

Servicio SSH

Juan M. Alberola

1. Modelo cliente-servidor

La mayoría de las aplicaciones que funcionan en un entorno de red siguen el **modelo cliente-servidor**, el cuál tiene dos componentes que se comunican entre ellos: la parte cliente y la parte servidor (Figura 1).

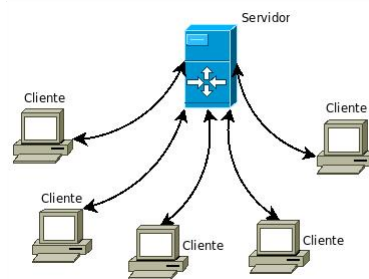


Figura 1: Modelo cliente-servidor

La parte **cliente** es un programa que se ejecuta en el host que solicita el servicio, el cuál es iniciado por un usuario o por un programa, y finaliza cuando termina el servicio. La parte **servidor** es un programa que se ejecuta en un host normalmente remoto, y que ofrece servicio a múltiples programas cliente. Este programa servidor ofrece los servicios de manera ininterrumpida y continuada. Por ejemplo, nosotros podemos tener un navegador web (Chrome, Firefox, Safari, etc.) que actúa como cliente HTTP, comunicándose con el programa servidor el cuál almacena un determinado sitio web. Este servidor, a parte de comunicarse con nuestro cliente, también se comunica con más clientes de otros usuarios que acceden a este sitio web.

Cada uno de los procesos que se está ejecutando en los hosts cliente y servidor tiene asignado un número de **puerto** que se utiliza para identificar las comunicaciones TCP/IP entre ambos hosts. Los puertos comprendidos entre el 0 y el 1023 se denominan puertos conocidos, y se asignan desde el sistema operativo. Los puertos comprendidos entre el 1024 y el 49151 son puertos accesibles por otros procesos y usuarios.

Unido al concepto de puerto, nos referimos como **socket** de un proceso al par formado por la dirección IP del host donde se está ejecutando dicho proceso y el puerto del proceso. Los sockets permiten crear conexiones virtuales entre un proceso en ejecución en un host cliente y un proceso en ejecución en un host servidor. El uso de puertos simplifica las comunicaciones TCP/IP y garantiza que se transmitan a las aplicaciones concretas.

En el tema anterior vimos cómo se enviaba una petición desde un host origen hasta un host destino a través de las IP públicas y privadas. Este proceso, realmente es un poco más complicado ya que además de las IP, también se registran los puertos origen y destino de los procesos que se comunican.

En la Figura 2 se ve un ejemplo en donde tenemos un host cliente con la IP 177.41.72.6 que realiza una petición a un host servidor que tiene la IP 41.199.222.3¹. Supongamos se trata de una petición HTTP a un sitio web alojado en el servidor. La petición HTTP se envía con el número de puerto conocido 80, que es el puerto reservado para los servidores HTTP (por tanto, el socket de este servicio es el 41.199.222.3:80). El puerto de la aplicación de navegador cliente tiene asignado también otro puerto de origen, en este caso, por ejemplo el 3022. Cuando la petición HTTP llega al host servidor, se entrega en el puerto 80, y por tanto, será tratada por el servidor HTTP. Cuando este servicio envía la respuesta, en última instancia será entregada por el router correspondiente al host cliente y al puerto 3022.

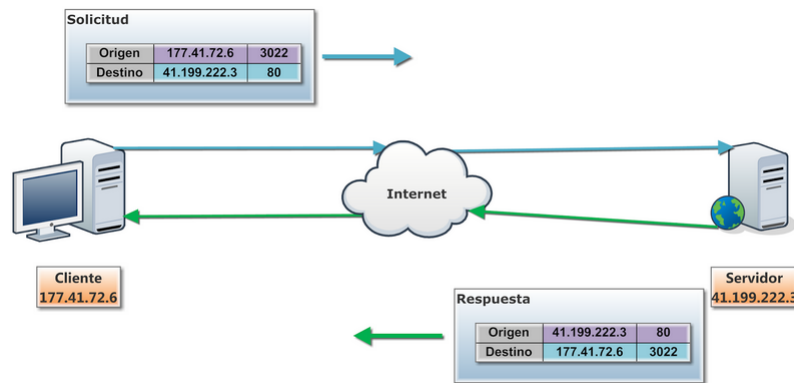


Figura 2: Estructura comunicación cliente-servidor

2. Servicios de la capa de aplicación

En el tema anterior ya vimos algunos de los servicios de la capa de aplicación del modelo TCP/IP, como son el servicio DNS y el servicio DHCP. Además de estos servicios, existen un conjunto de servicios que usamos frecuentemente y que conviene conocer.

El servicio **FTP** (File transfer protocol) permite a los clientes subir y bajar ficheros de un servidor que esté ejecutando este servicio. Este servicio no es dependiente de ningún sistema operativo, por lo que permite el intercambio entre distintas plataformas. La pila de protocolos TCP/IP incluye una utilidad FTP que permite el uso de FTP mediante el uso de comandos. Por defecto, el servidor FTP utiliza el puerto 20 para las conexiones y los clientes el 21.

El protocolo **HTTP** (Hypertext transfer protocolo) es el que gestiona la mayor parte del tráfico de Internet. Cuando un usuario solicita un recurso web, esta solicitud se realiza mediante HTTP. Cuando accedemos a una URL, el

¹En este ejemplo hemos obviado el proceso de traducción de IPs privadas a públicas a través de los routers correspondientes

servicio DNS resuelve la IP, y después, se envía una solicitud *get* al servidor web, el cuál devuelve un *send*, ambas operaciones usando HTTP. HTTP también hace uso del protocolo TCP y por defecto opera en el puerto 80.

El protocolo **HTTPS** (Hypertext transfer protocol secure) se utiliza para realizar transacciones de datos seguras vía web. Este protocolo utiliza una tecnología basada en certificados digitales para asegurar una autenticación mutua entre cliente y servidor. Además, HTTPS encripta todos los paquetes de datos, lo que garantiza su confidencialidad. HTTPS utiliza también TCP y por defecto opera en el puerto 443.

POP3 (Post office protocol v3) es un servicio de recepción de correo electrónico que proporciona al usuario el acceso a su carpeta de mensajes entrantes. POP3 se encarga de contactar con el servidor de correo y solicitar los mensajes recibidos en la cuenta del usuario. POP3 utiliza TCP y opera por defecto en el puerto 110.

IMAP (Internet message access protocol) permite a un cliente acceder a otras carpetas de correo electrónico además de las de mensajes recibidos (borrados, enviados, contactos, etc.). Este servicio también es exclusivo para el envío de mensajes. IMAP utiliza TCP y el puerto 143 por defecto.

SMTP (Simple mail transport protocol) se encarga de gestionar el envío de mensajes. Los mensajes se envían de un servidor SMTP a otro. Para ello, cada servidor utiliza el servicio DNS. SMTP utiliza TCP y por defecto opera en el puerto 25.

TELNET (Terminal emulation) permite conectar un cliente con un servidor remoto. Este protocolo proporciona comunicación bidireccional entre cliente y servidor. Por ejemplo, se puede usar para acceder desde clientes Windows a servidores Linux. El uso frecuente es para comprobar el correcto funcionamiento de servicios remotos como SMTP, POP3 o IMAP, y también para abrir puertos en el servidor remoto. Este protocolo también utiliza TCP y opera en el puerto 23 por defecto.

3. Servicio SSH

Secure Shell (SSH) es un protocolo de red para establecer comunicaciones seguras entre dos hosts siguiendo el modelo cliente-servidor, mediante el cual, un cliente SSH se conecta con un servidor SSH a través del puerto estándar 22 (Figura 3).

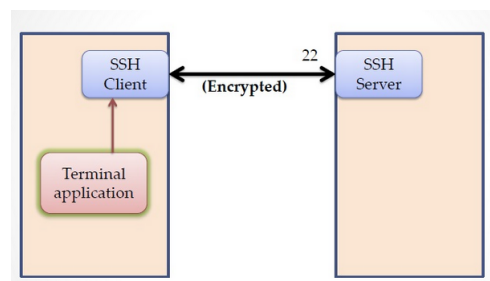


Figura 3: Comunicación SSH

SSH utiliza encriptado basado en claves públicas para autenticar el host cliente y el servidor. De esta manera, se ofrece confidencialidad e integridad de los datos en redes inseguras, como puede ser Internet.

La aplicación más común de este servicio es el acceso a shells remotos de sistemas Linux (acceder a otras máquinas a través de la red y trabajar con ellas como si estuviésemos en local), aunque también ofrece más funcionalidades como la transferencia de ficheros.

3.1. Parte servidor

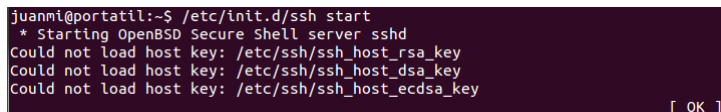
Para ofrecer este servicio nosotros utilizaremos la herramienta OpenSSH², la cuál podemos instalar mediante:

```
# sudo apt-get install openssh-server
```

El directorio donde se encuentra instalado SSH es `/etc/ssh`, en donde podemos encontrar los siguientes ficheros:

- `sshd_config`: archivo de configuración del servidor SSH
- `ssh_config`: archivo de configuración del cliente SSH
- `ssh_host*_key`: clave privada de la máquina (* puede ser rsa, dsa o ecdsa)
- `ssh_host*_key.pub`: clave pública de la máquina (* puede ser rsa, dsa o ecdsa).

Al instalar OpenSSH, en Ubuntu ya queda arrancado el servidor. En caso contrario, necesitamos arrancar el servidor SSH para que escuche las peticiones entrantes, ejecutando `service ssh start`. Si todo va bien, deberíamos ver que se ha arrancado correctamente (Figura 4).



```
juanmi@portatil:~$ /etc/init.d/ssh start
* Starting OpenBSD Secure Shell server sshd
Could not load host key: /etc/ssh/ssh_host_rsa_key
Could not load host key: /etc/ssh/ssh_host_dsa_key
Could not load host key: /etc/ssh/ssh_host_ecdsa_key
[ OK ]
```

Figura 4: Arranque servidor SSH

Otras opciones que podemos utilizar como parámetro son `stop` (para pararlo), `status` (para ver el estado) o `restart` (para reiniciarlo). Podemos ver que el servidor está escuchando por el puerto por defecto de SSH mediante `netstat` (Figura 5). Con el parámetro `-peanut` también lo puedes ver, y es fácil de recordar.

3.2. Parte cliente

Hay varias maneras de establecer una conexión SSH. La primera es utilizar un par de claves pública y privada generado automáticamente para encriptar la conexión de red, y después usar la autenticación mediante password para loguearse. Para ello, podemos conectarnos al host servidor donde tenemos el servidor corriendo mediante la aplicación cliente `ssh`:

²<http://www.openssh.com/>

```
juanmi@portatil:~$ netstat -ltu
Conexiones activas de Internet (solo servidores)
Proto Recib Envíad Dirección local Dirección remota Estado
tcp 0 0 *:http *:~* ESCUCHAR
tcp 0 0 localhost:5939 *:~* ESCUCHAR
tcp 0 0 *:ssh *:~* ESCUCHAR
tcp 0 0 localhost:ipp *:~* ESCUCHAR
tcp6 0 0 [::]:ssh [::]:~* ESCUCHAR
tcp6 0 0 ip6-localhost:ipp [::]:~* ESCUCHAR
udp 0 0 *:mdns *:~*
udp 0 0 *:34107 *:~*
udp6 0 0 [::]:mdns [::]:~*
udp6 0 0 [::]:57578 [::]:~*
juanmi@portatil:~$ _
```

Figura 5: Puertos en escucha del servidor

```
# ssh usuario@servidor
```

Donde **usuario** hace referencia al nombre de usuario con el que queremos conectarnos y **servidor** hace referencia al nombre del host al que queremos conectarnos. El nombre de usuario debe ser un usuario que exista en el host servidor, y por defecto nos situará en el directorio raíz de ese usuario. Como vemos en la Figura 6, nos hemos conectado desde el host cliente “sobremesa” al host servidor “portatil”.

```
juanmi@sobremesa:~$ ssh juanmi@portatil
The authenticity of host 'portatil (127.0.1.1)' can't be established.
ECDSA key fingerprint is bb:46:50:22:96:c9:bc:b7:b1:97:3f:61:a5:8f:0c:f8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'portatil' (ECDSA) to the list of known hosts.
juanmi@portatil's password:
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.8.0-29-generic i686)

 * Documentation:  https://help.ubuntu.com/

201 packages can be updated.
91 updates are security updates.

Last login: Sun Jan  5 16:17:56 2014 from 192.168.1.39
juanmi@portatil:~$
```

Figura 6: Conexión SSH

Otra opción es usar la dirección IP en vez del nombre del servidor (Figura 7).

```
juanmi@sobremesa:~$ ssh juanmi@192.168.1.33
juanmi@192.168.1.33's password:
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.8.0-29-generic i686)

 * Documentation:  https://help.ubuntu.com/

201 packages can be updated.
91 updates are security updates.

Last login: Sun Jan  5 16:20:15 2014 from localhost
juanmi@portatil:~$
```

Figura 7: Conexión SSH mediante IP

Una segunda forma de conectarnos es mediante un par de claves pública y privada generado manualmente para realizar la autenticación, de manera que se puede establecer la conexión sin necesidad de password. En este escenario, el usuario genera un par de claves, manteniendo la clave privada en el cliente

```

juanmi@sobremesa:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/juanmi/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/juanmi/.ssh/id_rsa.
Your public key has been saved in /home/juanmi/.ssh/id_rsa.pub.
The key fingerprint is:
ee:4d:b3:3b:ca:11:0d:dd:f8:53:94:9c:76:ed:8a:27 juanmi@sobremesa
The key's randomart image is:
+--[ RSA 2048 ]-----+
  .o.
  .o. = o
  .o...o
  o.
  S.o.
  .E.o
  o.o.o
  o+.o
  +.+o

```

Figura 8: Generación del par de claves

y depositando la clave pública en el servidor. En este tipo de autenticación, la misma clave no se transfiere a través de la red durante el proceso de autenticación. En este caso, mediante la llamada a `ssh-keygen` podemos generar el par de claves en el host cliente (Figura 8).

Después de esto, podemos ver que en el directorio `$HOME/.ssh` tenemos dos ficheros identificados como las claves, siendo por ejemplo, la clave pública la que se ve en la Figura 9.

```

juanmi@sobremesa:~$ cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCKkfj7q1neAWL6YBS09kXjpYr0LZ7jgs0RE1thggyi
E4fwyaAQG+SEaxuIK8sk5E+Xkz7ATVst6NjPjh9w71s3mL4v10UJX0LP91sM6hIW3yEeRa/6dTUG+Mfv
Ao93V/Gx/06jAknoNMx7177g/2f4E6ZDmyV/nK8UZH+qocsfvtn9+kLF/T7WBcmRVGmApFyOztkiUw8W
db7eHhvP9AYXZ2tmD69bfSvfX/KB30jJSA6reSgQ8Kg/Y7aNLnU1CZAS2deA4KkJ0dpztK8UTKnIrUno
YwRZ8x/2sHi8JNoklIHfEcTapAcmgioD83ea8JwhULJ/znZ1K1J6DHEgv94l juanmi@sobremesa
juanmi@sobremesa:~$

```

Figura 9: Clave pública

Esta clave pública deberemos copiarla en el servidor, en concreto en el fichero `$HOME/.ssh/authorized_keys`. Una forma segura de hacerlo es ejecutando la utilidad `ssh-copy-id (usuario@servidor)` en el cliente, que copiará la clave pública en el servidor (Figura 10).

```

juanmi@sobremesa:~$ ssh-copy-id 10.1.97.135
juanmi@10.1.97.135's password:
Now try logging into the machine, with "ssh '10.1.97.135'", and check in:

  ~/.ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.
juanmi@sobremesa:~$

```

Figura 10: Copia de la clave pública

Después de esto, en el servidor podemos ver que se ha copiado la clave pública del cliente (Figura 11) y por tanto, desde el host cliente ahora podemos hacer una conexión sin necesidad de contraseña (Figura 12).

```
juanmi@portatil:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCKkFj7q1neAWI6YBS09kXjpYr0LZ7jgs0RE1thgggi
E4fuyjaAQG+SExuIK8sk5E+Xkz7ATVst6NjPjh9w71s3mL4v10UJX0LP91sM6hIW3yEeRa/6dTuG+Mfu
ho93V/Gx/06jAknoMMX7177g/2f4E6ZDmgV/nK8UZH+qocsfutn9+kLF/T7WBcmRvGmApFyQztkiUw8W
db7eHhuP9AYXZ2tmD69bfSvfX/KB30jJSa6reSgQ8Kg/Y7aNLnU1CZAS2deA4kKjDdpztkBUTKnIrUno
YwRZ8x/2sHi8JNok1IHfEcTapAcnGioD83ea8JwhUIJ/znZ1K1J6DHEgv94l juanmi@sobremesa
juanmi@portatil:~$
```

Figura 11: Clave pública en el servidor

```
juanmi@sobremesa:~$ ssh 10.1.97.135
Welcome to Ubuntu 12.04.3 LTS (GNU/Linux 3.8.0-29-generic i686)

 * Documentation:  https://help.ubuntu.com/

443 packages can be updated.
239 updates are security updates.

New release '14.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Tue Mar 10 09:40:48 2015
juanmi@portatil:~$
```

Figura 12: Acceso al servidor sin contraseña

3.3. Copia de ficheros

Otra de las herramientas que nos ofrece SSH es la copia de ficheros entre un host local y uno remoto mediante la utilidad `scp` (Secure Copy). En esta utilidad, se realizan copias seguras y encriptadas usando SSH. La sintaxis es similar al comando `cp` pero cambiando el fichero destino para hacer referencia al host remoto:

```
# scp fichero_origen fichero_destino
```

En este caso, el fichero origen o el fichero destino puede hacer referencia a un host remoto. Por ejemplo, si queremos copiar el fichero en el directorio raíz del usuario `juanmi` del host `portatil`, la sintaxis sería la siguiente:

```
# scp fichero_origen juanmi@portatil:~/
```

Como vemos, después de indicar el usuario y host remoto, y a continuación con el carácter `:` indicamos dónde se copiará ese fichero. Con el carácter `~/` indicamos que queremos copiarlo en el directorio raíz del usuario remoto (también podríamos haber hecho `/home/juanmi`). De este modo, las copias se permiten también desde un host remoto a uno local e incluso entre dos hosts remotos.