

# **NOSS | Coursework 1 :**

**Build and deployment of a virtualised network infrastructure**

|                                                                                             |           |
|---------------------------------------------------------------------------------------------|-----------|
| <b>1. Research on hypervisors :</b>                                                         | <b>4</b>  |
| Introduction                                                                                | 4         |
| Types of hypervisors                                                                        | 5         |
| Architecture of hypervisors                                                                 | 7         |
| Hardware requirements comparison                                                            | 10        |
| Advantages and Scalability                                                                  | 14        |
| Security Considerations for hypervisors                                                     | 16        |
| Conclusion : all Type 1 and Type 2 functionalities on a table                               | 17        |
| <b>2. Research on IP addressing</b>                                                         | <b>19</b> |
| Introduction                                                                                | 19        |
| IP Address Structure and Format                                                             | 20        |
| Representation of IPv4 and IPv6 Addressing                                                  | 21        |
| Subnetting and Network Organization                                                         | 23        |
| Efficient Utilisation of IP Addresses                                                       | 24        |
| <b>3. Building the Topology</b>                                                             | <b>25</b> |
| Choice of hypervisor                                                                        | 25        |
| Downloading system images (ISOs) for Virtual Machines (VMs) setup                           | 27        |
| Creating the Virtual Machines                                                               | 30        |
| Time saving protocol : Cloning Windows Server and Client VMs to get 4 VMs fully operational | 34        |
| Network configuration on host, pre-configuration on VMs :                                   | 35        |
| Setup Network NOSS Topology                                                                 | 43        |
| NOSS Topology Final Proxmox Network topology scheme                                         | 47        |
| <b>Conclusion</b>                                                                           | <b>48</b> |
| <b>Sources used for this coursework :</b>                                                   | <b>49</b> |



## **1. Research on hypervisors :**

### **Introduction**

Virtualisation is a technology that allows the creation of virtual, rather than physical, versions of digital resources such as operating systems, servers, storage devices, and network resources. It involves using software to simulate hardware functionality, creating multiple simulated environments or dedicated resources from a single physical hardware system. This technology enhances flexibility and efficiency, as it enables better resource utilisation, easier maintenance, and the ability to run different operating systems and applications on the same hardware, leading to cost savings and simplified management in IT environments.

A key component of virtualisation is the hypervisor, a software layer that enables multiple operating systems to run concurrently on a single physical machine by efficiently distributing hardware resources such as CPU, memory, and storage among different virtual machines.

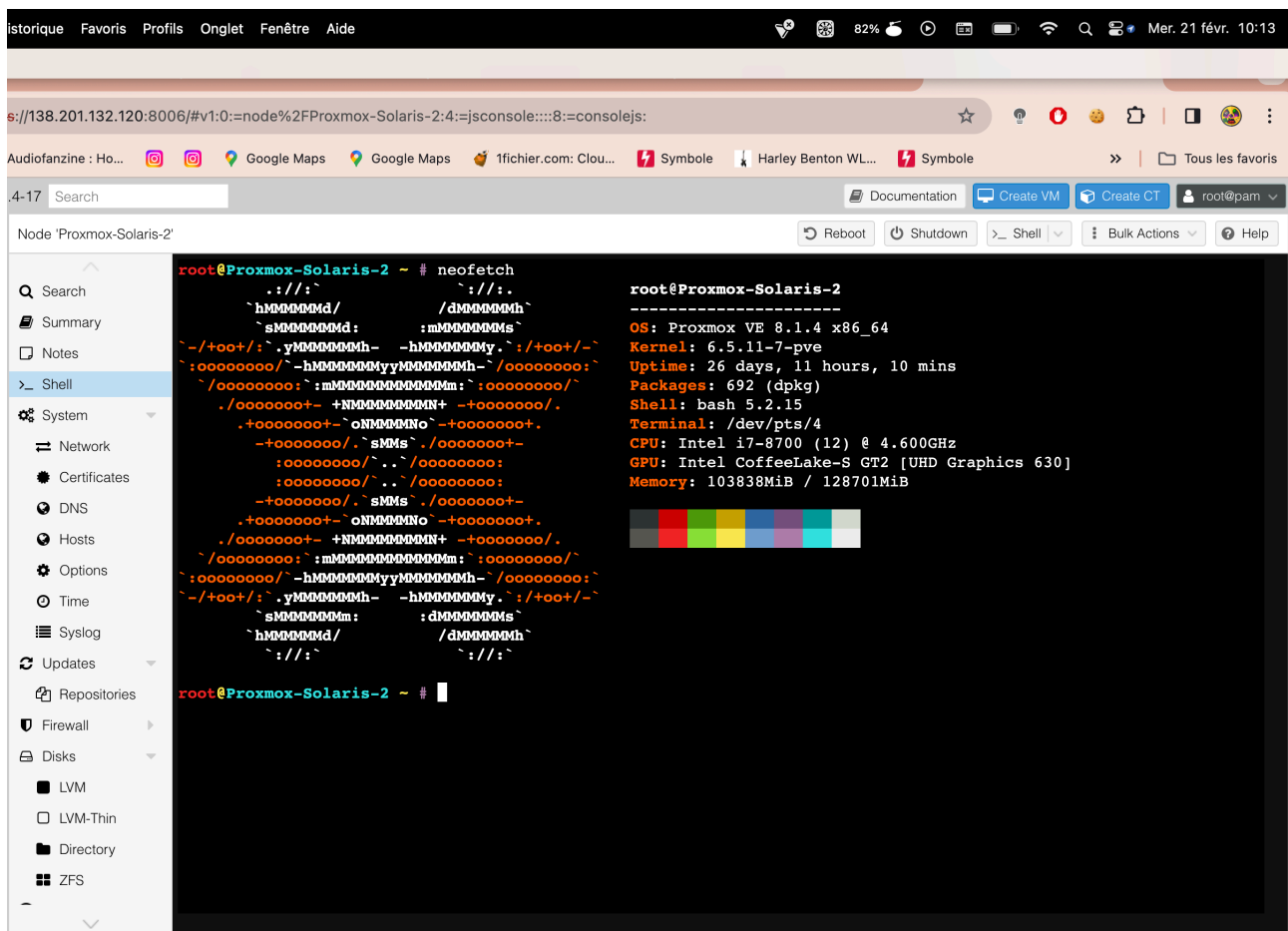
Two types of hypervisors are available on the market :

- Type 1, known as Bare-Metal supervisors, used for high availability and high scalability
- Type 2, running on the top of an existing operating system, more user friendly.

## Types of hypervisors

### - Type 1 Hypervisors :

A type 1 hypervisor is well known as a layer of software installed directly on the top of a hardware (server, supercomputer, computer for this usage only...). In some way, a type 1 hypervisor sits as the main operating system himself, or integrates one. For example, ProxMox (a type 1 hypervisor used everywhere because of its rich community and its availability, it is free) runs with a Debian (a Linux distribution) operating system, allowing to control the hypervisor as using the host machine itself. The most known type 1 hypervisors are Proxmox, VMWare ESXi, Microsoft HyperV, and Xen.



The screenshot shows the Proxmox VE 8.1 web interface. The top navigation bar includes 'historique', 'Favoris', 'Profils', 'Onglet', 'Fenêtre', and 'Aide'. The browser address bar shows 's://138.201.132.120:8006/#v1:0:=node%2FProxmox-Solaris-2:4:=jsconsole:::8:=consolejs:'. The left sidebar contains a search bar and a list of system components: Search, Summary, Notes, Shell (selected), System, Network, Certificates, DNS, Hosts, Options, Time, Syslog, Updates, Repositories, Firewall, Disks, LVM, LVM-Thin, Directory, and ZFS. The main terminal window displays the output of the 'neofetch' command, showing system information for Proxmox VE 8.1.4 x86\_64, Kernel 6.5.11-7-pve, Uptime 26 days, 11 hours, 10 mins, Packages 692 (dpkg), Shell bash 5.2.15, Terminal /dev/pts/4, CPU Intel i7-8700 (12) @ 4.600GHz, GPU Intel CoffeeLake-S GT2 [UHD Graphics 630], and Memory 103838MiB / 128701MiB. The terminal also shows a ASCII art logo for neofetch.

Proxmox VE 8.1 with Debian 12 Bookworm operating system

### - **Type 2 Hypervisors :**

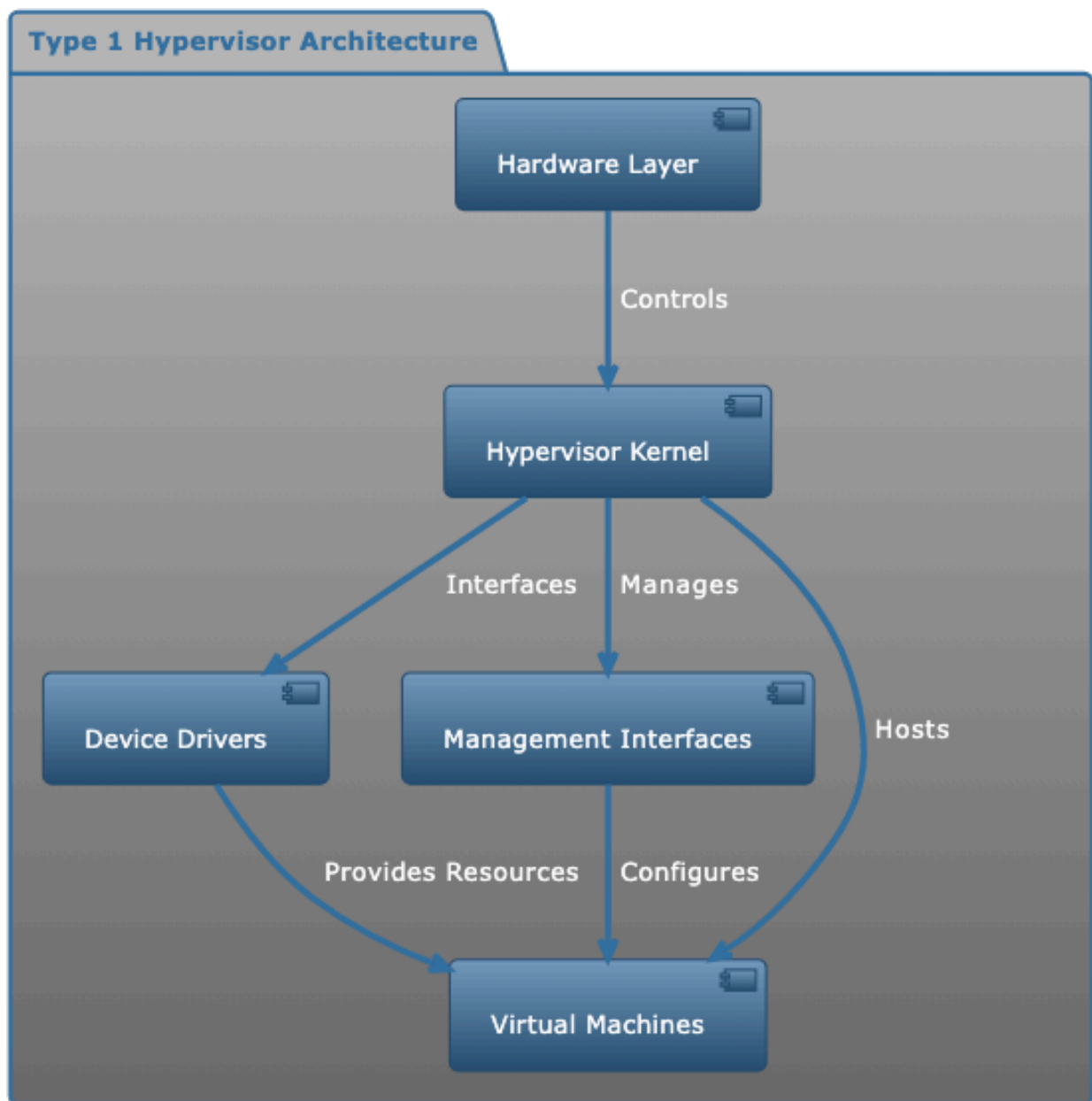
A type 2 hypervisors runs quite differently from a type 1 hypervisor. A type 2 hypervisor runs directly on the host's operating system, and is not acting as a deep system layer as type 2, the OS being running behind. Type 2 hypervisors have not a total control of the hardware like type 1 and are using the resources manager of the host's operating system. To install a type 2 hypervisor, the user just has to download a setup wizard on the internet.

The most known type 2 hypervisors are Oracle Virtualbox (open-source), VMWare Workstation, Parallels, and UTM.

## Architecture of hypervisors

### - Architecture of type 1 hypervisors :

Type 1 hypervisors run directly above the host's hardware. The kernel (a term to name the system core of an OS, or here an hypervisor) controls host's hardware. Kernel actions can be controlled by the user with interfaces. These interfaces are composed by device drivers, and different management interfaces (VM creation process, performance monitoring, network configurations...). All of these user and non-user (drivers) interfaces make up the hypervisor.

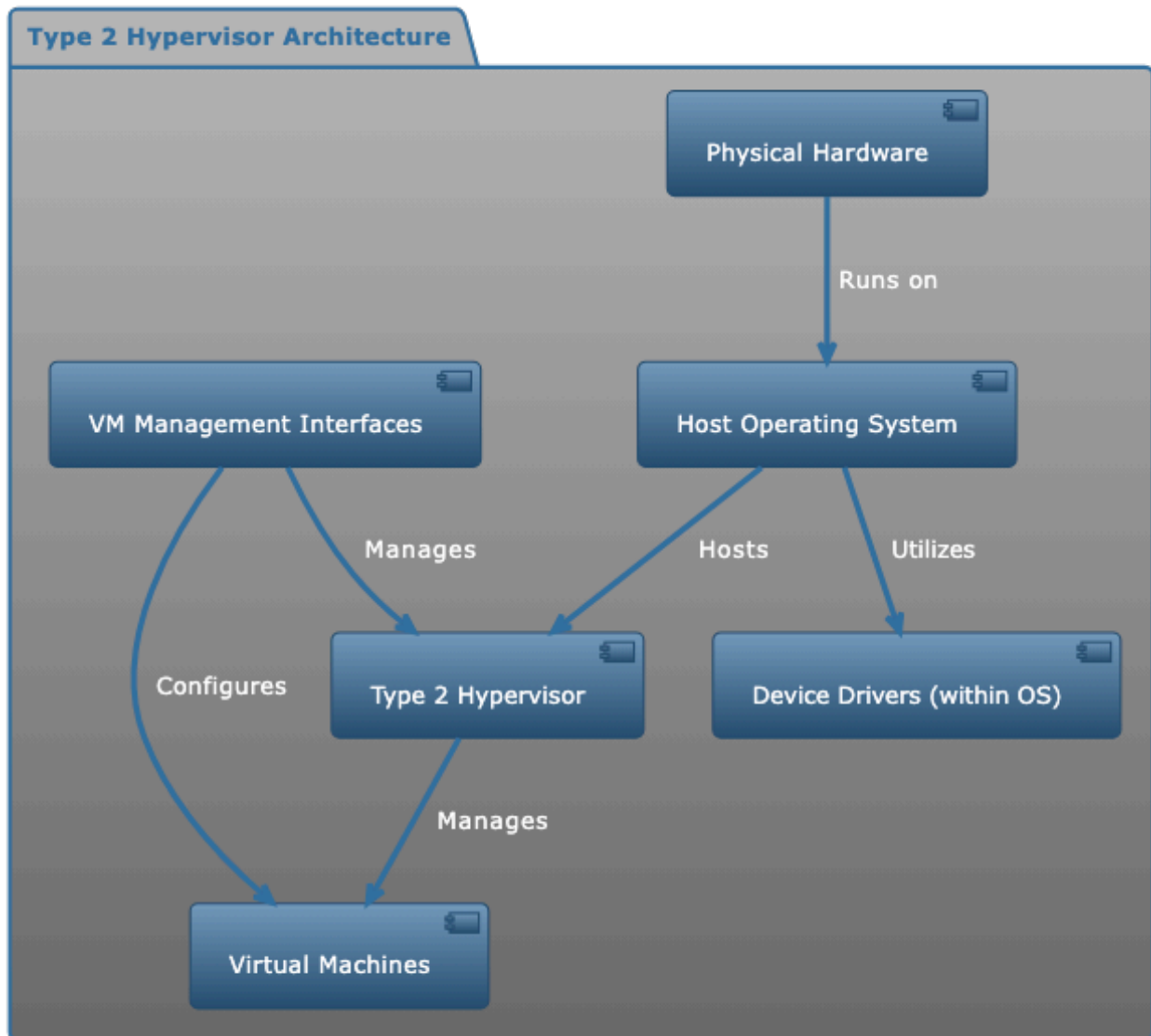


Type 1 Hypervisor Diagram, generated by GPT with Architecture description

- **Architecture of type 2 hypervisors :**

Type 2 hypervisors can be installed on host's Operating System already configured, such as Microsoft Windows, any Linux distribution, or MacOS. Then, resources and hardware management are not directly managed by the hypervisor, but by the local OS installed on the host.

For Type 2 Hypervisors, management interfaces are mixed between operating system and the hypervisor itself.



Type 2 Hypervisor Diagram, generated by GPT with Architecture description

**Conclusion on architecture overview :**

In conclusion, the architectures of Type 1 and Type 2 hypervisors represent two distinct approaches to virtualisation. Type 1 hypervisors, being installed directly on the host's hardware, offer a more efficient and direct control over the hardware resources. The kernel in these hypervisors directly interacts with the hardware, with user and system-level interfaces managing various aspects like VM creation and network configurations. This direct approach generally results in higher performance and better resource utilisation, making Type 1 hypervisors suitable for enterprise and high-performance environments.

On the other hand, Type 2 hypervisors are installed on an existing operating system, relying on the host OS for hardware resource management. This layer adds some overhead, potentially reducing efficiency compared to Type 1 hypervisors. However, Type 2 hypervisors offer greater flexibility and ease of use, particularly for users who need to run virtual environments on their existing OS without modifying the underlying hardware setup. This makes them ideal for development, testing, and educational purposes where simplicity and convenience are prioritized over raw performance.

## Hardware requirements comparison

### - For Type 1, taking Proxmox as reference :

According to proxmox and Debian official website, the resources needed to make the hypervisor running are :

- Any intel or AMD 1 Ghz processor with Intel Virtualisation technology (Intel-VT) or AMD-V, x86 or ARM64 (still in beta)
- Minimum 2 Gb of RAM
- SSD disk, minimum 25 GB of storage (Debian requires minimum 10 Gb, so we can consider that the Proxmox system will be 5 to 10 gb, plus one or two virtual machines hosting Debian or any equivalent Linux OS).
- Network Card (any / not specific)

### - For Type 2, taking VirtualBox as reference :

According to VirtualBox official website, the resources needed to make the hypervisor running are :

- Any processor with x86 architecture only (No Intel-VT or AMD-V needed)
- Minimum of 2 Gb of RAM to run at least a windows 7 VM or equivalent
- SSD disk with minimum 10 Gb available on host for a windows 8 VM or equivalent
- A supported host OS : Windows, many Linux distributions, Mac OS X (x86 ONLY, not the new ARM 64 chipsets at the moment), Solaris and OpenSolaris.

### Performances comparison :

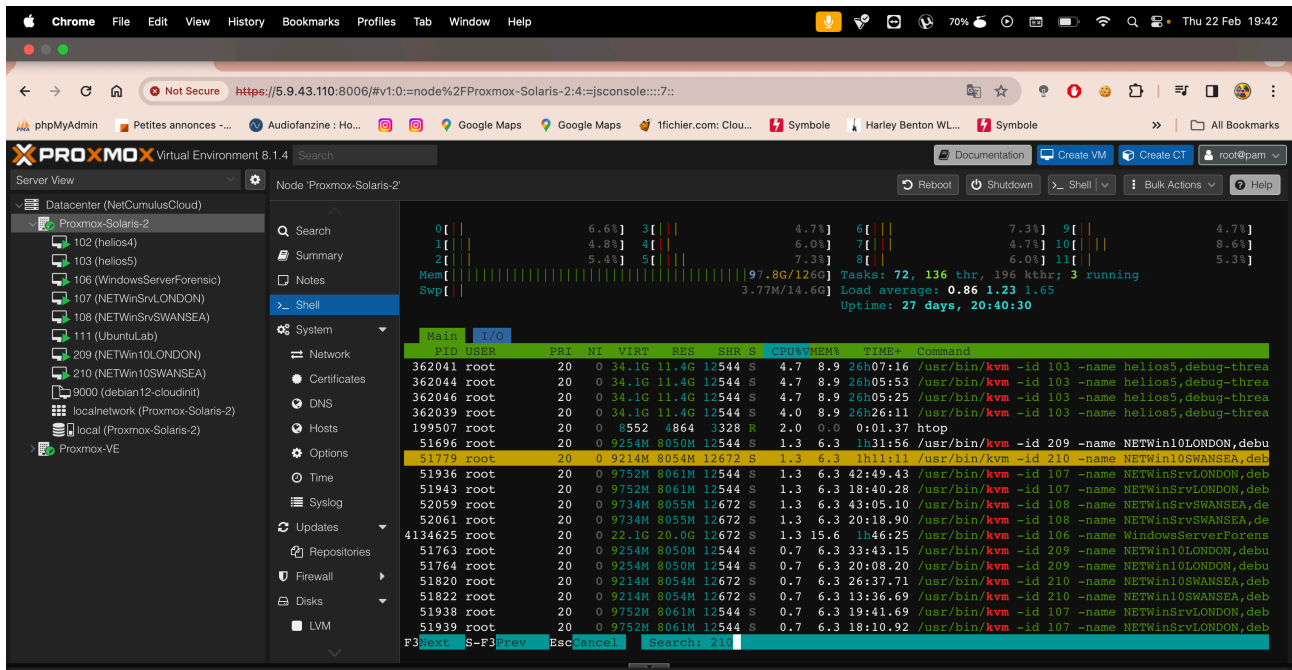
To have a better understanding about the hardware requirements and how running a type 1 or a type 2 affects the host, it has been decided to run the exact same Windows 10 Virtual machine on two hypervisors :

- Type 1 Proxmox
- Type 2 Virtualbox

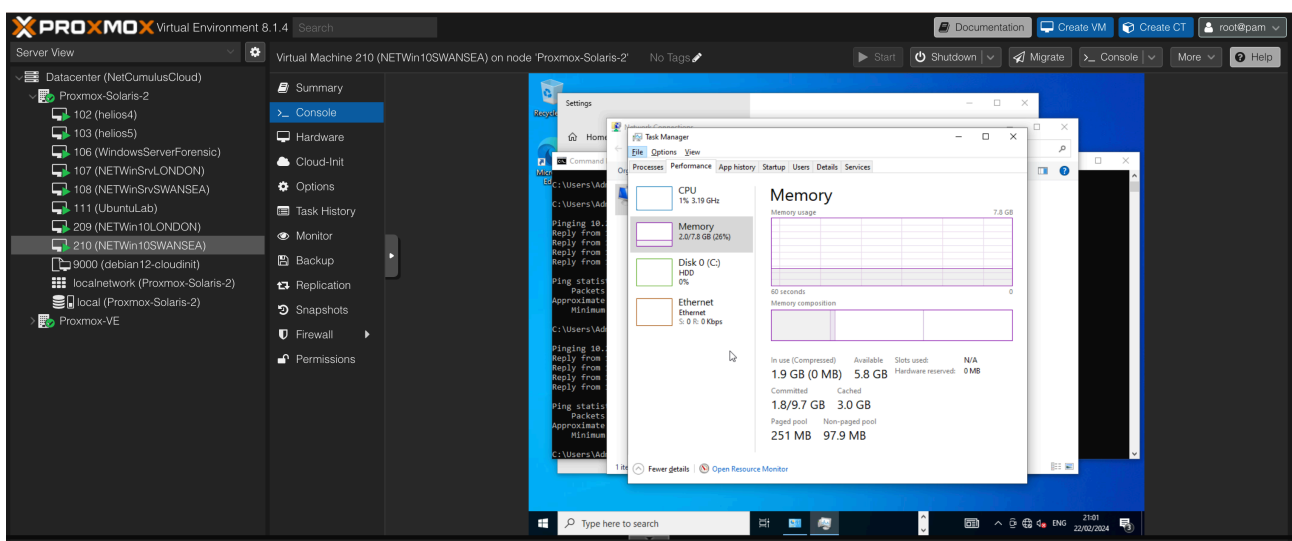
Then, we used a couple of minutes both virtual machines with same software, at the same time, with simple tasks (copying files, browsing, text edit, programs starting).

- On the type 1 hypervisor, 1.3% CPU is used, and on HTOP command, as 100% corresponds to one core used on host CPU, the server having 12 cores (1200% available), it corresponds to 0.10% of the entire processor used to run this virtual machine by doing these simple tasks. However, we see that the proxmox hypervisor allocates the 8 Gb of ram immediately, as the VM process uses 8054 Mb of ram. The VMs uses however only 2.0 Gb of ram, as it can be seen in the task manager. Proxmox guests (VMs) can be configured by using memory ballooning

technology (the allocated memory grows up til the limit set by the user only if it needs to, and goes down if all it is not used), but it is not used by default on Proxmox virtual machines.

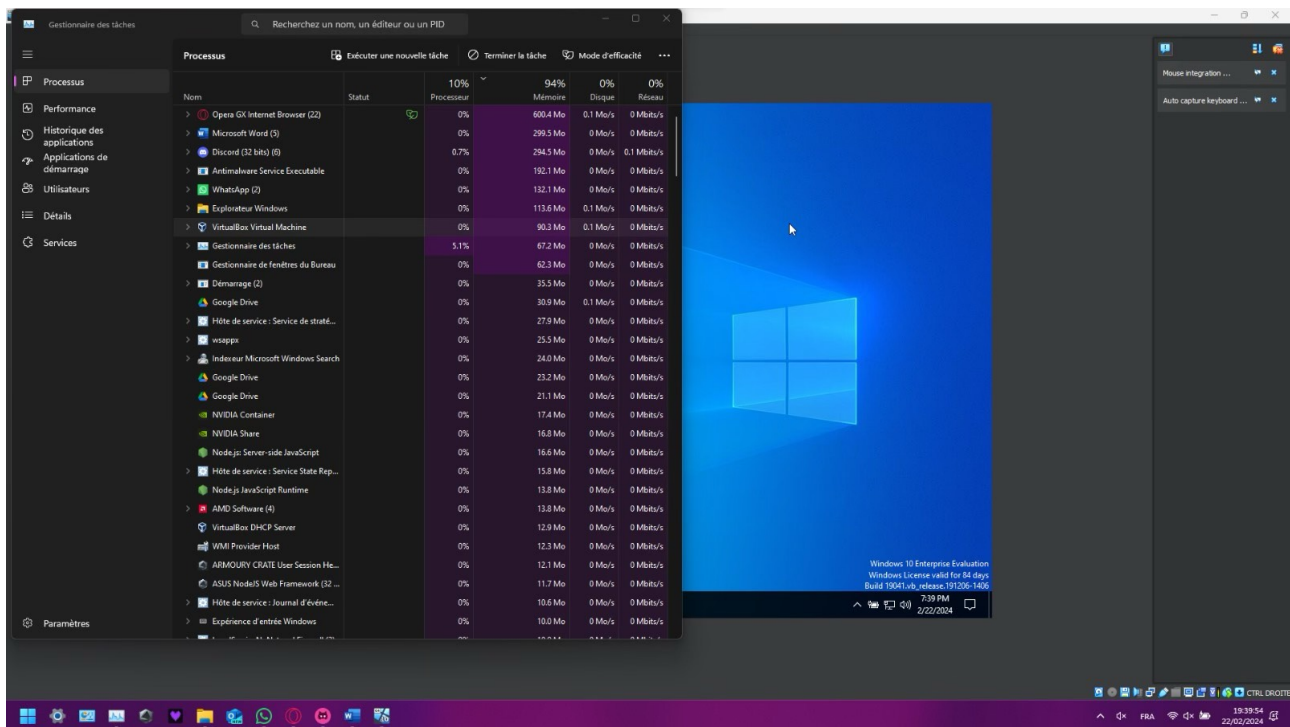


VM 210 (Windows 10 Client) hardware usage, yellow line, Proxmox host HTOP

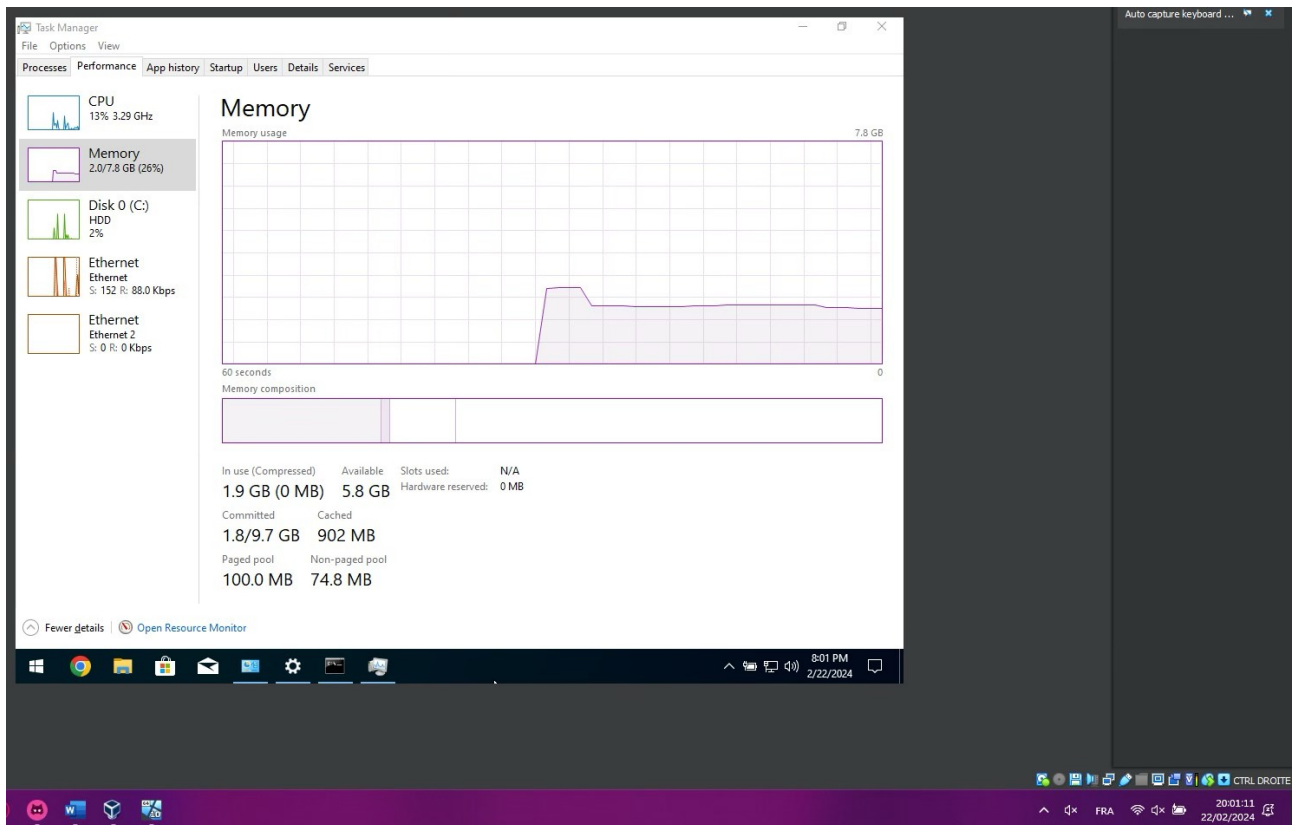


Proxmox VM ID 210 real RAM usage on the guest system, task manager

- On the type 2 hypervisor, on windows task manager, virtual box seems to use less than 1% of the global CPU. It means that there is no noticeable difference in the way that virtualbox uses CPU for VMs compared to Proxmox VMs. However, it uses less memory, with only 90.3 Mb of RAM used. It means that it probably uses a memory ballooning technology by default, got from the OS kernel. However, the VMs uses itself exactly the same amount of ram than on Proxmox, as it can be seen on the task manager (2.0 Gb of RAM).



Hardware resources used by Virtualbox on guest, host OS task manager.



Virtualbox Windows 10 VM, real RAM usage on the guest system, task manager, same values results than on the Type 1 hosted VM.

We can conclude both hypervisors deliver similar performances on virtual machines. However, the CPU activity is lower on the Proxmox host, due to the lack of OS user operations comparing to the Windows laptop host (Opera GX, Discord, and the OS itself take between 10 to 30% constantly on laptop's hardware capacity).

## **Advantages and Scalability**

### **Advantages of using a Type 1 Hypervisor :**

- Without any OS supervision like it is the case on type 2 hypervisors, type 1 hypervisors can provide a total control of the hardware, allowing a better ratio of VMs per hosted per machine (no resources used by an OS let more resources available to create and manage VMs)
- Type 1 hypervisors are useful for high availability infrastructures, being extremely stable without any risks of OS errors or instabilities, because the hypervisor is the only operating system (OS).
- A great part of Type 1 hypervisors are capable of working in node groups —> Clusters. For example, multiple servers with the same hypervisors (for example, proxmox) can be connected each other to obtain a fully redundant system, with backup or virtual machine replication, to prevent any blackout, or simply to share out / split the global system load average of an infrastructure.
- Type 1 hypervisors contain a lot of debug resources, scripts, programs, patches, updates, tutorials, established by communities, or the companies themselves (for example, proxmox has its own community forum, <https://forum.proxmox.com/>).
- Templates available to deploy VMs in just a couple of minutes

### **Advantages of using a Type 2 Hypervisor :**

- Interfaces are intuitive and visually designed to be easy to handle by beginners —> User Friendly
- No need to uninstall or reconfigure actual host's OS
- Fast installation, just a setup wizard,

### **Scalability :**

**Type 1 hypervisors** are engineered to offer robust scalability and flexibility, inherently free from the risks of operating system errors or instabilities since the hypervisor itself functions as the sole operating system. This architecture enhances system reliability and efficiency. Many Type 1 hypervisors, such as Proxmox, are adept at operating in node groups or clusters. When configured in such a manner, multiple servers running the same hypervisor can be interconnected to create a fully redundant system. This setup enables functionalities like backup and virtual machine replication, crucial for preventing downtime or system blackouts. Additionally, it facilitates the distribution or balancing of the overall system load across the infrastructure, optimizing

performance and resource utilisation. Scalability is the also greatest advantage of a type 1 hypervisor, as seen previously.

**Type 2 hypervisors**, operating atop a host operating system, exhibit a different approach to scalability compared to Type 1 hypervisors. Their scalability is inherently linked to the capabilities and limitations of the underlying host OS. While they may not offer the same level of resource efficiency and performance scalability as Type 1 hypervisors, Type 2 hypervisors are still quite adaptable for scaling up virtual environments within the constraints of the host system.

This makes them suitable for use cases where the scalability demands are moderate and do not require extensive direct hardware manipulation. In scenarios like software development, testing, and education, where the flexibility to run multiple and varied virtual environments on a single physical machine is more critical than raw performance, Type 2 hypervisors provide a practical and scalable solution. However, for large-scale, resource-intensive applications, their reliance on the host OS can be a limiting factor in scalability.

## **Security Considerations for hypervisors**

Type 1 hypervisors, like Proxmox, offer enhanced security by operating directly on hardware, minimising exposure to vulnerabilities common in operating systems. They benefit from isolation between VMs and direct hardware interaction. However, they must be vigilantly managed and updated to address hardware-level vulnerabilities and maintain their secure posture. Type 1 hypervisors like proxmox offers powerful pre-installed security functions, as an integrated fully-modifiable smart firewall, auto update protocols, user-access management (with role permissions, and groups), datacenter-scale protections (if a cluster scheme is used with multiple proxmox nodes connected each other).

Conversely, Type 2 hypervisors, such as Oracle VirtualBox, rely on the security of the host operating system. While this adds a layer of dependency and automatisms, it also allows them to leverage the host's full suite of security tools and updates, including firewalls and anti-malware programs. The security of Type 2 hypervisors is as strong as the host system's security measures, and not fully customisable.

## Conclusion : all Type 1 and Type 2 functionalities on a table

| Aspect                                | Type 1 Hypervisor                                                                                           | Type 2 Hypervisor                                                                                              |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>Installation</b>                   | Installed directly on hardware.                                                                             | Installed on an existing operating system.                                                                     |
| <b>Architecture</b>                   | Runs above the host's hardware. Kernel controls hardware directly.                                          | Runs on host's OS. Relies on the host OS for hardware resource management.                                     |
| <b>Hardware Interaction</b>           | Direct interaction with hardware.                                                                           | Indirect interaction through the host OS.                                                                      |
| <b>Performance</b>                    | Generally higher performance due to direct hardware control.                                                | Potentially lower performance due to additional OS layer.                                                      |
| <b>Resource Utilization</b>           | Efficient resource utilization, suitable for high-performance environments.                                 | Less efficient due to overhead of host OS.                                                                     |
| <b>Scalability</b>                    | Highly scalable, suitable for large-scale operations. Clusters for redundancy and load balancing.           | Scalability depends on host OS capabilities. Suitable for moderate scalability demands.                        |
| <b>Security</b>                       | Enhanced security due to direct hardware operation. Needs vigilant management.                              | Security depends on the host OS. Leverages host's security tools.                                              |
| <b>Use Cases</b>                      | Ideal for high availability infrastructures, enterprise environments, and where high scalability is needed. | Suitable for development, testing, educational purposes, and where ease of use is prioritized.                 |
| <b>Examples</b>                       | Proxmox, VMWare ESXi, Microsoft HyperV, Xen.                                                                | Oracle VirtualBox, VMWare Workstation, Parallels, UTM.                                                         |
| <b>R e s o u r c e Requirements</b>   | Higher resource needs (e.g., for Proxmox, 1 GHz CPU with VT support, 2 GB RAM, 25 GB SSD).                  | Lower resource needs (e.g., for VirtualBox, any x86 processor, 2 GB RAM, 10 GB on host SSD).                   |
| <b>M a n a g e m e n t Interfaces</b> | Advanced interfaces for VM management, performance monitoring, network configurations.                      | More user-friendly, intuitive interfaces.                                                                      |
| <b>Advantages</b>                     | Total control of hardware, high VM to host ratio, stable, supports clusters, robust debug resources.        | User-friendly interfaces, no need to alter host OS, quick installation, suitable for personal/small-scale use. |



## **2. Research on IP addressing**

### **Introduction**

In the dynamic world of digital communication, the role of IP (Internet Protocol) addressing is foundational and multifaceted. IP addresses serve as unique identifiers for each device (known as host) on a network, enabling them to communicate effectively over the global network. This report delves into the intricacies of IP addressing, discussing its structure, the transition from IPv4 to the more advanced IPv6, the concept of subnetting, and strategies for efficient utilisation.

## **IP Address Structure and Format**

### *Understanding IP Addresses*

An IP address is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. Its primary function is to identify the host or network interface and provide the location of the host in the network, thus facilitating the routing of internet traffic, letting it be more efficient and fast.

### *IPv4 vs. IPv6 Structures*

IPv4 (known as Internet Protocol Version 4), is the first Internet Protocol to be majorly deployed. It utilises a 32-bit address scheme, traditionally represented in decimal format as four numbers separated by periods (e.g., 192.168.0.1). It has been thought to be simple to memorise and type. However, due to the exponential growth of the internet and the scarcity of IPv4 addresses (limit of 32-bit Internet Protocol is 4 billion unique addresses), IPv6 was introduced, and solve the problem of IPv4 rarity. IPv6 addresses are 128-bit, significantly increasing the number of available addresses (trillions of unique identifiers...). They are typically represented in hexadecimal format, divided by colons (e.g., 2001:0db8:85a3:0000:0000:8a2e:0 370:7334).

### *Binary and Decimal Representation*

At their core, both IPv4 and IPv6 addresses are binary numbers. IPv4 addresses are often represented in decimal form for ease of use, whereas IPv6 addresses use hexadecimal representation to make them more human-readable due to their length. The binary representation underscores the fundamental nature of computing, where data transmission occurs in this binary form.

## Representation of IPv4 and IPv6 Addressing

### IPv4 Structure and Representation

IPv4 addresses, composed of 32 bits, are divided into four octets, each representing a byte (8 bits). The format, known as 'dotted decimal', separates these octets with periods. For example, in the address 192.168.1.1, each number between the periods represents an eight-bit binary number. IPv4 addresses are further categorised into different classes (A, B, C, D, and E) based on the network size and number of hosts they can accommodate, impacting how the address space is allocated.

| bit --> | 0               | 31              | Address Range               |                             |                             |                             |
|---------|-----------------|-----------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 0       | CLASS A ADDRESS |                 | 0.0.0.0 – 127.255.255.255   |                             |                             |                             |
| 1       | 0               | CLASS B ADDRESS | 128.0.0.0 – 191.255.255.255 |                             |                             |                             |
| 1       | 1               | 0               | CLASS C ADDRESS             | 192.0.0.0 – 223.255.255.255 |                             |                             |
| 1       | 1               | 1               | 0                           | CLASS D ADDRESS             | 224.0.0.0 – 239.255.255.255 |                             |
| 1       | 1               | 1               | 1                           | 0                           | RESERVED ADDRESS            | 240.0.0.0 – 247.255.255.255 |

### IPv6 Structure and Representation

IPv6, the successor to IPv4, is using 128 bits, allowing for a vastly greater number of unique addresses. Its hexadecimal representation, separated by colons, is more efficient for expressing large addresses. For instance, 2001:0db8:85a3:0000:0000:8a2e:0370:7334 is an IPv6 address. IPv6 also introduces features like address auto configuration and simplified header format, enhancing the protocol's efficiency and scalability.

### Efficiency and Capacity Comparison

The transition from IPv4 to IPv6 is primarily driven by the need for more address space, but it also offers improvements in routing efficiency, security, and address management. IPv6's larger address pool significantly reduces the need for techniques like Network Address Translation (NAT : NAT stands for network address translation. It's a way to map multiple private addresses inside a local network to a public IP address before transferring the information onto the internet.), enabling more direct and efficient routing of internet traffic.

## **Subnetting and Network Organization**

### *Concept of Subnetting*

Subnetting is a method used to divide a single IP network into multiple smaller networks, enhancing routing efficiency and network management. It involves partitioning the host part of the IP address into a network and a sub-network, allowing for finer control over IP address allocation and reducing broadcast domains (A broadcast domain is a logical area of a computer network where any computer connected to the network can transmit directly to all other computers in the same domain, without having to go through a router.).

### *Techniques in IPv4 and IPv6*

In IPv4, subnetting is achieved by manipulating the subnet mask, a 32-bit number that separates the IP address into network and host portions. IPv6 subnetting, while similar in concept, deals with a larger address space, allowing for more hierarchical and flexible network design.

## **Efficient Utilisation of IP Addresses**

### *Challenges in Address Allocation*

The growth of the internet has led to challenges in IP address allocation, particularly in IPv4, where address exhaustion is a significant concern. The only solution is to limit the using of the v4 protocol, or migrate progressively networks to v6 protocol.

Techniques like CIDR (Classless Inter-Domain Routing) in IPv4 help optimise address allocation by allowing more flexible division of address spaced (/24 /28 ...). In IPv6, the vast address space itself mitigates many of these challenges (because of auto-configuration, for example)

### *Role of Subnetting and Address Management*

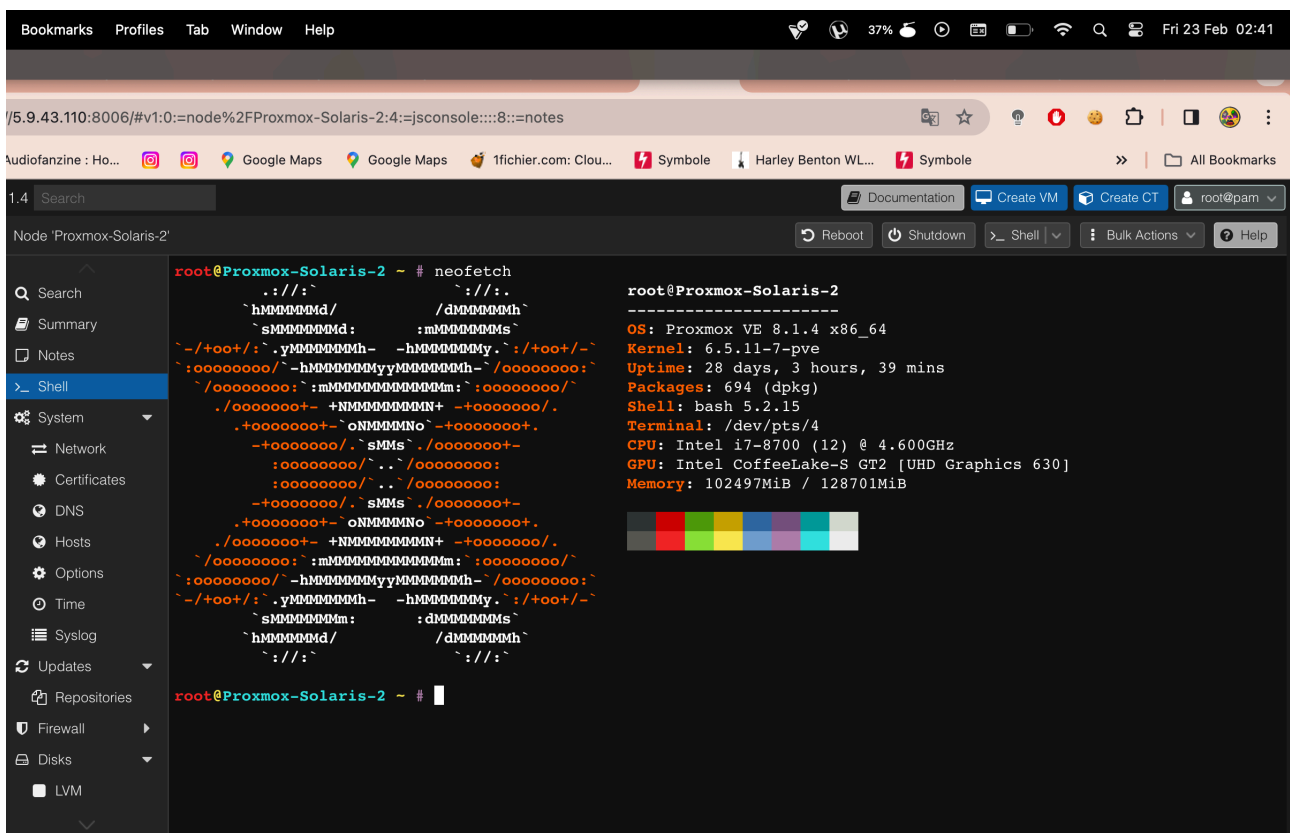
Subnetting lets optimising network performance by reducing routing table complexity and improving address allocation efficiency. Proper IP address management, including the adoption of IPv6, helps to maintain a stable network architecture.

### 3. Building the Topology

#### Choice of hypervisor

For this coursework, one decided to choose Proxmox, a type 1 hypervisor, for several reasons :

- The host used in local was not compatible with VMWare or Virtualbox type 2 hypervisors (recent Apple powered computers use ARM64 Central Processing Units architectures (CPUs). VMWare and Virtualbox are not available on this architecture at the moment.
- Experimental servers, hosted on Hetzner, were available for a cheap prize. Combined to hosting activity, investment is amortised for these machines.
- Proxmox was available immediately, as the server was installed before this coursework, and fully operational. One « just » had to find out how to configure the network topology and isolate it.
- Strong base knowledge in system admin, proxmox systems, and linux distributions.



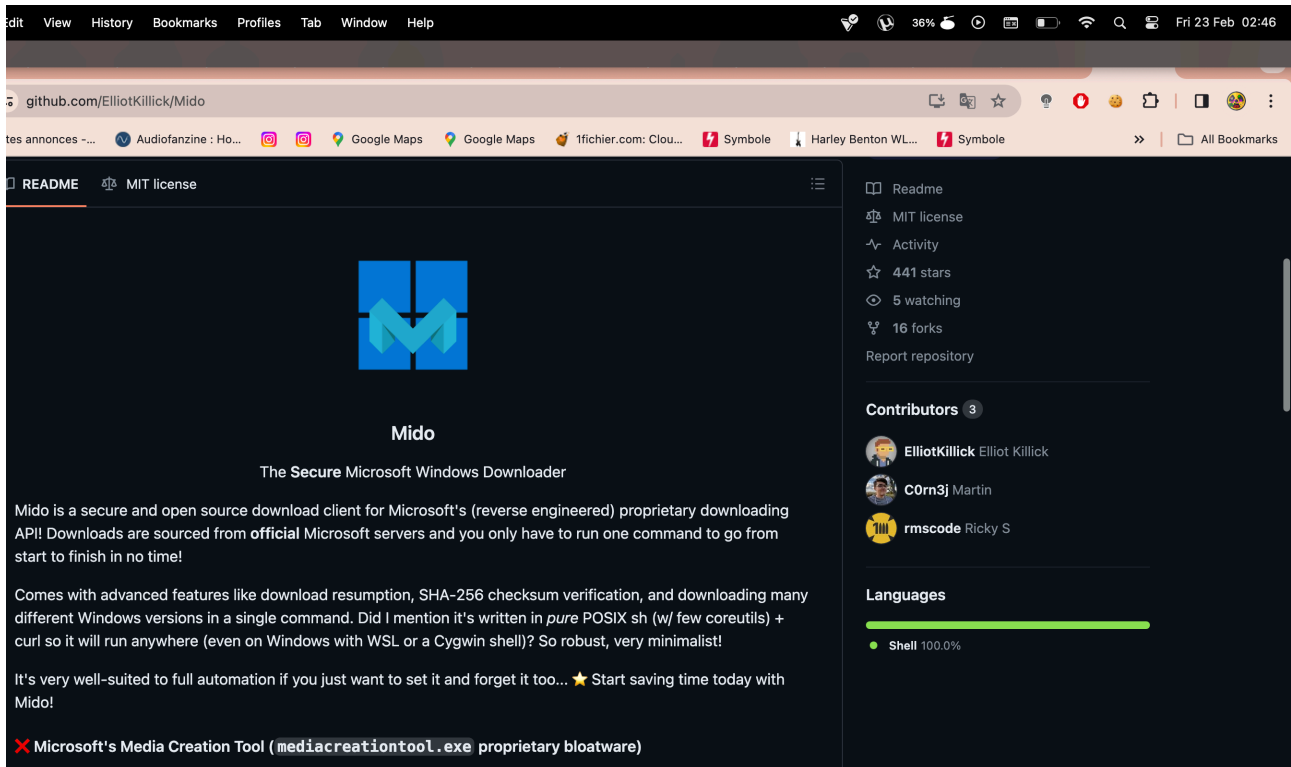
The screenshot shows a terminal window titled 'Node: Proxmox-Solaris-2'. The user has run the command 'neofetch', which displays a stylized ASCII art logo for 'Proxmox-Solaris-2'. To the right of the logo, system information is listed: OS: Proxmox VE 8.1.4 x86\_64, Kernel: 6.5.11-7-pve, Uptime: 28 days, 3 hours, 39 mins, Packages: 694 (dpkg), Shell: bash 5.2.15, Terminal: /dev/pts/4, CPU: Intel i7-8700 (12) @ 4.600GHz, GPU: Intel CoffeeLake-S GT2 [UHD Graphics 630], and Memory: 102497MiB / 128701MiB. The terminal also shows a sidebar with system management options like Network, Certificates, DNS, Hosts, Options, Time, Syslog, Updates, Repositories, Firewall, and Disks.

The proxmox now installed, running since 28 days, fully operational for network topology setup.

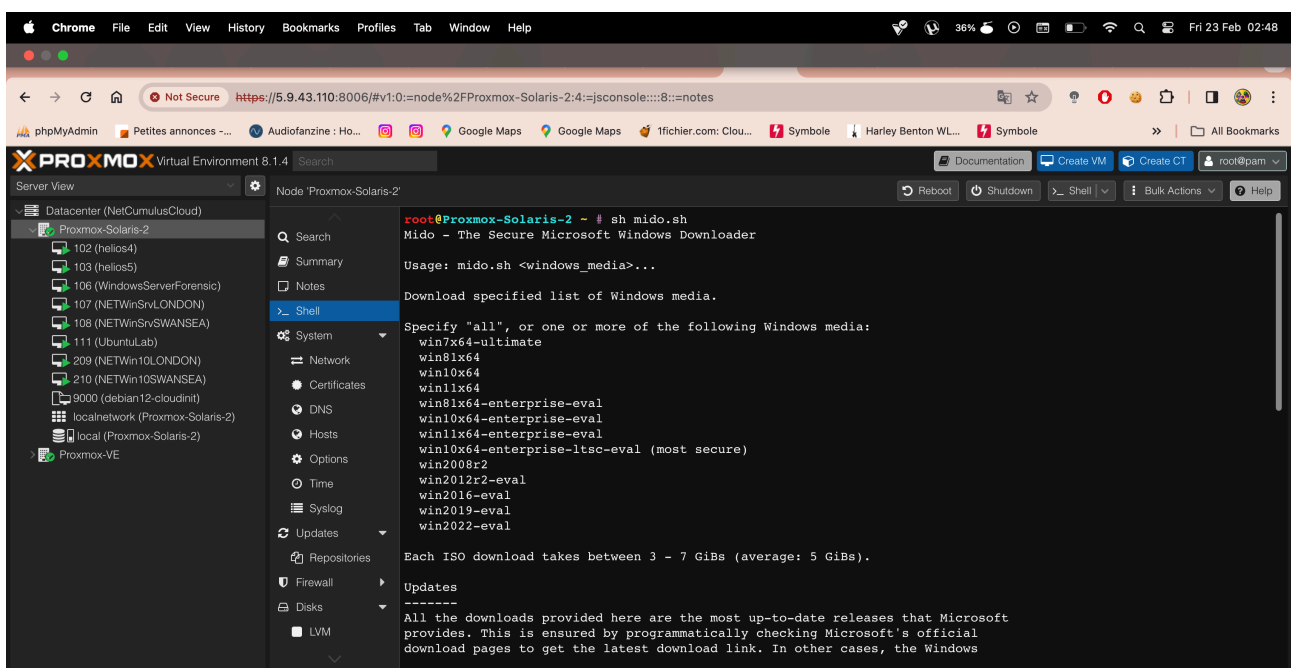


## Downloading system images (ISOs) for Virtual Machines (VMs) setup

To get ISOs faster, without downloading them to a local PC, then importing them in the server by using SFTP (very slow), one had to search for a bash tool to use on the node console.



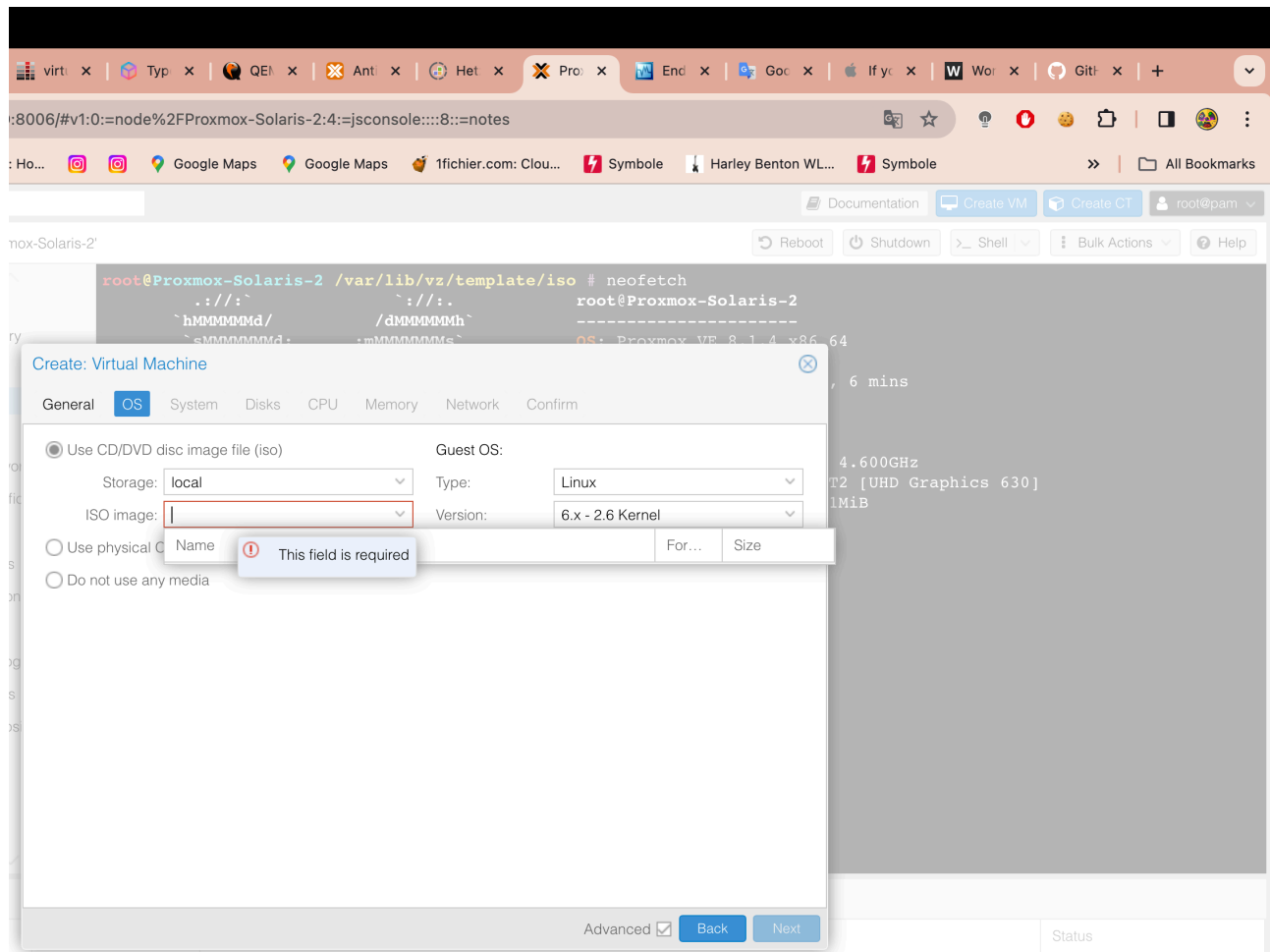
Mido, a tool found on GitHub to easily get ISOs from Microsoft servers API



Launching Mido on proxmox node, and downloading ISOs.

## ISOs troubleshooting : finding how to import them to a new VM :

After downloading the ISO in the root folder, it was not available on the list to get it installed.



No ISO visible on the menu

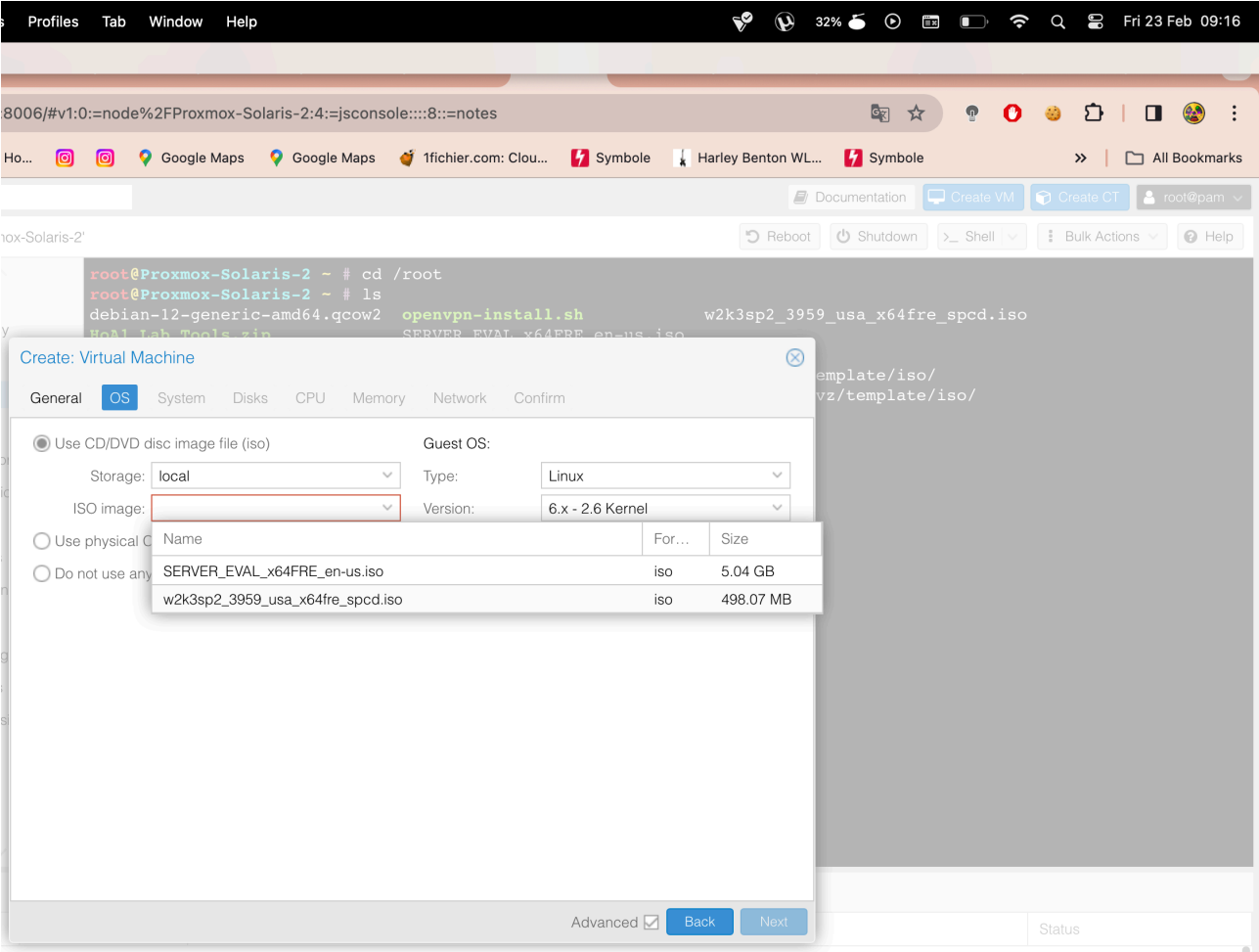
After some research, it appeared that all the downloaded ISOs had to be in the following directory :

- /var/lib/vz/template/iso

```
root@Proxmox-Solaris-2 ~ # cd /root
root@Proxmox-Solaris-2 ~ # ls
debian-12-generic-amd64.qcow2  openvpn-install.sh  w2k3sp2_3959_usa_x64fre_spcd.iso
HoAl_Lab_Tools.zip            SERVER_EVAL_x64FRE_en-us.iso
mido.sh                      Solaris-2.ovpn
root@Proxmox-Solaris-2 ~ # mv SERVER_EVAL_x64FRE_en-us.iso /var/lib/vz/template/iso/
root@Proxmox-Solaris-2 ~ # mv w2k3sp2_3959_usa_x64fre_spcd.iso /var/lib/vz/template/iso/
root@Proxmox-Solaris-2 ~ # date
Fri Feb 23 10:14:32 AM CET 2024
root@Proxmox-Solaris-2 ~ #
```

Moving ISOs to the good directory

Now, both Windows Server and Windows 10 ISOs were visible on the VM creation process.



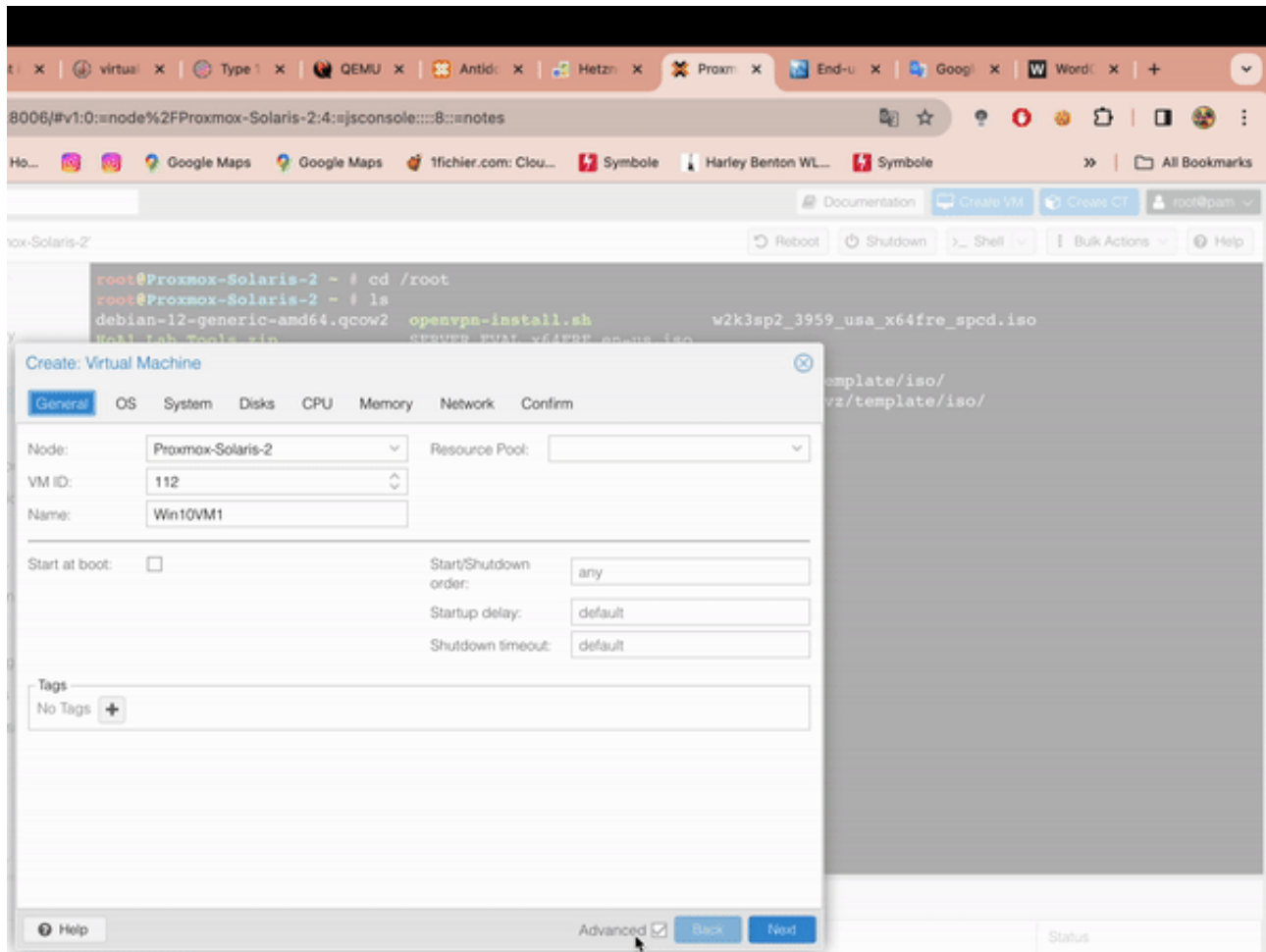
ISOs found by proxmox

## Creating the Virtual Machines

For this topology, one had to setup 4 virtual machines :

- Two Windows Server 2022
- Two Windows 10 (Client)

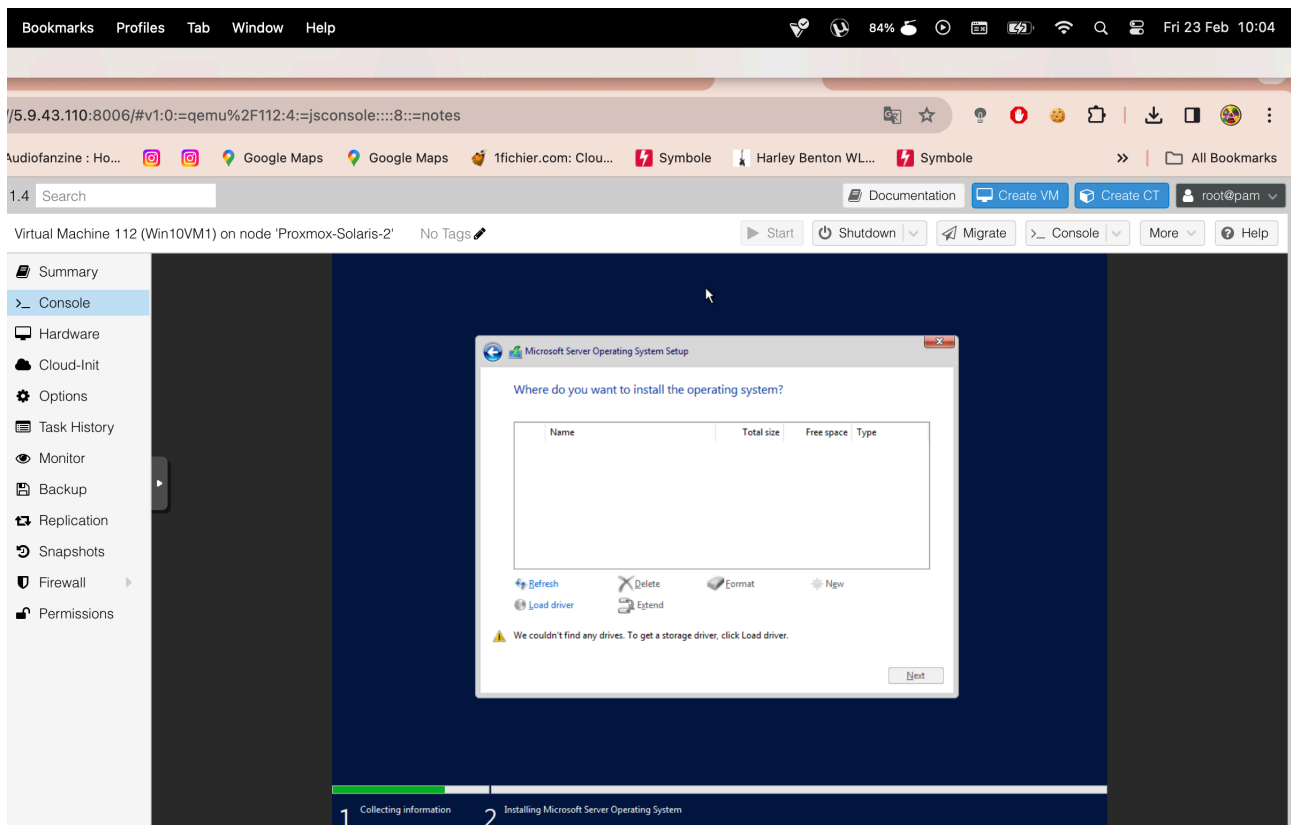
Each VM is configured by 50 to 100 Gb of VirtIO virtual disk, 8 Gb of memory, and 4 to 8 cores of CPU.



VM creation process

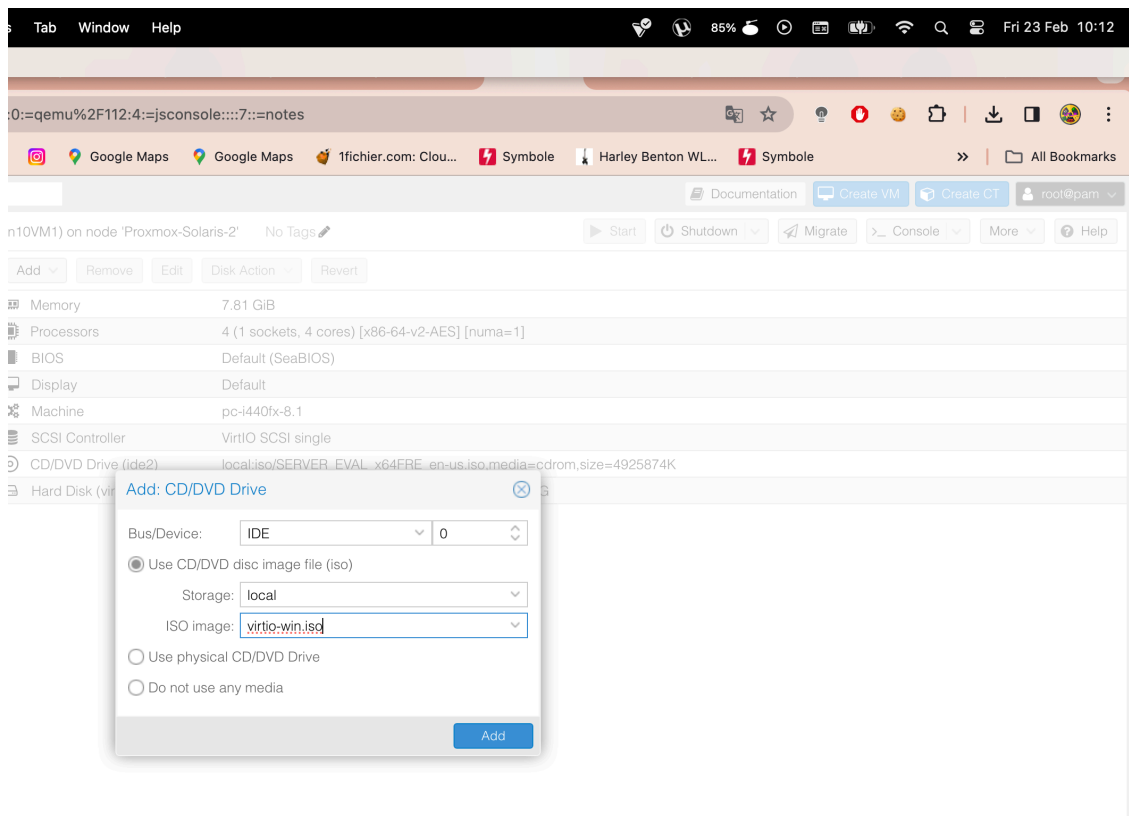
### Troubleshoot : Virtual Disk not detected on Windows Setup, missing VirtIO Driver :

When arriving on the disk selection section on both windows setups (Server or Client), no partition was detected.

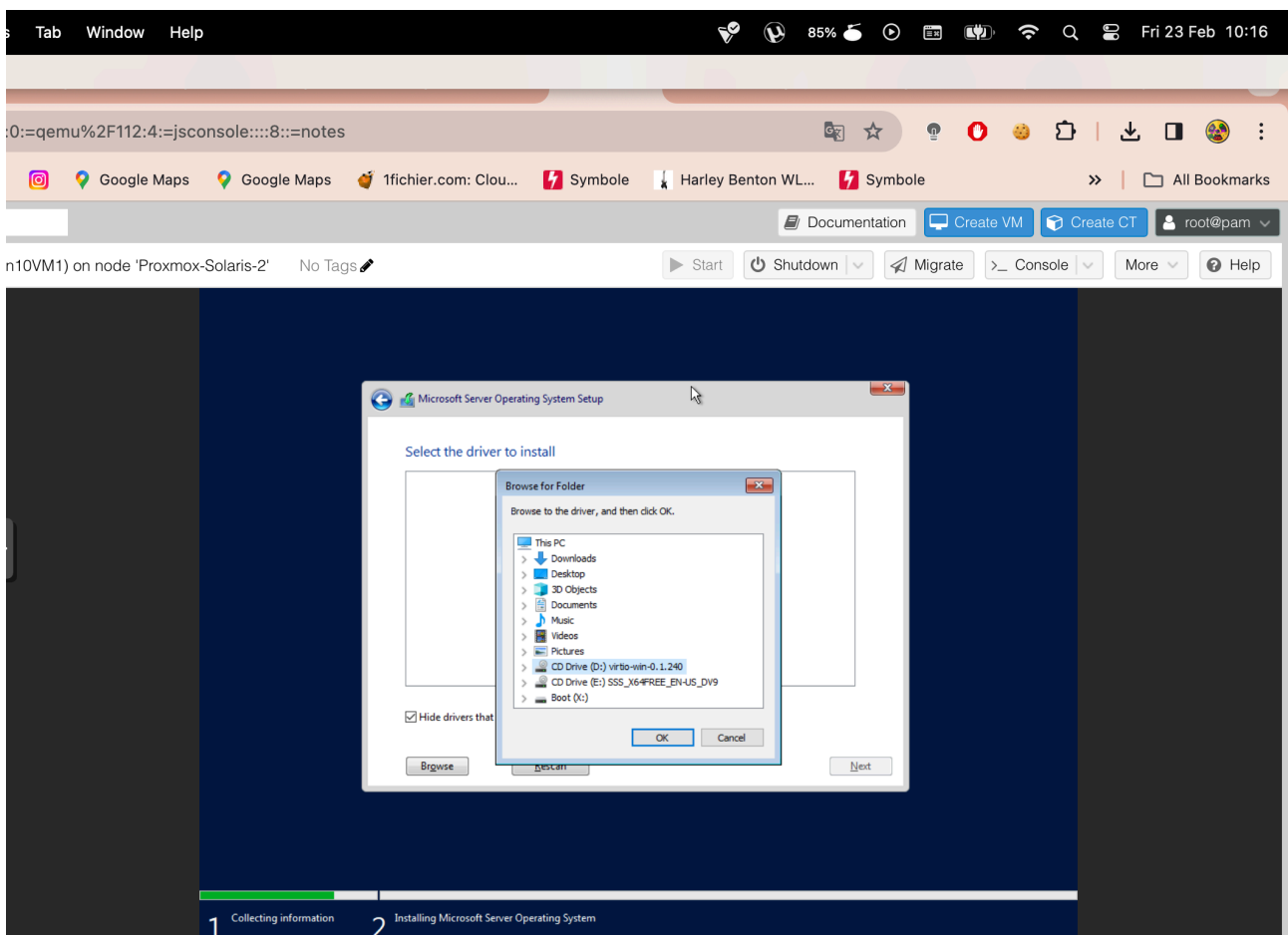


Windows Server installation : no disk

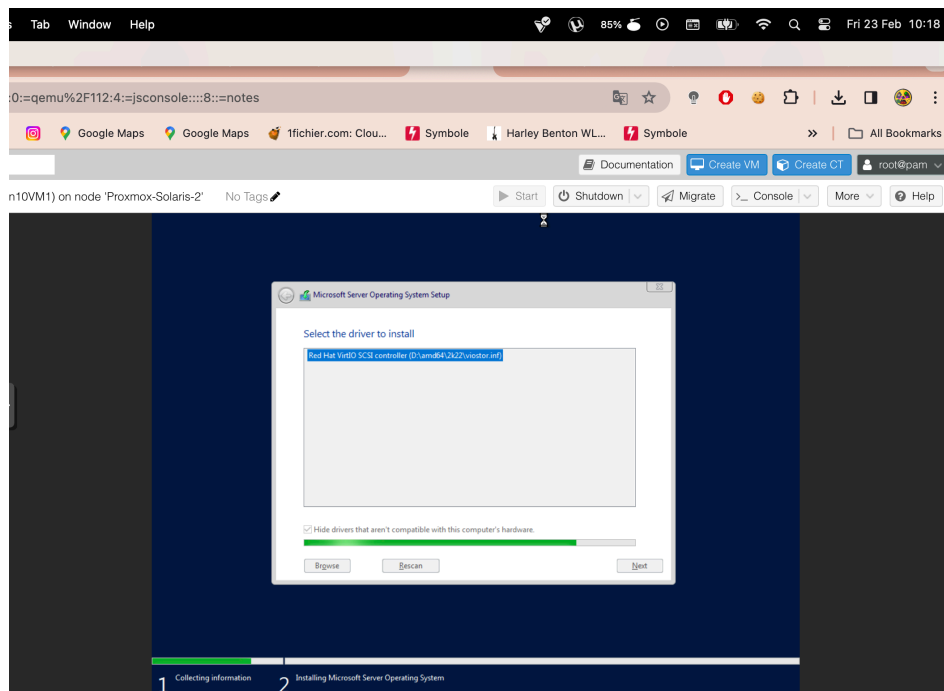
After some research, VirtIO drivers were missing and are not installed by default on the windows installation image. So, one had to find an ISO to add to the virtual machine, containing all the VirtIO drivers needed (found on proxmox's official documentation : [https://pve.proxmox.com/wiki/Windows\\_VirtIO\\_Drivers](https://pve.proxmox.com/wiki/Windows_VirtIO_Drivers)).



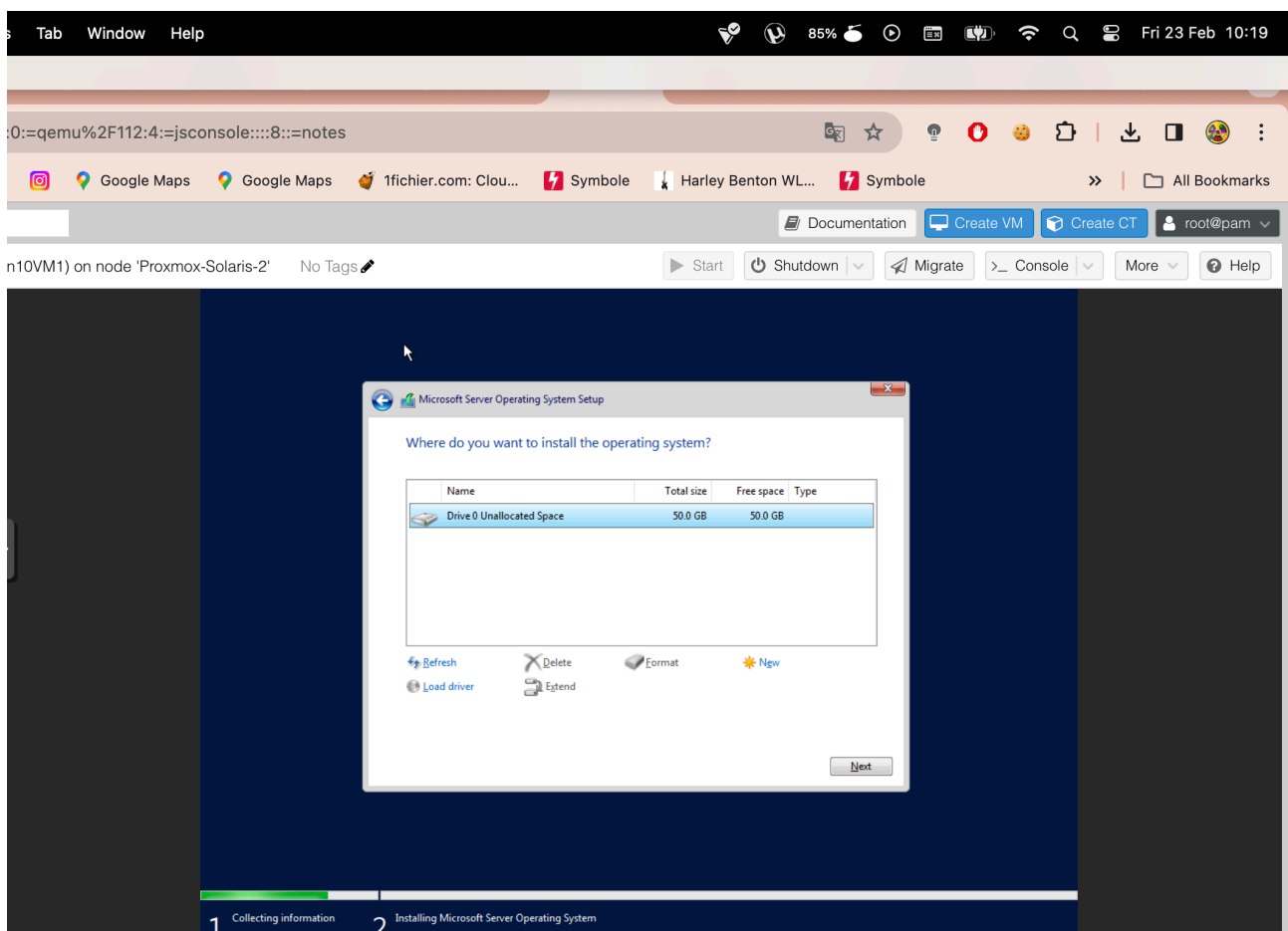
Adding VirtIO ISO drivers to the virtual machine



Finding the ISO on the Windows Setup



Driver installation process



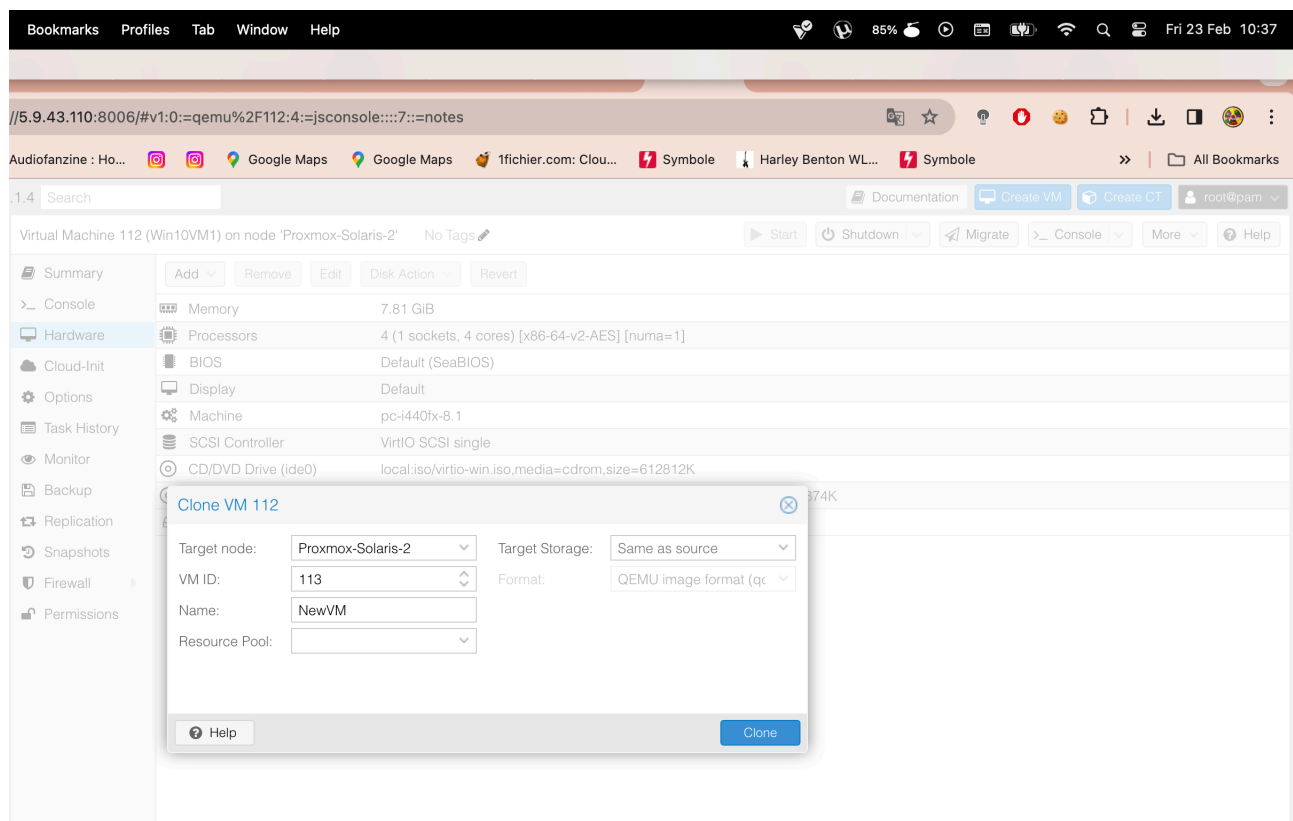
Virtual Disk found on VM, after VirtIO Driver installation

After the driver was installed, the virtual disk showed immediately on the Windows Setup. After this troubleshooting, Windows was installed correctly on the VM. One repeating this operation to install both Client and Server.

## Time saving protocol : Cloning Windows Server and Client VMs to get 4 VMs fully operational

Installing two times a server and a client would take a long time. Instead, one just have to clone the two operational VMs installed previously to have the 4 virtual machines needed to build the required network topology.

In Proxmox, this process is simple.



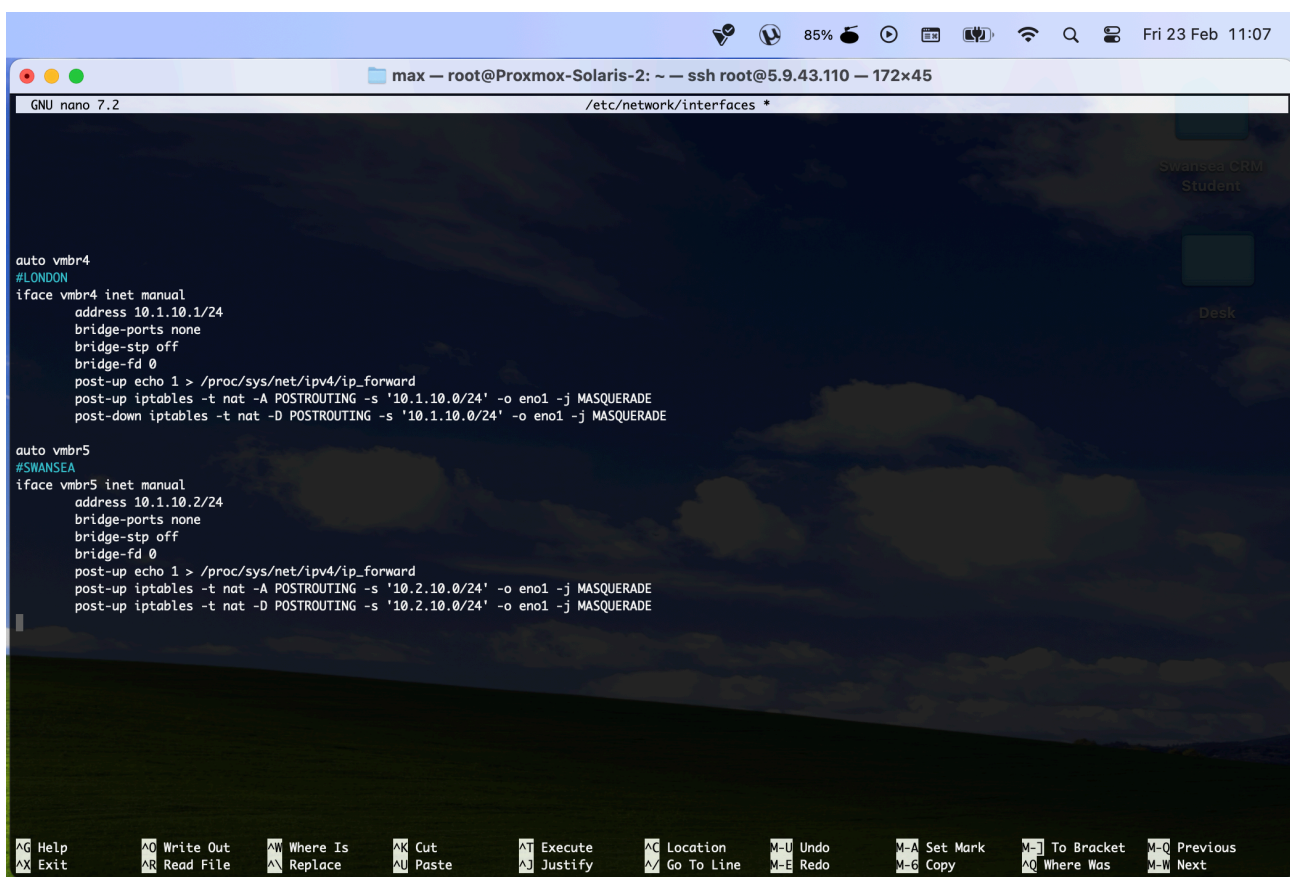
VM Clone process in one click - Proxmox

As the network is not setup as this state, we do not need to change any MAC address and IP address to avoid conflicts between VMs network interfaces.

## Network configuration on host, pre-configuration on VMs :

This part was particularly complex to realise with Proxmox type 1 hypervisor, as we cannot directly create internal networks like it is possible in Oracle VM Virtualbox. Proxmox works essentially with Network Bridges. A Linux network bridge is a virtual device that connects multiple network interfaces at the data link layer (Layer 2) of the OSI model. It allows different segments of a network to communicate as if they were a single network, effectively working like a software-based network switch. This is commonly used in virtualization setups to allow virtual machines to connect to a physical network. It basically works as a network switch.

### First step, creating two network bridges for London location, and Swansea Location :



```
GNU nano 7.2 /etc/network/interfaces *

auto vmbr4
#LONDON
iface vmbr4 inet manual
    address 10.1.10.1/24
    bridge-ports none
    bridge-stp off
    bridge-fd 0
    post-up echo 1 > /proc/sys/net/ipv4/ip_forward
    post-up iptables -t nat -A POSTROUTING -s '10.1.10.0/24' -o eno1 -j MASQUERADE
    post-down iptables -t nat -D POSTROUTING -s '10.1.10.0/24' -o eno1 -j MASQUERADE

auto vmbr5
#SWANSEA
iface vmbr5 inet manual
    address 10.1.10.2/24
    bridge-ports none
    bridge-stp off
    bridge-fd 0
    post-up echo 1 > /proc/sys/net/ipv4/ip_forward
    post-up iptables -t nat -A POSTROUTING -s '10.2.10.0/24' -o eno1 -j MASQUERADE
    post-up iptables -t nat -D POSTROUTING -s '10.2.10.0/24' -o eno1 -j MASQUERADE
```

### Proxmox VM Bridges configurations

We edit the server network configuration file, by default placed at `/etc/network/interfaces`. Now, with this configuration :

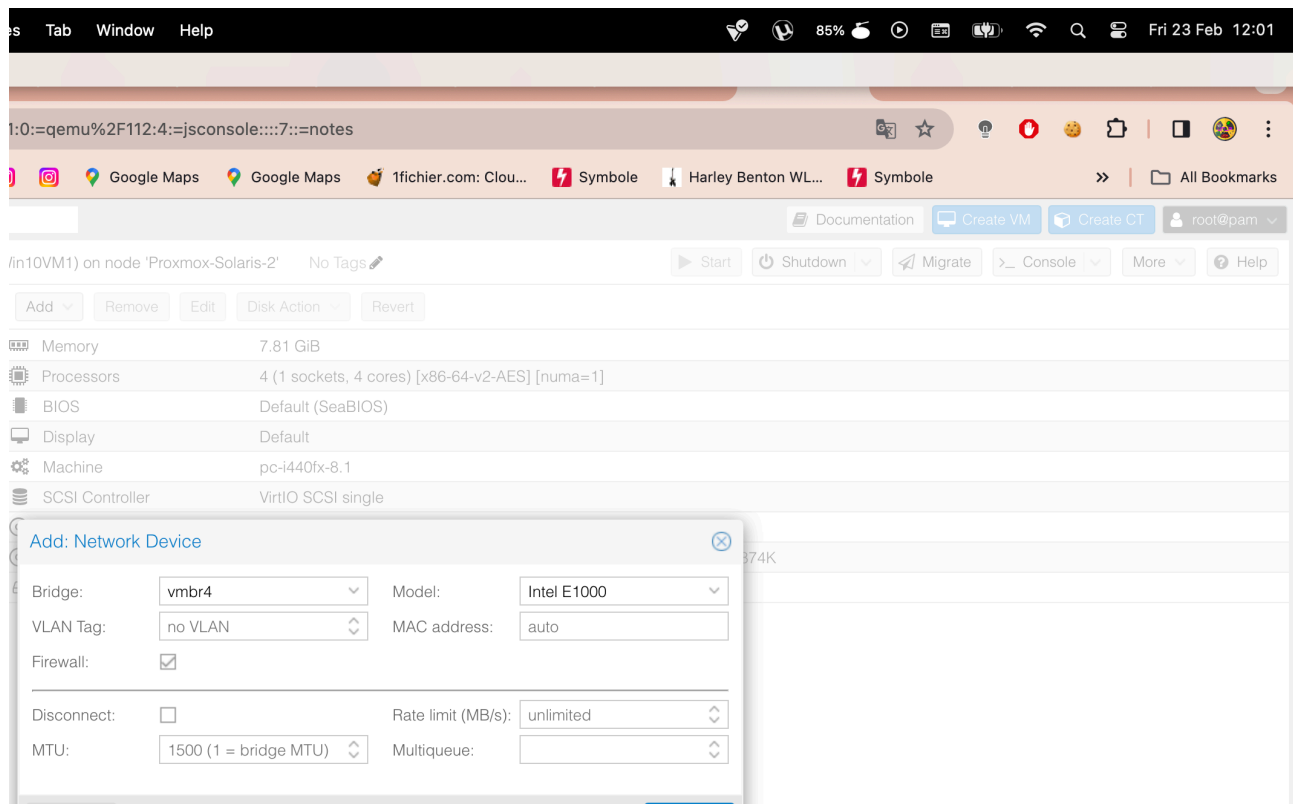
- Vmbr4 is the bridge for London. It is associated with a local subnet, 10.1.10.0/24.
- Vmbr5 is the bridge for Swansea. It is associated with a local subnet too, 10.2.10.0/24

NAT rules are here to give an internet access to the bridges. However, one chose to disable this feature later to completely isolate the network for this coursework.

The gateway for the VMs IP configurations will be the address of each respective bridge.

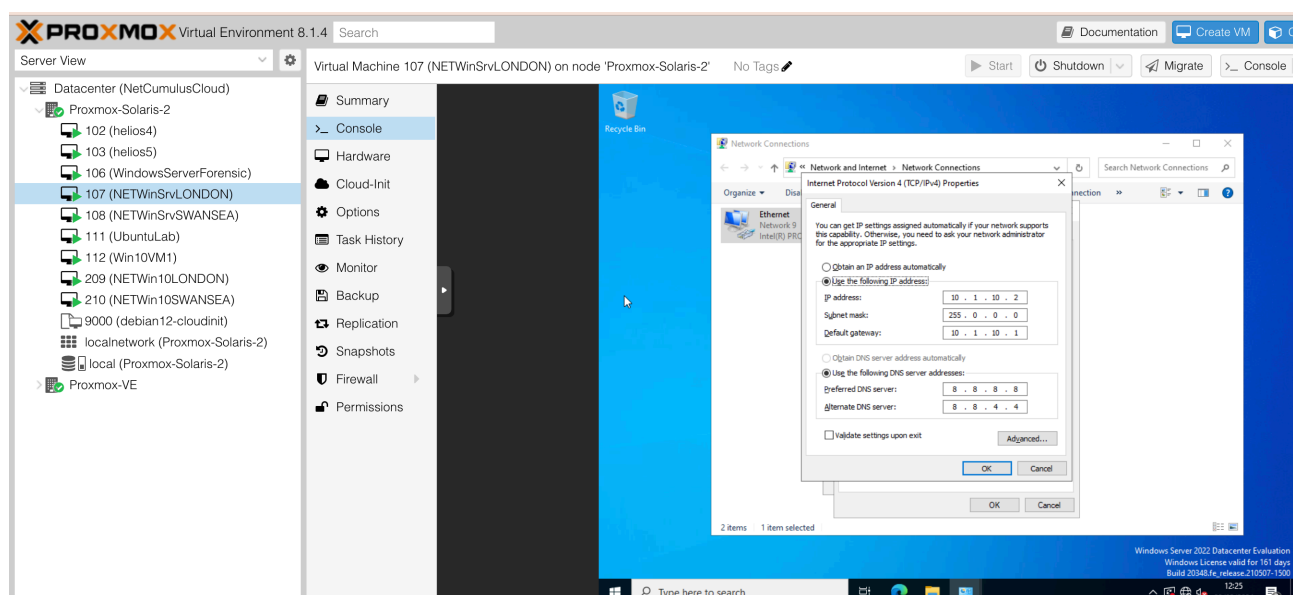
## Then, configuring network on virtual machines

For each virtual machine, we add a virtual network card, configured on London or Swansea bridge, depending on desired location for each virtual machine.



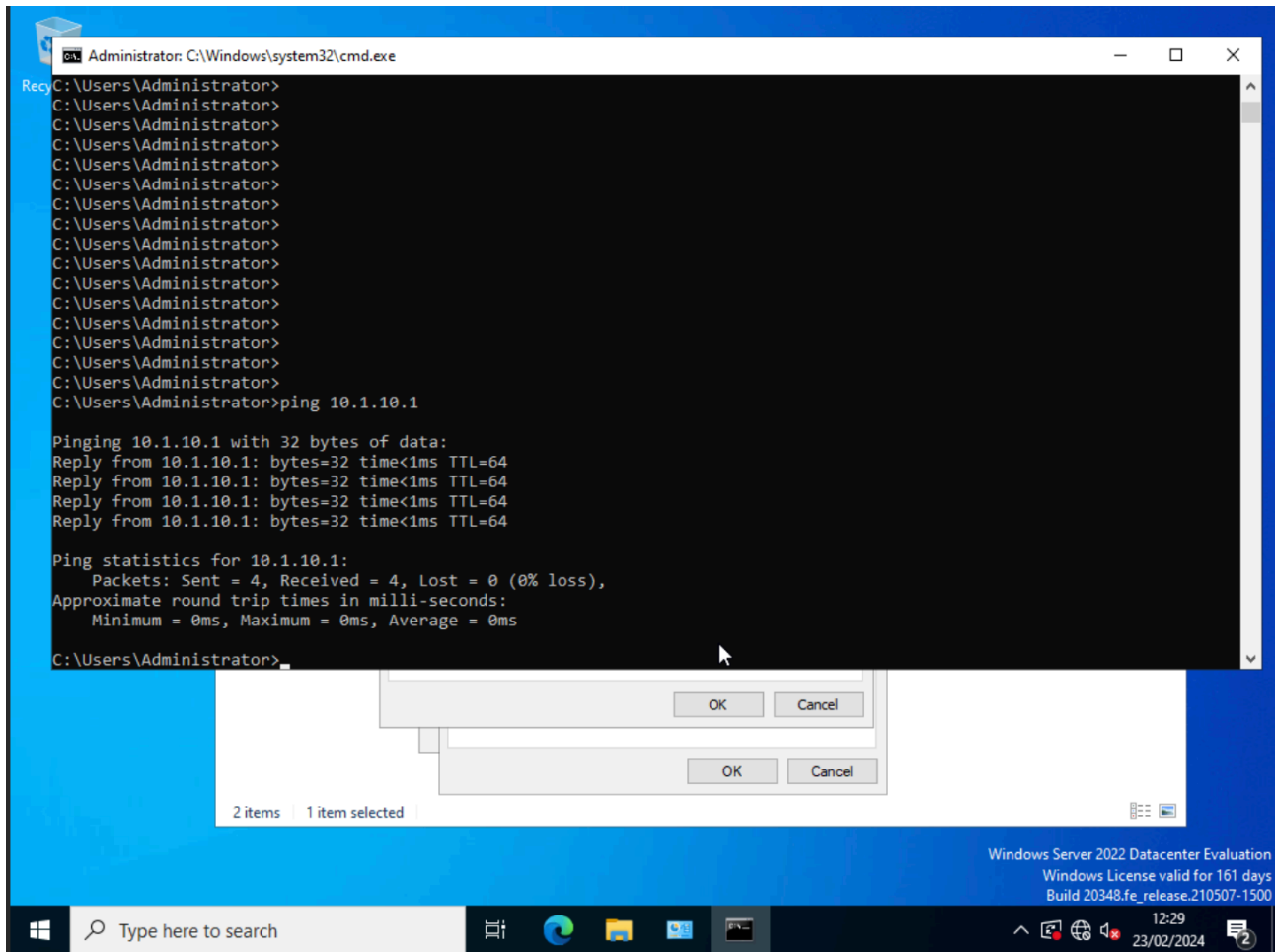
Adding a network device to London Windows Server

After having installed a network device on every VMs, we configure their IPs in the network connections panel in windows, like it is shown below



VM Server London, configuration example with address 10.1.10.2

Then we test the connectivity by pinging the gateway (for example, for the London network bridge, it will be 10.1.10.1). It is a success.



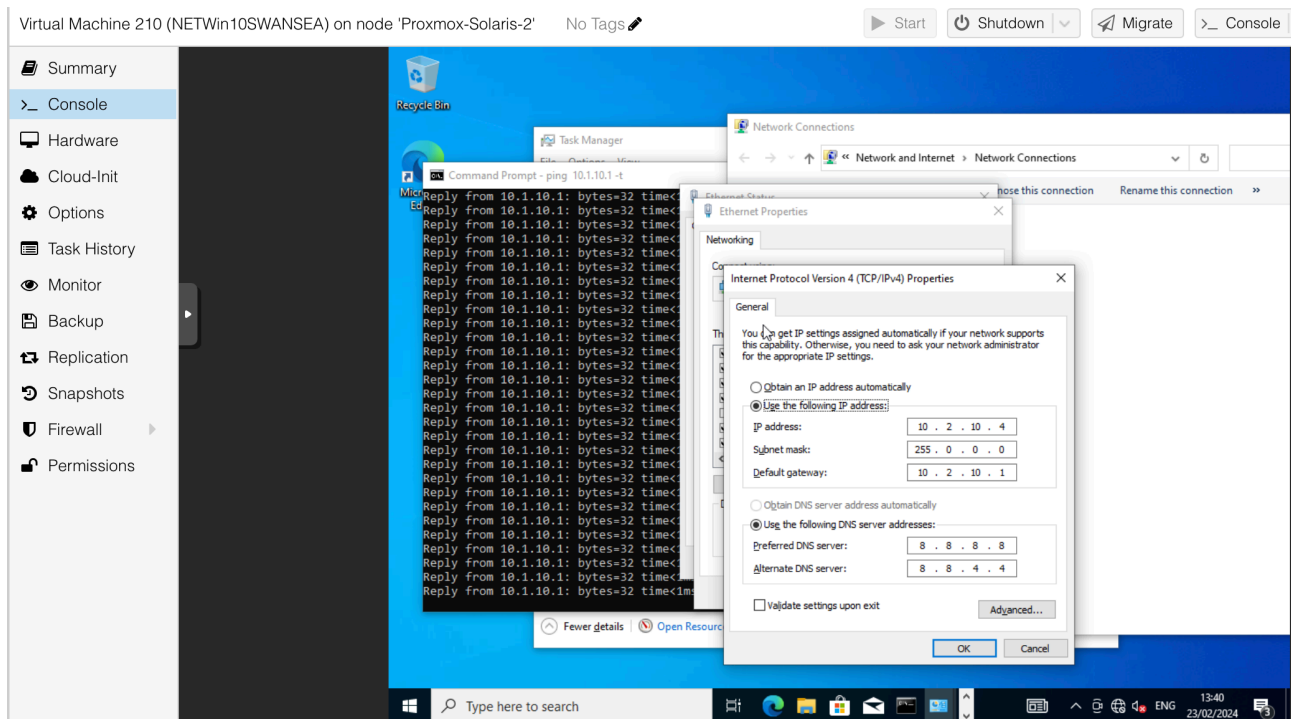
London server pinging London network bridge IP (gateway)

At this state, the network topology has been configured like this :

- 10.1.10.2 : Windows Server London
- 10.2.10.2 : Windows Server Client
- 10.1.10.4 : Windows 10 London Client
- 10.2.10.4 : Windows 10 Swansea Client

### Troubleshooting : network bridges not isolated at all :

At this state of the network configuration, a massive problem subsists : network bridges vmbr4 and vmbr5 are not isolated, and any VMs can ping any active assigned IP address on any bridge, without being linked to one of them.



Swansea Windows 10 Client being able to ping vmbr4 (London) IP without being connected to it

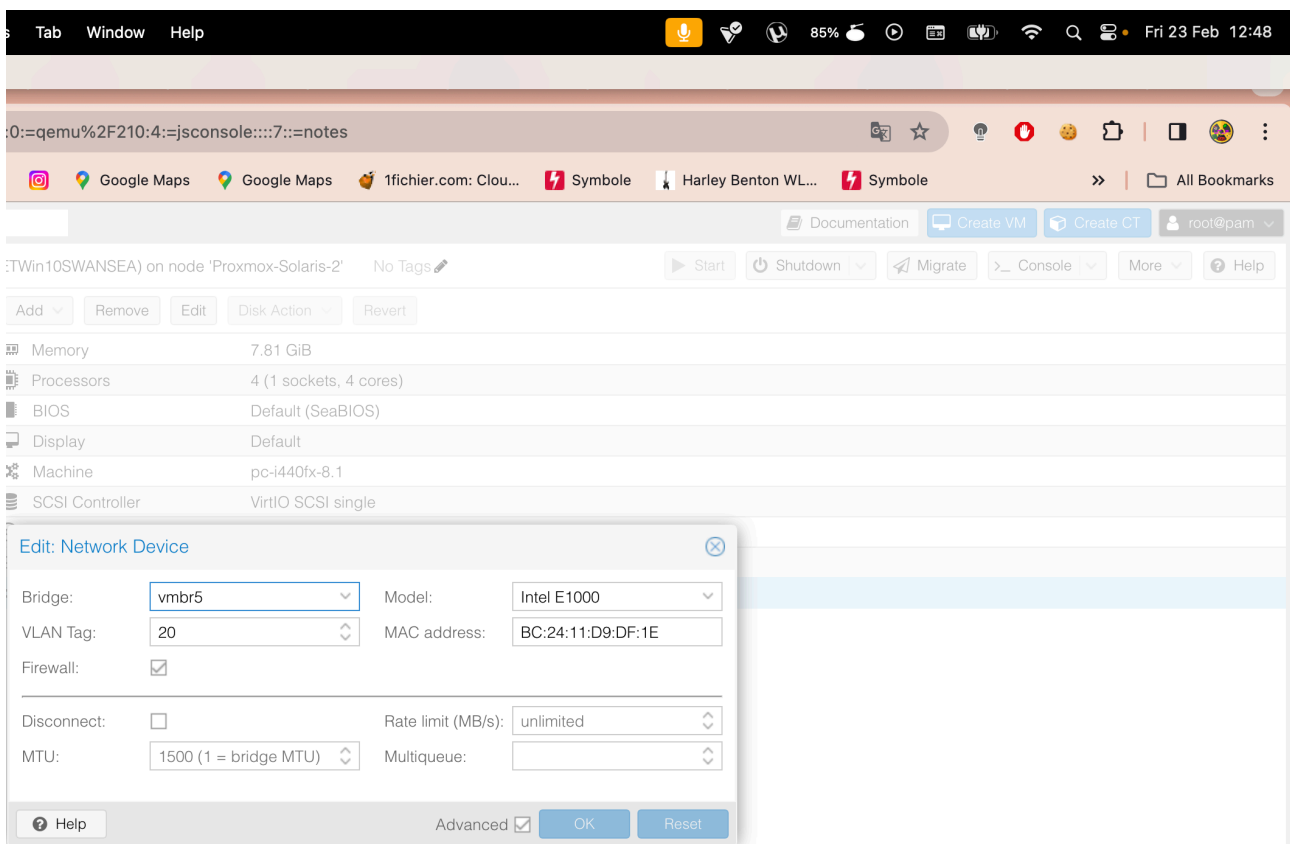
### First attempt to solve network isolation problem :

At this state of the network configuration, subnets were definitely able to communicate each other. However, with the given topology, that is not what we want. Both London and Swansea network must be isolated, and linked only on some parts (servers must be able to ping each other). To avoid this problematic and hopefully isolate networks between VMs in a first way, one tried to isolate subnets better, by creating VLANs associated to each bridge. A VLAN, or Virtual Local Area Network, is a network within a network, created to divide a larger network into smaller, isolated segments. It enhances network management and security by segregating groups of devices.

```
max — root@Proxmox-Solaris-2: ~ — ssh root@5.9.43.110 — 172x37
GNU nano 7.2 /etc/network/interfaces
auto vmbr4
#LONDON
iface vmbr4 inet manual
    bridge-ports none
    bridge-stp off
    bridge-fd 0
    post-up echo 1 > /proc/sys/net/ipv4/ip_forward
    post-up iptables -t nat -A POSTROUTING -s '10.1.10.0/24' -o eno1 -j MASQUERADE
    post-down iptables -t nat -D POSTROUTING -s '10.1.10.0/24' -o eno1 -j MASQUERADE
    bridge-vlan-aware yes
    bridge-vids 10
auto vmbr4.10
iface vmbr4.10 inet static
    vlan-raw-device vmbr4
    address 10.1.10.1/24

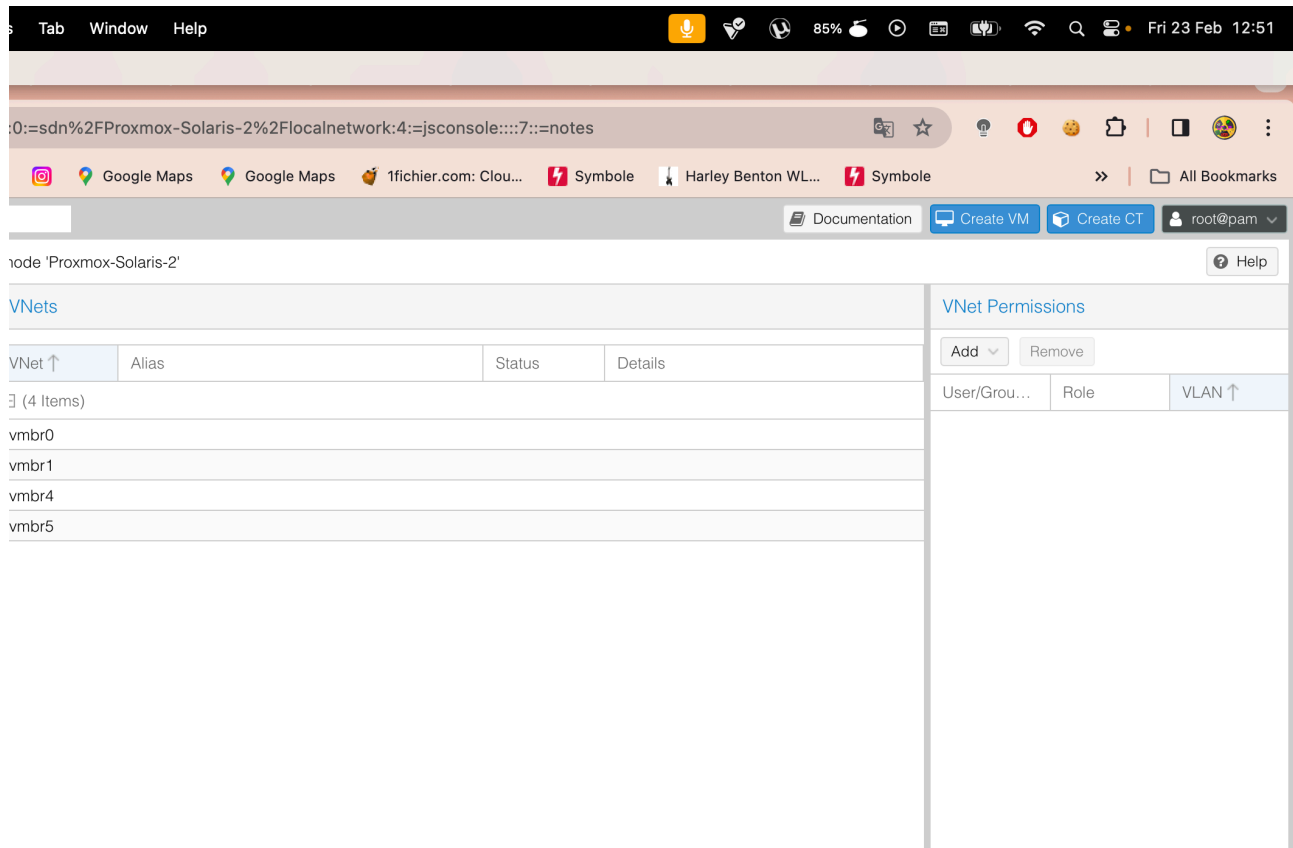
auto vmbr5
#SWANSEA
iface vmbr5 inet manual
    bridge-ports none
    bridge-stp off
    bridge-fd 0
    post-up echo 1 > /proc/sys/net/ipv4/ip_forward
    post-up iptables -t nat -A POSTROUTING -s '10.2.10.0/24' -o eno1 -j MASQUERADE
    post-up iptables -t nat -D POSTROUTING -s '10.2.10.0/24' -o eno1 -j MASQUERADE
    bridge-vlan-aware yes
    bridge-vids 20
auto vmbr5.20
iface vmbr5.20 inet static
    vlan-raw-device vmbr5
    address 10.2.10.1/24
```

Host network bridges configuration, adding VLANs (vmbr4.10 & vmbr5.20)

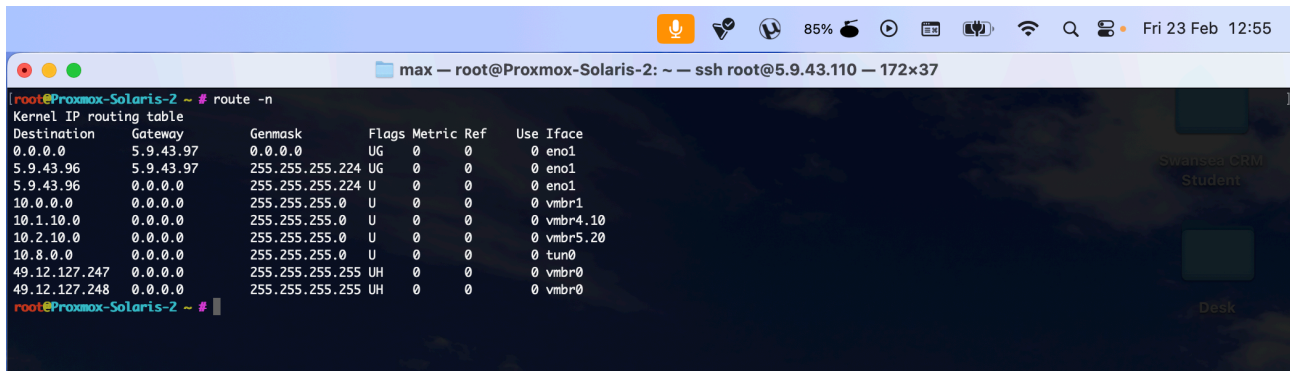


Reconfiguring VM network device by adding VLAN tag (vmbrXX.tag), refreshing MAC address

But after these modifications, nothing changed. The network was better organised (now we can arrange VMs by VLAN groups on the proxmox GUI, but the network still not isolated).



Bridges now separated into groups



Subnets now routed on VLANs vmbr4.10 & vmbr5.20

### Solution : use IPTABLES firewall rules to isolate VLANs

The only way to properly isolate networks was to use iptables rules. Iptables is the default firewall installed on proxmox and a great part of linux distributions. It is a powerful tool to route network properly

The rules topology has to be like this :

- Rule has to be defined on INPUT category (entering network packet)
- Rule has to filter and block any traffic coming from any other network interface/source than the local subnet —> DROP rule

```
iptables -D INPUT ! -i vmbr4.10 -d 10.1.10.0/24 -j DROP
```

```
iptables -D INPUT ! -i vmbr5.20 -d 10.2.10.0/24 -j DROP
```

—> If input packet source is different of interface vmbrX.X to the destination of a subnet, it is dropped / blocked.

```
root@Proxmox-Solaris-2 ~ # iptables -A INPUT ! -i vmbr4.10 -d 10.1.10.0/24 -j DROP
root@Proxmox-Solaris-2 ~ # iptables -A INPUT ! -i vmbr5.20 -d 10.2.10.0/24 -j DROP
root@Proxmox-Solaris-2 ~ # iptables -L -v
Chain INPUT (policy ACCEPT 542 packets, 80640 bytes)
 pkts bytes target    prot opt in     out     source               destination
  33  2628 DROP     all  --  !vmbr4.10 any    anywhere             10.1.10.0/24
    0     0 DROP     all  --  !vmbr5.20 any    anywhere             10.2.10.0/24

Chain FORWARD (policy ACCEPT 5679 packets, 755K bytes)
 pkts bytes target    prot opt in     out     source               destination

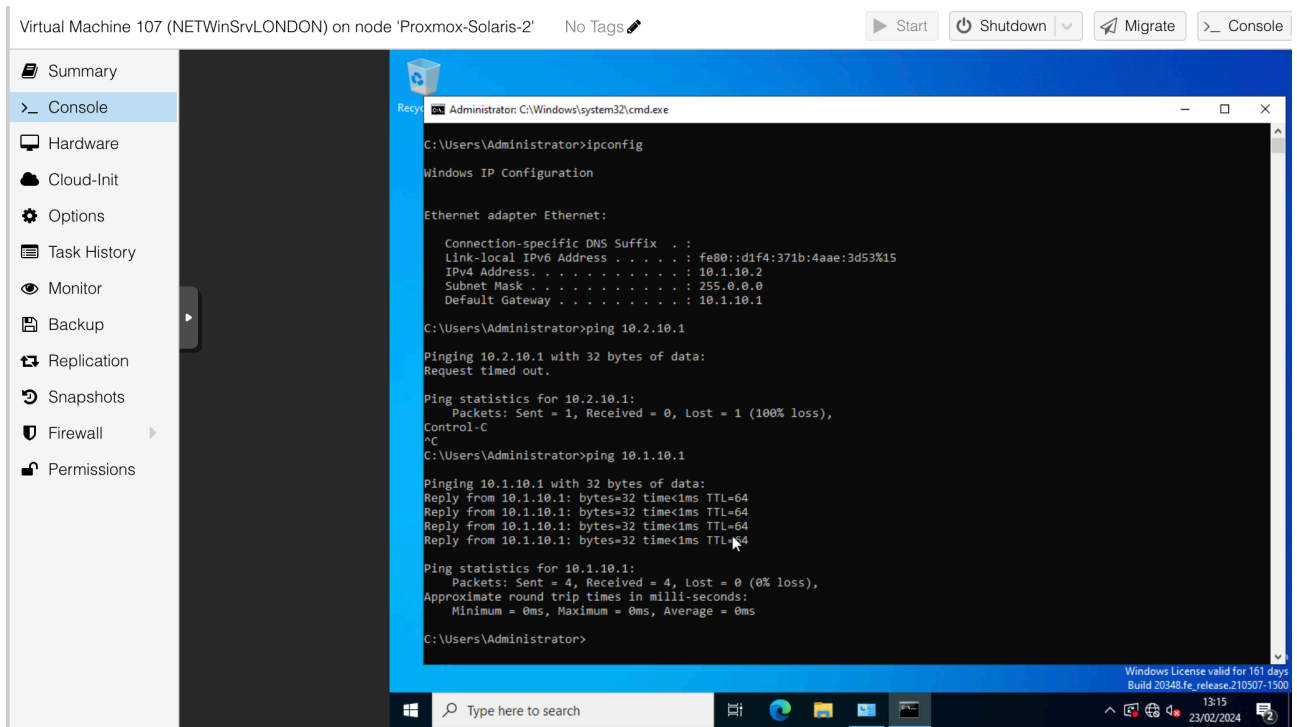
Chain OUTPUT (policy ACCEPT 489 packets, 84937 bytes)
 pkts bytes target    prot opt in     out     source               destination
root@Proxmox-Solaris-2 ~ # date
Fri Feb 23 02:11:30 PM CET 2024
root@Proxmox-Solaris-2 ~ # ping 10.1.10.1
PING 10.1.10.1 (10.1.10.1) 56(84) bytes of data.
^C
--- 10.1.10.1 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11250ms

root@Proxmox-Solaris-2 ~ # ping 10.2.10.1
PING 10.2.10.1 (10.2.10.1) 56(84) bytes of data.
^C
--- 10.2.10.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1023ms

root@Proxmox-Solaris-2 ~ # date
Fri Feb 23 02:12:13 PM CET 2024
```

Bridges not responding to ping on host after applying Iptables rules. They are now isolated.

And now, if we try to ping Swansea gateway on London server, it doesn't ping. But if we try to ping the London gateway, it pings.



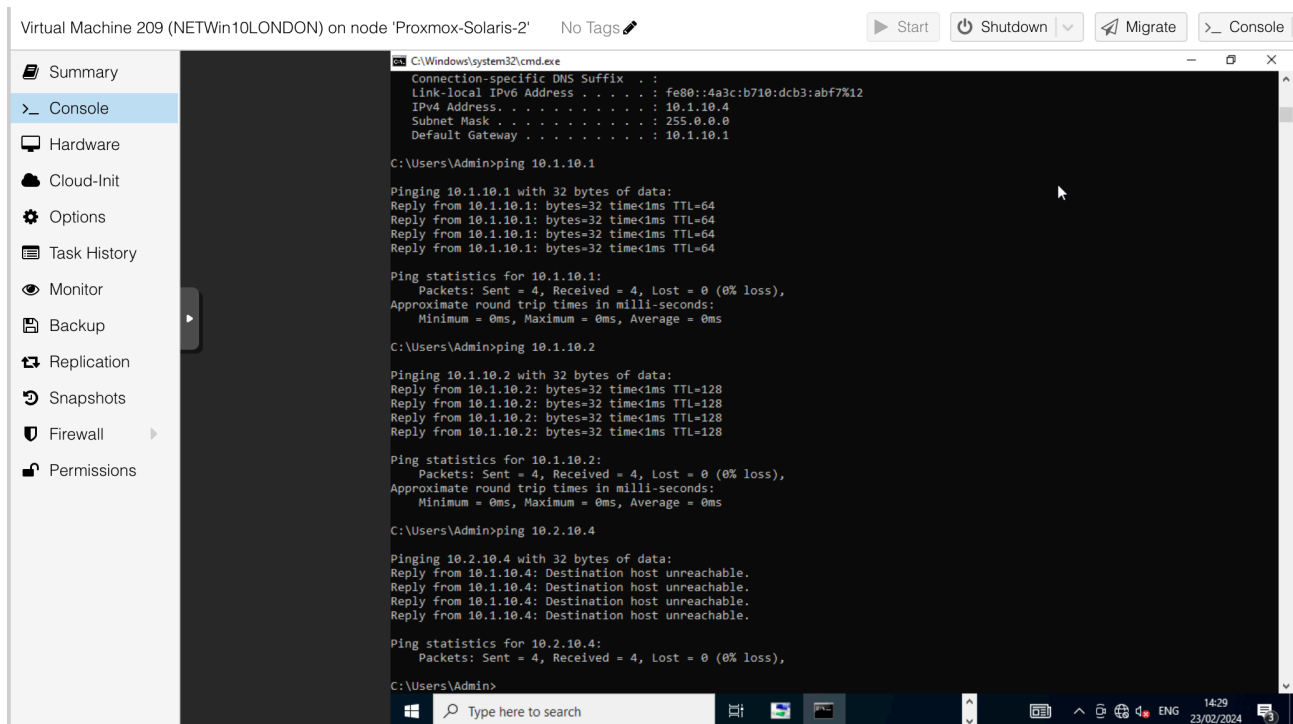
London server pinging London gateway, but not Swansea gateway. Networks are now fully isolated.

At this state, all VMs have an ip address assigned to their bridges. Now, the only remaining step is to setup the NOSS topology given in the coursework instructions.

## Setup Network NOSS Topology

### On clients :

The topology is already complete on client VMs, as they just have to be connected to their bridge depending on their locations (London or Swansea) and ping their gateways, without being able to communicate each other.



The screenshot shows a Windows 10 command prompt window within a Proxmox VM. The title bar reads 'C:\Windows\system32\cmd.exe'. The command prompt shows the following output:

```
Connection-specific DNS Suffix . :  
Link-local IPv6 Address . . . . . : fe80::4a3c:b710:dcb3:abf7%12  
IPv4 Address. . . . . : 10.1.10.4  
Subnet Mask . . . . . : 255.0.0.0  
Default Gateway . . . . . : 10.1.10.1  
  
C:\Users\Admin>ping 10.1.10.1  
  
Pinging 10.1.10.1 with 32 bytes of data:  
Reply from 10.1.10.1: bytes=32 time<1ms TTL=64  
Reply from 10.1.10.1: bytes=32 time<1ms TTL=64  
Reply from 10.1.10.1: bytes=32 time<1ms TTL=64  
Reply from 10.1.10.1: bytes=32 time<1ms TTL=64  
  
Ping statistics for 10.1.10.1:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 0ms, Maximum = 0ms, Average = 0ms  
  
C:\Users\Admin>ping 10.1.10.2  
  
Pinging 10.1.10.2 with 32 bytes of data:  
Reply from 10.1.10.2: bytes=32 time<1ms TTL=128  
Reply from 10.1.10.2: bytes=32 time<1ms TTL=128  
Reply from 10.1.10.2: bytes=32 time<1ms TTL=128  
Reply from 10.1.10.2: bytes=32 time<1ms TTL=128  
  
Ping statistics for 10.1.10.2:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 0ms, Maximum = 0ms, Average = 0ms  
  
C:\Users\Admin>ping 10.2.10.4  
  
Pinging 10.2.10.4 with 32 bytes of data:  
Reply from 10.1.10.4: Destination host unreachable.  
Reply from 10.1.10.4: Destination host unreachable.  
Reply from 10.1.10.4: Destination host unreachable.  
Reply from 10.1.10.4: Destination host unreachable.  
  
Ping statistics for 10.2.10.4:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
  
C:\Users\Admin>
```

London Windows 10 Client can ping London gateway and server (vmbr4 ipv4) without being able to ping Swansea Windows 10 Client.

### On servers :

Both servers, following the topology given in the NOSS instructions, should be able to communicate each other. To do so, it is necessary to add a network interface on each Server VM to make them connected to both London and Swansea network bridges :

- We add 10.2.10.3 on London Server (Swansea bridge IP address)
- And we add 10.1.10.3 on Swansea Server (London Bridge IP address)

We get these results :

```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.20348.587]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::ec13:53a3:36bd:292d%15
    IPv4 Address. . . . . : 10.2.10.2
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : 10.2.10.1

Ethernet adapter Ethernet 5:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::dd65:4694:cb8b:b996%4
    IPv4 Address. . . . . : 10.1.10.3
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : 10.1.10.1

C:\Users\Administrator>_
```

Added network device with London network IP address on Swansea server

Same operation has been done on London server.

**Troubleshooting : Despite adding a new network interfaces, servers cannot ping each other and both gateways.**

By adding some route rules to windows server and make them persistent by using « -p » option, this problem was solved.

Example of command used :

- *Route add 10.2.10.0 mask 255.255.255.0 10.2.10.3* (this rules makes all the traffic going to Swansea subnet 10.2.10.0/24 going through the interface 10.2.10.3 on London Server).

Now, the NOSS topology is complete, and servers can ping each other, and join both bridge IPs (gateways).

```
Administrator: C:\Windows\system32\cmd.exe
Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::d1f4:371b:4aae:3d53%15
    IPv4 Address. . . . . : 10.1.10.2
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : 10.1.10.1

Ethernet adapter Ethernet 5:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::4c52:40d2:8584:5c20%12
    IPv4 Address. . . . . : 10.2.10.3
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : 10.2.10.1

C:\Users\Administrator>ping 10.2.10.2

Pinging 10.2.10.2 with 32 bytes of data:
Reply from 10.2.10.2: bytes=32 time<1ms TTL=128
Reply from 10.2.10.2: bytes=32 time<1ms TTL=128
Reply from 10.2.10.2: bytes=32 time<1ms TTL=128
Reply from 10.2.10.2: bytes=32 time<1ms TTL=128

Ping statistics for 10.2.10.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Administrator>ping 10.2.10.1

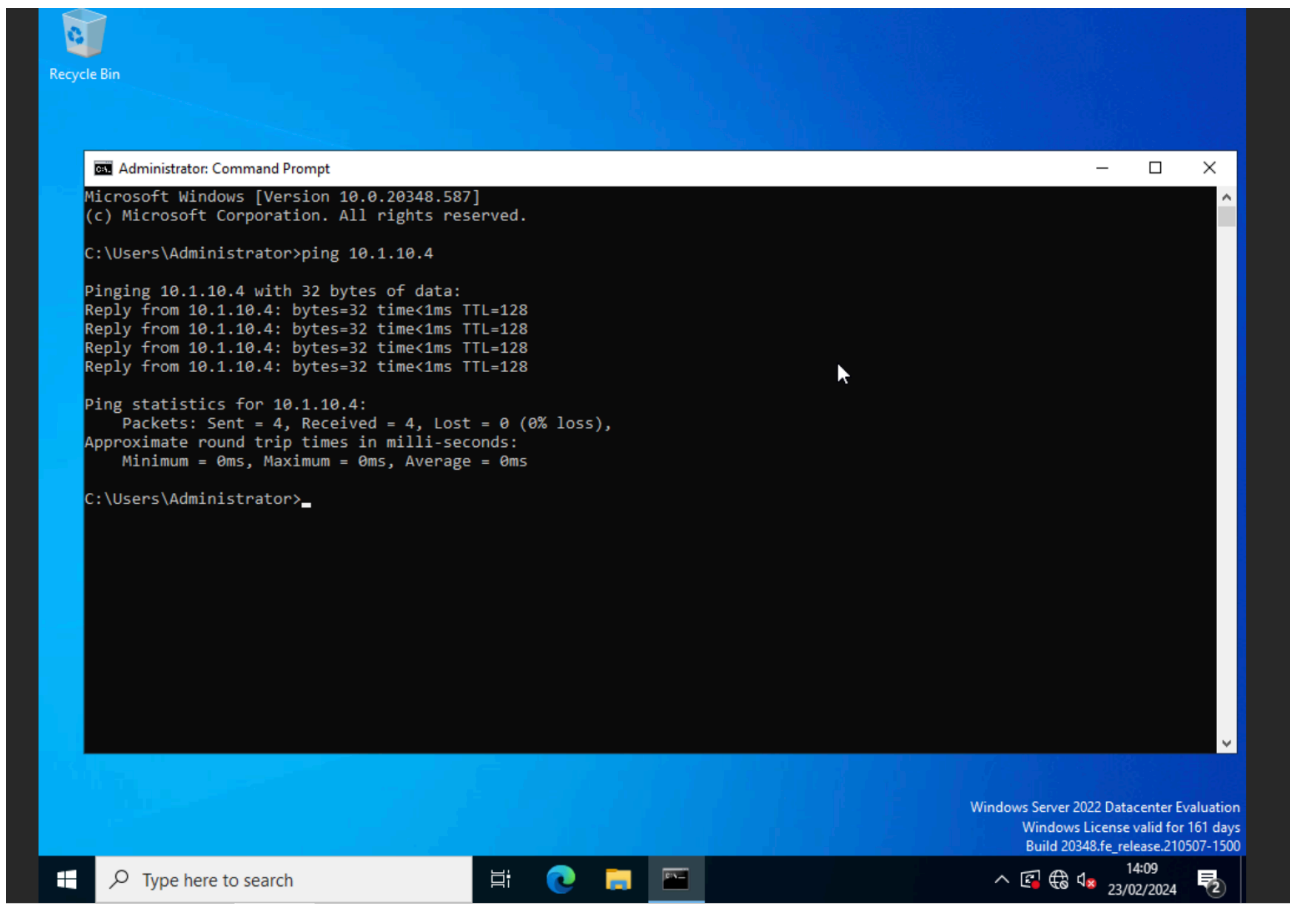
Pinging 10.2.10.1 with 32 bytes of data:
Reply from 10.2.10.1: bytes=32 time<1ms TTL=64
Reply from 10.2.10.1: bytes=32 time<1ms TTL=64
Reply from 10.2.10.1: bytes=32 time<1ms TTL=64
Reply from 10.2.10.1: bytes=32 time<1ms TTL=64

Ping statistics for 10.2.10.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Administrator>
```

London server pinging successfully Swansea server and Swansea gateway

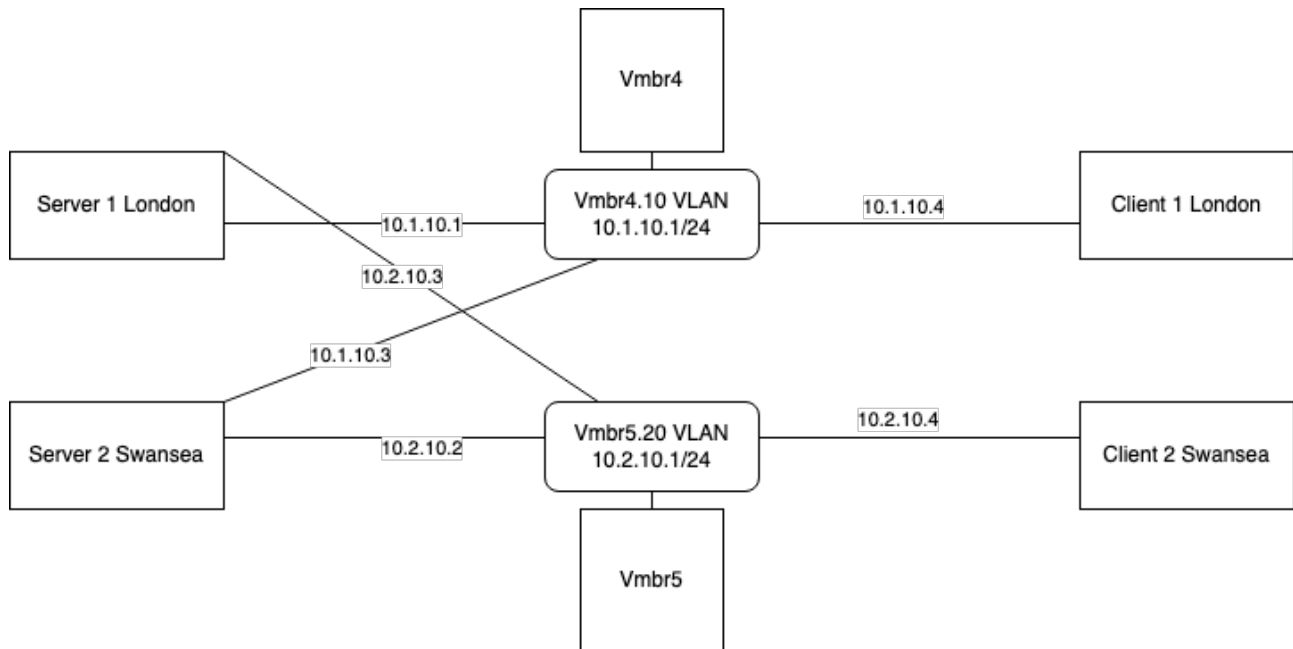
Servers are also able to ping their respective linked clients.



London server ping 10.1.10.4 London Client Win10 PC

## NOSS Topology Final Proxmox Network topology scheme

Servers are now connected together and can ping each other. Clients are connected to their respective locations /servers.



## **Conclusion**

In this coursework, the implementation of a virtualized network infrastructure using Proxmox, a Type 1 hypervisor, was successfully achieved despite some initial challenges. The selection of Proxmox was driven by compatibility issues with ARM64 architecture and its availability on pre-configured servers. The process involved downloading system images directly to the server, troubleshooting ISO visibility issues, and creating virtual machines with specific configurations.

Significant steps included resolving VirtIO driver issues for Windows installations and employing a cloning strategy to efficiently replicate VMs. The network setup, a crucial component, was intricately designed using Linux network bridges to simulate real-world network environments in London and Swansea locations, with VLANs initially intended for better network organization.

However, the desired network isolation was only attained through the application of iptables firewall rules, effectively segregating network traffic between the two locations. This setup ensured that while servers in different locations could communicate, client VMs remained isolated within their respective networks.

Ultimately, the configured NOSS topology met the coursework requirements, with servers capable of intercommunication and clients correctly linked to their designated servers. This project not only demonstrated the robust capabilities of Proxmox as a Type 1 hypervisor but also highlighted the importance of network configuration and troubleshooting skills in virtualized environments.

### **Sources used for this coursework :**

- <https://aws.amazon.com/compare/the-difference-between-type-1-and-type-2-hypervisors/#:~:text=The%20type%20hypervisor%20sits,on%20the%20host%20operating%20system.>
- <https://phoenixnap.com/kb/what-is-hypervisor-type-1-2#type-1-vendors>
- <https://serverfault.com/questions/855094/is-kvm-a-type-1-or-type-2-hypervisor#:~:text=The%20KVM%20kernel%20module%20turns,less%20to%20do%20with%20th>  
is.
- <https://www.qemu.org/>
- <https://forum.proxmox.com/threads/antiddos-firewall.31157/>
- <https://stackoverflow.com/questions/9739156/how-to-flush-route-table-in-windows>
- GPT-3.5 / 4 to understand some basic concepts