

MONITORIZACIÓN DE SERVIDORES WEB EN ENTORNO CLOUD

Proyecto Gestión y Administración de Redes

Github del proyecto



Álvaro Colmenares Gil

Javier Díaz Fuentes

Pablo Domínguez Alonso

David Hernández Puerta

Diego Sanz Martín

Contenido

Definición de red de gestión mediante SDN (Álvaro Colmenares).....	2
Introducción.....	2
Creación de una entidad de certificación	2
Creación del certificado del servidor, clave, archivos encriptados e instalación de OpenVPN.....	4
Generar certificados de clientes y par de claves.	5
Configuración del servicio OpenVPN	5
Estableciendo la configuración de red del servidor.....	5
Apertura de puertos en AWS.....	6
Asignación de ip elástica (estática) en AWS	7
Iniciar OpenVPN.....	7
Crear la infraestructura de configuración de clientes	7
Generar las configuraciones de clientes.....	8
Monitorización y gestión de alarmas mediante AWS.....	8
Despliegue de servicios en los sistemas mediante Ansible (Diego Sanz)	9
Creación de un playbook.yml	9
Monitorización mediante Grafana, Influxdb y Telegraf	11
Monitorización mediante Google Cloud.....	17
Servidores web y monitorización de logs (Javier Díaz)	20
Estructura de red	20
Balanceador de carga.....	21
Monitorización con telegraf.....	25
Integración servicio SNMP (Pablo Domínguez)	28
Instalación SNMP	28
Configuración SNMP	29
Desarrollo del script para el Balanceador de Carga.....	30
Seguridad en la infraestructura (David Hernández)	35
Redireccionamiento SSH.....	40
Anexo: Código implementado para el redireccionamiento SSH	43
Conclusiones finales.....	45
Archivos del proyecto	45

Definición de red de gestión mediante SDN (Álvaro Colmenares)

Introducción

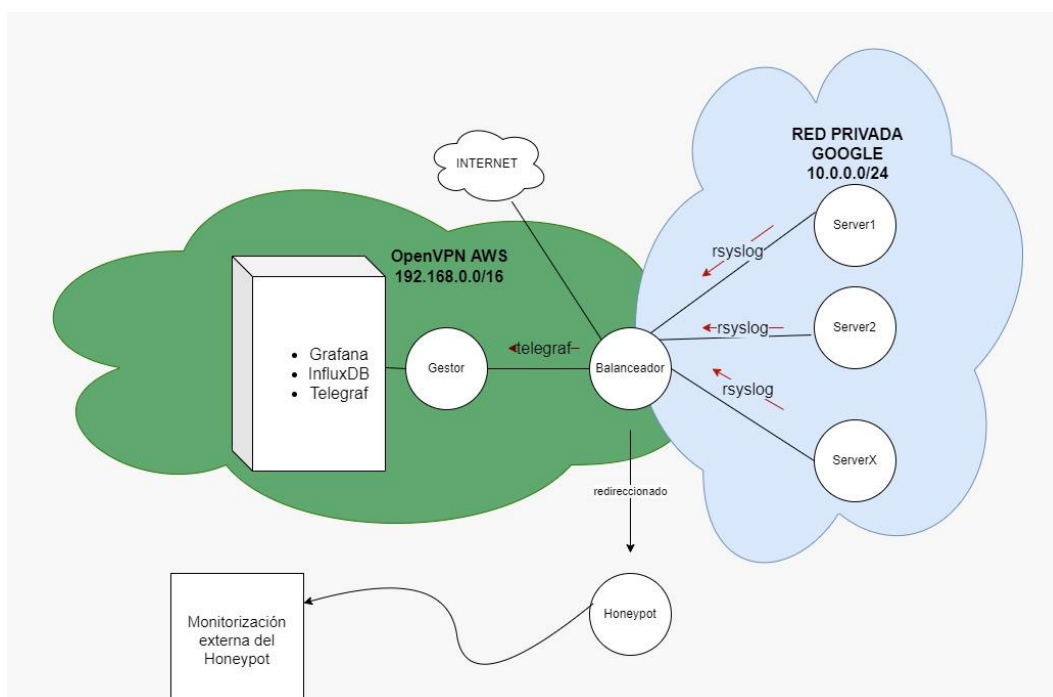
En nuestro planteamiento inicial, necesitamos generar una red de mantenimiento y supervisión para poder interconectar todas las máquinas, de manera que sea una red escalable, económica, con alto grado de disponibilidad y compatibilidad con gran cantidad de tecnologías.

Por ello decidimos utilizar una aproximación SDN, en este caso, utilizamos los servicios de Cloud Computing de Amazon Web Services.

Mediante el uso de una instancia Ubuntu de bajo coste crearemos una red VPN que servirá de red de gestión y monitorización a través del software OpenVPN el cuál nos permitirá tener conectividad entre múltiples tecnologías de forma segura y estable a un precio reducido.

De esta forma podemos crear una red de monitorización escalable con independencia de la ubicación de los sistemas a monitorizar (mientras tengan conexión a internet).

En nuestro caso para demostrar la fiabilidad del planteamiento utilizamos dos servicios, AWS para hospedar el servidor OpenVPN y Google Cloud para hospedar los sistemas que formarán la parte interna de la red de monitorización.



Creación de una entidad de certificación

El primer paso que realizar será realizar la instalación de easy-rsa, una herramienta de gestión de entidades de certificación que se utilizará para tratar de "securizar" la vpn

mediante el uso de una clave privada y un certificado root público, que se utilizará para firmar las solicitudes de comunicación entre los clientes y el servidor. Para ello utilizaremos los siguientes comandos:

```
wget -P ~/ https://github.com/OpenVPN/easy-rsa/releases/download/v3.0.8/EasyRSA-3.0.8.tgz
tar xvf EasyRSA-3.0.8.tgz
```

El siguiente paso será entrar al directorio que se ha creado tras la descompresión y definir los campos para la creación de nuevos certificados.

```
cd ~/EasyRSA-3.0.8/
```

Los definimos de la siguiente forma (los datos físicos pertenecientes a la entidad de certificación):

```
set_var EASYRSA_REQ_COUNTRY    "ES"
set_var EASYRSA_REQ_PROVINCE   "Comunidad de Madrid"
set_var EASYRSA_REQ_CITY       "Alcalá de Henares"
set_var EASYRSA_REQ_ORG        "Proyecto GAR"
set_var EASYRSA_REQ_EMAIL      "proyecto.gar.2122@gmail.com"
set_var EASYRSA_REQ_OU         "Community"
```

Una vez terminada la definición procedemos a inicializar la infraestructura de clave pública de la entidad de certificación mediante:

```
./easyrsa init-pki
```

Que nos da la siguiente salida:

```
ubuntu@gestor:~/EasyRSA-3.0.8$ ./easyrsa init-pki
Note: using Easy-RSA configuration from: /home/ubuntu/EasyRSA-3.0.8/vars
init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /home/ubuntu/EasyRSA-3.0.8/pki
```

Después utilizamos el siguiente comando para procesar las certificaciones y crear la entidad de certificación en el gestor:

```
./easyrsa build-ca nopass
```

Que nos da la siguiente salida:

```
ubuntu@gestor:~/EasyRSA-3.0.8$ ./easyrsa build-ca nopass
Note: using Easy-RSA configuration from: /home/ubuntu/EasyRSA-3.0.8/vars
Using SSL: openssl OpenSSL 1.1.1 11 Sep 2018
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:gar
CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/home/ubuntu/EasyRSA-3.0.8/pki/ca.crt
```

Creación del certificado del servidor, clave, archivos encriptados e instalación de OpenVPN

En esta parte, crearemos una petición de certificado desde la máquina donde vamos a crear el servidor OpenVPN, en este caso, al no ser un escenario real, lo realizaremos todo desde la misma máquina, algo no recomendable desde el ámbito de la ciberseguridad ya que si está máquina es atacada se comprometen dos servicios que deberían estar separados.

El primer paso será crear la petición de certificado:

```
./easysrsa gen-req server nopass
```

Nos da la siguiente salida:

```
ubuntu@gestor:~/EasyRSA-3.0.8$ ./easysrsa gen-req server nopass
Note: using Easy-RSA configuration from: /home/ubuntu/EasyRSA-3.0.8/vars
Using SSL: openssl OpenSSL 1.1.1 11 Sep 2018
Generating a RSA private key
.....+++++
.....+++++
writing new private key to '/home/ubuntu/EasyRSA-3.0.8/pki/easy-rsa-5728.5g7I4w/tmp.yX1ijw'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [server]:gar

Keypair and certificate request completed. Your files are:
req: /home/ubuntu/EasyRSA-3.0.8/pki/reqs/server.req
key: /home/ubuntu/EasyRSA-3.0.8/pki/private/server.key
```

Después, firmamos la petición con la entidad de certificación:

```
./easysrsa sign-req server server
```

Con la consiguiente salida:

```
Certificate is to be certified until Apr 26 09:56:43 2024 GMT (825 days)
Write out database with 1 new entries
Data Base Updated
Certificate created at: /home/ubuntu/EasyRSA-3.0.8/pki/issued/server.crt
```

Ahora realizaremos la instalación de OpenVPN como en cualquier otra distribución basada en Debian:

```
sudo apt install openvpn
```

Ahora copiamos los ficheros .req .crt a la carpeta de gestión de OpenVPN, ya que serán:

```
sudo cp pki/reqs/server.req pki/ca.crt /etc/openvpn/
```

Tras este paso, crearemos una clave Diffie-Hellman para usar durante el intercambio de claves:

```
./easysrsa gen-dh
```

Tras un par de minutos, nos generará una salida así:

```
.....+
.....+*****+
DH parameters of size 2048 created at /home/ubuntu/EasyRSA-3.0.8/pki/dh.pem
```

Cuando el proceso anterior finaliza, generamos una firma HMAC para fortalecer la verificación de integridad TLS del servidor:

```
openvpn --genkey --secret ta.key
```

Cuando estos procesos terminan copiamos los ficheros ta.key y dh.pem generados al directorio de OpenVPN:

```
sudo cp ta.key pki/dh.pem /etc/openvpn/
```

[Generar certificados de clientes y par de claves.](#)

El primer paso será crear un árbol de directorios para organizar las claves:

```
mkdir -p ~/client-configs/keys  
chmod -R 700 ~/client-configs
```

Ahora procederemos a crear el par de claves para los clientes, en este caso el manager (se realizan todos de manera análoga):

```
./easysrsa gen-req manager nopass
```

[Configuración del servicio OpenVPN](#)

Extraemos la carpeta de ejemplo de configuración por defecto para realizar nuestra configuración:

```
sudo cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz  
/etc/openvpn/  
sudo gzip -d /etc/openvpn/server.conf.gz
```

Modificamos las siguientes líneas en el fichero /etc/openvpn/server.conf:

```
tls-auth ta.key 0 # This file is secret  
cipher AES-256-CBC  
auth SHA256  
dh dh.pem  
server 192.168.0.0  
user nobody  
group nogroup  
client-to-client
```

[Estableciendo la configuración de red del servidor](#)

El primer paso a realizar es habilitar ip forwarding en ipv4, para poder enrutar el tráfico que llegue hacia la VPN para ello modificamos el fichero /etc/sysctl.conf añadiendo:

```
net.ipv4.ip_forward=1
```

Para modificar los valores anteriores utilizamos el siguiente comando:

```
sudo sysctl -p
```

Que nos da la siguiente salida:

```
ubuntu@gestor:~/EasyRSA-3.0.8$ sudo sysctl -p  
net.ipv4.ip_forward = 1
```

El siguiente paso que realizar será la modificación del firewall ufw de Ubuntu para permitir el “forward de paquetes”, ya que necesitamos que esta máquina pueda reenviar paquetes, es decir, actuar como un “router”, para ello modificaremos en el fichero /etc/default/ufw la siguiente línea:

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

También abriremos el puerto 1194 UDP ya que es por donde se gestionará el tráfico de la VPN:

```
sudo ufw allow 1194/udp
```

Después reiniciaremos el firewall:

```
sudo ufw disable
```

```
sudo ufw enable
```

El siguiente paso es averiguar que interfaz utilizamos para dirigir el tráfico a internet, para ello utilizamos el siguiente comando:

```
ip route list default
```

Que nos da la siguiente salida:

```
ubuntu@ip-172-31-19-230:~$ ip route list default
default via 172.31.16.1 dev eth0 proto dhcp src 172.31.19.230 metric 100
```

Entonces podemos deducir que tenemos que usar la interfaz *eth0*, y modificamos el fichero */etc/ufw/before.rules* para establecer unas reglas antes de que cargue el firewall:

```
# START OPENVPN RULES
# NAT table rules
*nat
:POSTROUTING ACCEPT [0:0]
# Allow traffic from OpenVPN client to eth0 (change to the interface you
discovered!)
-A POSTROUTING -s 192.168.0.0/16 -o eth0 -j MASQUERADE
COMMIT
# END OPENVPN RULES
```

Apertura de puertos en AWS

Para que el servicio funcione correctamente, es necesaria la apertura del puerto 1194/UDP, para ello modificamos el grupo de seguridad perteneciente a nuestra instancia (servidor virtual) de esta forma (el puerto 22/TCP corresponde a SSH):

▼ Reglas de entrada				
Q Filtrar reglas				
ID de la regla del grupo ...	Intervalo de p...	Protocolo	Origen	Grupos de seguridad
sgr-01bc50abb96d05012	1194	UDP	0.0.0.0/0	launch-wizard-1
sgr-02ca0205bd4da9c6c	22	TCP	0.0.0.0/0	launch-wizard-1
▼ Reglas de salida				
Q Filtrar reglas				
ID de la regla del grupo ...	Intervalo de p...	Protocolo	Destino	Grupos de seguridad
sgr-062720029a876f161	Todo	Todo	0.0.0.0/0	launch-wizard-1

Asignación de ip elástica (estática) en AWS

Para que el servicio funcione correctamente es necesario que la ip del servicio no sea dinámica, es decir, que no pueda cambiar y pueda ser encontrada siempre por los clientes, para ello activamos el servicio en la consola de AWS, queda de la siguiente forma:

Dirección IPv4 asignada 52.47.61.47	Tipo IP pública	ID de asignación eipalloc-063a1ecab685ce6ec	Registro DNS inverso -
ID de asociación eipassoc-0e7a2b85b97b42d79	Ámbito VPC	ID de la instancia asociada i-0c09aa62d85ad1bca	Dirección IP privada 172.31.19.230
ID de la interfaz de red eni-0a7a2d6b333164b46	ID de la cuenta del propietario de la interfaz de red 751069267100	DNS público ec2-52-47-61-47.eu-west-3.compute.amazonaws.com	ID de la gateway NAT -

Iniciar OpenVPN

El siguiente paso es hacer que el servicio OpenVPN arranque al inicio del sistema, para ello utilizamos el siguiente comando:

```
sudo systemctl -f enable openvpn-server@server.service
```

Ahora iniciamos el servicio OpenVPN:

```
sudo service openvpn-server@server start
```

Por último, comprobamos el estado del servicio una vez lanzado:

```
sudo service openvpn-server@server status
```

```
ubuntu@ip-172-31-19-230:~$ sudo service openvpn-server@server status
● openvpn-server@server.service - OpenVPN service for server
   Loaded: loaded (/lib/systemd/system/openvpn-server@.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-01-22 13:33:19 UTC; 40min ago
     Docs: man:openvpn(8)
           https://community.openvpn.net/openvpn/wiki/Openvpn24ManPage
           https://community.openvpn.net/openvpn/wiki/HOWTO
   Main PID: 504 (openvpn)
   Status: "Initialization Sequence Completed"
     Tasks: 1 (limit: 1147)
    Memory: 2.6M
   CGroup: /system.slice/system-openvpn\x2dservers.slice/openvpn-server@server.service
           └─504 /usr/sbin/openvpn --status /run/openvpn-server/status-server.log --status-version 2 --suppress-timestamps --config server.conf

Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: Could not determine IPv4/IPv6 protocol. Using AF_INET
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: Socket Buffers: R=[212992->212992] S=[212992->212992]
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: UDPv4 link local (bound): [AF_INET][undef]:1194
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: UDPv4 link remote: [AF_INET][undef]:1194
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: GID set to nogroup
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: UID set to nobody
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: MULTI: multi_init called, r=256 v=256
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: IFCONFIG POOL: base=192.168.0.4 size=16382, ipv6=0
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: IFCONFIG POOL LIST
Jan 22 13:33:19 ip-172-31-19-230 openvpn[504]: Initialization Sequence Completed
```

Crear la infraestructura de configuración de clientes

El primer paso será crear un fichero de configuración para clientes en ~/client-configs/base.conf para establecer los parámetros con los que deben conectarse a nuestra VPN, las configuraciones que realizaremos serán las siguientes:

```
Remote 52.47.61.47
Proto udp
User nobody
Group nogroup
ca ca.crt
cert client.crt
key client.key
tls-auth ta.key 1
cipher AES-256-GCM
auth SHA256
key-direction 1
```


A continuación, realizaremos un script para generar los ficheros. ovpn que utilizarán los clientes para conectarse a la VPN:

```
#!/bin/bash

# First argument: Client identifier

KEY_DIR=~/.client-configs/keys
OUTPUT_DIR=~/.client-configs/files
BASE_CONFIG=~/.client-configs/base.conf

cat ${BASE_CONFIG} \
  <(echo -e '<ca>') \
  ${KEY_DIR}/ca.crt \
  <(echo -e '</ca>\n<cert>') \
  ${KEY_DIR}/${1}.crt \
  <(echo -e '</cert>\n<key>') \
  ${KEY_DIR}/${1}.key \
  <(echo -e '</key>\n<tls-crypt>') \
  ${KEY_DIR}/ta.key \
  <(echo -e '</tls-crypt>') \
  > ${OUTPUT_DIR}/${1}.ovpn
```

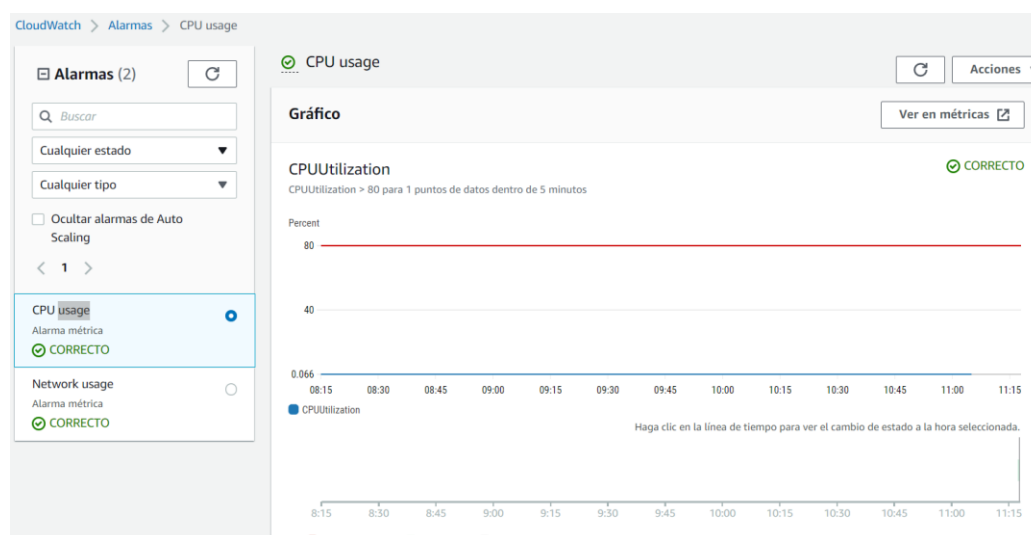
Generar las configuraciones de clientes

Para generar las configuraciones de clientes simplemente ponemos los ficheros .crt y .key junto al script anterior y este nos generará el fichero de configuración.

Monitorización y gestión de alarmas mediante AWS

Por último, al usar un servicio cloud como AWS, este nos proporciona un gran número de herramientas para la gestión de los sistemas y las alarmas.

Configuraremos una serie de métricas con alarmas establecidas (con notificación por correo electrónico) para poder gestionar la posible actividad inusual en la red, ya sea por ataques informáticos o caída del servicio.



Alarmas (2) <input type="checkbox"/> Ocultar alarmas de Auto Scaling <input type="button" value="Borrar selección"/> <input type="button" value="Actualizar"/> <input type="button" value="Crear alarma compuesta"/> <input type="button" value="Acción"/>					
<input type="text" value="Buscar"/>				Cualquier es...	Cualquier tipo
<input type="checkbox"/>	Nombre	Estado	Última actualización del estado	Condiciones	
<input type="checkbox"/>	CPU usage	✔ CORRECTO	2022-01-23 12:17:26	CPUUtilization > 80 para 1 puntos de datos dentro de 5 minutos	
<input type="checkbox"/>	Network usage	✔ CORRECTO	2022-01-23 12:08:45	NetworkIn > la banda (ancho: 2) para 1 puntos de datos dentro de 5 minutos	

Despliegue de servicios en los sistemas mediante Ansible (Diego Sanz)

Creación de un playbook.yml

Para poder realizar un reescalado de los servicios, hemos decidido realizar una instalación con Ansible.

En el archivo de configuración de los hosts de Ansible, “/etc/ansible/hosts”, definimos cada host en distintos grupos, de forma que si queremos realizar una instalación en un nuevo servidor, únicamente deberemos añadir su dirección IP a su grupo y ejecutar el playbook. En dicho playbook tenemos dos partes:

```
diego@manager: ~/project
[all:vars]
ansible_python_interpreter=/usr/bin/python3

[gestor]
192.168.0.4

[loadbalancer]
192.168.0.6
```

Gestor

En la que realizaremos la instalación de telegraf y copiaremos el archivo de configuración “telegraf.conf” en la ruta predeterminada de dicho archivo.

Además instalaremos también docker, necesario para poder descargar los contenedores con las utilidades de grafana e influxdb, y docker-compose, con el que orquestaremos el enlace de dichos componentes.

```
diego@manager: ~/project
---
- hosts: gestor
  become: yes
  become_method: sudo

  tasks:
    #Instalación de telegraf
    - include: tasks/telegraf.yml
    - copy:
        src: gestor/telegraf.conf
        dest: /etc/telegraf/telegraf.conf
    #Instalación de docker-grafana-influxdb
    - include: tasks/docker.yml
    - copy:
        src: gestor/docker-compose.yml
        dest: /home/ubuntu/docker-compose.yml
```

La instalación de estos componentes se realiza mediante el uso de archivos “.yaml” en el directorio tasks.

El archivo “telegraf.conf” contiene las direcciones del repositorio Telegraf y la clave necesaria para la descarga de los binarios. Mediante el uso de la variable “ansible_distribution_release” se elegirá el paquete correspondiente a la distribución de nuestros hosts. Finalmente se procede con la instalación de telegraf.

```

- name: Telegraf repository key
  apt_key:
    url: https://repos.influxdata.com/influxdb.key
    keyring: /etc/apt/trusted.gpg.d/telegraf.gpg
    state: present

- name: Telegraf repository
  apt_repository:
    repo: "deb https://repos.influxdata.com/ubuntu {{ ansible_distribution_release }} stable"
    state: present
    filename: telegraf
    register: repo_telegraf

- name: Update cache
  apt:
    update_cache: yes
    when: repo_telegraf.changed

- name: Telegraf packages
  apt:
    name:
      - telegraf
    state: present
    register: install_telegraf

```

En el archivo “docker.yml” se procederá del mismo modo, obtendremos los paquetes necesarios para la instalación y se instalará el binario. Además realizaremos una instalación de docker-compose utilizando un paquete alojado en github.

```

- name: ensure repository key is installed
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: ensure docker registry is available
  apt_repository: repo='deb https://download.docker.com/linux/ubuntu bionic stable' state=present

- name: ensure docker and dependencies are installed
  apt: name=docker-ce update_cache=yes

- service: name=docker state=restarted

- name: Install Docker Compose
  get_url:
    url: "https://github.com/docker/compose/releases/download/1.28.2/docker-compose-Linux-x86_64"
    dest: /usr/local/bin/docker-compose
    mode: u+x,g+X,o+X

```

Balanceador de carga

En esta máquina también instalaremos telegraf y copiaremos su archivo de configuración (distinto al anterior).

Instalaremos docker para poder utilizar un contenedor específico que junto con la instalación del binario swatchdog formarán el balanceador de carga.

```

- hosts: loadbalancer
  become: yes
  become_method: sudo

  tasks:
    #Instalación de telegraf
    - include: tasks/telegraf.yml
    - copy:
        src: loadbalancer/telegraf.conf
        dest: /etc/telegraf/telegraf.conf

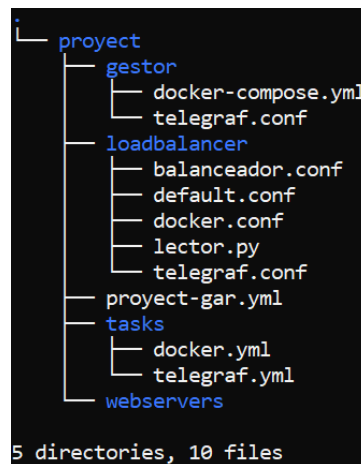
    #Instalación de docker-grafana-influxdb
    - include: tasks/docker.yml

    #Instalación de Swatch
    - command: sudo apt update
    - command: sudo apt install -y swatch

```

Para que los binarios que instalamos anteriormente funcionen, será necesario realizar una copia de los ficheros de configuración en las rutas especificadas.

La estructura necesaria para que la ejecución de “proyect-gar.yml” sea satisfactoria es:



Monitorización mediante Grafana, Influxdb y Telegraf

Generaremos un archivo docker-compose desde el que ejecutaremos Grafana e Influxdb, para ello definiremos dos servicios con dos imágenes influxdb y grafana-enterprise, obtenidas de docker.hub.

Cada servicio dispondrá de unos puertos, los cuales serán expuestos en la máquina mediante la configuración “ports”, de forma que la página GUI de Influxdb y Grafana se consultará desde la dirección IP del gestor y los puertos “8086” y “80” respectivamente.

Para poder visualizar en el navegador el GUI mediante la IP externa del gestor, será necesario activar el enrutamiento HTTP de la instancia de Google y crear una nueva regla en el firewall de la red VPC, para que el puerto 8086 sea accesible desde Internet.

```

version: '2.1'

services:
  influxdb:
    image: influxdb
    container_name: influxdb
    ports:
      - "8083:8083"
      - "8086:8086"
      - "8090:8090"

  grafana:
    image: grafana/grafana-enterprise
    container_name: grafana
    ports:
      - "80:3000"
    user: "0"
    links:
      - influxdb
        
```

Red de VPC

- Redes de VPC
- Direcciones IP externas
- Usa tu propia IP
- Firewall**
- Rutas
- Intercambio de tráfico entre ...
- VPC compartida
- Acceso a VPC sin servidores
- Duplicación de paquetes

Dirección
 Entrada

Acción en caso de coincidencia
 Permitir

Destinos

Etiquetas de destino
influxdb

Filtros de fuente

Rangos de IP
0.0.0.0/0

Protocolos y puertos
 tcp:8086

Aplicación
 Habilitada

Desde el host gestor ejecutaremos el comando “docker-compose up” en la misma ruta que el archivo anterior:

11

```

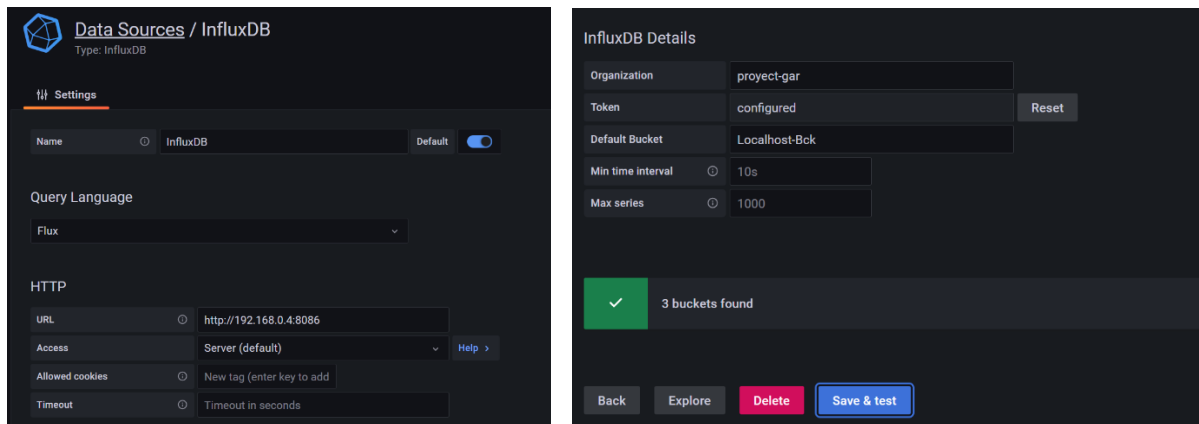
ubuntu@gestor:~$ sudo docker-compose up
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Pulling influxdb (influxdb:latest)...
latest: Pulling from library/influxdb
9b99af5931b3: Pull complete
b6013b3e77fe: Pull complete
bbced17b6899: Pull complete
d4e540b687b1: Pull complete
d80bad39181a: Pull complete
8f68ff56df29: Pull complete
f3136e30866a: Pull complete
cf057a88c2ba: Pull complete
cea3b1950251: Pull complete
40a6ad39ca1f: Pull complete
Digest: sha256:1a48c5c4b957b795cdf381bcf91e0d7de9dea2d9be984afbd6e4922e2e24484
Status: Downloaded newer image for influxdb:latest
Pulling grafana (grafana/grafana-enterprise:latest)...
latest: Pulling from grafana/grafana-enterprise
59bfc1c3509f3: Pull complete
6df6620c7b78: Pull complete
fe93f932613d: Pull complete
07017d3a9628: Pull complete
3f44d94b12c2: Pull complete
60414add1ef4: Pull complete
6aef916532ab: Pull complete
37637756541b: Pull complete
1a32aed80ab3: Pull complete
68e28e3dde1b: Pull complete
Digest: sha256:33e5a0b2264797f7b3083899a1c3a903710829495788271ec0fa5f56b8f8aba8
Status: Downloaded newer image for grafana/grafana-enterprise:latest
Creating influxdb ... done
Creating grafana ... done
Attaching to influxdb, grafana
influxdb | 2022-01-22T16:58:03.839357020Z    warn    boltadb not found at configured path, but DOCKER_INFLUXDB_INIT_MODE not s
"system": "docker", "bolt_path": ""}
grafana  | Grafana server is running with elevated privileges. This is not recommended
grafana  | t=2022-01-22T16:58:04+0000 lvl=warn msg="falling back to legacy setting of 'min_interval_seconds'; please use the
d alerting` section if Grafana 8 alerts are enabled." logger=settings

```

Configuración de Grafana e Influxdb

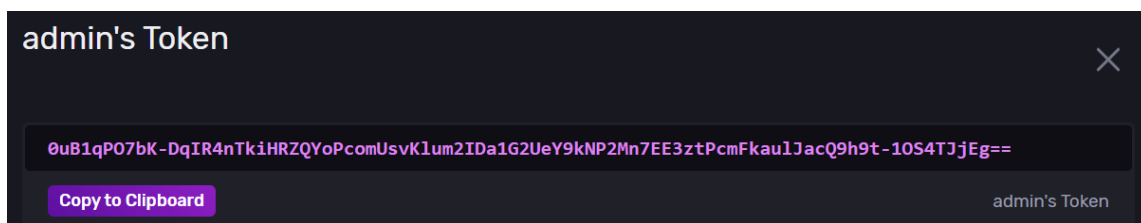
Si entramos en el GUI de Influxdb por primera vez tendremos que crear un usuario y, dado que la versión que hemos instalado es la “v2”, existirá una organización y un bucket asociados a dicho usuario. Dicho bucket corresponderá con una base de datos, en este caso la del servidor local.

Del mismo modo, la primera vez que entremos al GUI de Grafana tendremos que modificar la contraseña del administrador y ya podremos empezar a usar la aplicación. Una vez hayamos iniciado sesión deberemos crear un Data Source con la siguiente configuración:



El lenguaje Query utilizado será Flux, ya que nos permite securizar el sistema de monitorización mediante el uso de tokens.

La obtención de dicho token será desde el GUI de Influxdb Data>API Tokens



Configuración de Telegraf

Los archivos se generarán mediante el comando:

```
$ telegraf -sample-config -input-filter cpu:mem:disk -output-filter influxdb_v2 > telegraf.conf
```

Dicho archivo servirá tanto para configurar el Gestor como el Balanceador de carga, con la diferencia de que modificaremos el bucket al que se envían los datos “Localhost-Bck” y “LoadBalancer-Bck” respectivamente:

```
#####
#                               #
#                               #
#####

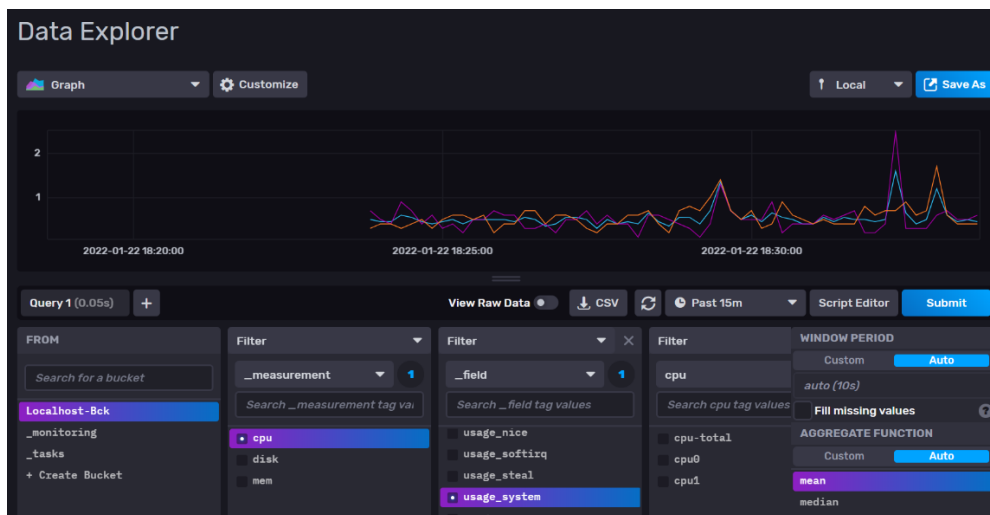
# Configuration for sending metrics to InfluxDB
[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ## ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["http://192.168.0.4:8086"]

  ## Token for authentication.
  token = "0uB1qP07bK-DqIR4nTkiHRZQYoPcomUsvKlum2IDa1G2UeY9kNP2Mn7EE3ztPcmFkaulJacQ9h9t-10S4TJjEg=="

  ## Organization is the name of the organization you wish to write to; must exist.
  organization = "proyect-gar"

  ## Destination bucket to write into.
  bucket = "Localhost-Bck"
```

Para poder visualizar los datos dispondremos tanto de la GUI de Influxdb como la de Grafana, de forma que generaremos un Query en el “Data Explorer” seleccionando los filtros que necesitamos:



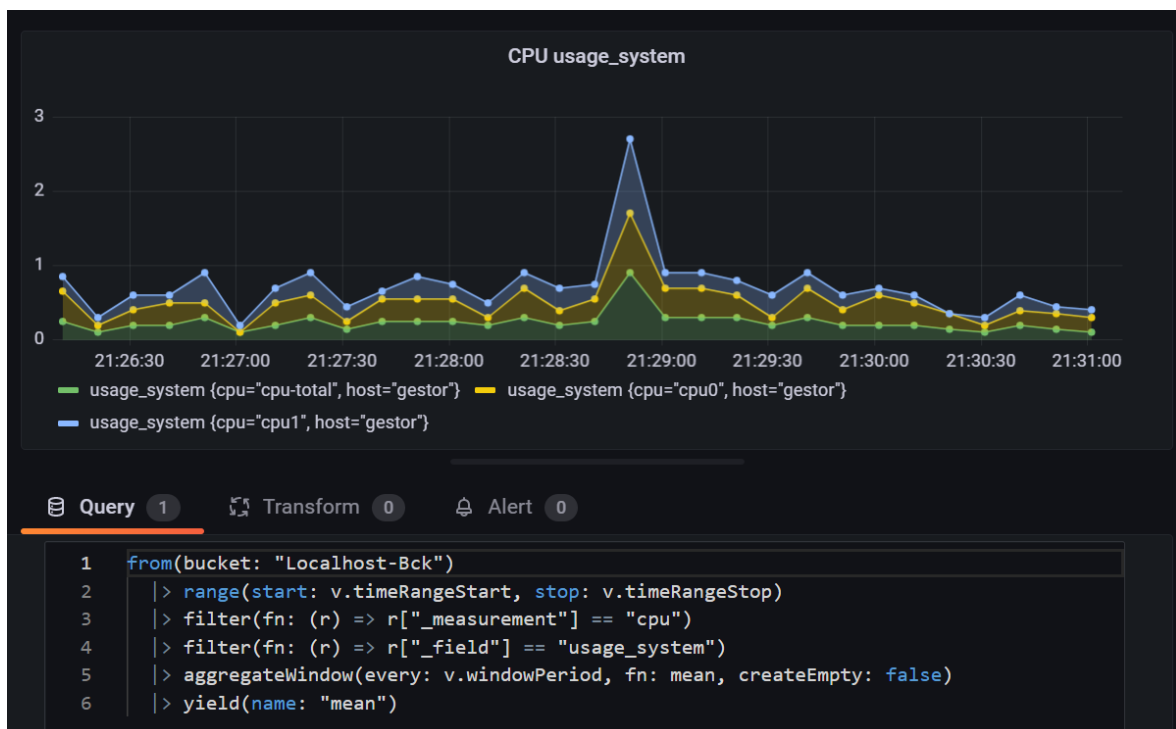
Para poder obtener el Query anterior en el lenguaje “Flux” bastará con seleccionar el Script Editor:

```

1 from(bucket: "Localhost-Bck")
2   |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3   |> filter(fn: (r) => r["_measurement"] == "cpu")
4   |> filter(fn: (r) => r["_field"] == "usage_system")
5   |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
6   |> yield(name: "mean")

```

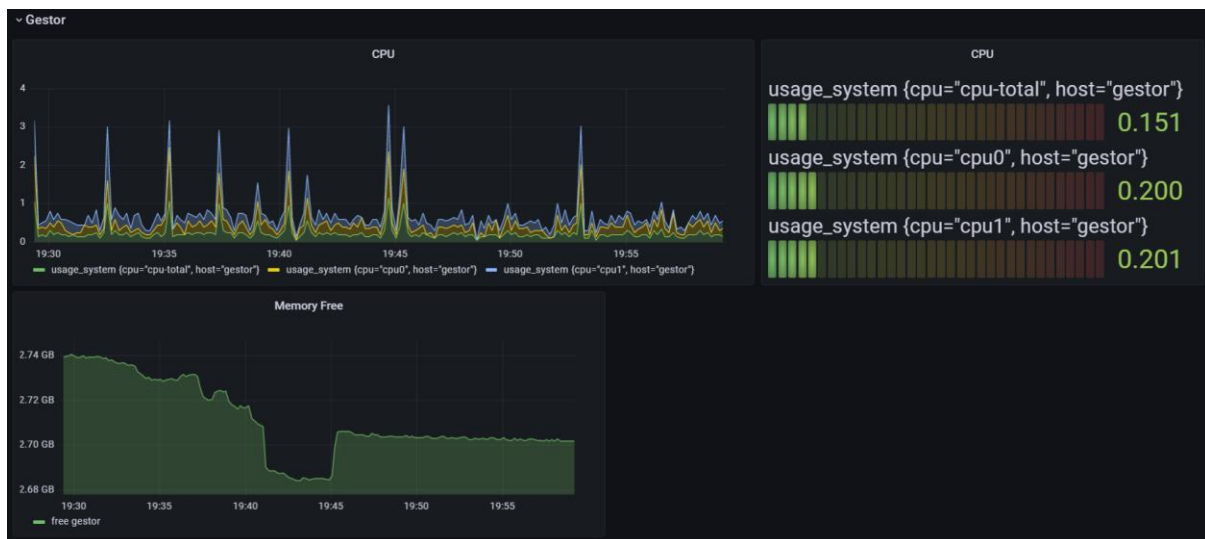
Para añadir la gráfica anterior a Grafana, copiaremos el código Flux generado en un nuevo panel del Dashboard:



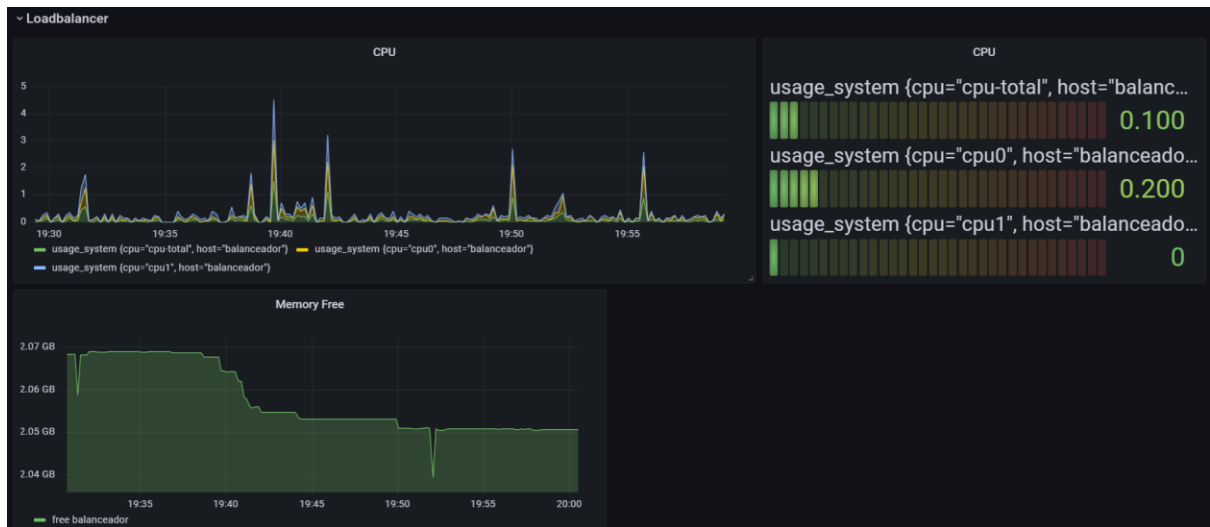
Procederemos de la misma forma para todas las métricas que queramos obtener de ambos servidores, p.ej. “Memory Free”:



El conjunto de métricas del host Local (Gestor) se representa conjuntamente en la dashboard de la siguiente forma:



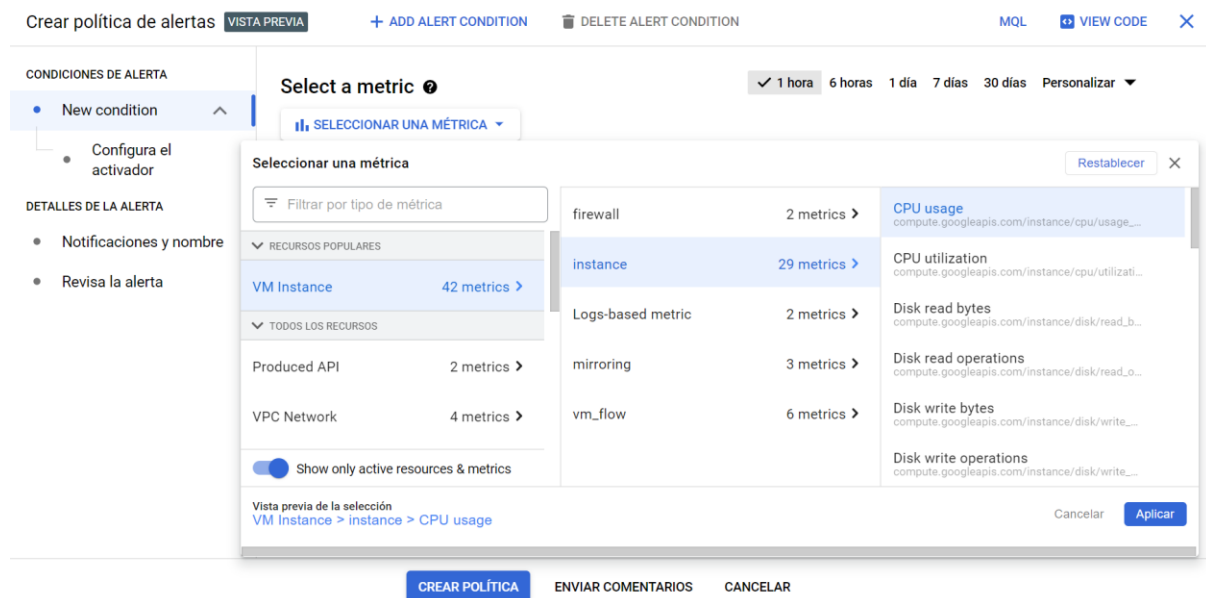
Igualmente crearemos una agrupación de paneles para el Balanceador de carga:



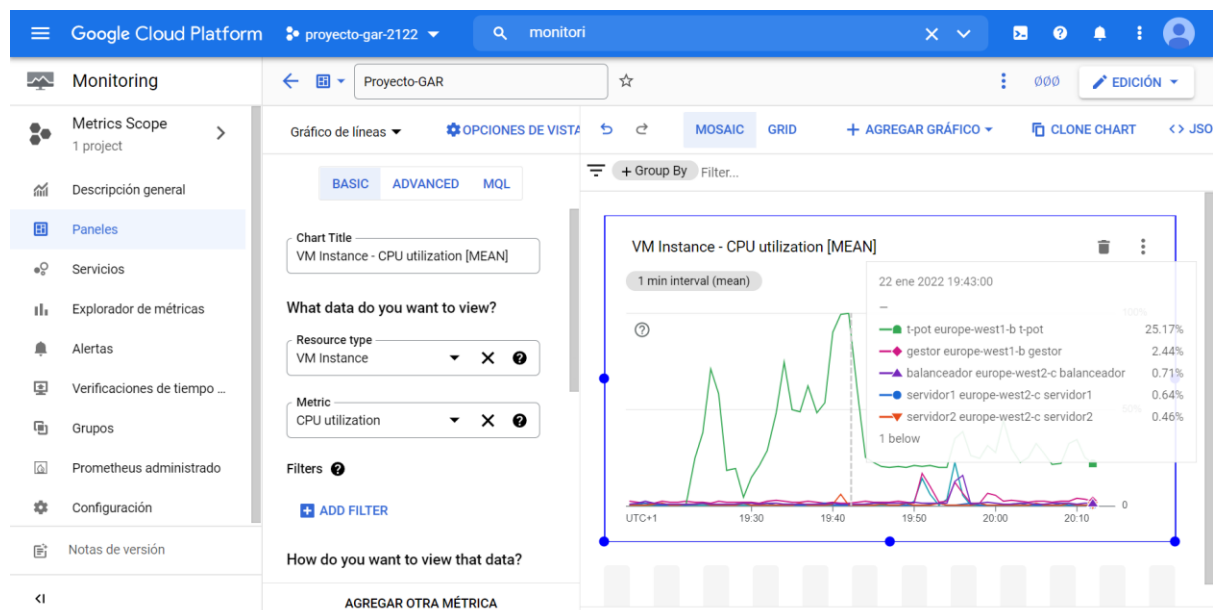
Monitorización mediante Google Cloud

Google Cloud nos provee de un sistema de monitorización asociado a todos los proyectos de nuestra cuenta, de forma que desde el apartado “Monitoring” podremos generar una dashboard similar a la generada anteriormente con Grafana.

Crearemos un nuevo panel en el que podremos seleccionar una métrica entre los distintos servicios que estén activos en nuestro proyecto.



En este caso representaremos sobre una gráfica lineal el uso de CPU de las instancias encendidas.



Aunque resulta mucho más útil la utilización de un “Alert Chart”, en el que seleccionaremos un umbral de 80% CPU-usage en nuestro caso.

Crear política de alertas **VISTA PREVIA** + ADD ALERT CONDITION DELETE ALERT CONDITION MQL VIEW CODE X

CONDICIONES DE ALERTA

VM Instance - CPU usage

Configura el activador

DETALLES DE LA ALERTA

- Notificaciones y nombre
- Revisa la alerta

Configure alert trigger

1 hora 6 horas 1 día 7 días 30 días Personalizar

Condition type

Threshold

Condition triggers if a time series rises above or falls below a value for a specific duration window

Metric absence

Condition triggers if any time series in the metric has no data for a specific duration window

Alert trigger

Cualquier serie temporal tiene incumplimientos

Posición del umbral

Por encima del umbral

Valor del umbral

80

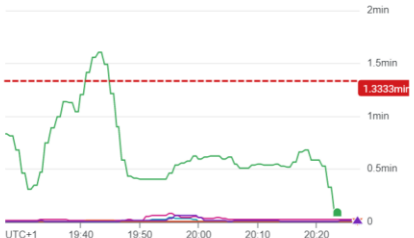
Período para volver a probar *

0 s

Se usa para volver a probar alertas. Expande el período si las alertas tienen una tendencia a los falsos positivos.

VM Instance - CPU usage

2 5 min interval (mean)



Name (fro...	zone	instan...	Valor
balanceador	europe...	balan...	0.014min
gestor	europe...	gestor	0.019min

CREAR POLÍTICA

ENVIAR COMENTARIOS

CANCELAR

Será necesario generar un nuevo canal de notificación, entre todas las opciones hemos optado por una alerta vía correo electrónico:

Edit Email Channel

Email addresses can be set to receive notifications from your alerting when a new incident is created.

Email Address *
proyecto.gar.2122@gmail.com

Display Name *
Email

Otra métrica que resulta interesante observar es el número de paquetes perdidos en el firewall de nuestros hosts.

Crear política de alertas **VISTA PREVIA** + ADD ALERT CONDITION DELETE ALERT CONDITION MQL VIEW CODE X

CONDICIONES DE ALERTA

New condition

Configura el activador

DETALLES DE LA ALERTA

- Notificaciones y nombre
- Revisa la alerta

Select a metric

1 hora 6 horas 1 día 7 días 30 días Personalizar

SELECCIONAR UNA MÉTRICA

Seleccionar una métrica

Filtrar por tipo de métrica

RECURSOS POPULARES

VM Instance 42 metrics

TODOS LOS RECURSOS

Produced API 2 metrics

VPC Network 4 metrics

Show only active resources & metrics

firewall 2 metrics

instance 29 metrics

Logs-based metric 2 metrics

mirroring 3 metrics

vm_flow 6 metrics

Dropped bytes

compute.googleapis.com/firewall/dropped_byt...

Dropped packets

compute.googleapis.com/firewall/dropped_pac...

Vista previa de la selección

VM Instance > firewall > Dropped packets

Cancelar Aplicar

CREAR POLÍTICA

ENVIAR COMENTARIOS

CANCELAR

Seleccionamos un umbral del 40% para detectar cuando la red se empieza a congestionar

Editar política de alertas **VISTA PREVIA** + ADD ALERT CONDITION DELETE ALERT CONDITION MQL VIEW CODE

CONDICIONES DE ALERTA

VM Instance - Dropped packets

Configura el activador

DETALLES DE LA ALERTA

- Notificaciones y nombre
- Revisa la alerta

Configure alert trigger

1 hora 6 horas 1 día 7 días 30 días Personalizar

Condition type

Threshold

Condition triggers if a time series rises above or falls below a value for a specific duration window

Metric absence

Condition triggers if any time series in the metric has no data for a specific duration window

Alert trigger

Cualquier serie temporal tiene incumplimientos

Posición del umbral

Por encima del umbral

Valor del umbral

40

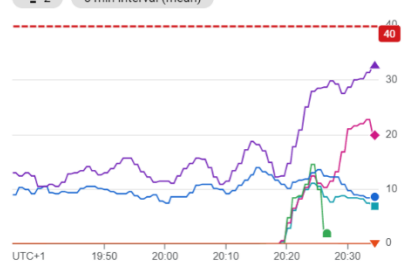
Periodo para volver a probar *

0 s

Se usa para volver a probar alertas. Expande el periodo si las alertas tienen una tendencia a los falsos positivos.

VM Instance - Dropped packets

2 5 min interval (mean)



Name (fro...	zone	instan...	Valor
balanceador	europe...	balan...	32.67
gestor	europe...	gestor	20

Tras realizar una prueba de estrés en un servidor web Google Cloud nos notificó que el servidor1 asociado a el “proyecto-gar” había superado un 80% del uso de CPU.

Alert firing

VM Instance - CPU utilization

CPU utilization for proyecto-gar-2122 servidor1 with metric labels {instance_name=servidor1} is above the threshold of 0.800 with a value of 0.958.

Summary

Start time

Jan 23, 2022 at 6:14PM UTC (less than 1 sec ago)

Project

proyecto-gar-2122

Policy

CPU-utilization

Condition

VM Instance - CPU utilization

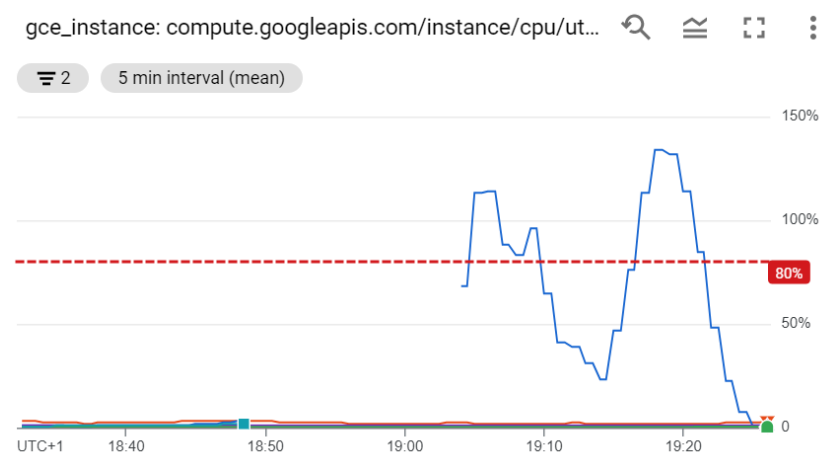
Metric

compute.googleapis.com/instance/cpu/utilization

Threshold

0.800

En el correo podemos acceder mediante un enlace a los detalles del incidente:



Servidores web y monitorización de logs (Javier Díaz)

Estructura de red

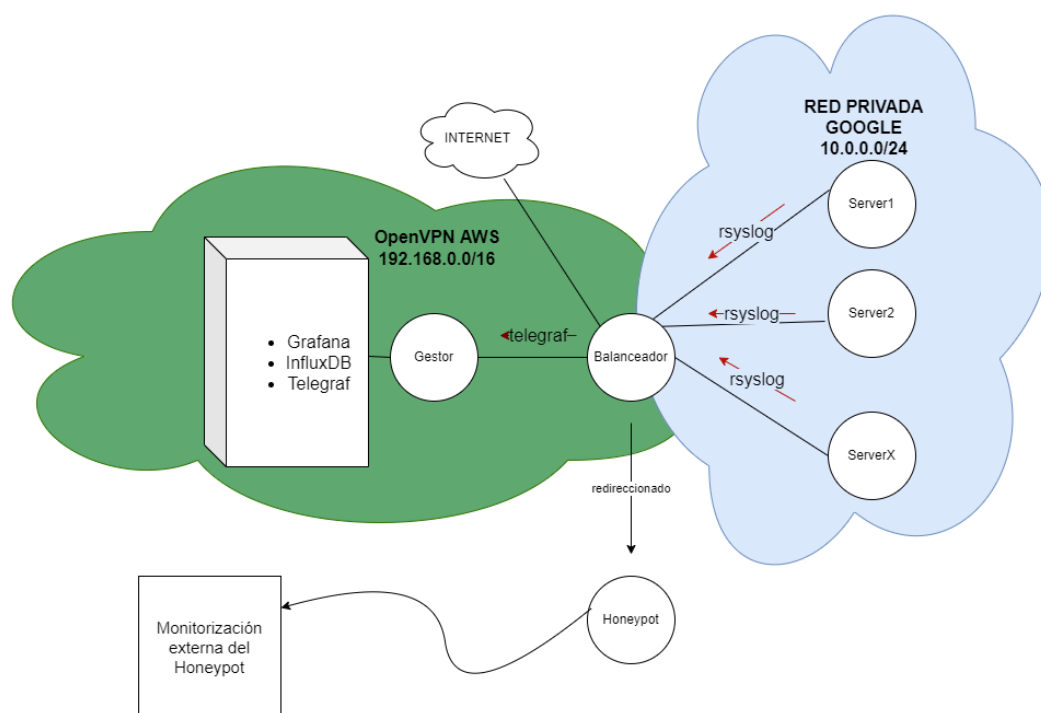


Ilustración 1.-Estructura de red

En esta parte vamos a comentar la estructura y gestión de la red de servidores que nos proporcionarán el servicio web correspondiente. Como podemos ver, estos servidores están aislados del resto de equipos de gestión, en una red privada definida en Google Cloud. El único punto de acceso que se tiene a estos equipos es a través del balanceador de carga, que además del reparto de carga, funciona de router NAT y recoge todos los logs a través del rsyslog.

Como no es el objetivo de esta práctica el proporcionar un servicio web espectacular, para simplificar la tarea, hemos utilizado de servicio web un servicio web basado en flask que muestra por pantalla el nombre del servidor que recibe la petición, junto con una hoja de estilos .css. El conjunto del servicio está implementado en un Docker accesible a todo el mundo a través de DockerHub, bajo el nombre de “javierdf/servidor_web”. De esta forma fomentamos la escalabilidad del servicio, pues las máquinas para proveer el servicio tan solo necesitan descargar el Docker y ejecutarlo, lo cual podemos hacer todo en un mismo comando:

```
docker run --log-driver syslog --log-opt syslog-address=udp://10.0.0.2:514 --log-opt tag=docker -p80:5000 -e HOSTNAME=$HOSTNAME -it javierdf/servidor_web
```

Como explicación del comando, estamos indicando que se ejecute el Docker “javierdf/servidor_web” (si no está descargado, lo descarga de DockerHub), y le añadimos las siguientes opciones para el correcto funcionamiento:

- --log-driver syslog: Indicamos al Docker que de sistema de logs use syslog

- `--log-opt`: Viene seguido de la opción de syslog que queremos configurar. En nuestro caso indicamos que el servidor que escucha los logs está en la dirección 10.0.0.2 (balanceador) en el puerto 514 y que utiliza udp. Y también indicamos que el tag de los logs sea “docker”
- `-p80:5000` : Vinculamos el puerto 80 de la máquina con el puerto 5000 del Docker, puerto en el que escucha Flask por defecto
- `-e HOSTNAME=$HOSTNAME`: Indica que usamos una variable de entorno que recoge el nombre del servidor.

Para la generación de los servidores en Google Cloud, utilizamos una imagen de un servidor previo con Docker ya instalado. Así, podemos automatizar la generación de nuevas máquinas con todo lo necesario para ofrecer el servicio mediante un comando de Google Cloud. En este comando podemos incluir también un comando de inicio, que, si ponemos el comando que carga el Docker del servidor web, podemos generar servidores nuevos y listos para ofrecer el servicio sin necesidad de tocar estos servidores, promoviendo así la escalabilidad del servicio. El comando para automatizar el despliegue de nuevos servidores sería el siguiente:

```
gcloud beta compute instances create servidor3 --project=proyecto-gar-2122 --zone=europe-west2-c --
machine-type=e2-medium --network-interface=subnet=red-servidores,no-address --metadata=startup-
script=sudo\ docker\ run\ --log-driver\ syslog\ --log-opt\ syslog-address=udp://10.0.0.2:514\ --log-opt\
tag=docker\ --rm\ -p80:5000\ -e\ HOSTNAME=$HOSTNAME\ -it\ javierdf/servidor_web --maintenance-
policy=MIGRATE --service-account=384234051972-compute@developer.gserviceaccount.com --
scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.googleapis.com/auth/logging.
write,https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/servicecontrol,h
ttps://www.googleapis.com/auth/service.management.readonly,https://www.googleapis.com/auth/trace.app
end --min-cpu-platform=Automatic --tags=http-server --no-shielded-secure-boot --shielded-vtpm --shielded-
integrity-monitoring --reservation-affinity=any --source-machine-image=servidores
```

Balanceador de carga

Como hemos dicho antes, el punto de acceso a los servidores web será a través del balanceador de carga, que hace de router NAT y resgistra los logs de los servidores.

El balanceador utilizado es un balanceador nginx. Por comodidad, hemos decidido utilizar el Docker de nginx, al que le montamos el archivo “default.conf”, donde se encuentran las directrices para el balanceo de carga.

```

upstream prueba {
    #least_conn;

    #SERVIDORES
    server 10.0.0.6;
    server 10.0.0.5;
}

server {
    listen 80;

    location / {
        proxy_pass http://prueba;
    }
    location /servidor1 {
        rewrite ^/servidor1(.*) / break;
        proxy_pass http://10.0.0.5;
    }
    location /servidor2 {
        rewrite ^/servidor2(.*) / break;
        proxy_pass http://10.0.0.6;
    }
}

```

Ilustración 2-Balanceador de carga: default.conf

Como vemos tenemos una primera sección “upstream prueba” donde definimos los servidores que van a estar proporcionando el servicio, y, por tanto, entre los que tenemos que repartir la carga. En este caso tenemos 2 servidores, 10.0.0.6 y 10.0.0.5. Por defecto, y es tal y como lo tenemos configurado, el sistema de balanceo de carga es un round robin puro, es decir, las peticiones se asignan de forma secuencial, según la disponibilidad de los servidores. Pero también tenemos otras opciones de balanceo, como son “least_conn” que le asigna la petición a aquel que tiene menos carga; o también podemos tener “ip_hash” que a cada petición se asigna a un servidor específico en base al hash de la ip origen, lo que permite que todas las peticiones de un usuario vayan a un mismo servidor. En la segunda sección “server” definimos el balanceador de carga. Así decimos que éste escucha en el puerto 80, y si recibimos una petición a este servidor, al path raíz ‘/’ entonces lo redirigimos a uno de los servidores definidos en “upstream pruebas”. Los otros dos location los usamos por si queremos acceder a un servidor específico, donde ponemos en el path el nombre del servidor, y como vemos nos redirige al servidor indicado al directorio raíz.

Para lanzar el balanceador de carga, simplemente ejecutamos el siguiente comando:

```

docker run --name balanceador --log-driver syslog --log-opt syslog-address=udp://10.0.0.2:514 --log-opt
tag=balanceador -p80:80 --rm -v /home/ubuntu/balanceador/default.conf:/etc/nginx/conf.d/default.conf
nginx

```

Que como vemos, lanza el Docker con nombre “balanceador”, para que así sea más fácil la tarea de reiniciarlo, pues si no tendríamos que referenciarlo con el nombre por defecto que se genera uno nuevo cada vez que ejecutamos el Docker. También indicamos que registre logs y les ponga el tag “balanceador”, también vinculamos el puerto 80 del balanceador (máquina) con el puerto 80 del Docker, y ejecutamos el montado del archivo “default.conf” en el directorio “/etc/nginx/conf.d” del docker.

Buscando la escalabilidad y dinamismo de la red lo hemos configurado de forma que en el grupo “upstream prueba” se encuentren solamente los servidores activos, pues la idea es

que estos vayan aumentando o disminuyendo de número en función del tráfico cursado. Para conseguir este objetivo vamos a usar el rsyslog.

El balanceador tiene que estar escuchando estos logs generados por los Docker de los servidores. Para ello añadimos el siguiente fichero “/etc/rsyslog.d/balanceador.conf”, que como vemos va a filtrar todos los logs recibidos en carpetas en función del programa que generó el log. Así los logs que generan los servidores, y que hemos puesto el tag “docker”, se guardarán en el fichero /var/log/docker.log

```
#PATH: Gestor - /etc/rsyslog.d/host1.conf

#UDP
module(load="imudp")
input(type="imudp" port="514")

$template RemoteLogs,"/var/log/remote/%PROGRAMNAME%.log"
*.? ?RemoteLogs
```

Ilustración 3/etc/rsyslog.d/balanceador.conf

Generando así los siguientes logs los servidores:

```
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Serving Flask app 'main' (lazy loading)#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Environment: production#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: #033[2m Use a production WSGI server instead.#033[0m#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Debug mode: on#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Running on all addresses.#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: WARNING: This is a development server. Do not use it in a production deployment.#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Restarting with stat#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Debugger is active!#015
Jan 22 22:45:01 10.0.0.4 docker[1370]: * Debugger PIN: 138-484-603#015
Jan 22 22:45:11 10.0.0.4 docker[1370]: 10.0.0.2 - - [22/Jan/2022 22:45:11] "GET / HTTP/1.0" 200 -#015
Jan 22 22:45:11 10.0.0.4 docker[1370]: 10.0.0.2 - - [22/Jan/2022 22:45:11] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
Jan 22 22:45:20 10.0.0.4 docker[1370]: 10.0.0.2 - - [22/Jan/2022 22:45:20] "GET /esquema HTTP/1.0" 200 -#015
Jan 22 22:45:20 10.0.0.4 docker[1370]: 10.0.0.2 - - [22/Jan/2022 22:45:20] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
Jan 22 22:45:20 10.0.0.4 docker[1370]: 10.0.0.2 - - [22/Jan/2022 22:45:20] "#033[36mGET /images/esquema.jpg HTTP/1.0#033[0m" 404 -#015
```

Ilustración 4/var/log/remote/docker.log

Y los siguientes logs el balanceador:

```
Jan 22 22:45:11 balanceador.europe-west2-c.c.proyecto-gar-2122.internal balanceador[29983]: 92.190.99.38 - - [22/Jan/2022:22:45:11 +0000] "GET /static
like Gecko) Chrome/97.0.4692.99 Safari/537.36" "-"
Jan 22 22:45:20 balanceador.europe-west2-c.c.proyecto-gar-2122.internal balanceador[29983]: 92.190.99.38 - - [22/Jan/2022:22:45:20 +0000] "GET /esquem
2.99 Safari/537.36" "-"
Jan 22 22:45:20 balanceador.europe-west2-c.c.proyecto-gar-2122.internal balanceador[29983]: 92.190.99.38 - - [22/Jan/2022:22:45:20 +0000] "GET /static
KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36" "-"
Jan 22 22:45:20 balanceador.europe-west2-c.c.proyecto-gar-2122.internal balanceador[29983]: 92.190.99.38 - - [22/Jan/2022:22:45:20 +0000] "GET /images
it/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36" "-"
Jan 22 22:58:29 balanceador.europe-west2-c.c.proyecto-gar-2122.internal balanceador[29983]: 27.112.69.60 - - [22/Jan/2022:22:58:29 +0000] "GET / HTTP/
```

Ilustración 5 /var/log/balanceador.log

Como muestra del funcionamiento del algoritmo de reparto round robin, si la página web está formada por un único fichero, entonces lo que hará el balanceador es pedir la página primero a un servidor y luego a otro, pero si utilizamos un servicio web formado por varios archivos como es el Docker, que contiene el html y una hoja de estilos, entonces lo que hace es, si solamente tenemos dos servidores web, pedirle el html a uno, y la hoja de estilos al siguiente, como podemos ver en los logs:


```

proyecto_gar_2122@servidor1:~$ sudo docker run --log-driver syslog --log-opt
tag=docker --rm -p80:5000 -e HOSTNAME=$HOSTNAME -it javierdf/servidor_web
* Serving Flask app 'main' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production depl
Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production depl
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 213-433-304
10.0.0.2 - - [23/Jan/2022 14:31:15] "GET /static/hello.css HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:19] "GET /static/hello.css HTTP/1.0" 304 -
10.0.0.2 - - [23/Jan/2022 14:31:34] "GET /static/hello.css HTTP/1.0" 304 -
10.0.0.2 - - [23/Jan/2022 14:31:35] "GET /static/hello.css HTTP/1.0" 304 -
10.0.0.2 - - [23/Jan/2022 14:31:37] "GET /static/hello.css HTTP/1.0" 304 -
10.0.0.2 - - [23/Jan/2022 14:31:37] "GET / HTTP/1.0" 200 -

proyecto_gar_2122@servidor3:~$ sudo docker run --log-driver syslog
tag=docker --rm -p80:5000 -e HOSTNAME=$HOSTNAME -it javierdf/servid
* Serving Flask app 'main' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a prod
Use a production WSGI server instead.
* Debug mode: on
* Running on all addresses.
WARNING: This is a development server. Do not use it in a prod
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 796-286-560
10.0.0.2 - - [23/Jan/2022 14:31:15] "GET / HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:19] "GET / HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:34] "GET / HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:35] "GET / HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:37] "GET / HTTP/1.0" 200 -
10.0.0.2 - - [23/Jan/2022 14:31:37] "GET / HTTP/1.0" 200 -

```

Ilustración 6 Servidor 1 proporciona la hoja de estilos y servidor3 el html

Pero, para ir actualizando al balanceador de los servidores activos necesitamos un programa que lea este archivo constantemente para que cuando se reciba la notificación de que ha entrado un servidor nuevo lo incluya en el balanceador, o lo elimine si este deja de proporcionar su servicio. Para esto usamos swatchdog.

Swatchdog es un software que lee el fichero de logs que le mandemos, buscando un patrón y si lo encuentra, ejecuta la orden configurada. Para ello tenemos que crear nuestro propio archivo de configuración de swatchdog:

```

ubuntu@balanceador:~$ cat swatchdog/docker.conf
watchfor /docker-entrypoint.d\| is not empty, will attempt to perform configuration/
echo green
exec python3 /home/ubuntu/lector.py incluye '$_[3]'
exec sudo docker container restart balanceador

watchfor /!#1: exit/
echo red
exec python3 /home/ubuntu/lector.py elimina '$_[3]'
exec sudo docker container restart balanceador

watchfor /Serving Flask app 'main'/
echo green
exec python3 /home/ubuntu/lector.py incluye '$_[3]'
exec sudo docker container restart balanceador

watchfor /\^C/
echo red
exec python3 /home/ubuntu/lector.py elimina '$_[3]'
exec sudo docker container restart balanceador

```

Ilustración 7 swatchdog/docker.conf

Donde como vemos, hemos configurado que busque varios patrones (watchfor), y swatchdog los mostrará por pantalla además de ejecutar “python3 /home/ubuntu/lector.py” y reiniciará el balanceador de carga para que incluya los cambios producidos en “default.conf”. “/home/ubuntu/lector.py” solamente se encarga de actualizar “default.conf” con los nuevos servidores, y tiene el siguiente aspecto:

```
#!/usr/bin/python3

import os
import sys

def incluye(ip):
    cadena = "\tserver "+ ip + ';\n'
    with open("/home/ubuntu/balanceador/default.conf", 'r') as file:
        newline=file.readlines()
        #Comprobamos si ya está esa ip
        ya_presente = False
        for i in newline:
            if i == cadena:
                print('La ip ya está incluida en el archivo')
                ya_presente = True
        if not ya_presente:
            for i in newline:
                if i == '\t#SERVIDORES\n':
                    newline.insert(newline.index(i) +1 , cadena)

            with open("/home/ubuntu/balanceador/default.conf", "w") as file:
                for i in newline:
                    file.writelines(i);

def elimina(ip):
    cadena = "\tserver "+ ip + ';\n'
    with open("/home/ubuntu/balanceador/default.conf", 'r') as file:
        newline=file.readlines()
        for i in newline:
            if i == cadena:
                newline.pop(newline.index(i))
    with open("/home/ubuntu/balanceador/default.conf", "w") as file:
        for i in newline:
            file.writelines(i);

if __name__ == "__main__":
    #incluye(sys.argv[1])

    if sys.argv[1] == "incluye":
        incluye(sys.argv[2])
    else:
        elimina(sys.argv[2])
```

Para ejecutar swatchdog, pondremos al arrancar el balanceador el siguiente comando:

```
swatchdog -c /home/ubuntu/swatchdog/docker.conf -t /var/log/remote/docker.log --daemon
```

Donde estamos indicando que, lea los logs del archivo “/var/log/remote/docker.log” y actúe en consecuencia, a partir del archivo de configuración “/home/ubuntu/swatchdog/docker.conf”.

Monitorización con telegraf

Con el objetivo de que el gestor pueda almacenar todos los logs generados por el balanceador y los servidores, hemos configurado telegraf en el balanceador. Para ello, en el archivo de configuración de telegraf, hemos configurado los datos para que apunte al gestor, e indicamos la organización y bucket donde guardar los datos en influxdb:

```
#####
#                               OUTPUT PLUGINS                               #
#####

# Configuration for sending metrics to InfluxDB
[[outputs.influxdb_v2]]
  ## The URLs of the InfluxDB cluster nodes.
  ##
  ## Multiple URLs can be specified for a single cluster, only ONE of the
  ## urls will be written to each interval.
  ## ex: urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
  urls = ["http://192.168.0.4:8086"]

  ## Token for authentication.
  token = "0uB1qP07bK-DqIR4nTkiHRZQYoPcomUsvKlum2IDa1G2UeY9kNP2Mn7EE3ztPcmFkau1JacQ9h9t-10S4TjEg=="

  ## Organization is the name of the organization you wish to write to; must exist.
  organization = "proyect-gar"

  ## Destination bucket to write into.
  bucket = "LoadBalancer-Bck"
```

Ilustración 8/etc/telegraf/telegraf.conf

Además, hemos configurado el el plugin de syslog, donde indicamos donde está el servidor de syslog, en este caso, en localhost:

```
#####
#                               SERVICE INPUT PLUGINS                          #
#####

# Accepts syslog messages following RFC5424 format with transports as per RFC5426,
[[inputs.syslog]]
  ## Specify an ip or hostname with port - eg., tcp://localhost:6514, tcp://10.0.0.1:6514
  ## Protocol, address and port to host the syslog receiver.
  ## If no host is specified, then localhost is used.
  ## If no port is specified, 6514 is used (RFC5425#section-4.1).
  server = "tcp://localhost:6514"
```

Ilustración 9-Plugin syslog

Por último, configuramos el syslog para que me genere los reportes oportunos para el telegraf:

```
$ActionQueueType LinkedList # use asynchronous processing
$ActionQueueFileName srvrfrwd # set file name, also enables disk mode
$ActionResumeRetryCount -1 # infinite retries on insert failure
$ActionQueueSaveOnShutdown on # save in-memory data if rsyslog shuts down

# forward over tcp with octet framing according to RFC 5425
*. * @@(o)127.0.0.1:6514;RSYSLOG_SyslogProtocol23Format

# uncomment to use udp according to RFC 5424
#*. * @127.0.0.1:6514;RSYSLOG_SyslogProtocol23Format
```

Ilustración 10-/etc/rsyslog.d/telegraf.conf

Con todo esto configurado, accedemos al interfaz web de InfluxDB, podemos configurar peticiones a la base de datos, para filtrar los logs del bucket “LoadBalancer-Bck”, con tag “docker” (que son los logs de los servidores) y “balanceador” (logs del balanceador), quedando el siguiente query.

```

from(bucket: "LoadBalancer-Bck")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "syslog")
  |> filter(fn: (r) => r["appname"] == "docker" or r["appname"] == "balanceador")
  |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
  |> yield(name: "last")

```

Ahora introducimos este query en un panel de Grafana, y seleccionando el formato de logs, nos queda un panel como el siguiente

```

> 2022/01/23 14:34:07 [error] 30#30: *31 upstream timed out (110: Connection timed out) while connecting to upstream, client: 92.190.99.38, server: , request: "GET /esquema HTTP/1.1",
> 29983
> 92.190.99.38 - - [23/Jan/2022:14:34:07 +0000] "GET /images/esquema_imagen.png HTTP/1.1" 404 232 "http://34.142.80.169/esquema" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
> 10.0.0.2 - - [23/Jan/2022 14:34:07] "#033[33mGET /images/esquema_imagen.png HTTP/1.0#033[0m" 404 -#015
> 1334
> 10.0.0.2 - - [23/Jan/2022 14:34:07] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
> 1196
> 2022/01/23 14:33:07 [error] 30#30: *31 upstream timed out (110: Connection timed out) while connecting to upstream, client: 92.190.99.38, server: , request: "GET /esquema HTTP/1.1",
> 29983
> 92.190.99.38 - - [23/Jan/2022:14:32:02 +0000] "GET /favicon.ico HTTP/1.1" 404 232 "http://34.142.80.169/servidor1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
> 10.0.0.2 - - [23/Jan/2022 14:32:02] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
> 1334
> 10.0.0.2 - - [23/Jan/2022 14:32:02] "#033[33mGET /favicon.ico HTTP/1.0#033[0m" 404 -#015
> 1196
> 92.190.99.38 - - [23/Jan/2022:14:31:37 +0000] "GET /static/hello.css HTTP/1.1" 304 0 "http://34.142.80.169/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
> 10.0.0.2 - - [23/Jan/2022 14:31:37] "GET / HTTP/1.0" 200 -#015
> 1334
> 10.0.0.2 - - [23/Jan/2022 14:31:37] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
> 1196
> 92.190.99.38 - - [23/Jan/2022:14:31:36 +0000] "GET / HTTP/1.1" 499 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari
> 92.190.99.38 - - [23/Jan/2022:14:34:07 +0000] "GET /images/esquema_imagen.png HTTP/1.1" 404 232 "http://34.142.80.169/esquema" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
> 92.190.99.38 - - [23/Jan/2022:14:31:35 +0000] "GET /static/hello.css HTTP/1.1" 304 0 "http://34.142.80.169/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
> 10.0.0.2 - - [23/Jan/2022 14:31:35] "GET / HTTP/1.0" 200 -#015
> 1334
> 10.0.0.2 - - [23/Jan/2022 14:31:35] "#033[36mGET /static/hello.css HTTP/1.0#033[0m" 304 -#015
> 1196
> 92.190.99.38 - - [23/Jan/2022:14:31:34 +0000] "GET /static/hello.css HTTP/1.1" 304 0 "http://34.142.80.169/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li

```

Ilustración 11-Panel de logs en Grafana

Integración servicio SNMP (Pablo Domínguez)

A pesar de que nuestra red cuenta con distintos sistemas de monitorización, más intuitivos y amigables que SNMP, hemos decidido integrarlo, no ya como herramienta de monitorización, sino como nexo que ayuda a implementar otras funciones necesarias para el correcto funcionamiento de la red.

La idea principal que subyace tras la integración de este protocolo consiste en medir el tráfico de entrada del balanceador e informar al gestor sobre el mismo, para que así, éste último pueda aumentar el número de servidores a los que redirige el balanceador de carga, ofreciendo al usuario una experiencia de red apropiada, sin saturación o pérdidas.

Instalación SNMP

El primer paso a llevar a cabo consiste en la instalación y puesta en funcionamiento del protocolo, asegurando la comunicación entre la máquina gestor y la máquina balanceador de carga.

Empezando por nuestra máquina gestora, hemos de ejecutar los siguientes comandos, que permiten instalar al final el servicio snmp

```
sudo apt update
```

Con este comando, actualizamos el índice de paquetes del administrador de paquetes

```
sudo apt install snmp snmp-mibs-downloader
```

Este comando instala dos paquetes. El primero, snmp, contiene las herramientas y comandos de ejecución que permitirán enviar solicitudes SNMP a nuestra máquina balanceador de carga. El segundo paquete, snmp-mibs-downloader, instalará los archivos de base de información gestionada (MIBs).

Moviéndonos ahora a nuestra máquina balanceador de carga, empezamos siguiendo el primer paso que también hemos seguido en la máquina gestora.

```
sudo apt update
```

La explicación de este comando no varía con respecto a la anterior. El segundo comando a ejecutar será el siguiente.

```
sudo apt install snmpd
```

Este comando instalará el demonio SNMP en nuestra máquina balanceador de carga, que quedará a la escucha de posibles peticiones SNMP por parte de la máquina gestora, para responderlas si así se produjesen.

Con estos comandos, ya tenemos correctamente instaladas las herramientas que nos permitirán comunicarnos entre máquinas mediante el protocolo SNMP. El siguiente paso, llegados a este punto, es la correcta configuración del servicio, para poder plantear la correcta comunicación SNMP entre máquinas.

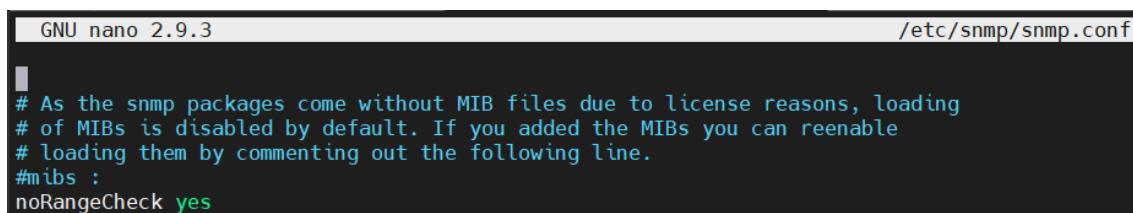
Configuración SNMP

A pesar de estar instalado ya el servicio que permite la comunicación entre las máquinas mediante el protocolo SNMP, todavía no pueden comunicarse entre ellas, puesto que hemos de modificar distintos archivos de configuración en ambas máquinas.

En nuestra máquina gestor, hemos de modificar el archivo de configuración `snmp.conf`. Para ello, nos valemos de los siguientes comandos:

```
sudo nano /etc/snmp/snmp.conf
```

Tras abrir el archivo con nuestro editor de texto, hemos de modificar el archivo para dejarlo de esta forma:



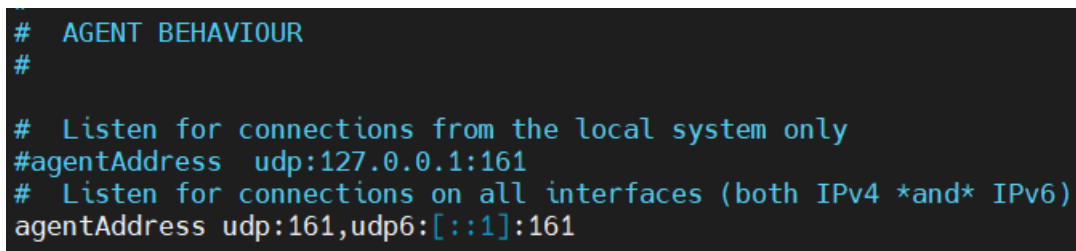
```
GNU nano 2.9.3 /etc/snmp/snmp.conf
# As the snmp packages come without MIB files due to license reasons, loading
# of MIBs is disabled by default. If you added the MIBs you can reenale
# loading them by commenting out the following line.
#mibs :
noRangeCheck yes
```

Hemos comentado la penúltima línea, dejándola como “`#mibs`”. También hemos añadido la última línea, “`noRangeCheck yes`”, que nos permite acceder a ciertas ramas de las mibs que pueden quedar fuera de rango en determinadas situaciones.

Tras este paso, la máquina gestora está correctamente configurada, queda configurar la máquina balanceadora de carga. Para ello, hemos de modificar en dicha máquina el archivo de configuración del demonio snmp, `snmpd.conf`. Así pues, ejecutamos el siguiente comando.

```
sudo nano /etc/snmp/snmpd.conf
```

Después de la ejecución del comando, nos encontramos ante un archivo de configuración muy amplio, que modificaremos ligeramente para proveer de conexión SNMP a ésta parte de la red.



```
# AGENT BEHAVIOUR
#
# Listen for connections from the local system only
#agentAddress udp:127.0.0.1:161
# Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161,udp6:[::1]:161
```

Comentamos la línea de `agentAdress udp_127.0.0.1:161`, para evitar que la máquina sólo pueda recibir peticiones de sí misma, y descomentamos la última línea que aparece en la imagen, para permitir el envío de peticiones desde otras máquinas.

```
# ACCESS CONTROL
#
view systemonly included .1.3.6.1.2.1.1 # system + hrSystem groups only
view systemonly included .1.3.6.1.2.1.25.1
rocommunity public 10.132.0.6 # Full access from the local host
rocommunity public default -V systemonly # Default access to basic system info
rocommunity6 public default -V systemonly # rocommunity6 is for IPv6
```

En la parte de Access Control, hemos de añadir una comunidad pública para la dirección IP de nuestra máquina gestora, en este caso 10.132.0.6. Tras esto, podremos enviar peticiones sólo de lectura (snmpget) desde nuestra máquina gestora.

Una vez quedan configurados estos archivos, podemos verificar el correcto funcionamiento, en primer lugar, comprobando el funcionamiento del servicio en la máquina balanceadora de carga, y en segundo lugar, intentando enviar un snmpget desde la máquina gestora.

```
diego@balanceador:~$ sudo systemctl status snmpd
● snmpd.service - Simple Network Management Protocol (SNMP) Daemon.
   Loaded: loaded (/lib/systemd/system/snmpd.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-01-22 15:51:07 UTC; 19h ago
     Process: 12116 ExecStartPre=/bin/mkdir -p /var/run/agentx (code=exited, status=0/SUCCESS)
    Main PID: 12123 (snmpd)
      Tasks: 1 (limit: 4662)
   CGroup: /system.slice/snmpd.service
           └─12123 /usr/sbin/snmpd -Lsd -Lf /dev/null -u Debian-snmp -g Debian-snmp -I -smux mteTrigger mteTriggerConf -f
```

En la primera captura de la máquina balanceador de carga, vemos que el demonio snmpd está cargado y activo, listo para recibir peticiones. Así pues, intentaremos enviar una petición desde nuestra máquina gestora, para ver si resuelve correctamente.

```
diego@gestor:~$ snmpget -v 2c -c public 10.154.0.2 system.sysLocation.0
SNMPv2-MIB::sysLocation.0 = STRING: Balanceador de carga
```

Vemos que a nuestro snmpget pidiendo el OID sysLocation, la máquina nos devuelve el valor esperado.

Así pues, una vez tenemos correctamente configuradas ambas máquinas, y podemos realizar peticiones SNMP, las cuales reciben respuesta, pasamos a la segunda parte a desarrollar.

Desarrollo del script para el Balanceador de Carga

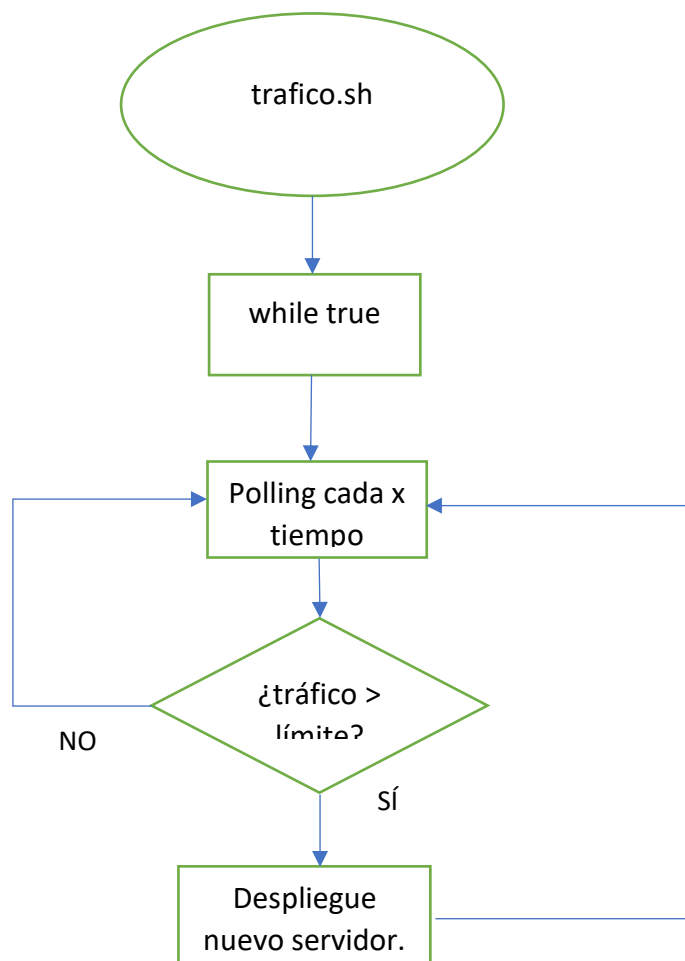
En esta parte, buscamos que, a través de información requerida y dispensada por las máquinas gestora y balanceadora de carga, respectivamente, mediante un proceso de polling, monitorizaremos el estado del enlace de la máquina balanceadora con el exterior. La idea es medir el tráfico entrante al balanceador, para que, cuando éste supere un límite fijado por nosotros, el gestor despliegue un nuevo servidor que disponga de conectividad directa con nuestro balanceador de carga, para que, de esta forma, el tráfico entrante se reparta entre varios servidores, evitando la saturación de la red, retardos y pérdidas de paquetes.

En primer lugar, y para que cuando acontezca la generación de nuevos servidores, éstos sean idénticos al primero, y se creen listos para su funcionamiento inmediato, hemos de

generar una imagen de nuestro servidor, la cual, a través de las distintas opciones de la consola de Google Cloud Platform, guardaremos y utilizaremos en el despliegue de otros servidores.

Un punto positivo que tiene la consola de GCP es que, aunque trabajemos a través de su interfaz, más amigable que la típica CLI, nos da la opción de sacar los comandos equivalentes a cada una de nuestras acciones, lo cual será de extrema utilidad para el desarrollo de nuestro script.

Para el desarrollo de nuestro script, planteamos un pequeño diagrama de flujo que nos ayuda al desarrollo del script a ejecutar.



Con este pequeño esquema pretendemos resumir de forma visual la metodología de trabajo que seguirá nuestro script `trafico.sh`. Dicho script, tras su desarrollo, queda como:


```
#!/bin/bash

NUMEROSERVIDOR=2
OCTETOSINICIALES=$(snmpget -v 2c -c public 10.154.0.2 1.3.6.1.2.1.2.2.1.10.2 | cut -d " " -f 4)

echo Siguiente servidor a crear es el servidor $NUMEROSERVIDOR
sleep 10
OCTETOSFINALES=$(snmpget -v 2c -c public 10.154.0.2 1.3.6.1.2.1.2.2.1.10.2 | cut -d " " -f 4)

while true
do

    let RESULTADO=${OCTETOSFINALES}-${OCTETOSINICIALES}
    let BRESULTADO=${RESULTADO}*8
    let BPS=${BRESULTADO}/10
    OCTETOSINICIALES=$(snmpget -v 2c -c public 10.154.0.2 1.3.6.1.2.1.2.2.1.10.2 | cut -d " " -f 4)
    sleep 10
    OCTETOSFINALES=$(snmpget -v 2c -c public 10.154.0.2 1.3.6.1.2.1.2.2.1.10.2 | cut -d " " -f 4)
    echo $BPS
    if [ $BPS -gt 1000 ]
    then
        gcloud beta compute instances create servidor$NUMEROSERVIDOR --project=proyecto-gar-2122 --zone=
        echo Se ha creado el servidor $NUMEROSERVIDOR
        NUMEROSERVIDOR=$((NUMEROSERVIDOR+1))
        echo Siguiente servidor a crear es el servidor $NUMEROSERVIDOR
    fi
done
```

En primer lugar, definimos un par de variables de entorno en nuestro script. Dichas variables de entorno se encargan de parametrizar el número de servidor que se creará a continuación (NUMEROSERVIDOR) y los octetos medidos por la interfaz a través de la cual nuestro balanceador de carga recibe peticiones (OCTETOSINICIALES).

El valor que asignamos a esta segunda variable de entorno, OCTETOS INICIALES, viene dado por el valor obtenido mediante un snmpget a nuestra máquina balanceador, a la cuál le pedimos el valor del OID ifInOctets, el cual representa el número de octetos que se han recibido a través de una interfaz.

Como el número plano no nos da ninguna información, es necesario realizar un proceso de polling, en este caso fijado a 10 segundos, y declarar una nueva variable de entorno OCTETOSFINALES, a la cual se le asigna de forma análoga a lo descrito anteriormente el valor de ifInOctets, pero como ha transcurrido un tiempo entre ambas medidas, el valor entre ellas diferirá.

Restando OCTETOSFINALES a OCTETOSINICIALES, obtenemos el valor que representa el número de octetos que se han recibido en un periodo de tiempo, en nuestro caso, 10 segundos. Dicho valor será almacenado en otra variable de entorno RESULTADO. Para representar el tráfico de forma más visual, nuestra variable RESULTADO será multiplicada por 8, para así obtener los bits durante ese periodo de tiempo, y por último, dividiremos este BRESULTADO entre el tiempo de polling, en nuestro caso 10 segundos, para obtener una media de los bits por segundo enviados en ese periodo de tiempo. Tras estas operaciones, mostramos por pantalla los bits por segundo que se han recibido.

Por último, nos ayudamos de la comparación de la variable BPS con nuestro valor límite fijado. Si no se supera dicho valor, no ocurre nada, pero cuando se supera, se procede a la creación de un nuevo servidor. Aquí entra en juego la explicación anterior sobre la posibilidad que brinda la consola de GCP de facilitar el comando de una serie de operaciones que hemos hecho desde la interfaz de usuario. El comando en cuestión es:


```
gcloud beta compute instances create servidor$NUMEROSERVIDOR --project=proyecto-gar-2122 --
zone=europe-west2-c --machine-type=e2-medium --network-interface=subnet=red-servidores,no-address --
metadata=startup-script=sudo\ docker\ run\ --log-driver\ syslog\ --log-opt\ syslog-address=udp://10.0.0.2:514\ --
log-opt\ tag=docker\ --rm\ -p80:5000\ -e\ HOSTNAME=\$HOSTNAME\ -it\ javierdf/servidor_web --
maintenance-policy=MIGRATE --service-account=384234051972-compute@developer.gserviceaccount.com --
scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.googleapis.com/auth/logging.write
,https://www.googleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/servicecontrol,https://ww
w.googleapis.com/auth/service.management.readonly,https://www.googleapis.com/auth/trace.append --min-
cpu-platform=Automatic --tags=http-server --no-shielded-secure-boot --shielded-vtpm --shielded-integrity-
monitoring --reservation-affinity=any --source-machine-image=servidores
```








Dicho comando se encarga de crear un nuevo servidor, asignándole el nombre que hemos parametrizado mediante la variable de entorno NUMEROSERVIDOR, y desplegándolo en nuestro proyecto, con las características indicadas (viene a ser un clon de nuestro primer servidor, con Docker levantado y funcional).

Tras la creación del nuevo servidor, se incrementa nuestra variable de entorno NUMEROSERVIDOR, y se continúa con la ejecución del polling.

Como muestra del funcionamiento, adjunto las siguientes capturas de pantalla.

Las instancias de VM son máquinas virtuales altamente configurables para ejecutar cargas de trabajo en la infraestructura de Google. [Más información](#)

 **Filtro** Ingresar el nombre o el valor de la propiedad

<input type="checkbox"/>	Estado	Nombre 	Zona	Recomendaciones	En uso por
<input type="checkbox"/>		balanceador	europe-west2-c		
<input type="checkbox"/>		gestor	europe-west1-b		
<input type="checkbox"/>		honeypot	us-central1-a		
<input type="checkbox"/>		manager	europe-west1-b		
<input type="checkbox"/>		prueba-pablo	southamerica-east1-b		
<input type="checkbox"/>		servidor1	europe-west2-c		

En la primera captura de pantalla, podemos ver que sólo existe el servidor1 en nuestra red (en este caso está parado, pero para la demostración, no tiene ninguna importancia).

```

diego@gestor:~$ ./trafico.sh
Siguiendo servidor a crear es el servidor 2
69
278
331
139
244
8604
Created [https://www.googleapis.com/compute/beta/projects/proyecto-gar-2122/zones/europe-west2-c/instances/servidor2].
NAME      ZONE      MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
servidor2 europe-west2-c e2-medium    10.0.0.19    10.0.0.19    RUNNING
Se ha creado el servidor 2
Siguiendo servidor a crear es el servidor 3
192
479
1933
Created [https://www.googleapis.com/compute/beta/projects/proyecto-gar-2122/zones/europe-west2-c/instances/servidor3].
NAME      ZONE      MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
servidor3 europe-west2-c e2-medium    10.0.0.20    10.0.0.20    RUNNING
Se ha creado el servidor 3
Siguiendo servidor a crear es el servidor 4
^C
diego@gestor:~$

```

Vemos que la ejecución del script funciona como se espera, imprime por pantalla los bits por segundo que ha recibido la interfaz en un periodo de tiempo, y cuando dichos bps superan el valor 1000, se crea un nuevo servidor. Tras su creación se continúa con el proceso de polling de forma normal.

INSTANCIAS

PROGRAMA DE LA INSTANCIA

Las instancias de VM son máquinas virtuales altamente configurables para ejecutar cargas de trabajo en la infraestructura de Google. [Más información](#)

Filtro Ingresar el nombre o el valor de la propiedad

<input type="checkbox"/>	Estado	Nombre ↑	Zona	Recomendaciones	En uso por
<input type="checkbox"/>	✓	balanceador	europe-west2-c		
<input type="checkbox"/>	✓	gestor	europe-west1-b		
<input type="checkbox"/>	✓	honeypot	us-central1-a		
<input type="checkbox"/>	✓	manager	europe-west1-b		
<input type="checkbox"/>	⦿	prueba-pablo	southamerica-east1-b		
<input type="checkbox"/>	⦿	servidor1	europe-west2-c		
<input type="checkbox"/>	✓	servidor2	europe-west2-c		
<input type="checkbox"/>	✓	servidor3	europe-west2-c		

Al volver a la consola de GCP, vemos que, efectivamente, se han creado los nuevos servidores, servidor2 y servidor3, y que estos se encuentran en ejecución, y son alcanzables dentro de la red.

Seguridad en la infraestructura (David Hernández)

Un tema también importante en la gestión de redes es la seguridad de la infraestructura de una red diseñada. En nuestra red tenemos expuesto a Internet el balanceador de carga, por tanto, puede sufrir ataques a través de diversos puertos, puerto 22, puerto 80, 443, etc.

Se pueden aplicar diferentes políticas para proteger dicha infraestructura. Se establecen dos métodos, uno creando un señuelo con muchas vulnerabilidades con el objetivo de darles acceso y que sean monitorizados para aprender como actuarían si vulnerasen el balanceador y qué medidas tomaríamos para mejorar la seguridad. Por otro lado, los atacantes podrían atacar al balanceador intentado acceder remotamente a través del puerto 22 SSH. Por tanto, el segundo método consiste en la redirección de un host al honeypot cuando se conoce que está siendo atacado, por ejemplo, mediante fuerza bruta ya sea mediante contraseña o con clave privada.

TPOT es una imagen basada en Debian que contiene una gran cantidad de honeypots dockerizados ya preparados, y configurados para su uso. Algunos de los honeypots que incluye son: adbhoney, ciscoasa, citrixhoneypot, conpot, cowrie, ddsospot, dicompot, honeytrap, snare, tanner, etc ...

Un punto importante de esta imagen es que la gestión del honeypot se puede controlar desde un entorno web. Se puede comprobar el almacenamiento, las redes, los dockers, cuentas de usuario, actualizaciones de software. Esto lo hace una plataforma sencilla y bastante intuitiva para el cliente.

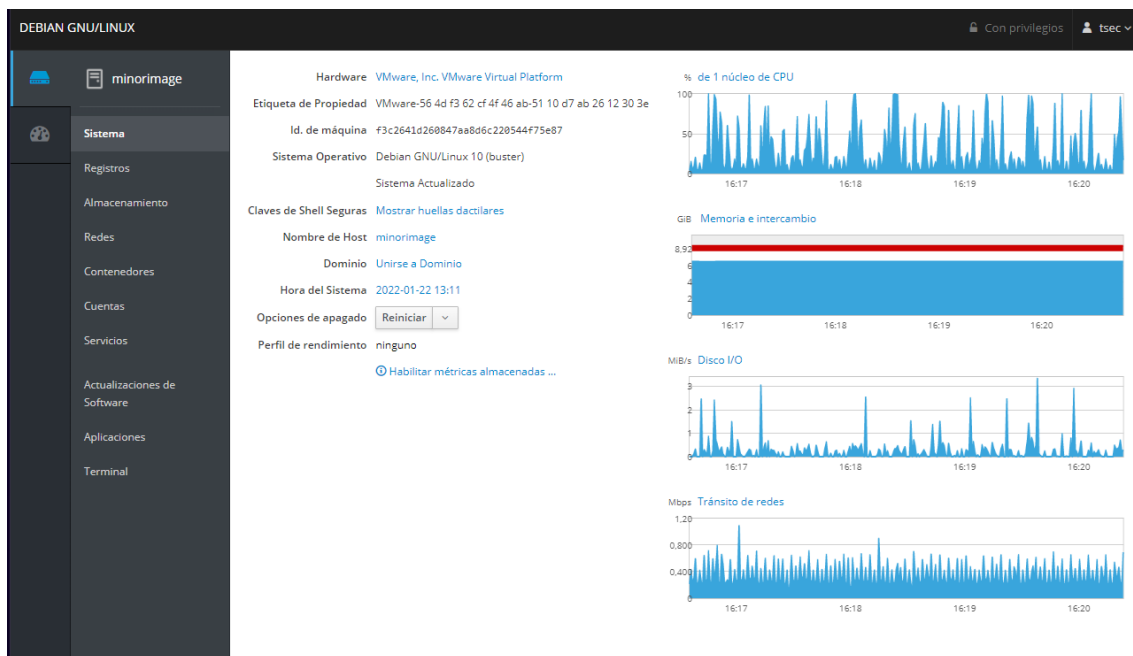
En Google Cloud se levanta una máquina de 8GB de memoria y 128GB de disco duro con una distribución Debian 10. Se instala la plataforma de TPOT.

```
sudo su  
  
apt update && apt upgrade  
  
apt git install  
  
git clone https://github.com/telekom-security/tpotce.git  
  
cd tpotce  
  
./install.sh --type=user
```

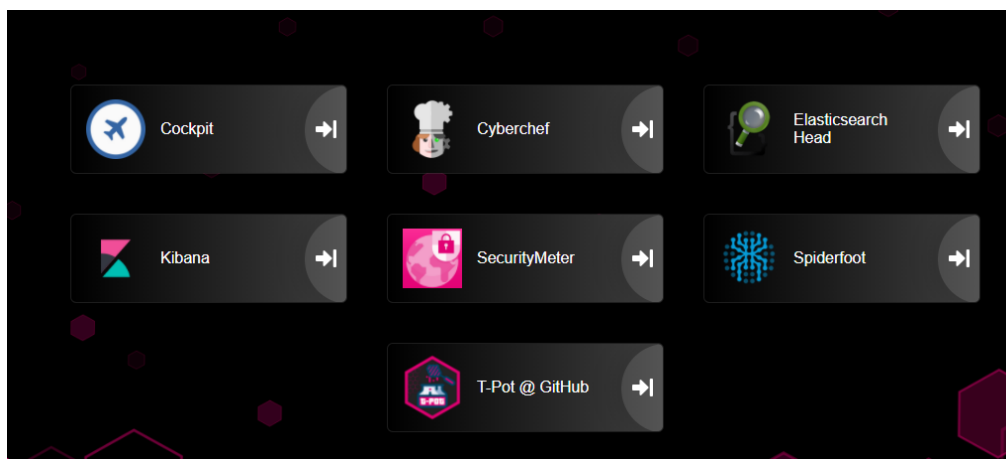
Una vez instalado y configurado el firewall de Google Cloud se intenta acceder a los controles de TPOT a través de https y se obtiene un error 504 por tanto se opta por establecer TPOT en una máquina local de la red LAN de casa. Para ello se abre el puerto 22 del router y se asocia dicho puerto externo con el puerto interno 22 a la dirección IP de la máquina donde corre TPOT. De esta manera no da error un error 504 y se puede acceder a los controles.

Para acceder a este sistema de control se puede realizar directamente a través de:

<https://<direcciónipTPOT>:64294>



o de forma indirecta a través de: <https://<direcciónipTPOT>:64297> pinchando en Cockpit.



Para llevar a cabo la gestión y monitorización de ataques al puerto SSH se usa el honeypot de Cowrie y para obtener los datos y graficar se utiliza otra herramienta contenida en TPOT: Kibana

Cowrie es un tipo de honeypot SSH y Telnet con el objetivo de monitorizarlos logs de un ataque de fuerza bruta a una máquina y la interacción del atacante con la shell. Con este modelo, se puede conseguir una mejor comprensión de los vectores de ataque como métodos, herramientas y otros procedimientos que realiza para atacar a una máquina y del escalado de privilegios para obtener el control total mediante permisos de administrador.

Como Cowrie es una simulación del servidor, no hay problema que el atacante entre en el sistema, ya que cuando consiga acceso, el sistema le dejará entrar a un sistema falso (señuelo), por tanto, el atacante pensará que está dentro del sistema real sin saber que es un

señuelo. De esta manera no se expone el servidor y se puede monitorizar las acciones llevadas cabo por el atacante.

“Kibana es una aplicación que se encuentra sobre el Elastic Stack y proporciona capacidades de visualización de datos y de búsqueda para los datos indexados en Elasticsearch. Comúnmente conocida como la herramienta de representación para el Elastic Stack (anteriormente llamado ELK Stack por Elasticsearch, Logstash y Kibana), Kibana también actúa como la interfaz de usuario para monitorear, gestionar y asegurar un cluster del Elastic Stack; además de como concentrador centralizado de las soluciones integradas desarrolladas en el Elastic Stack. “

En esta práctica se utiliza Kibana para obtener los datos de los ataques al sistema (Cowrie), muestra las IPs de los atacantes, el nombre del usuario y contraseña con el que han intentado penetrarlo. Además, también muestra estadísticas de la monitorización de la Shell una vez que el atacante está dentro del sistema, por ejemplo, comandos utilizados.

Para acceder a Kibana hay que entrar en: <https://<direcciónipTPOT>:64297> y redirigirse a la página de Kibana donde se encuentran dashboards ya creados de la monitorización de los diferentes honeypots.

Title	Description	Tags	Actions
<input type="checkbox"/> >T-Pot	T-Pot Dashboard		
<input type="checkbox"/> >T-Pot Live Attack Map	T-Pot Live Attack Map		
<input type="checkbox"/> Adbhoney	Adbhoney Dashboard		
<input type="checkbox"/> Ciscoasa	Ciscoasa Dashboard		
<input type="checkbox"/> CitrixHoneypot	CitrixHoneypot Dashboard		
<input type="checkbox"/> Conpot	Conpot Dashboard		
<input type="checkbox"/> Cowrie	Cowrie Dashboard		
<input type="checkbox"/> Ddospot	Ddospot Dashboard		
<input type="checkbox"/> Dicompot	Dicompot Dashboard		
<input type="checkbox"/> Dionaea	Dionaea Dashboard		
<input type="checkbox"/> ElasticPot	ElasticPot Dashboard		
<input type="checkbox"/> Endlessh	Endlessh Dashboard		
<input type="checkbox"/> Fatt	Fatt Dashboard		
<input type="checkbox"/> Glutton	Glutton Dashboard		
<input type="checkbox"/> Hellpot	Hellpot Dashboard		

En la imagen podemos observar que existe un rango amplio de dashboards con el nombre de sus respectivos Honeypots. El interesante en esta práctica es el de Cowrie, que es al que vamos a acceder, pero hay otro que es necesario explicar por su utilidad, T-POT. Este dashboard contiene toda la información resumida de todos los honeypots, por tanto, es útil primero acceder a dicho dashboard para comprobar qué honeypots han sido atacados, de esta manera sabremos de manera qué dashboards son interesantes acceder para obtener datos de los ataques.

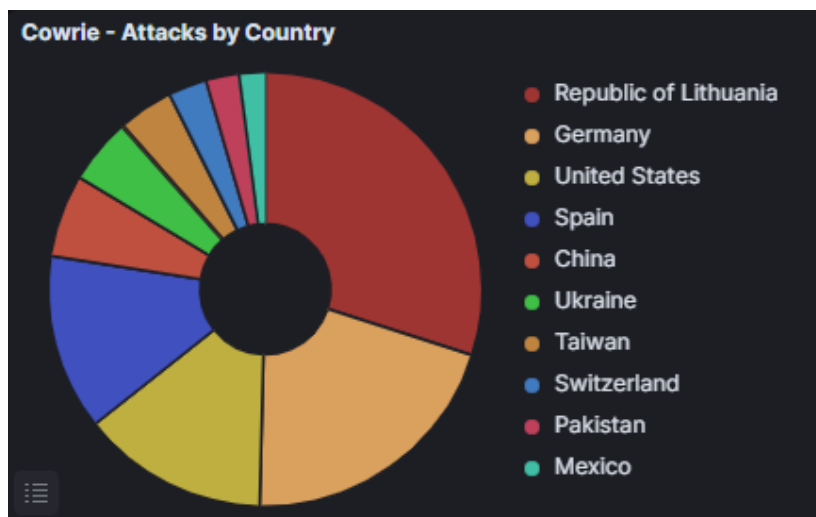
1,755	21	9	2	1
Cowrie - Attacks	ConPot - Attacks	Honeytrap - Attacks	CitrixHoneypot - Attacks	Adbhoney - Attacks

En el dashboard de Cowrie se pueden obtener histogramas sobre los ataques, ataques por puertos, ataques por países, usuarios y contraseñas de accesos, IPs de los atacantes, comandos registrados en la Shell ...

- Mapa de ataque: se observa en un mapa mundial las distintas localizaciones donde se ha lanzado los ataques.



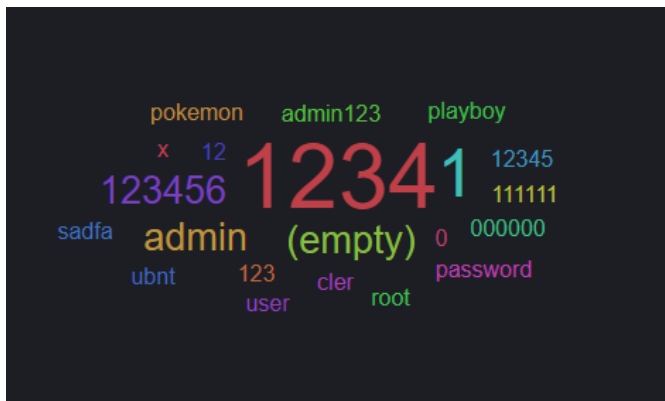
- Ataques por países: el nombre de los países desde se ejecutan los ataques



- Nombres de usuario



- Contraseñas



- Dirección IP de los atacantes

Source IP	Count
192.168.1.83	172
157.230.24.204	50
87.220.61.143	32
141.98.10.82	25
185.196.220.60	24
172.24.0.1	16
141.98.11.22	14
141.98.10.47	13
45.88.137.100	12
141.98.10.60	7

- Comandos ejecutados en la Shell del honeypot

Command Line Input	Count
exit	4
clear	3
nproc;uname -a	1
sudo hive-passwd 34g1gasd25345hjsdgtjhsftgsdfj2dsf12; sudo pkill...	1

Gracias a la instalación de un señuelo dentro de la red de una empresa configurado de manera de correcta y aislada de la red mediante firewalls y ubicada en la zona DMZ se logra que hackers que intenten penetrar el sistema no entren en la red interna pero sí en el Honeypot. La empresa, a través del señuelo, monitoriza dichos accesos lo que permite observar las acciones llevadas por los atacantes. Esto pone a la empresa en un lugar ventajoso, ya que, a raíz del análisis de los datos, se puede valorar la seguridad de la empresa para valorar la implementación de más o mejores métodos de seguridad para conseguir que la empresa sea segura.

El problema viene cuando no atacan al Honeypot sino atacan directamente al servidor. Para solventar dicho problema, se ha decidido crear un método de redireccionamiento SSH.

Redireccionamiento SSH

En caso de que un atacante intente entrar en el balanceador de carga, que es el servidor que está expuesto a Internet, mediante conexión remota a través del puerto 22 SSH, y consiga el acceso, conseguir que dicha conexión establecida se redirija a través de SSH a un Honeypot. La función de esta redirección es que no haga daño al sistema principal sino al honeypot, y además una posterior monitorización como se ha visto anteriormente.

El método establecido es monitorizar los logs para recoger la dirección IP y llevarla a un fichero donde se almacenan estas IPs. Por cada IP leída del fichero es un intento de acceso fallido. A los tres intentos de acceso fallido, es decir, si en el fichero se lee tres veces la misma dirección IP esta dirección se envía a la configuración de servicio SSH para que realice la redirección al Honeypot.

Para llevar a cabo la redirección se debe observar el fichero donde se registran todos los logs de autenticación. Este fichero es `"/var/log/auth.log"`. Cada vez que una máquina intenta acceder al sistema y consigue o se le deniega su conexión queda grabado en dicho archivo.

Con la configuración por defecto del servicio de logs: Rsyslog se genera un problema, cuando se repite el mismo mensaje de log más de dos veces, no imprime se graba cada mensaje, sino que con un solo mensaje se indica que dicho log se ha repetido varias veces. Supone que para la monitorización del archivo no se va a poder establecer un patrón de búsqueda fijo, lo que implica un código más complejo. Para facilitar dicha tarea de extracción de logs según un patrón se ha de establecer que los logs se graben línea a línea.

En el archivo de configuración de servicio de Rsyslog “/etc/Rsyslog.conf” existe una variable para cambiar dicho efecto. Posteriormente se reinicia el servicio.

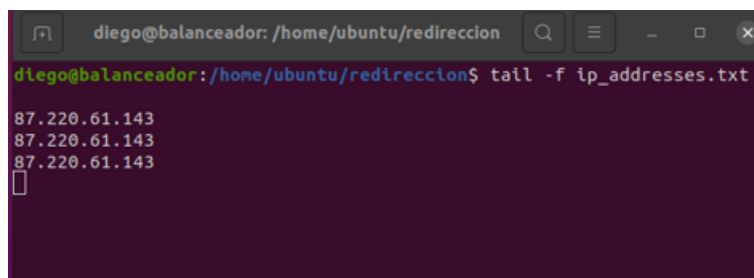
```
# Filter duplicated messages
$RepeatedMsgReduction off
```

De esta manera se consigue que se graben los mensajes de logs línea a línea y no con un solo mensaje de x logs repetidos.

Para establecer el diseño de la redirección primero se establece un patrón de búsqueda y este es el siguiente:

```
Jan 23 03:31:13 balanceador sshd[12856]: Connection closed by authenticating user diego
87.220.61.143 port 40342 [preauth]
```

Esta línea es la que indica que la conexión no ha sido establecida por error de autenticación, pero la dirección IP cambiada vez que un host intenta acceder, por tanto, este patrón no puede ser fijo ya que hay IPs distintas. Por ello se seleccionan las palabras “Connection”, “closed” y “authenticating”. Cuando la línea a leer tenga tantas palabras como la línea del patrón, contenga las tres palabras en dichas las posiciones correspondientes, “Connection”: posición 5, “closed”: posición 6 y “authenticating”: posición 8, se recoge la dirección IP que está localizada en la posición 11. La dirección IP se envía a un fichero donde se guardan todas las direcciones cuando hay intento de acceso fallido.

A terminal window titled 'diego@balanceador: /home/ubuntu/redireccion' shows the command 'tail -f ip_addresses.txt' being executed. The output displays three identical lines of the IP address '87.220.61.143' stacked vertically, with a cursor at the bottom.

A continuación, se lee este fichero. Cuando dentro de este archivo se leen tres direcciones IPs idénticas, como se puede observar en la imagen, se envía esta dirección al fichero de configuración “/etc/ssh/sshd_config”. Si a partir del tercer intento fallido se reenvía la dirección IP al fichero de configuración SSH, al cabo de muchos intentos con denegación de acceso por fallo, el fichero será bastante.

La lógica implementada para contar el número de intentos y determinar cuando y únicamente se envíe sólo una vez la dirección IP al archivo de configuración es la siguiente: cada vez que se lee una dirección IP se establece se guarda en una lista acompañada de dos flags: número de veces que se repite la IP, es decir, el número de intentos fallidos, y un campo (0 o 1) para determinar si la IP ya ha sido enviada al archivo de configuración.

La mecánica consiste en leer constantemente el archivo de direcciones IP y cuando aparece una nueva dirección se almacena con número de intento 1 y flag enviado 0. Cada vez que entra una nueva dirección IP se lee esta última comparándola con la lista de direcciones IP ya etiquetadas. Si la dirección IP se encuentra dentro de dicha lista se incrementa en uno el

número de envíos en caso contrario se establece una nueva entrada con dirección IP-número de intento, dirección IP- flag_enviado. Cuando el número de intentos es 3 se envía la dirección IP al archivo de configuración SSH y se establece el flag_enviado a 1 para que a partir de ahora, como en la configuración de SSH ya tiene registrada dicha IP, no se envíe más en caso de nueva lectura y misma dirección IP.

```
diego@balanceador:/home/ubuntu/redireccion$ python3 ssh_attempts.py
87.220.61.143 : 1 0
87.220.61.143 : 2 0
87.220.61.143 : 3 1
█
```

El archivo de configuración de SSH tiene la dirección IP pero necesita dirigir dicha IP al honeypot, para ello se consigue mediante este comando:

```
ForceCommand      ssh      -i      /home/ubuntu/redireccion/clave/manager-vm
diego@35.184.140.141
```

En el fichero se tiene lo siguiente:

```
Match Address 87.220.61.143
    ForceCommand ssh -i /home/ubuntu/redireccion/clave/manager-vm diego@35.184.140.141
```

Cuando lleva más de tres intentos fallidos y consigue acceso, se dispara el match y redirige la conexión al honeypot.

```
gar@ubuntu:~$ ssh -i clave/fake-vm diego@34.142.80.169
Load key "clave/fake-vm": invalid format
diego@34.142.80.169: Permission denied (publickey).
gar@ubuntu:~$ ssh -i clave/fake-vm diego@34.142.80.169
Load key "clave/fake-vm": invalid format
diego@34.142.80.169: Permission denied (publickey).
gar@ubuntu:~$ ssh -i clave/fake-vm diego@34.142.80.169
Load key "clave/fake-vm": invalid format
diego@34.142.80.169: Permission denied (publickey).
gar@ubuntu:~$ ssh -i clave/manager-vm diego@34.142.80.169
Linux honeypot 4.19.0-18-cloud-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Jan 23 03:16:38 2022 from 34.142.80.169
diego@honeypot:~$ ip addr
Object "addr" is unknown, try "ip help".
diego@honeypot:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 42:01:0a:80:00:06 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.6/32 brd 10.128.0.6 scope global dynamic ens4
        valid_lft 3098sec preferred_lft 3098sec
    inet6 fe80::4001:aff:fe80:6/64 scope link
        valid_lft forever preferred_lft forever
diego@honeypot:~$ █
```

Anexo: Código implementado para el redireccionamiento SSH

Código para leer el fichero “etc/log/auth.log”

```
from pathlib import Path
import os

#CONSTANTS
FILE_NAME = "/var/log/auth.log"
NUMBER_WORDS_LINE_FAILED_PASSWORD = 15

#GLOBAL VARIABLES
bytes_file = 0

def read_file(file_name):
    auth_file = open(file_name, 'r')
    try:
        auth_file.seek(bytes_file)
        for line in auth_file:
            word_list = line.split('\n')[0].split() #split by space
            number_words = len(word_list)
            if number_words == NUMBER_WORDS_LINE_FAILED_PASSWORD:
                #print (line.split('\n')[0])
                if word_list[5] == "Connection" and word_list[6] ==
"closed" and word_list[8] == "authenticating":
                    ip = word_list[11]
                    #print(ip)
                    command = "echo " + ip + " >>
/home/ubuntu/redireccion/ip_addresses.txt"
                    os.system(command)

        finally:
            auth_file.close()

if __name__ == "__main__":
    while True:
        file_size = Path(FILE_NAME).stat().st_size
        if bytes_file < file_size:
            read_file(FILE_NAME)

        bytes_file = file_size
```

Código para contabilizar el número de intentos fallidos

```

from pathlib import Path
import subprocess
import os

#variables gloables
bytes_file=0
ip_addresses = {}
ip_sent = {}

def visualizar_ip():
    for ip in ip_addresses:
        print (ip.strip('\n'), ":", ip_addresses[ip], ip_sent[ip])

#funcion para leer archivo
def read_file(filename):
    ip_file = open(filename, 'r')
    try:
        ip_file.seek(bytes_file)
        for ip in ip_file:
            if not ip_addresses or ip_addresses.get(ip)==None:
                ip_addresses[ip] = int (1)
                ip_sent[ip] = int (0)
            else:
                ip_addresses[ip]=ip_addresses.get(ip) + 1
    finally:
        ip_file.close()

#funcion para enviar una direccion ip cuando esté repetida más de 3 veces
(más de 3 intentos de conxi3n ssh fallidos)
#Cunado la direccin ip
def redirect_ip():
    for ip in ip_addresses:
        if ip_addresses.get(ip)>=3:
            #si la ip no ha sido enviada
            if ip_sent.get(ip) == 0:
                #llamar a redirect.sh y enviarle direccion ip
                command = './redirect.sh ' + ip.strip('\n')
                os.system(command)
                ip_sent[ip]=int(1) #poner la direccion ip como enviada
#flag_enviado

if __name__ == '__main__':
    filename = 'ip_addresses.txt'
    while True:
        file_size = Path(filename).stat().st_size
        if bytes_file < file_size:
            read_file(filename)
            redirect_ip()
            visualizar_ip()

        bytes_file = file_size

```

Script en bash para redirigir IP a “/etc/ssh/sshd_config”

```
IP=$1
```

```
FILE="/home/ubuntu/redireccion/clave/manager-vm"
```

```
sudo bash -c 'echo -e "Match Address '${IP}' \n \t ForceCommand ssh -i '${FILE}'  
diego@35.184.140.141" >> /etc/ssh/sshd_config'
```

```
sudo service ssh restart
```

Conclusiones finales

Una vez hemos finalizado el trabajo, hemos concluido que la experiencia de realizar un sistema de monitorización escalable en Google Cloud y AWS nos prepara para futuros proyectos que puedan surgir en este ámbito.

La creación de una VPN nos ha acercado a un sistema implementado ampliamente en el entorno empresarial y que ha facilitado la conexión de los distintos servicios implementados.

Una de las partes más complicadas del proyecto ha sido llevar a la práctica las ideas propuestas en el entorno Google Cloud, ya que la mayoría de las prácticas realizadas en la asignatura son en un entorno local.

En general estamos muy contentos con el resultado y esperamos que se haya podido transmitir todo el entusiasmo y el trabajo que hemos invertido en el proyecto.

Archivos del proyecto

Todos los archivos de configuración han sido subidos a un github:

<https://github.com/JavierDiazF/Proyecto-GAR>