

PRACTICA-2ISE.pdf



S_GRND



Ingeniería de Servidores



3º Grado en Ingeniería Informática



**Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Universidad de Granada**



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



LECCION 1

ssh viene de "Secure Shell" aunque en realidad ssh hace referencia a un protocolo, no a la implementación de un Shell.

Este servicio proporciona:

- Autenticación: vamos a obtener la posibilidad de autenticarnos
- Seguridad: mediante la encriptación de los datos que viajan
- Integridad de los datos que están viajando.

Antiguamente cuando un usuario quería conectarse a un servidor hacía uso de telnet. Los mensajes de telnet viajan en texto plano → cualquiera que tenga acceso a la red esta viendo el tráfico. Por esto, se decidió implementar un protocolo que supliera esa carencia de telnet, es decir, que encriptara los datos de manera que nadie fuese capaz de averiguar las contraseñas. Se desarrolló la primera implementación del protocolo ssh (ssh-1), luego evolucionó a la versión que tenemos ahora: ssh-2

Nosotros vamos a tener siempre un cliente que se conecta con un servidor → Arquitectura cliente/servidor.

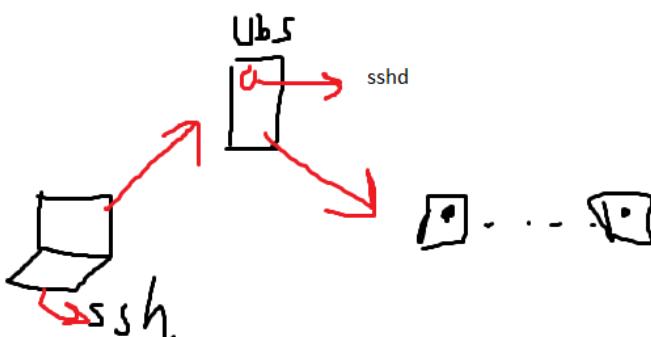
CUIDADO: El término ssh es ambiguo y puede hacer referencia tanto al programa cliente como al servidor.

Imaginemos que yo me quiero conectar a mi máquina destino (por ejemplo, Ubuntu server). Desde el ordenador invoco al cliente ssh. Tras una negociación, seguimiento del protocolo, autenticación, etc... consigo conectarme con Ubuntu server.

Imaginemos también que Ubuntu Server tiene un servicio que está escuchando constantemente. Este servicio lo denominaremos sshd (d de demon).

La parte de Ubuntu Server (el servidor) sería la parte del servicio; en cambio, la parte de mi ordenador sería la parte del cliente.

Resulta que Ubuntu Server no solo puede ser servidor de ssh sino que además puede ser cliente de otro servidor (*por ejemplo podría ser el front end de una máquina de procesamiento paralelo y en el servidor del que es cliente podríamos tener un cluster de computadores*). Ubuntu Server se conectaría también utilizando el protocolo ssh. En este caso, ssh invocaría al cliente y tendríamos distintos servidores en el clúster.



Vamos a tener un problemilla. Hay que tener mucho cuidado con la configuración de nuestro servicio. Hay que ser muy conscientes de cuando estemos modificando la configuración de nuestro servicio. En cada máquina donde tengamos instalado el servicio, vamos a tener dos archivos:

/etc/ssh/sshd_config → modificar el comportamiento del servidor

/etc/ssh/ssh_config → modificar los parámetros por defecto del cliente (aunque cada usuario puede tener su archivo local)

Hay que prestar mucha atención a la “d” (que ya sabemos que viene de demon).

Ambos tienen una estructura muy similar. Tienen parámetros comunes. CUIDADO: No confundirse. Hay que prestar atención.

Vamos a instalar el servicio en Ubuntu Server y vamos a realizar unas opciones básicas de configuración: cambio de puerto, deshabilitar el acceso de root, acceder sin contraseña... → cuestiones de la disciplina server hardening (configuración de algunos aspectos de seguridad).

Tenemos un servicio y un archivo que lo modifica → comportamiento muy similar.

Tendremos que modificar los archivos de texto con “vi” y “string editor (sed)”. Este último es otra herramienta que nos permite modificar los archivos de texto e incrustar esas órdenes en los scripts para alcanzar automatización.

Empezamos:

Comprobamos si tenemos configurada la red con “ip addr”

Vamos a utilizar el comando apt. Con este comando vamos a comunicarnos con los servidores de paquetes y buscar, instalar, actualizar... etc.

Vamos a buscar el ssh.

Ponemos “apt search ssh”. Salen muchos comandos. Vamos a filtrarlo mejor.

Ponemos “apt search ssh | grep server”. Podemos ver que hay un paquete que es openssh-server. Este es el que vamos a utilizar. Vamos a instalarlo poniendo “sudo apt install openssh-server”

```
[redacted]@ubuntu:~$ sudo apt install openssh-server
[sudo] password for [redacted]
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Lo interesante es saber qué versión del paquete estamos instalando. Puede dar un error. El error que da dice que no encuentra la IP. Actualizamos con “sudo apt update” para actualizar las direcciones IP de los repositorios

Ahora ya podemos instalarlo si nos había dado error (en mi caso no)

Estudiar sin publi es posible.



Compra Wuolah Coins y que nada
te distraiga durante el estudio



Ahora pueden surgir varias cuestiones. ¿Somos conscientes de si el instalador va a iniciar el servicio directamente? Podemos comprobarlo con “ps -Af | grep sshd”

```
ubuntu:~$ ps -Af | grep sshd
root      1535      1  0 12:37 ?
tartups
ubuntu:~$ 2196    1046  0 12:40 ttys000
00:00:00 grep --color=auto sshd
```

Para nuestra sorpresa, vemos cómo el instalador, una vez instalado el paquete, ha iniciado el servicio. Podemos hacer una prueba de invocar al cliente y conectarnos con nuestro servidor:

“ssh localhost”

```
ubuntu:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:ZQ+vBHqJ+Gz1o/QWvkh58h2thJr3DHAIhyjJZXLxUAo.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Aquí, como es la primera vez que establecemos conexión con el localhost, nos está mostrando la huella (fingerprint) de nuestro servidor. Esto tiene cierta ventaja: nos permite asegurarnos de que la máquina a la que nos queremos conectar es esa y no ha habido un usuario en medio. Ponemos “yes” y nos fijamos que ha añadido “localhost” a la lista de hosts conocidos. Ahora nos pide que nos identifiquemos y ya nos hemos autenticado satisfactoriamente. Hemos podido acceder a nuestra máquina a través del protocolo ssh. Con CNTRL + D salimos y vamos a ver qué ha ocurrido.

```
ubuntu:~$ logout
Connection to localhost closed.
```

Con “ls -la” vemos que en el home tenemos un nuevo directorio → “ssh”. Nos metemos y dentro vemos que se ha creado “known_hosts” y vemos el fingerprint para el localhost. De esta forma si reinstalamos el servicio o el sistema y el fingerprint cambiase, nos lanzaría una alerta y no nos dejaría conectarnos.

```
ubuntu:~$ ls -la
total 36
drwxr-xr-x 5 4096 Oct 22 12:42 .
drwxr-xr-x 4 4096 Sep 25 12:09 ..
-rw----- 38 Oct 22 12:27 .bash_history
-rw-r--r-- 220 Feb 25 2020 .bash_logout
-rw-r--r-- 771 Feb 25 2020 .bashrc
drwx----- 96 Sep 25 12:09 .cache
-rw-r--r-- 807 Feb 25 2020 .profile
drwx----- 2 4096 Oct 22 12:46 .ssh
-rw-r--r-- 1 0 Oct 3 09:37 .sudo_as_admin_successful
drwxr-xr-x 2 4096 Oct 3 09:38 .vim

ubuntu:~$ cd .ssh/
ubuntu:~/.ssh$ ls
known_hosts

ubuntu:~/.ssh$ cat known_hosts
|1|nIat+RS+UMMQ7Vu1ak07GKoJ558=|k/RXP5SwQS5n1BbxyPpYJ/lhCAQ= ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTI
tbm1zdHAyNTYAAAAA1bm1zdHAyNTYAAABBC7VgDMeVP0MHBo18tdP1LRWPssyH0SG2Vxf1dLFMCKBMyAEQUAtWrSLFN+Cyg0je1L
1k0Gi6ZYEKYQCnnd/6po=
ubuntu:~/.ssh$
```

Dentro de las primeras opciones con las que tenemos que tener cuidado cuando estamos ejecutando el servicio ssh es: Tenemos que ser conscientes de que el usuario que está en todas las máquinas es el usuario root. Vamos a editar nuestro archivo de configuración para deshabilitar el acceso de root.

Para acceder remotamente a la administración de nuestro servidor, tendremos que utilizar un usuario como pasarela y ese usuario posteriormente cambiará de usuario y pasará a tener privilegios. (Podemos discutir si es recomendable que un usuario con privilegios de administrador tenga acceso a través de ssh → No es una buena práctica porque si la contraseña de ese usuario no es buena, existen riesgos).

Editamos el archivo de configuración del servicio poniendo: "sudo vi /etc/ssh/sshd_config"
Buscamos la opción de PermitRootLogin con "/Permit"

```
#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```

Por defecto está deshabilitada. No se permite el acceso de root si no es sin contraseña. De todas formas lo hacemos directamente y ponemos "no"

```
#LoginGraceTime 2m
#PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```

Hacemos "systemctl restart sshd"

Hay que tener cuidado porque Ubuntu Server tanto si especificamos ssh o sshd lo va a hacer correctamente. Para ser homogéneos y no confundirnos con centOS (que ahí es sshd), ponemos sshd para evitar ambigüedades

```
[redacted]@ubuntu:~/.ssh$ systemctl restart sshd
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Authentication is required to restart 'ssh.service'.
Authenticating as: [redacted] ([redacted])
Password:
==== AUTHENTICATION COMPLETE ===
```

Ahora tenemos que comprobar, nos vamos a una terminal e intentamos acceder con el usuario root a nuestra máquina.

Ponemos "ssh -l root 192.168.56.105"

```
C:\Users\redacted>ssh -l root 192.168.56.105
The authenticity of host '192.168.56.105 (192.168.56.105)' can't be established.
ECDSA key fingerprint is SHA256:ZQ+vBHqJ+Gzlo/QWvkh58h2thJr3DHaihyjJZXLxUAo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.105' (ECDSA) to the list of known hosts.
root@192.168.56.105's password:
Permission denied, please try again.
root@192.168.56.105's password: [redacted]
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Como podemos ver, no nos deja. Permission denied.

Otra forma de especificar el usuario puede ser nombre_de_usuario@ip
(<root@192.168.56.195>)

Una cosa interesante que debemos conocer:

En ssh parámetro -v sirve para hacer la conexión verbose. Es decir, que sea explícita: que nos cuente paso a paso lo que va haciendo. Gracias a esto podremos depurar problemas de conexión y ver los errores que nos está devolviendo el servidor.

Ahora vamos a irnos a centOS. Vamos a ir viendo las diferencias que hay.

Primero ponemos "ps -Af | grep sshd" para ver si el servicio está. Vemos que por defecto en la instalación, se inicia el servicio sshd. Otra forma de comprobar el servicio es con el comando "systemctl status sshd" (vemos que si ponemos "ssh" no nos va. Aquí si hay diferencia de nomenclatura)

```
[REDACTED]@localhost ~]$ systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-10-24 16:26:46 EDT; 5min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
     Main PID: 892 (sshd)
        Tasks: 1 (limit: 5019)
       Memory: 2.4M
      CGroup: /system.slice/sshd.service
              └─892 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,...
```

oct 24 16:26:46 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...
oct 24 16:26:46 localhost.localdomain sshd[892]: Server listening on 0.0.0.0 port 22.
oct 24 16:26:46 localhost.localdomain sshd[892]: Server listening on :: port 22.
oct 24 16:26:46 localhost.localdomain systemd[1]: Started OpenSSH server daemon.

Vamos a hacer la prueba de conexión desde aquí: Ponemos "ssh localhost". Como vemos, está operativo y con CNTRL D salimos.

```
[REDACTED]@localhost ~]$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:ck9/9c5F+SUmFRqUcRMDYfeQ/sBZ/4sU3ep00jUWho.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
[REDACTED]@localhost's password:
Activate the web console with: systemctl enable --now cockpit.socket
```

Ahora vamos a hacer la prueba de conexión desde fuera. "ssh 192.168.56.10 -l [REDACTED]"

```
C:\Users\[REDACTED]\>ssh 192.168.56.110 -l [REDACTED]
[REDACTED]@192.168.56.110's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sun Oct 24 16:33:53 2021 from ::1
[REDACTED]@localhost ~]$
```

Ya estamos dentro de centOS.

En Ubuntu hemos desactivado root y prohibía que se accediera mediante password. Vamos a ver como lo tiene configurado centOS. Vamos a intentar acceder como root:

```
C:\Users\███████>ssh 192.168.56.110 -l root
root@192.168.56.110's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sat Oct  2 11:22:01 2021
[root@localhost ~]#
```

Para nuestra sorpresa, hemos podido acceder como root a centOS. (en ubuntu no podíamos)

Vamos a editar el archivo de configuración con “sudo vi /etc/ssh/sshd_config”. Buscamos el permitrootlogin, quitamos “yes” y ponemos “no”.

```
# Authentication:
#
#LoginGraceTime 2m
PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
"/etc/ssh/sshd_config" 148L, 4424C written
```

Desde el terminal, vamos a comprobar que efectivamente no puedo acceder como root:

```
C:\Users\███████>ssh 192.168.56.110 -l root
root@192.168.56.110's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sun Oct 24 16:42:54 2021 from 192.168.56.1
[root@localhost ~]#
```

ME DEJA! ¿Por qué? Porque he cometido la imprudencia de no reiniciar el servicio. Es un error muy común. Recordemos que en UNIX algunos servicios pueden hacer una monitorización del archivo de configuración y en caso de que haya un cambio, se recarga. Pero esto no es lo normal. El comportamiento por defecto es que si hacemos una modificación en la configuración, tenemos que recargar el servicio. Por tanto, hacemos “systemctl restart sshd”. Ya lo tenemos reiniciado.

```
[root@localhost ~]# systemctl restart sshd
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====
Se requiere autenticación para reiniciar 'sshd.service'.
Authenticating as: ██████████
Password:
==== AUTHENTICATION COMPLETE ====
[root@localhost ~]#
```

Vamos a volver a comprobar si podemos entrar como root desde la terminal.

```
C:\Users\██████████>ssh 192.168.56.110 -l root  
root@192.168.56.110's password:  
Permission denied, please try again.  
root@192.168.56.110's password:
```

No nos deja.

Con esto, ya tenemos ssh configurado tanto en ubuntu como en centOS y ya hemos inhabilitado el acceso del usuario root (es lo minimo que hay que hacer cuando utilizamos el servicio ssh).

Otro elemento a tener en cuenta es que el puerto por defecto de ssh es el 22. Entonces, una buena práctica puede ser cambiar el número de puerto para evitar que sea trivial intentar acceder a él. Para ello tenemos el archivo de configuración y la directiva port.

Vamos a probar a modificar el archivo de configuración. En este caso, en lugar de utilizar vi, vamos a utilizar el string editor (sed). Este comando nos va a permitir buscar una cadena y sustituirla. Para ello vamos a hacer la sustitución de la cadena “Port 22” por por ejemplo “Port 22022” y lo hacemos en modo inline (-i) dentro del archivo de configuración /etc/ssh/sshd_config

Ponemos “sed s/Port 22/’Port 22022’/-i /etc/ssh/sshd_config”

```
██████████@localhost ~]$/ sudo sed s/'Port 22'/'Port 22022'/-i /etc/ssh/sshd_config  
[sudo] password for ██████████
```

Reiniciamos el servicio con “systemctl restart sshd”

```
██████████@localhost ~]$/ systemctl restart sshd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====  
Se requiere autenticación para reiniciar 'sshd.service'.  
Authenticating as: ██████████  
Password:  
==== AUTHENTICATION COMPLETE ====  
██████████@localhost ~]$_
```

Ahora intentamos conectarnos desde la terminal y comprobamos

```
C:\Users\██████████>ssh ██████████@192.168.56.110  
██████████@192.168.56.110's password:  
Activate the web console with: systemctl enable --now cockpit.socket  
  
Last login: Sun Oct 24 16:37:05 2021 from 192.168.56.1
```

¿Qué pasa? Que sigue accediendo en el puerto 22. Le tenemos que especificar el puerto distinto mediante la opción “-p 22022”

Ponemos "ssh nombre_de_usuario@192.168.56.110 -p 22022"

No funciona! Pero si hemos reiniciado el servicio y hemos hecho el cambio de puerto, ¿qué ha podido ocurrir?

Vamos a inspeccionar visualmente el archivo de configuración.

```
#  
#Port 22022 → Está comentado!  
#AddressFamily any
```

Lo descomentamos.

Podíamos haber hecho la edición con el comando:

```
"sed s/'#Port 22'/'Port 22022' /i /etc/ssh/sshd_config"
```

Reiniciamos el servicio ahora

```
[redacted]@localhost ~]$ systemctl restart sshd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====  
Se requiere autenticación para reiniciar 'sshd.service'.  
Authenticating as: [redacted]  
Password:  
==== AUTHENTICATION COMPLETE ====  
Job for sshd.service failed because the control process exited with error code.  
See "systemctl status sshd.service" and "journalctl -xe" for details.
```

Tenemos un problema

Para monitorizar estos problemas tenemos el comando "journalctl"

Vamos a comprobar antes el estado con "systemctl status sshd"

Vemos que tenemos un fallo al iniciar el servicio. Puede que nos hayamos equivocado en el archivo de configuración o cualquier otra cosa.

```
sshd.service: Main process exited, code=exited, status=255/n/a  
sshd.service: Failed with result 'exit-code'.  
Failed to start OpenSSH server daemon.  
~  
~  
~  
~
```

Pero no tenemos más información por lo que vamos a invocar a "journalctl"

Ponemos "journalctl -xe"

```
-- Unit sshd.service has begun starting up.  
oct 24 17:05:08 localhost.localdomain sshd[1831]: error: Bind to port 22022 on 0.0.0.0 failed: Permission denied  
oct 24 17:05:08 localhost.localdomain sshd[1831]: error: Bind to port 22022 on :: failed: Permission denied  
oct 24 17:05:08 localhost.localdomain sshd[1831]: fatal: Cannot bind any address.  
oct 24 17:05:08 localhost.localdomain systemd[1]: sshd.service: Main process exited, code=exited, status=255/n/a  
oct 24 17:05:08 localhost.localdomain systemd[1]: sshd.service: Failed with result 'exit-code'.  
oct 24 17:05:08 localhost.localdomain systemd[1]: Failed to start OpenSSH server daemon.  
-- Subject: Unit sshd.service has failed  
-- Defined-By: systemd
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Aquí ya vemos que nos está indicando que hay un error al intentar enlazar/asignar a ssh el puerto 22022. Esto es porque alguien no nos está dando permiso. ¿Qué elemento de seguridad nos permite controlar qué procesos tocan qué archivos, puertos, etc...? SELinux

La solución al problema la tenemos en el mismo archivo de configuración.
"vi /etc/ssh/sshd_config"

```
# If you want to change the port on a SELinux system, you have to tell
# SELinux about this change.
# semanage port -a -t ssh_port_t -p tcp #PORTNUMBER
#
```

Esto nos dice que si queremos cambiar el puerto en un sistema que está ejecutando SELinux, le tenemos que informar a SELinux sobre este cambio.

Esto lo hacemos con el comando semanage. Le tenemos que decir que vamos a operar sobre un puerto en concreto (port), que lo vamos a añadir (-a), qué tipo de puerto (-t ssh_port_t), qué protocolo (-p tcp). Por último lugar le especificamos el número de puerto.

Por tanto ponemos: "semanage port -a -t ssh_port_t -p tcp 22022"

Es posible que no tengamos instalado en el sistema el "semanage". Para buscar quién nos proporciona el comando semanage utilizaremos "provides" del comando "dnf".

Ponemos "dnf provides semanage" (si no va, ponemos antes "dhclient")

```
[8]~$ dnf provides semanage
[8]~$ dnf provides semanage
CentOS-8 - Base                               4.5 MB/s | 7.5 MB   00:01
CentOS-8 - Extras                            51 kB/s | 10 kB   00:00
policycoreutils-python-utils-2.9-14.el8.noarch : SELinux policy core python utilities
Repositorio : BaseOS
Resultado de:
Archivo    : /usr/sbin/semanage
```

Aquí nos dice que el binario semanage lo proporciona el paquete policycoreutils.

En este caso podemos utilizar yum o dnf. (yum redirige a dnf)

Ponemos "semanage port -l" para listar los tipos de puertos. Hacemos un grep para los tipos de puertos relacionados con ssh con "semanage port -l | grep ssh

```
[8]~$ sudo semanage port -l | grep ssh
ssh_port_t          tcp      22
[8]~$
```



Ahora si, ponemos: "semanage port -a -t ssh_port_t -p tcp 22022"

```
[redacted]localhost ~]$ sudo semanage port -a -t ssh_port_t -p tcp 22022
```

Comprobamos listando los puertos

```
[redacted]localhost ~]$ sudo semanage port -l | grep ssh  
ssh_port_t          tcp      22022, 22
```

Y vemos que ya tenemos nuestro puerto 22022 añadido como puerto valido para el servicio ssh. Reiniciamos el servicio de nuevo con systemctl restart ssh

```
[redacted]localhost ~]$ systemctl restart sshd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====  
Se requiere autenticación para reiniciar 'sshd.service'.  
Authenticating as: [redacted]  
Password:  
==== AUTHENTICATION COMPLETE ====
```

Ya no tenemos error.

Intentamos conectarnos desde la terminal.

Seguimos sin tener posibilidad

Vamos a hacer una prueba desde la propia máquina: ssh localhost -p 22022

```
[redacted]localhost ~]$ ssh localhost -p 22022  
[redacted]localhost's password:  
Activate the web console with: systemctl enable --now cockpit.socket  
Last login: Sun Oct 24 16:56:10 2021 from 192.168.56.1  
[redacted]localhost ~]$ _
```

Me deja.

¿Qué está pasando? Qué tecnología de software nos permite controlar/cerrar algunos puertos? El firewall.

CentOS tiene el comando firewall-cmd y Ubuntu Server tiene el uncomplicated firewall.

Vamos a proceder a configurar el cortafuegos para que no permita acceder el puerto 22022. Para ello, nos vamos a centOS y tenemos el comando “firewall-cmd”. Es un frontend para iptables. Nos va a permitir tener una interfaz más amigable que definir esas cadenas. Tiene una funcionalidad muy completa.

En nuestro caso, lo que queremos hacer es añadir un puerto. Lo añadiremos de forma permanente para que se aplique la configuración y abra el puerto la proxima vez que recarguemos el comando. Pero en el momento en el que le damos a intro no abre el puerto por defecto. Si quitamos la opción, este comando abre el puerto pero cuando recarguemos el

firewall o reiniciemos la máquina el puerto estará cerrado. Hay que utilizar una combinación de ambos.

Hacemos “sudo firewall-cmd --add-port 22022/tcp --permanent” y luego “sudo firewall-cmd --add-port 22022/tcp”

```
[redacted]@localhost ~]$ sudo firewall-cmd --add-port 22022/tcp --permanent  
[sudo] password for redacted:  
success  
[redacted]@localhost ~]$ sudo firewall-cmd --add-port 22022/tcp  
success
```

(o directamente, en lugar del segundo comando podemos recargar el firewall con “sudo firewall-cmd –reload”).

```
[redacted]@localhost ~]$ sudo firewall-cmd --reload  
success
```

Ahora ya, vamos a la terminal y probamos a conectarnos con el puerto 22022

```
C:\Users\redacted>ssh redacted@192.168.56.110 -p 22022  
redacted@192.168.56.110's password:  
Activate the web console with: systemctl enable --now cockpit.socket  
  
Last login: Sun Oct 24 17:25:20 2021 from ::1  
[redacted]@localhost ~]$
```

Vemos como ahora si nos deja comunicarnos con centOS a través del puerto 22022.

Vamos a pasar a hacer la misma configuración en Ubuntu. Utilizaremos el uncomplicated firewall → “ufw”. Este nos permite manipular los puertos.

Primero vamos a modificar el valor:

“sudo vi /etc/ssh/ssh_config”

Cambiamos “#Port 22” por “Port 22022”

```
Port 22022  
#AddressFamily any  
#ListenAddress 0.0.0.0  
#ListenAddress ::
```

Reiniciamos el servicio con “systemctl restart sshd”.

```
[redacted]@ubuntu:~$ systemctl restart sshd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===  
Authentication is required to restart 'ssh.service'.  
Authenticating as: redacted  
Password:  
==== AUTHENTICATION COMPLETE ===
```

Comprobamos desde la terminal que efectivamente tenemos acceso

```
C:\Users\redacted>ssh redacted@192.168.56.105 -p 22022  
redacted@192.168.56.105's password:  
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-86-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com  
 * Management: https://landscape.canonical.com  
 * Support: https://ubuntu.com/advantage  
  
System information as of Sun Oct 24 21:40:32 UTC 2021
```

Si nos fijamos en este caso me ha dejado sin ningún problema. No hemos tenido que indicarle a ubuntu que añada el puerto. Esto es porque uncomplicated firewall, aunque está instalado por defecto, también está desactivado por defecto.

Vamos a activarlo y vamos a añadirle ese puerto 22022.

Vamos a ver el estado con “sudo ufw status”

Está inactivo.

```
NOTIFICATION COMPLETE ---  
[REDACTED]@ubuntu:~$ sudo ufw status  
Status: inactive
```

Vamos a pasar a activarlo.

Ponemos: “sudo ufw enable”

```
[REDACTED]@ubuntu:~$ sudo ufw enable  
Firewall is active and enabled on system startup
```

Ya tenemos el cortafuegos activo. Vamos a comprobarlo. Al activar el cortafuegos vemos que nos ha cerrado la conexión en la terminal de nuestro host. (no podemos escribir en ella). Cerramos y abrimos otra.

Nos intentamos conectar a la máquina con el puerto 22022 y vemos que no nos deja. Esto es porque el firewall está activado.

Tenemos que indicarle que permita el tráfico con el puerto 22022.

Ponemos “sudo ufw allow 22022”

```
[REDACTED]@ubuntu:~$ sudo ufw allow 22022  
Rule added  
Rule added (v6)
```

Intentamos conectarnos y vemos que ahora si nos deja:

```
C:\Users\[REDACTED]>ssh [REDACTED]@192.168.56.105 -p 22022  
[REDACTED]@192.168.56.105's password:
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



SSH CLASE LABORATORIO – SESIÓN 1

Vale para centOS y Ubuntu Server

1. Instalamos ssh
2. Deshabilitar el acceso de root

Normalmente cuando hacemos un acceso a ssh hacemos paco@ip

Vamos a deshabilitar el acceso directo con root root@ip → acceso privilegiado directo deshabilitado.

Accederemos como root accediendo como usuario normal y luego haciendo sudo.

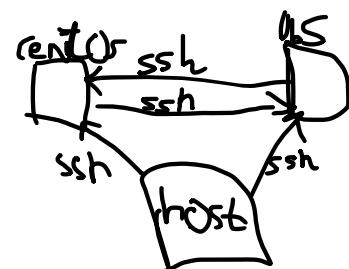
3. Cambiamos el puerto de ssh (por defecto es 22).
Esto es para practicar, no tiene un objetivo de seguridad.

Tendremos que tocar probablemente el firewall

4. Acceso por ssh sin contraseña

Deberíamos ser capaces de conectarnos desde nuestra máquina ubuntu a nuestra máquina centOS y al revés. Y desde nuestro host. CON y SIN contraseña

WINDOWS: permite almacenar los hosts, etc → herramientas util → putty



DIFERENCIAS ENTRE UBUNTU SERVER Y CENTOS

Ubuntu Server	CentOS
En la instalación pregunta para instalar ssh	Está por defecto (no nos ha preguntado nada)
Ssh y sshd es válido respecto a la nomenclatura	Solo es válido sshd
Permit root login no con password	Permit root OK
Uncomplicated firewall	Firewall cmd
Firewall OFF por defecto	Firewall ON por defecto

LECCION 2

COMANDO dd

Comando que nos permite hacer una copia de seguridad de nuestro dispositivo → de cualquier volumen físico

Realiza una copia COMPLETA.

CPIO

Permite copiar una estructura de directorio y reconstruirlo tal cual en el destino → conserva incluso los atributos de los archivos (propietarios, fecha, etc...)

RSYNC

Herramienta muy útil para sincronizar el contenido de dos directorios → Mantener dos copias de un directorio.

Compara la estructura del directorio origen con el destino y copia las diferencias
Funciona sobre ssh

Capítulo 2 →



Learningor

GIT

Para aprender:

- Web: Learngitbranching
- Libro de git: Capítulo 10. gitinternals
- Atlassian.bitbucket.git

Creamos un directorio nuevo

- mkdir prueba

Nos metemos

- cd prueba

Ejecutamos git

- git init .

El repositorio de git es el archivo .git

Gitignore es un fichero oculto al que le decimos extensiones de archivos que no queremos subir a git accidentalmente → archivos binarios sobre todo. Si subimos archivos binarios, cada vez que hagamos una modificación, va a hacer una copia y tendremos una por cada modificación.

Otra característica de este tipo de sistemas git (control de versiones): podemos tener varias versiones.

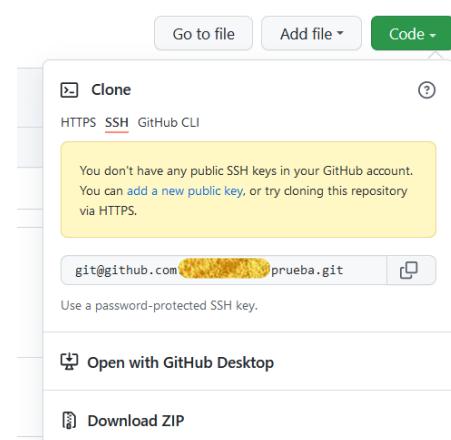
Si hacemos un cambio (por ejemplo creo otro archivo) y lo guardo, git va a crear una snapshot. Siempre puedo volver a la versión anterior.



Línea de desarrollo

Con git, cuando hago un clone además de bajarme la ultima foto del proyecto, me bajo TODA la historia del proyecto, por tanto una vez hecho el clone, aunque no tenga acceso a internet, puedo acceder a lo que hizo el usuario hace tiempo.

Para hacerlo con ssh:



"git clone git@github.com:████████/prueba.git"

Nos sale lo de siempre: la consulta sobre la llave pública

Si sale permission denied:

Los permisos de git, son readonly, públicos. Al hacerlos ssh, tenemos que registrar en git nuestra llave pública.

Genero la llave pública.

Voy a dar de alta la llave pública que he creado. Voy a añadirla a mi red.

Perfil > Settings > SSH and GPG keys > New SSH Key → pego la llave

Repite el git clone y ya me la ha copiado

Creo un nuevo archivo con vi dentro

Le pongo lo que sea

Si ahora hago git status, me dice que mi rama está actualizada con el origen (tengo la rama master) → me avisa que tengo un nuevo archivo (y su nombre)

Tengo mi directorio de trabajo. Tenemos que indicarle a git que lo añada al final. → git add de lo que quiero subir y git commit para confirmar (commit registra el cambio).

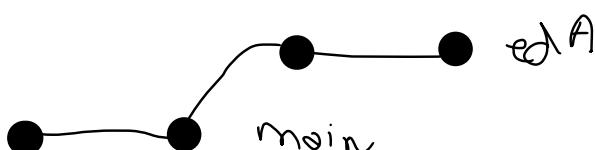
Luego hago push para subirlos a github.

GIT ADD → LOS LLEVAMOS AL STAGE

GIT COMMIT → LOS LLEVAMOS AL REPOSITORIO LOCAL

GIT PUSH → PARA LLEVARLOS AL REPOSITORIO DE ORIGEN

Estoy aquí:



git checkout main → me posiciono en el main





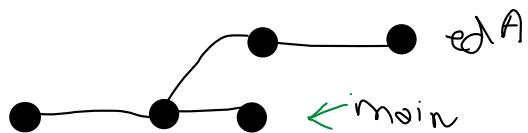
BURN
ENERGY DRINK

GANA UN FIAT 500 CON BURN ENERGY

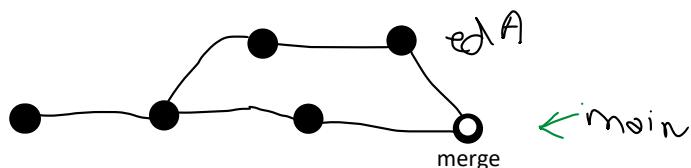
1 LATA = 1 PARTICIPACIÓN



git pull → me he bajado el cambio



Git merge edA



Como ha habido 2 versiones que han tocado lo mismo, me da un conflicto en el archivo.

Hay que arreglarlo manualmente.

Si hacemos status, vemos que hay ramas que hemos intentado mezclar pero no se han mezclado todavía porque hay conflictos. Podemos arreglarlo y solucionar el conflicto o echar para atrás.

Luego hacemos merge add, merge commit y merge push

*Pos

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

Estudiar sin publi es posible. Compra Coins.

LECCION 3

En la lección 1 teníamos nuestra máquina centOS y Ubuntu Server. En ambas máquinas configuramos el servicio sshd, que estaba escuchando en el puerto 22022. Además, habíamos deshabilitado el acceso del usuario root.

Hoy, lo primero que haremos será permitir el acceso sin contraseña.

Esto se realiza por:

- Contraseñas muy largas (incomodo/tedioso)
- Poca seguridad (cámaras de seguridad, huellas, etc...)
- Máquina Front-End que va a ejecutar varios trabajos en un backend en 100 máquinas y se va a conectar a través de ssh → ¿vamos a estar tecleando para cada máquina la contraseña? No es lo mejor



Este sistema de acceso sin contraseña es la solución para todos estos ejemplos (y muchos más).

¿Cómo vamos a conseguir el acceso sin contraseña?

La comunicación de ssh entre dos máquinas va cifrada mediante un cifrado simétrico. Nosotros utilizaremos un cifrado asimétrico teniendo una llave privada que tendrá nuestra máquina cliente. Utilizaremos una llave pública que copiaremos en nuestro servidor.

Vamos a generar ese par de archivos (llave pública y privada) → Utilizaremos el comando ssh-keygen



Nos vamos a la máquina centOS. Para el usuario en la máquina centOS vamos a generar las llaves con "ssh-keygen" y nos empieza a cuestionar:

- ¿Queremos guardar la llave en la ubicación por defecto? (dentro de la carpeta del usuario, dentro de .ssh) → por defecto usa el tipo rsa
 - Podríamos cambiar la ubicación por cuestiones de seguridad pero no es necesario

- ¿Queremos añadirle una contraseña a la llave privada? Si perdemos el pen drive donde está la llave (por ejemplo) podría ser peligroso → se añadiría una capa extra de seguridad. En este caso no lo añadimos.

```
[redacted]localhost ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/redacted/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/redacted/.ssh/id_rsa.
Your public key has been saved in /home/redacted/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:xxg1+fdcICu4rIedjUsvu21xfxUZTLHdXx3W0UdHrsg [redacted]localhost.localdomain
The key's randomart image is:
+---[RSA 3072]---+
|          o001
|          .+@|
|          o . . .XI
|          B . + =o1
|          S = E . +l
|          o o o.o...l
|          oo+. oo...
|          ..=.oo. o1
|          ....o=o .l
+---[SHA256]---
```

Nos aparece hasheada la llave pública. Tenemos ya los dos archivos. Lo comprobamos:

Ponemos “ls -la .ssh/”

```
[redacted]localhost ~]$ ls -la .ssh/
total 12
drwx-----. 2 [redacted] 57 nov  3 05:46 .
drwx-----. 3 [redacted] 95 oct 24 16:33 ..
-rw-----. 1 [redacted] 2622 nov  3 05:46 id_rsa
-rw-r--r--. 1 [redacted] 584 nov  3 05:46 id_rsa.pub
-rw-r--r--. 1 [redacted] 171 oct 24 16:33 known_hosts
```

Llave privada: id_rsa → Permisos solamente de escritura y lectura para el usuario que las posee

Llave pública → id_rsa.pub → La vamos a ir distribuyendo en las máquinas donde nos queramos autenticar → Sí podemos tener permisos de lectura para los demás usuarios

Idea intuitiva: Voy a cifrar un paquete con mi llave privada en mi cliente y voy a enviarselo a la máquina remota → será capaz de descifrarlo porque tendrá la llave pública. A partir de ahí se negocia la autorización.

El método de acceso sin contraseña es autorización. La parte de cifrado se hace con llave simétrica por cuestiones de eficiencia.

Vamos ahora a copiar nuestra llave pública al servidor



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



En este caso vamos a copiar nuestra llave publica en nuestra máquina ubuntu con "ssh-copy-id" (desde centOS)

Podriamos copiar la llave publica con cp u otro método pero con este script (ssh-copy-id) podemos especificar de forma transparente y cómoda dónde va a ir la llave.

192.168.56.105 es Ubuntu | 102.158.56.110 es CentOS

-p para especificar el puerto

Ponemos en la máquina centOS: "ssh-copy-id 192.168.56.105 -p 22022"

Como es la primera vez que establecemos la conexión nos dice lo de siempre respecto a la llave pública, le decimos que si.

```
[REDACTED]localhost ~]$ ssh-copy-id 192.168.56.105 -p 22022
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/[REDACTED].ssh/id_rsa.pub"
The authenticity of host '[192.168.56.105]:22022 ([192.168.56.105]:22022)' can't be established.
ECDSA key fingerprint is SHA256:[REDACTED]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install all the new keys
```

Ahora viene una cosa importante: Cómo nos vamos a autenticar en la máquina de destino. En este caso utilizaremos el password pero cuando tengamos el acceso sin contraseña podremos desactivar el acceso por password.

```
[REDACTED]192.168.56.105's password:
Number of key(s) added: 1
Now try logging into the machine, with: "ssh -p '22022' '192.168.56.105'"
and check to make sure that only the key(s) you wanted were added.
```

Nos informa que ha copiado la llave pública en la máquina de destino y nos recomienda/invita a que iniciemos sesión para comprobarlo.

Vamos a comprobarlo:

Ponemos en centOS: "ssh 192.168.56.105 -p 22022" y vemos que hemos iniciado sesión en UbuntuServer con el usuario _____ y no hemos tenido que teclear contraseña.

```
[REDACTED]localhost ~]$ ssh 192.168.56.105 -p 22022
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-86-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Wed Nov  3 10:26:49 UTC 2021

 System load:  0.0          Processes:           124
 Usage of /home: 0.2% of 468MB   Users logged in:      1
 Memory usage: 23%
 Swap usage:  0%
 IPv4 address for enp0s3: 10.0.2.15
 IPv4 address for enp0s8: 192.168.56.105

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

 https://ubuntu.com/blog/microk8s-memory-optimisation

 166 updates can be installed immediately.
 22 of these updates are security updates.
 To see these additional updates run: apt list --upgradable

 Last login: Wed Nov  3 10:11:41 2021
 [REDACTED]Ubuntu:~$
```

Ahora vamos a desactivar el acceso por contraseña. Para ello, vamos a irnos al servidor

de Ubuntu y vamos a editar el archivo de configuración

Ponemos en UbServer: "sudo vi /etc/ssh/sshd_config"

Le vamos a decir que no deje acceder por contraseña. Cambiamos en "#Password Authentication" el "yes" por "no" y quitamos la almohadilla

```
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no
```

Guardamos, salimos y reiniciamos el servicio con

"systemctl restart sshd"

```
[redacted]@ubuntu:~$ systemctl restart sshd
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Authentication is required to restart 'ssh.service'.
Authenticating as [redacted]
Password:
==== AUTHENTICATION COMPLETE ===
```

Ahora vamos a comprobar que efectivamente desde centOS tenemos la posibilidad de seguir accediendo

```
[redacted]@localhost ~]$ ssh 192.168.56.105 -p 22022
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-86-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Wed Nov  3 10:32:55 UTC 2021

 System load:  0.0              Processes:           124
 Usage of /home: 0.2% of 468MB  Users logged in:      1
 Memory usage: 23%              IPv4 address for enp0s3: 10.0.2.15
 Swap usage:   0%              IPv4 address for enp0s8: 192.168.56.105

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

166 updates can be installed immediately.
22 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Wed Nov  3 10:26:49 2021 from 192.168.56.110
```

Efectivamente, podemos acceder exitosamente desde la máquina centOS habiendo deshabilitado el acceso por password.

Si yo ahora desde una consola de mi anfitrion intento acceder a Ubuntu

```
C:\Users\██████ ssh 192.168.56.105 -p 22022  
████████: Permission denied (publickey).
```

Podriamos poner -v al comando y vemos que dice no tiene forma de acceder porque no tiene mas alternativas para autenticarse.

```
debug1: No more authentication methods to try.  
████████: Permission denied (publickey).
```

¿Qué ocurre si yo quisiera copiar mi llave pública del anfitrion a ubuntu server? No podria. La unica forma de hacerlo sería editando de nuevo el archivo de configuracion y poniendo “PasswordAuthentication” a “yes”, cargando de nuevo el servicio con system restart sshd y ahora ya si podria acceder por contraseña y copiar nuestras llaves.

```
# To disable tunneled clear text passwords, change to no here!  
PasswordAuthentication yes  
#PermitEmptyPasswords no
```

```
████████@ubuntu:~$ systemctl restart sshd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===  
Authentication is required to restart 'ssh.service'.  
Authenticating as: ██████████  
Password:  
==== AUTHENTICATION COMPLETE ===  
████████@ubuntu:~$
```

Y copiamos la clave desde la terminal del anfitrion a UbuntuServer

Luego habria que volver a poner “PasswordAuthentication” a “no” en el archivo de configuracion y reiniciar el servicio de nuevo.

Ssh-copy-id en windows?

Vamos a ver ahora otra medida de seguridad extra: Dejar exclusivamente a unos únicos usuarios que puedan acceder. Especificaremos en el archivo de configuración "AllowUsers" → separados por espacios pondremos los nombres de usuario de los usuarios que si pueden loggearse

Vamos a crear usuario en centOS y le asignamos una contraseña

```
[redacted]@localhost ~]$ sudo adduser pepe  
[sudo] password for [redacted]:  
[redacted]@localhost ~]$ sudo passwd pepe  
Cambiando la contraseña del usuario pepe.  
Nueva contraseña:  
Vuelva a escribir la nueva contraseña:  
passwd: todos los tokens de autenticación se actualizaron exitosamente.
```

Si intentamos acceder a nuestra máquina Ubuntu con nuestro usuario pepe, no nos deja acceder porque no tenemos activado el acceso por contraseña

```
[redacted]@localhost ~]$ ssh 192.168.56.105 -l pepe -p 22022  
pepe@192.168.56.105: Permission denied (publickey).
```

Nuevamente ponemos PasswordAuthentication a yes y reiniciamos servicio. Vemos que no puedo acceder como pepe porque no hemos creado el usuario en UbuntuServer (en el servidor, en la máquina receptora).

```
[redacted]@localhost ~]$ ssh 192.168.56.105 -l pepe -p 22022  
pepe@192.168.56.105's password:  
Permission denied, please try again.  
pepe@192.168.56.105's password:  
Permission denied, please try again.
```

Creamos la cuenta en Ubuntu Server :

```
[redacted]@ubuntu:~$ sudo adduser pepe  
Adding user `pepe' ...  
Adding new group `pepe' (1001) ...  
Adding new user `pepe' (1001) with group `pepe' ...  
Creating home directory `/home/pepe' ...  
Copying files from `/etc/skel' ...  
New password:  
Retype new password:  
passwd: password updated successfully  
Changing the user information for pepe  
Enter the new value, or press ENTER for the default  
    Full Name []:  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n] Y
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Probamos de nuevo

```
pepe@192.168.56.105's password:  
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-86-generic x86_64)  
  
* Documentation: https://help.ubuntu.com  
* Management: https://landscape.canonical.com  
* Support: https://ubuntu.com/advantage  
  
System information as of Wed Nov 3 11:35:33 UTC 2021  
  
System load: 0.0 Processes: 128  
Usage of /home: 0.2% of 468MB Users logged in: 1  
Memory usage: 22% IPv4 address for emp0s3: 10.0.2.15  
Swap usage: 0% IPv4 address for emp0s8: 192.168.56.105  
  
* Super-optimized for small spaces - read how we shrank the memory  
footprint of MicroK8s to make it the smallest full K8s around.  
  
https://ubuntu.com/blog/microk8s-memory-optimisation  
  
171 updates can be installed immediately.  
28 of these updates are security updates.  
To see these additional updates run: apt list --upgradable  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
pepe@ubuntu:~$ _
```

Efectivamente ya puedo entrar como nico a la otra máquina.

Vamos a decirle que únicamente sea mi usuario principal el que pueda entrar:
Editamos el archivo de configuración. Buscamos la opción de AllowUsers. No la vamos a encontrar. Por eso, lo añadimos nosotros.

```
Port 22022  
#AddressFamily any  
<#ListenAddress 0.0.0.0  
#ListenAddress ::  
AllowUsers nico  
#HostKey /etc/ssh/ssh_host_rsa_key  
#HostKey /etc/ssh/ssh_host_ecdsa_key  
#HostKey /etc/ssh/ssh_host_ed25519_key
```

Reiniciamos el servicio.

Intentamos acceder como usuario pepe

```
localhost ~ $ ssh 192.168.56.105 -l pepe -p 22022  
pepe@192.168.56.105's password:  
Permission denied, please try again.  
pepe@192.168.56.105's password:  
Permission denied, please try again.  
sshd[159]: Connection closed by 192.168.56.105 [preauth]
```

No tenemos acceso

Intentamos acceder como mi usuario principal

```
[REDACTED]@localhost ~]$ ssh 192.168.56.105 -l sergiohc -p 22022
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-86-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 System information as of Wed Nov  3 11:40:41 UTC 2021
```

y todo va genial

Ya tenemos configurado lo básico en cuanto a aspectos de seguridad que debemos contemplar.

Vamos a conocer ahora una utilidad: **sshfs**

Vamos a montar de forma transparente una ubicación remota y el acceso a esa información remota va a ser por ssh, con lo cual nos garantizamos la seguridad.

Creamos un directorio en nuestro pc

```
[REDACTED]@LAPTOP-6VCTDUOQ:~$ mkdir ./ubuntuserver
[REDACTED]@LAPTOP-6VCTDUOQ:~$ l
ubuntuserver/
```

Ahora le voy a indicar que monte en 192.168.56.105 con el usuario **sergiohc** y le especifico la ruta /home/**sergiohc** y que me lo monte en ./ubuntuserver/ con puerto 22022

Escribo: "sshfs **sergiohc**@192.168.56.105:/home/**sergiohc** ./ubuntuserver/ -p 22022"

```
[REDACTED]@LAPTOP-6VCTDUOQ:~$ sshfs [REDACTED]@192.168.56.105:/home/[REDACTED] ./ubuntuserver/ -p 22022
The authenticity of host '[192.168.56.105]:22022 ([192.168.56.105]:22022)' can't be established.
ECDSA key fingerprint is SHA256:ZQ+vBHqJ+Gzlo/QWvkh58h2thJr3DHAihyjJZXLxUAo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
[REDACTED]@192.168.56.105's password:
```

(me pide lo de la llave pública y la contraseña porque es la primera vez que entro → he empezado a trabajar con ubuntu wsl)

Le he echo un vistazo a “mount” para comprobar:

```
[192.168.56.105:/home/... on /home/.../ubuntuserver type fuse.sshfs (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
@LAPTOP-6VCTDUOQ:~$
```

Vemos como efectivamente tenemos acceso a la carpeta /home/... en el usuario ... en la máquina de ubuntuserver

Así, cuando haga “ls ./ubuntuserver”, voy a ver contenido de ubuntu server:

```
@LAPTOP-6VCTDUOQ:~$ ls -la ./ubuntuserver/
total 40
drwxr-xr-x 1 ... 4096 Nov  3 12:03 .
drwxr-xr-x 6 ... 4096 Nov  4 22:34 ..
-rw----- 1 ... 570 Nov  3 12:44 .bash_history
-rw-r--r-- 1 ... 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 ... 3771 Feb 25 2020 .bashrc
drwx----- 1 ... 4096 Sep 25 14:09 .cache
-rw----- 1 ... 4096 Nov  3 12:03 .cat.swp
-rw-r--r-- 1 ... 807 Feb 25 2020 .profile
drwx----- 1 ... 4096 Nov  3 12:20 .ssh
-rw-r--r-- 1 ... 0 Oct  3 11:37 .sudo_as_admin_successful
drwxr-xr-x 1 ... 4096 Oct  3 11:38 .vim
```

Si hacemos touch “hola” en la máquina virtual de ubuntu server:

```
@ubuntu:~$ touch hola
@ubuntu:~$ ls -la
total 40
drwxr-xr-x 5 ... 4096 Nov  4 21:40 .
drwxr-xr-x 5 ... 4096 Nov  3 11:33 ..
-rw----- 1 ... 570 Nov  3 11:44 .bash_history
-rw-r--r-- 1 ... 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 ... 3771 Feb 25 2020 .bashrc
drwx----- 2 ... 4096 Sep 25 12:09 .cache
-rw----- 1 ... 4096 Nov  3 11:03 .cat.swp
-rw-r--r-- 1 ... 807 Feb 25 2020 .profile
drwx----- 2 ... 4096 Nov  3 11:20 .ssh
-rw-r--r-- 1 ... 0 Oct  3 09:37 .sudo_as_admin_successful
drwxr-xr-x 2 ... 4096 Oct  3 09:38 .vim
-rw-rw-r-- 1 ... 0 Nov  4 21:40 hola
```

Remotamente también se ve:

```
@LAPTOP-6VCTDUOQ:~$ ls -la ./ubuntuserver/
total 40
drwxr-xr-x 1 ... 4096 Nov  4 22:40 .
drwxr-xr-x 6 ... 4096 Nov  4 22:34 ..
-rw----- 1 ... 570 Nov  3 12:44 .bash_history
-rw-r--r-- 1 ... 220 Feb 25 2020 .bash_logout
-rw-r--r-- 1 ... 3771 Feb 25 2020 .bashrc
drwx----- 1 ... 4096 Sep 25 14:09 .cache
-rw----- 1 ... 4096 Nov  3 12:03 .cat.swp
-rw-r--r-- 1 ... 807 Feb 25 2020 .profile
drwx----- 1 ... 4096 Nov  3 12:20 .ssh
-rw-r--r-- 1 ... 0 Oct  3 11:37 .sudo_as_admin_successful
drwxr-xr-x 1 ... 4096 Oct  3 11:38 .vim
-rw-rw-r-- 1 ... 0 Nov  4 22:40 hola
```

Otra funcionalidad bastante interesante es la de xForwarding. Con esto, nosotros en nuestro servidor podemos tener una aplicación que haga uso de la interfaz de ventanas mediante el xServer, cuando haga la petición de pintar una ventana (pintar su barra de menú, mostrar en display...) . Vamos a hacer que estas peticiones, en Igual de hacerse al servidor local se reenvíen al servidor de nuestro cliente y esa interfaz le aparezca a nuestro cliente. Pero la ejecución se estará realizando en nuestro servidor. Esto puede ser util por ejemplo cuando una aplicación se está ejecutando en un servidor remoto y queremos tener la interfaz en nuestro cliente.

Vamos a ver como hacer esta conexión.

Desde la terminal nos conectamos a Ubuntu Server con ssh. Pero esta vez con la opcion -X

```
@LAPTOP-6VCTDUOQ:~$ ssh 192.168.56.105 -p 22022 -X  
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-89-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com
```

Con la opción -X estamos haciendo el -xForward. Instalamos gedit con “sudo apt install gedit”. Al iniciarla vemos que nos abre la interfaz en nuestro anfitrión. La clave es que la interfaz la esta ejecutando el cliente pero el procesamiento que puede llevar detrás la aplicación así como la ubicación donde guarde los archivos será en la máquina remota (en el servidor). Puedo escribir cualquier cosa, guardarla y comprobar que efectivamente se almacena en el servidor.

Por último, vamos a conectar un par de utilidades: screen y tmux.

Estas dos aplicaciones nos van a permitir lanzar un trabajo en una terminal y dejar esa terminal sin ninguna sesión vinculada y volverla a retomar en el momento que queramos. Esto es útil especialmente para scripts o ejecuciones que van a tardar una cantidad de tiempo considerable. Nos va a permitir desloguearnos, apagar nuestro cliente y pasado un tiempo volver a conectar con la máquina y retomamos el control.

Otro escenario que puede ocurrir es que estamos ejecutando un procedimiento y a mitad se nos va la luz en el cliente o perdemos la conexión . Tendriamos que volver a loguearnos y reiniciar el procedimiento. Estos dos programas nos pueden ayudar bastante a trabajar con servidores remotos.

Ejemplo de ejecución

Vamos a abrir una terminal y vamos a conectarnos a UbuntuServer. Vamos a ejecutar ahí “vi miarchivo”



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



```
SS      0:00  \_ /usr/bin/dbus-daemon --session --address=st
Ss      0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100
Ss      0:00  \_ sshd: [priv]
S      0:00      \_ sshd: pts/0
Ss      0:00      \_ -bash
S+     0:00      \_ vi miarchivo
S=1    0:00  /usr/libexec/freedesktop/
```

En el lado del servidor vamos a ver qué está ocurriendo. Ponemos “ps axf” → Tenemos nuestro ssh con su bash y la aplicación vi editando “miarchivo”. Si cierro la terminal donde estaba editando el archivo, veo que poniendo otra vez “ps axf”, cualquier proceso de vi también se termina en cascada. Para evitar eso vamos a utilizar “screen”.

```
ubuntu:~$ ps axf | grep vi
801 ?      Ss  0:00 /usr/lib/accountsservice/accounts-daemon
1203 tty1    T   0:00  \_ sudo vi /etc/ssh/sshd_config
1204 tty1    T   0:00  |  \_ vi /etc/ssh/sshd_config
1249 tty1    T   0:00  \_ sudo vi /etc/ssh/sshd_config
1250 tty1    T   0:00  |  \_ vi /etc/ssh/sshd_config
12047 tty1   R+  0:00  \_ grep --color=auto vi
```

Nos logueamos en ubuntu de nuevo desde la terminal y llamamos a screen. Le damos a intro y ahí entramos en la pantalla. Desde ahí ponemos “vi miotroarchivo” y vemos que ocurre en el lado del servidor.

```
1705 ?      Ss  0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
12048 ?      Ss  0:00  \_ sshd: [priv]
12129 ?      S  0:00  \_ sshd: pts/0
12130 pts/0   Ss  0:00      \_ -bash
12139 pts/0   S+  0:00      \_ screen
12140 ?      Ss  0:00      \_ SCREEN
12141 pts/1   Ss  0:00      \_ /bin/bash
12148 pts/1   S+  0:00      \_ vi miotroarchivo
2412 ?      Ss  0:00  /usr/libexec/freedesktop/
```

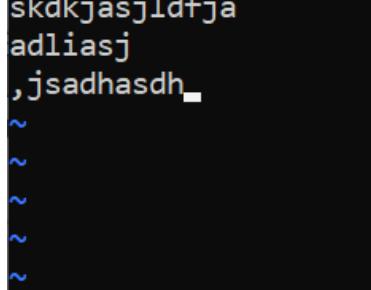
Como podemos ver, tenemos una salida distinta. Vemos otra cosa. En esta ocasión sshd que tiene bash. Bash ha invocado al comando screen y screen ha creado otro proceso llamado SCREEN, del cual cuelga un bash, del cual cuelga el “vi miotroarchivo”. Si yo ahora cierro la terminal donde estaba ejecutando “vi”, vemos que ocurre en el servidor:

```
12140 ?      Ss  0:00 SCREEN
12141 pts/1   Ss  0:00  \_ /bin/bash
12148 pts/1   S+  0:00      \_ vi miotroarchivo
```

Sigue SCREEN → bash → vi miotroarchivo

¿Cómo puedo recuperar mi otra pantalla? Puedo recuperarlo en la misma máquina UbuntuServer pero vamos a abrir terminal en nuestro anfitrion y vamos a conectarnos a UbuntuServer.

Desde ahí, vamos a escribir “screen -r -d”. Ahí tenemos mi otro archivo. Si escribiese algo y volviese a cerrarlo sin querer sin haberlo guardado, no pasaría nada. Puedo volver a recuperarlo



A terminal window showing a screen session. The session contains several lines of text: "skdkjasjldfja", "adliasj", ",jsadhasdh", and several blank lines starting with a tilde (~). The cursor is positioned at the end of the third line.

Incluso puedo hacer screen desde la propia máquina UbuntuServer y recuperar la última ventana con el último texto escrito. Escribo en UbuntuServer “screen -r -d” y podría volver a editar el archivo por donde iba.

¿Cómo podemos utilizar un “detached” del screen sin tener que cerrar la salida? Cntrl a + Cntrl d y ahora nos podríamos ir por ejemplo a una terminal, conectarnos y desde ahí hacer un listado de los screen (podemos tener varios)
“screen -list”

Me conecto con el que quiera con “screen -r nombredelarchivo -d”. El -d es para hacer detached de la máquina que lo esté usando. → cerrará el screen.

Tmux

Tiene la misma utilidad/funcionalidad. Ponemos en la terminal, conectados a UbuntuServer “tmux”.

Con CNTRL D podemos crear dos sesiones. La funcionalidad es exactamente igual que con screen.

Podemos hacer “CNTRL BD” para hacer el detached. Luego podemos hacer “tmux attach” y lo tenemos de nuevo

FAILTOBAN

Es un servicio (por lo tanto, lo tendremos en ejecución). Interactuaremos con él a través de systemd y con un comando especial: “fail2ban client”, que es el cliente. Lo que va a hacer fail2ban es un sondeo de los archivos del log que están en /var/log y va a analizar el contenido de esos archivos para ver las autenticaciones erróneas que se han producido. En caso de que haya una autenticación errónea, pasará a banearlo durante un tiempo determinado (este tiempo está configurado en el correspondiente archivo de configuración).

Podemos ver la funcionalidad en su página web: www.fail2ban.org/wiki/index.php

Se bloquean direcciones del protocolo de Internet (IPs), no usuarios. Si bloqueamos una IP a través de la cual acceden varios usuarios, estamos bloqueando a todo ese conjunto de usuarios.

Este servicio es fundamental tenerlo configurado para evitar ataques de fuerza bruta. Con esto impedimos que alguien nos pueda averiguar la contraseña por fuerza bruta.

Vamos a pasar a instalarlo/configurarlo en nuestro centOS.

Si ponemos “dnf search fail2ban” vemos que no encuentra nada. Esto es porque Fail2ban está dentro del conjunto de paquetes extendido. Ponemos “dns search epel” y encontramos epel-release. Lo instalamos con “sudo ndf install epel-release”

```
localhost ~]# dnf search epel
=====
Coincidencia en Nombre: epel =====
epel-release.noarch : Extra Packages for Enterprise Linux repository configuration
epel-next-release.noarch : Extra Packages for Enterprise Linux Next repository configuration
```

Recordemos levantar la interfaz de red con “sudo ifup enp0s3”

```
localhost ~]# sudo ifup enp0s3
[...]
localhost ~]# sudo dnf install epel-release
CentOS-8 - AppStream           11 kB/s | 4.3 kB     00:00
CentOS-8 - AppStream           96% [=====] 696 kB/s | 9.3 MB   00:00 ETA
```

Instalado.

Ahora ya si podemos hacer la búsqueda de fail2ban con “dnf search fail2ban”:

```
localhost ~]# dnf search fail2ban
Extra Packages for Enterprise Linux Modular 8 - x86_64          524 kB/s | 955 kB     00:01
Extra Packages for Enterprise Linux 8 - x86_64                  850 kB/s | 11 MB      00:12
Última comprobación de caducidad de metadatos hecha hace 0:00:01, el vie 05 nov 2021 09:18:51 EDT.
=====
Cocincidencia exacta en Nombre: fail2ban =====
fail2ban.noarch : Daemon to ban hosts that cause multiple authentication errors
=====
Cocincidencia en Nombre , Resumen: fail2ban =====
fail2ban-tests.noarch : Fail2Ban testcases
fail2ban-mail.noarch : Mail actions for Fail2Ban
fail2ban-selinux.noarch : SELinux policies for Fail2Ban
fail2ban-sendmail.noarch : Sendmail actions for Fail2Ban
fail2ban-firewalld.noarch : Firewalld support for Fail2Ban
fail2ban-shorewall.noarch : Shorewall support for Fail2Ban
fail2ban-server.noarch : Core server component for Fail2Ban
fail2ban-all.noarch : Install all Fail2Ban packages and dependencies
fail2ban-shorewall-lite.noarch : Shorewall lite support for Fail2Ban
fail2ban-systemd.noarch : Systemd journal configuration for Fail2Ban
fail2ban-hostsdeno.noarch : Hostsdeno (tcn wrrappers) support for Fail2Ban
```

Ponemos “sudo dnf install fail2ban”

Ya tenemos nuestro fail2ban instalado.

Vamos a comprobar el estado del servicio con “systemctl status fail2ban”

```
localhost ~]# systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
  Loaded: loaded (/usr/lib/systemd/system/fail2ban.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
    Docs: man:fail2ban(1)
```

Como vemos, esta deshabilitado (disabled). Esto significa que está cargado pero deshabilitado e inactivo. Tenemos que habilitarlo para que en el proximo reinicio de la máquina se active. Ponemos “systemctl enable fail2ban”

```
[redacted]@localhost ~]$ systemctl enable fail2ban
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-unit-files ====
Authentication is required to manage system service or unit files.
Authenticating as: [redacted]
Password:
==== AUTHENTICATION COMPLETE ====
Created symlink /etc/systemd/system/multi-user.target.wants/fail2ban.service → /usr/lib/systemd/system/fail2ban.service.
```

Volvemos a comprobar el estado:

```
[redacted]@localhost ~]$ systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
  Loaded: loaded (/usr/lib/systemd/system/fail2ban.service; enabled; vendor preset: disabled)
  Active: inactive (dead)
    Docs: man:fail2ban(1)
```

Vemos que efectivamente ahora está habilitado (enabled) pero aún está inactivo (inactive). Por lo tanto, la próxima vez que reiniciemos la máquina se iniciará el servicio pero todavía sigue inactivo.

Iniciamos el servicio con “sudo systemctl start fail2ban”

```
[redacted]@localhost ~]$ sudo systemctl start fail2ban
[sudo] password for [redacted]
```

Y comprobamos el estado:

```
[redacted]@localhost ~]$ systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
  Loaded: loaded (/usr/lib/systemd/system/fail2ban.service; enabled; vendor preset: disabled)
  Active: active (running) since Fri 2021-11-05 09:31:32 EDT; 21s ago
    Docs: man:fail2ban(1)
   Process: 23123 ExecStartPre=/bin/mkdir -p /run/fail2ban (code=exited, status=0/SUCCESS)
   Main PID: 23124 (fail2ban-server)
     Tasks: 3 (limit: 5019)
    Memory: 11.1M
      CGroup: /system.slice/fail2ban.service
              └─23124 /usr/bin/python3.6 -s /usr/bin/fail2ban-server -xf start

nov 05 09:31:32 localhost.localdomain systemd[1]: Starting Fail2Ban Service...
nov 05 09:31:32 localhost.localdomain systemd[1]: Started Fail2Ban Service.
nov 05 09:31:32 localhost.localdomain fail2ban-server[23124]: Server ready
```

Vemos que ya tenemos el servicio habilitado y activo. Está en ejecución.

Vamos a pasar a configurarlo.



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Usaremos el comando “fail2ban-client”. Con esto podremos interactuar con las jails (cárcel). Se van a analizar los distintos logs, a partir de ahí se buscarán las direcciones IP y estas direcciones IP pasarán a una serie de cárceles (jails). Estas últimas las definiremos para cada servicio en concreto.

Las vamos a definir en el archivo de configuración de fail2ban.

Ponemos “cd /etc/fail2ban” y “ls”

```
[REDACTED]@localhost ~]$ cd /etc/fail2ban  
[REDACTED]@localhost fail2ban]$ ls  
action.d      fail2ban.d  jail.conf  paths-common.conf  
fail2ban.conf  filter.d   jail.d    paths-fedora.conf
```

El archivo jail.conf es el que nos va a permitir configurar las cárceles.

Podemos echar un vistazo a jail.conf con “less jail.conf”.

```
# HOW TO ACTIVATE JAILS:  
#  
# YOU SHOULD NOT MODIFY THIS FILE.  
#  
# It will probably be overwritten or improved in a distribution update.  
#  
# Provide customizations in a jail.local file or a jail.d/customisation.local.  
# For example to change the default bantime for all jails and to enable the  
# ssh-iptables jail the following (uncommented) would appear in the .local file.  
# See man 5 jail.conf for details.  
#  
# [DEFAULT]  
# bantime = 1h  
#  
# [sshd]  
# enabled = true  
#  
# See jail.conf(5) man page for more information
```

Vemos que nos pone en mayúscula como activar las cárceles. También nos indica que NO debemos modificar este archivo. Esto es porque este archivo es el archivo por defecto y en caso de que se actualice fail2ban, sobreescribiría nuestra configuración → sería un desastre.

Mejor hacemos una configuración local. Ponemos “sudo cp -a jail.conf jail.local”

```
[REDACTED]@localhost fail2ban]$ sudo cp -a jail.conf jail.local  
[sudo] password for [REDACTED]:
```

En la configuración local ya sí que podemos editar tranquilamente el jail.local para configurar los parámetros de nuestra cárcel (en nuestro caso para ssh)

Hay unas líneas de ejemplos [DEFAULT] [sshd] por ejemplo.

Buscamos la cárcel [sshd], la cual está predefinida. Vamos a pasar a activarla. Ponemos “sudo vi jail.local”

```
#  
# JAILS  
#  
  
#  
# SSH servers  
#  
  
[sshd]  
  
# To use more aggressive sshd modes set filter parameter "mode" in jail.local:  
# normal (default), ddos, extra or aggressive (combines all).  
# See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and details.  
#mode = normal  
port = ssh  
logpath = %(sshd_log)s  
backend = %(sshd_backend)s
```

Añadimos una línea “enabled = true”

```
[sshd]  
  
# To use more aggressive sshd  
# normal (default), ddos, ext  
# See "tests/files/logs/sshd"  
#mode = normal  
port = ssh  
enabled = true  
logpath = %(sshd_log)s  
backend = %(sshd_backend)s
```

Guardamos y salimos con wq

Hacemos “sudo fail2ban-client status sshd” para comprobar el estado de la carcel sshd. Como vemos, no encuentra el jail sshd.

```
[redacted]@localhost fail2ban]$ sudo fail2ban-client status sshd  
2021-11-05 09:46:43,030 fail2ban                         [23170]: ERROR    NOK: ('sshd',)  
Sorry but the jail 'sshd' does not exist
```

Reiniciamos el servicio “systemctl restart fail2ban”

```
[redacted]@localhost fail2ban]$ systemctl restart fail2ban  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====  
Se requiere autenticación para reiniciar 'fail2ban.service'.  
Authenticating as: [redacted]  
Password:  
==== AUTHENTICATION COMPLETE ====
```

Ahora comprobamos el estado de la carcel sshd de nuevo

```
[root@localhost fail2ban]$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 0
| |- Total failed: 0
| `-- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 0
  |- Total banned: 0
  '-- Banned IP list:
```

Como vemos, está operativo.

Ahora vamos a comprobar que funciona.

Abrimos la terminal y hacemos “ssh 192.168.56.110 -p 22022” para conectarnos a centOS.

Me equivoco con la contraseña 3 veces a la fuerza.

```
C:\Users\██████████>ssh 192.168.56.110 -p 22022
██████████@192.168.56.110's password:
Permission denied, please try again.
██████████@192.168.56.110's password:
Permission denied, please try again.
██████████@192.168.56.110's password:
██████████@192.168.56.110: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
```

Como vemos, me deniega la petición.

Comprobamos ahora el estado de la carcel desde centOS:

```
[root@localhost fail2ban]$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 1
| |- Total failed: 4
| `-- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 0
  |- Total banned: 0
  '-- Banned IP list:
```

Ha detectado que ha habido 4 intentos erróneos (uno lo he hecho yo antes sin querer, sería 3) del servicio ssh. Pero por defecto tienen que haber 5 intentos para que me banee. Volvemos a equivocarnos con la contraseña hasta 5 o más:

```
C:\Users\██████████>ssh 192.168.56.110 -p 22022
██████████@192.168.56.110's password:
Permission denied, please try again.
██████████@192.168.56.110's password:
Permission denied, please try again.
██████████@192.168.56.110's password:
██████████@192.168.56.110: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
```

Volvemos a ver el estado

```
[localhost fail2ban]$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 1
| |- Total failed:    8
| `-' Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 1
  |- Total banned:     1
  `-' Banned IP list:   192.168.56.1
```

Vemos que al procesar el archivo de log ha detectado que la IP de mi terminal, la va a bloquear.

Vamos a intentar entrar ahora si con la contraseña correcta:

```
C:\Users\██████████@192.168.56.110 -p 22022
██████████@192.168.56.110's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Fri Nov  5 09:10:52 2021
```

Como vemos, me deja!

Esto se debe a que hemos cometido una improdudencia. No hemos modificado el puerto. Al activar la carcel de ssh, habia una variable “port”. Vamos a modificar el port por defecto de ssh para que sea el 22022.

Ponemos “sudo vi jail.local” y lo modificamos.

```
[sshd]
# To use more aggressive
# normal (default), ddos
# See "tests/files/logs"
#mode    = normal
port      = 22022
enabled   = true
logpath  = %(sshd_log)s
```

Reiniciamos el servicio con “sudo systemctl restart fail2ban”.

Comprobamos que efectivamente ya no nos deja entrar desde la terminal a centOS.

```
C:\Users\██████████@192.168.56.110 -p 22022
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Comprobamos el estado de la cárcel y vemos que nuestra IP sigue baneada y automáticamente ha modificado las reglas para impedir nuestro acceso por el puerto correcto.

```
[REDACTED]@localhost fail2ban]$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 0
| |- Total failed: 0
| ` Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 1
  |- Total banned: 1
  `- Banned IP list: 192.168.56.1
```

Lo último que nos queda es saber sacar una IP de un usuario que se ha equivocado por despiste.

Utilizamos el comando "sudo fail2ban-client set sshd unbanip 192.168.56.1"

```
[REDACTED]@localhost fail2ban]$ sudo fail2ban-client set sshd unbanip 192.168.56.1
[REDACTED]@localhost fail2ban]$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
| |- Currently failed: 0
| |- Total failed: 0
| ` Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
  |- Currently banned: 0
  |- Total banned: 1
  `- Banned IP list:
```

Como vemos, no se actualiza el contador, es un bugg. Si nos intentamos conectar vemos que sí nos deja:

```
C:\Users\[REDACTED]\ssh [REDACTED]@192.168.56.110 -p 22022
[REDACTED]@192.168.56.110's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Fri Nov  5 09:55:31 2021 from 192.168.56.1
[REDACTED]@localhost ~]$
```

Esto es todo lo básico.

Podríamos modificar el tiempo que hay que pasar baneado. Fail2ban establece un tiempo de baneo. No banea indefinidamente

Podemos modificar parámetros como maxretry (numero máximo de reintentos), el bantime (tiempo de baneo)...

Recordemos: Esto no es exclusivo de sshd sino que se puede aplicar por otros servicios
→ siempre que pase por el log

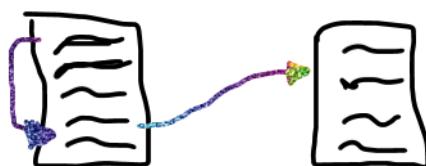
LAMP

Vamos a ver la instalación de la pila LAMP en centOS. Antes de continuar con el proceso de instalación, vamos a ver por qué de esta pila.

Cuando hablamos de la pila LAMP, el concepto de pila lo podemos entender porque vamos apilando una capa de software sobre otra.

LAMP surge por ciertos problemas de comunicación entre los físicos que estaban compartiendo los resultados de sus experimentos. El inventor de LAMP propuso una forma de ir organizando la información aprovechando la existencia de los hipertextos.

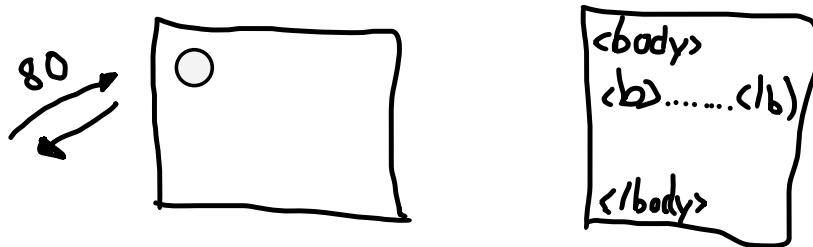
Tenemos un documento y dentro de ese documento enlazamos a un segundo documento. Podemos hacer referencia también a otra parte del mismo documento.



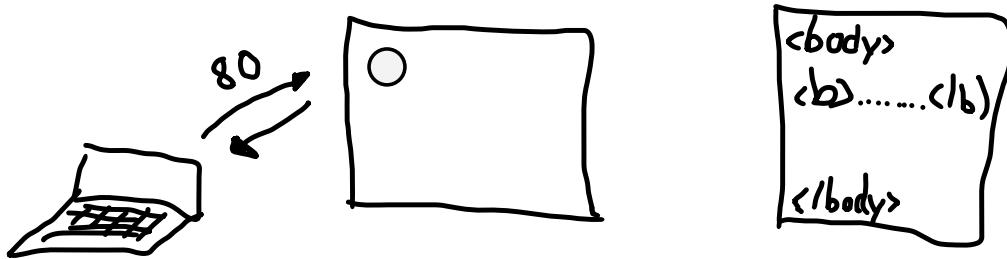
A partir de aquí, desarrolló el protocolo HTTP. (Hyper Text Transfer Protocol).

Con esto lo que estamos haciendo es lo siguiente:

Imaginemos nuestro servidor en el que tenemos un programita/servicio que está escuchando peticiones constantemente en el puerto 80 y una vez que recibe una petición (p.e get), devuelve un documento HTML (Hyper Text Markup Language). Es un lenguaje de marcas para hipertexto. HTML tiene la forma <body></body> (por ejemplo).



Ya tenemos nuestro programa que envía este documento HTML y el cliente cuando recibe este documento, tiene un programa especial: el navegador web. Este último coge, analiza el documento y muestra en pantalla el resultado de interpretar el documento.

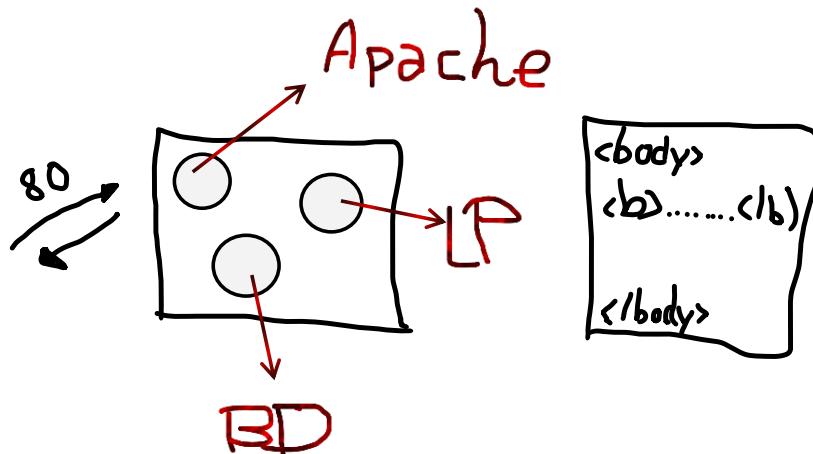


Después se le quiso dar mas interactividad y lógica (por ejemplo, que dentro del documento HTML que aparezca un botoncito que cuando pulsemos nos puestre información). → Aparece JavaScript como lenguaje de scripting que se podía incrustar dentro del lenguaje HTML utilizando las etiquetas `<script>codigo</script>`. El código se ejecutaría en la parte del cliente una vez enviado. De este modo, si quisieramos añadir una contraseña para mostrar la información del botón, deberíamos incrustar/hardcodear la contraseña dentro. Esto es una mala práctica porque el cliente tendría acceso a la contraseña. O por ejemplo, si tenemos un blog y queremos que nos hagan comentarios y otras personas lo vean, ¿cómo damos ese dinamismo y como almacenamos esos datos? Vamos a ver otros servicios que necesitamos.

Por una parte tenemos la lógica de control que va a venir por un intérprete. Puede ser un programa pero normalmente en la web se utilizan lenguajes interpretados como PHP, Python, Perl... Al ser interpretados, tienen varias ventajas como poder modificar alguna línea sin tener que recompilar todo y continuar dando servicio, acceder a gran cantidad de bibliotecas para procesar los strings de HTML... etc

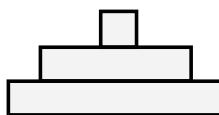
Si tenemos comentarios, pares de usuario/contraseña, etc tenemos que almacenarlos en algún lado. En archivos no sería elegante/eficiente. Necesitamos otro servicio que nos de acceso a los datos. Desde el punto de vista de las bases de datos relacionales tenemos MySQL, MariaDB... Recientemente hay una tendencia: no tener bases de datos relacionales. Esto se debe a que para ciertos usuarios podemos tener mucha información pero para otros no (solo usuario y contraseña por ejemplo). Estructurar esto dentro de una tabla rígida daría lugar a muchos campos vacíos. No es el modo más interesante de almacenarlos. En esta línea aparecen las bases de datos que se conocen como NoSQL o No Relational Data Base.

Ya tenemos nuestros elementos. El programita que está escuchando nuestras peticiones puede ser Apache o lighttpd



Todo esto va a ser ejecutado en una máquina Linux. Para Windows sería XAMP.

Si nos fijamos, tenemos una estructura de capas donde tenemos en primer lugar un lugar donde almacenamos los datos (mysql), sobre este construimos una lógica que nos permite acceder a los datos y modificarlos y como interfaz hacia afuera tenemos nuestro servidor web. Ya tenemos el concepto de la pila LAMP claro.



Tenemos muchos CMS (Content Management System) como WordPress, PrestaShop, PHP, Joomla....). Queremos tener una plataforma web con un mínimo de funcionalidad. Con esta pila satisfacemos los requisitos.

Vamos a pasar a la consola y vamos a ir instalando los componentes uno a uno en nuestra máquina centOS.

El primer componente que vamos a instalar es el apache. Hacemos “dnf search apache” y vemos unos cuantos paquetes. Como vemos, uno de ellos es el httpd → el servidor apache. Vamos a instalarlo con “sudo dnf install httpd”.

```
==== Coincidencia en Resumen: apache =====
httpd.x86_64 : Apache HTTP Server
librdkafka.i686 : The Apache Kafka C library
[root@localhost ~]# sudo dnf install httpd
CentOS-8 - AppStream                               5.3 kB/s | 4.3 kB     00:00
CentOS-8 - Base                                    3.8 MB/s | 3.9 kB     00:00
CentOS-8 - Extras                                 3.4 kB/s | 1.5 kB     00:00
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Una vez instalado, sabemos que tenemos que comprobar las cosas como siempre.

Vamos a utilizar "systemctl status httpd"

```
[REDACTED]localhost ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
            └─php-fpm.conf
    Active: inactive (dead)
   Docs: man:httpd.service(8)
```

Como vemos, nos aparece deshabilitado e inactivo. Queremos que el servicio esté habilitado y en el próximo servicio queremos que el servicio de httpd esté funcionando. Lo habilitamos poniendo "sudo systemctl enable httpd":

```
[REDACTED]localhost ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/
httpd.service.
```

Comprobamos el estado:

```
[REDACTED]localhost ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
            └─php-fpm.conf
    Active: inactive (dead)
   Docs: man:httpd.service(8)
```

Como vemos, habilitarlo no significa que comience a funcionar. Para el resto de la sesión queremos hacer que nuestro servicio esté funcionando. Ponemos "sudo systemctl start httpd" y volvemos a comprobar el estado

```
[REDACTED]localhost ~]$ sudo systemctl start httpd
[REDACTED]localhost ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
            └─php-fpm.conf
    Active: active (running) since Sun 2021-11-07 13:40:41 EST; 4s ago
      Docs: man:httpd.service(8)
   Main PID: 1475 (httpd)
     Status: "Started, listening on: port 80"
        Tasks: 213 (limit: 5019)
       Memory: 29.1M
      CGroup: /system.slice/httpd.service
              ├─1475 /usr/sbin/httpd -DFOREGROUND
              ├─1481 /usr/sbin/httpd -DFOREGROUND
              ├─1482 /usr/sbin/httpd -DFOREGROUND
              ├─1483 /usr/sbin/httpd -DFOREGROUND
              ├─1484 /usr/sbin/httpd -DFOREGROUND

nov 07 13:40:41 localhost.localdomain systemd[1]: Starting The Apache HTTP Server...
nov 07 13:40:41 localhost.localdomain httpd[1475]: AH00558: httpd: Could not reliably determine the
nov 07 13:40:41 localhost.localdomain systemd[1]: Started The Apache HTTP Server.
nov 07 13:40:42 localhost.localdomain httpd[1475]: Server configured, listening on: port 80
```

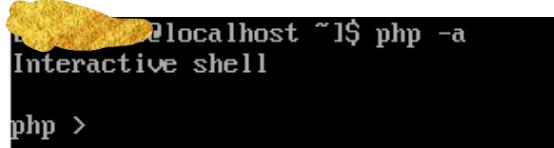
Ya lo tenemos en ejecución.

Ahora vamos a comprobar que funciona. Para ello, utilizaremos el comando “curl” (ver url). Podemos poner “curl localhost” haciendo referencia a nuestra máquina. Como vemos, nuestro servidor httpd está funcionando en el puerto por defecto (80) y nos devuelve una página web.

Ahora vamos a pasar a instalar PHP.

Hacemos “dnf search PHP” y ponemos “sudo dnf install php”.

¿Ahora como comprobamos que está instalado correctamente? Esto no es un servicio. Es el intérprete que se ejecutará cuando se le llame por el módulo de apache. Podemos comprobar que funciona con “php -a”



```
localhost ~]$ php -a
Interactive shell

php >
```

Tenemos nuestro intérprete en funcionamiento.

Por último, nos queda el último módulo de la pila: MySQL o MariaDB. En nuestro caso el último. Ponemos “dnf search mariadb” y luego “sudo dnf install mariadb”.

Vamos a comprobar como está el estado del servicio. Ponemos “systemctl status mariadb”.

Tenemos un error. Esto ha ocurrido porque solo hemos instalado el cliente (mariadb) y no el servidor. No tenemos ningún servicio activado. Hay que instalar el paquete server con “sudo dnf install mariadb-server”.

Una vez instalado, vamos a comprobar el estado del servicio.



```
localhost ~]$ systemctl status mariadb
● mariadb.service - MariaDB 10.3 database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
```

Como vemos, está desactivado y deshabilitado. Lo habilitamos y lo iniciamos.



```
localhost ~]$ sudo systemctl enable mariadb
[sudo] password for [REDACTED]
Sorry, try again.
[sudo] password for [REDACTED]
Created symlink /etc/systemd/system/mysql.service → /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/mysqld.service → /usr/lib/systemd/system/mariadb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service → /usr/lib/systemd/system/mariadb.service.

localhost ~]$ sudo systemctl start mariadb
```

Comprobamos su estado de nuevo.

```
[redacted]@localhost ~]$ systemctl status mariadb
● mariadb.service - MariaDB 10.3 database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2021-11-07 13:51:11 EST; 1s ago
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
  Process: 1850 ExecStartPost=/usr/libexec/mysql-check-upgrade (code=exited, status=0/SUCCESS)
  Process: 1780 ExecStartPre=/usr/libexec/mysql-prepare-db-dir mariadb.service (code=exited, status=0/SUCCESS)
  Process: 1756 ExecStartPre=/usr/libexec/mysql-check-socket (code=exited, status=0/SUCCESS)
 Main PID: 1818 (mysqld)
    Status: "Taking your SQL requests now..."
      Tasks: 30 (limit: 5019)
     Memory: 104.7M
        CGroup: /system.slice/mariadb.service
                  └─1818 /usr/libexec/mysqld --basedir=/usr
```

Ya está activo y en ejecución.

Ahora viene un paso muy importante. Tendremos que irnos a la documentación oficial.

Si nosotros ejecutamos “mysql” (intentamos conectarnos con el usuario), nos dice que no. No podemos conectarnos con el servidor a través de nuestro usuario.

```
[redacted]@localhost ~]$ mysql
ERROR 1045 (28000): Access denied for user 'redacted'@'localhost' (using password: NO)
```

Podemos probar como usuario root “mysql -u root”

```
[redacted]@localhost ~]$ mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.3.28-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Tengo acceso como usuario root. No he tenido que introducir ninguna contraseña. El manual indica que después de una instalación tenemos que ejecutar “mysql_secure_installation”

```
[redacted]@localhost ~]$ mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
```

Nos pregunta por la contraseña de root. Actualmente es ninguna desde fuera. Alguien podría entrar/conectarse desde fuera a nuestro servidor de base de datos.

Le ponemos contraseña al root.

```
Enter current password for root (enter for none):  
OK, successfully used password, moving on...  
  
Setting the root password ensures that nobody can log into the MariaDB  
root user without the proper authorisation.  
  
Set root password? [Y/n] y  
New password:  
Re-enter new password:  
Password updated successfully!  
Reloading privilege tables..  
... Success!
```

Nos pregunta si nos elimina un usuario anónimo. Lo quitamos también (Y).

```
Remove anonymous users? [Y/n] y  
... Success!  
  
Normally, root should only be allowed to connect from 'localhost'. This  
ensures that someone cannot guess at the root password from the network.  
  
Disallow root login remotely? [Y/n]
```

Vamos a quitarle la posibilidad a mi usuario root de mysql (tengamos en cuenta que hay usuario root para el sistema y usuario root para mysql). Ponemos “Y” para deshabilitarle el acceso a root remoto

Eliminamos también la base de datos que hay de prueba (Y). Recargamos la tabla de privilegios (“Y”) y ya tenemos configurada nuestra base de datos de MariaDB

```
Disallow root login remotely? [Y/n] y  
... Success!  
  
By default, MariaDB comes with a database named 'test' that anyone can  
access. This is also intended only for testing, and should be removed  
before moving into a production environment.  
  
Remove test database and access to it? [Y/n] y  
- Dropping test database...  
... Success!  
- Removing privileges on test database...  
... Success!  
  
Reloading the privilege tables will ensure that all changes made so far  
will take effect immediately.  
  
Reload privilege tables now? [Y/n] y  
... Success!  
  
Cleaning up...  
  
All done! If you've completed all of the above steps, your MariaDB  
installation should now be secure.  
  
Thanks for using MariaDB!
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Para comprobar que funciona, hacemos "mysql" y no nos deja conectarnos ni con usuario root ("mysql -u root"). Nos pide la autentificación por contraseña.

Ponemos "mysql -u root -p" y con la contraseña sí podemos acceder:

```
[redacted]@localhost ~]$ mysql  
ERROR 1045 (28000): Access denied for user [redacted]'localhost' (using password: NO)  
[redacted]@localhost ~]$ mysql -u root  
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)  
[redacted]@localhost ~]$ mysql -u root -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 24  
Server version: 10.3.28-MariaDB MariaDB Server
```

Vamos a conectarnos con Apache. Vamos a abrir la terminal fuera de centOS y vamos a ver que apache es capaz de servirnos una pagina. Ponemos "curl 192.168.56.110".

```
[redacted]@LAPTOP-6VCTDUOQ:~$ curl 192.168.56.110  
curl: (7) Failed to connect to 192.168.56.110 port 80: No route to host  
[redacted]@LAPTOP-6VCTDUOQ:~$
```

No nos deja.

Comprobamos que tenemos ping:

```
[redacted]@LAPTOP-6VCTDUOQ:~$ ping 192.168.56.110  
PING 192.168.56.110 (192.168.56.110) 56(84) bytes of data.  
64 bytes from 192.168.56.110: icmp_seq=1 ttl=63 time=0.788 ms  
64 bytes from 192.168.56.110: icmp_seq=2 ttl=63 time=1.35 ms  
64 bytes from 192.168.56.110: icmp_seq=3 ttl=63 time=1.25 ms  
64 bytes from 192.168.56.110: icmp_seq=4 ttl=63 time=1.07 ms
```

Sí tenemos comunicación.

¿Qué está causando este problema? El cortafuegos es el que está impidiendo que tengamos acceso al puerto. Lo añadimos de forma permanente (queremos que permanezca cuando reiniciemos). Ponemos por tanto "sudo firewall-cmd --add-port=80/tcp --permanent"

```
[redacted]@localhost ~]$ sudo firewall-cmd --add-port=80/tcp --permanent  
[sudo] password for [redacted]:  
success
```

Recargamos:

```
[redacted]@localhost ~]$ sudo firewall-cmd --reload  
success
```



WUOLAH

Vamos a probar que tenemos conexión desde la terminal con “curl 192.168.56.110” y efectivamente si nos deja.

Ahora solo nos queda ver que podemos integrar un script de PHP que se conecta a una base de datos y que el resultado nos lo muestra a través de un documento HTML.

Buscamos en google “php mysql connect”

Nos aparece la documentación. Nos aparece la función de “mysql_connect” pero está obsoleta. Mejor “mysqli_connect”

Tenemos un ejemplo de “mysqli_connect”.

```
<?php  
$enlace = mysqli_connect("127.0.0.1", "mi_usuario", "mi_contraseña", "mi_bd");  
  
if (!$enlace) {  
    echo "Error: No se pudo conectar a MySQL." . PHP_EOL;  
    echo "errno de depuración: " . mysqli_connect_errno() . PHP_EOL;  
    echo "error de depuración: " . mysqli_connect_error() . PHP_EOL;  
    exit;  
}  
  
echo "Éxito: Se realizó una conexión apropiada a MySQL! La base de datos mi_bd es genial." . PHP_EOL;  
echo "Información del host: " . mysqli_get_host_info($enlace) . PHP_EOL;  
  
mysqli_close($enlace);  
?>
```

En primer lugar tenemos las etiquetas para comenzar y finalizar el código de php. Y vemos una variable: enlace. Esta tiene asignado el resultado de llamar a “mysqli_connect” con los parámetros de localhost, del usuario, la contraseña y la base de datos a la que nos queremos conectar. En base al resultado de esa variable, hacemos una comprobación. Si ha habido un problema se mostrará un mensaje de error. En caso de éxito mostrará la información del host y cerraremos el enlace al final.

Así, podremos ver si todos nuestros componentes están funcionando de forma correcta.

Copiamos ese script en nuestro equipo de centOS. Accedemos a través de SSH a centOS.

Nuestro archivo lo tenemos que ubicar donde sea accesible a través de http. Podemos visualizar el archivo de configuración de apache:

Ponemos “less /etc/httpd/conf/httpd.conf” y veremos que el parámetro “DocumentRoot”, que es el que nos indica dónde se almacenan por defecto los documentos, está en /var/www/html. Ahí es donde debemos ubicar nuestro script.

```
#  
# DocumentRoot: The directory out of which you will serve your  
# documents. By default, all requests are taken from this directory, but  
# symbolic links and aliases may be used to point to other locations.  
#  
DocumentRoot "/var/www/html"
```

Nos vamos a ese directorio y hacemos “sudo vi index.php”. Aquí inserto lo que hemos copiado (el ejemplo de php).

```
[REDACTED]@localhost ~]$ cd /var/www/html  
[REDACTED]@localhost html]$ sudo vi index.php
```

Tenemos que manipular el código antes de guardarla.

- ¿Es necesario modificar el localhost? Si PHP (el intérprete) se está ejecutando dentro de centOS, el servidor de mysql también está dentro de centos. El parámetro de la IP está correcto.
- ¿Qué usuario tenemos en MySQL/MariaDB? Solo tenemos root. Lo ponemos.
- Ponemos la contraseña del usuario root
- mi_bd es una base de datos que aun no tenemos y la vamos a crear ahora.

```
$enlace = mysqli_connect("127.0.0.1", "root", "practicas,ise", "mi_bd");
```

Para ello, nos conectamos como usuario root a mysql con “mysql -u root -p” y hacemos un “CREATE DATABASE mi_bd;”

```
[REDACTED]@localhost html]$ mysql -u root -p  
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.  
Your MariaDB connection id is 26  
Server version: 10.3.28-MariaDB MariaDB Server  
  
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
MariaDB [(none)]> CREATE DATABASE mi_bd;  
Query OK, 1 row affected (0.000 sec)
```

Nuestro script ya puede funcionar.

Vamos a probarlo. Nos vamos a un navegador y ponemos “192.168.56.110”. Como vemos, nos funciona la página de prueba.

Si pedimos el archivo php poniendo en el navegador “192.168.56.110/index.php”, ¿qué ocurre? Nos lo sirve como un documento de texto. Reflexionamos: ¡Estamos accediendo a nuestro servidor a través de la IP y estamos viendo la contraseña del usuario root y el nombre de la base de datos a los que nos vamos a conectar!. Es importante que no se introduzcan los parámetros hardcodeados dentro del código. Que esto esté en otro archivo protegido para que en el caso de haber un error en el servidor, no muestre más información de la estrictamente necesaria. Por esto es importante una adecuada configuración.

Lo que está ocurriendo en este caso es que Apache no está interpretando PHP → Está solamente devolviendo el documento. Vamos a modificar la configuración de php para indicarle que interprete este archivo.

Nos vamos a centOS y editamos el archivo de configuración httpd. Ponemos “sudo vi /etc/httpd/conf/httpd.conf”

Le podemos indicar en el directorio index que sirva por defecto cualquier archivo con la extensión php. (añadimos a “DirectoryIndex index.html” un “*.php”)

```
#  
# DirectoryIndex: sets the file that Apache will serve if a directory  
# is requested.  
#  
<IfModule dir_module>  
    DirectoryIndex index.html  
</IfModule>  
  
<IfModule dir_module>  
    DirectoryIndex index.html *.php  
</IfModule>
```

Si volvemos a recargar el navegador, volvemos a tener el problema. Hay que reiniciar el servicio httpd. Ponemos “systemctl restart httpd”

```
[REDACTED]@localhost ~]$ systemctl restart httpd  
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ====  
Se requiere autenticación para reiniciar 'httpd.service'.  
Authenticating as: [REDACTED]  
Password:  
==== AUTHENTICATION COMPLETE ===
```

Ahora si, si recargamos el navegador tenemos un “HTTP ERROR 500” Esto significa que el servidor ha tenido un error. → Vamos a ver qué ocurre

¿Somos capaces de ejecutar nuestro script en el servidor? Ponemos “cd /var/www/html/” y luego php index.php. Con esto vamos a decirle al intérprete de php que ejecute nuestro archivo.

```
[REDACTED]@localhost ~]$ cd /var/www/html  
[REDACTED]@localhost html]$ php index.php  
  
PHP Fatal error:  Uncaught Error: Call to undefined function mysqli_connect() in /var/www/html/index.php:3  
Stack trace:  
#0 {main}  
    thrown in /var/www/html/index.php on line 3
```



GANAR UN FIAT 500 CON BURN ENERGY

1 LATA = 1 PARTICIPACIÓN



Nos encontramos con un error. Nos dice que hay una función que no conoce "mysqli_connect()". Esto es porque no hemos instalado la biblioteca que comunica php con mysql

Ponemos "sudo dnf search php | grep mysql" y luego "sudo dnf install php-mysqlnd"

```
[REDACTED]localhost html]$ dnf search php | grep mysql  
php-mysqlnd.x86_64 : A module for PHP applications that use MySQL databases  
[REDACTED]
```

Instalado:

```
php-mysqlnd-7.2.24-1.module_el8.2.0+313+b04d0a66.x86_64  
php-pdo-7.2.24-1.module_el8.2.0+313+b04d0a66.x86_64
```

¡Listo!

Ahora vamos a probar de nuevo a ejecutar el index.php con "php index.php".

```
[REDACTED]localhost html]$ php index.php  
Exito: Se realizó una conexión apropiada a MySQL! La base de datos mi_bd es genial.  
Información del host: 127.0.0.1 vía TCP/IP
```

Ahora si, desde el propio servidor tenemos acceso a la base de datos. Vamos a ver si desde el navegador funciona.

No, tenemos otro problema → De permisos (Permission denied)

Echamos un vistazo a "/var/log/audit/audit.log" con sudo. Nos vamos al final del archivo y podemos ver un permiso asociado a httpd. No se le da permiso a httpd para que conecte con mysql. Selinux es el que impide esto. Selinux tiene una lista de booleanos (podemos consultarla con "getsebool -a"). Hacemos un filtro con los referentes a httpd con "getsebool -a | grep httpd"

```
[REDACTED]localhost html]$ getsebool -a | grep httpd  
httpd_anon_write --> off  
httpd_builtin_scripting --> on  
httpd_can_check_spam --> off  
httpd_can_connect_ftp --> off  
httpd_can_connect_ldap --> off  
httpd_can_connect_mythtv --> off  
httpd_can_connect_zabbix --> off  
httpd_can_network_connect --> off  
httpd_can_network_connect_cobbler --> off  
httpd_can_network_connect_db --> off  
httpd_can_network_memcache --> off  
httpd_can_network_relay --> off  
httpd_can_sendmail --> off
```

Nos interesa “httpd_can_network_connect_db”. Vemos que está a off. Hay que ponerlo a on. Ponemos “sudo setsebool -P httpd_can_network_connect_db=on” (-P para de manera permanente)

```
[root@localhost ~]# sudo setsebool -P httpd_can_network_connect_db=on
[ 5865.576721] SELinux:  Converting 2340 SID table entries...
[ 5866.034818] SELinux:  policy capability network_peer_controls=1
[ 5866.035031] SELinux:  policy capability open_perms=1
[ 5866.035222] SELinux:  policy capability extended_socket_class=1
[ 5866.035485] SELinux:  policy capability always_check_network=0
[ 5866.035687] SELinux:  policy capability cgroup_seclabel=1
[ 5866.035883] SELinux:  policy capability nnp_nosuid_transition=1
```

Comprobamos si se ha modificado

```
[root@localhost ~]# getsebool -a | grep httpd
httpd_anon_write --> off
httpd_builtin_scripting --> on
httpd_can_check_spam --> off
httpd_can_connect_ftp --> off
httpd_can_connect_ldap --> off
httpd_can_connect_mythtv --> off
httpd_can_connect_zabbix --> off
httpd_can_network_connect --> off
httpd_can_network_connect_cobbler --> off
httpd_can_network_connect_db --> on
httpd_can_network_memcache --> off
```

Finalmente, si nos vamos al navegador y recargamos, ya tenemos la pila LAMP configurada en nuestro servidor centOS.



LECCION 2 – GIT

*