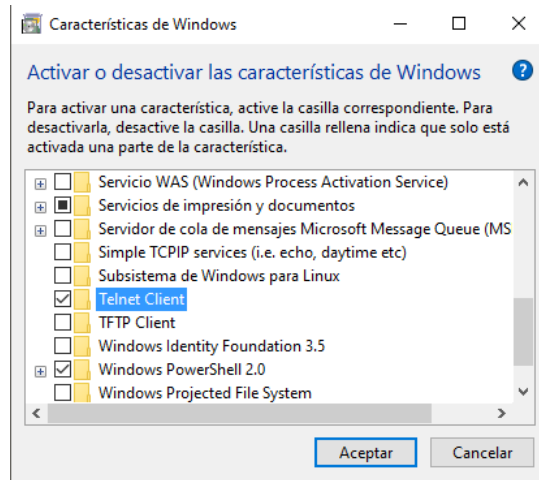


Figura 2.2: activamos Telnet Client

Una vez activada esta función, ejecutamos el comando nombrado anteriormente y, si recibimos un error como “No se puede abrir la conexión al host”, quiere decir que no tenemos un servidor web corriendo en este momento en Windows (más específicamente, que no estamos usando el puerto 80) y que podremos continuar sin problemas.

```
C:\Users\Admin>telnet localhost 80
Conectándose a localhost...No se puede abrir la conexión al host, en puerto 80: Error en la conexión
```

En nuestra máquina virtual corremos `telnet localhost 80` (ahora nos conectamos al puerto 80 de la Virtual Box) y escribimos `GET / HTTP/1.1`. Presionamos la tecla **enter** una vez y escribimos `Host: localhost`. La respuesta debería ser similar a la que se muestra en la imagen 2.3.

```
GET / HTTP/1.1
Host: localhost
```

```
euge@ciruela:~$ telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
```

Figura 2.3: pedimos al servidor que nos muestre la página “/” y le indicamos que nos comunicamos en HTTP/1.1

Pero ¿qué hace el comando “GET / HTTP/1.1”? GET le pide al servidor el contenido de la página / (una página que es generada por defecto). El último argumento que indicamos, HTTP/1.1, es el protocolo y la versión del mismo con los que nos comunicamos como cliente. Debemos recordar que el pedido que hace el navegador debe ser comprendido por el servidor web. Es decir, deben funcionar en el mismo protocolo: HTTP. Como primera parte de la respuesta obtuvimos “HTTP/1.1 200 OK”, que indica que el servidor usa, en este caso, el mismo protocolo que nosotros y que nos entiende. La otra parte (no visible en la figura anterior), es el código `html` de nuestra página.

Debemos recordar indicar el `Host` porque es un requisito de esta versión del protocolo y, de lo contrario, obtendríamos el error “Bad request”, el servidor no nos entendería. Por otro lado, si escribiéramos GET / HTTP/1.0, al ser una versión más vieja, no sería necesario este segundo paso y se entablaría la comunicación correctamente, como puede observarse en la figura 2.4.

```
euge@ciruela:~$ telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
```

Figura 2.4: pedimos al servidor que nos muestre la página “/” y le indicamos que nos comunicamos en HTTP/1.0

Sólo queda configurar los puertos de nuestra máquina virtual para que el servidor sea accesible desde otras máquinas. Para eso iremos a la aplicación VirtualBox y crearemos una nueva regla de reenvío de puertos, como hicimos al principio de este trabajo. En el nombre de la regla, indicamos “Apache”. La configuramos para que tenga protocolo TCP y que los puertos de anfitrión e invitado sean ambos el 80. Los IPs quedarán vacíos para que se acomoden

automáticamente.

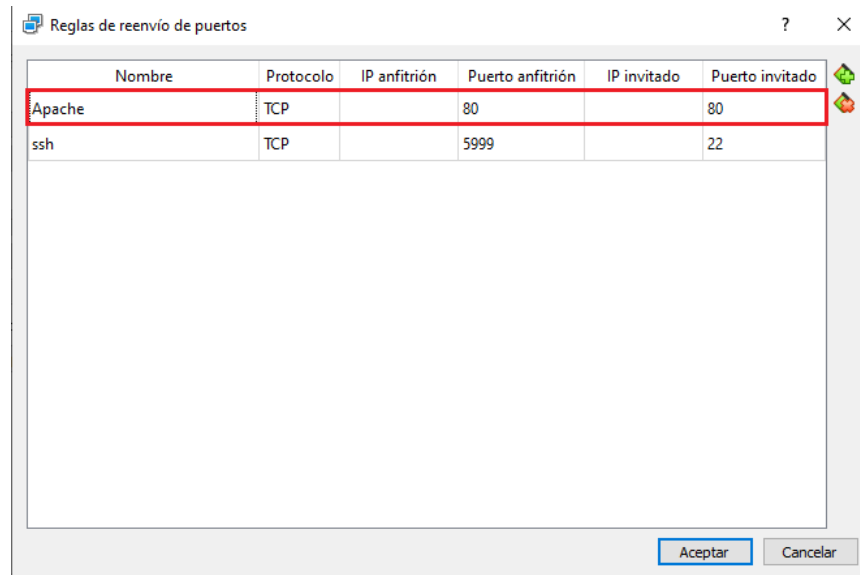


Figura 2.5: Caption

Por último, entrando a un navegador y usando la dirección “http://localhost” o simplemente “localhost” (tanto en Windows como en la máquina virtual), podremos verificar que todo esté andando bien (en la pantalla dirá inicialmente “It works!”, después podremos editarlo).

3. Comandos usados

A continuación se encuentran todos los comandos utilizados en este trabajo, correspondientes a las imágenes presentadas.

```
ps ax | grep 'ssh'  
/etc/init.d/ssh status
```

[1.2]

```

----- BEGIN SSH2 PUBLIC KEY -----
Comment: 'rsa-key-20200831'
AAAAB3NzaC1yc2EAAAABJQAAAQEAhkS24rsmFKN63BDW+BpZZVkc1
z64xRfa2dPeAdJVp6wJzo23oEizBxKs2/OIOE2/2uQ22sThb1Gi5j
rRvZQRFwAtiRPygwEd0pzcFortg+G9x98iZwYnA317Hh8il1JrNy
ZamEsZNzchpAwXYlaXI92jY3ABsC5HGDGNMK3rS63hsgArgpKjCZS
5+IftXJLAxjhGXSXS0bbf5bJ3KkBtjghRmKCibm6b/zbBrjVN1ms1
HLBbofNbtgPmXjLlMtXB6wWfmz4epySU9lRY4Qqbr/zuW+GrbpdRo
8T40xls8S06pqfro8om7jubGoCB0/t2rEGgniaNihUbs3VOQ74TQ==
----- END SSH2 PUBLIC KEY -----

```

[1.6]

```
sudo /etc/init.d/ssh restart
```

[1.7]

```

ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAAQEAhkS24rsmFKN63BDW+
BpZZVkc1z64xRfa2dPeAdJVp6wJzo23oEizBxKs2/OIOE2/2uQ22s
Thb1Gi5jrRvZQRFwAtiRPygwEd0pzcFortg+G9x98iZwYnA317Hh
8il1JrNyZamEsZNzchpAwXYlaXI92jY3ABsC5HGDGNMK3rS63hsgA
rgpKjCZS5+IftXJLAxjhGXSXS0bbf5bJ3KkBtjghRmKCibm6b/zbB
rjVN1ms1HLBbofNbtgPmXjLlMtXB6wWfmz4epySU9lRY4Qqbr/zuW
+GrbpdRo8T40xls8S06pqfro8om7jubGoCB0/t2rEGgniaNihUbs3
VOQ74TQ==

```

[1.10]

```
sudo apt-get install apache2
```

[2.1]

```
ps ax | grep 'apache'
```

[2.1]

```
telnet localhost 80
```

[??]

```
telnet localhost 80  
GET / HTTP/1.1  
Host: localhost
```

[2.3]