OpenNebula: Installation manual for IONOS dedicated (baremetal) servers

1. Introduction

This document describes the automatic installation process, according to the configuration OneDeploy, of the OpenNebula virtualization platform on IONOS' dedicated (baremetal) server infrastructure. By following this guide, you will be able to:

- Configure the Ansible control node (installing Ansible if necessary).
- Download the Ansible playbooks (configuration files).
- Modify the playbooks according to your specific needs to, for example, configure the OpenNebula version, define the virtual network for the hosts, storage options, etc.
- Run the configuration files.
- Verify that the installation has been carried out correctly.

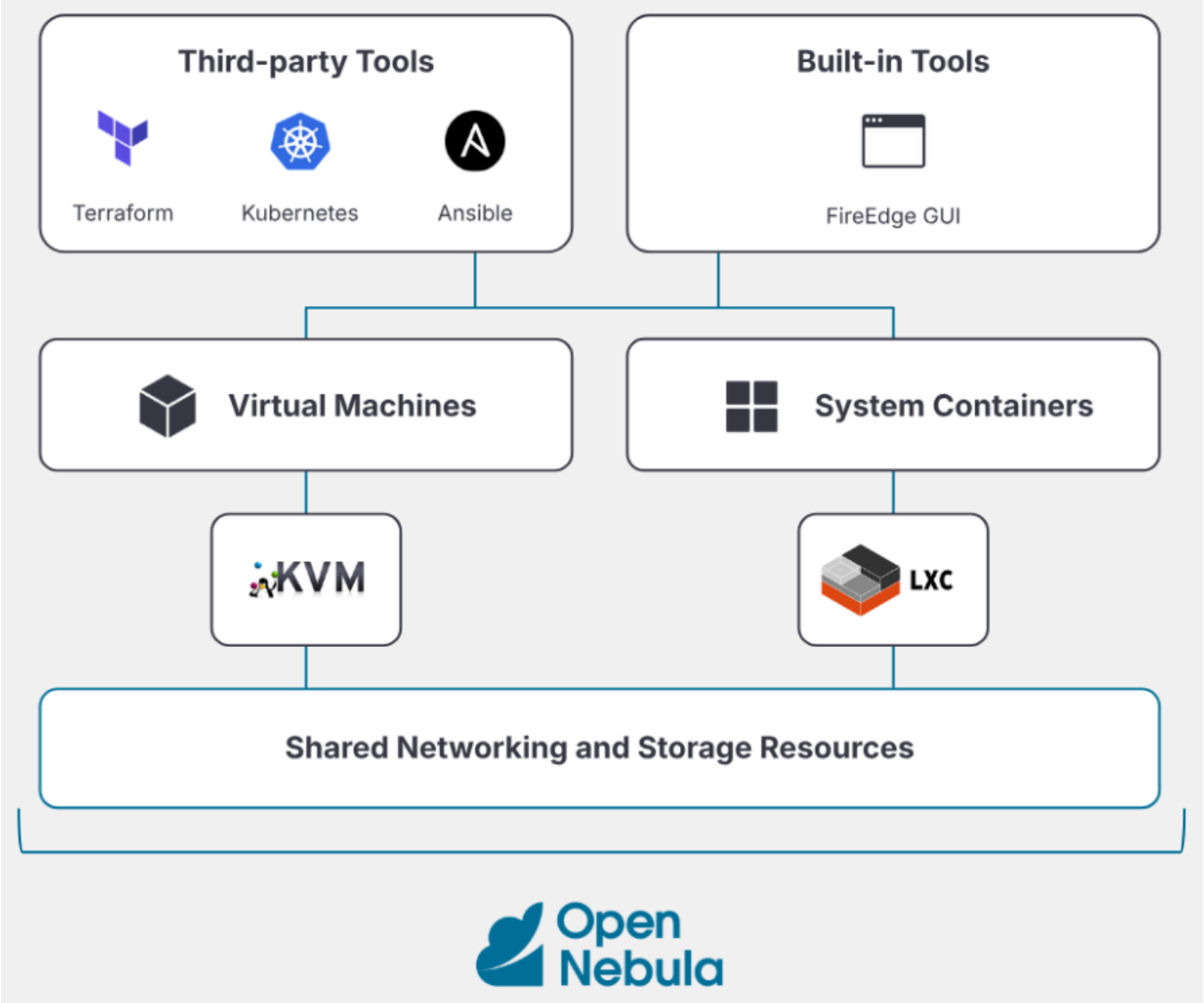
Table of Contents

2. The OpenNebula platform

OpenNebula is a powerful open-source platform that enables the agile and simple creation and management of enterprise cloud environments and virtualized data centers. By combining existing virtualization technologies with advanced multi-tenant functionalities, automated provisioning, and flexibility across different cloud environments, it unifies the management of technology infrastructure and applications within a single framework, eliminating vendor lock-in and reducing complexity, resource consumption, and operating costs.

An infrastructure orchestrated by OpenNebula can be deployed in different environments, whether cloud, edge, on-premises, hybrid, or multi-cloud. Virtualization is based on the *hypervisor* KVM (*Kernel-based Virtual Machine*)with support for LXC (*Linux Containers*), while the overall management of the different components that make up the infrastructure is carried out through one or more *front-ends* or control panels.

Before continuing with the installation process, you can access the detailed documentation that provides OpenNebula to learn and delve into the basic concepts that define its architecture, learn about configuration and deployment options, and familiarize yourself with the general operation of the platform.

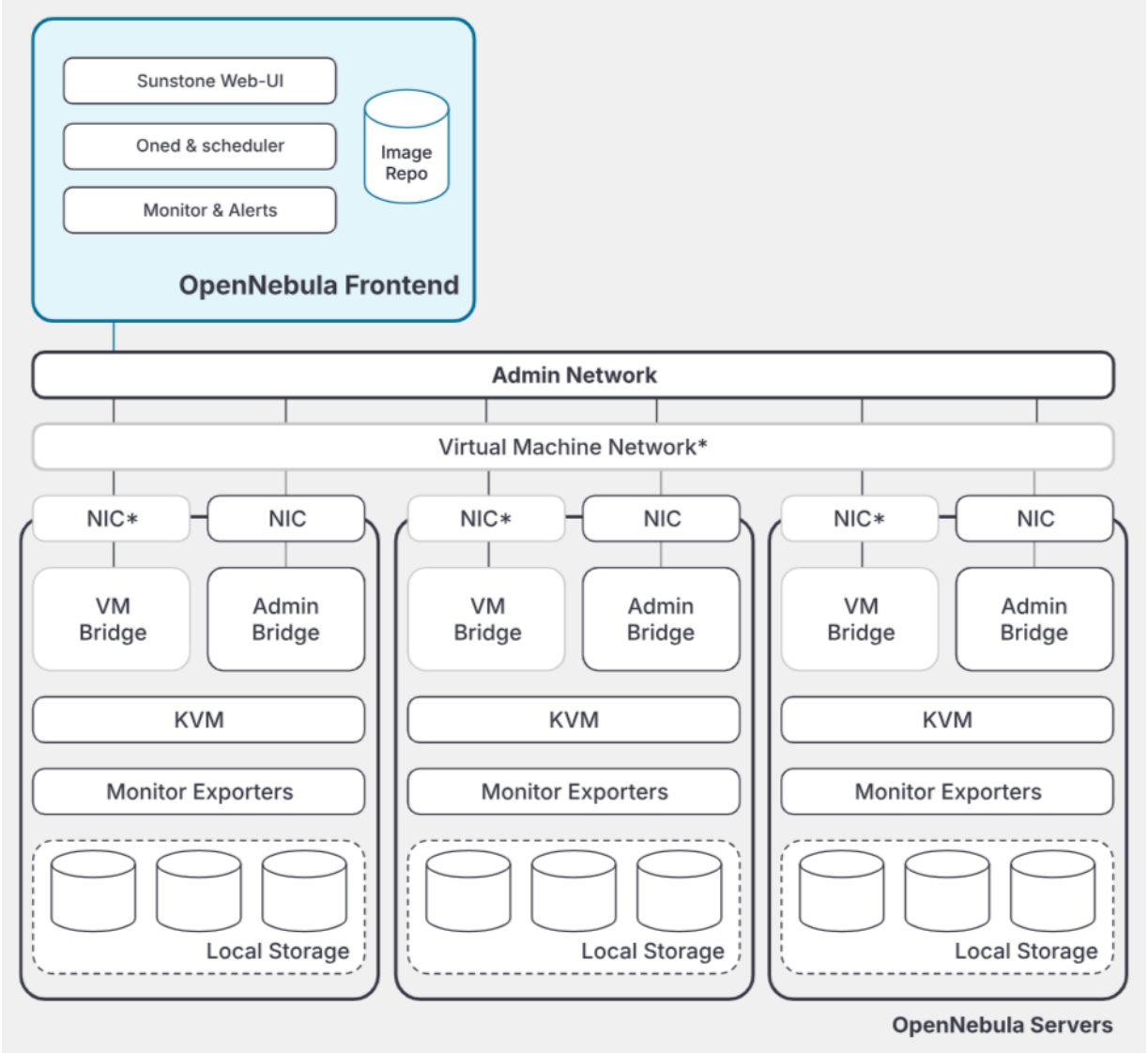


Platform overview. Source:Key concepts of OpenNebula

3. Deployment Architecture

OneDeploy uses a set of playbooks Ansible to install a full instance of OpenNebula in a production environment. These playbooks contain the deployment configuration for two possible reference architectures, depending on whether local or shared storage is used to access the image data stores.

This manual uses a local storage implementation. In this configuration, virtual disk images are transferred from the front end to the hypervisors' local storage using the SSH protocol. This requires three dedicated servers: one for the front end and two for the hosts where client virtual machines can be created.



Example architecture for a deployment with local storage. Source:OpenNebula.

This example architecture uses a basic network configuration: a flat (bridged) network, where the IP address of each virtual machine is part of the same network as the hypervisors.

4. Initial requirements

The hosts (i.e., the servers that will host the OpenNebula front-end and the two hypervisors) must meet the following requirements:

- Operating System: Ubuntu 24.04 with Netplan >=0.105, Rocky 10 or Alma Linux 10.
- Passwordless SSH login, as root, from the front-end node to the hypervisor nodes.
- The user performing the installation must be able to access the root account using sudo.
- Having several free IPs available on the same network that connects the front-end and the hypervisors.

5. Creation and configuration of the servers

From the IONOS advanced cloud panel, create three dedicated servers (Fig.1).

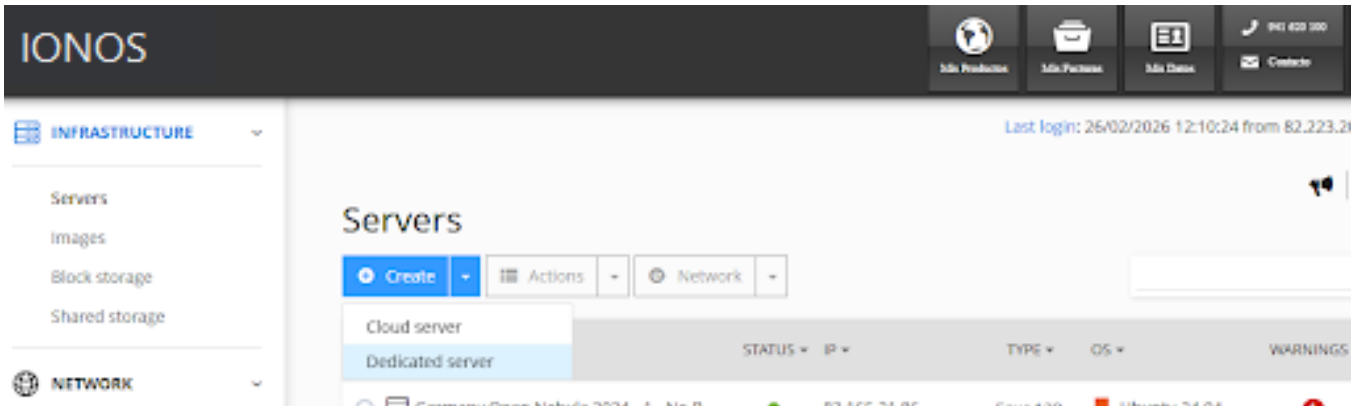


Fig 1: Creating servers from the IONOS cloud panel.

Select the following configuration/minimum recommended:

Parameter	Value
CPU	Modern (≥2016 virtualization enabled)
RAM	>32 GB
Disc	512GB NVMe
Image	Alma Linux 10 Ubuntu 24 Rocky 10
Data Center and Availability Zone	It will depend on the type of server you contract.
Backup	
Advanced options	-

Once the servers aresuppliedIt is necessary to create a VLAN network (Fig. 2) through which the servers will communicate internally and will also serve to allow the assignment of IPs to the virtual machines created in the hypervisors.

The **VLAN** (*Virtual Bridged Local Area Networks*Virtual networks (VTNs) are networks that allow you to divide an existing physical network into several logical networks. They offer significant improvements in network security, privacy, and protection of sensitive data, while also making it easier to implement changes to the infrastructure.

Complete the required fields:

Parameter	Value
Subnet	172.20.0.0
Mask	255.255.255.0
Gateway	172.20.0.1
Datacenter	Spain

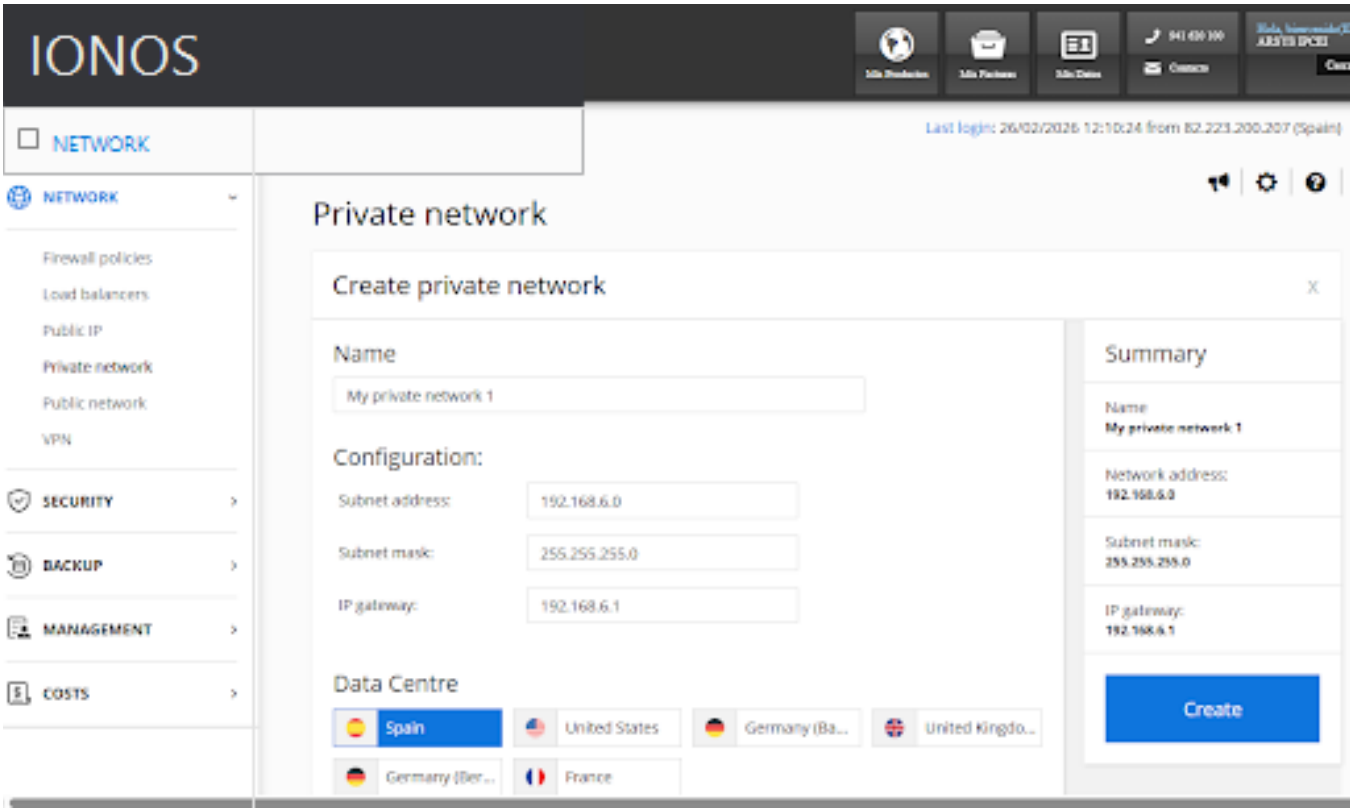


Fig 2: Creating private networks from the IONOS cloud panel.

Once it appears in the list of available networks in your cloud panel, click to select it and in the “Assigned Servers” section, select the three servers you created in the first step of the process (Fig.3).

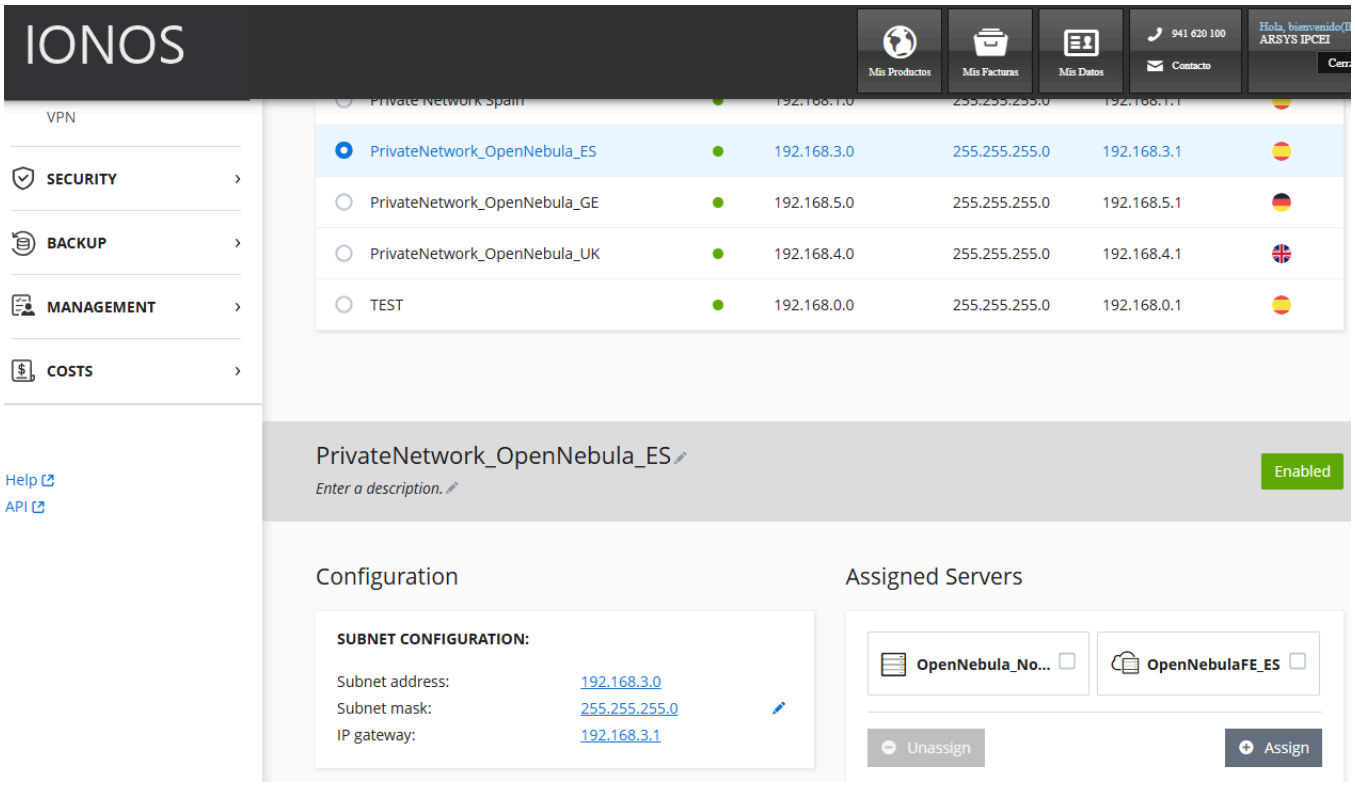


Fig 3: Assignment of servers to a VLAN network.

Create a new firewall policy with the following values (Fig.4).

Allowed IP	PROTOCOL	Port(s)
All	TCP	22
All	TCP	2616

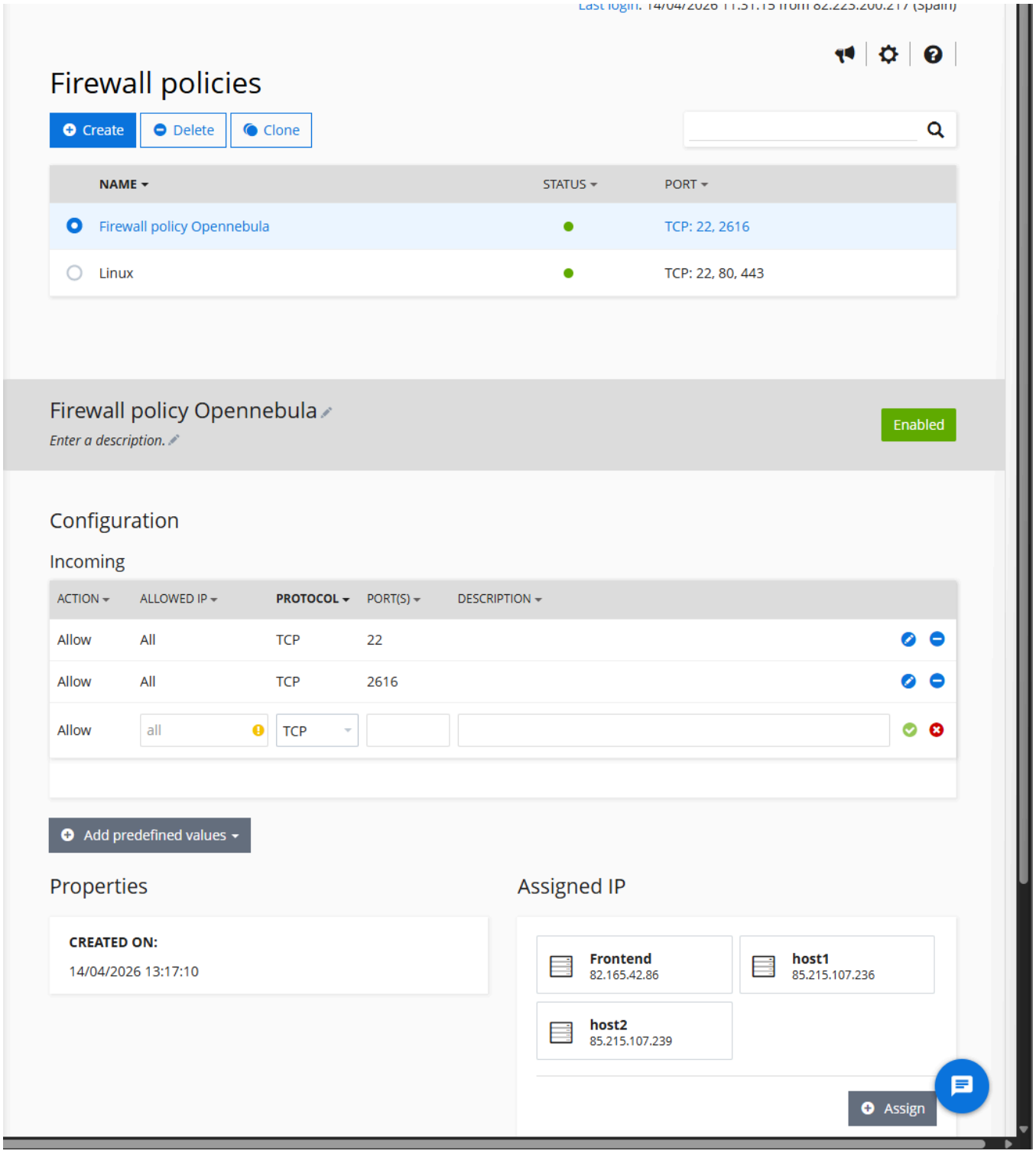


Fig 4: Creating a new Firewall policy.

Once we have the servers, the private network, and the firewall policy, we must configure the internal network within the servers:

The following commands must be executed on each server to configure the internal network. You will need to modify them according to the IP addresses of the three dedicated servers and the VLAN parameters created in the previous step. Server creation and configuration

In it frontend:

```
nmcli connection add type vlan con-name vlan2808 ifname eth0.2808 dev eth0
id2808ip4172.20.0.2/24 ipv4.gateway172.20.0.1

nmcli connection up vlan2808

bash -c 'echo -e "172.20.0.2f1\n172.20.0.3n1\n172.20.0.4 n2" >> /etc/host
```

On host1:

```
nmcli con add type bridge con-name br0 ifname br0 ipv4.method manual
ipv4.addresses 172.20.0.3/24 bridge.stp no

nmcli con add type vlan con-name vlan2808 ifname eth0.2808dev eth0 id 2808

nmcli con modify vlan2808 master br0

nmcli con modify vlan2808 slave-type bridge

nmcli con up br0

nmcli with upvlan2808

bash -c 'echo -e "172.20.0.2f1\n172.20.0.3n1\n172.20.0.4 n2" >> /etc/host
```

And finally, on host 2:

```
nmcli con add type bridge con-name br0 ifname br0 ipv4.method manual
ipv4.addresses 172.20.0.4/24 bridge.stp no

nmcli con add type vlan con-name vlan2808 ifname eth0.2808dev eth0 id 2808

nmcli con modify vlan2808 master br0

nmcli con modify vlan2808 slave-type bridge

nmcli con up br0

nmcli with upvlan2808

bash -c 'echo -e "172.20.0.2f1\n172.20.0.3n1\n172.20.0.4 n2" >> /etc/host
```

6.Platform Installation

6.1. Environment configuration

First, from the server we're going to use as the front-end, we need to install two packages for Python:

- Pip: tool for installing and managing packages written in Python.
- Hatch: allows you to manage Python projects, centralizing the entire development lifecycle in a single command-line tool.

```
sudo yum install python3-pip pipx
```

When the package manager installation is complete, clone the repository that contains the files for deployment. OneDeploy from OpenNebula:

```
git clone https://github.com/OpenNebula/one-deploy.git
```

Next, install the project manager *Hatch*:

```
pipx install hatch
pipx ensurepath
source ~/.bashrc
```

Access the directory *OneDeploy*, which contains the cloned files from the repository:

```
cd one-deploy
```

And proceed to install the necessary components for deployment:

```
export HATCH_PYTHON=$(which python3)
 git config --global http.sslVersion tlsv1.2
make requirements
```

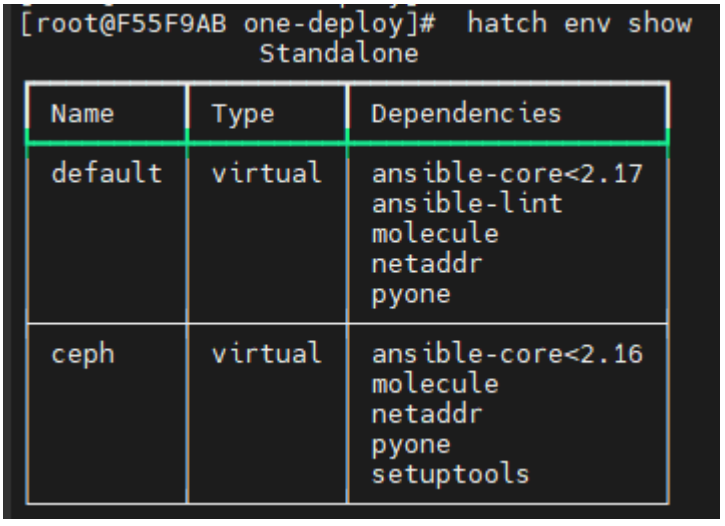
Hatch will automatically create two virtual environments and install the necessary components

```
hatch env create default
hatch env run -e default -- ansible-galaxy collection install --requirements-file
/root/one-deploy/requirements.yml
```

To list the available environments, run the following command:

```
hatch env show
```

Hatch will show a summary of the two newly created environments (Fig. 4), the default environment (*default*) and the environment *ceph* (which isolates the ceph-ansible dependencies in a different virtual environment).



```
[root@F55F9AB one-deploy]#  hatch env show
                Standalone
```

Name	Type	Dependencies
default	virtual	ansible-core<2.17 ansible-lint molecule netaddr pyone
ceph	virtual	ansible-core<2.16 molecule netaddr pyone setuptools

Fig. 4. List of created virtual environments and associated dependencies.

To enter the default environment (*default*), execute the following command:

```
hatch shell
```

After doing so, you will see that the string (one-deploy) has been included before the prompt (highlighted with a blue background):

```
front-end:~/one-deploy$ hatch shell
source
"/home/frontend/.local/share/hatch/env/virtual/one-deploy/RdxhOVxs/one-deploy/bin/activate"
(one-deploy) :~/one-deploy$
```

6.2. Configuring cloud parameters

Create and access a new directory, which for this installation example we will call *my-one*:

```
mkdir my-one
cd my-one
```

Within this directory, we will create and edit two new files:

```
touch example.yml ansible.cfg
```

- example.yml: contains the parameter definitions necessary to install OpenNebula.
- ansible.cfg, which is our Ansible configuration file.

The following shows the content to be included in the first file *example.yml.* You will need to modify it according to the IPs of the three dedicated servers and VLAN parameters created in the previous step. Server creation and configuration.

```
---
all:
  whose:
    ansible_user: root
    one_version: '7.0'
    one_pass:opennebulapass
    ensure_hosts: true
    ensure_hostname: true
    vn:
admin_net:
      managed: true
      template:
        VN_MAD: bridge
        PHYDEV: eth0.2812
        BRIDGE: br0
        VLAN_ID: 2812
        WITH:
          TYPE: IP4
          IP: 192.168.0.100
          SIZE: 48
        NETWORK_ADDRESS: 192.168.0.0
        NETWORK_MASK: 255.255.255.0
        GATEWAY: 192.168.0.1
        DNS: 1.1.1.1
```

```
frontend:
  hosts:
    f1: { ansible_host: 192.168.0.2 }

node:
  hosts:
    n1: { ansible_host: 192.168.0.3 }
    n2: { ansible_host: 192.168.0.4 }
```

The following table shows a list of the file parameters *example.yml* paying special attention to those you should update according to your preferences and the configuration of the servers and VLANs created in the point 5 of this guide.

Parameter	Description
one_version	OpenNebula version to install.
one_pass	Password for the user *oneadmin*.
ensure_hosts	Boolean to populate /etc/hosts on all nodes using the ansible_host IP and inventory hostnames (f1, n1 and n2).
vn	Parameters for the OpenNebula virtual network (admin_net) that will be created for the virtual machines.
PHYDEV	The physical interface on the servers that will connect to the virtual network.
WITH	Address range (first IP and size) available to assign to virtual machines.
GATEWAY	Default gateway for the network.
DNS	Network DNS server.
ansible_host	IP address for the front-end (f1) and the hypervisors (n1 and n2).

Regarding the configuration file *ansible.cfg*:

```
[defaults]
inventory=./example.yml
gathering=explicit
host_key_checking=false
display_skipped_hosts=true
retry_files_enabled=false
any_errors_fatal=true
stdout_callback = ansible.builtin.default
result_format = yaml
timeout=30
collections_paths=/root/one-deploy/ansible_collections

[ssh_connection]
pipelining=true
ssh_args=-q -o ControlMaster=auto -o ControlPersist=60s

[privilege_escalation]
become      = true
become_user = root
```

In this file, update the content of *collections_paths* according to the correct path to the directory *one-deploy* created in section 6.1 of the guide, environment configuration.

6.3. Verifying connectivity between nodes

After configuring the parameters above, it's advisable to verify that connectivity between nodes is working correctly. Run the following command:

```
ansible -i example.yml all -m ping -b
```

The output, if everything is configured correctly, should look similar to this:

```
f1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
n2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
n1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

If there is a problem accessing any of the hosts, or if access via SSH is not configured correctly, the command output will be similar to this:

```
[WARNING]: Unhandled error in Python interpreter discovery for host n2: Failed to connect to the host via ssh:
[WARNING]: Unhandled error in Python interpreter discovery for host n1: Failed to connect to the host via ssh:
n2 | UNREACHABLE! => {
    "changed": false,
    "msg": "Data could not be sent to remote host \"192.168.0.4\". Make sure this host can be reached over ssh: ",
    "unreachable": true
}
n1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Data could not be sent to remote host \"192.168.0.3\". Make sure this host can be reached over ssh: ",
```

```
    "unreachable": true
}
```

6.4. Execution of configuration files

The next step is to run the Ansible playbooks. Before you begin, make sure you are in the correct environment. *Hatch* is verifying that the command prompt begins with the string (one-deploy), as explained at the end of section 6.1, environment configuration.
To run the playbooks, enter the directory *my-one* (that you created in section 6.2. Configuring cloud parameters) and run this command:

```
ansible-playbook -vvv opennebula.deploy.main
```

The Ansible playbooks should run and complete the platform installation. Please note that this process may take a few minutes. An example of the output information might look similar to the following:

```
(one-deploy) front-end:~/my-one$ ansible-playbook -v opennebula.deploy.main

Using /home/basedeployer/my-one/ansible.cfg as config file
running playbook inside collection opennebula.deploy
[WARNING]: Could not match supplied host pattern, ignoring: bastion

PLAY [bastion]
*********************************************************************************
**********
skipping: no hosts matched
[WARNING]: Could not match supplied host pattern, ignoring: grafana
[WARNING]: Could not match supplied host pattern, ignoring: mons
[WARNING]: Could not match supplied host pattern, ignoring: mgrs
[WARNING]: Could not match supplied host pattern, ignoring: osds

PLAY [frontend,node,grafana,mons,mgrs,osds]
************************************************************

TASK [opennebula.deploy.helper/python3 : Bootstrap python3 intepreter]
***********************************
skipping: [f1] => changed=false
  attempts: 1
  msg: /usr/bin/python3 exists, matching creates option
```

```
skipping: [n2] => changed=false
  attempts: 1
  msg: /usr/bin/python3 exists, matching creates option
skipping: [n1] => changed=false
  attempts: 1
  msg: /usr/bin/python3 exists, matching creates option

...

TASK [opennebula.deploy.prometheus/server : Enable / Start / Restart Alertmanager service
(NOW)] *********
skipping: [f1] => changed=false
  false_condition: features.prometheus | bool is true
  skip_reason: Conditional result was False

PLAY [grafana]
*********************************************************************************
**********************
skipping: no hosts matched

PLAY RECAP
*********************************************************************************
**************
f1    : ok=84   changed=33   unreachable=0   failed=0   skipped=75   rescued=0   ignored=0
n1    : ok=37   changed=12   unreachable=0   failed=0   skipped=57   rescued=0   ignored=0
n2    : ok=37   changed=12   unreachable=0   failed=0   skipped=48   rescued=0   ignored=0
```

At this point, the OpenNebula platform should be up and running and functioning correctly.

6.5. Installation verification

From the server acting as *front-end,* you can verify that the OpenNebula services are running:

```
systemctl status opennebula.service
```

```
(one-deploy) [root@F55F9AB my-one]# systemctl status opennebula.service

● opennebula.service - OpenNebula Cloud Controller Daemon
     Loaded: loaded (/usr/lib/systemd/system/opennebula.service; enabled; preset: disabled)
     Active: active (running) since Thu 2025-12-18 11:46:47 UTC; 2 weeks 0 days ago
   Main PID: 215802 (oned)
```

```
     Tasks: 80 (limit: 408618)
    Memory: 674.4M (peak: 1.0G)
       CPU: 34min 55.859s
    CGroup: /system.slice/opennebula.service
            ├─215802 /usr/bin/oned -f
            ├─215834 /usr/bin/ruby /usr/lib/one/mads/one_hm.rb -p 2101 -l 2102 -b 127.0.0.1
            ├─215858 /usr/bin/ruby /usr/lib/one/mads/one_vmm_exec.rb -t 15 -r 0 kvm -p
            ├─215875 /usr/bin/ruby /usr/lib/one/mads/one_vmm_exec.rb -t 15 -r 0 lxc
            ├─215892 /usr/bin/ruby /usr/lib/one/mads/one_vmm_exec.rb -t 15 -r 0 kvm
            ├─215911 /usr/bin/ruby /usr/lib/one/mads/one_tm.rb -t 15 -d
dummy,lvm,shared,fs_lvm,fs_lvm_ssh,qcow2,ssh,local,ceph,dev,iscsi_libvirt,netapp
            ├─215934 /usr/bin/ruby /usr/lib/one/mads/one_auth_mad.rb --authn
ssh,x509,ldap,server_cipher,server_x509
            ├─215949 /usr/bin/ruby /usr/lib/one/mads/one_datastore.rb -t 15 -d
dummy,fs,lvm,ceph,dev,iscsi_libvirt,restic,rsync,netapp -s
dummy,shared,ssh,local,ceph,fs_lvm,fs_lvm_ssh,qcow2,netapp
            ├─215972 /usr/bin/ruby /usr/lib/one/mads/one_market.rb -t 15 -m
http,s3,one,linuxcontainers
            ├─215989 /usr/bin/ruby /usr/lib/one/mads/one_ipam.rb -t 1 -i
dummy,aws,equinix,vultr,scaleway
            ├─216003 /usr/bin/ruby /usr/lib/one/mads/one_sched.rb -t 15 -p rank -o one_drs
            ├─216030 /usr/lib/one/mads/onemonitord "-c monitord.conf"
            ├─216047 /usr/bin/ruby /usr/lib/one/mads/one_im_exec.rb -r 3 -t 15 -w 90 kvm
            ├─216060 /usr/bin/ruby /usr/lib/one/mads/one_im_exec.rb -r 3 -t 15 -w 90 lxc
            └─216073 /usr/bin/ruby /usr/lib/one/mads/one_im_exec.rb -r 3 -t 15 -w 90 qemu
Dec 18 11:46:43 F55F9AB.online-server.cloud systemd[1]: Starting OpenNebula Cloud Controller
Daemon...
Dec 18 11:46:43 F55F9AB.online-server.cloud opennebula[215798]: gzip: /var/log/one/oned*.log:
No such file or directory
Dec 18 11:46:43 F55F9AB.online-server.cloud opennebula[215800]: gzip:
/var/log/one/monitor*.log: No such file or directory
Dec 18 11:46:47 F55F9AB.online-server.cloud systemd[1]: Started OpenNebula Cloud Controller
Daemon.
```

Next, we will verify that the resources are operational. First, we must log in with our user account *oneadmin*:

```
sudo -i -u oneadmin
```

And now we can verify that the hosts are working correctly by running the following command:

```
onehost list
```

```
[oneadmin@F55F9AB ~]$ onehost list

ID NAME            CLUSTER   TVM   ALLOCATED_CPU    ALLOCATED_MEM    STAT
1 192.168.0.4      default   1     100 / 1600 (6%)  128 / 62.4G (0%)  on
0 192.168.0.3      default   0     100 / 1600 (6%)  128 / 62.4G (0%)  on
```

We can verify that the two servers specified in the file *example.yml*, which we defined in section 6.2. Configuring cloud parameters, They are running as platform hypervisor nodes, since the STAT column (highlighted in blue) shows the value "on" and not "err".

Next, to check the data stores, we run the following command:

```
onedatastore list
```

```
[oneadmin@F55F9AB ~]$ onedatastore list
ID  NAME      SIZE      AVA     CLUSTERS   IMAGES   TYPE   DS     TM      STAT
 2  files     932.3G    94%     0                0   fil    fs     local   on
 1  default   932.3G    94%     0                1   img    fs     local   on
 0  system    -         -       0                0   sys    -      local   on
[oneadmin@F55F9AB ~]$
```

As in the case of *onehost, we* verified that the last column of the command output, STAT, shows the value “on” and not “err”.

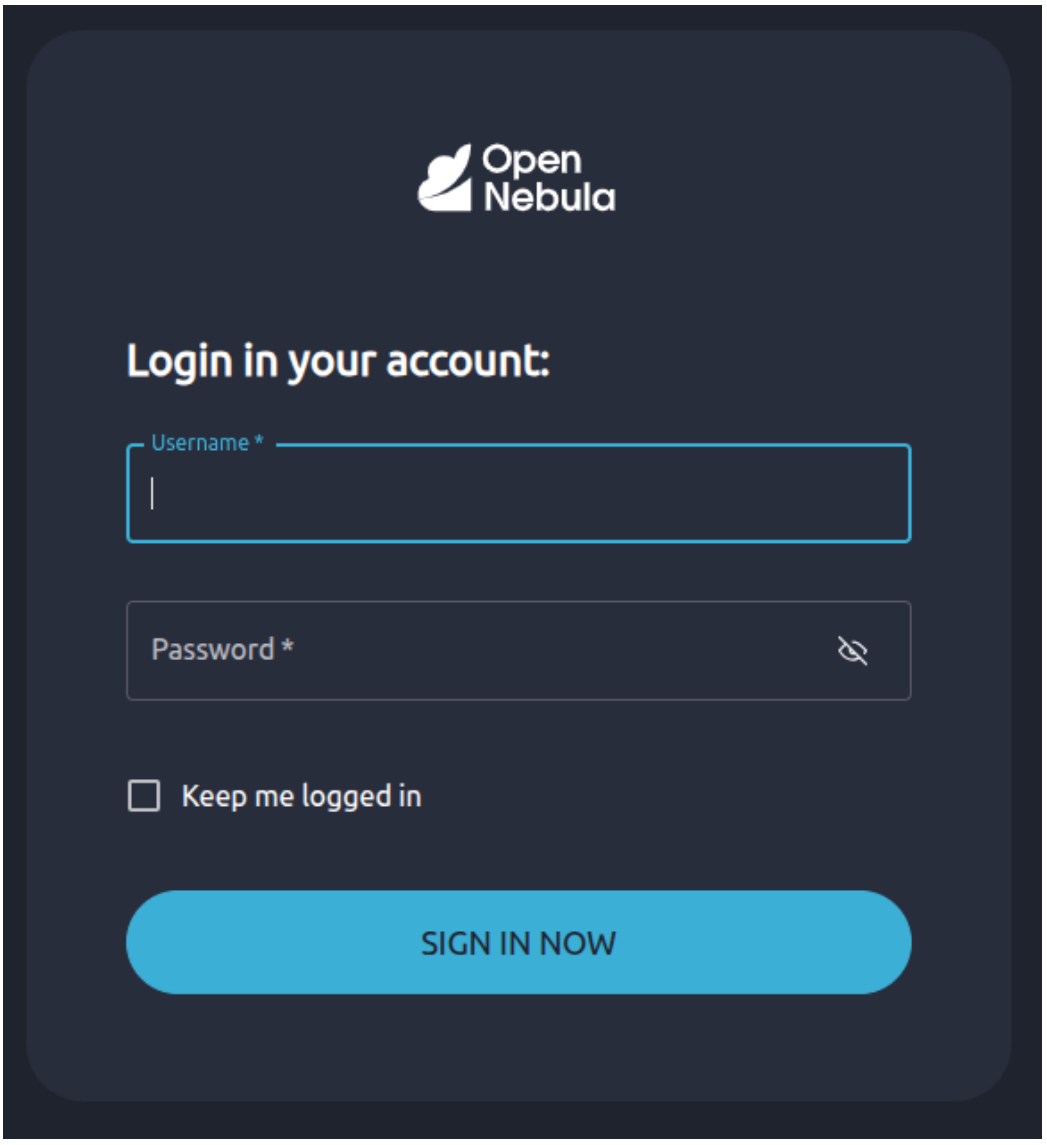
Finally, we verified that the virtual network created as part of the implementation (remember that, in this example, we defined it as *admin_net* within the file *example.yml*), by executing the following command:

```
onevnet list
```

```
[oneadmin@F55F9AB ~]$  onevnet list
  ID  USER          GROUP         NAME          CLUSTERS  BRIDGE  STATELEASES OUTD ERROR
0 oneadmin oneadmin admin_net 0 br0 rdy 1 0 0
[oneadmin@F55F9AB ~]$
```

In this case, the STATE column should display the value “rdy”.

Everything is working! We can connect to the user interface of Sunstone OpenNebula's web interface, which allows us to manage and interact with our virtual machines, and is accessible at the node that acts as *front-end.* On any machine that has connectivity to this node, type the following address into your browser: http://<*IP from the front-end*>:2616.



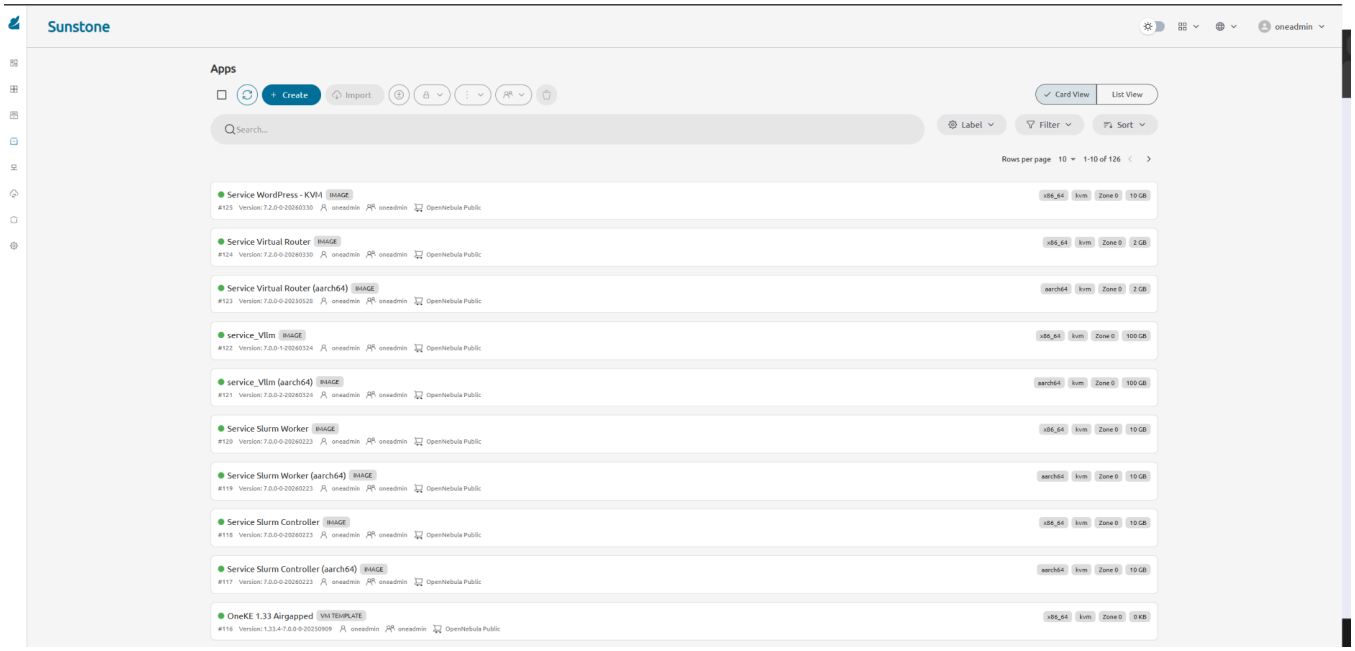
You can log in with the username *oneadmin* and the password you defined within the parameter*one_pass*in the configuration file *example.yml.* Next, we will test the platform's functionality by creating and deploying a test virtual machine.

7. Creating servers withOpenNebula

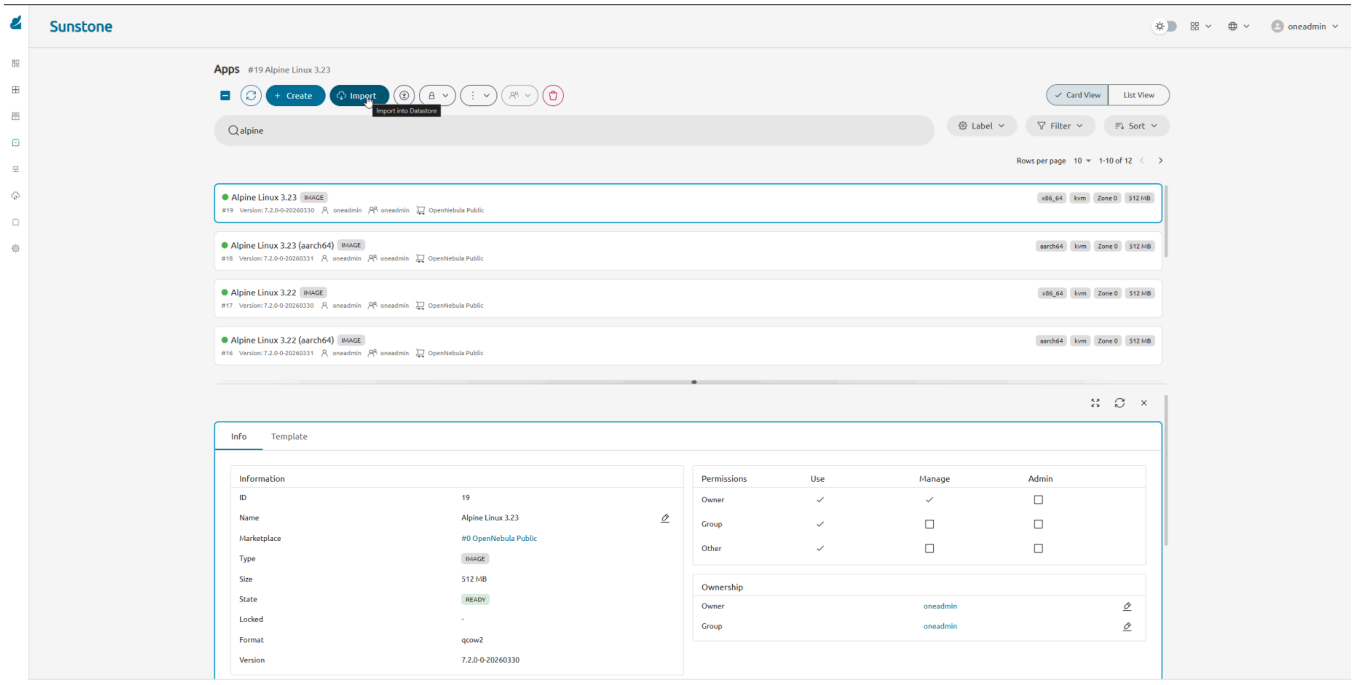
Before starting the installation of a virtual machine, it is necessary to download at least one image.

7.1. Download an image

Once you have logged into the platform, go to the Storage → App section in the left panel and find the image you are going to use:

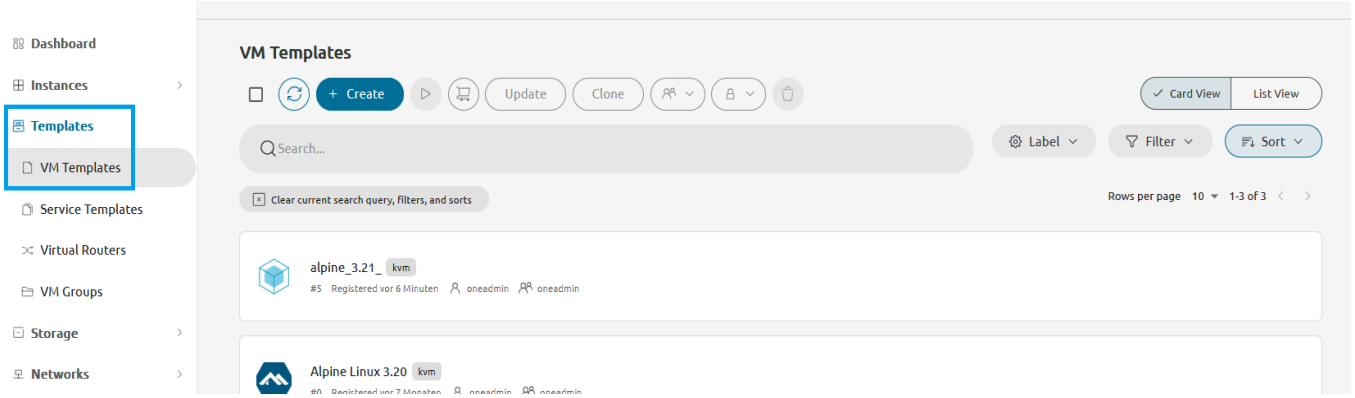


Once you've located it, click on *Matter* to start the process that will automatically create the template from the selected image.

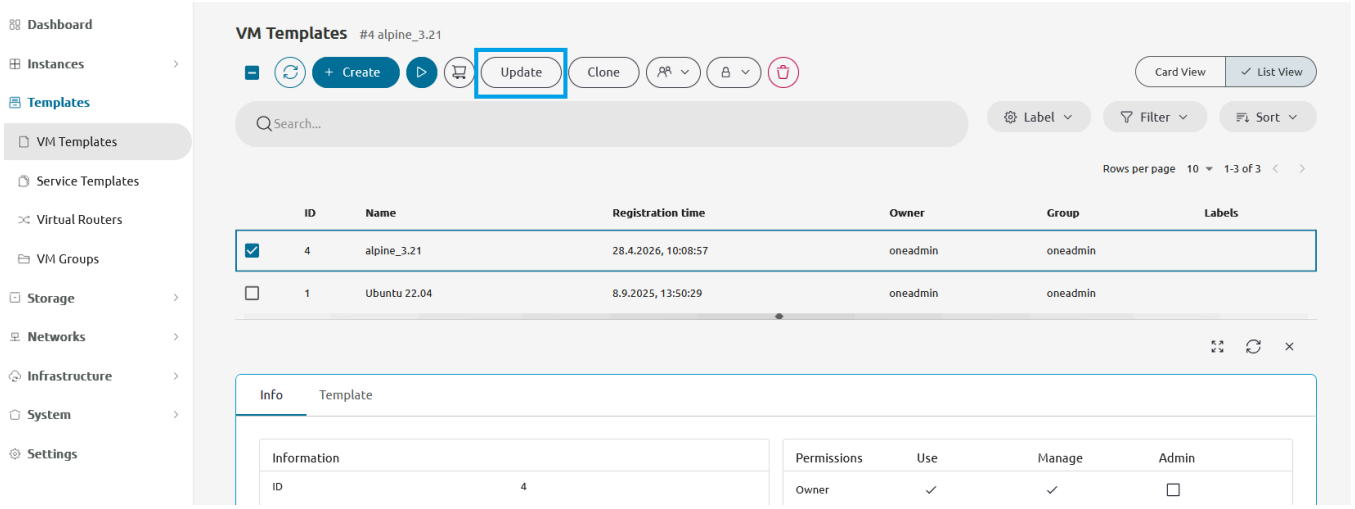


7.2. Updating the VM creation template

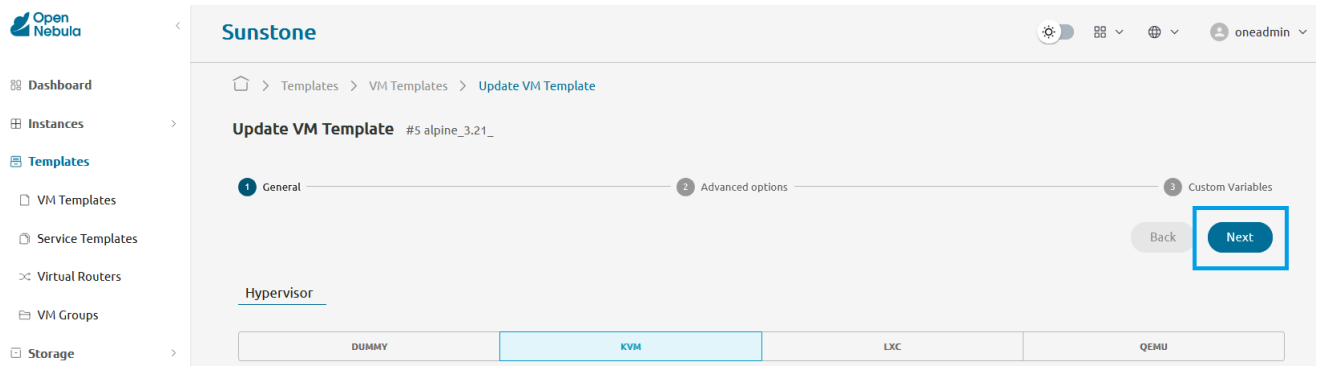
From the side menu, access the virtual machine templates section:



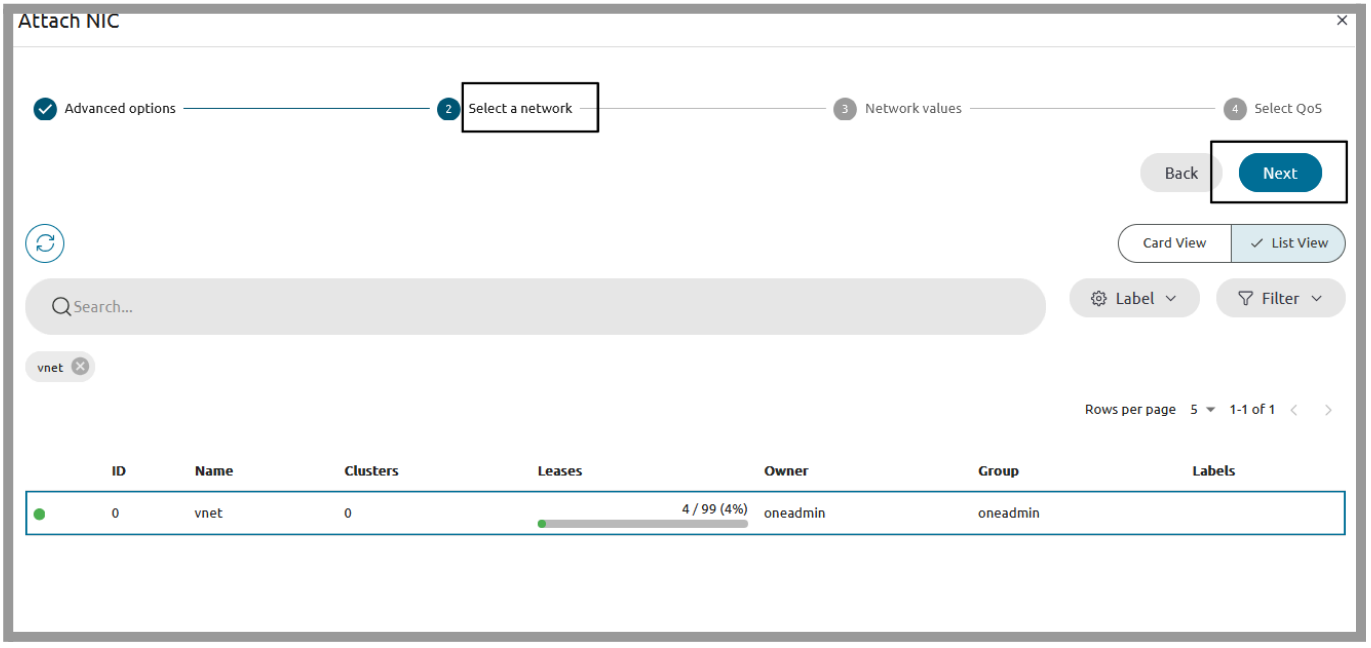
Select the template you will use to create your virtual machines, and press the button *Update*:



And press the button directly *Following* to access advanced template options:



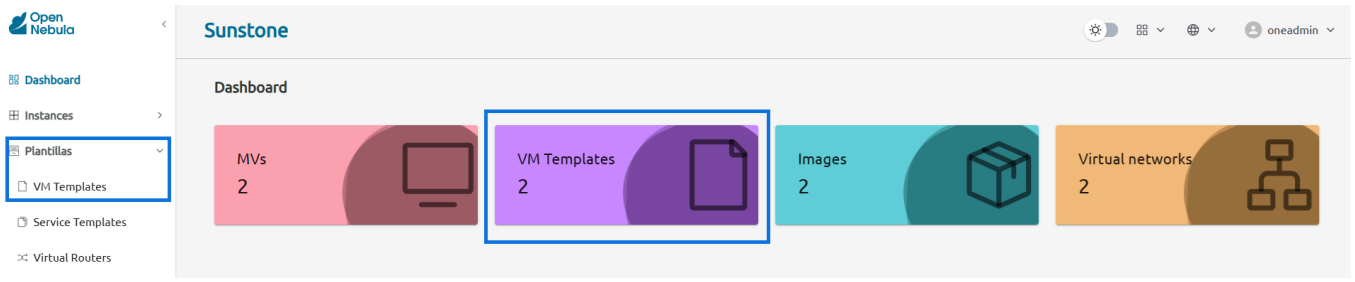
Within these, go to the tab of *Red*. Press the button directly *Following* to access the list of available networks, select the VNET you just created, then click again *Following* and finally *Finish* to finish updating the template.



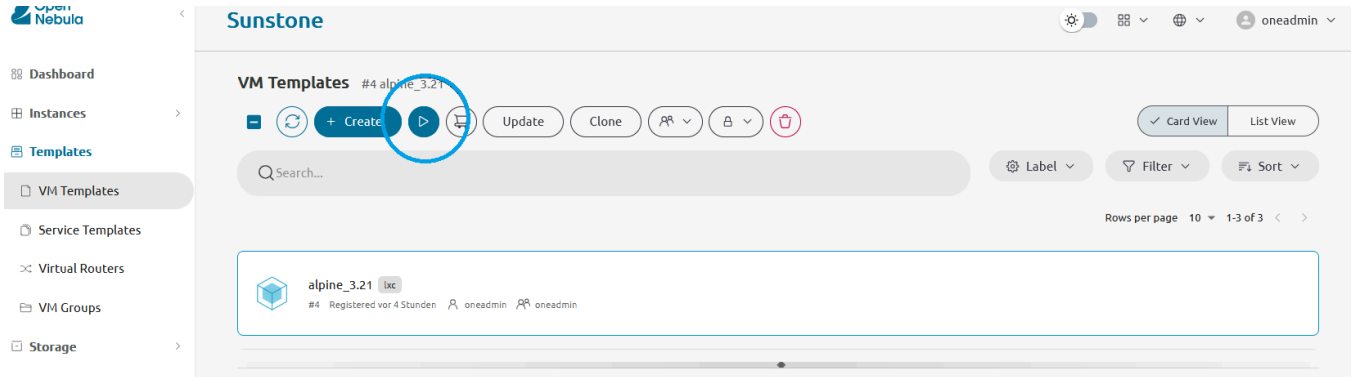
Now, any VM created from the template will apply the appropriate network configuration.

7.3. Instantiate a virtual machine

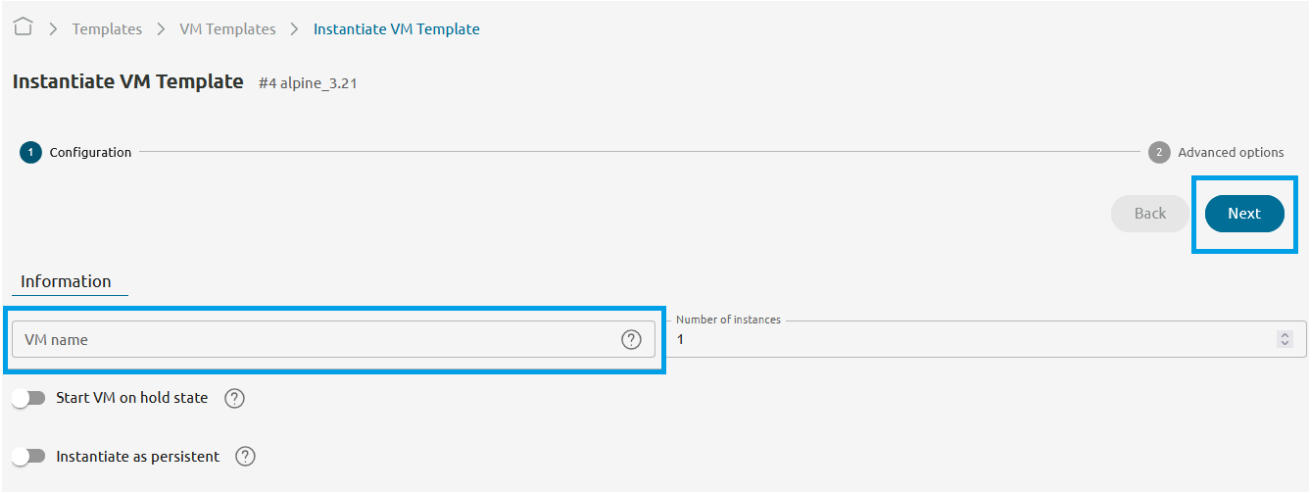
Access the templates section again, either from the side menu as we did at the beginning of the section 7.2 VM creation template update or you can also find a shortcut from the panel's dashboard itself.



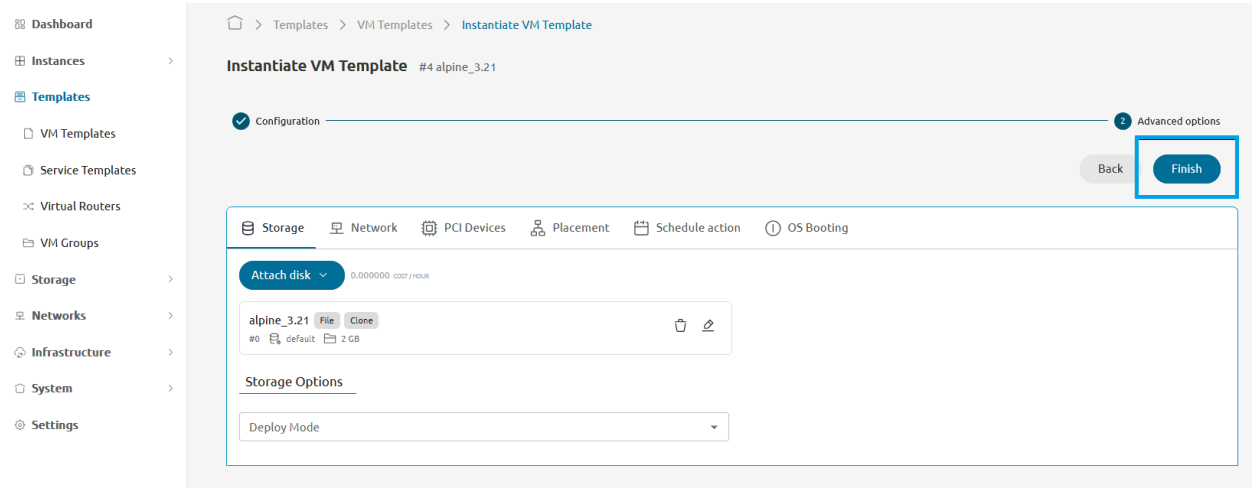
Select the template we just updated and press the icon *Instantiate*:



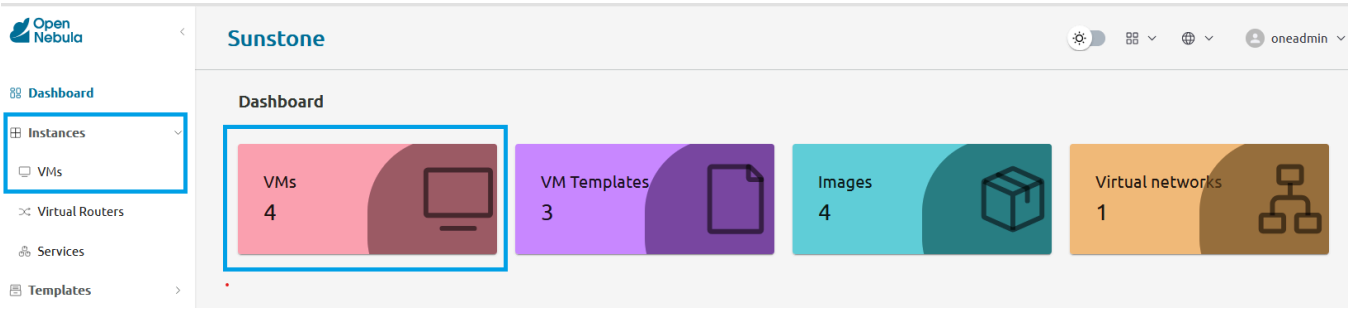
At this point we can configure several parameters, but the only mandatory one is the *VM name* Complete it, and press *Following* to access advanced creation options.



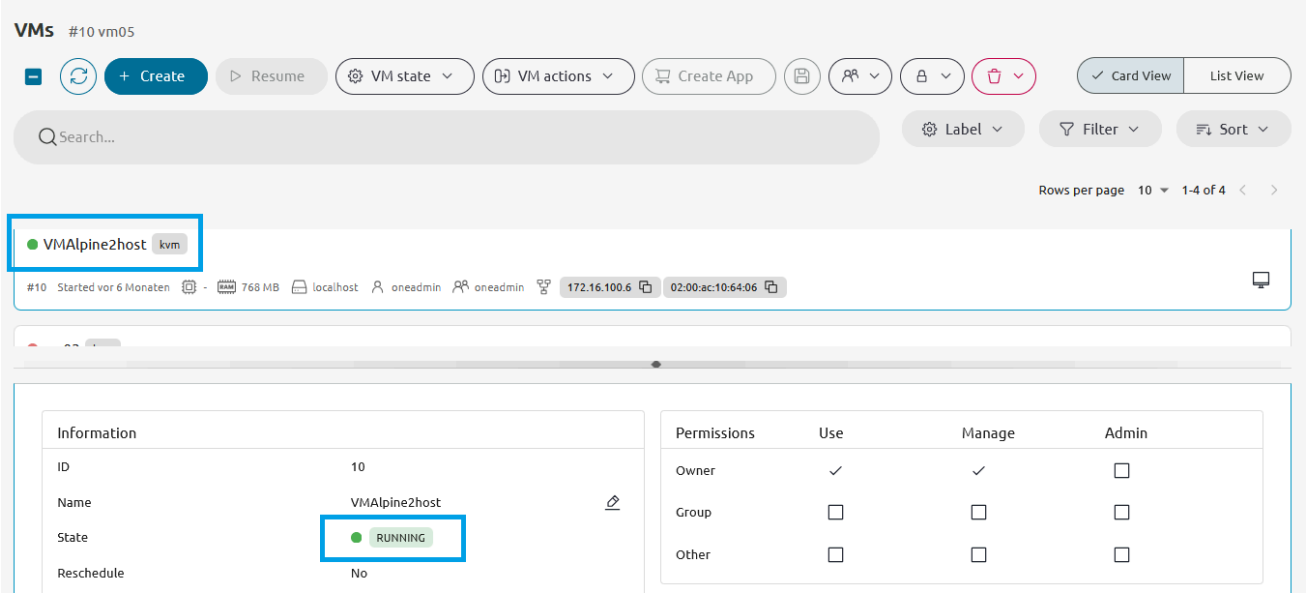
Since we have already configured the template according to our needs in the previous section, simply click the button *Finish* to launch the installation process.



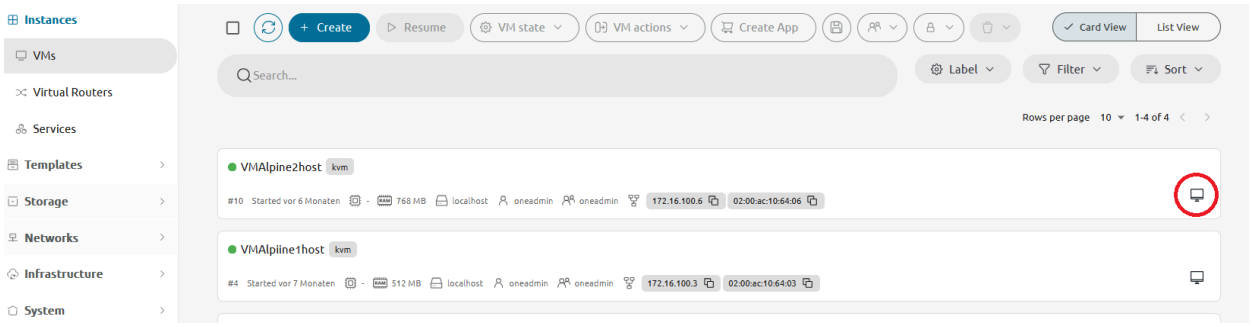
Accessing the section of *VMs* inside *Instances* from the side menu, or from the panel's dashboard itself, you can view the virtual machines you have created.



If the creation process we just launched has finished successfully, you will see the new machine already deployed, active, and in state *RUNNING*.

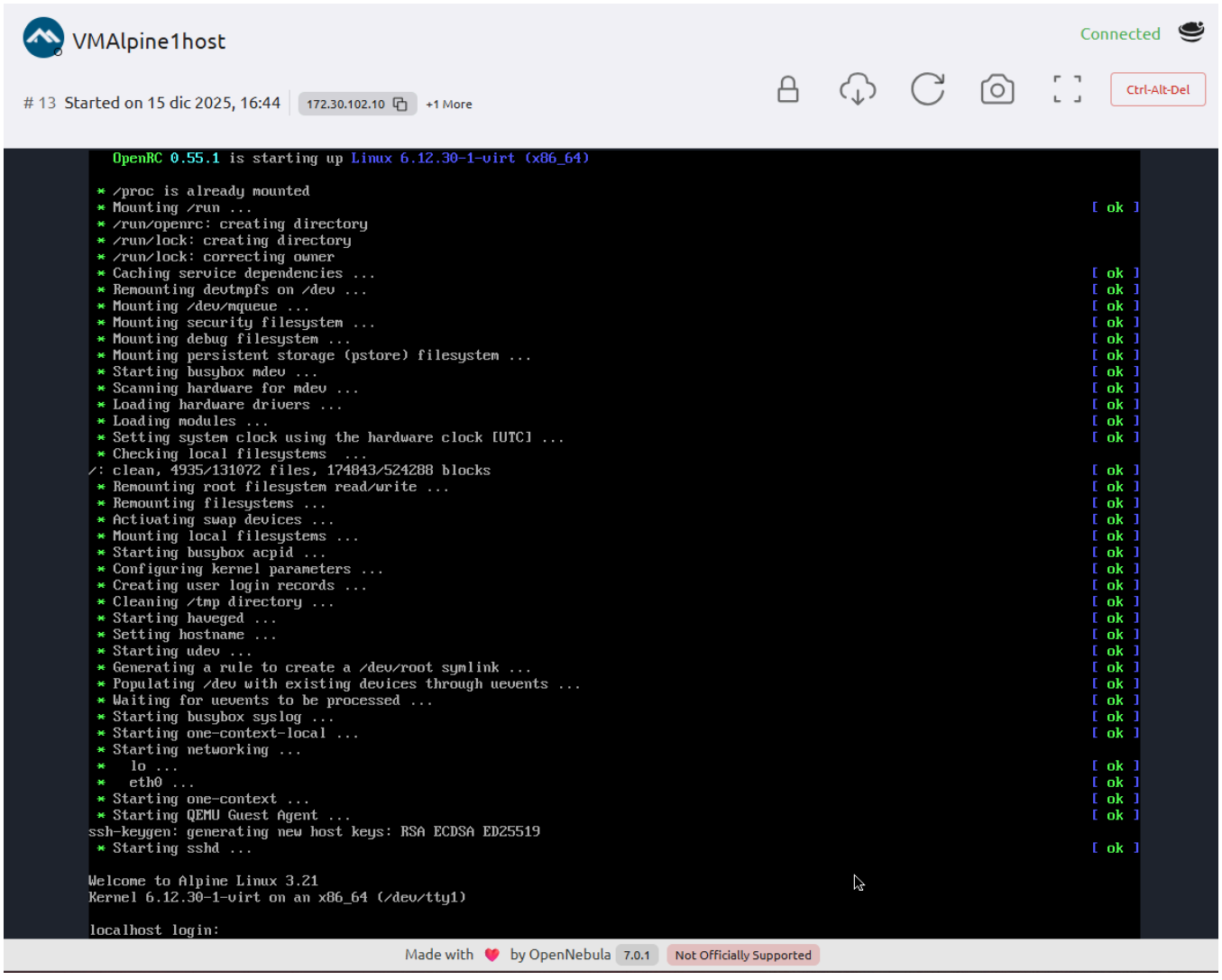


Access the MV console by pressing the VNC icon on the right side of the summary list:



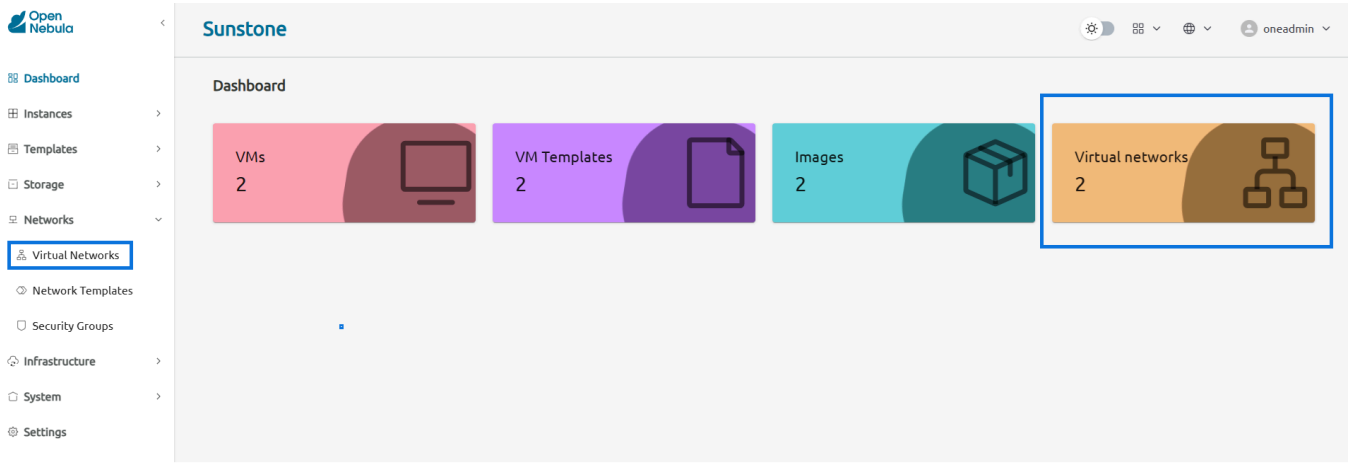
Finally, to log in to the virtual machine, enter the following credentials:

- User: root
- Password: opennebula

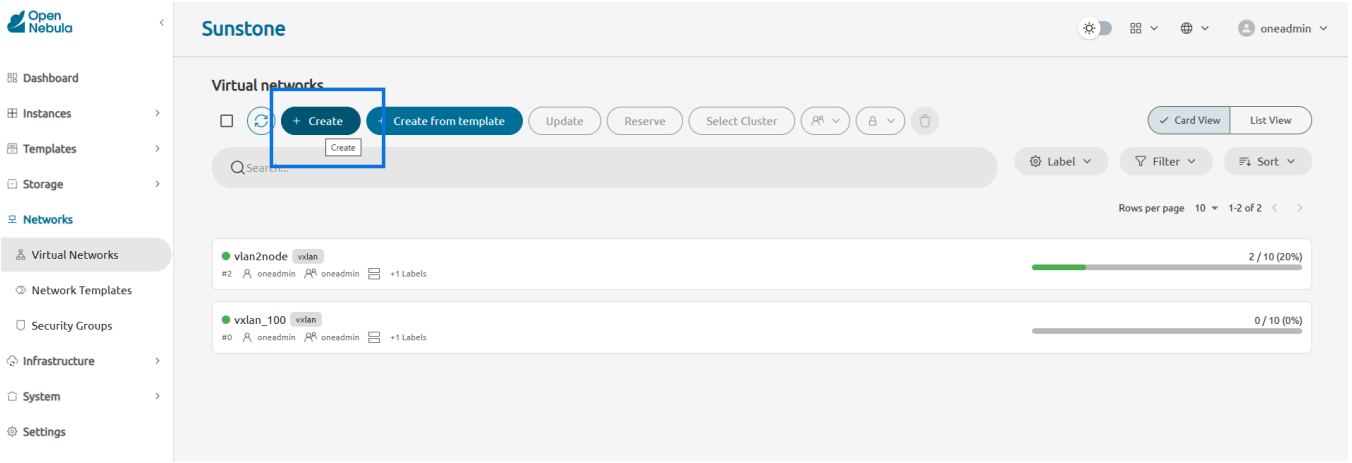


APPENDIX A: Creation of a virtual network

Once you have logged into the platform, access the section of *Virtual networks.* You can do it from the side menu, or from the application's dashboard itself.

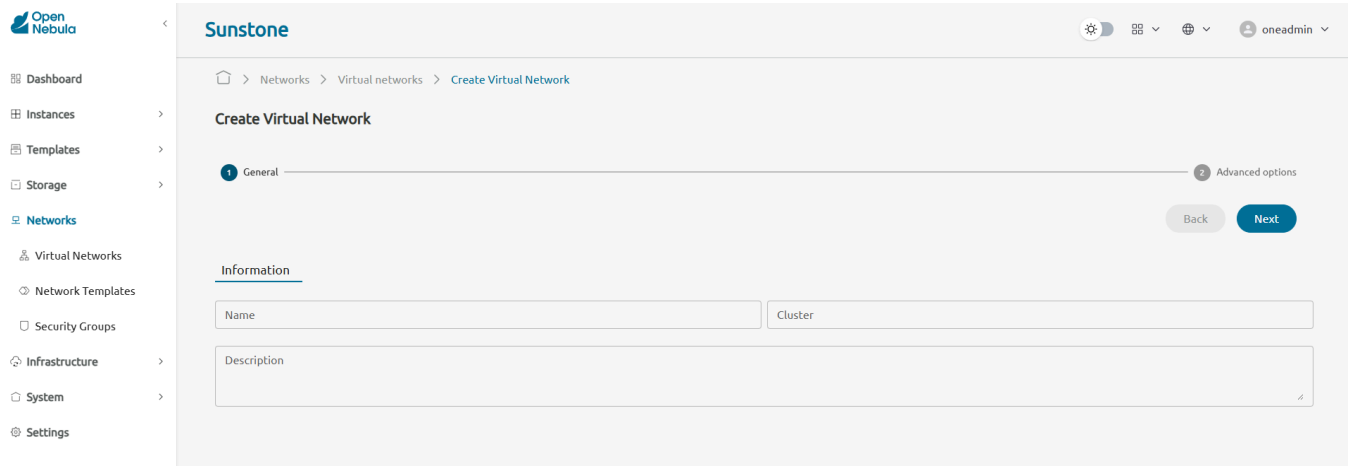


In our example panel, we see that we already have two networks created. In your case, you can use the button *Create* to add a new one:

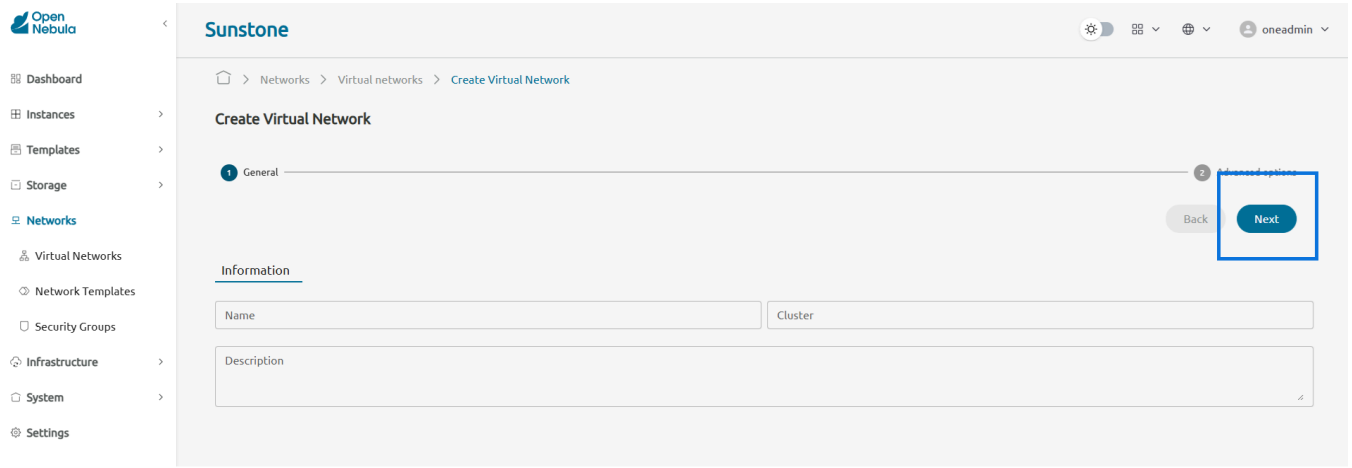


Complete the basic network information (name, description, and a cluster with which the network will be associated). You could specify, for example:

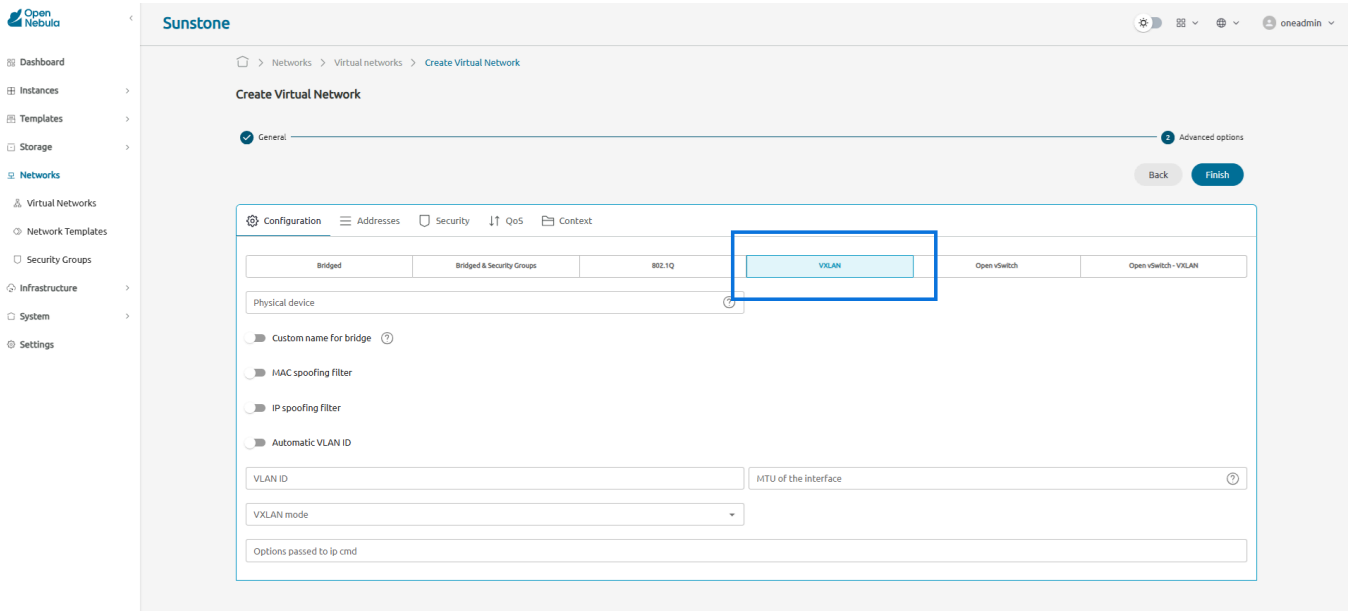
- Name: *vxlan_100*
- Cluster:*#0 default*
- Description: *Setting up the first virtual machine deployment*.

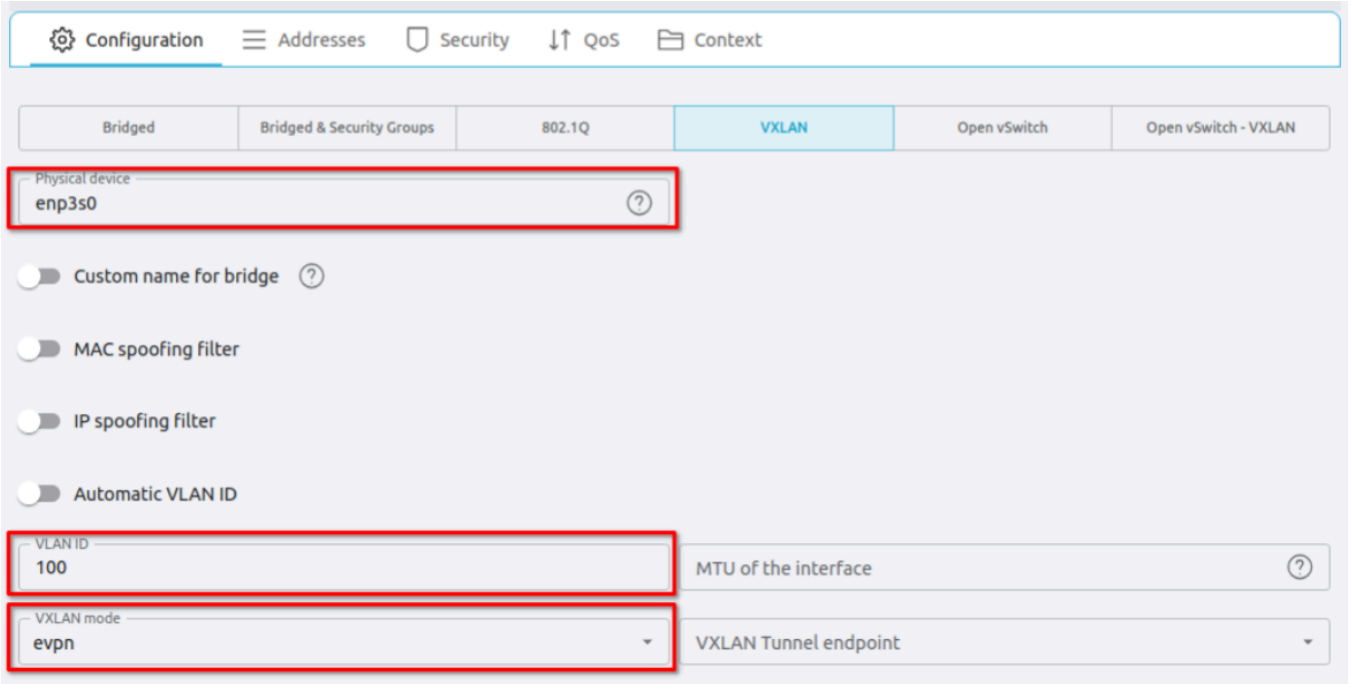


Press the Next button to continue:

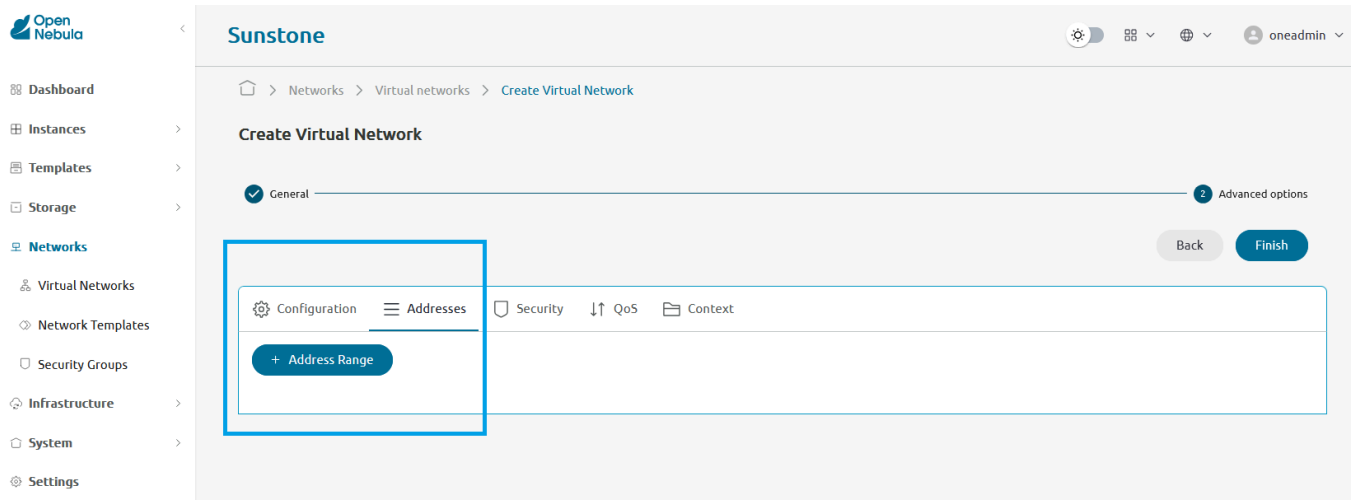


Within the VXLAN tab of the options menu, specify the name of the physical network card to which the traffic will be routed (for example, *enp3s0*), the VLAN identifier and configure the VXLAN mode in the drop-down menu to the value*vpn*.

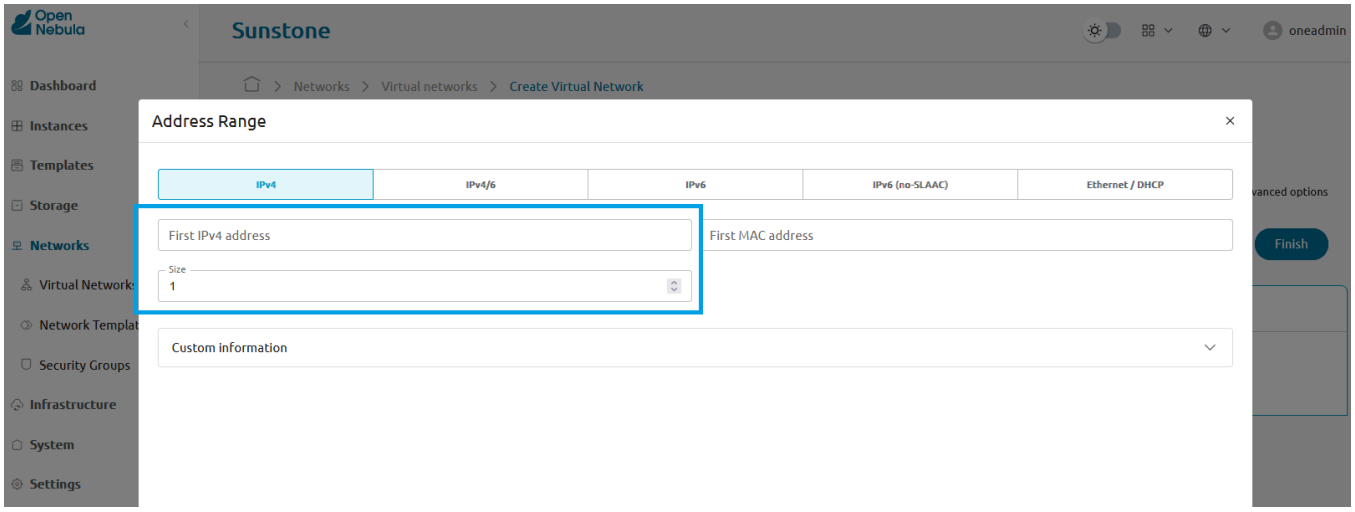


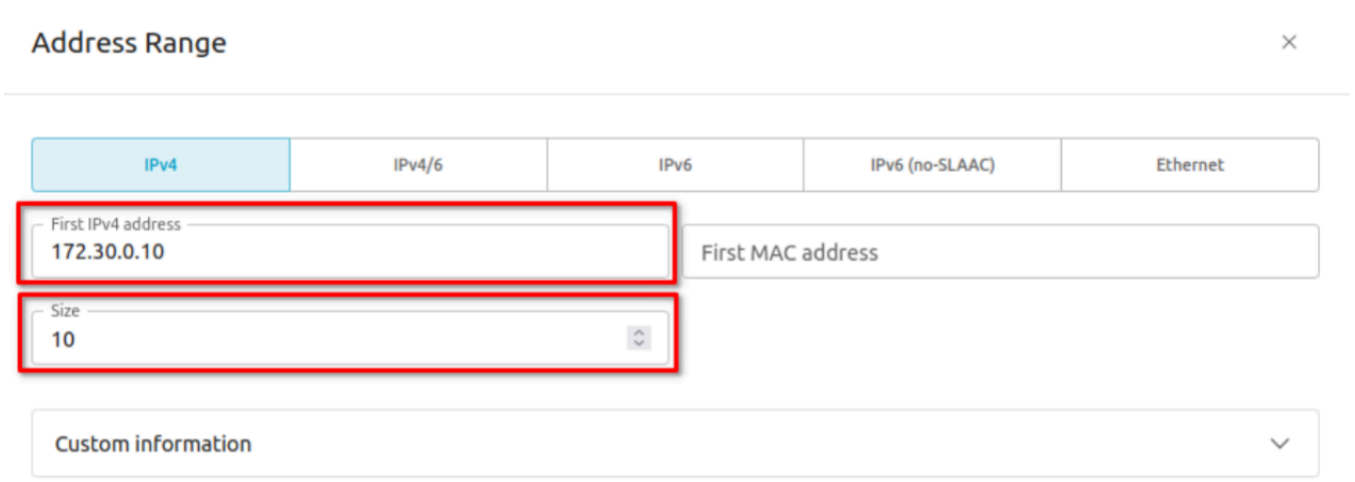


Within the tab *Addresses*, select the option *Add Range*.

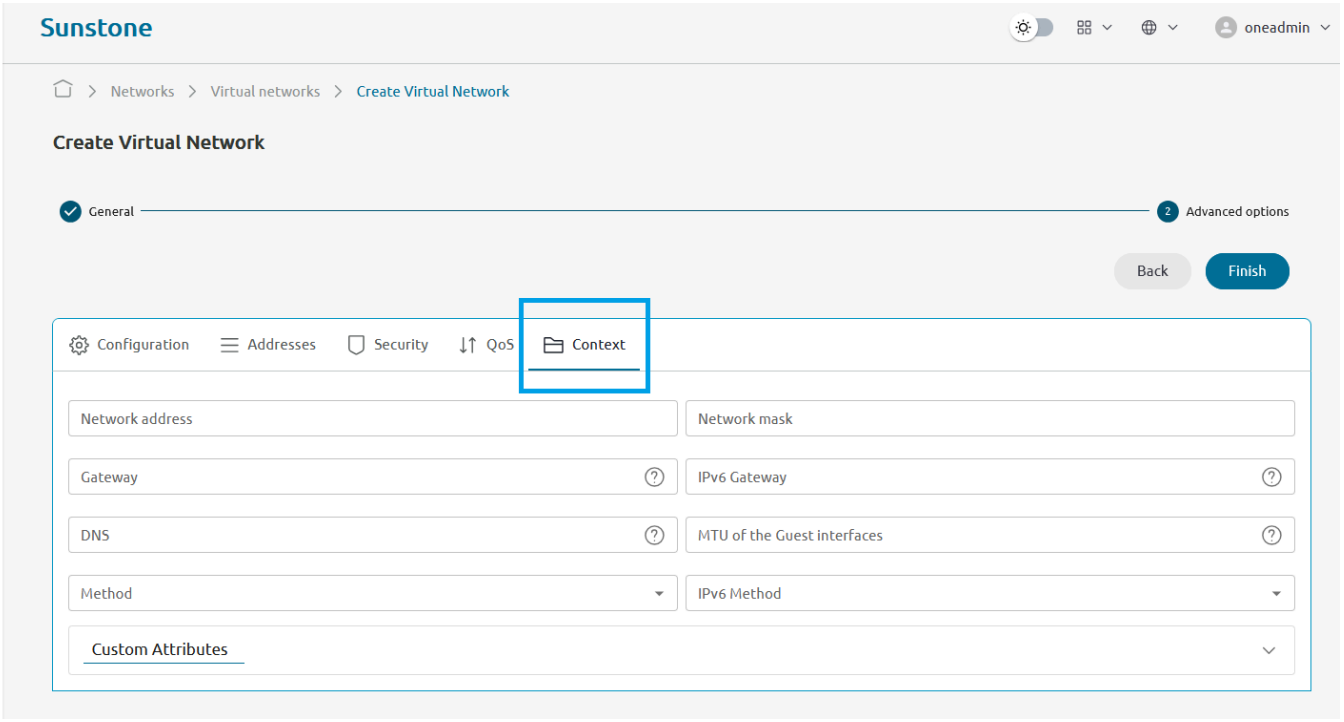


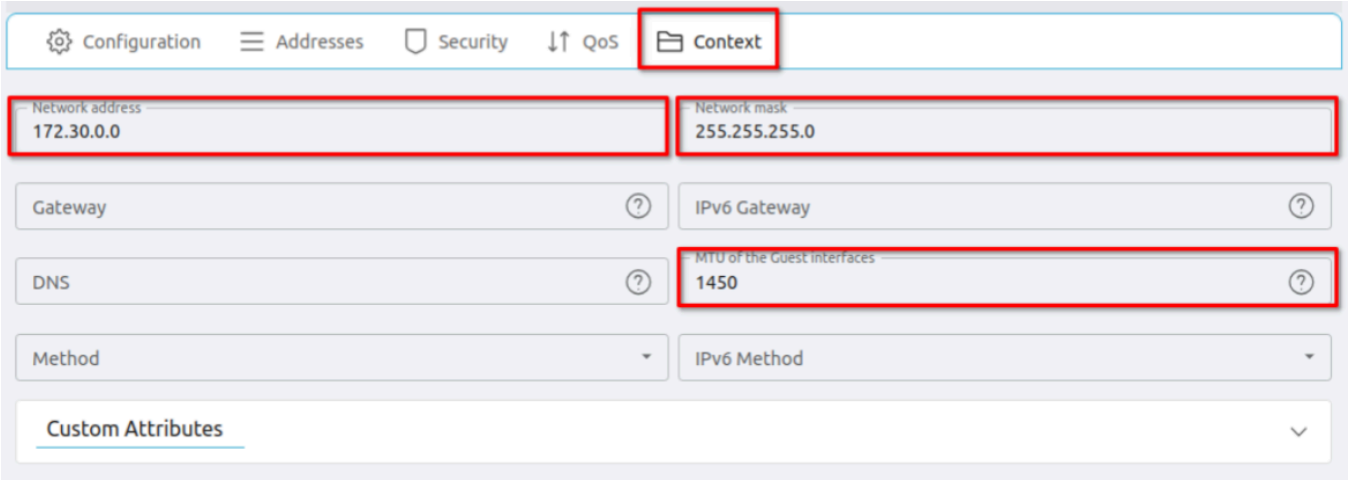
It indicates the first IP address of the network, and its size:





Finally, in the context tab, configure the network address, network mask, MTU value for the guest interface, and any other parameters that may be necessary.





Press the button *Finish* to apply the configuration and create the virtual network.

Imprint

IONOS SE
Elgendorfer Str. 57
56410 Montabaur, Germany

Contact IONOS
Phone: +49 (0) 721 170 5522
E-mail info@ionos.de
Website: https://www.ionos.de

Executive Board
Hüseyin Dogan, Arthur Mai, Dr. Andreas Nauerz, Dr. Markus Noga, Dr. Jens-Christian Reich, Patrik Heider, Achim Weiß

Chairman of the Supervisory Board
Sven Fritz

Commercial register
IONOS SE: Amtsgericht Montabaur / HRB 24498

VAT ID:
IONOS SE: DE815563912

Copyright