Introducción:

Esta práctica ha sido ideada con el objetivo de poner en marcha un servidor linux, securizarlo, mantenerlo y tener levantada una api-rest en python.

Para ello el desarrollo comenzó intentando usar Azure, el servicio de Microsoft, pero terminé decantándome por AWS por parte de Amazon. En este se montó una máquina virtual de Amazon Linux, pero al intentar usar el comando apt saltaba un error que decía que no lo reconocía. Investigando aprendí que este sistema operativo no utiliza el gestor de paquetes de APT (Advanced Package Tool), este usa comandos "yum" y pensé en dejarlo así, pero más adelante dio problemas al instalar ciertos paquetes. Lo anterior se debía a que Amazon prioriza la estabilidad y por ello usa versiones anteriores de python, conociendo esto desistí y pasé a usar otra instancia con Ubuntu.

Seguridad:

Acceso:

Para la conexión mediante ssh usé una clave RSA (Rivest-Shamir-Adleman) que son un método popular para la encriptación y la firma digital. Además, dejé solo 3 puertos abiertos:

- El 22 para la conexión mediante ssh.
- El 80 para el http.
- El 443 para el https.

Desde la ubicación de la clave, mediante terminal de windows se puede acceder con:

```
ssh -i "accesoUbuntu.pem" ubuntu@ec2-16-171-144-75.eu-north-1.compute.amazonaws.com
```

Habiendo ejecutado previamente este comando para garantizar que la clave no se pueda ver públicamente:

```
chmod 400 "accesoUbuntu.pem"
```

Una vez dentro creo el usuario walter y le doy privilegios de sudo:

```
adduser walter
sudo usermod -aG sudo walter
su walter
```

Fail2ban:

Fail2ban es una herramienta de seguridad de código abierto diseñada para proteger los servidores contra ataques de fuerza bruta y otros tipos de actividad maliciosa en línea. Funciona monitoreando los registros de actividad del sistema en busca de patrones específicos que puedan indicar intentos de intrusión, como, en nuestro caso, múltiples intentos fallidos de inicio de sesión.

Con la ayuda de este tutorial procedí a la instalación:

```
sudo apt install python3 python3-pip
sudo apt install fail2ban
```

Y, seguidamente, a la configuración, donde solo tenía que modificar el nombre del archivo fail.conf a fail.local y dentro de este establecer los parámetros deseados. En mi caso establecí un

baneo de 10min para aquel que en un intervalo de 10min fallara 5 veces en su intento de inicio de sesión.

Después de lo anterior ponerlo a andar es tan fácil como:

```
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
sudo systemctl status fail2ban
```

Backup y eliminación de los ficheros logs residuales:

Sirviéndome de este [tutorial] (https://medium.com/@sudheer.barakers/linux-shell-script-to-backup-files-96d2127e7b54) realicé un script que hace backup a todo home y elimina los ficheros de log más antiguos de 7 días

```
#!/bin/bash

# Directorios a respaldar y directorio de copias de seguridad:
dGuardar="/etc /var/lib /home/*"
dBackup= "/home/dBackup"
# Copia de seguridad:
fecha=$(date +"%Y-%m-%d")
archivoFecha= "$dBackup/backup_$fecha.tar.gz"
tar -czvf "$archivoFecha" $dGuardar

# Eliminar ficheros de log más antiguos de 7 días:
find / ruta/a/logs -type f -name "*.log" -mtime +7 -exec rm {} \;

# Mensaje a mostrar:
echo "Copia de seguridad realizada en $archivoFecha"
```

```
echo "Se han eliminado los ficheros de log con una
antiguedad de más de 7"
```

Luego se añade a crontab como se muestra a continuación y así el script se ejecuta cada día a las 2am, mismo horario que otros servicios como Whatsapp:

```
0 2 * * * /home/walter/scriptShell.sh >
/home/directorio_backup/log_backup.log 2>&1
```

Api:

Una vez montado el servicio y asegurado su seguridad ante los tipos de ataques más comunes pasamos a crear la api con python. Viendo que el paquete recomendado, bottle, daba problemas a mis compañeros decidí investigar otro y terminé con flask psutil. El código que implementa los recursos de /hi (saludo y fecha) y /status (servicios en acción) queda de la siguiente manera:

```
from flask import Flask, jsonify
import psutil
from datetime import datetime

app = Flask(__name__)

@app.route('/hi')
def say_hello():
    now = datetime.now()
    current_time = now.strftime("%d/%m/%Y %H:%M:%S")
    return f'Hola, hoy es {current_time}'

@app.route('/status')
def system_status():
    running_services = []
```

```
    for proc in psutil.process_iter(['pid', 'name']):
        running_services.append(proc.info)
    return jsonify(running_services)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=(8080)
```

Luego pasé a darle permisos de ejecución y redirigir el tráfico del puerto 8080 al 80:

```
sudo iptables -A INPUT -p tcp --match tcp --dport 8080
-j ACCEPT
sudo iptables -t nat -A PREROUTING -p tcp --dport
8080 -j REDIRECT --to-port 80
sudo mkdir -p iptables/
sudo iptables-saves > rules.v4
```

Y para probarlo solo hay que hacer:

```
curl http://localhost:8080/hi # o /status
```

Supervisor:

Para asegurar que la api no falle implementé Supervisor: una herramienta de gestión de procesos en sistemas operativos Unix-like, diseñada para monitorear y controlar los procesos de manera automática; asegurando que estén en ejecución y reiniciándolos automáticamente si fallan.

Siguiendo este recurso:

```
sudo apt-get install supervisor
```

Y se crea un archivo de configuración, api.conf, dentro de este:

```
[program:my_api]

command=/usr/bin/python3 /home/walter/api.py
directory=/home/walter/

autostart=true

autorestart=true

stderr_logfile=/var/log/api.err.log

stdout_logfile=/var/log/api.out.log
```

Después de ello es necesario actualizar la configuración del mismo:

```
sudo supervisorctl reread
sudo supervisorctl update
sudo supervisorctl start my_api
```

Nginx:

Nginx (pronunciado "engine-x") es un servidor web de código abierto, así como un servidor proxy inverso y un balanceador de carga. Un proxy inverso es un servidor que actúa como intermediario entre los clientes que solicitan recursos y los servidores que proporcionan esos recursos. A diferencia de un proxy normal, que se coloca entre un cliente y un servidor de destino para acceder a recursos en nombre del cliente, un proxy inverso se coloca entre los clientes y los servidores para proteger, mejorar y controlar el acceso a los recursos del servidor.

Con esta ayuda se instala:

```
sudo apt-get install nginx
```

Y al igual que los anteriores creamos el archivo de configuración:

```
server {
    listen 80;
    ssoo2 16.171.144.75;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
    }
}
```

Tras esto solo se requiere habilitar el sitio y reiniciar Nginx:

```
sudo ln -s /etc/nginx/sites-available/api.conf
/etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

Conclusiones:

En conclusión, se implementaron medidas de seguridad, como el acceso mediante claves RSA y la configuración de puertos específicos para SSH, HTTP y HTTPS, junto con herramientas como Fail2ban para proteger contra ataques de fuerza bruta.

Además, se estableció un sistema de respaldo automatizado y limpieza de archivos de registro para garantizar la integridad y el

rendimiento del servidor a lo largo del tiempo. La creación de una API REST en Python utilizando Flask y psutil permitió exponer funcionalidades importantes del sistema de forma segura.

Para garantizar la disponibilidad continua de la API, se implementaron herramientas como Supervisor para supervisar y reiniciar automáticamente el servicio en caso de fallo, y Nginx como servidor proxy inverso y balanceador de carga, mejorando así la escalabilidad y la seguridad de la infraestructura.