Realizzare una appliance di rete personalizzata su AWS usando i Gateway Load Balancer

30 Settembre 2022 - 9 min. read

Advanced Networking ***Gateway Load Balancer***

"Quando guardi a lungo nell'abisso, l'abisso ti guarda dentro" (F. Nietzsche).

Nel nostro articolo precedente sulle configurazioni avanzate dei Load Balancer avevamo volutamente tralasciato momentaneamente i Gateway Load Balancer per dedicare a questo aspetto un approfondimento specifico.

I Gateway Load Balancer possono permetterci di osservare e filtrare il traffico di rete in uscita usando appliance dedicate. Nei paragrafi successivi vedremo come fare, assumendo che sia già stata progettata ed implementata una soluzione di networking centralizzato con un Transit Gateway.

Per utilizzare un IDS o firewall custom possiamo configurare il routing della VPC per inotrare il traffico ad una interfaccia ENI di una istanza EC2. Questa soluzione è semplice da implementare, ma non è altamente affidabile, né scalabile: se l'istanza smette di funzionare, la regola impostata rimane legata alla ENI e non è possibile fare in modo che cambi automaticamente.

I Gateway Load Balancer permettono di risolvere questo problema - e guadagnare quindi in affidabilità - instradando il traffico di Livello 3 distribuendolo a istanze EC2 in alta affidabilità, a prescindere dal protocollo o porta utilizzati. Usando altri load balancer si resterebbe legati a listener con specifici protocolli o porte: non sarebbe possibile, ad esempio, inoltrare traffico ICMP.

Molti fornitori software, come Cisco, F5 e Fortinet mettono già a disposizione appliance pronte all'uso. L'elenco completo è disponibile qui.

In questo articolo creeremo una semplice appliance personalizzata che faccia da IDS e router utilizzando Linux, iptables e Suricata. In questo modo, avremo modo di capire come funziona la tecnologia dietro le quinte di queste soluzioni

Come funzionano i Gateway Load Balancer

Prima di passare alla parte pratica di implementazione descriviamo in breve questo tipo di load balancer.

Il Gateway Load Balancer (GWLB) può fare routing di tutto il traffico IP (TCP, UDP, ICMP, GRE). Il protocollo GENEVE è la componente tecnologica alla base del funzionamento di un GWLB.

GENEVE è un nuovo protocollo di incapsulamento definito nella RFC 8926, standard per tecnologie e vendor differenti. L'acronimo significa Generic Network Virtualization Encapsulation; permette di incapsulare il traffico in un tunnel, in modo che la rete fisica sottostante non sia a conoscenza del traffico trasportato. È utilizzato frequentemente per estendere VLAN e VXLAN fra più reti utilizzando internet.

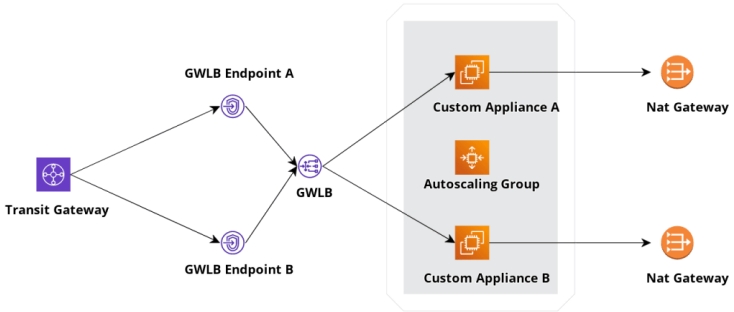
La nostra architettura di esempio

Il nostro obiettivo è ottenere un setup scalabile e fault tolerant. Il Gateway Load Balancer, come i "fratelli" Network ed Application, può essere distribuito in più Availability Zone.

Faremo anche in modo che il ciclo di vita delle nostre appliance sia gestito da un Autoscaling Group, così da aggiungere elasticità al nostro design.

Useremo i NAT Gateway per semplificare la gestione degli IP pubblici: alcuni partner o fornitori potrebbero limitare l'accesso a indirizzi IP pubblici predefiniti. Un NAT Gateway fa in modo che, se una istanza viene aggiunta dall'autoscaling group in una Availability Zone, l'ip pubblico di uscita rimanga sempre uguale usando un Elastic IP assegnato.

Di seguito lo schema dell'architettura che utilizzeremo:



Mettiamo ora mano alla parte pratica, con command line e Console AWS !

Prima di creare e configurare il nostro Network Load Balancer, avremo infatti bisogno di creare una AMI. Useremo Ubuntu 22.04, potremo aggiungere personalizzazioni più avanti.

Installazione del software e del tunnel manager

Una istanza EC2 può essere utilizzata come target per un GWLB se supporta la creazione di un tunnel GENEVE. Quando il tunnel risulta funzionante, il Gateway Load Balancer inizierà a distribuire il traffico.

Per prima cosa, dunque, bisogna fare in modo che il protocollo GENEVE sia supportato sulla nostra appliance custom.

È possibile usare il comando standard Linux

```
ip
```

per creare il tunnel, ma, per nostra fortuna, AWS mette a disposizione un tool che facilita il nostro compito. È disponibile qui.

Dando per scontata la creazione di una nuova istanza ed una AMI, partiamo con la compilazione del tool per la gestione del tunnel e l'installazione.

Installeremo anche **Suricata**, un Intrusion Detection System open-source. Useremo la configurazione di default

```
apt update
apt install -y build-essential "Development Tools"
apt install -y cmake g++ suricata
```

```
snap install aws-cli --classic
suricata-update #update rules for suricata
cd /opt
git clone https://github.com/aws-samples/aws-gateway-load-balancer-tunnel-handler
cd aws-gateway-load-balancer-tunnel-handler
cmake .
make
```

Al termine di queste operazioni il tunnel handler sarà pronto, viene creato un file eseguibile con nome "gwlbtun". Eseguendolo con il parametro "-h" viene visualizzata la pagina di help:

```
root@ip-10-101-5-238:/opt/aws-gateway-load-balancer-tunnel-handler# ./gwlbtun -h
AWS Gateway Load Balancer Tunnel Handler
Usage: ./gwlbtun [options]
Example: ./gwlbtun

 -h          Print this help
 -c FILE     Command to execute when a new tunnel has been built. See
 below for arguments passed.
 -r FILE     Command to execute when a tunnel times out and is about t
o be destroyed. See below for arguments passed.
 -t TIME     Minimum time in seconds between last packet seen and to c
onsider the tunnel timed out. Set to 0 (the default) to never time ou
t tunnels.
             Note the actual time between last packet and the destroy
 call may be longer than this time.
 -p PORT     Listen to TCP port PORT and provide a health status repor
t on it.
 -s          Only return simple health check status (only the HTTP res
ponse code), instead of detailed statistics.
 -d          Enable debugging output.
 -x          Enable dumping the hex payload of packets being processed
.
```

```
--------------------------------------------------------------------------------
-----------------------------------
Tunnel command arguments:
The commands will be called with the following arguments:
1: The string 'CREATE' or 'DESTROY', depending on which operation is
 occurring.
2: The interface name of the ingress interface (gwi-<X>).
3: The interface name of the egress interface (gwo-<X>).  Packets can
be sent out via in the ingress
  as well, but having two different interfaces makes routing and ipta
bles easier.
4: The GWLBE ENI ID in base 16 (e.g. '2b8ee1d4db0c51c4') associated w
ith this tunnel.

The <X> in the interface name is replaced with the base 60 encoded EN
I ID (to fit inside the 15 character
device name limit).
```

Oltre a stabilire la connessione con il Gateway Load Balancer e creare il tunnel GENEVE, gwlbtun mette a disposizione una porta utilizzabile per permettere al Target Group di effettuare gli health check, così da non dover implementare controlli ad hoc.

In aggiunta a questo, quando il tunnel viene creato o terminato, gwlbtun può eseguire uno script. Useremo questa funzione per fare in modo che uno script bash possa abilitare il routing IP, usando iptables per aggiungere e rimuovare le regole di NAT.

Nota: per far funzionare la nostra istanza è necessario disabilitare una feature di sicurezza chiamata "source/destination check". Questa impostazione fa in modo che il traffico non originato o diretto da o per l'istanza venga automaticamente bloccato.

L'istanza stessa deve essere in grado di farlo, per cui vedremo che dovremo creare ed assegnare un instance role con una autorizzazione specifica.

Lo script che segue pùo essere memorizzato nella directory

```
/opt/aws-gateway-load-balancer-tunnel-handler
```

con il nome

tunnel-handler.sh

```
#!/bin/bash

# Note: This requires this instance to have Source/Dest check disable
d; we need to assign a role to the ec2 instance to enable and disable
it


echo "Running tunnel handler script... "
echo Mode is $1, In Int is $2, Out Int is $3, ENI is $4

iptables -F
iptables -t nat -F
INSTANCE_ID=$(curl 169.254.169.254/latest/meta-data/instance-id

case $1 in
        CREATE)
                        echo "Disabling source and destination chec
k."
                        aws ec2 modify-instance-attribute --instance-
id=$INSTANCE_ID --source-dest-check

                echo "Setting up NAT and IP FORWARD"
                iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
                iptables -A FORWARD -i $2 -o $2 -j ACCEPT
                echo 1 > /proc/sys/net/ipv4/ip_forward
                echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
                echo 0 > /proc/sys/net/ipv4/conf/$2/rp_filter
                ;;
        DESTROY)
                        echo "Enabling source and destination check."
                        aws ec2 modify-instance-attribute --instance-
```

```
id=$INSTANCE_ID --no-source-dest-check
                echo "Removing IP FORWARD"
                echo 0 > /proc/sys/net/ipv4/ip_forward
                echo 1 > /proc/sys/net/ipv4/conf/all/rp_filter
                echo 1 > /proc/sys/net/ipv4/conf/$2/rp_filter
                ;;
        *)
                echo "invalid action."
                exit 1
                ;;
esac
```

A questo punto non ci resta che scrive una unit systemd per avviare gwlbtun. Useremo aws-gwlb.service come nome, mettendo il file nella directory

```
/lib/systemd/system/
```

```
[Unit]
Description=AWS GWLB Tunnel Handler
After=network.target

[Service]
ExecStart=/opt/aws-gateway-load-balancer-tunnel-handler/gwlbtun -c /o
pt/aws-gateway-load-balancer-tunnel-handler/tunnel-handler.sh -r /opt
/aws-gateway-load-balancer-tunnel-handler/tunnel-handler.sh -p 80
Restart=always
RestartSec=5s

[Install]
WantedBy=multi-user.target
Alias=aws-gwlb
```

Con i comandi che seguono abilitiamo il servizio (possiamo non avviarlo perché l'istanza su cui stiamo lavorando verrà usata come template)

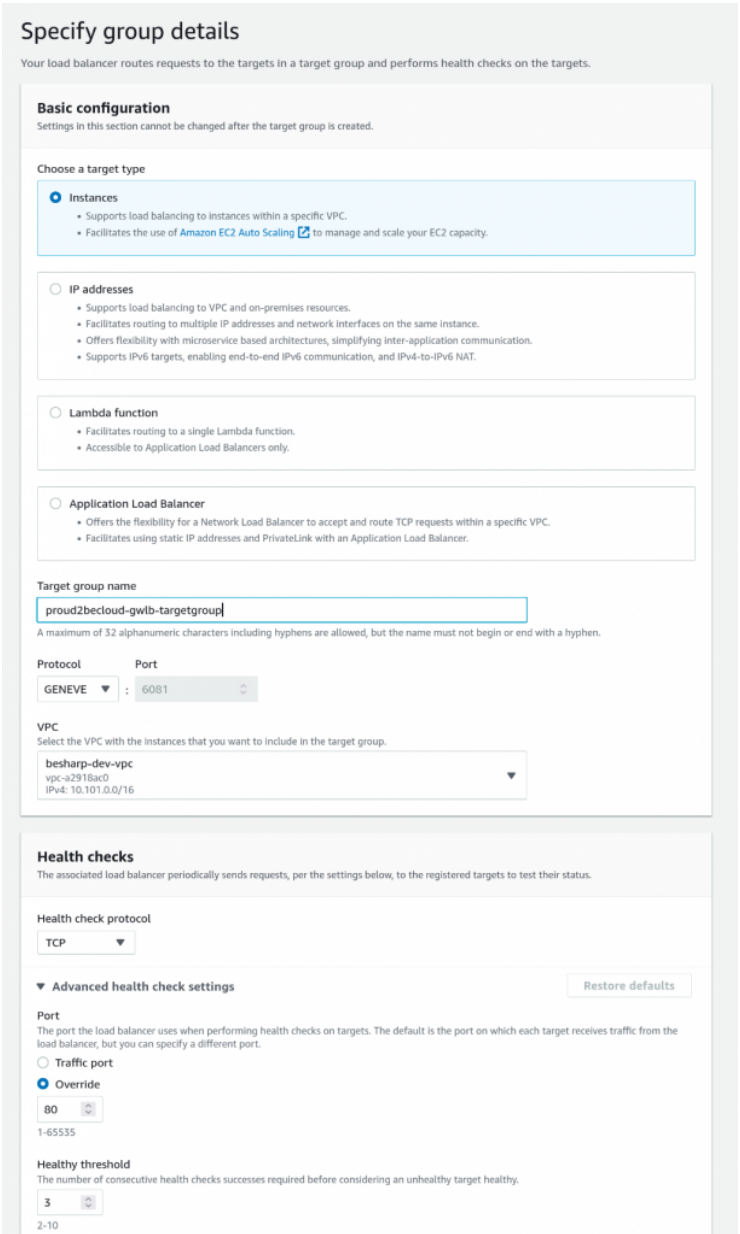
```
systemctl daemon-reload
systemctl enable aws-gwlb
```

Possiamo ora creare una AMI e iniziare a creare il Gateway Load Balancer.

Configurazione del Load Balancer

Per prima cosa va creato un Target Group, cliccando sulla sezione "**Target Group**"

Il tipo di target sarà "Instances", il nome è libero. Il protocollo deve essere "GENEVE". Siccome nella unit systemd abbiamo specificato il flag "-p 80" useremo la porta 80 per l'health check

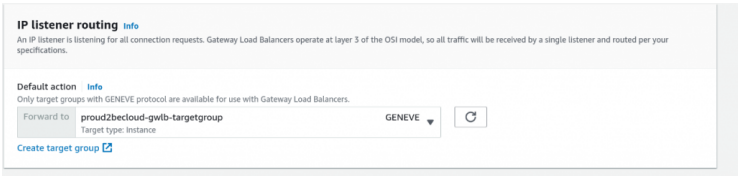


Al prossimo step non selezioneremo nessuna istanza, la gestione sarà compito dell'Autoscaling Group.

Terminata la configurazione del Target Group, passiamo alla creazione del load balancer. Facendo click su "**Load Balancers**" aggiungiamo un load balancer selezionando "Gateway Load Balancer".

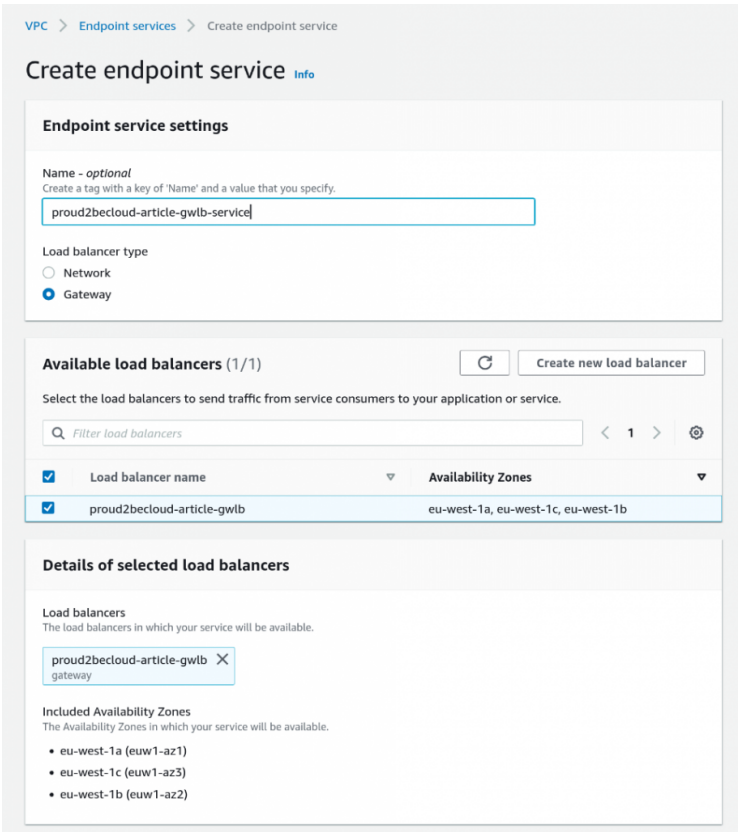
La configurazione di base è condivisa con gli altri tipi di load balancer: occorre assegnare un nome, selezionare la VPC e le subnet.

Alla sezione "IP listener routing" invece troveremo il target group appena creato:

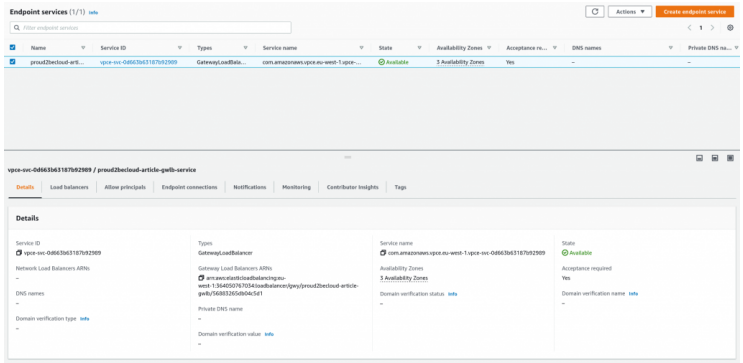


Per poter referenziare il Gateway load balancer nelle routing table occorre definire un endpoint service. Basta fare click su "Endpoint Services" nella sezione VPC della console AWS e crearne uno nuovo. Il processo è lo stesso usato per gli endpoint basati sui load balancer di tipo network (come descritto qui).

Il tipo dovrà essere "Gateway" e, una volta assegnato un nome, occorre selezionare il load balancer appena creato.

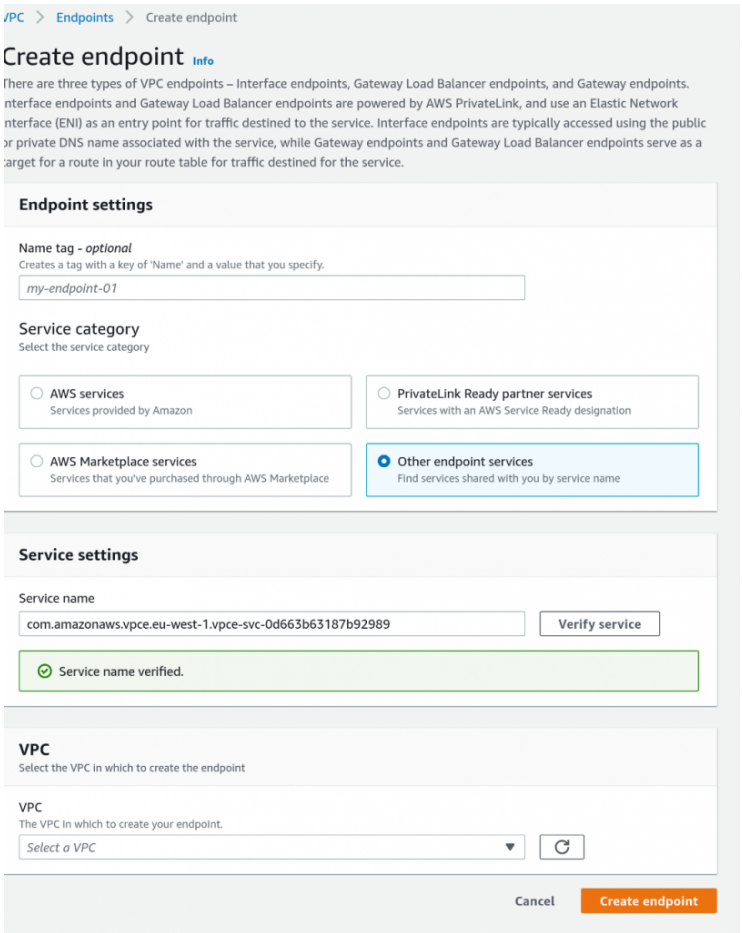


Una volta creato il campo "service name" è necessario per creare l'endpoint



Ora, alla sezione "Endpoint" e facendo click su "Create Enpoint" occorre selezionare "Other endpoint services", incollare il service name e fare click su "Verify Service".

Una volta verificato il servizio possiamo selezionare la VPC e una subnet in cui impostare l'endpoint (in questo caso useremo una subnet raggiungibile dal Transit Gateway)



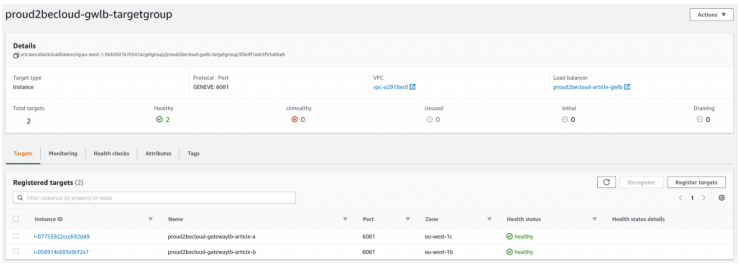
A questo punto occorre ripetere i passi per tutte le subnet, non dimenticate di accettare le connessioni!

La configurazione della parte rete è ora terminata. A questo punto non ci rimane che creare un autoscaling group. Per brevità non descriveremo questi passaggi.

È ora il momento di creare un ruolo per le nostre istanze e aggiungere la definizione al Launch Template. Il ruolo deve contenere la policy che permette alle istanze di disabilitare il source/destination check con una policy simile a questa:

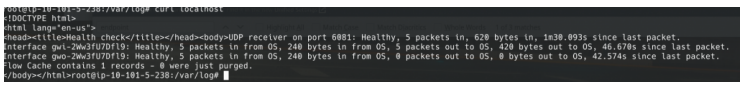
```
{
    "Sid": "Allow Source-Dest check modification",
    "Effect": "Allow",
    "Action": "ec2:ModifyInstanceAttribute",
    "Resource": "*"
}
```

Terminata la configurazione dell'autoscaling possiamo finalmente vedere le istanze in esecuzione nel target group

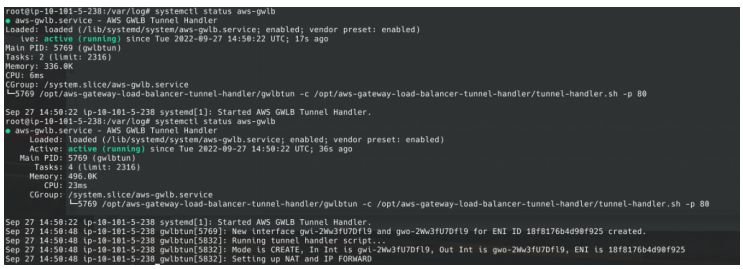


Effettuando l'accesso ad una istanza potremo vedere che:

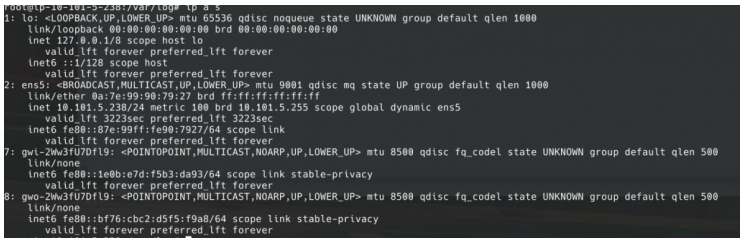
1. La porta specificata per l'health check è raggiunbile e fornisce statistiche.



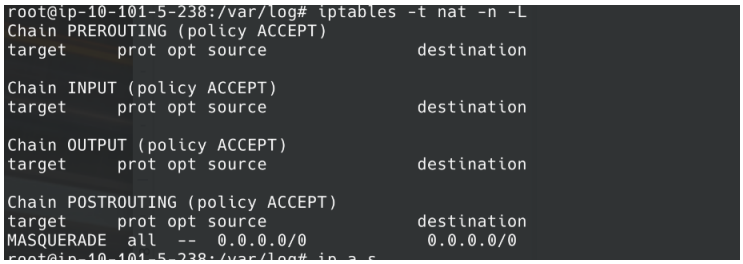
2. Il servizio è avviato e funzionante:



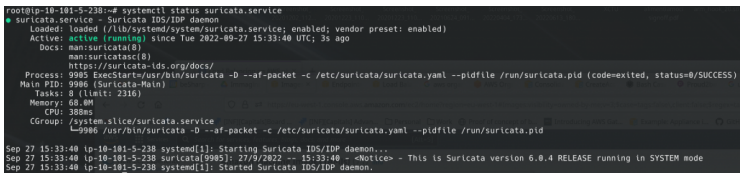
3. Due interfacce di rete (gwi-* e gwo-*) sono state create da gwlbtun



4. Le regole di firewall sono presenti



5. Ultimo (ma non ultimo), Suricata sarà pronto per catturare gli eventi di sicurezza



Prossimi passi

Per fare in modo che tutte le richieste HTTP uscenti siano tracciate, possiamo installare un proxy server in modalità transparent (come squid, ad esempio) sul template, centralizzando i log su CloudWatch Logs.

Suggerimento: nel file /etc/squid/squid.conf occorre abiltiare il "transparent mode", "SSL bumping" ed inserire le regole di NAT con iptables.

Facendo evolvere lo script di firewall si può filtrare il traffico in uscita (o usare uno strumento come EasyWall.

Conclusioni

Utilizzando un Gateway Load Balancer è possibile aumentare il controllo e la visibilità sul traffico di rete in uscita, mantenendo una soluzione affidabile, scalabile e personalizzabile.

Ora sappiamo come funzionano dietro le quinte le appliance di sicurezza di terze parti. Le implementazioni possono essere differenti, ma la tecnologia alla base è in comune.

Vi siete imbattuti in scenari particolari o avete idee per cui un Gateway Load Balancer può essere utile? Fatecelo sapere nei commenti!

About Proud2beCloud

Proud2beCloud è il blog di beSharp, APN Premier Consulting Partner italiano esperto nella progettazione, implementazione e gestione di infrastrutture Cloud complesse e servizi AWS avanzati. Prima di essere scrittori, siamo Solutions Architect che, dal 2007, lavorano quotidianamente con i servizi AWS. Siamo innovatori alla costante ricerca della soluzione più all'avanguardia per noi e per i nostri clienti. Su Proud2beCloud condividiamo regolarmente i nostri migliori spunti con chi come noi, per lavoro o per passione, lavora con il Cloud di AWS. Partecipa alla discussione!

Damiano Giorgi

Ex sistemista on-prem, pigro e incline all'automazione di task noiosi. Alla ricerca costante di novità tecnologiche e quindi passato al cloud per trovare nuovi stimoli.L'unico hardware a cui mi dedico ora è quello del mio basso; se non mi trovate in ufficio o in sala prove provate al pub o in qualche aeroporto!
