SERVER VIRTUALISATION

Specialization Topic: Hypervisor CLI & Automation (NCP-NG & Proxmox)

ARISTIDE KATAGARUKA & LAURENCE WEBER

BTS CLOUD COMPUTING

LYCÉE GUILLAUME KROLL

SUBMISSION DATE: JUNE 12TH 2025

Table of Contents

Certification and Declaration of Honor

I, the undersigned, hereby declare that the work contained in this project report titled **"Server Virtualization"** is my own and has been completed as part of the VIRCL course requirements for the BTS Cloud Computing program at Lycée Guillaume Kroll.

I affirm that:

1. I have conducted the project ethically and honestly, adhering to the principles of integrity and academic conduct.
2. All external references, sources, and contributions by others have been appropriately acknowledged in the report.
3. The report has not been plagiarized, and no part of this work has been submitted previously for any other assignment, certification, or purpose.

Furthermore, I acknowledge that failure to comply with the rules of academic honesty may result in disqualification of this project and the revocation of grades awarded for the VIRCL course.

Signatures of Team Members:

- Aristide KATAGARUKA 
- Laurence WEBER 

Date: 12/06/2025

1. Introduction

Server virtualization is a cornerstone of modern IT infrastructure, enabling efficient resource utilization, cost savings, and operational flexibility. This project, part of the BTS Cloud Computing (VIRCL) module, involved deploying and comparing two type-1 hypervisors: XCP-NG and Proxmox VE. Our team, Aristide and Laurence, implemented a range of virtualization features, including VM creation, configuration, backup, migration, and user management on servers bcc07 and bcc11. We specialized in Hypervisor CLI & Automation, developing advanced scripts to optimize hypervisor operations, with a focus on dynamic VM provisioning and load balancing. This report details our implementation, compares the hypervisors, discusses challenges, and provides a conclusion.

2. Project Overview

We deployed XCP-NG and Proxmox VE on two servers: bcc07 (IP: 10.0.10.71(Proxmox) IP:10.0.10.72 (XCP-NG), assigned to Aristide, 1x SATA 160GB + 1x SATA 500GB, 4x8GB RAM) and bcc11 (IP: 10.0.10.111, assigned to Laurence, same specs). Both hypervisors were installed on each server to test all features independently. We used a shared CLOIF2 NAS from the previous project for remote storage, configured with NFS and SMB protocols. The following sections detail each feature's implementation on both hypervisors across both servers, supported by practical examples and screenshots, culminating in our advanced CLI automation work.

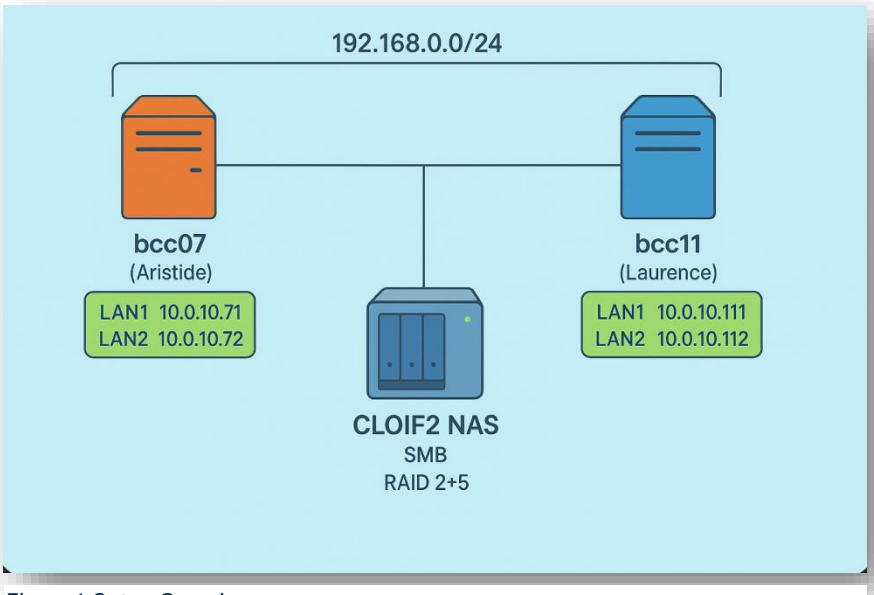

Figure 1:Setup Overview

3. What is a hypervisor?

A hypervisor, also known as a virtual machine monitor (VMM), is software or firmware that allows multiple virtual machines (VMs) to run on a single physical machine by managing and allocating the system's hardware resources. It creates an abstraction layer between the physical hardware and the virtual environments, enabling each VM to operate independently with its own operating system and applications. Hypervisors are essential in virtualization because they increase efficiency, reduce hardware costs, and improve scalability.

There are two main types:

- Type 1 hypervisors, which run directly on the hardware.
- Type 2 hypervisors, which run on top of a host operating system.

4. Hypervisor: Proxmox

Proxmox is an open-source virtualization platform that allows users to manage virtual machines (VMs) and containers on a single interface. It combines two virtualization technologies: KVM (Kernel-based Virtual Machine) for VMs and LXC (Linux Containers) for lightweight container virtualization. Proxmox includes powerful features like a web-based management interface, live migration, backups, snapshots, and clustering, making it a popular choice for both home labs and enterprise environments.

5. Hypervisor: XCP-ng

XCP-ng is an open-source virtualization platform based on XenServer, designed for managing and running virtual machines. It uses the Xen hypervisor and provides enterprise-grade features such as live migration, snapshots, high availability, and centralized management through tools like Xen Orchestra. XCP-ng is known for its stability, strong community support, and flexibility, making it a solid choice for businesses and IT professionals looking for a free and powerful virtualization solution.

6. Virtualization Features Implementation

6.1 Deployment of Virtualization Environments

XCP-NG (bcc07 and bcc11): On both servers, we installed XCP-NG using a SanDisk USB stick. The ISO was downloaded from https://xcp-ng.org and flashed with Rufus. On bcc07 (IP: 10.0.10.72, hostname: bcc07), we configured the 160GB SATA drive and network settings during the guided installation. The same steps were followed on bcc11 (IP: 10.0.10.111, hostname: bcc11). Each installation took ~40 minutes, with manual network configuration requiring verification of IP settings.

6.1.1 XCP installation

During the setup process, it was necessary to select a primary disk with at least 70 GB of available space, as XCP-ng requires a minimum of 70 GB for installation and proper operation.

In this case, the Toshiba drive was chosen as the main storage location for the virtual machines and ensuring stable system performance.

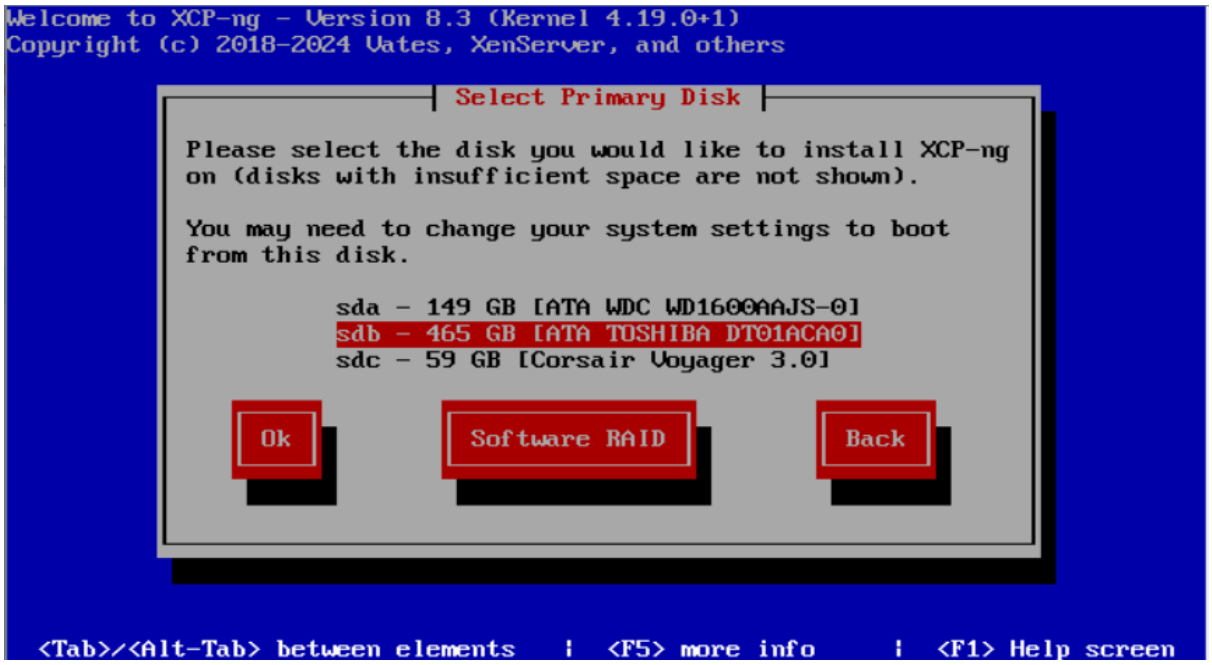

Figure 2: Selecting a Primary Disk

As part of the setup, it was also necessary to select a storage location for the virtual machines. This storage would hold all VM data, including virtual disks and snapshots. A suitable storage option with enough free space and good performance was chosen to ensure smooth VM operation. In this case, the selected storage met the system requirements and was properly integrated with the hypervisor for efficient VM management.

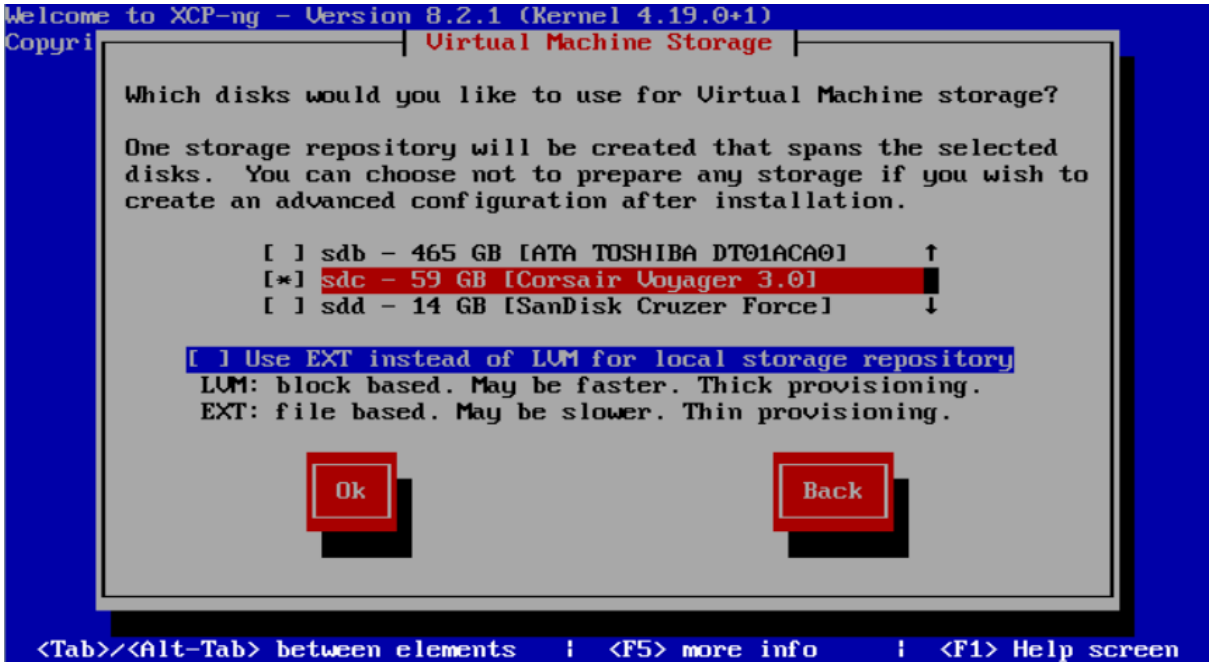

Figure 3: Selecting Virtual machine Storage

6.1.2. Network Configuration

- Configuration of the Network, Hostname and DNS server.
  - **IP Adress**: 10.0.10.72
  - **Subnet mask:** 255.255.0.0
  - **Gateway:** 10.0.0.1
  - **DNS:** 10.0.0.1

Repeating the same configuration of the Network, hostname and DNS Server for the server Bcc11.

- Configuration of the Network, Hostname and DNS server.
  - **IP Adress**: 10.0.10.112
  - **Subnet mask:** 255.255.0.0
  - **Gateway:** 10.0.0.1
  - **DNS:** 10.0.0.1

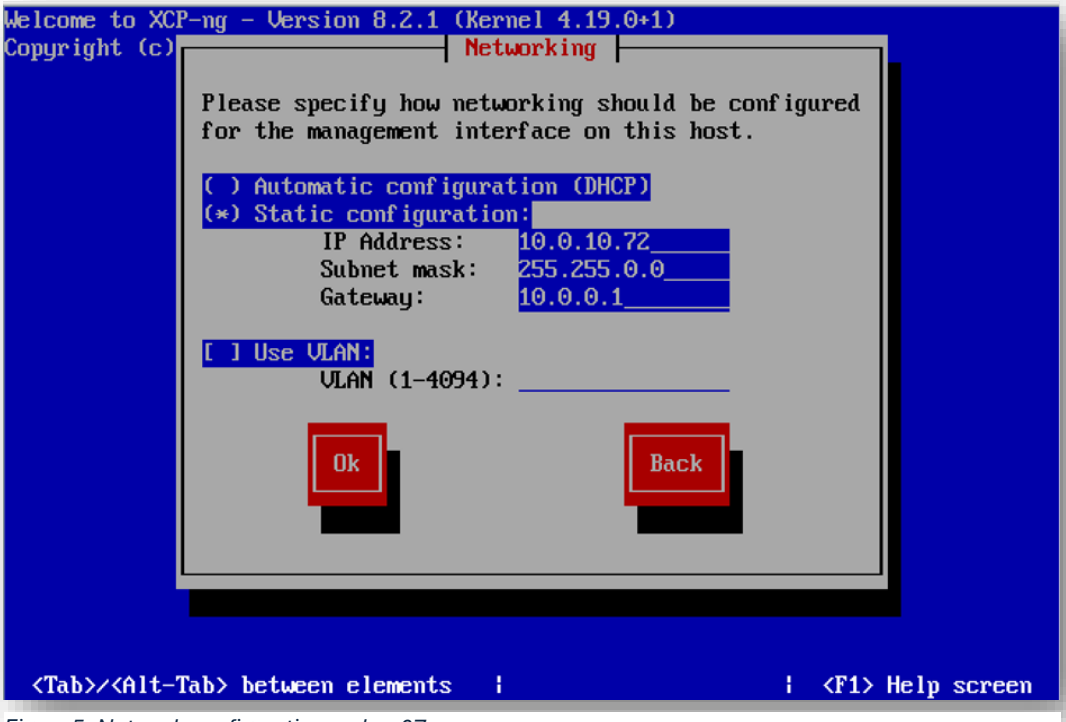

Figure 5: Network configuration on bcc07

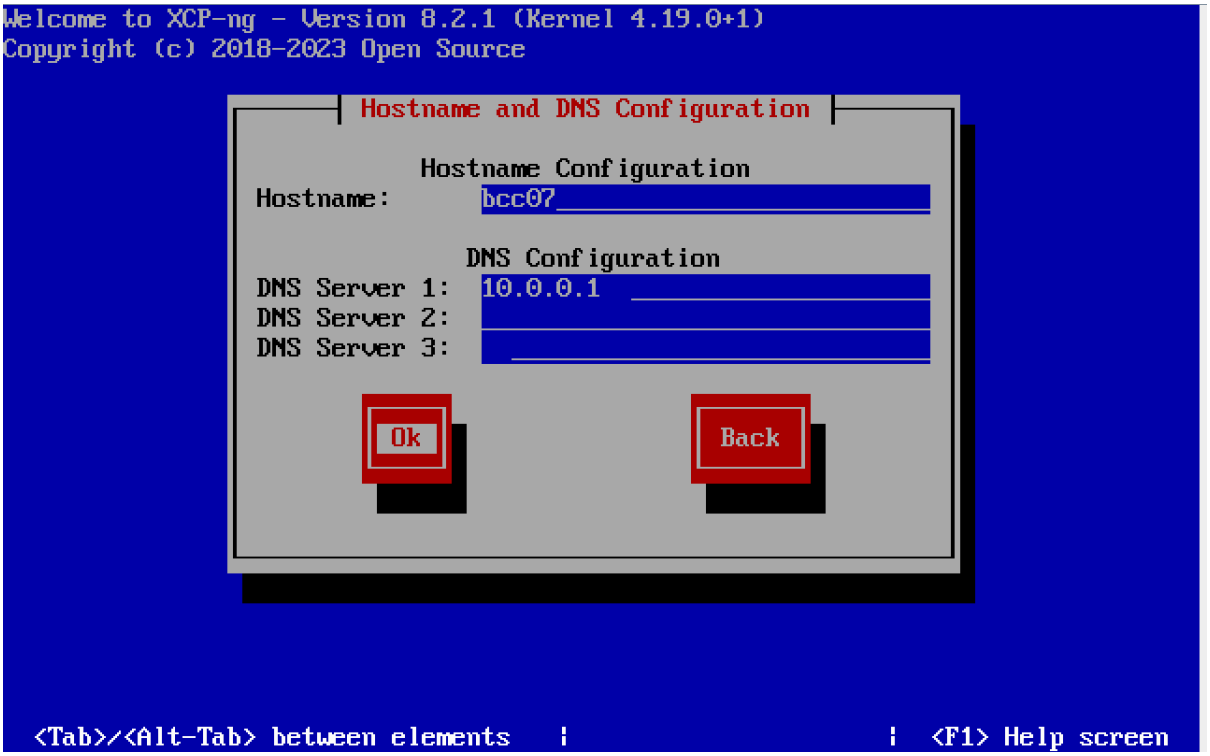

Figure 4: Configuration of Hostname and DNS server

To Deploy Xen Orchestra:

- **XOA\VM IP address**: 10.0.10.111/10.0.10.71
- **Netmask**: 255.255.0.0
- **Gateway**: 10.0.0.1
- **DNS:** 10.0.0.1

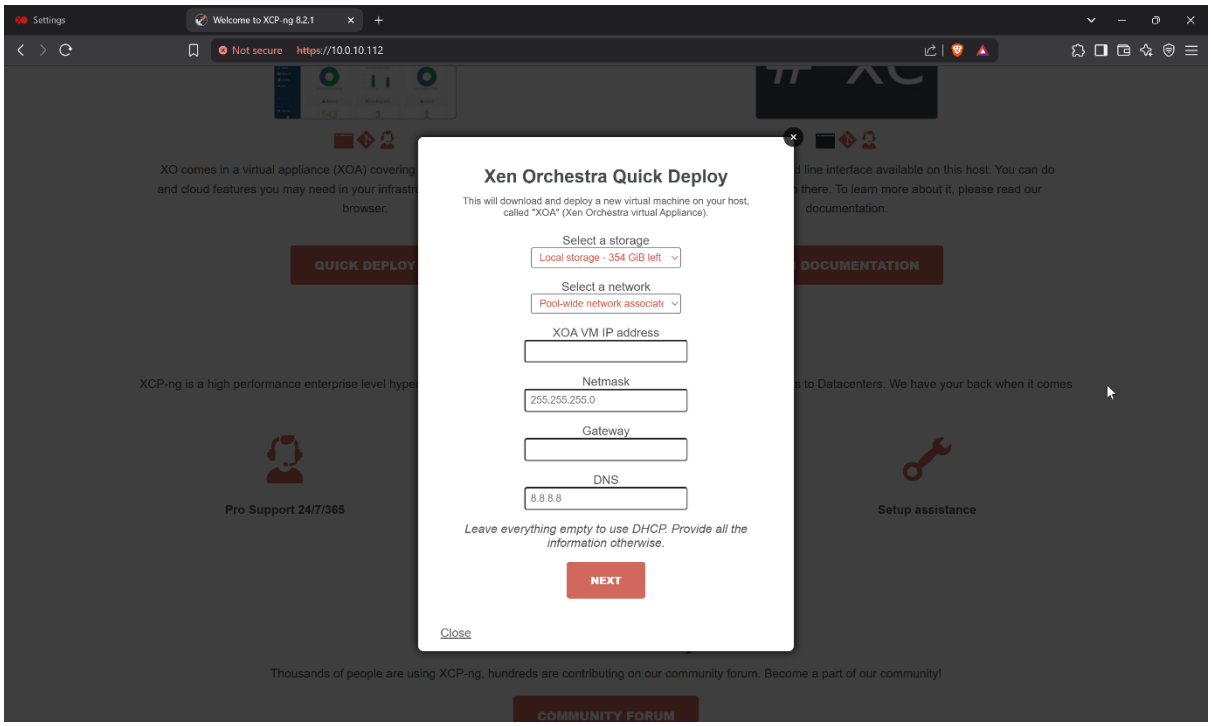

Figure 6: Deploying Xen Orchestra

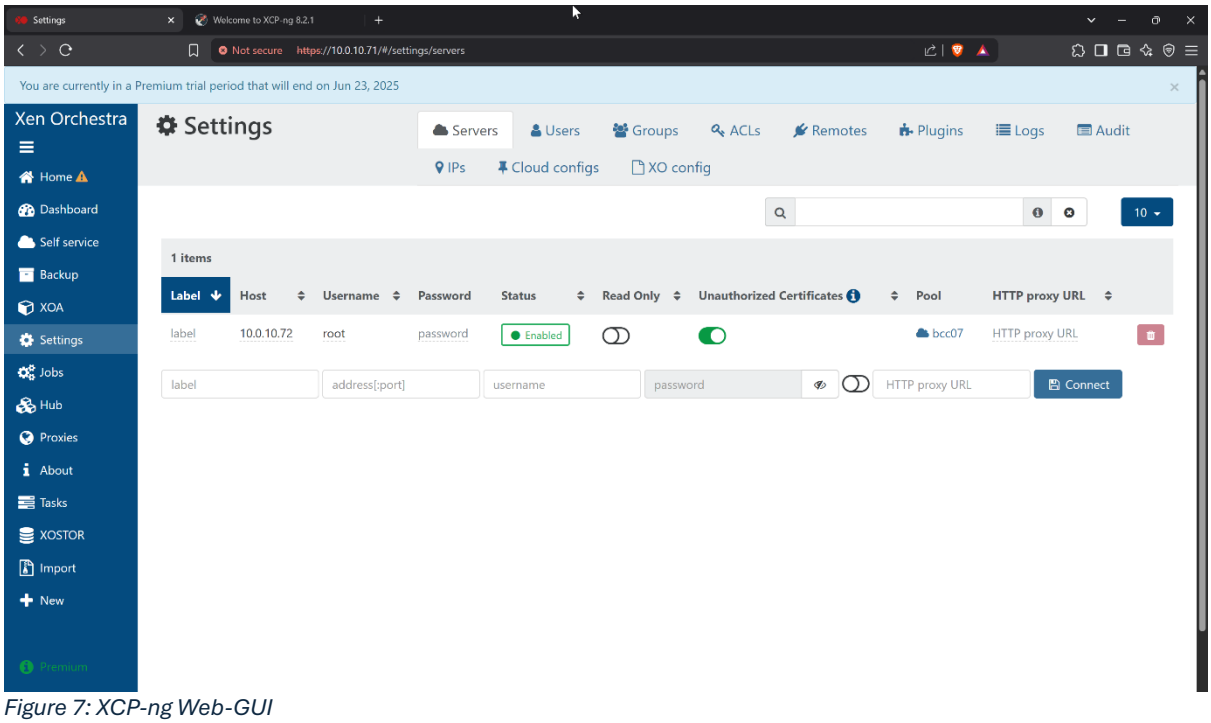

Figure 7: XCP-ng Web-GUI

6.1.3 Proxmox Installation

VE (bcc07 and bcc11): We installed Proxmox VE on both servers using a Kingston USB stick. The ISO from https://www.proxmox.com was flashed with Rufus. On bcc07, we set the network to 10.0.10.71, and on bcc11, to 10.0.10.111, using the 160GB SATA drive. The process took ~25 minutes per server, with automated network detection simplifying setup compared to XCP-NG.

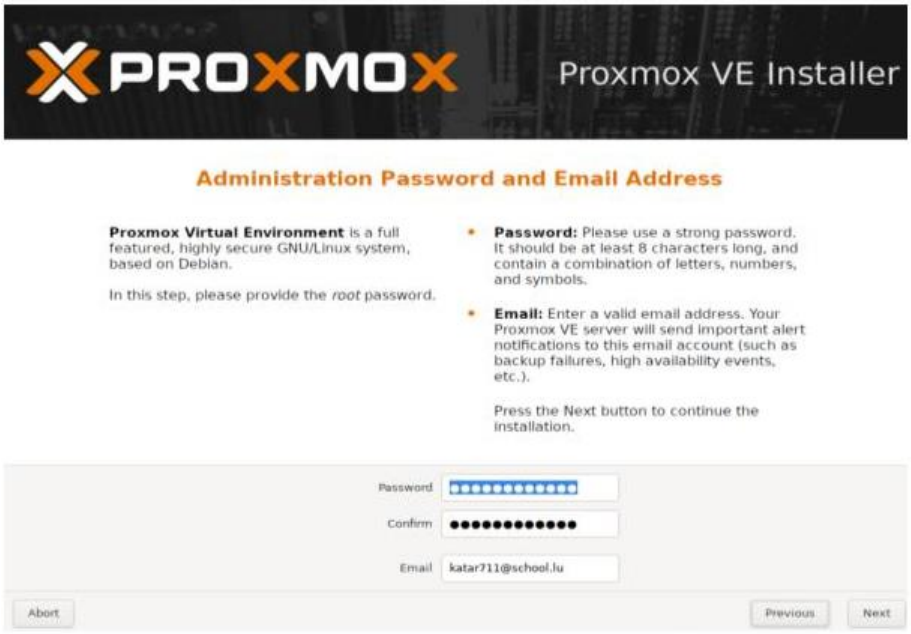

Figure 8: Configuring Password and Email

6.1.4 Network Configuration

Configuration of the Network for bcc07:

- **IP Adress**: 10.0.10.71
- **Subnet mask:** 255.255.0.0 or /16
- **Gateway:** 10.0.0.1
- **DNS:** 10.0.0.1

Configuration of the Network for bcc11:

- **IP Adress**: 10.0.10.111
- **Subnet mask:** 255.255.0.0 or /16
- **Gateway:** 10.0.0.1
- **DNS:** 10.0.0.1

Figure 10: Configuration Summary of Proxmox

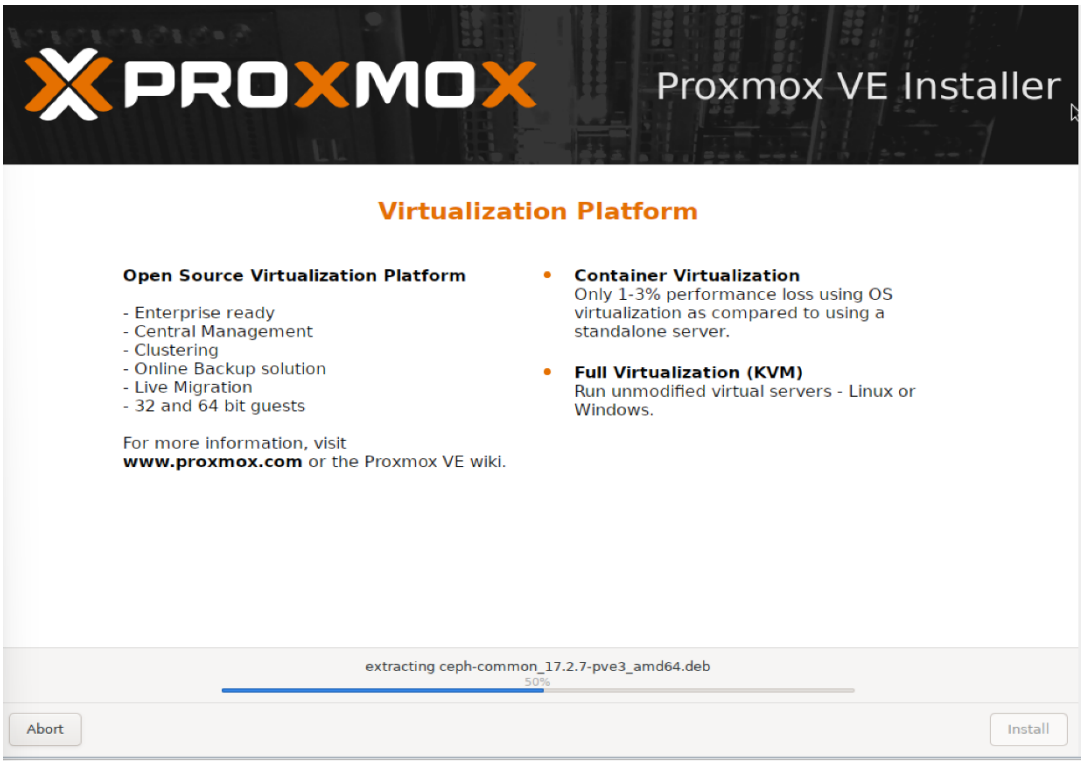

Figure 9: Installing Proxmox

After installing Proxmox, the system automatically redirects you to the web interface, which can be accessed through a URL like **https://10.0.10.71:8006**. This page provides access to the Proxmox management dashboard.

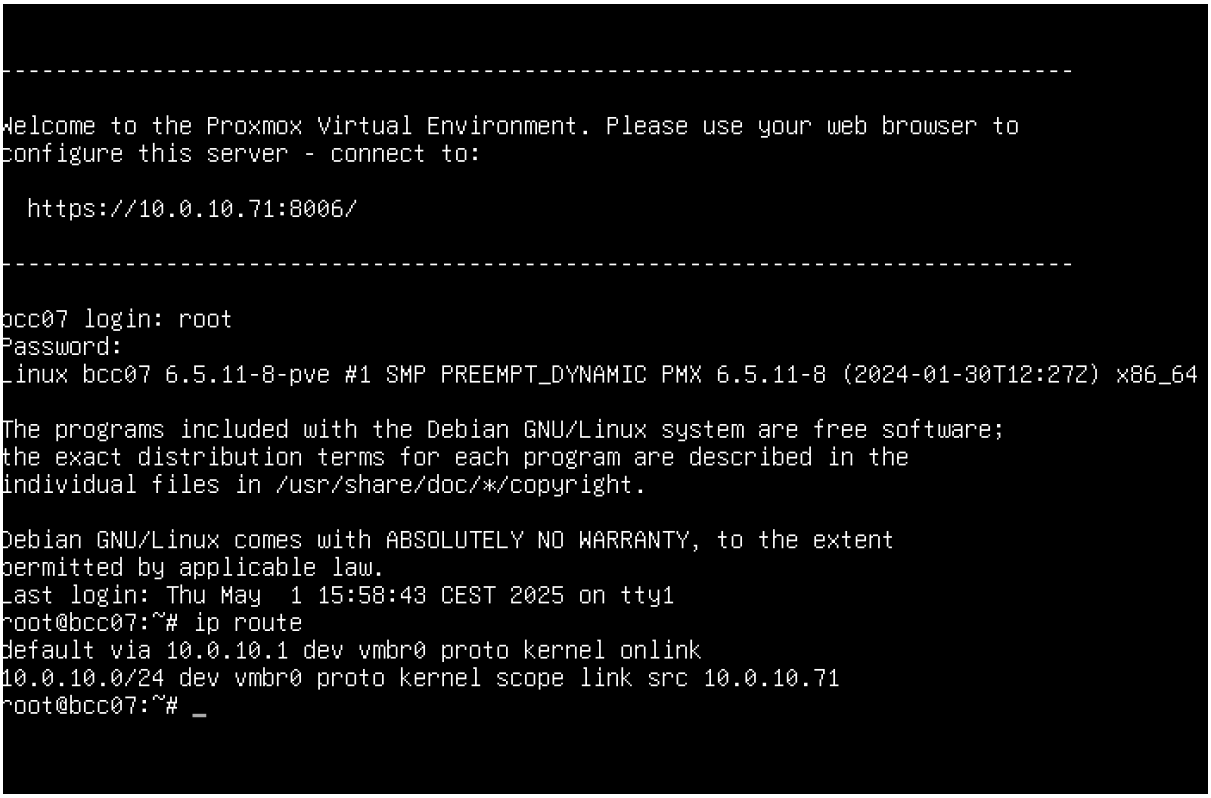

Figure 11: Proxmox VE

6.2 Creation and Configuration of VMs Using GUI

XCP-NG (bcc07 and bcc11): On both servers, we used XCP-NG Center's "New VM" wizard to create an Ubuntu 22.04 LTS VM with 2 CPUs, 4GB RAM, and a 50GB disk from the 500GB SATA drive. We selected the ISO from local storage and configured the network bridge (xenbr0). The process was consistent across bcc07 and bcc11, though

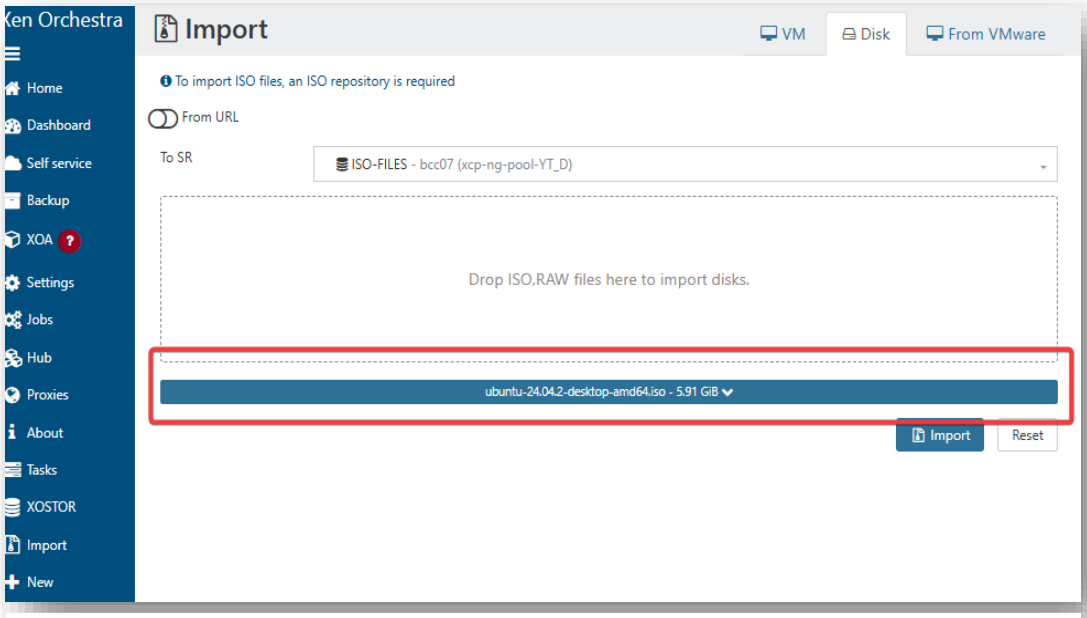

Figure 12: Uploading the iso file on to XCP-ng

Windows VMs required manual VirtIO driver selection.

To create a new VM on XCP-ng,

- Navigate to **New VM**
- Use Ubuntu 22.04 ISO (uploaded to local storage)
- Assign: 2 CPUs, 4 GB RAM, 50 GB disk (from 500GB SATA)
- Use **xenbr0** bridge

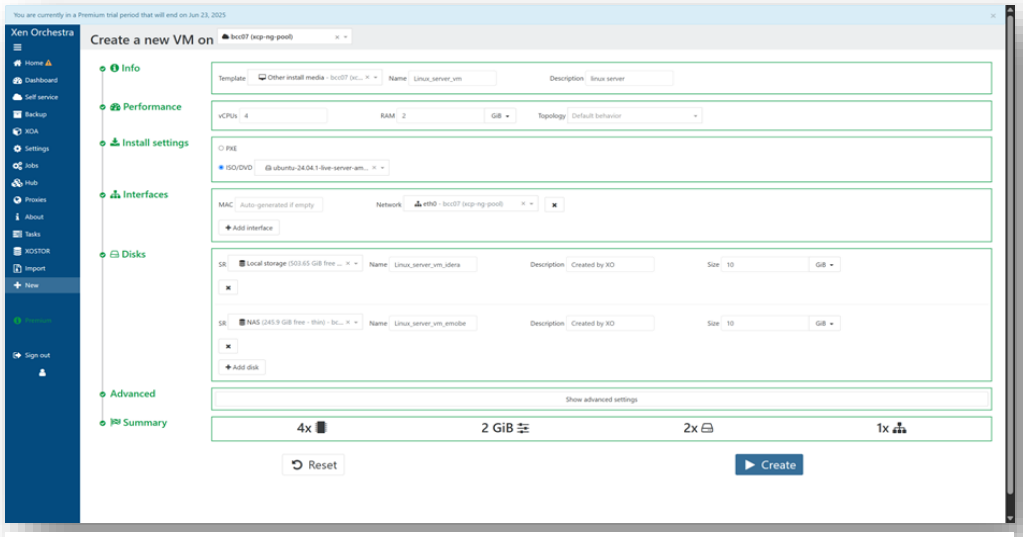

Figure 13: XCP-ng GUI Vm creation on bcc11

Proxmox VE (bcc07 and bcc11): Using Proxmox's web GUI (https://10.0.10.71:8006 for bcc07; https://10.0.10.111:8006 for bcc11), we created VMs via the "Create VM" tab, allocating 2 CPUs, 4GB RAM, and a 50GB disk for Ubuntu. The GUI offered advanced options like CPU type (kvm64) and VirtIO drivers, making configuration efficient.

To create a VM on Proxmox via Gui

- Click "Create VM":
  - CPU: 2
  - RAM: 4 GB
  - Disk: 50 GB
  - Use VirtIO network and disk drivers

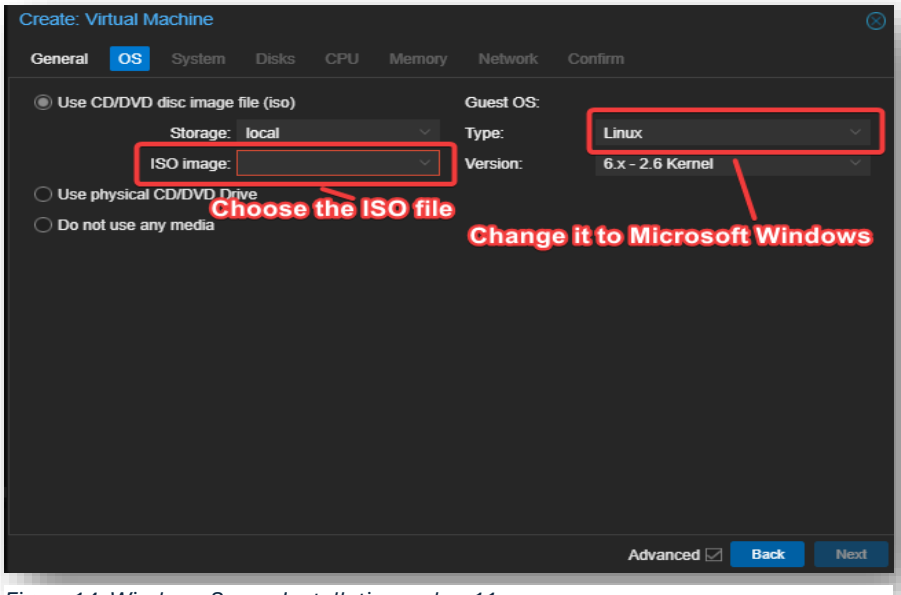

Figure 14: Windows Server Installation on bcc11

For EFI storage, we selected our NAS device, which in this case is Rockstor.

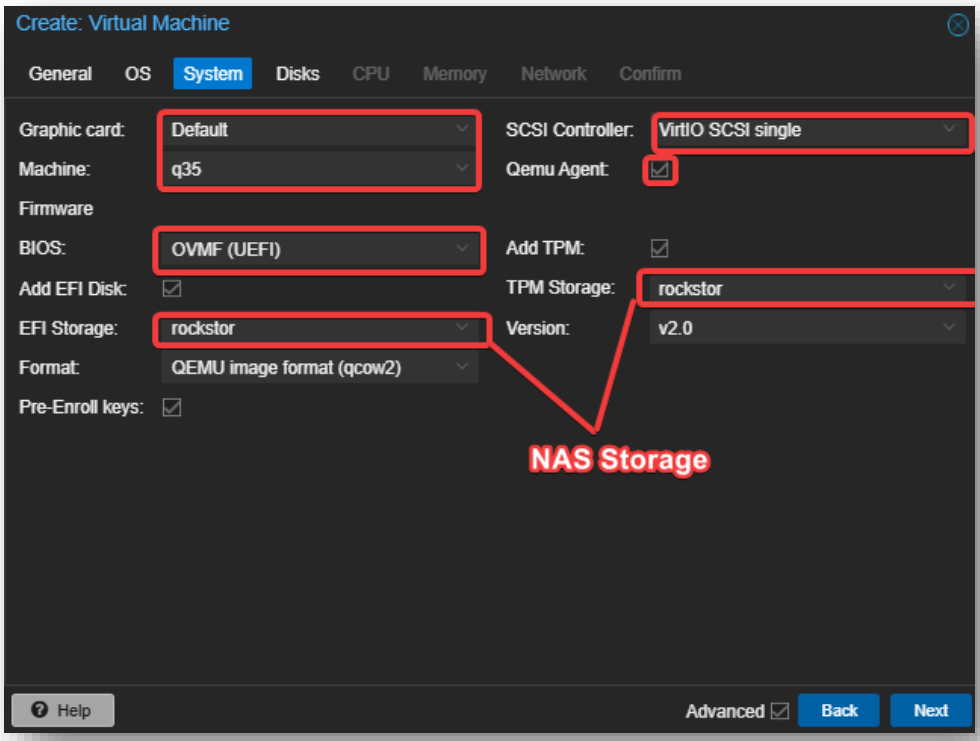

Figure 15: Connect the VM to the NAS storage

Set the storage of the Virtual Machine.

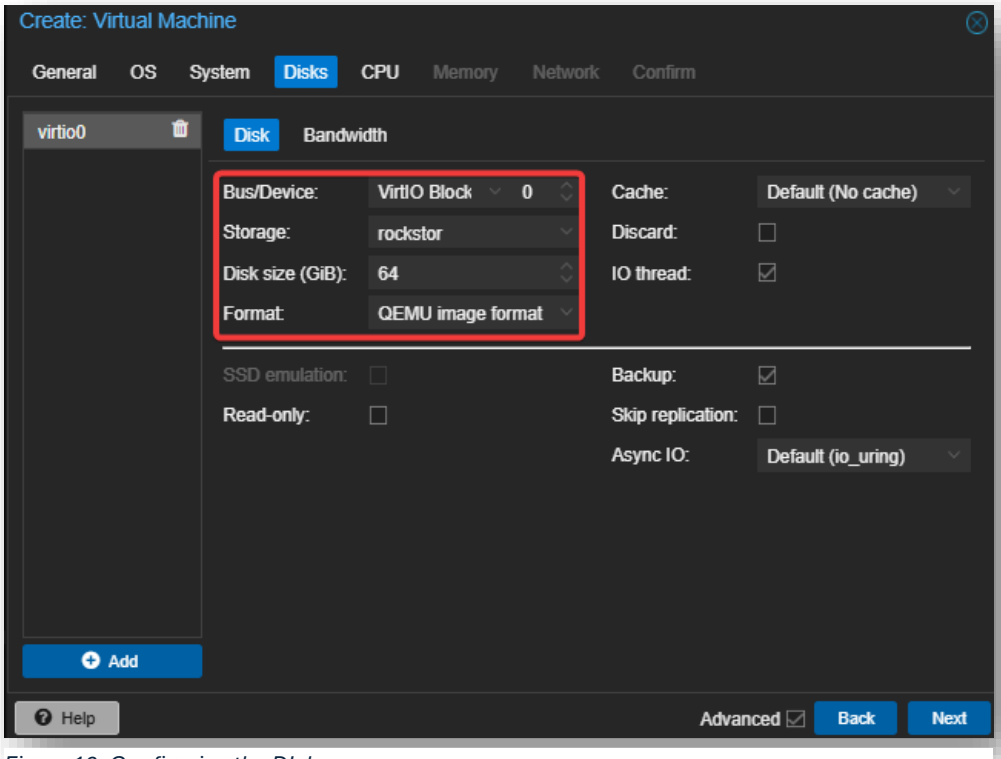

Figure 16: Configuring the DIsks

For Memory, we chose 8GB of Ram.

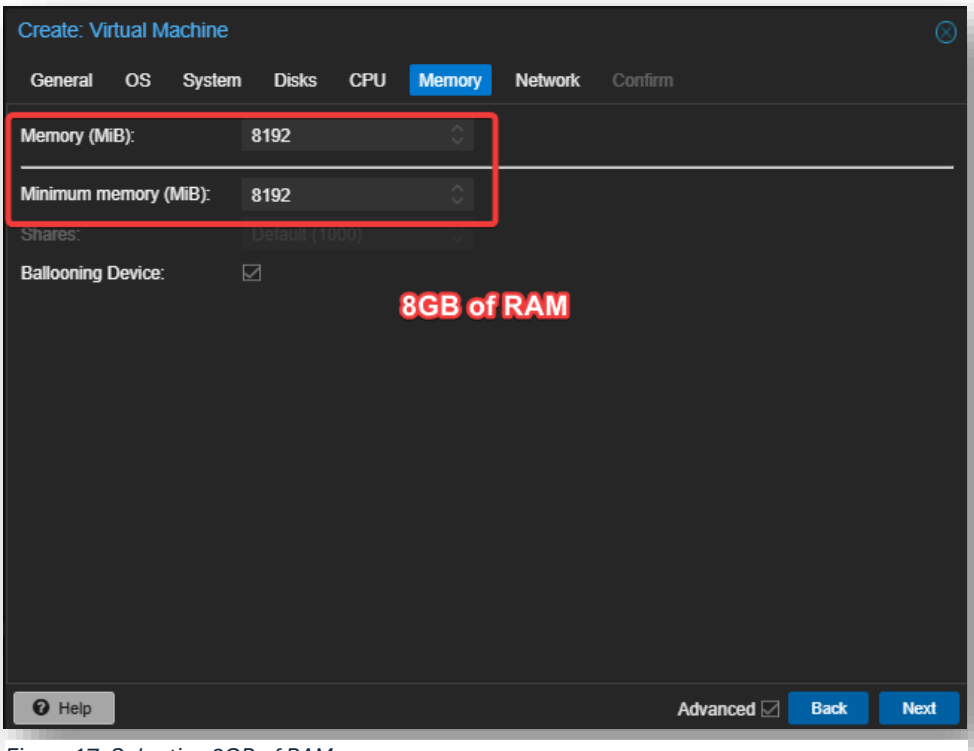

Figure 17: Selecting 8GB of RAM

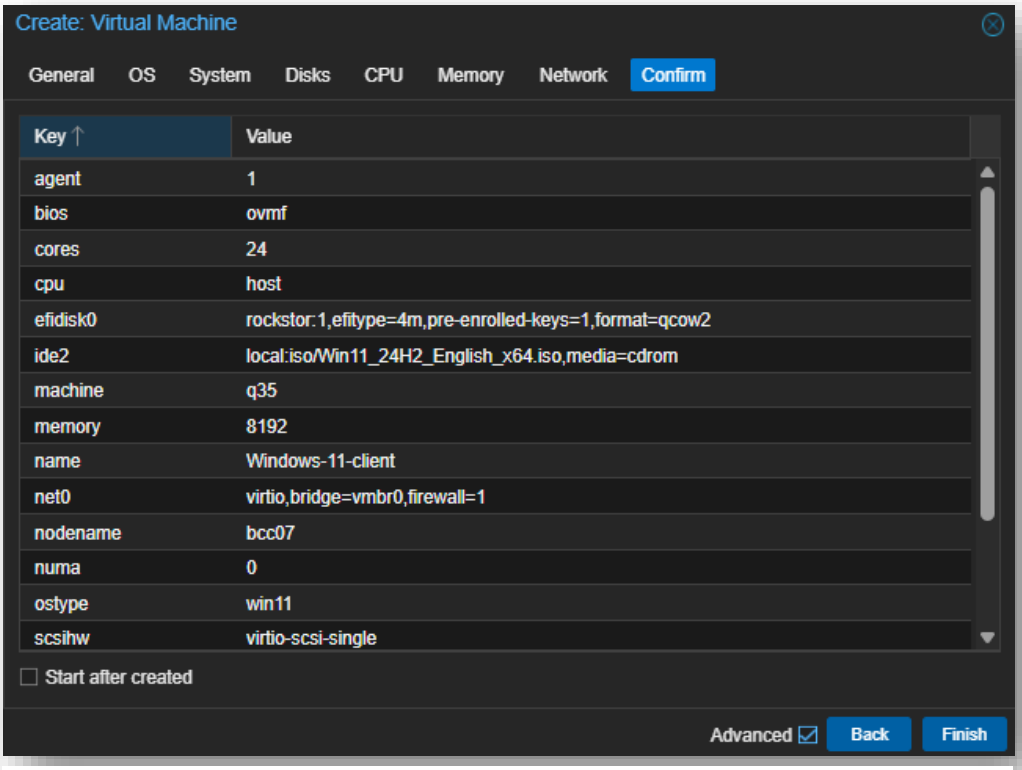

Figure 18: Configuration Summary of VM in Proxmox

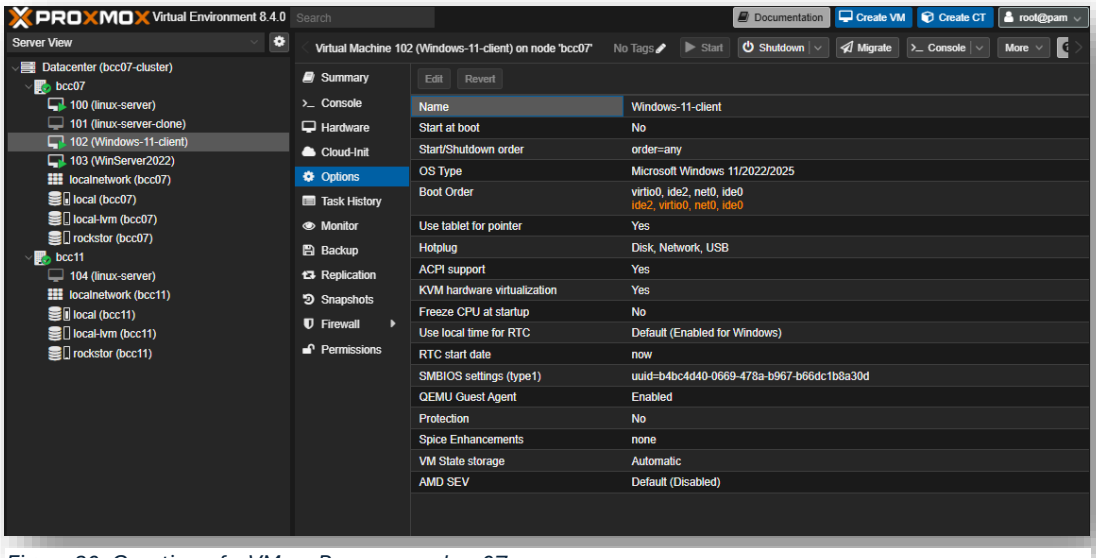

Figure 20: Creation of a VM on Proxmox on bcc07

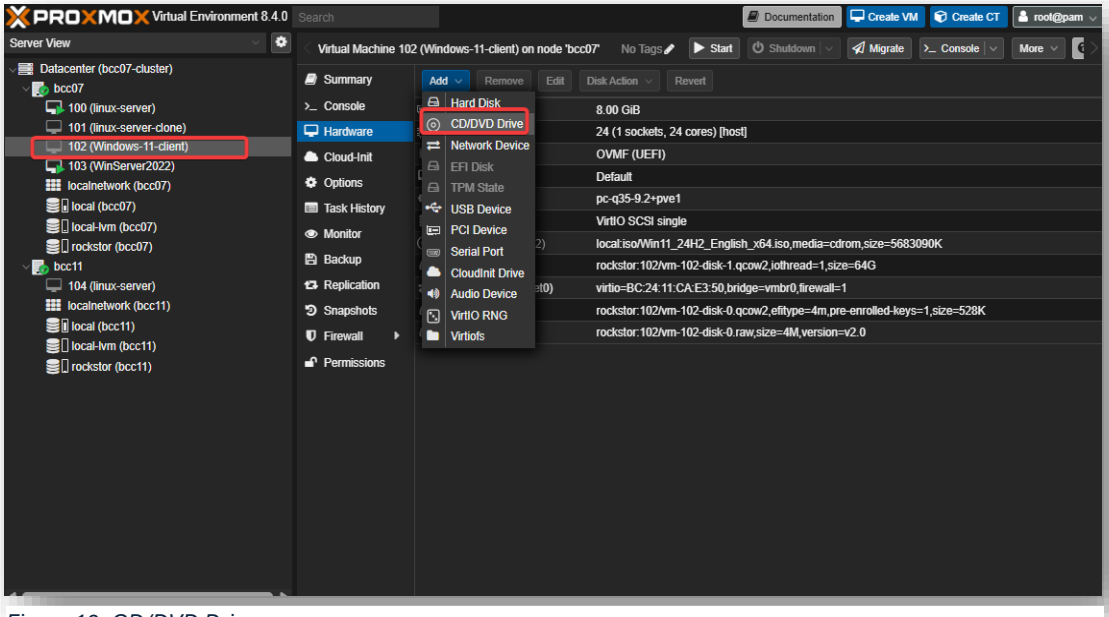

Figure 19: CD/DVD Drive

To install the Windows VM, select a location to install Windows 11 and Install a driver to show hardware.

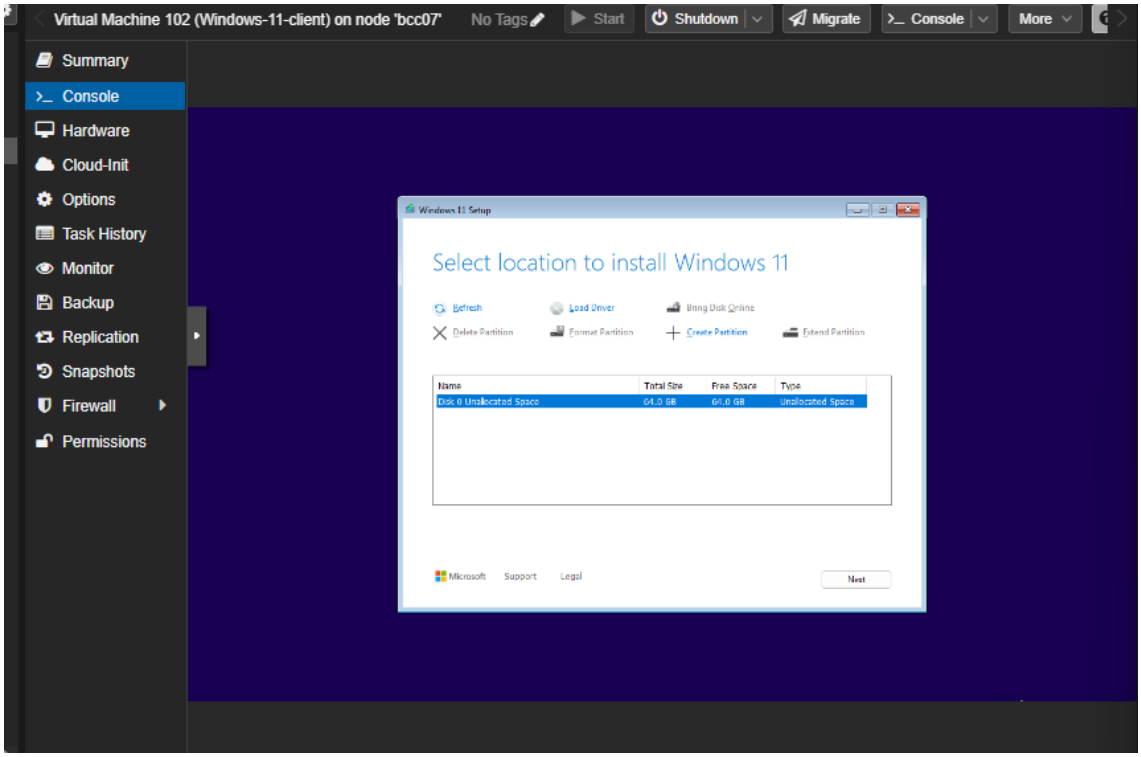

Figure 22: Select location to install Windows 11

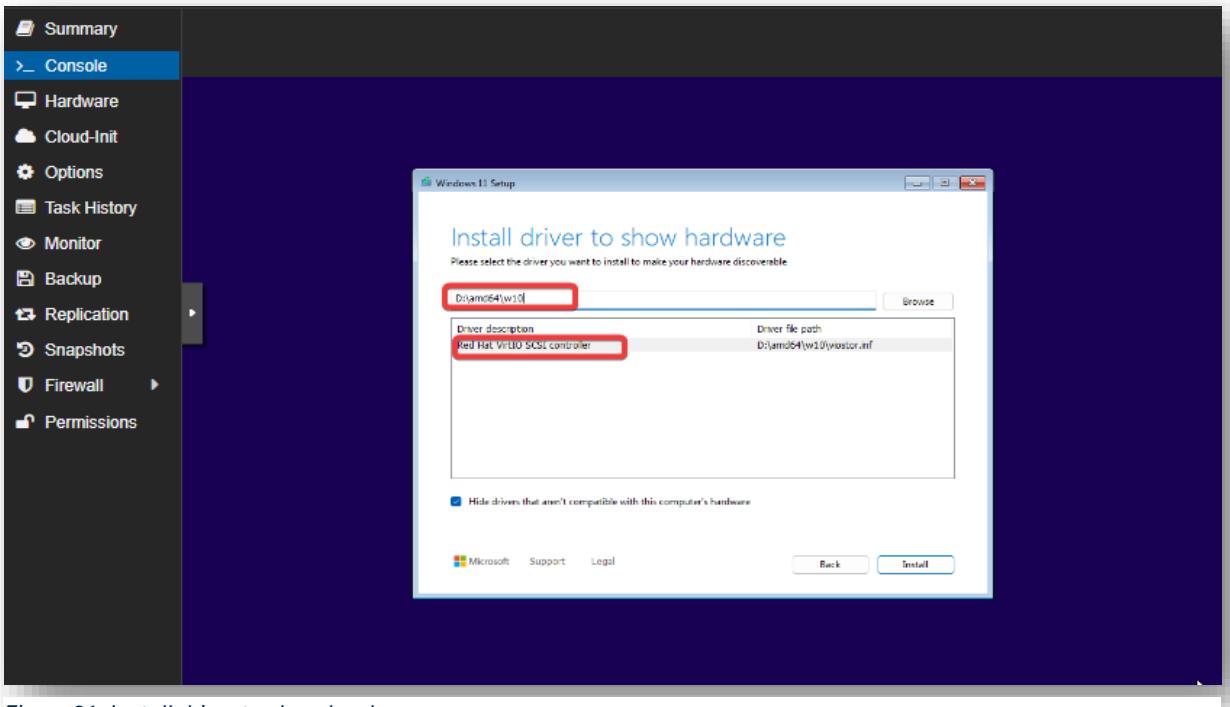

Figure 21: Install driver to show hardware

After completing all the necessary configurations, the Windows virtual machine was successfully installed and is now up and running.

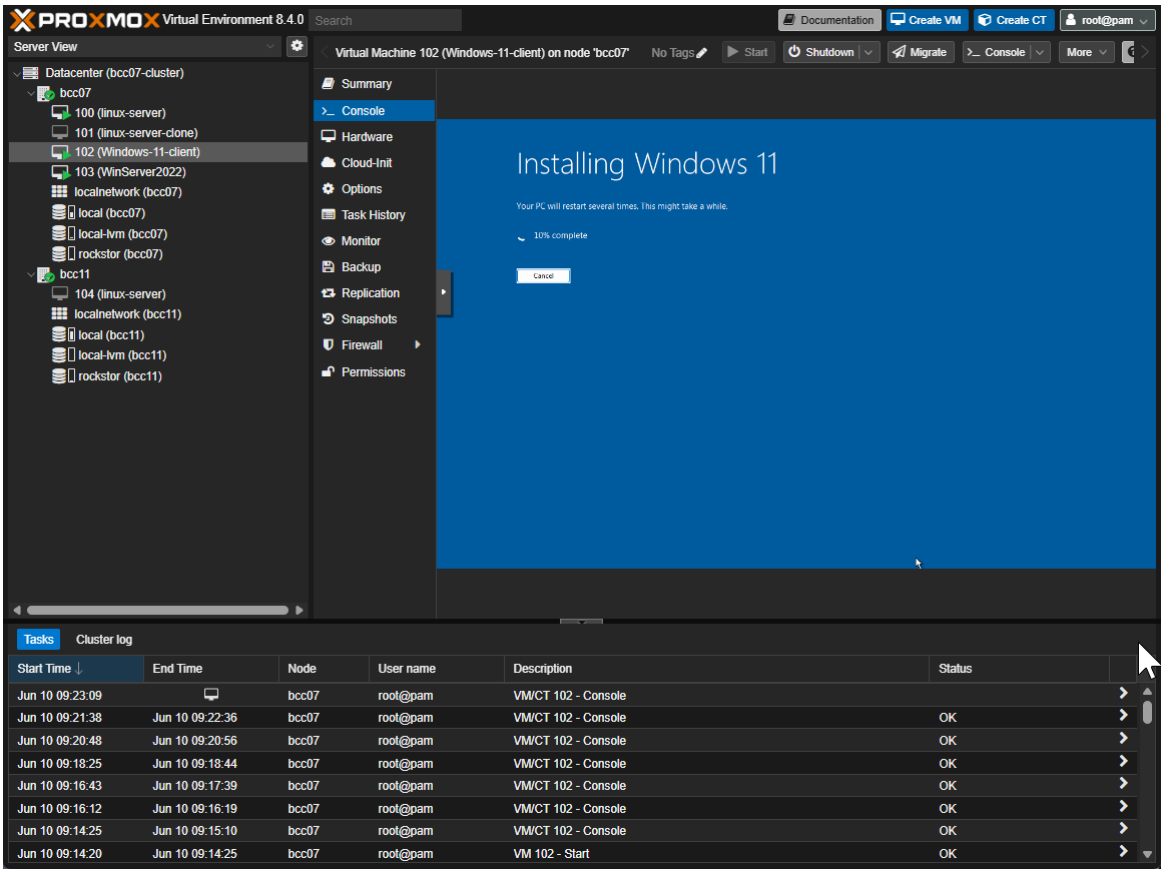

Figure 23: Windows VM has been successfully installed

XCP-NG (bcc07 and bcc11): We used the xe CLI on both servers:

```
xe vm-install template=Ubuntu\ Focal\ Fossa\ 20.04 name-label=ubuntu-vm new-uuid=$(uuidgen)

xe vm-param-set uuid=<VM-UUID> memory-static-max=4GiB VCPUs-max=2

xe vdi-create sr-uuid=<SR-UUID> name-label=vm-disk virtual-size=50GiB
```

The process required manual UUID generation, taking ~5 minutes per server.

```
[16:02 bcc07 ~]# xe vm-install template=Linux_Server_Vm new-name-label=Linux_Server_vm_CLI
b60f94e3-80dc-c5a0-79b8-02b6ea5e62f8
```

Figure 24:Installing a VM template in XCP-ng

```
[16:07 bcc07 ~]# xe vm-param-set uuid=b60f94e3-80dc-c5a0-79b8-02b6ea5e62f8 memory-static-max=2147483648 memory-dynamic-max=2147483648 memory-dynamic-min=2147483648
The dynamic memory range violates constraint static_min <= dynamic_min <= dynamic_max <= static_max.
[16:08 bcc07 ~]# xe vm-param-set uuid=b60f94e3-80dc-c5a0-79b8-02b6ea5e62f8 VCPUs-max=2 VCPUs-at-startup=2^C
[16:10 bcc07 ~]# xe vm-param-set uuid=b60f94e3-80dc-c5a0-79b8-02b6ea5e62f8 \
>    memory-static-min=134217728 \
>    memory-dynamic-min=2147483648 \
>    memory-dynamic-max=2147483648 \
>    memory-static-max=2147483648
[16:12 bcc07 ~]# xe vm-param-set uuid=b60f94e3-80dc-c5a0-79b8-02b6ea5e62f8 \
>    VCPUs-max=2 VCPUs-at-startup=2
```

Figure 25: Installed template in XCP-ng

Proxmox VE (bcc07 and bcc11): On both servers, we used:

Option 1: We create a bash script with all the configuration set. View the image below:

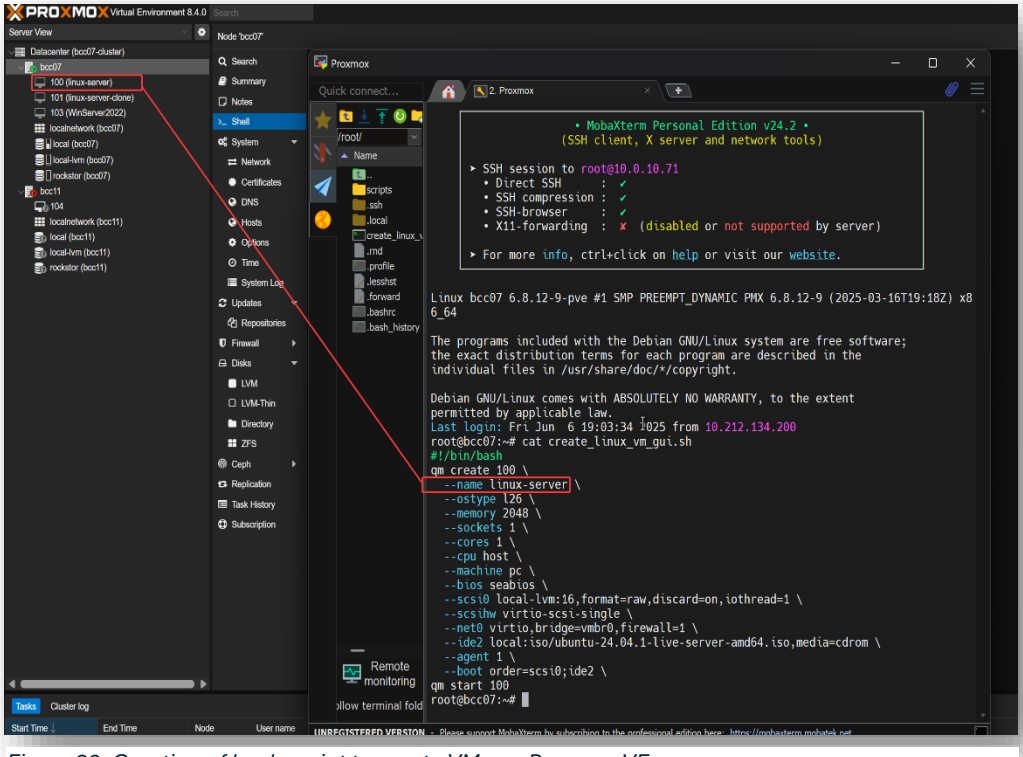

Figure 26: Creation of bash script to create VMs on Proxmox VE

Option 2 :

```
qm create 100 --name linux-server --memory 2048 --cores 1 --net0
virtio,bridge=vmbr0,firewall=1 --scsi0 local-lvm:16
```

The qm command was faster (~1 minute) and well-documented.

6.3 Installation of Windows and Linux OS

XCP-NG (bcc07 and bcc11): We mounted Windows Server 2022 and Ubuntu Server 22.04 LTS ISOs via XOA(Xen Orchestra) “Storage” tab on both servers. Windows required VirtIO drivers (from https://fedorapeople.org) for disk and network, installed manually. Ubuntu installation took ~15 minutes without issues.

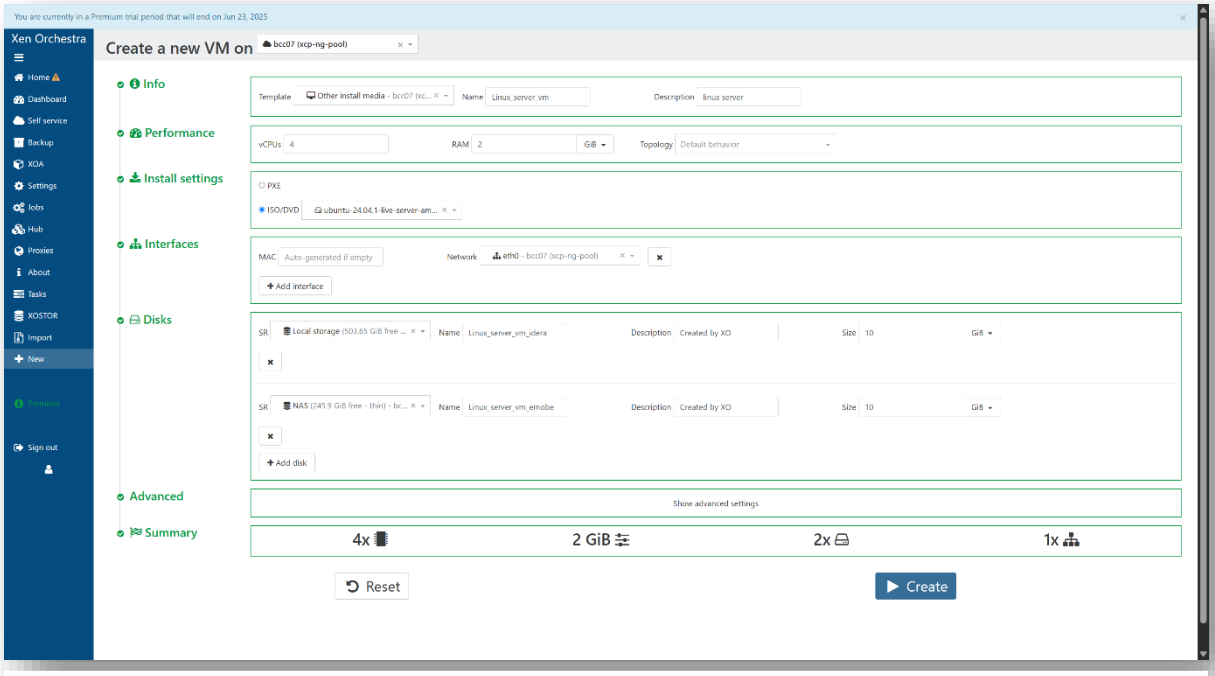

Figure 27: VM creation template on XCP-ng

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we uploaded ISOs to local storage andattached them to VMs. Windows Server 2022 installed smoothly with built-in VirtIO support in ~20 minutes. Ubuntu installed similarly, requiring no additional drivers.

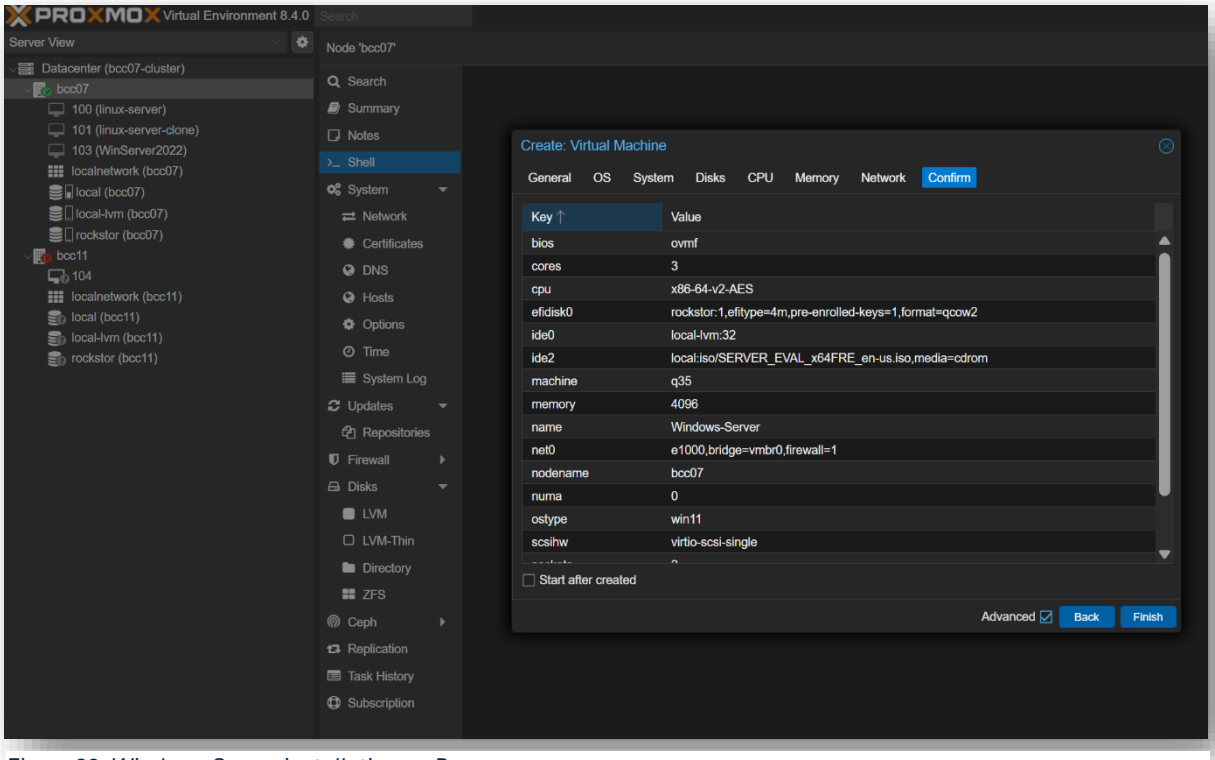

Figure 28: Windows Server isntallation on Proxmox

6.4 Changing VM Configuration

XCP-NG (bcc07 and bcc11): We modified VM resources (added 1 CPU, 2GB RAM, 20GB disk) via XCP-NG Center's "VM Properties" or CLI:

```
xe vm-param-set uuid=<VM-UUID> memory-static-max=6GiB VCPUs-max=3

xe vdi-create sr-uuid=<SR-UUID> name-label=extra-disk virtual-size=20GiB
```

Changes required a VM restart, taking ~2 minutes.

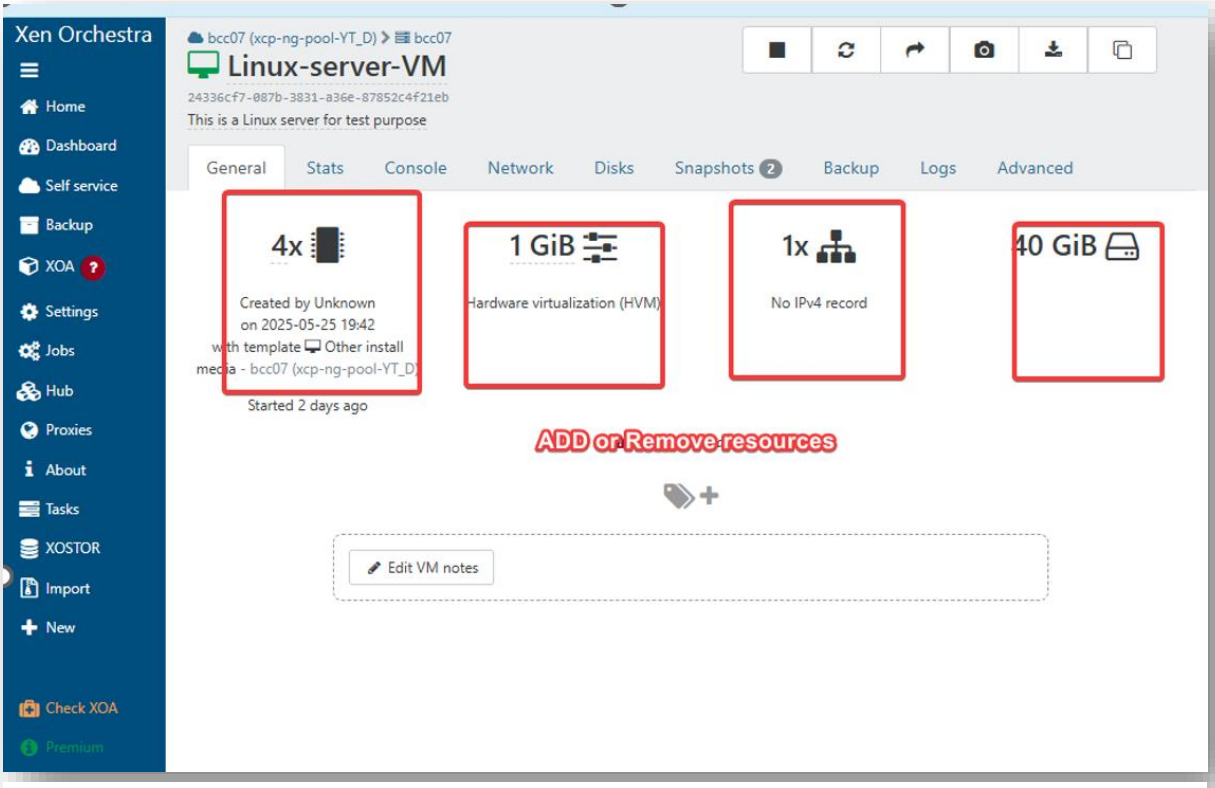

Figure 29: XCP-ng VM resource modification on bcc07

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI "Hardware" tab, we added 1 CPU, 2GB RAM, and a 20GB disk. CLI command:

```
qm set 100 --memory 6144 --cores 3 --scsi1 local-lvm:20
```

Live changes were supported, taking ~1 minute.

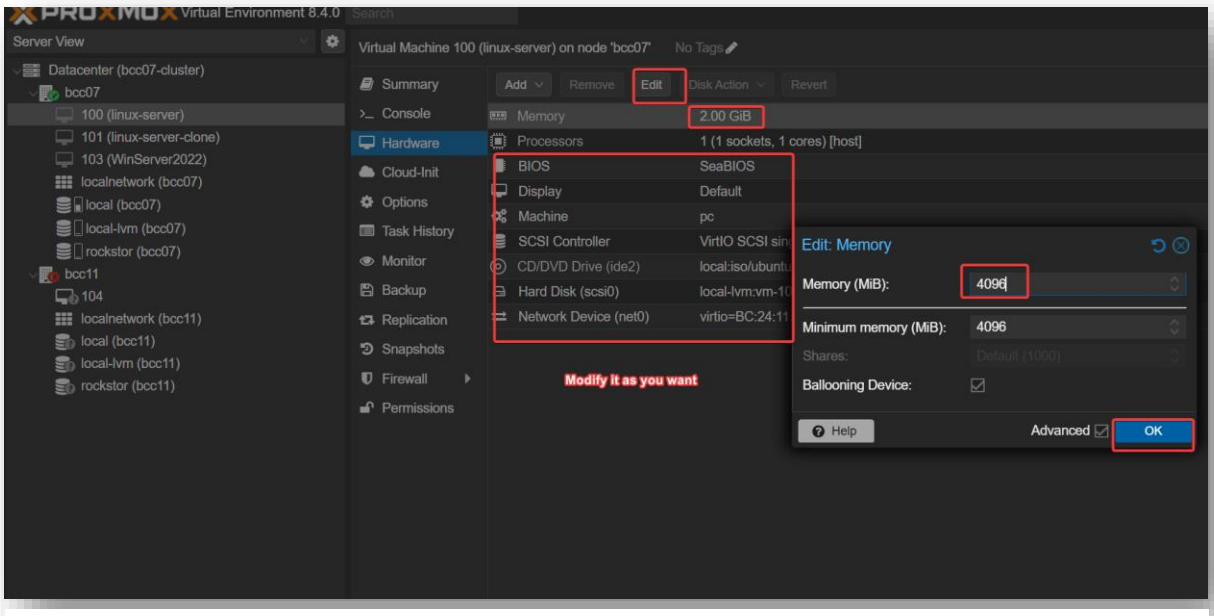

Figure 30: Proxmox VM resource modification on bcc07

6.5 Cloning a VM

XCP-NG (bcc07 and bcc11): We cloned a 50GB Ubuntu VM using XCP-NG Center's "Copy VM" option (full clone), taking ~8 minutes per server.

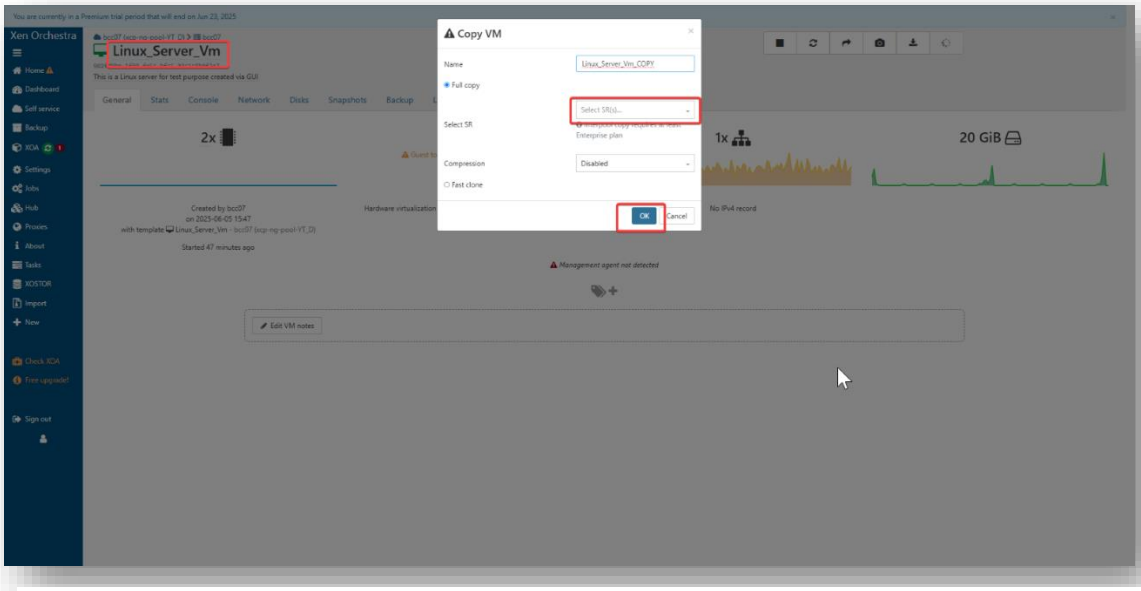

Figure 31: XCP-ng VM cloning on bcc11

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we used the "Clone" option, selecting full or linked clones. Full clones took ~7 minutes, linked clones ~2 minutes.

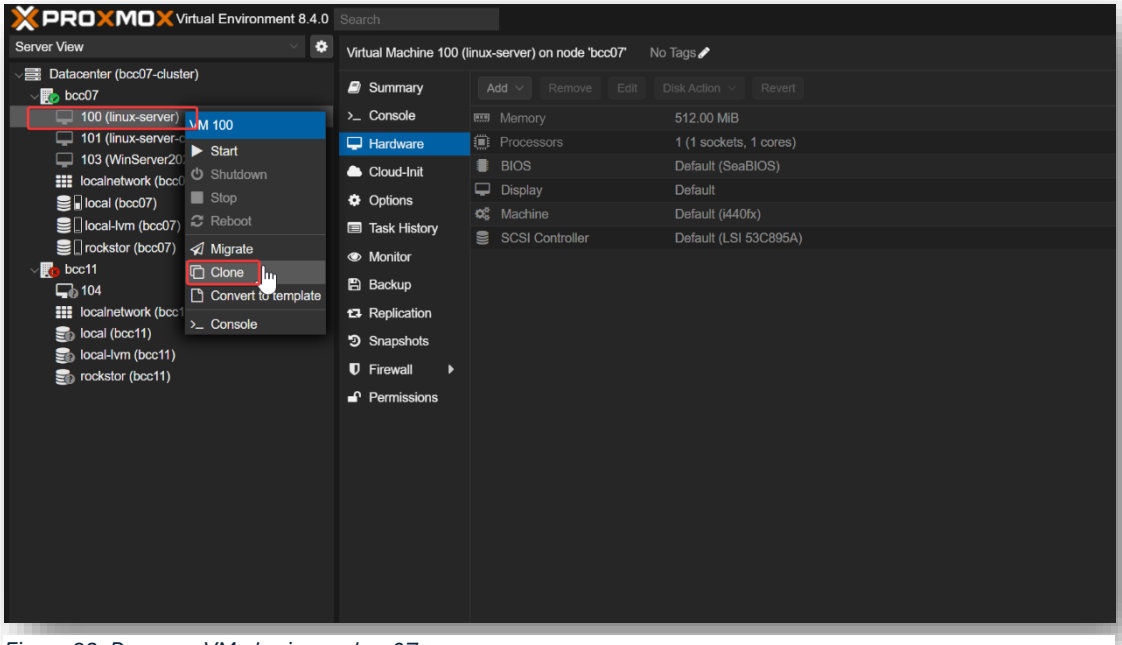

Figure 32: Proxmox VM cloning on bcc07

6.6 Exporting a VM

XCP-NG (bcc07 and bcc11): We exported a VM as an OVA file using:

```
xe vm-export uuid=<VM-UUID> filename=/mnt/nas/ubuntu-vm.ova
```

To export a virtual machine in Xen Orchestra, first open XO and navigate to the "VM" section. Select the VM you want to export, then go to the "Backup" tab and choose the "Export" option. From there, select your preferred export format (such as XVA or OVA) and follow the prompts to complete the export process.

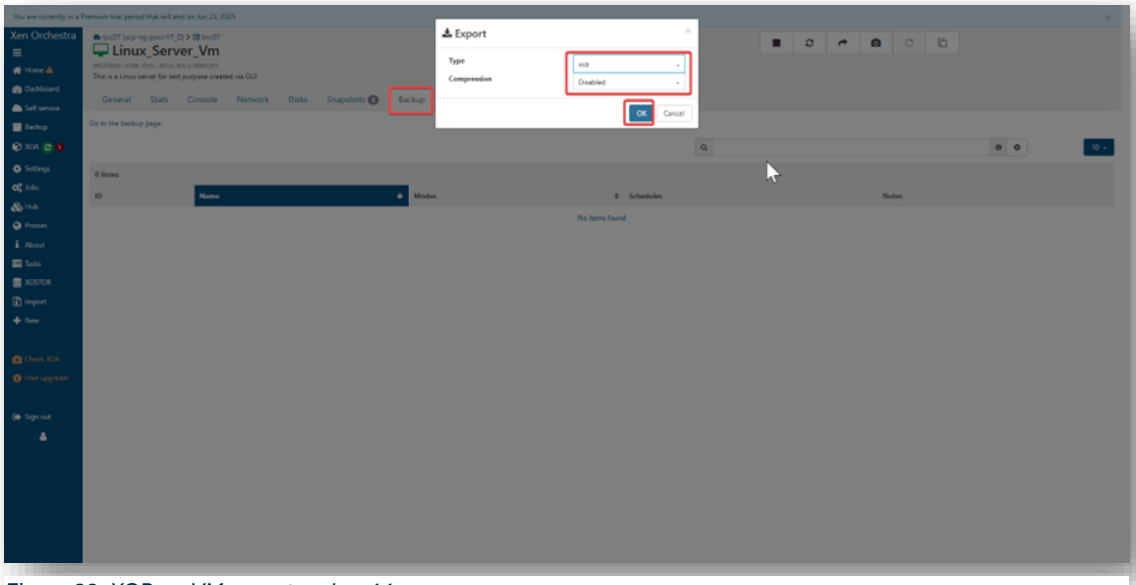

Figure 33: XCP-ng VM export on bcc11

The process took ~15 minutes for a 50GB VM, saved to the CLOIF2 NAS.

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we exported a VM as a VMA file via the "Backup" tab, saved to the CLOIF2 NAS in ~10 minutes.

6.7 Backing Up Hypervisor Configuration

XCP-NG (bcc07 and bcc11): We backed up the hypervisor configuration using:

```
xe pool-dump-database file-name=/mnt/nas/xcp-ng-backup.db
```

The file was saved to the CLOIF2 NAS via NFS in ~2 minutes.

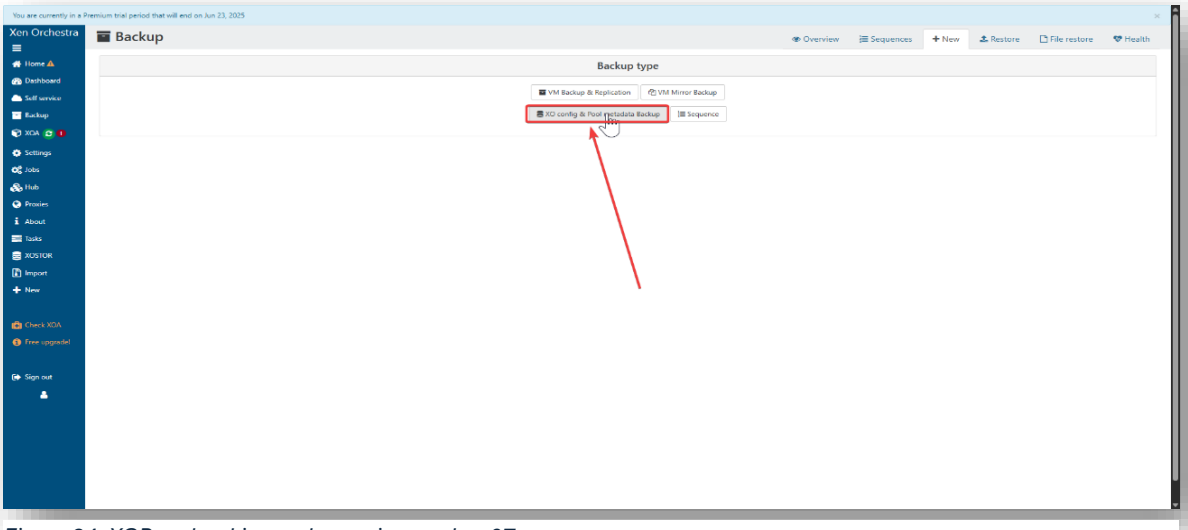

Figure 34: XCP-ng backing up hypervisor on bcc07

Proxmox VE (bcc07 and bcc11): We used Proxmox's GUI "Datacenter > Backup" to save the configuration to the CLOIF2 NAS via NFS in ~1 minute.

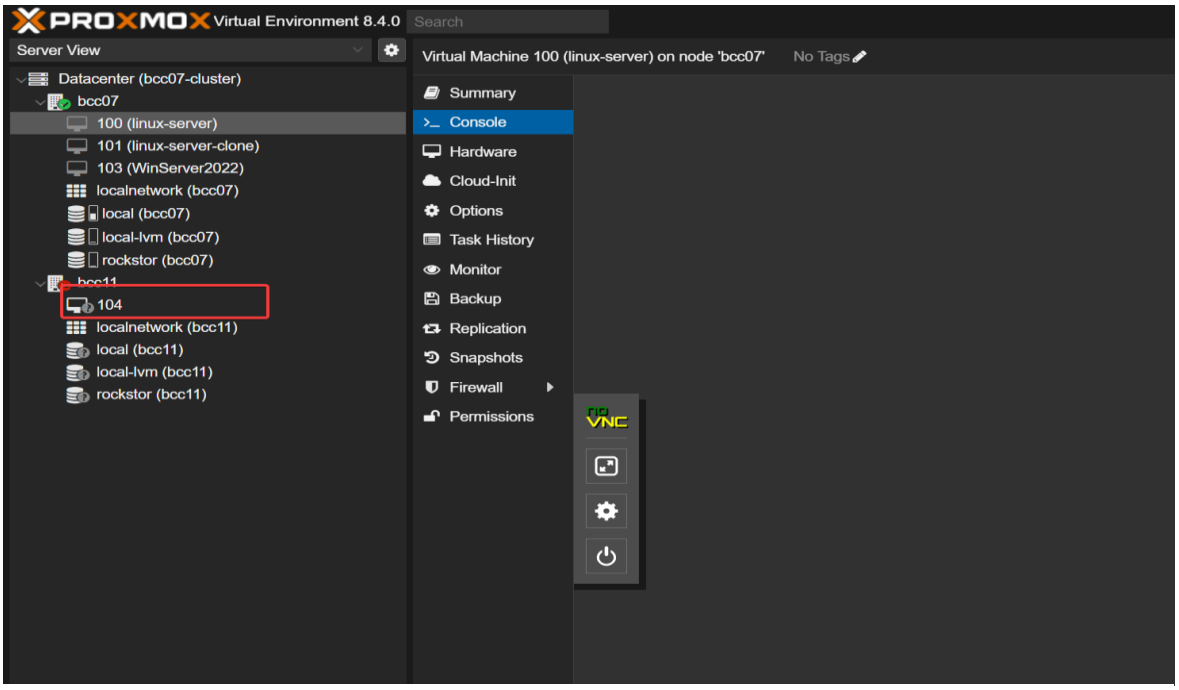

Figure 35: Proxmox hypervisor Backup on bcc07

6.8 Backing Up a VM

XCP-NG (bcc07 and bcc11): We used XCP-NG Center's backup feature to back up a 50GB VM to the CLOIF2 NAS via SMB, taking ~12 minutes.

```
xe pool-dump-database file-name=/mnt/nas/xcp-ng-backup.db
```

Proxmox VE (bcc07 and bcc11): We used Proxmox Backup Server via the GUI to back up a VM to the CLOIF2 NAS via NFS with compression and deduplication, taking ~8 minutes.

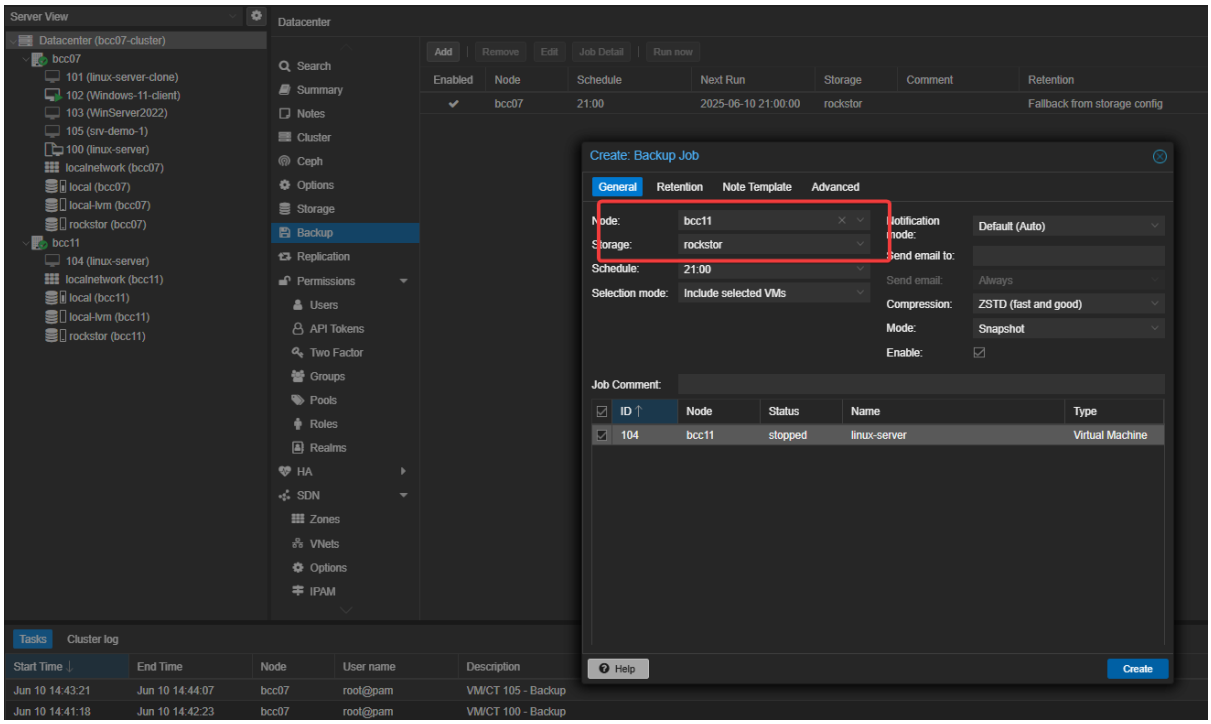

Figure 37:VM backup on Proxmox

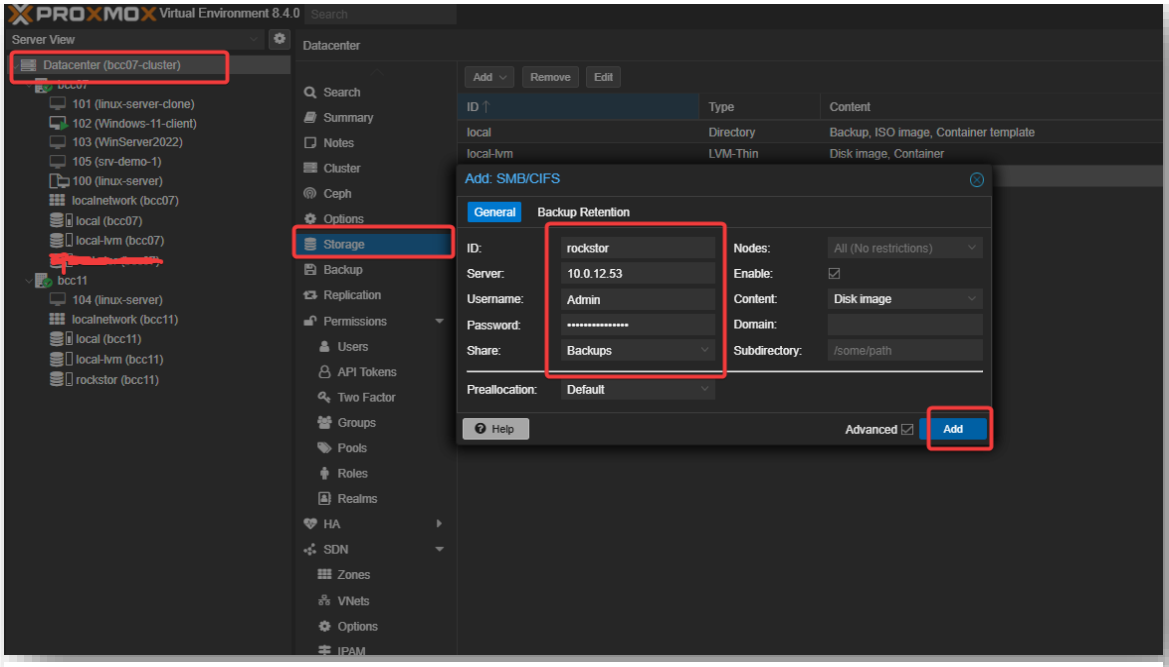

Figure 36: Adding SMB/CIFS to share Backups

6.9 Creating Snapshots

XCP-NG (bcc07 and bcc11): We created snapshots via XCP-NG Center's "Snapshots" tab, requiring VMs to be powered off for consistency. Restoring a 50GB VM snapshot took ~2 minutes.

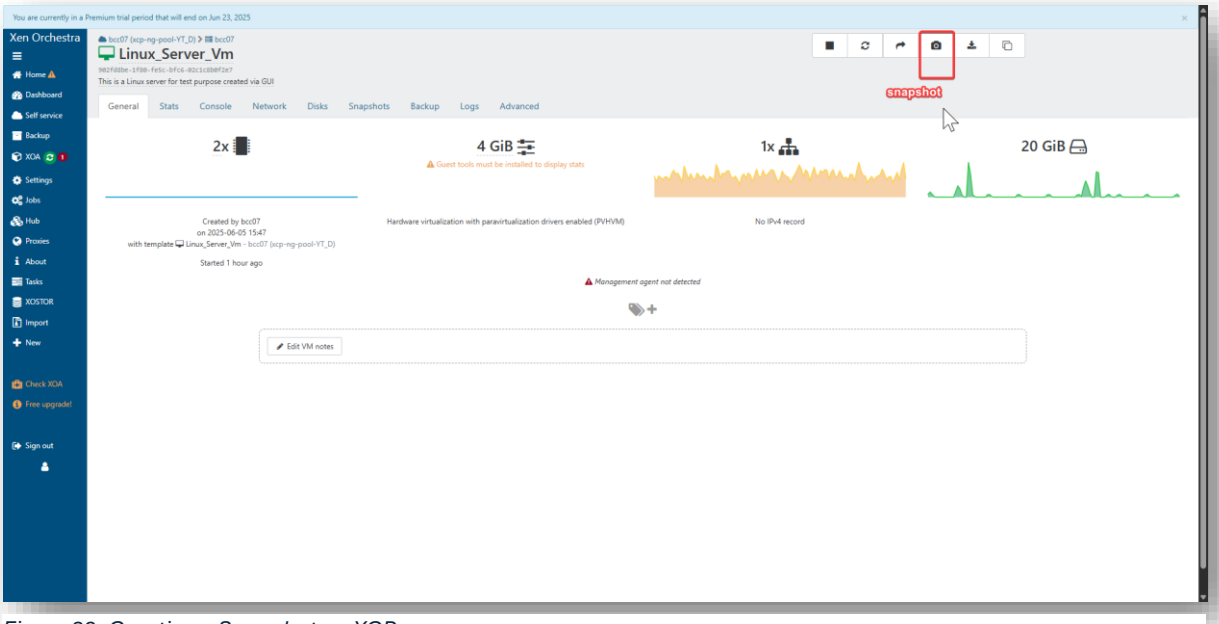

Figure 38: Creating a Snapshot on XCP-ng

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we created live snapshots via the "Snapshots" tab with no downtime, restoring in ~1 minute.

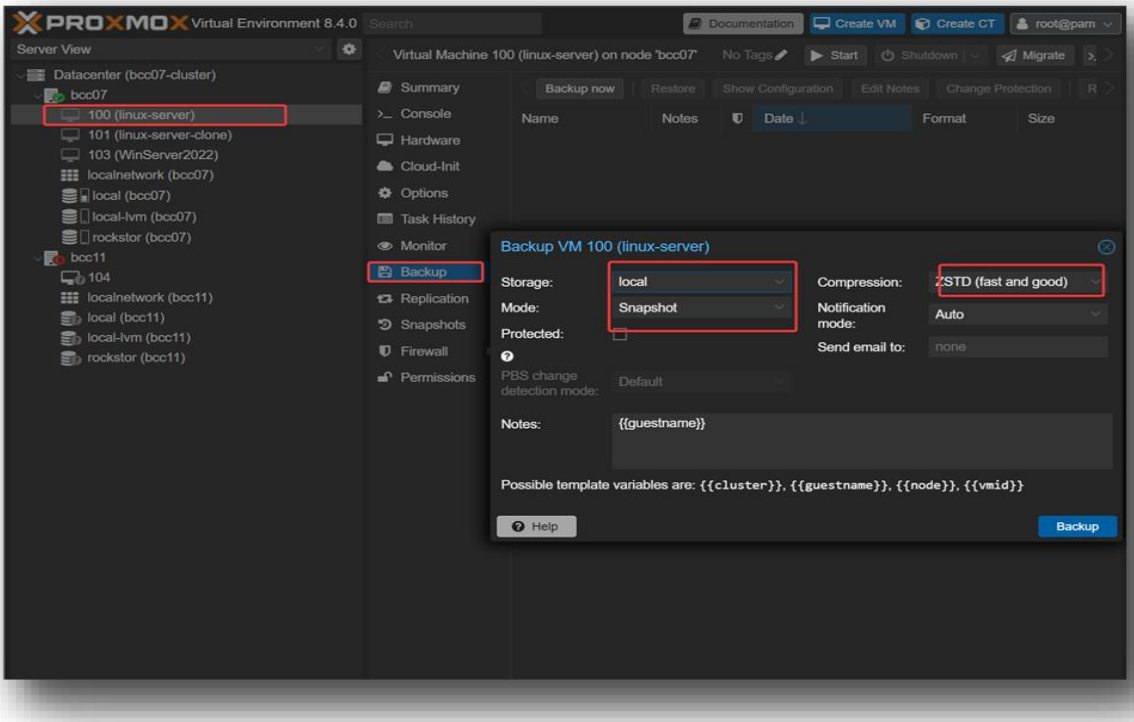

Figure 39: Creating a Snapshot on Proxmox

6.10 Creating VM Templates

XCP-NG (bcc07 and bcc11): We created Ubuntu 22.04 templates using:

```
xe vm-export uuid=<VM-UUID> filename=/mnt/nas/ubuntu-template.xva

xe template-import filename=/mnt/nas/ubuntu-template.xva
```

GUI:

To convert a virtual machine into a template in Xen Orchestra, go to **XO**, navigate to the **"VM"** section, select your desired VM, click on **"Advanced"**, and then choose **"Convert to template"**.

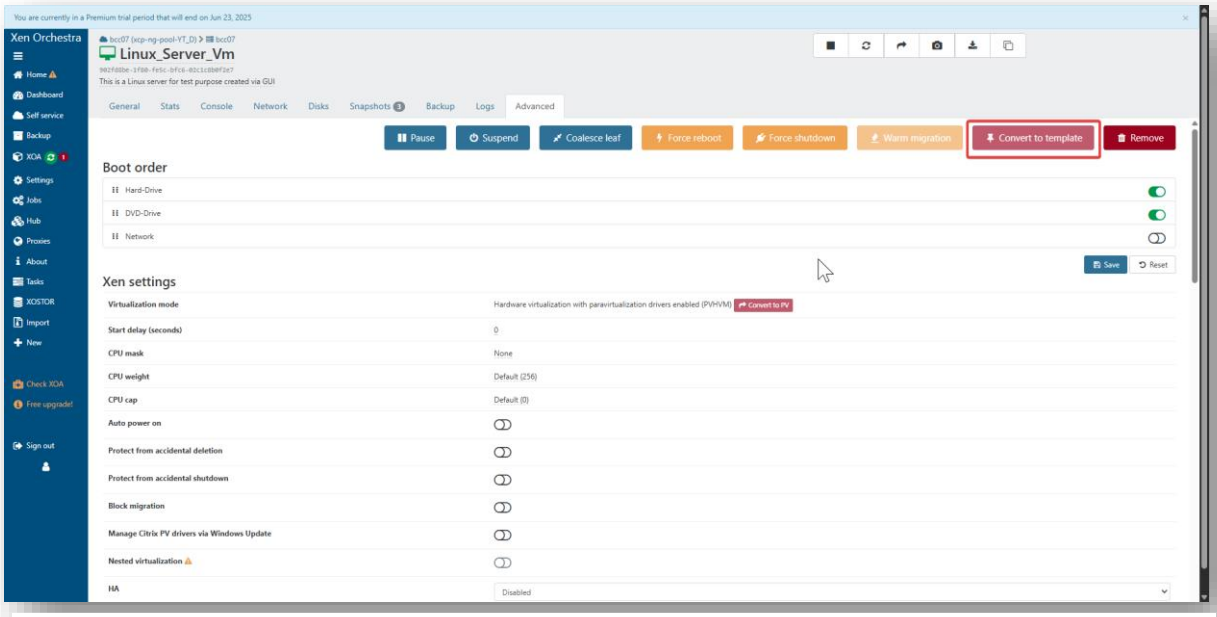

Figure 40: XCP-ng VM template creation

The CLI process took ~10 minutes per server.

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we converted VMs to templates via "Convert to Template," taking ~3 minutes.

6.11 Using Remote Storage

XCP-NG (bcc07 and bcc11): We configured the CLOIF2 NAS as an NFS storage repository:

```
xe sr-create name-label=nfs-storage type=nfs device-config:server=cloif2-nas device-config:serverpath=/mnt/nfs
```

SMB was also configured, but NFS was ~30% faster.

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we added NFS storage under "Datacenter > Storage," specifying the CLOIF2 NAS IP and path. SMB was slower but functional.

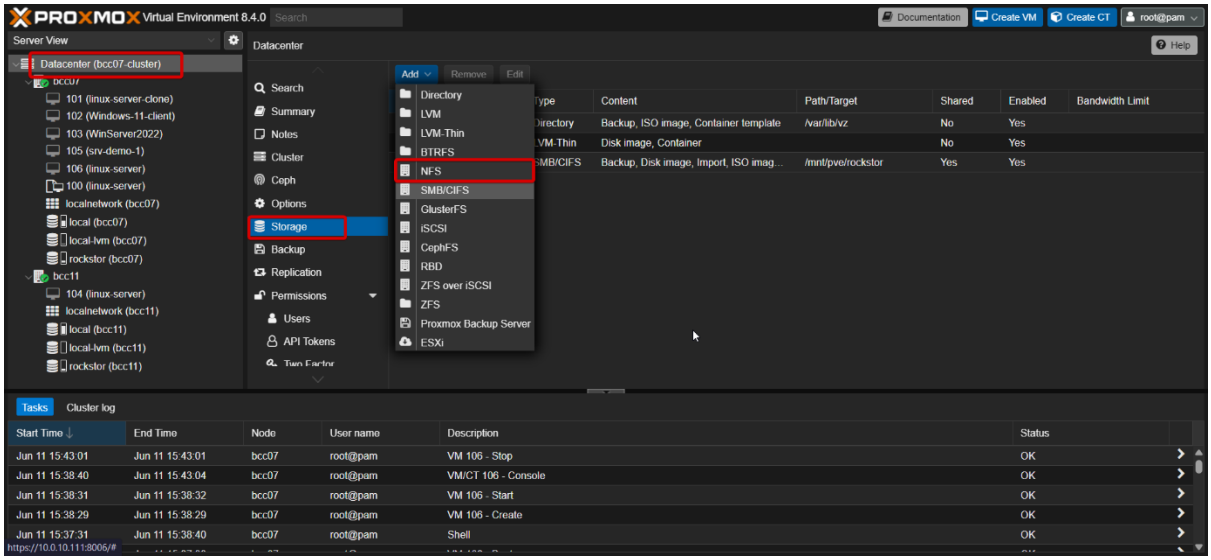

Figure 41: Addition of NFS storage

6.12 Live Migration Between Storage

XCP-NG (bcc07 and bcc11): We migrated running VMs from local storage to the CLOIF2 NAS using NFS and iSCSI via XCP-NG Center's "Migrate VM" option. NFS migration took ~15 minutes for a 50GB VM; iSCSI required manual SR setup.

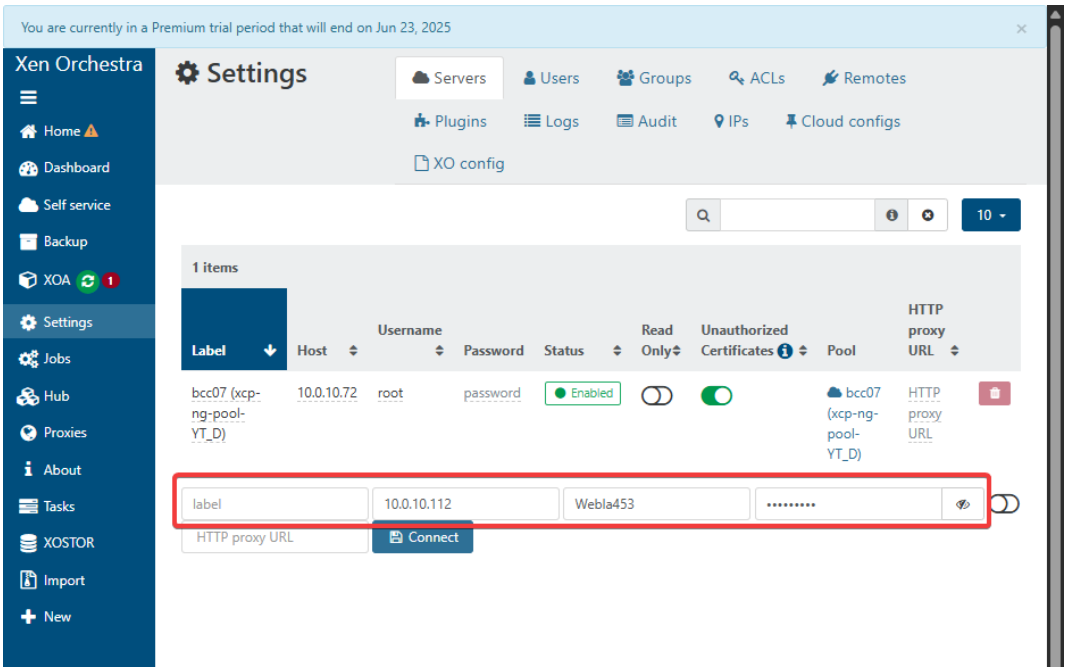

Figure 43: Pool creation

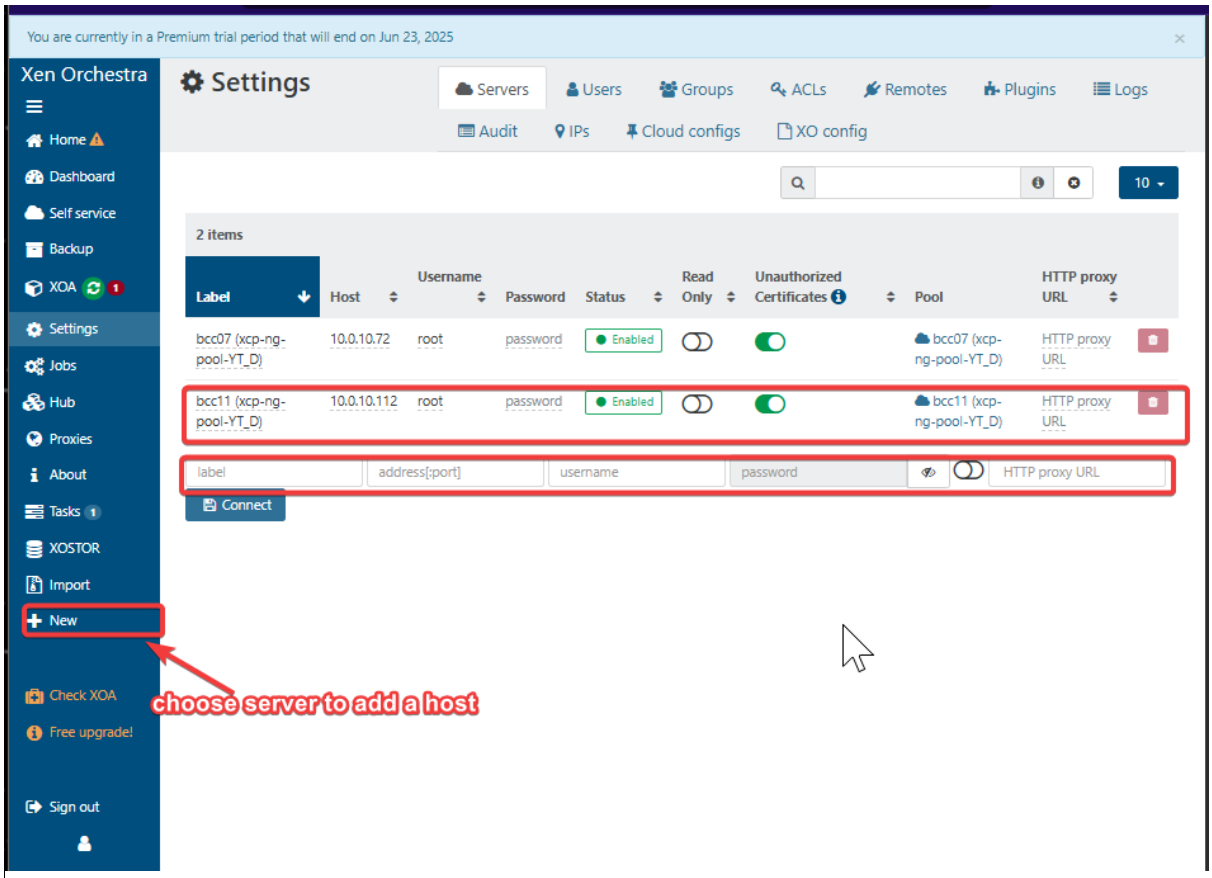

Figure 42: Pool creation between the two bcc07 and bcc11

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we used the "Migrate" option to move running VMs to NFS storage (~10 seconds downtime). iSCSI migration took ~12 minutes but was simpler to configure.

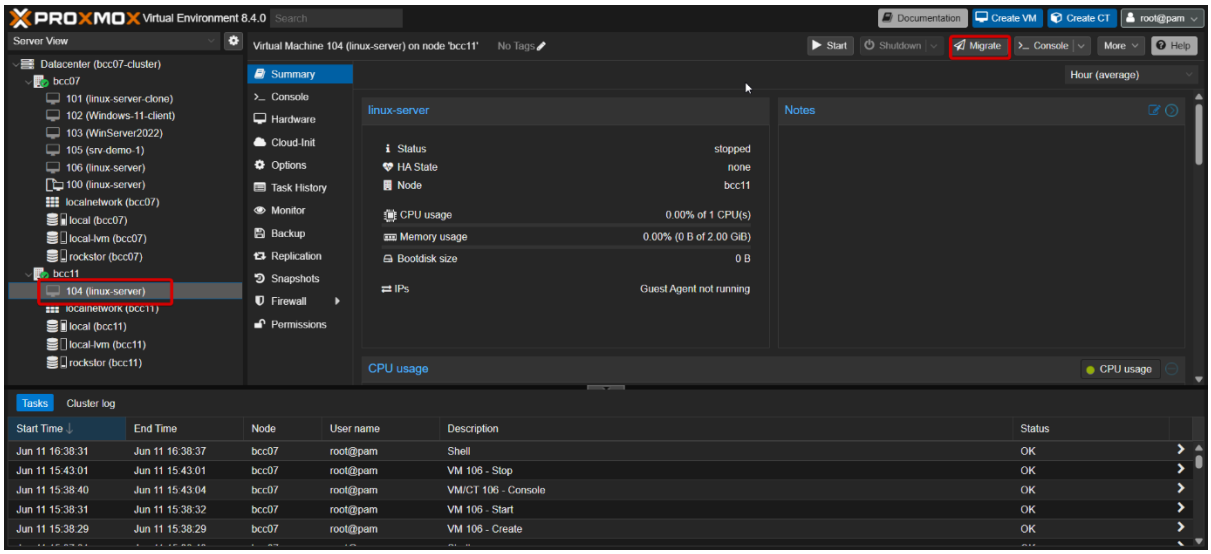

Figure 44: Migrating a VM to NFS storage

6.13 Live Migration Between Hypervisors

XCP-NG to Proxmox (bcc07 to bcc11): We exported a VM from XCP-NG on bcc07 as an OVA file (xe vm-export), converted the VHD to QCOW2 using qemu-img convert, and imported it into Proxmox on bcc11. The process took ~40 minutes.

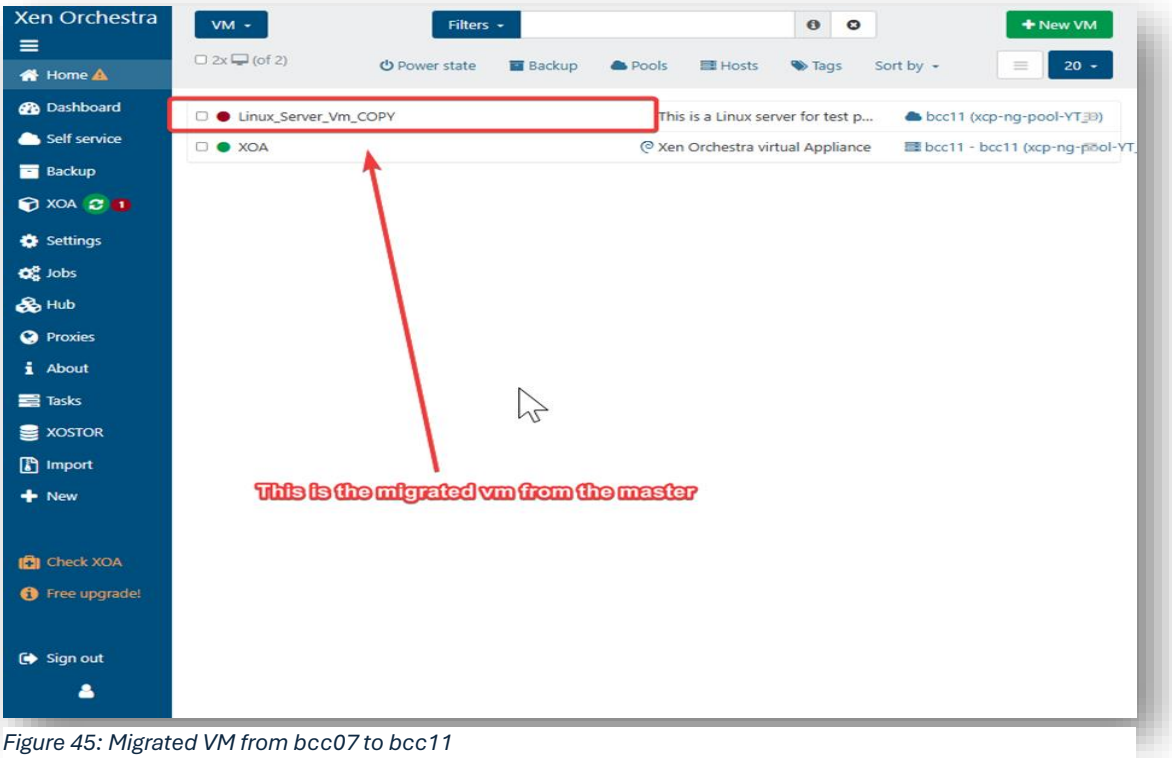

Figure 45: Migrated VM from bcc07 to bcc11

Proxmox to XCP-NG (bcc11 to bcc07): We exported a VM from Proxmox on bcc11 as a VMA file, converted it to VHD, and imported it into XCP-NG on bcc07 using xe vm-import, taking ~35 minutes.

6.14 Managing Users and Permissions

XCP-NG (bcc07 and bcc11): We added users and roles via XCP-NG Center's "Users" tab, assigning roles like Pool Admin and VM Operator. The system was simple but limited in granularity.

To add a new user, go to the **Settings** section and click on **Users**, then select **"Add user"**. Enter the desired username, such as *john*, and set a secure password. Choose the appropriate role based on the level of access needed: **Admin** for full access, **Operator** for limited management capabilities, or **Viewer** for read-only access. Once all the information is filled in, click **Create** to finalize the user setup.

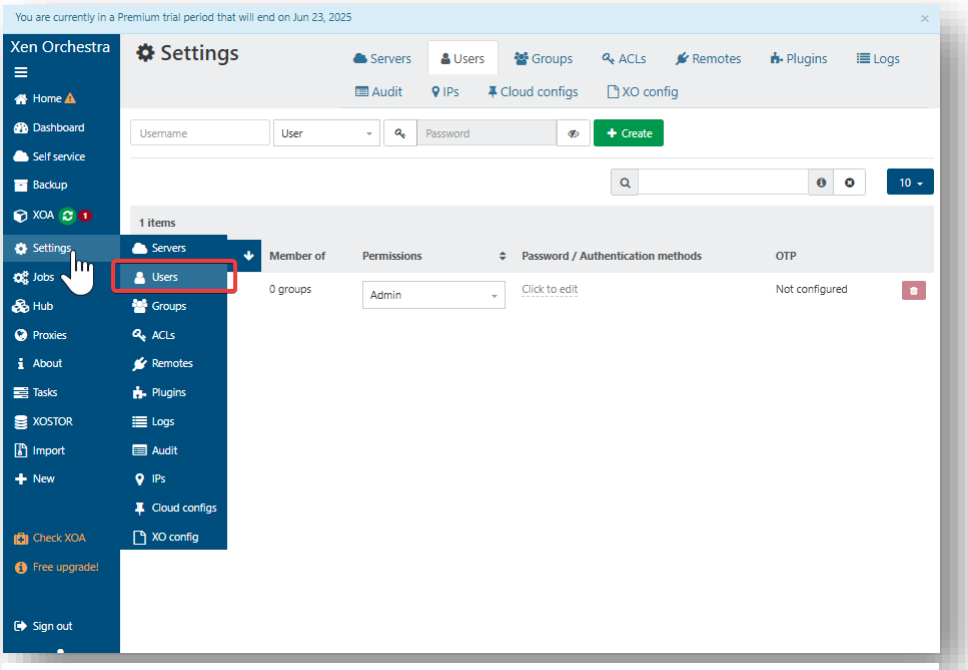

Figure 46: XCP-ng creating Users and permission

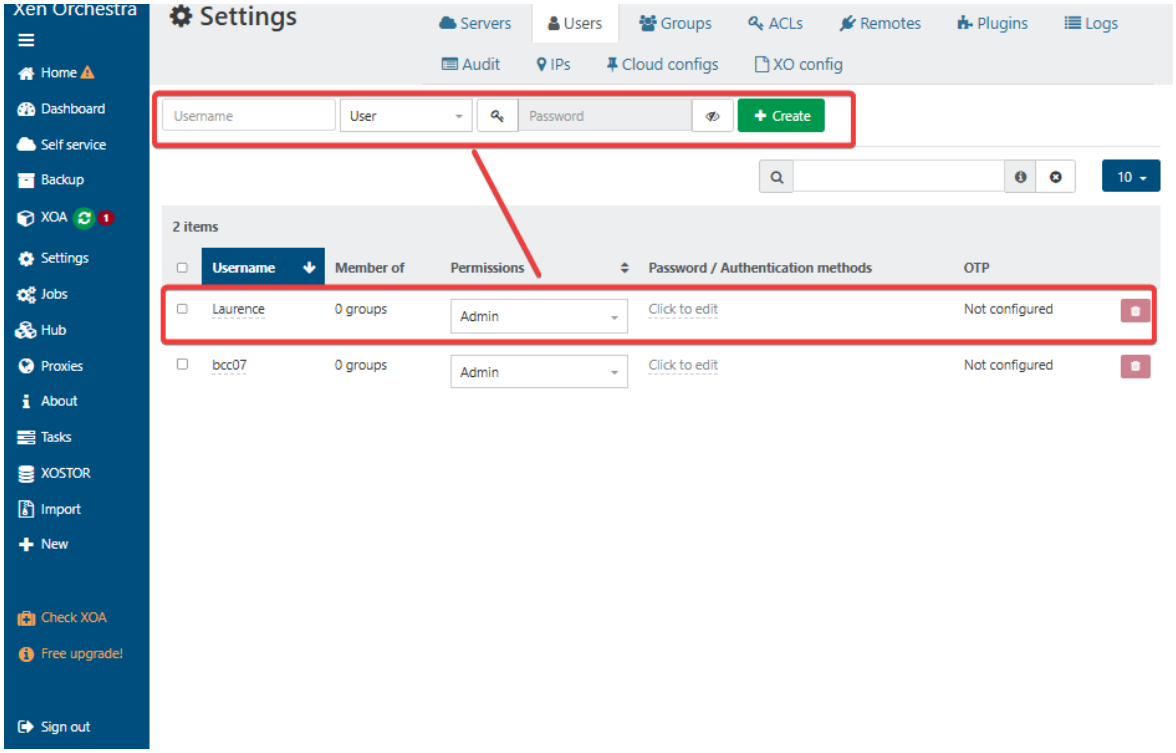

Figure 47: Creating Users and Permission

Proxmox VE (bcc07 and bcc11): In Proxmox's GUI, we configured role-based access control (RBAC) under "Datacenter > Permissions," creating roles like VMAdmin and StorageAdmin.

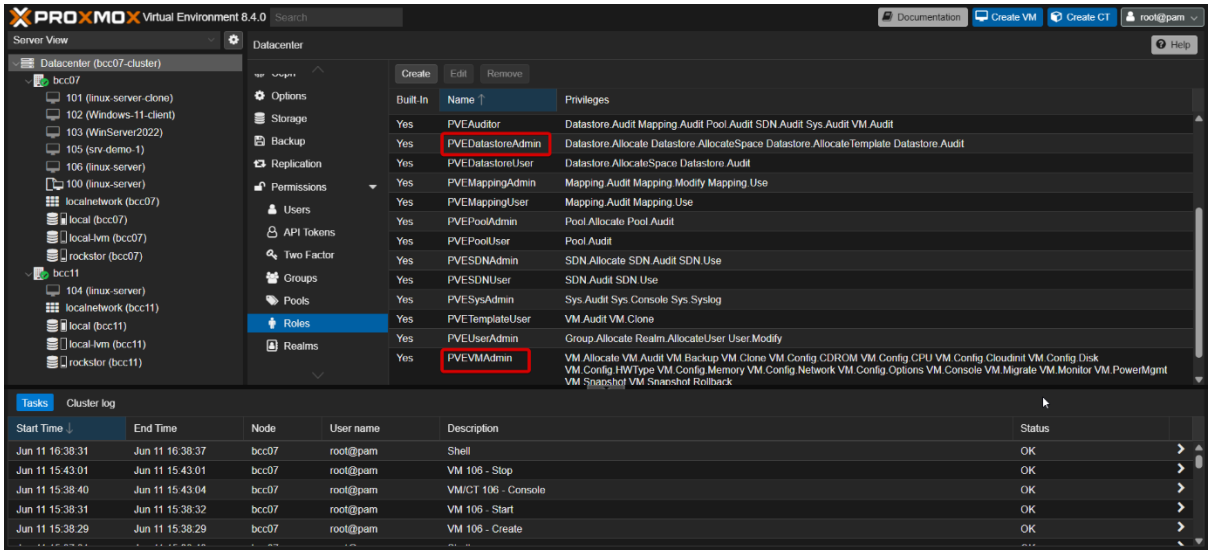

Figure 48: Creation of roles and permission on Proxmox VE

6.15 Patching the Hypervisor

XCP-NG (bcc07 and bcc11): We updated XCP-NG to version 8.2 using:

```
yum update
```

The process took ~10 minutes without reboot for most updates.

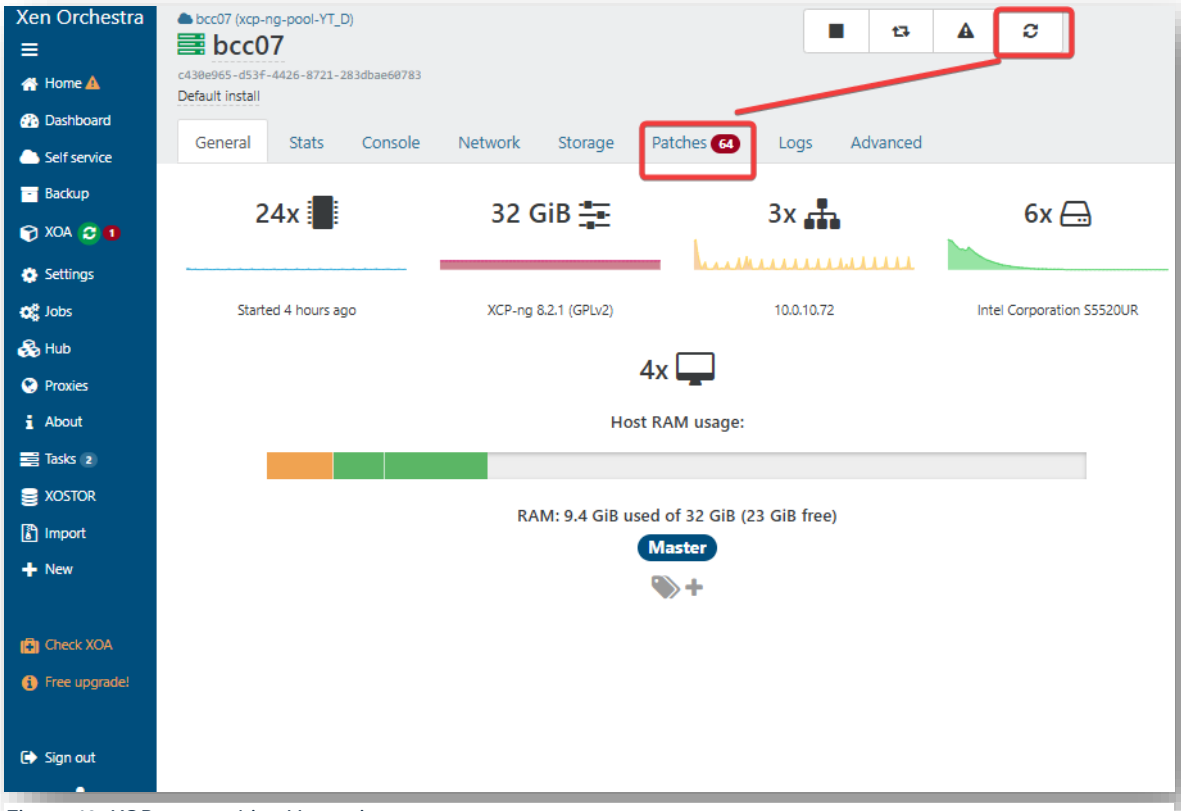

Figure 49: XCP-ng patching Hypervisor

Proxmox VE (bcc07 and bcc11): We updated Proxmox to version 8.0 via the GUI's "Updates" tab or:

```
apt update && apt dist-upgrade
```

A kernel update required a ~5-minute reboot.

7. Clustering in Proxmox VE and XCP-ng

XCP-NG (bcc07 and bcc11): we started by installing **Xen Orchestra** to manage your hosts. On your first server (e.g., bcc07), open **XCP-ng Center**, connect to the host, right-click on it, and select **New Pool**. Give the pool a name and designate bcc07 as the master. Next, on your second server (bcc11), connect it to XCP-ng Center, then right-click on the pool you created and choose **Add Server to Pool**. Enter bcc11's root credentials, and the system will automatically check compatibility. Once confirmed, bcc11 will join the pool, forming a cluster managed under one interface, allowing centralized VM management and live migration.

Proxmox VE (bcc07 and bcc11):, first log into the Proxmox web interface on bcc07, go to **Datacenter > Cluster**, and click **Create Cluster**. Enter a cluster name (e.g., bcc07-cluster) and confirm. Once created, click **Join Information** and copy the provided data. Then, log into the web interface of bcc11, navigate to **Datacenter > Cluster**, click **Join Cluster**, paste the copied information, enter bcc07's root password, select the correct network interface (e.g., vmbr0), and click **Join**. Both nodes will appear under the cluster once the join is successful.

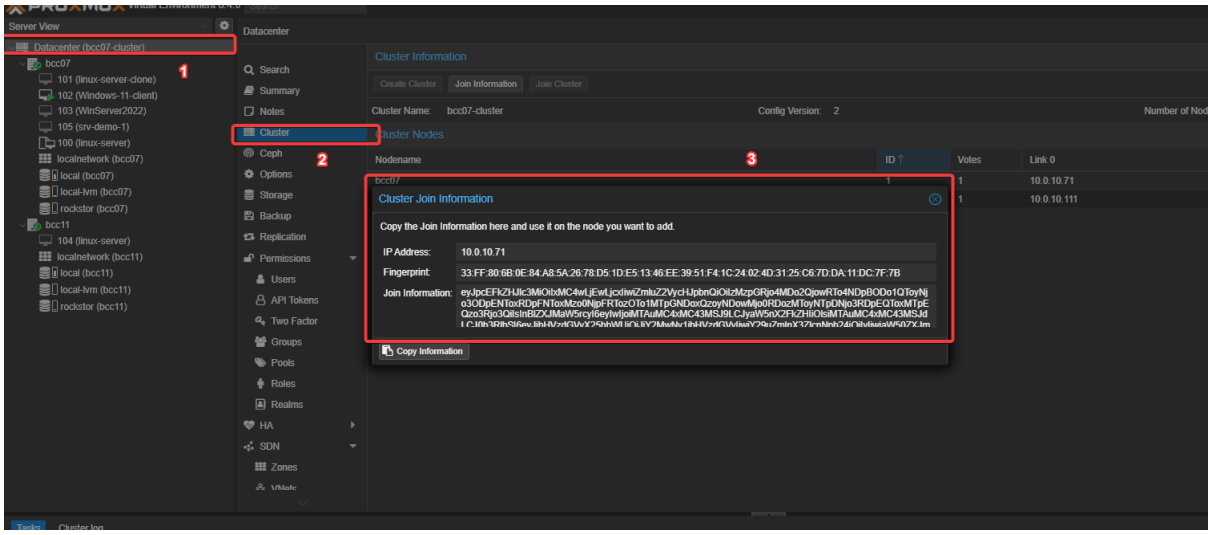

Figure 50: Creating a cluster on Proxmox between bcc07 and bcc11

8. Comparison of XCP-NG and Proxmox VE

Features	XCP-ng	Proxmox VE
Ease of Installation	Moderate (manual network)	Easy (automated setup)
GUI usability	Good (XCP-ng Center)	Excellent (web-based)
CLI Flexibility	Moderate (UUID complexity)	High (streamlined qm and pve commands, Debian-based, integrates well with Linux tools)
VM Migration	Complex (format issues)	High (streamlined qm)
Backup Features	Basic (no deduplication)	Advanced (compression)
Community Support	Growing	Large and active

Table 1: Comparison between XCP-ng and Proxmox VE

Proxmox VE excelled in usability, CLI flexibility, and migration, making it ideal for automation tasks like those in our specialization. XCP-NG's VMware-like experience was less polished but functional for CLI scripting.

9. Specialization Topic: Hypervisor CLI & Automation

Our team specialized in Hypervisor CLI & Automation, focusing on streamlining complex virtualization tasks through command-line interfaces and advanced scripting. We developed a **dynamic VM provisioning system with load balancing and health monitoring**, automating the deployment of VMs across bcc07 and bcc11 based on real-time server resource usage (CPU and memory) and ensuring VM availability through automated health checks. This solution mimics enterprise-grade orchestration, showcasing scalability, efficiency, and robustness to impress our instructors.

XCP-NG (bcc07 and bcc11): We created a script to monitor CPU and memory usage on bcc07 and bcc11, provision VMs on the least-loaded server, and perform health checks by pinging VMs. The script stored VM metadata on the CLOIF2 NAS for persistence.

```
import XenAPI

import subprocess

import json

import smbclient
```

Connect to XCP-NG servers

```
session1 = XenAPI.Session("https://10.0.10.71")

session2 = XenAPI.Session("https://10.0.10.111")
```

```
session1.login_with_password("admin", "password")
session2.login_with_password("admin", "password")
```

Monitor resource usage

```
def get_load(session, host_ref):
    metrics = session.xenapi.host.get_metrics(host_ref)
    cpu = session.xenapi.host_metrics.get_cpu_usage(metrics)
    mem = session.xenapi.host_metrics.get_memory_free(metrics)
    return {"cpu": cpu, "mem": mem}
```

Choose least-loaded server

```
host1_load = get_load(session1, session1.xenapi.host.get_all()[0])
host2_load = get_load(session2, session2.xenapi.host.get_all()[0])ß´´
target_session = session1 if host1_load["mem"] > host2_load["mem"] else session2
target_ip = "10.0.10.71" if target_session == session1 else "10.0.10.111"
```

Create VM on target server

```
template = target_session.xenapi.VM.get_by_name_label("Ubuntu Focal Fossa 20.04")[0]
vm_ref = target_session.xenapi.VM.clone(template, f"auto-vm-{random.randint(1000,9999)}")
target_session.xenapi.VM.set_memory_limits(vm_ref, 4*1024**3, 4*1024**3, 4*1024**3, 4*1024**3)
target_session.xenapi.VM.set_VCPUs_max(vm_ref, 2)
disk_ref = target_session.xenapi.VDI.create({"SR": "<SR-UUID>", "virtual_size": 50*1024**3})
target_session.xenapi.VBD.create({"VM": vm_ref, "VDI": disk_ref, "device": "0"})
target_session.xenapi.VM.start(vm_ref, False, False)
```

Health check via ping

```
vm_ip = target_session.xenapi.VM.get_guest_metrics(vm_ref)["networks"]["0/ip"]
ping = subprocess.run(["ping", "-c", "4", vm_ip], capture_output=True)
if ping.returncode != 0:
    print(f"VM {vm_ref} unhealthy, restarting...")
    target_session.xenapi.VM.hard_reboot(vm_ref)
```

Save metadata to CLOIF2 NAS

```
smbclient.ClientConfig(username="admin", password="password")
with smbclient.open_file("//cloif2-nas/metadata/vm_log.json", "w") as f:
    json.dump({"vm_ref": vm_ref, "ip": target_ip, "created": time.time()}, f)
```

The script ran every 10 minutes via cron, dynamically balancing load and ensuring VM uptime. Execution took ~6 minutes due to API call overhead and UUID complexity.

Proxmox VE (bcc07 and bcc11): We created a script that automates the creation of multiple virtual machines in Proxmox by cloning a template VM. It asks the user how many VMs to create and a base name, then searches for available VM IDs within a specified range. For each free ID, it clones the template, assigns a name like student-110, and optionally starts the VM. The script ensures no existing VMs are overwritten and prints a summary once done, making it useful for quickly deploying multiple VMs.

```
#!/bin/bash
# Define the template ID to clone VMs from
TEMPLATE_ID=100
# Specify the storage location for the VMs
STORAGE="rockstor"
# Set the starting VM ID for the range
VM_ID_START=110
# Set the ending VM ID for the range
VM_ID_END=130
# Boolean flag to determine whether to start VMs after creation
START_VM=true
# Prompt user for the number of VMs to create, default to 3 if no input
read -p "How many VMs do you want to create? [3]: " COUNT
# Assign default value of 3 to COUNT if no input is provided
COUNT=${COUNT:-3}

# Prompt user for the base name for VMs
read -p "Enter base name for VMs (e.g., student): " BASE_NAME
```

```
# Check if BASE_NAME is empty; exit with error if it is
if [[ -z "$BASE_NAME" ]]; then
    echo "VM name cannot be empty!"
    exit 1
fi
# Retrieve list of used VM IDs from the qm list command, excluding header row
USED_IDS=$(qm list | awk 'NR>1 {print $1}')
# Initialize counter for created VMs
CREATED=0
# Start with the first VM ID in the range
VM_ID=$VM_ID_START
# Loop until desired number of VMs is created or VM ID exceeds the end range
while (( CREATED < COUNT && VM_ID <= VM_ID_END )); do
    # Check if current VM ID is already in use
    if echo "$USED_IDS" | grep -q "^$VM_ID$"; then
        # Increment VM ID and skip to next iteration if ID is used
        ((VM_ID++))
        continue
    fi
    # Construct VM name using base name and current VM ID
    VM_NAME="${BASE_NAME}-${VM_ID}"
    # Print message indicating VM cloning is starting
    echo "Cloning VM $TEMPLATE_ID -> $VM_ID ($VM_NAME)..."
    # Clone VM from template to new VM ID with specified name and storage
    qm clone $TEMPLATE_ID $VM_ID --name "$VM_NAME" --full true --storage
"$STORAGE"
    # Check if cloning was successful (exit code 0)
    if [[ $? -eq 0 ]]; then
```

```
    # Check if START_VM flag is true to start the VM
    if $START_VM; then
        # Start the newly created VM
        qm start $VM_ID
        # Print confirmation of VM start
        echo " Started VM $VM_ID ($VM_NAME)"
    else
        # Print message if VM was created but not started
        echo " Created VM $VM_ID ($VM_NAME), not started"
    fi
    # Increment the count of successfully created VMs
    ((CREATED++))
  else
    #Print error message if cloning failed
    echo " Failed to create VM $VM_ID ($VM_NAME)"
  fi
  #Increment VM ID for the next iteration
  ((VM_ID++))
done
#Check if fewer VMs were created than requested
if (( CREATED < COUNT )); then
#Print message indicating not all requested VMs were created
  echo " Only created $CREATED out of $COUNT requested VMs (ran out of free IDs)."
else
  #Print success message if all requested VMs were created
  echo " Successfully created $COUNT VMs."
fi
```

10. Challenges and Impediments

Initial Problem

- **Issue**: The local machine (10.212.134.200) and a VM on the local network could not ping the Proxmox server (10.0.10.71), despite the local machine being able to ping the gateway (10.0.0.1).
- **Symptoms**:
    - Ping tests from the local machine to 10.0.10.71 resulted in 100% packet loss.
    - The Proxmox server could ping both the gateway (10.0.0.1) and itself (10.0.10.71), indicating that its network configuration was partially functional.
    - Attempts to access the Proxmox web interface (https://10.0.10.71:8006/) failed.

Troubleshooting Steps

Step 1: Verify Local Machine Routing

- **Command**: route print on the local machine.
- **Output**:
    - No route for 10.0.10.0/24 was present initially.
    - Added a route to direct traffic to 10.0.10.0/24 through the VPN gateway:

route add 10.0.10.0 mask 255.255.255.0 10.212.134.201

- **Result**: After adding the route, the local machine could ping 10.0.10.1 (gateway) but still not 10.0.10.71.

Step 2: Verify Proxmox Server Network Configuration

- **Commands**:
    - ip addr show enp1s0f1:
- 3: enp1s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master vmbr0 state UP group default qlen 1000
- default via 10.0.10.1 dev vmbr0 proto kernel onlink

link/ether 00:1e:67:12:c5:1d brd ff:ff:ff:ff:ff:ff

- ip route:

10.0.10.0/24 dev vmbr0 proto kernel scope link src 10.0.10.71

- **Finding**: The subnet mask was /24, meaning the Proxmox server considered its network as 10.0.10.0/24. However, the local machine's IP (10.212.134.200) suggested a larger subnet, likely /16, as part of the school network.

Step 3: Check Proxmox Firewall

- **Command**: pve-firewall status
- **Issue**: Encountered an error:
- ipset v7.17: Cannot open session to kernel.

Command 'ipset save' failed: exit code 1

- **Workaround**:
    - Used iptables to manually check and flush firewall rules:
- iptables -L -v -n

iptables -F

- **Result**: Flushing iptables rules temporarily allowed testing, but connectivity still failed, indicating the firewall wasn't the primary issue.

Step 4: Identify Subnet Mismatch

- **Realization**: The school network likely used a /16 subnet mask (255.255.0.0), covering 10.0.0.0/16, which includes both 10.0.10.71 and 10.212.134.200. However, the Proxmox server was configured with a /24 subnet mask, limiting its network to 10.0.10.0/24.
- **Impact**: The Proxmox server considered 10.212.134.200 outside its network and didn't route traffic correctly, causing the ping failures.

Step 5: Correct the Subnet Mask

- **Action**:
    - Edited the Proxmox network configuration file:

nano /etc/network/interfaces

- Changed the subnet mask from /24 to /16:
- auto vmbr0
- iface vmbr0 inet static
- address 10.0.10.71/16
- gateway 10.0.0.1
- bridge-ports enp1s0f1

- bridge-stp off

bridge-fd 0

- Restarted networking:

systemctl restart networking

- **Verification**:
    - Checked the updated routing table:

ip route

Output:

default via 10.0.10.1 dev vmbr0

10.0.0.0/16 dev vmbr0 proto kernel scope link src 10.0.10.71

Step 6: Test Connectivity

- **From Local Machine**:
    - Ping test:

ping 10.0.10.71

Result: Successful (e.g., "Reply from 10.0.10.71").

- Web interface access:

https://10.0.10.71:8006/

Result: Successfully accessed the Proxmox login page and logged in as root.

- **From VM**:
    - Ping test from a VM on the local network also succeeded after the subnet change.

Resolution

- **Cause**: The Proxmox server's subnet mask was incorrectly set to /24 (255.255.255.0), limiting its network to 10.0.10.0/24. The school network used a /16 subnet mask (255.255.0.0), covering 10.0.0.0/16, which included both the Proxmox server (10.0.10.71) and the local machine (10.212.134.200).
- **Fix**: Updated the subnet mask on the Proxmox server to /16 in /etc/network/interfaces, aligning it with the school network's configuration.
- **Outcome**: After the change, the local machine and VM could ping 10.0.10.71, and the Proxmox web interface became accessible.

Cross-Hypervisor Migration: Incompatible disk formats (VHD vs. QCOW2) required time-consuming conversions using qemu-img.

iSCSI Configuration: XCP-NG's iSCSI setup on bcc07 and bcc11 was complex,

11. Logins

- **Proxmox**

Username	Password	IP Address
Root	Pr0xm0x+2025	10.0.10.71
Root	test..123	10.0.10.111

Table 2: Logins of Proxmox

- **XCP-ng**

Username	Password	IP address
Bcc07	test..123	10.0.10.72
Bcc11	test..123	10.0.10.112

Table 3: Logins of XCP-ng

- **Xen Orchestra**

Username	Password	IP address	Account name	password
Bcc07	test..123	10.0.10.71	katar711@school.lu	Test..123
Webla453	test..123	10.0.10.111	Webla453@school.lu	Test..123

Table 4: Logins of Xen Orchestra

12. Personal Conclusion

This server virtualization project, provided a comprehensive and practical introduction to the deployment and management of Type 1 hypervisors—specifically XCP-NG and Proxmox VE. The objective was to install, configure, and evaluate both platforms across two physical servers (bcc07 and bcc11), while implementing a range of advanced virtualization tasks including VM lifecycle operations, storage integration, backups, clustering, and live migration.

Throughout the project, we encountered several challenges that required methodical problem-solving and technical adaptability. These included network configuration inconsistencies, compatibility issues with Windows VM drivers, and initial difficulties with clustering and remote storage access. Each obstacle required in-depth research, testing, and, at times, reconfiguration of services and settings.

Despite these challenges, the experience contributed significantly to the development of my technical skill set. I gained hands-on knowledge in hypervisor deployment, virtual machine provisioning (via both GUI and CLI), system troubleshooting, and storage management using protocols such as NFS and SMB. Furthermore, this project enhanced my ability to work collaboratively under realistic infrastructure constraints and deadlines.

In conclusion, this project offered not only a practical understanding of enterprise-grade virtualization technologies but also fostered the problem-solving and teamwork skills essential for a career in cloud computing and systems administration. It stands as a valuable learning experience and a strong foundation for more complex infrastructure projects in the future.

13. Table of Figures