NEST: a Nebula Enrollment over Secure Transport system. Experimental Results

Use case

A Client who wants to establish a Nebula Mesh VPN has no officially provided way to automatically provision CA signed Nebula certificates and configuration files to the future end nodes of the network. Instead, he/she should rely on manual deployment or secure copy (scp) of such files, implying human intervention is mandatory. The hereby proposed system wants to achieve this distribution in an automated and secure way, insuring an authenticated, tamper-proof and confidential exchange of these crucial files in order to ensure that only the authorized hosts will join the secure Nebula network, using the intended configurations. This system could help set-up a Nebula Mesh VPN in an Industrial Control System (or IIoT) settings, i.e., deploying the Nebula certs. and configs. on PLCs (programmable logic controllers), SBC (Single Board Computers, i.e., Raspberry Pis) and EWS (Engineering Workstations), to create secure, peer to peer networks between these crucial assets, that will communicate over secure and isolated channels instead of the probably unsecure and unsegregated OT network. It can also help enforce Zero Trust principles by leveraging the identity-based routing and communication infrastructure provided by the Nebula Mesh VPN.

Desired results

1. The nest_service should be securely accessible to remote clients
2. The remote clients should be authenticated by the nest_service, to avoid foreign and unknown clients to gain access to legit Nebula keys and Configuration files
3. All 3 services should communicate securely among them, to also support scenarios in which the 3 services are deployed onto 3 different machines
4. The system should also support low-powered devices. To do so, eliptic curve encryption should be adopted by the nest_service and the system should support a serverkeygen option, that will ensure that secure nebula cryptographic material will be created server-side for the client and finally securely delivered to them.
5. The system should securely generate and distribute nebula certificates, keys and configuration files to the clients

Obtained results

Testbed

The following tests have been performed on an hybrid environment, consisting in both real and virtual machines, as shown in the following two tables:

Table 1: services description

Service name	Machine name	IP address	NEST system Nebula IP	Port
NEST CA	Lighthouse	192.168.80.1	192.168.80.1	53535
NEST Config	Lighthouse	192.168.80.2	192.168.80.2	61616
NEST Service	Lighthouse		192.168.80.3	8080

Table 2: clients description

Hostname	Arch-OS	Machine type	NEST clients Nebula IP	Enrollment endpoint
lighthouse	arm64-Linux	Raspberry Pi4	192.168.90.1	/enroll
nest_client_lin_64	amd64-Linux	Docker container	192.168.90.2	/enroll
nest_client_win	amd64-Windows	Real laptop	192.168.90.3	/enroll
nest_client_lin_386	386-Linux	Docker container	192.168.90.4	/serverkeygen

The hybrid environment shows off the capabilities of the system to be run on all the possible Operative systems and architectures, thanks to the go language cross compilation. In particular: the services are run on a Raspberry Pi4 (arm64), as well as the lighthouse client. The windows client is being run on a Windows machine, all the other clients, are run on Docker containers on separate virtual networks to simulate network segregation and different locations.

The services and the lighthouse client are on the same machine for convenience reasons: they're the only machines with a public IP in my possession and I wanted to create an interesting environment in which the services were publicly available to remote clients and the lighthouse could serve as a real, production-ready lighthouse as in a common nebula deployment.

The following demo has been also conducted in a fully virtual environment, with both clients and services running as Docker containers, yielding the same results.

Demo

First of all, let's check that all the services and clients complete their setup phase and start correctly.

```
/home/nest_client # echo $HOSTNAME
nest_client_lin_64
/home/nest_client # ip a
[...]
4: nebula: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc fq_codel state
UNKNOWN qlen 500
    link/[65534]
    inet 192.168.90.2/24 scope global nebula
       valid_lft forever preferred_lft forever
[...]
```

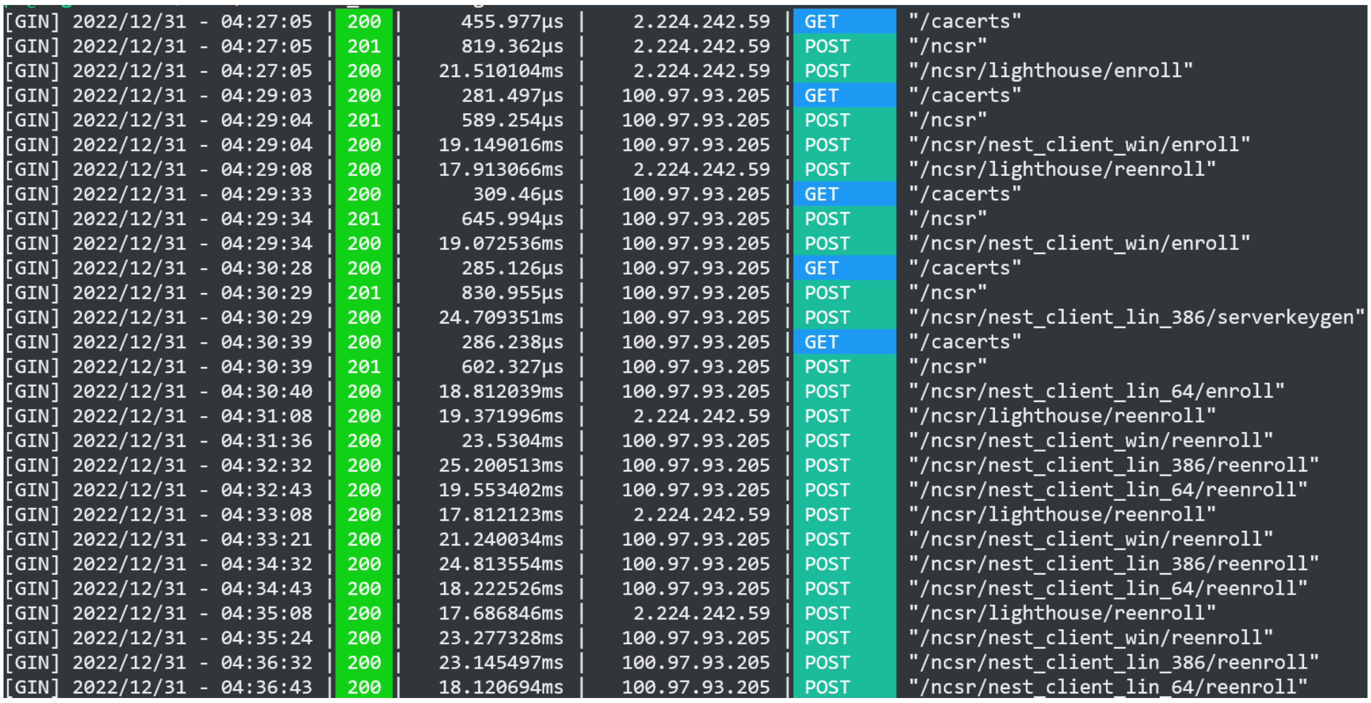
```
/home/nest_client # echo $HOSTNAME
nest_client_lin_386
/home/nest_client # ip a
[...]
4: nebula: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc fq_codel state
UNKNOWN qlen 500
    link/[65534]
    inet 192.168.90.4/24 scope global nebula
       valid_lft forever preferred_lft forever
[...]
```

```
PS D:\Uni\Tesi\Magistrale\nest_client_win> ipconfig
Configurazione IP di Windows

Scheda sconosciuta nebula:

   Suffisso DNS specifico per connessione:
   Indirizzo IPv4. . . . . . . . . . . . : 192.168.90.3
   Subnet mask . . . . . . . . . . . . . : 255.255.255.0
   Gateway predefinito . . . . . . . . . :
```

```
$ uname -a
Linux lighthouse 5.15.76-v8+ #1597 SMP PREEMPT Fri Nov 4 12:16:41 GMT 2022 aarch64
GNU/Linux
$ ip a
[...]
32: nebula_service: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc
pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.80.3/24 scope global nebula_service
       valid_lft forever preferred_lft forever
33: nebula_ca: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc pfifo_fast
state UNKNOWN group default qlen 500
    link/none
    inet 192.168.80.1/24 scope global nebula_ca
       valid_lft forever preferred_lft forever
34: nebula_config: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc pfifo_fast
state UNKNOWN group default qlen 500
    link/none
    inet 192.168.80.2/24 scope global nebula_config
       valid_lft forever preferred_lft forever
35: nebula: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc pfifo_fast state
UNKNOWN group default qlen 500
    link/none
    inet 192.168.90.1/24 scope global nebula
       valid_lft forever preferred_lft forever
[...]
```

As we can see from the logs, all the nebula interfaces on both the clients (first three logs) and the lighthouse (one nebula interface per service + the "nebula" interface for the lighthouse client).

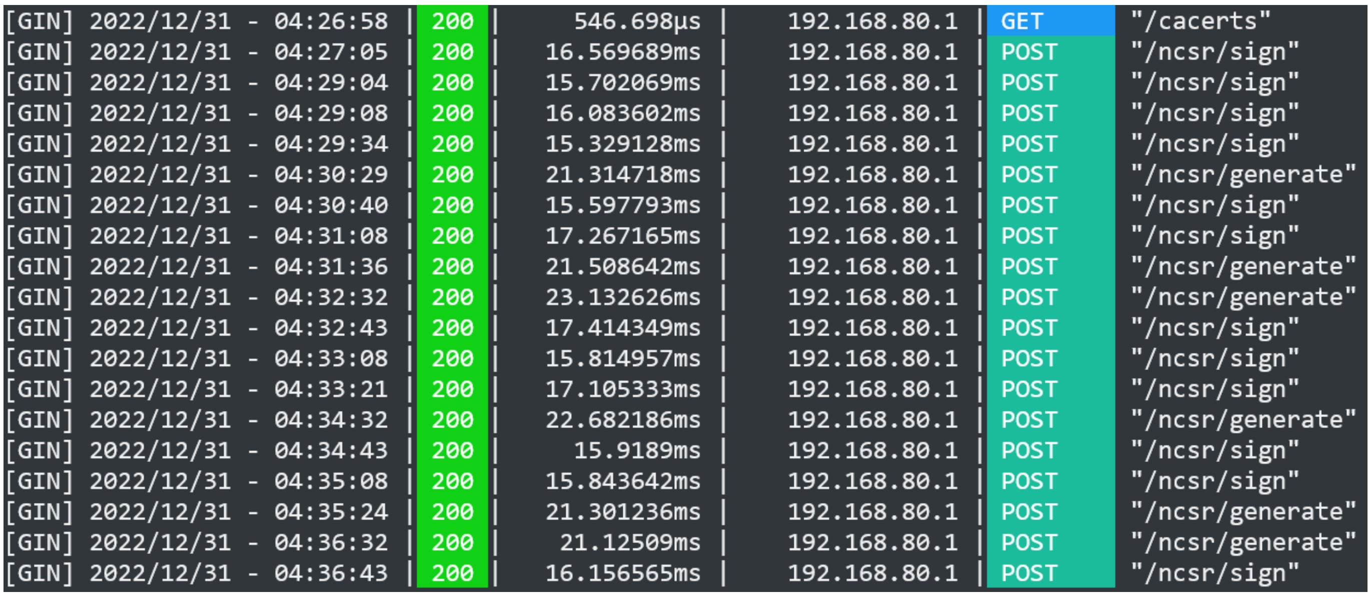
Now, let's investigate the services log files to see if the clients' request are coeherent with the environment we set up.



```
[GIN] 2022/12/31 - 04:27:05 | 200 |     455.977µs |     2.224.242.59 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:27:05 | 201 |     819.362µs |     2.224.242.59 | POST     "/ncsr"
[GIN] 2022/12/31 - 04:27:05 | 200 |   21.510104ms |     2.224.242.59 | POST     "/ncsr/lighthouse/enroll"
[GIN] 2022/12/31 - 04:29:03 | 200 |     281.497µs |    100.97.93.205 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:29:04 | 201 |     589.254µs |    100.97.93.205 | POST     "/ncsr"
[GIN] 2022/12/31 - 04:29:04 | 200 |   19.149016ms |    100.97.93.205 | POST     "/ncsr/nest_client_win/enroll"
[GIN] 2022/12/31 - 04:29:08 | 200 |   17.913066ms |     2.224.242.59 | POST     "/ncsr/lighthouse/reenroll"
[GIN] 2022/12/31 - 04:29:33 | 200 |      309.46µs |    100.97.93.205 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:29:34 | 201 |     645.994µs |    100.97.93.205 | POST     "/ncsr"
[GIN] 2022/12/31 - 04:29:34 | 200 |   19.072536ms |    100.97.93.205 | POST     "/ncsr/nest_client_win/enroll"
[GIN] 2022/12/31 - 04:30:28 | 200 |     285.126µs |    100.97.93.205 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:30:29 | 201 |     830.955µs |    100.97.93.205 | POST     "/ncsr"
[GIN] 2022/12/31 - 04:30:29 | 200 |   24.709351ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_386/serverkeygen"
[GIN] 2022/12/31 - 04:30:39 | 200 |     286.238µs |    100.97.93.205 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:30:39 | 201 |     602.327µs |    100.97.93.205 | POST     "/ncsr"
[GIN] 2022/12/31 - 04:30:40 | 200 |   18.812039ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_64/enroll"
[GIN] 2022/12/31 - 04:31:08 | 200 |   19.371996ms |     2.224.242.59 | POST     "/ncsr/lighthouse/reenroll"
[GIN] 2022/12/31 - 04:31:36 | 200 |     23.5304ms |    100.97.93.205 | POST     "/ncsr/nest_client_win/reenroll"
[GIN] 2022/12/31 - 04:32:32 | 200 |   25.200513ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_386/reenroll"
[GIN] 2022/12/31 - 04:32:43 | 200 |   19.553402ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_64/reenroll"
[GIN] 2022/12/31 - 04:33:08 | 200 |   17.812123ms |     2.224.242.59 | POST     "/ncsr/lighthouse/reenroll"
[GIN] 2022/12/31 - 04:33:21 | 200 |   21.240034ms |    100.97.93.205 | POST     "/ncsr/nest_client_win/reenroll"
[GIN] 2022/12/31 - 04:34:32 | 200 |   24.813554ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_386/reenroll"
[GIN] 2022/12/31 - 04:34:43 | 200 |   18.222526ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_64/reenroll"
[GIN] 2022/12/31 - 04:35:08 | 200 |   17.686846ms |     2.224.242.59 | POST     "/ncsr/lighthouse/reenroll"
[GIN] 2022/12/31 - 04:35:24 | 200 |   23.277328ms |    100.97.93.205 | POST     "/ncsr/nest_client_win/reenroll"
[GIN] 2022/12/31 - 04:36:32 | 200 |   23.145497ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_386/reenroll"
[GIN] 2022/12/31 - 04:36:43 | 200 |   18.120694ms |    100.97.93.205 | POST     "/ncsr/nest_client_lin_64/reenroll"
```

Image 1: nest_service logs.

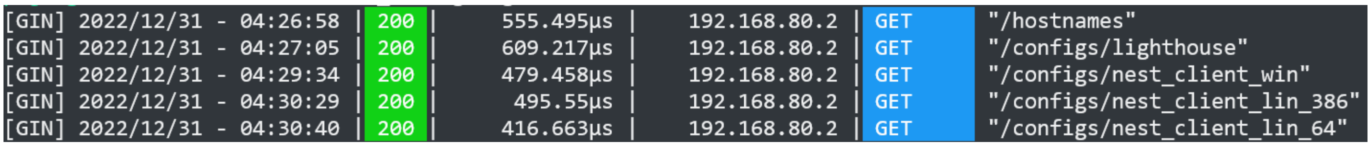
The first three lines represent the lighthouse requesting the nest_ca nebula cert, the lighthouse authenticating to the nest_service and then finally enrolling. We can see the same process for the other nest_clients (nest_client_lin_386 enrolls via serverkeygen, as it should be by the environment setup). Every 2 minutes, the clients re-enroll (nest_ca was set up to generate certificates whose validity was 2 minutes).



```
[GIN] 2022/12/31 - 04:26:58 | 200 |     546.698µs |     192.168.80.1 | GET      "/cacerts"
[GIN] 2022/12/31 - 04:27:05 | 200 |   16.569689ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:29:04 | 200 |   15.702069ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:29:08 | 200 |   16.083602ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:29:34 | 200 |   15.329128ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:30:29 | 200 |   21.314718ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:30:40 | 200 |   15.597793ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:31:08 | 200 |   17.267165ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:31:36 | 200 |   21.508642ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:32:32 | 200 |   23.132626ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:32:43 | 200 |   17.414349ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:33:08 | 200 |   15.814957ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:33:21 | 200 |   17.105333ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:34:32 | 200 |   22.682186ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:34:43 | 200 |     15.9189ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:35:08 | 200 |   15.843642ms |     192.168.80.1 | POST     "/ncsr/sign"
[GIN] 2022/12/31 - 04:35:24 | 200 |   21.301236ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:36:32 | 200 |    21.12509ms |     192.168.80.1 | POST     "/ncsr/generate"
[GIN] 2022/12/31 - 04:36:43 | 200 |   16.156565ms |     192.168.80.1 | POST     "/ncsr/sign"
```

Image 2: nest_ca logs.

The nest_ca logs are identical with respect to the nest_service ones (One sign operation per every enroll or re-enroll request and one generate operation for every serverkeygen on re-enroll with serverkeygen option). The first line represents the nest_service requesting the nest_ca nebula certificate to be sent to the clients.



```
[GIN] 2022/12/31 - 04:26:58 | 200 |      555.495µs |     192.168.80.2 | GET      "/hostnames"
[GIN] 2022/12/31 - 04:27:05 | 200 |      609.217µs |     192.168.80.2 | GET      "/configs/lighthouse"
[GIN] 2022/12/31 - 04:29:34 | 200 |      479.458µs |     192.168.80.2 | GET      "/configs/nest_client_win"
[GIN] 2022/12/31 - 04:30:29 | 200 |       495.55µs |     192.168.80.2 | GET      "/configs/nest_client_lin_386"
[GIN] 2022/12/31 - 04:30:40 | 200 |      416.663µs |     192.168.80.2 | GET      "/configs/nest_client_lin_64"
```

Image 3: nest_config logs.

The nest_config logs are much shorter, as configuration files are requested only at enrollment time, so once per client. The first line represents the nest_service requesting the list of valid and expected hostnames with which it will validate the clients' requests.

To check if the Nebula firewall rules are correctly enforced, let's try simple pings with the nebula IPs (ICMP should be allowed on every client).

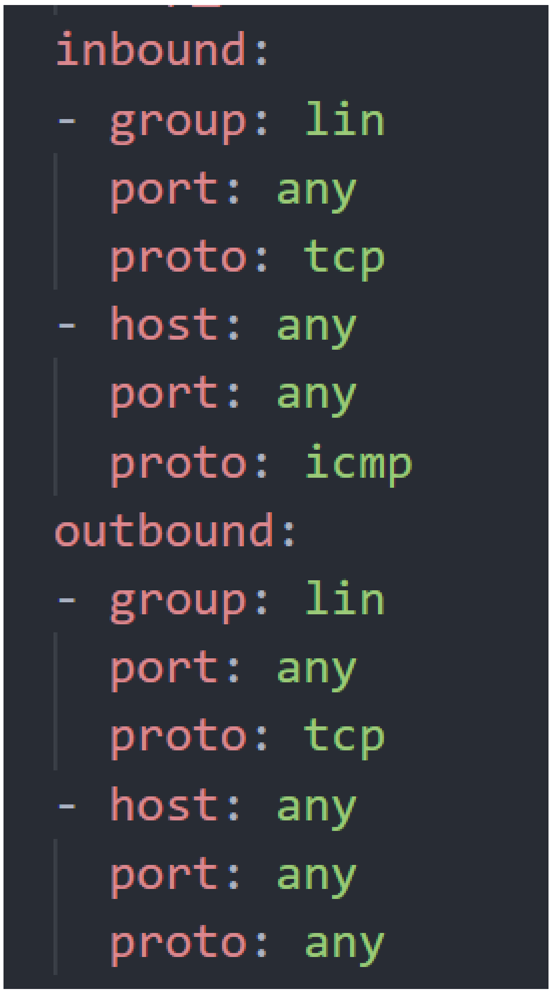
```
pi@lighthouse:~/nest$ ping 192.168.90.2
PING 192.168.90.2 (192.168.90.2) 56(84) bytes of data.
64 bytes from 192.168.90.2: icmp_seq=1 ttl=64 time=41.5 ms
64 bytes from 192.168.90.2: icmp_seq=2 ttl=64 time=59.4 ms
64 bytes from 192.168.90.2: icmp_seq=3 ttl=64 time=36.2 ms
64 bytes from 192.168.90.2: icmp_seq=4 ttl=64 time=34.5 ms
^C
--- 192.168.90.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 34.500/42.901/59.399/9.868 ms
pi@lighthouse:~/nest$ ping 192.168.90.3
PING 192.168.90.3 (192.168.90.3) 56(84) bytes of data.
64 bytes from 192.168.90.3: icmp_seq=1 ttl=128 time=34.3 ms
64 bytes from 192.168.90.3: icmp_seq=2 ttl=128 time=35.6 ms
64 bytes from 192.168.90.3: icmp_seq=3 ttl=128 time=36.2 ms
64 bytes from 192.168.90.3: icmp_seq=4 ttl=128 time=57.7 ms
^C
--- 192.168.90.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 34.258/40.933/57.730/9.721 ms
pi@lighthouse:~/nest$ ping 192.168.90.4
PING 192.168.90.4 (192.168.90.4) 56(84) bytes of data.
64 bytes from 192.168.90.4: icmp_seq=1 ttl=64 time=75.1 ms
64 bytes from 192.168.90.4: icmp_seq=2 ttl=64 time=43.3 ms
64 bytes from 192.168.90.4: icmp_seq=3 ttl=64 time=35.9 ms
64 bytes from 192.168.90.4: icmp_seq=4 ttl=64 time=39.7 ms
^C
--- 192.168.90.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 35.851/48.510/75.118/15.588 ms
```

```
/home/nest_client # echo $HOSTNAME
nest_client_lin_386
/home/nest_client # ping 192.168.90.1
PING 192.168.90.1 (192.168.90.1): 56 data bytes
64 bytes from 192.168.90.1: seq=0 ttl=64 time=61.342 ms
64 bytes from 192.168.90.1: seq=1 ttl=64 time=43.490 ms
64 bytes from 192.168.90.1: seq=2 ttl=64 time=36.768 ms
64 bytes from 192.168.90.1: seq=3 ttl=64 time=60.698 ms
^C
--- 192.168.90.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 36.768/50.574/61.342 ms
/home/nest_client # ping 192.168.90.2
PING 192.168.90.2 (192.168.90.2): 56 data bytes
64 bytes from 192.168.90.2: seq=0 ttl=64 time=260.231 ms
64 bytes from 192.168.90.2: seq=1 ttl=64 time=103.998 ms
64 bytes from 192.168.90.2: seq=2 ttl=64 time=80.732 ms
64 bytes from 192.168.90.2: seq=3 ttl=64 time=107.241 ms
^C
--- 192.168.90.2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 80.732/138.050/260.231 ms
/home/nest_client # ping 192.168.90.3
PING 192.168.90.3 (192.168.90.3): 56 data bytes
64 bytes from 192.168.90.3: seq=0 ttl=128 time=97.297 ms
64 bytes from 192.168.90.3: seq=1 ttl=128 time=85.847 ms
64 bytes from 192.168.90.3: seq=2 ttl=128 time=108.280 ms
64 bytes from 192.168.90.3: seq=3 ttl=128 time=134.976 ms
^C
--- 192.168.90.3 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 85.847/106.600/134.976 ms
/home/nest_client #
```

```
/home/nest_client # echo $HOSTNAME
nest_client_lin_64
/home/nest_client # ping 192.168.90.1
PING 192.168.90.1 (192.168.90.1): 56 data bytes
64 bytes from 192.168.90.1: seq=0 ttl=64 time=41.411 ms
64 bytes from 192.168.90.1: seq=1 ttl=64 time=58.904 ms
64 bytes from 192.168.90.1: seq=2 ttl=64 time=49.086 ms
64 bytes from 192.168.90.1: seq=3 ttl=64 time=38.031 ms
^C
--- 192.168.90.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 38.031/46.858/58.904 ms
/home/nest_client #
/home/nest_client # ping 192.168.90.3
PING 192.168.90.3 (192.168.90.3): 56 data bytes
64 bytes from 192.168.90.3: seq=0 ttl=128 time=150.738 ms
64 bytes from 192.168.90.3: seq=1 ttl=128 time=74.996 ms
64 bytes from 192.168.90.3: seq=2 ttl=128 time=101.831 ms
64 bytes from 192.168.90.3: seq=3 ttl=128 time=89.599 ms
^C
--- 192.168.90.3 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 74.996/104.291/150.738 ms
/home/nest_client # ping 192.168.90.4
PING 192.168.90.4 (192.168.90.4): 56 data bytes
64 bytes from 192.168.90.4: seq=0 ttl=64 time=104.862 ms
64 bytes from 192.168.90.4: seq=1 ttl=64 time=151.047 ms
64 bytes from 192.168.90.4: seq=2 ttl=64 time=84.320 ms
64 bytes from 192.168.90.4: seq=3 ttl=64 time=108.792 ms
^C
--- 192.168.90.4 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 84.320/112.255/151.047 ms
```

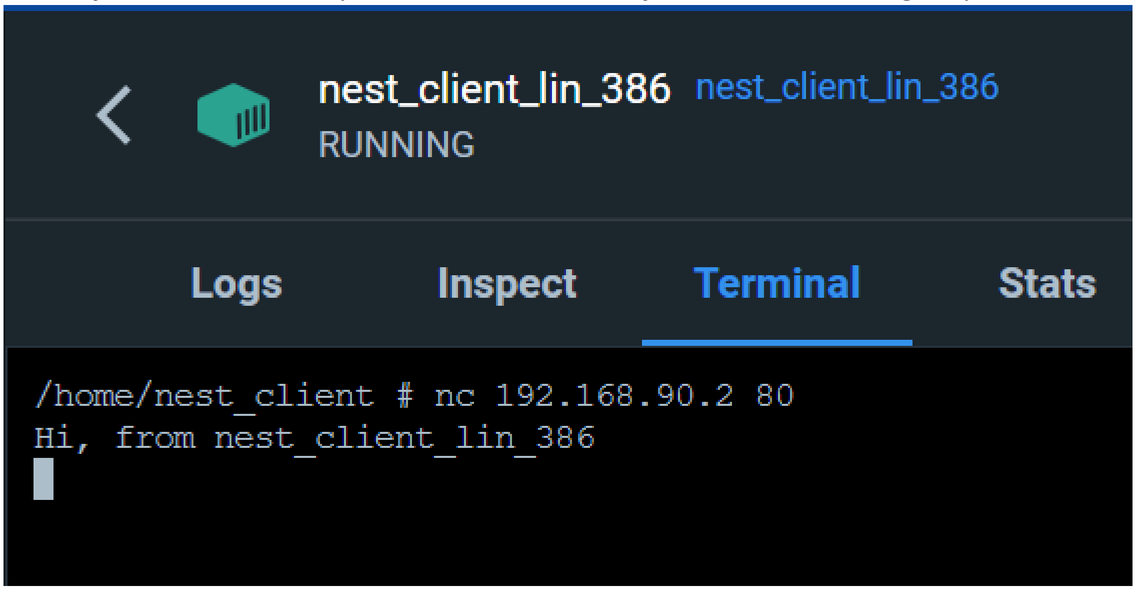
```
PS D:\Uni\Tesi\Magistrale\nest_client_win> ping 192.168.90.1
Esecuzione di Ping 192.168.90.1 con 32 byte di dati:

Risposta da 192.168.90.1: byte=32 durata=38ms TTL=64
Risposta da 192.168.90.1: byte=32 durata=51ms TTL=64
Risposta da 192.168.90.1: byte=32 durata=67ms TTL=64
Risposta da 192.168.90.1: byte=32 durata=52ms TTL=64

Statistiche Ping per 192.168.90.1:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 38ms, Massimo =  67ms, Medio =  52ms
PS D:\Uni\Tesi\Magistrale\nest_client_win> ping 192.168.90.2

Esecuzione di Ping 192.168.90.2 con 32 byte di dati:
Risposta da 192.168.90.2: byte=32 durata=474ms TTL=64
Risposta da 192.168.90.2: byte=32 durata=148ms TTL=64
Risposta da 192.168.90.2: byte=32 durata=87ms TTL=64
Risposta da 192.168.90.2: byte=32 durata=105ms TTL=64

Statistiche Ping per 192.168.90.2:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 87ms, Massimo =  474ms, Medio =  203ms
PS D:\Uni\Tesi\Magistrale\nest_client_win> ping 192.168.90.4

Esecuzione di Ping 192.168.90.4 con 32 byte di dati:
Risposta da 192.168.90.4: byte=32 durata=429ms TTL=64
Risposta da 192.168.90.4: byte=32 durata=109ms TTL=64
Risposta da 192.168.90.4: byte=32 durata=92ms TTL=64
Risposta da 192.168.90.4: byte=32 durata=110ms TTL=64

Statistiche Ping per 192.168.90.4:
    Pacchetti: Trasmessi = 4, Ricevuti = 4,
    Persi = 0 (0% persi),
Tempo approssimativo percorsi andata/ritorno in millisecondi:
    Minimo = 92ms, Massimo =  429ms, Medio =  185ms
```

Then, let's see if tcp is allowed between the Linux clients (the clients belonging to the "lin" nebula group should be allowed to communicate through tcp on any port, as per image 4)



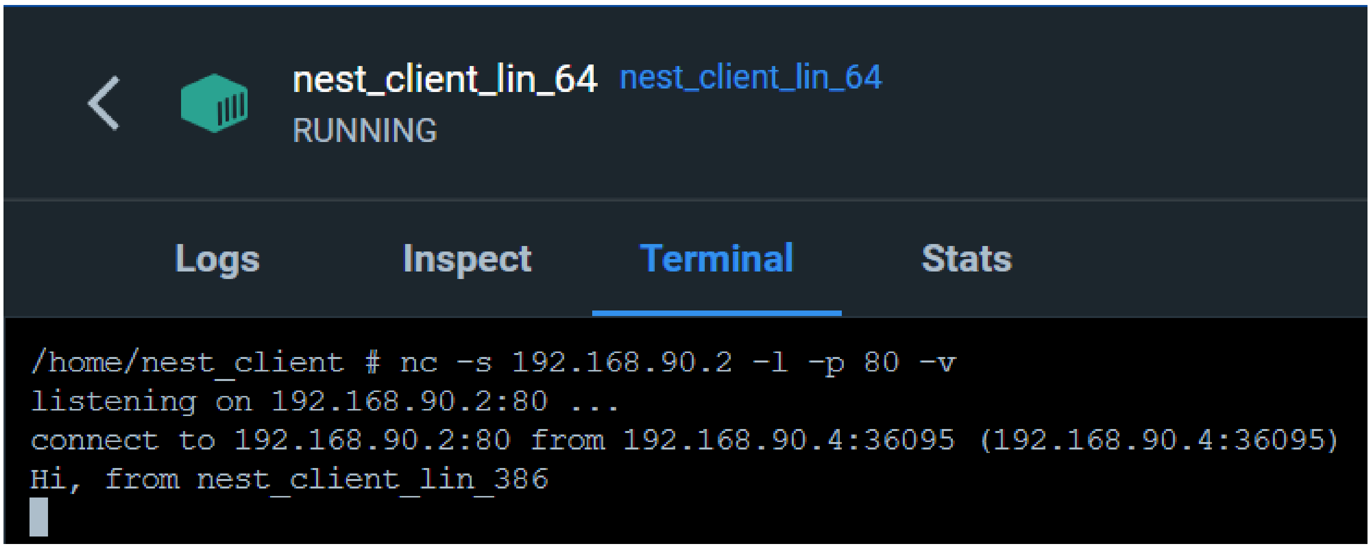
```
inbound:
- group: lin
  port: any
  proto: tcp
- host: any
  port: any
  proto: icmp
outbound:
- group: lin
  port: any
  proto: tcp
- host: any
  port: any
  proto: any
```

Image 4: firewall section of the nebula config of the linux clients. Note as ICMP traffic is allowed, and we correctly tested so, and tcp traffic is allowed only whithin the "lin" group



```
/home/nest_client # nc 192.168.90.2 80
Hi, from nest_client_lin_386
```

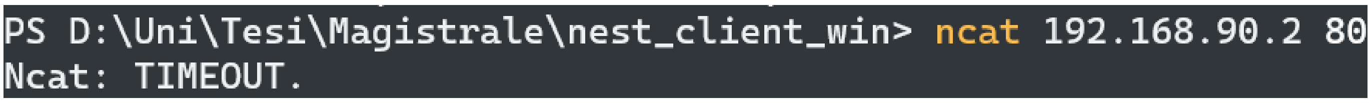
Image 5: nest_client_lin_386 sends a string via tcp connection to nest_client_lin_64.



```
/home/nest_client # nc -s 192.168.90.2 -l -p 80 -v
listening on 192.168.90.2:80 ...
connect to 192.168.90.2:80 from 192.168.90.4:36095 (192.168.90.4:36095)
Hi, from nest_client_lin_386
```

Image 6: nest_client_lin_64 correctly receives the string from nest_client_lin_386

And, to check if the firewall is correctly blocking incoming tcp from unauthorized hosts, lets see if the windows machine is correctly blocked if it tries to reach one of the linux clients on tcp port 80



```
PS D:\Uni\Tesi\Magistrale\nest_client_win> ncat 192.168.90.2 80
Ncat: TIMEOUT.
```

Image 6: nest_client_win cannot connect to the open tcp port on the nest_client_lin_64 machine, as the nebula firewall correctly prevents it