PROJECT
Multi-tenant cybersecurity operations centre designed
as an open cloud service with machine learning elements

Linked to work package KPB1 – Design of the virt. solution

Linked to output
V3 – Description of the selected solution meeting the defined criteria

type – documentation

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Introduction **3**
1 OpenStack cloud platform **4**
1.1 What is OpenStack? 4
1.2 Installation of KIS OpenStack Yoga [7] 5
1.2.1 What is the Metal-as-a-Service(MAAS) server used for? 5
1.2.2 What is the Juju server used for? 6
1.2.3 Type of OpenStack installation 6
1.2.4 Installation of the MAAS server 10
1.2.5 Basic DHCP configuration in MAAS 14
1.2.6 Upgrading MAAS 14
1.2.7 Deployment of the Juju server 15
1.2.8 Adding servers intended for running OpenStack 19
1.2.9 Network configuration in the MAAS system 20
1.2.10 Changing the network configuration for the Controller Node 21
1.2.11 Changing the network configuration for Compute Nodes 24
1.2.12 Changing the disk configuration for Compute Nodes 26
1.2.13 Setting the minimal kernel 27
1.2.14 Installation package (bundle) for OpenStack 29
1.2.15 Installation of OpenStack Yoga 32
1.2.16 Configuring the MAAS server for automatic authentication for the OpenStack CLI client 35
1.2.17 Creating external networks for OpenStack 36
1.2.18 Deleting component IP addresses in MAAS 38
2 Verification **40**
Conclusion **44**
List of references **45**

Introduction

Cloud and cloud services are an inseparable part of today's world. We are undoubtedly convinced of this in situations where large companies move the majority of their infrastructure to the cloud. For the client, everything is often simpler, faster and more cost-effective. Examples include Amazon Web Services, Google Cloud Platform, Microsoft Azure or IBM Services. In recent years, not only computing power and data but also the gaming industry (gaming-as-a-service) have moved to the cloud, including GeForce Now, Google Stadia, Playstation Now and others.

In 2010, a project was launched by its original authors, the National Aeronautics and Space Administration (NASA) and Rackspace Hosting, known today as OpenStack. It is mostly deployed as “infrastructure as a service” (IaaS) in both public and private clouds, where virtual servers and other resources are made available to users. The software platform consists of interconnected components that control diverse hardware pools of processing, storage and networking resources from multiple vendors throughout the data centre. Users manage it either through a web-based dashboard or command-line tools. OpenStack has a modular architecture with various names for its components, such as: Compute (Nova), Networking (Neutron), Block storage (Cinder), Identity (Keystone), Object storage (Swift), Dashboard (Horizon). With each new version, more and more new components were added; compared with the initial version with two components, Nova and Swift, the current version, named Yoga (January 2023), offers the option of installing 38 components. There are several options for installing OpenStack: OpenStack Charms, OpenStack “kolla-ansible”, OpenStack-Ansible or TripleO.

It is precisely this free open-source cloud computing platform that we will focus on in this work.

1 OpenStack cloud platform

1.1 What is OpenStack?

OpenStack is a cloud operating system that controls large pools of compute, storage and networking resources within a data centre, with all of them managed and provisioned through APIs with common authentication mechanisms. A dashboard is also available that gives administrators control and at the same time allows users to provision resources through a web interface. In addition to the standard infrastructure as a service functionality, additional components provide, among other services, orchestration, fault management and service management to ensure high availability of user applications.

OpenStack is divided into services that allow components to be plugged in depending on the user's needs. The OpenStack map provides a view of the OpenStack lifecycle so that the user can see where these services fit and how they can work together [2].

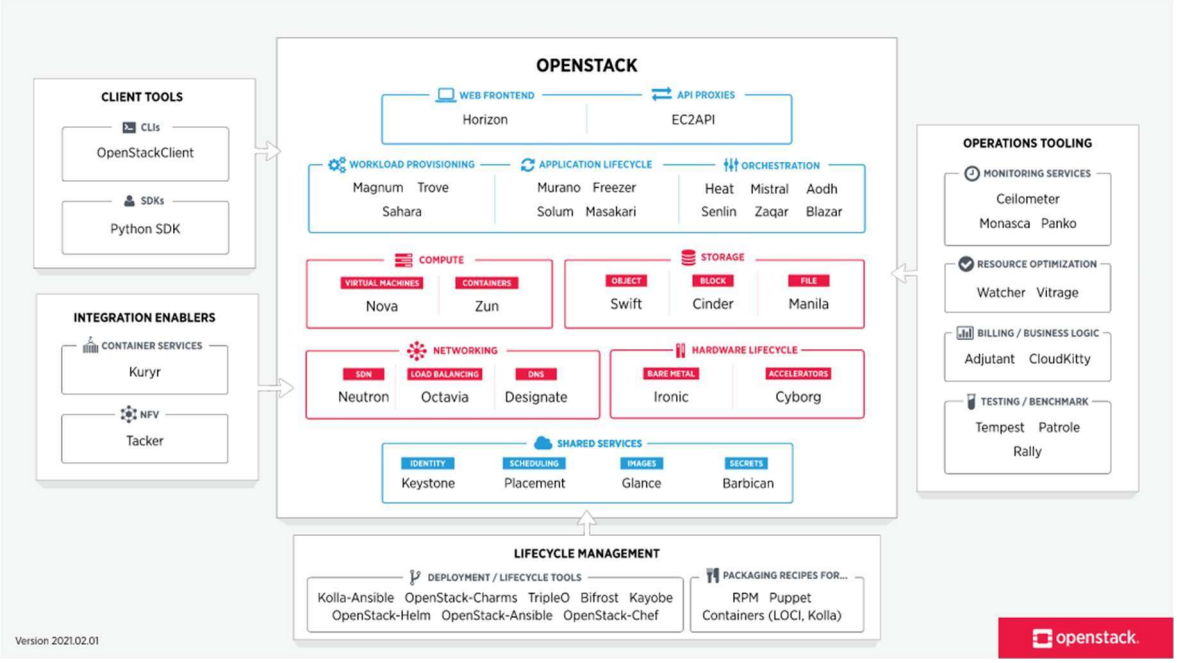


Figure 1 – OpenStack map

1.2 Installation of KIS OpenStack Yoga [7]

1.2.1 What is the Metal-as-a-Service(MAAS) server used for?

Metal-as-a-Service (MAAS) provides complete automation of physical servers for efficient operation of servers/data centres. The software is open-source and is backed by Canonical, the company behind the Ubuntu Linux distribution. MAAS treats physical servers like virtual machines (instances) in the cloud. This solution makes it possible to manage several servers at once and from a single location, so that they do not have to be managed individually and in a complicated way. Standard or customized operating systems can thus be deployed remotely to physical or virtual machines. MAAS comprehensively meets the need for rapid deployment, destruction and reconfiguration of the operating system. It is currently deployed in banking, telecommunications and industrial environments, as well as for specialized purposes such as front-end supercomputer verification, music streaming services, disaster recovery and computer security risk analysis. For our purposes, the main key function of the MAAS server will be automation, whereby, using the Intelligent Platform Management Interface (IPMI), MAAS powers on/restarts the given machine and, using the Preboot eXecution Environment (PXE) either over an IPv4 or IPv6 network, installs the requested or default operating system for this task. Other useful functions include storage configuration, network interface configuration and IP address management using the Dynamic Host Configuration Protocol (DHCP) [8].

1.2.2 What is the Juju server used for?

Juju is an open-source framework that uses Charmed Operators or “Charms” to deploy cloud infrastructure and applications and to manage their operations from launch. With Juju, it is possible to install, maintain, upgrade and integrate applications across clusters, Kubernetes containers, virtual and physical machines on public or private clouds. The vendors describe Juju as software that manages your software. It helps you take control of all your applications, infrastructure and environments. Use of this software should ensure:

- saving time spent on managing scripts
- minimize costs
- ensure redundancy and resilience
- monitor all activities across the infrastructure
- maximize the hybrid cloud architecture

Charms are small applications that contain common sets of function instructions to ensure repeatable and reliable code. This makes it possible to manage applications and scenarios instead of configurations; however, if necessary, there is still the option to dive into the configuration files YAML (YAML Ain’t Markup Language). Charmed Operators are used not only for deploying and managing individual applications, but also for connecting them with each other in models. The application model defines which applications provide a service and how they are related. Juju can be controlled via the command line (CLI) or via a graphical web interface (GUI) such as the Juju GUI [9].

1.2.3 Type of OpenStack installation

For the installation of OpenStack itself, OpenStack-Charms was chosen as the most suitable method; Charmed Operators or also called “Charms” are used to deploy and manage OpenStack clouds using Metal-as-a-Service (MAAS) and Juju. First, it will be necessary to install the MAAS server and use it to deploy the Juju server. Subsequently, using the bundle configuration file, we will be able to build OpenStack using the Charms method. For the installation we have available virtual servers, which will be used to install MAAS, Juju and as the Controller

Node. Other physical servers will primarily serve to run virtual instances in OpenStack using the Nova component. Below in the tables we find the allocation of individual components, the private IP addresses that are assigned to the servers, and the tagging they will receive in the MAAS system.

Machine	Components
controller	Ceph-radosgw, Cinder, Designate, Glance, Heat, Keystone, MySQL (x3), Neutron- api, Nova cloud controller, Openstack- dashboard, Ovn-central (x3), Placement, RabbitMQ, Vault
compute-1	Nova, NTP, Ovn-chassis
compute-2	Nova, NTP, Ovn-chassis
compute-3	Nova, NTP, Ovn-chassis
compute-4	Nova, NTP, Ovn-chassis
compute-7	Nova, NTP, Ovn-chassis, Ovn-Central, Ceph-Osd, Ceph-Mon, Manila, Manila- Ganesha, Ceph-FS
compute-8	Nova, NTP, Ovn-chassis, Ovn-Central, Ceph-Osd, Ceph-Mon, Manila, Manila- Ganesha, Ceph-FS
compute-9	Nova, NTP, Ovn-chassis, Ovn-Central, Ceph-Osd, Ceph-Mon, Manila, Manila- Ganesha, Ceph-FS
compute-10	Nova, NTP, Ovn-chassis, Ovn-Central, Ceph-Osd, Ceph-Mon, Manila, Manila- Ganesha, Ceph-FS
compute-11	Nova, NTP, Ovn-chassis, Ovn-Central, Ceph-Osd, Ceph-Mon, Manila, Manila- Ganesha, Ceph-FS

Table 1 – Allocation of components to individual machines

IP address	Purpose	Virtual/physical machine
10.11.0.1	Default gateway	-
10.11.0.2	MAAS	maas1 (VMWare)
10.11.0.3	Juju	juju1 (VMWare)
10.11.0.11	Controller Node	controller1 (VMWare)
10.11.0.21	Compute Node 1	IBM X3650M5
10.11.0.22	Compute Node 2	IBM X3650M5
10.11.0.23	Compute Node 3	IBM X3650M3
10.11.0.24	Compute Node 4	IBM X3650M5
10.11.0.27	Compute Node 7	HP DL380G9 iLO
10.11.0.28	Compute Node 8	HP DL380G9 iLO
10.11.0.29	Compute Node 9	HP DL380G9 iLO
10.11.0.30	Compute Node 10	Dell R720 iDRAC
10.11.0.31	Compute Node 11	IBM X3650M3

Table 2 – IP address range for OpenStack machines

Server designation in MAAS	Tag in MAAS
juju.maas	juju
controller.maas	controller
compute-1.maas	compute1
compute-2.maas	compute2
compute-3.maas	compute3
compute-4.maas	compute4
compute-7.maas	compute7
compute-8.maas	compute8
compute-9.maas	compute9
compute-10.maas	compute10
compute-11.maas	compute11

Table 3 – Tagging for machines in MAAS

1.2.4 Installation of the MAAS server

For the MAAS server, a virtual machine with 4 vCPUs (virtual central processing units), 8GB of RAM and a 50GB disk was created in our infrastructure. In this case, it was still necessary to manually install the operating system, specifically Linux Ubuntu in version 22.04.1 LTS. During the installation of the operating system, the static IP address 10.11.0.2 and the default gateway 10.11.0.1 were configured, and the DNS servers were set to the department's servers with IP addresses 158.193.152.4 and 158.193.152.11. The server name was set to "maas" and a user account "student" was created with the agreed password. As a first step, secure shell (SSH) keys were generated, which we will later use during initialization. Using the following command, we generate new SSH keys into the file /home/student/.ssh/student:

```
ssh-keygen -t rsa
```

Next, we can start installing MAAS. We will install MAAS in version 3.1; for the init command, we will use the IP address that we configured during the installation of the operating system. With the createadmin command, we create a user account with the name "student",

<PASSWORD> is replaced with the real password and the email is set to "student@kis.fri.uniza.sk.

```
sudo snap install maas-test-db
sudo snap install maas --channel=3.1/stable
sudo maas init region+rack --maas-url http://10.11.0.2:5240/MAAS --database-uri
maas-test-db:///
sudo maas createadmin --username student --password <PASSWORD> --email
student@kis.fri.uniza.sk
sudo maas apikey --username student > maas-api-key #saving the api key to a file
```

Next, we can continue with the initialization of MAAS directly in the browser, specifically at the address from the initialization: http://10.11.0.2:5240/MAAS. After the first login, it is necessary to perform the basic configuration. In the first step, we set the region name to "maas" and specify the department's DNS servers with IP addresses 158.193.152.4 and 158.193.152.11 as the DNS servers.

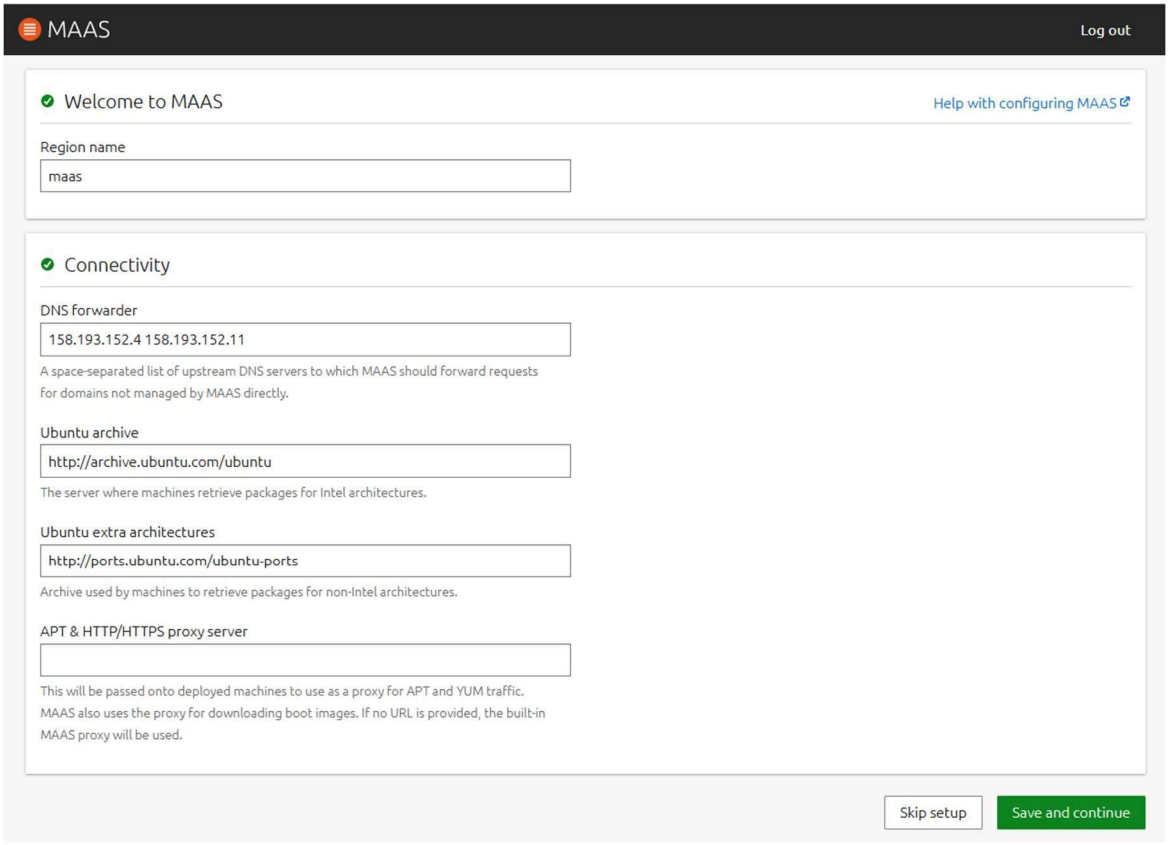


Figure 2 - Initial setup of MAAS – region name + DNS forwarders

In the second step, it is possible to download additional images for booting besides the base version 20.04 LTS with the amd64 architecture; given that we want the OpenStack Yoga version, it is necessary to also download the 22.04 LTS image with amd64 architecture and confirm with the button “Update selection”.

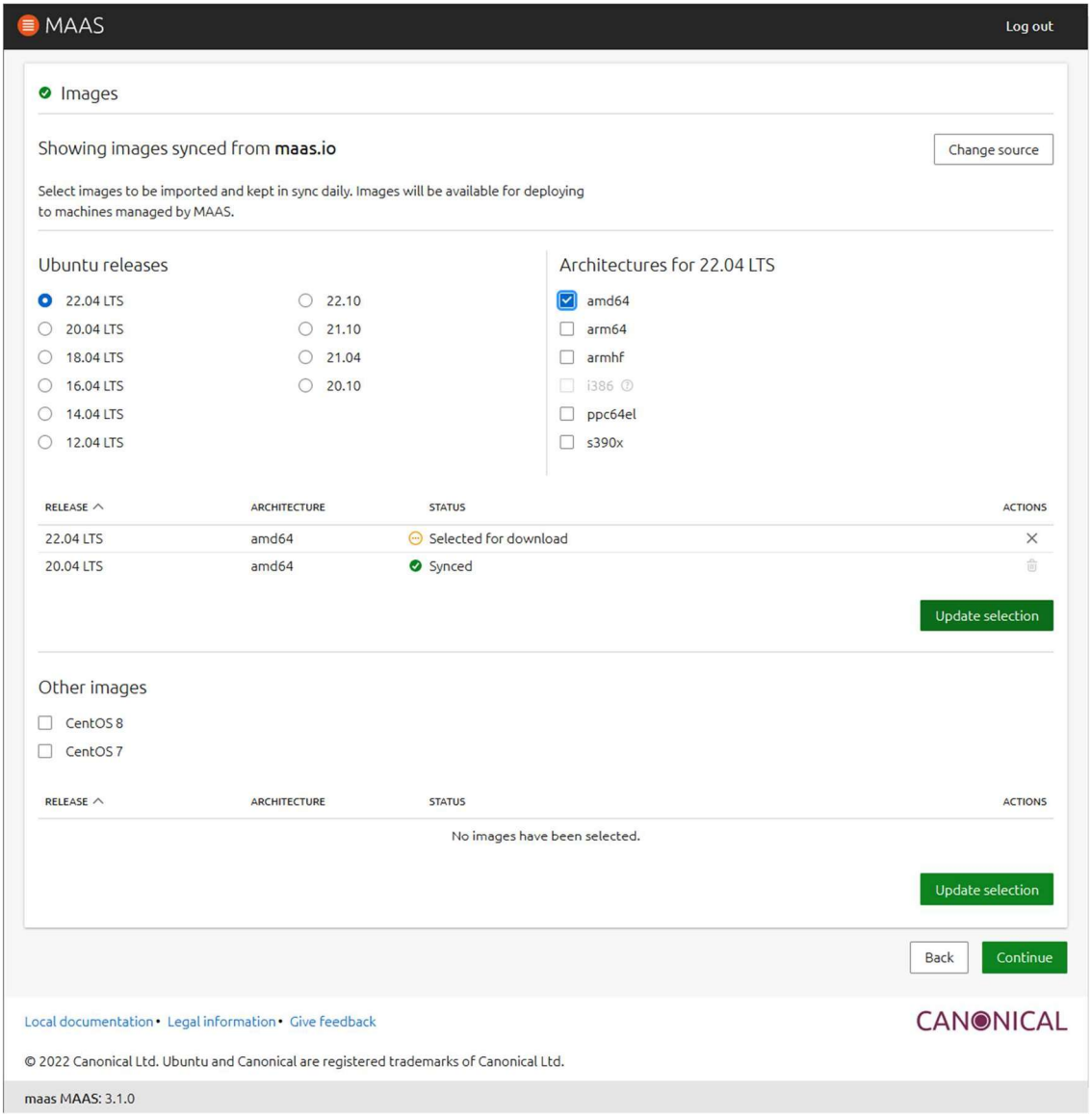


Figure 3 - Initial setup of MAAS – Images for booting

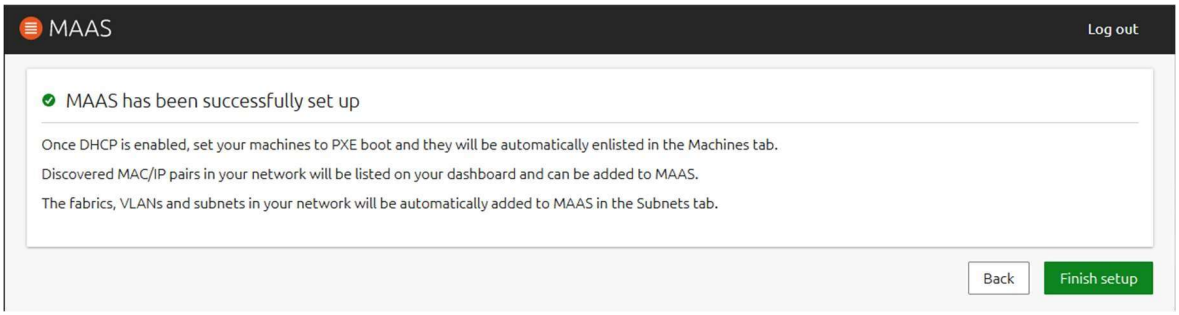


Figure 4 - Initial setup of MAAS – successful completion

In the next step, it is necessary to import the generated SSH key, which we generated during installation. As the source, we select the “Upload” type and paste the public SSH key, printed for example with the cat command, into the “Public key” field.

```
cat /home/student/.ssh/student.pub
```

1.2.5 Basic DHCP configuration in MAAS

One of the advantages that were mentioned in the features of MAAS is the DHCP server it provides. We must configure this DHCP server after installation and initialization. We will make all these changes using the MAAS CLI and therefore we first need to log in on the MAAS server using the MAAS user account and API key.

```
maas login student http://10.11.0.2:5240/MAAS - <  maas-api-key
```

Since we want the containers in OpenStack to receive IP addresses from the range 10.11.1.0 – 10.11.1.255, we must create two range reservations for static IP addresses and then a dynamic range for the remainder, which can be used for DHCP. A reserved range for static IPs (10.11.0.1 - 10.11.0.255), which we want to use ourselves, and a dynamic range for DHCP (10.11.2.0 - 10.11.255.254), and as a result it will assign the range 10.11.1.0 - 10.11.1.255 to the containers in OpenStack

```
maas student ipranges create type=dynamic \
    start_ip=10.11.2.0 end_ip=10.11.255.254

maas student ipranges create type=reserved \
    start_ip=10.11.0.1 end_ip=10.11.0.255 \
    comment='Static IPs'
```

We can then enable DHCP on the untagged local area network (VLAN) in “fabric-0”.

MAAS creates a “fabric” during installation for each discovered subnet; it is something like a connector between VLANs. If the MAAS server were connected to multiple VLANs, the fabric would enable the connection; it is something like a switch or a combination of switches using trunking.

```
maas student vlan update 0 untagged dhcp_on=True \
    primary_rack=maas
```

1.2.6 Upgrade MAAS

As with any software, the MAAS version can also be upgraded; version 3.1 was used during installation and stable version 3.2 has been released in the meantime. The upgrade can be performed with a single command when “region+rack” is used (this can be read from the installation in the init command):

```
sudo snap refresh --channel=3.2 maas
```

The version is specified after the "--channel" switch. After running it and entering the password, it is not necessary to reinitialize MAAS.

1.2.7 Deploying the Juju server

At this point, for the deployment we can already use the features of the MAAS server, which will make the installation of the Juju server easier. First of all, it will be necessary to add the server juju1 (VMWare), for which 4 vCPUs, 4 GB of RAM and 50 GB of disk space were allocated. A virtual or physical machine can be added to MAAS in two ways. The first way is to power on the server and, if it has booting via PXE enabled, then MAAS will assign it an IP address and subsequently provide an image for booting. After the image has loaded, MAAS collects basic information about the machine and displays it in the list in the web browser. It is now possible to modify variables such as the name, IP address and IPMI as needed. The second way is to fill in, in the MAAS web interface, the data required to add the machine and start bringing it into operation (commissioning).

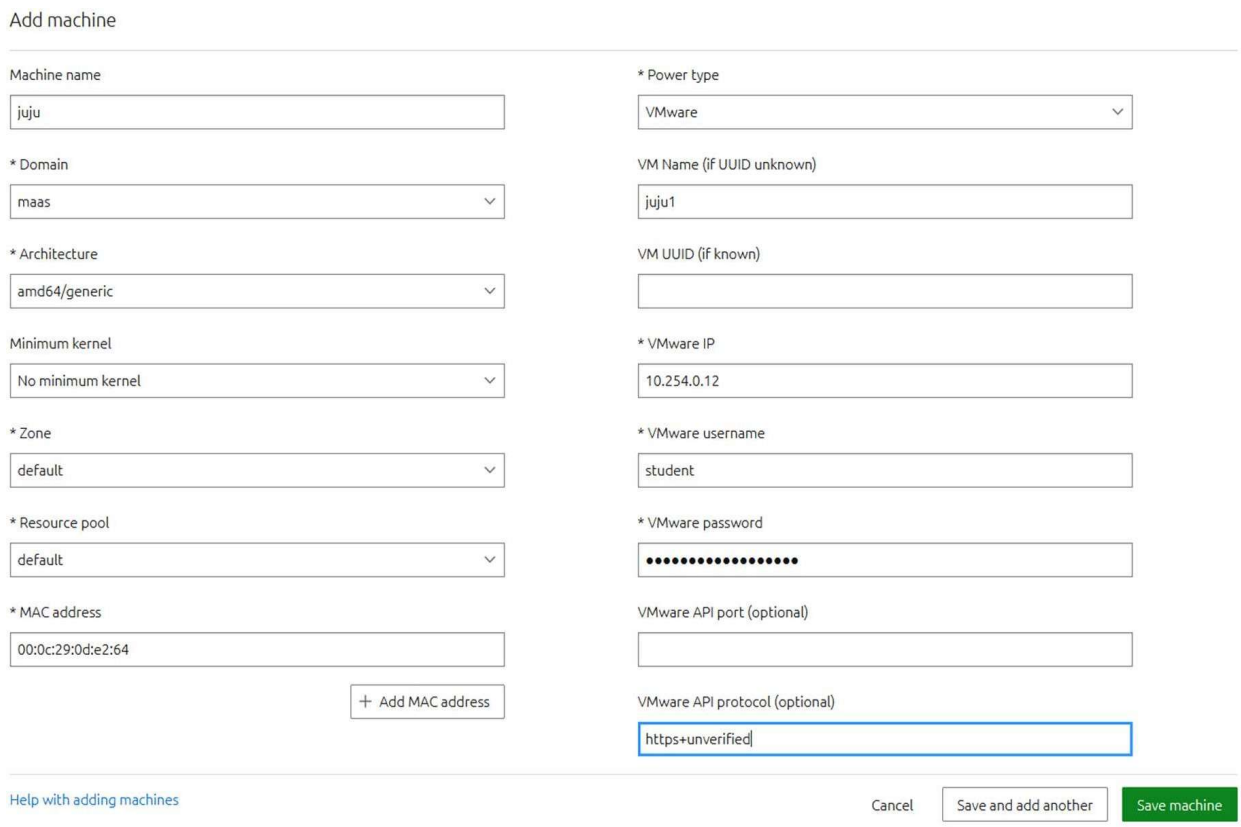


Figure 5 – Adding the Juju machine to MAAS via the web interface

After the machine has been successfully commissioned, it will still be necessary to make changes to the configuration for this machine, namely changing the IP address to the static address 10.11.0.3 and creating a label, a so-called tag. All these changes will be made in the web interface. After clicking on "juju.maas", first, in the "Network" tab, in our case for the interface ens160, we click on the arrow pointing down under the "Actions" section and select the "Edit physical"

option. A window with editing options then opens. To set a static IP address, it is only necessary to set “IP mode” to “Static assign”, enter the required IP address, namely 10.11.0.3, in the “IP address” field, and then simply save the interface using the green “Save interface” button.

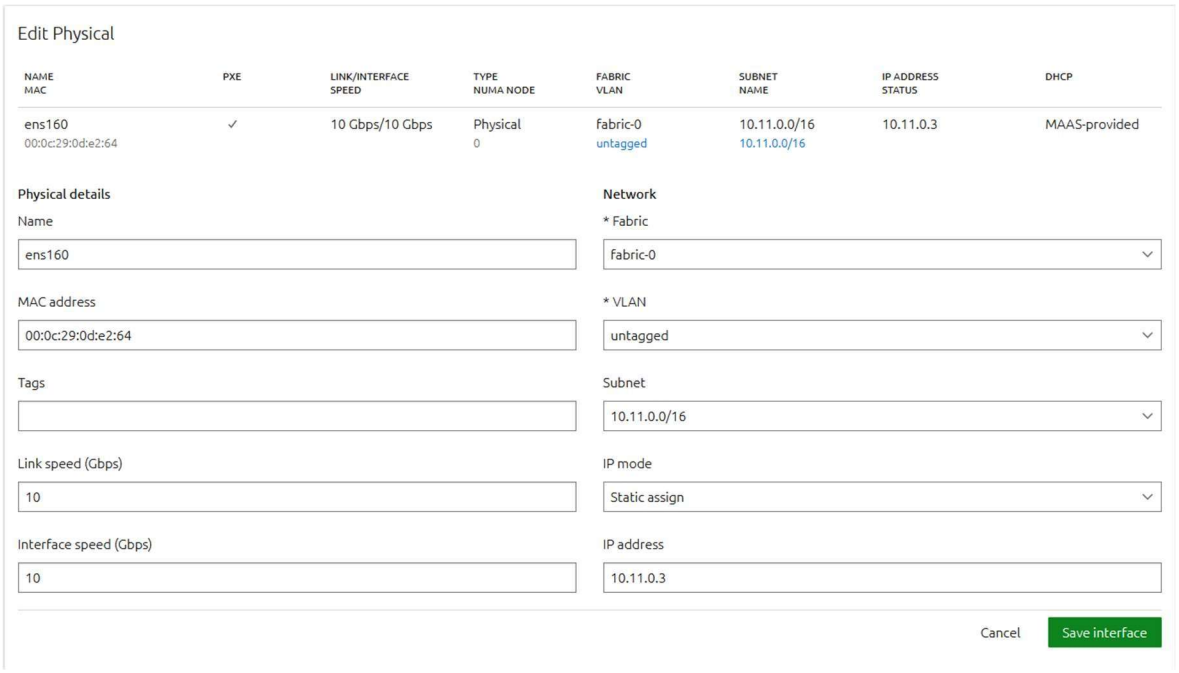


Figure 6 – Setting a static IP address in the MAAS web interface

We create the label in the “Configuration” tab in the Tags section, where we click on “Edit”. Then, using the search field that has appeared, we create the label “juju” by typing it into the field, and then the option “*Create tag* “juju”” appears, and clicking on this option displays a window for creating the label, where we can also write a comment or specify options for the Kernel. Then we simply finish using the button “Create and add to tag changes”. In the edited “Tags” section we can see that the juju label has been added, and we simply save using “Save”.



Figure 7 – Creating a label (tag) for the Juju server in the MAAS web interface

Next, we will continue in the CLI directly on the MAAS server. First of all, it is necessary to install juju.

```
Sudo snap install juju –classic
```

We create a YAML configuration file with the following variables. The IP address is that of the MAAS system, and after creating the file we add a cloud named “maas-one”.

```
Nano maas-cloud.yaml #creating and opening the file

#start of file maas-cloud.yaml
clouds:
  maas-one:
    type: maas
    auth-types: [oauth1]
    endpoint: http://10.11.0.2:5240/MAAS #end
of file maas-cloud.yaml

#adding a cloud with the configuration file maas-cloud.yaml named maas-one
juju add-cloud –client -f maas-cloud.yaml maas-one
```

We add the credentials from MAAS so that Juju can integrate with the newly added cloud. For the import, we again use a YAML configuration file named maas-creds.yaml (the name can be arbitrary), and we add them using the “juju add-credential” command.

```
#maas-oauth is the maas-api-key key from the installation, which we find in the
file with the same name, namely maas-api-key

nano maas-creds.yaml #creating and opening the file

#start of file maas-creds.yaml
```

```
credentials:
  maas-one:
    anyuser:
      auth-type: oauth1
      maas-oauth:
VcLNh2986dQcHD3fCt:t9gpGYeNjTKEFfjQgj:uLeqhLZZ4UlcghArpNWe9yRueGCpX2n4 #end
of file maas-creds.yaml

#adding credentials with the configuration file maas-creds.yaml to the cloud
maas-one juju add-credential -client -f maas-creds.yaml maas-one
```

We can now start the installation of the Juju controller on the server. It is important to specify "tags=juju" in the "– constraints" switch, which we assigned to the server during the configuration in MAAS, so that it is possible to designate the server on which it is to be installed. This is followed by the name of the cloud, i.e. "maas-one", and the name of the controller, "maas-controller".

```
Juju bootstrap -bootstrap-series=focal -constraints tags=juju maas-one maas-
controller
```

After entering this command, we should see the server being powered on in the web interface, and then we only need to wait until everything has been installed and completed. After a successful installation, we add the administrator account "student" for uniform access.

```
#creating the account, we will ignore the subsequent output juju
add-user student

#assigning superuser privileges to the account student juju grant
student superuser

#creating a password for the account student
juju change-user-password student

#verification in the output
juju users
```

```
student@maas:~$ juju users
Controller: maas-controller

Name      Display name  Access     Date created  Last connection
admin*    admin         superuser  2022-10-27    just now
student                 superuser  2022-10-27    2022-11-09

student@maas:~$
```

Figure 8 – Verification of the added student account for Juju

We can then display the address of the page location (URL), either with the credentials for the admin account or just the URL itself.

```
#output of URL + admin
credentials juju dashboard
```

```
#output of URL without credentials
juju dashboard -hide-credential
```

Since we assigned a static IP address to the machine for Juju, the address of the page location will be: https://10.11.0.3:17070/dashboard. We can then log in and see the web interface for Juju. Direct access to the server is then possible via SSH using an SSH key.

```
Ssh ubuntu@10.11.0.3 -i /home/student/.local/share/juju/ssh/juju_id_rsa
```

1.2.8 Adding servers intended for running OpenStack

As in the case of the Juju server, in order to be able to use the automation feature, we add the servers on which OpenStack will be installed to MAAS. These will be one controller node, the so-called Controller Node, and three compute nodes, the so-called Compute Nodes. The Controller Node is a virtual machine running on the same physical server as MAAS and Juju, with 32 vCPUs, 64GB of RAM and 100GB of disk space. It can be added in the same way as the Juju server. Subsequently, we also create a tag for it in the same way according to the table “Table 2 – Tagging (Tags) for machines in MAAS”. It will not yet be necessary to configure the IP address, because we will be making changes to the network settings at a later stage.

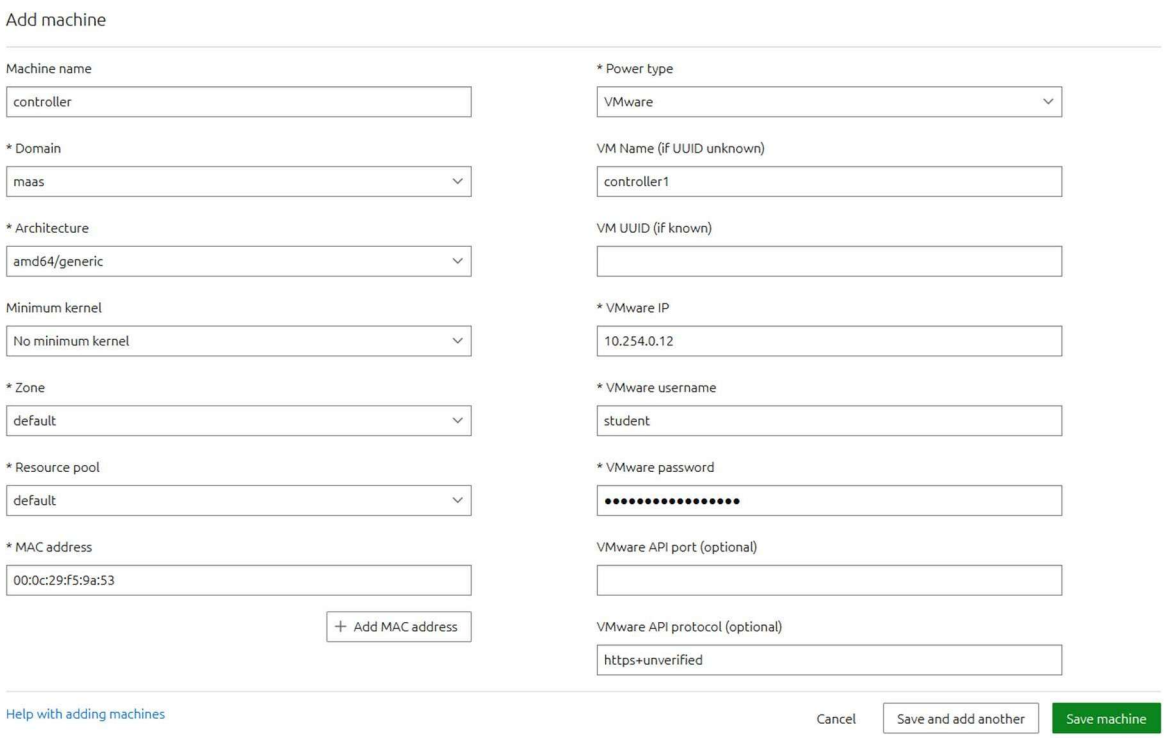


Figure 9 - Adding the Controller Node to MAAS via the web interface

For the Compute Nodes, we will use the second method. First, it will be necessary to set PXE booting as the priority on the individual servers, and then, after they are powered on, MAAS will assign these servers to itself and discover their information. In this case, it will not be necessary to set up IPMI, since MAAS will do this for us. When using this method of adding, it will be necessary to distinguish the individual servers, because MAAS assigns them a random name by default, and this will need to be modified, along with creating a tag. We will not configure IP addresses for these servers either. We change the server name by clicking on the server, then clicking on the name in the upper left corner, entering the required name and saving it using the “Save” button.

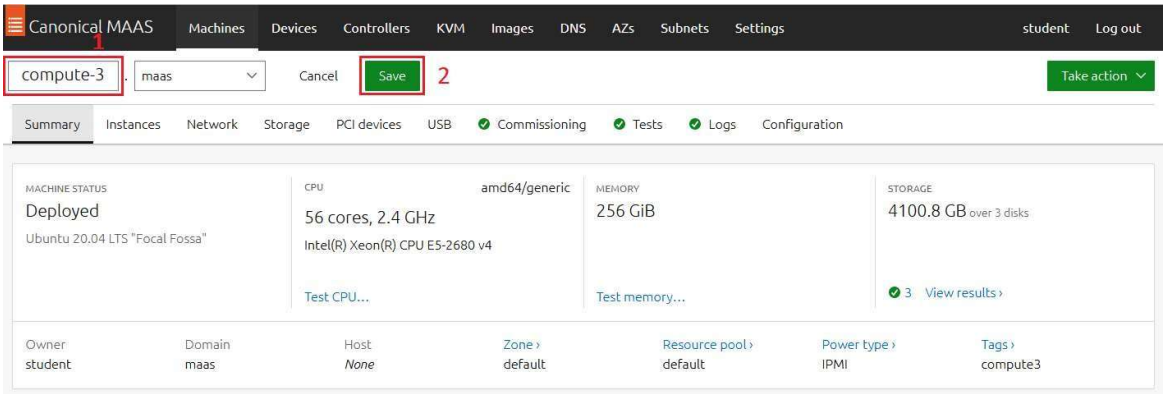


Figure 10 – Changing the server name in the MAAS web interface

1.2.9 Network configuration in the MAAS system

As the first change and preliminary preparation for the configuration of the interfaces on the servers, we must modify the subnets in the MAAS system. First, we create spaces, which serve as a grouping of subnets and allow them to communicate with each other. Next, we modify the space for “fabric-0” and create VLANs and subnets. We will make all these changes using the MAAS CLI, and therefore we first need to log in on the MAAS server using the MAAS user account and API key.

```
maas login student http://10.11.0.2:5240/MAAS - <  maas-api-key
```

We can then create the spaces. After each command, we receive information on whether the creation was successful, in the form of the output “Success”, together with information about the structure in question, such as its name and identifier. We will need two spaces named “openstack-mgmt” and “openstack-vxlan”, which we specify as a required parameter during creation.

```
maas student spaces create name="openstack-mgmt"
maas student spaces create name="openstack-vxlan"
```

Once the spaces have been created, we can create the VLANs and modify the untagged VLAN in “fabric-0”. First, we modify the untagged VLAN by assigning the space “openstack-mgmt” to it.

```
maas student vlan update 0 untagged space="openstack-mgmt"
```

The VLANs that we create next will all have the required parameters: the “fabric” identifier 0 and the vid according to the required value. We will also use an optional parameter for better identification, namely the name. To the VLAN with identifier 500 and the name “brVXLAN”, we also assign the second space created, “openstack-vxlan”.

```
maas student vlans create 0 name="brEX" vid=153
maas student vlans create 0 name="brEX2" vid=154
maas student vlans create 0 name="brVXLAN" vid=500 space="openstack-vxlan"
```

Then all that remains is to create the subnets with Classless Inter-Domain Routing (CIDR), a name, the “fabric” identifier and the VLAN identifier.

```
maas student subnets create cidr="158.193.153.0/24" name="EXT1" fabric=0 vid=153 maas student
subnets create cidr="158.193.154.0/24" name="EXT2" fabric=0 vid=154
maas student subnets create cidr="10.12.0.0/24" name="VXLAN" fabric=0 vid=500
```

1.2.10 Changing the network configuration for the Controller Node

For the needs of OpenStack, we need to ensure identical network interface configurations across all nodes. For the Controller Node, the configuration is slightly different, since it is a virtual server and has only one network interface. We rename this interface and create bridges with the corresponding VLANs and IP addresses. The configuration can be changed both via the CLI and via the web GUI. For efficiency and the possibility of automating this step, we will work in the CLI on the MAAS server, but the annexes will also include instructions for the GUI. In the first step, if we are not already logged in, we log in on the MAAS server under the student account.

```
maas login student http://10.11.0.2:5240/MAAS - <  maas-api-key
```

During this step and the next one, we will need the identifiers (IDs) of the servers that MAAS has assigned to them. We display them using the following command.

```
maas student machines read | jq '.[] | .hostname, .system_id'
```

Name	ID
juju	7ckx7w
controller	kgseyr
compute-1	8gsdnw
compute-2	nn7n7k
compute-3	bt8cge
compute-4	xkhy38
compute-7	anhqyc
compute-8	hhshym
compute-9	c6xpxx
compute-10	ncwf6s
compute-11	eyttr6

Table 4 – List of server identifiers in MAAS

Now that we know the IDs of the individual servers, we use the read command to also display the IDs of the network interfaces on the Controller Node.

```
maas student interfaces read pbpdab | jq '.[] | .id, .name'
```

From the output, we found that the only interface, named ens160, has ID 5. We use this ID and the server ID in the MAAS system to rename the network interface ens160 to “bond”.

```
maas student interface update kgseyr 5 name=bond
```

We now assign to this network interface the VLANs that we created in the previous step. Again, we will also need to find out the IDs of these VLANs that the MAAS system assigned to them in our Fabric-0.

```
maas student vlans read 0
```

VLAN	ID
untagged	5001
153(brEX)	5007
154(brEX2)	5009
500(brVXLAN)	5008

Table 5 – List of VLAN identifiers in MAAS

We can then create the individual VLANs on top of this interface using the server ID, the VLAN ID and the ID of this network interface. When a VLAN is created successfully, the output will also include its ID, which needs to be noted down. The ID of the first one is sufficient, since the numbering is sequential. That is, if the first VLAN added has ID 520, the next one will have ID 521.

```
maas student interfaces create-vlan kgseyr vlan=5007 parent=5 maas student
interfaces create-vlan kgseyr vlan=5009 parent=5
maas student interfaces create-vlan kgseyr vlan=5008 parent=5
```

We now create Open vSwitch (OVS) bridges on top of the individual VLANs and the network interface itself. In this step, it is also necessary to keep track of the new IDs for the last step.

```
maas student interfaces create-bridge kgseyr name=brEX vlan=5007 parent=520
bridge_type=ovs
maas student interfaces create-bridge kgseyr name=brEX2 vlan=5009 parent=521
bridge_type=ovs
maas student interfaces create-bridge kgseyr name=brVXLAN vlan=5008 parent=522
bridge_type=ovs
maas student interfaces create-bridge kgseyr name=brMGMT vlan=5001 parent=5
bridge_type=ovs
```

The last step is to add subnets for the newly created network interfaces; however, we will first need the IDs of the individual subnets.

```
maas student subnets read
```

Subnet name	ID
10.11.0.0/16	1
EXT1	3
EXT2	5
VXLAN	2

Table 6 – List of subnet identifiers in MAAS

To the interfaces for VLANs 153(brEX) and 154(brEX2), we assign only a subnet and set the mode to “LINK_UP”, which means that the interface is brought up only for the given subnet and no IP address is assigned.

```
maas student interface link-subnet kgseyr 524 mode=LINK_UP subnet=3
maas student interface link-subnet kgseyr 525 mode=LINK_UP subnet=5
```

For the management interface (untagged traffic) and VLAN 500 (brVXLAN), we use the static mode and set static IP addresses.

```
maas student interface link-subnet kgseyr 526 mode=STATIC subnet=2
ip_address=10.12.0.11
maas student interface link-subnet kgseyr 527 mode=STATIC subnet=1
ip_address=10.11.0.11
```

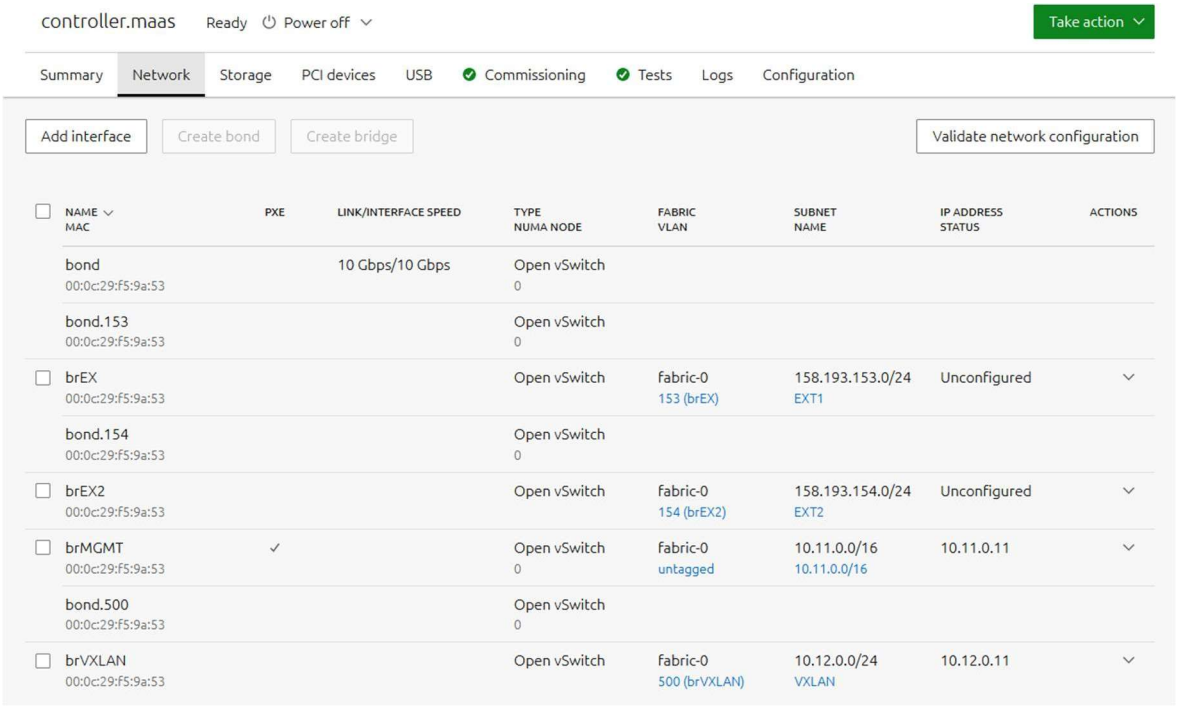


Figure 11 – Display of the entire network configuration for the Controller Node in the GUI

1.2.11 Changing the network configuration for the Compute Nodes

In the case of the Compute Nodes, we will proceed similarly to the previous step, apart from a few changes. Right in the first step, we will not merely rename the network interface; instead, we will combine two physical interfaces into one using a bond. In this case, we will also need the MAC (Media Access Control) address of one of the interfaces; we choose the first one in order.

```
maas student interfaces read xpxd4w | jq '.[] | .id, .mac_adress, .name'
```

From the output, we found that our two interfaces have IDs 38 (enp68s0f0) and 43 (enp68s0f1), and the MAC address that we will use, from interface enp68s0f0, is 00:1b:21:d4:62:3c. We use bond mode 802.3ad and set the hash policy to “layer2+3”.

```
maas student interfaces create-bond xpxd4w name=bond parents=38
mac_address=00:1b:21:d4:62:3c parents=43 bond_mode=802.3ad
bond_xmit_hash_policy=layer2+3
```

Not forgetting to note down the ID of the created bond, we add the VLANs.

```
maas student interfaces create-vlan xpxd4w vlan=5007 parent=529 maas student
interfaces create-vlan xpxd4w vlan=5009 parent=529
maas student interfaces create-vlan xpxd4w vlan=5008 parent=529
```

We then continue as in the case of the Controller Node: we create the bridges and assign the subnets.

```
maas student interfaces create-bridge xpxd4w name=brEX vlan=5007 parent=530
bridge_type=ovs
maas student interfaces create-bridge xpxd4w name=brEX2 vlan=5009 parent=531
bridge_type=ovs
maas student interfaces create-bridge xpxd4w name=brVXLAN vlan=5008 parent=532
bridge_type=ovs
maas student interfaces create-bridge xpxd4w name=brMGMT vlan=5001 parent=529
bridge_type=ovs
```

```
maas student interface link-subnet xpxd4w 533 mode=LINK_UP subnet=3
maas student interface link-subnet xpxd4w 534 mode=LINK_UP subnet=5
maas student interface link-subnet xpxd4w 535 mode=STATIC subnet=2
ip_address=10.12.0.25
maas student interface link-subnet xpxd4w 536 mode=STATIC subnet=1
ip_address=10.11.0.25
```

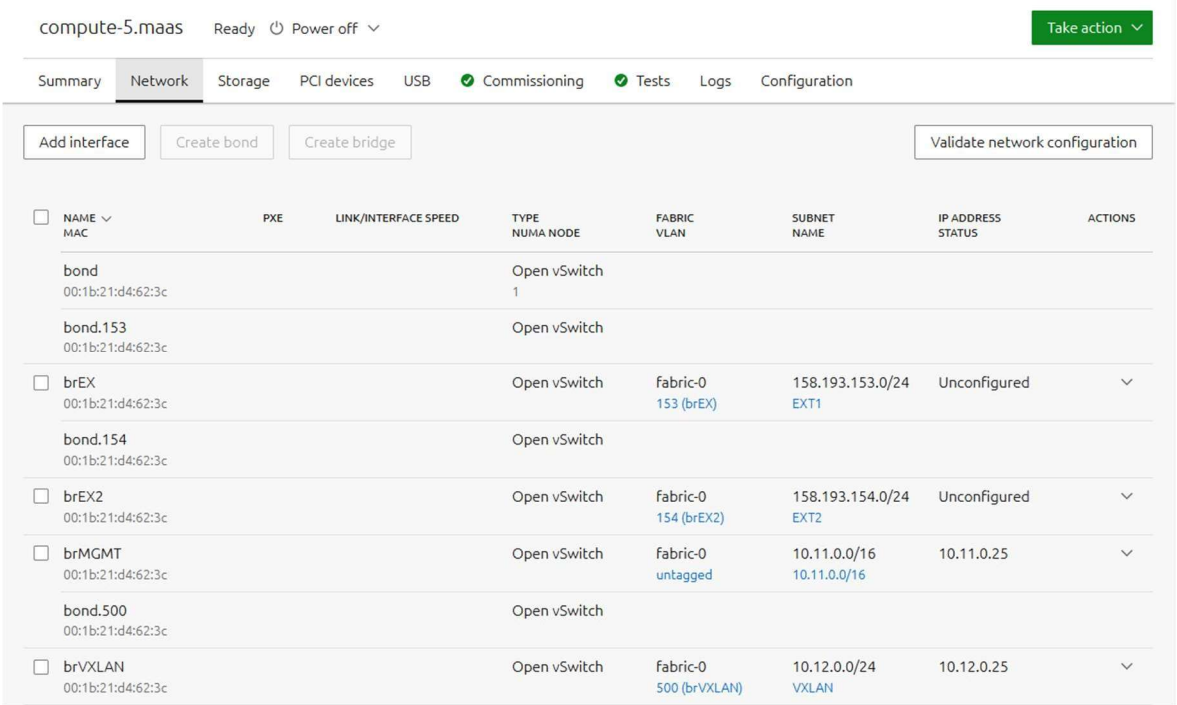


Figure 12 - Display of the entire network configuration for the Compute Node in the GUI

1.2.12 Changing the disk configuration for the Compute Nodes

After a server has been added to the MAAS system, it is not unambiguous which of the server's disks it will select as the system disk. It is therefore necessary to check this configuration and, if necessary, modify it. We will proceed similarly to the configuration of the network interfaces, again using the CLI. We use the already obtained IDs of the individual servers and display the disks they have. Specifically, we display only the disk sizes and their IDs.

```
maas student block-devices read xpxd4w | jq '.[] | .size, .id'
```

We know that the servers are fitted with disks of 300GB, 1.8TB and 2TB. We want to use precisely the smallest, 300GB disk as the system disk. In this case, this disk has ID 13. Setting the system disk consists of just one command, specifying the server ID and the disk ID.

```
maas student block-device set-boot-disk xpxd4w 13
```

The system disk setting can be checked using the GUI: after clicking on the server in question, go to the “Storage” tab and then look for the tick symbol in the “Boot” column.

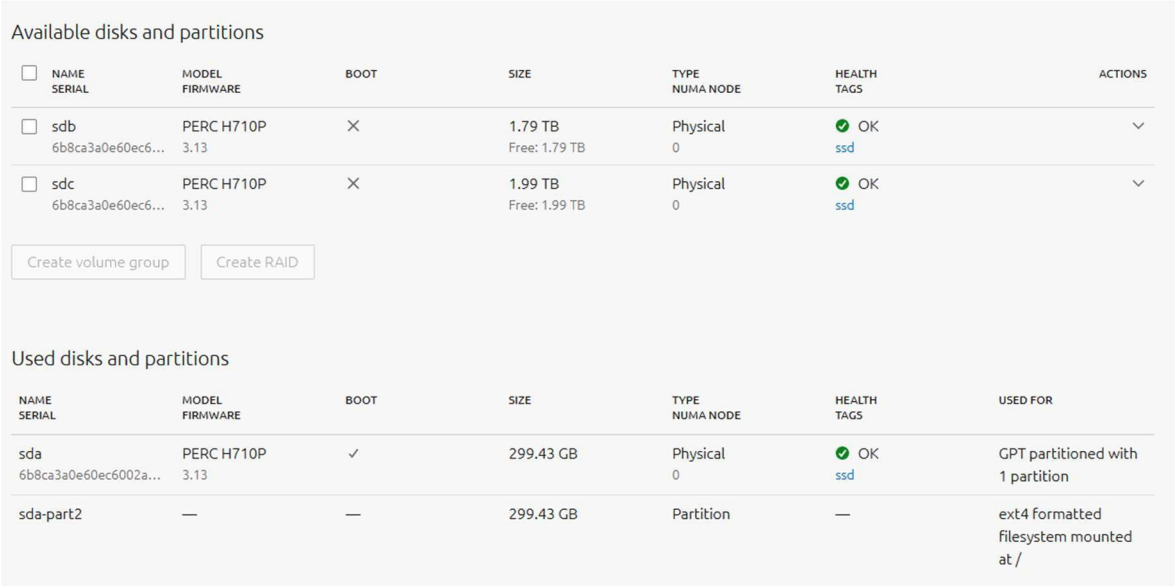


Figure 13 – Display of the system disk setting in the GUI

1.2.13 Setting the minimum kernel

During a trial deployment of OpenStack, a communication problem occurred between MAAS and one of the compute nodes, specifically compute-6. The problem lay in the communication between the server and MAAS after the operating system had been installed. When the installation was started, the commands from MAAS to power on the server and then upload the operating system completed successfully, but after that the server's communication failed; according to the error message, it had no network connectivity. Given our heterogeneous composition of servers, such a situation can also arise, where compute-6 is the same type of server as compute-3 and compute-4 but has a different network card. The solution consisted in setting a minimum kernel for the operating system that also contained the driver for the network card fitted in the compute-6 server. I would like to thank Ing. Martin Kontšek, PhD. for identifying this. The minimum operating system kernel can be set both via the CLI and in the web interface. Via the CLI, we log in to MAAS, if we are not already logged in, and use the command for setting the minimum kernel together with the system ID of the server to set the required value. In this case, we suppress the output that would otherwise be printed by using "> /dev/null", since it is too long. For verification, we use a command that displays only the item containing the value we are checking.

```
maas login student http://10.11.0.2:5240/MAAS - < /home/student/maas-api-key

maas student machine update 8h7kbn min_hwe_kernel=hwe-20.04 > /dev/null

maas student machine read 8h7kbn | jq '.hostname, .min_hwe_kernel'
```

```
student@maas:~$ maas student machine read 8h7kbn | jq '.hostname, .min_hwe_kernel'
"compute-6"
"hwe-20.04"
student@maas:~$
```

Figure 14 - Example of the output of the check value when changing the minimum kernel

When setting it via the web interface, we open “compute-6.maas” in the “Machines” tab. In the “Configuration” tab, we click the “Edit” button located in the “Machine configuration” section. We open the “Minimum kernel” drop-down menu and select the required value, in our case “focal (hwe-20.04), and click the “Save changes” button. After performing these actions, we can check that the “Minimum kernel” value was previously set to “—” and now contains our required value “hwe-20.04”.

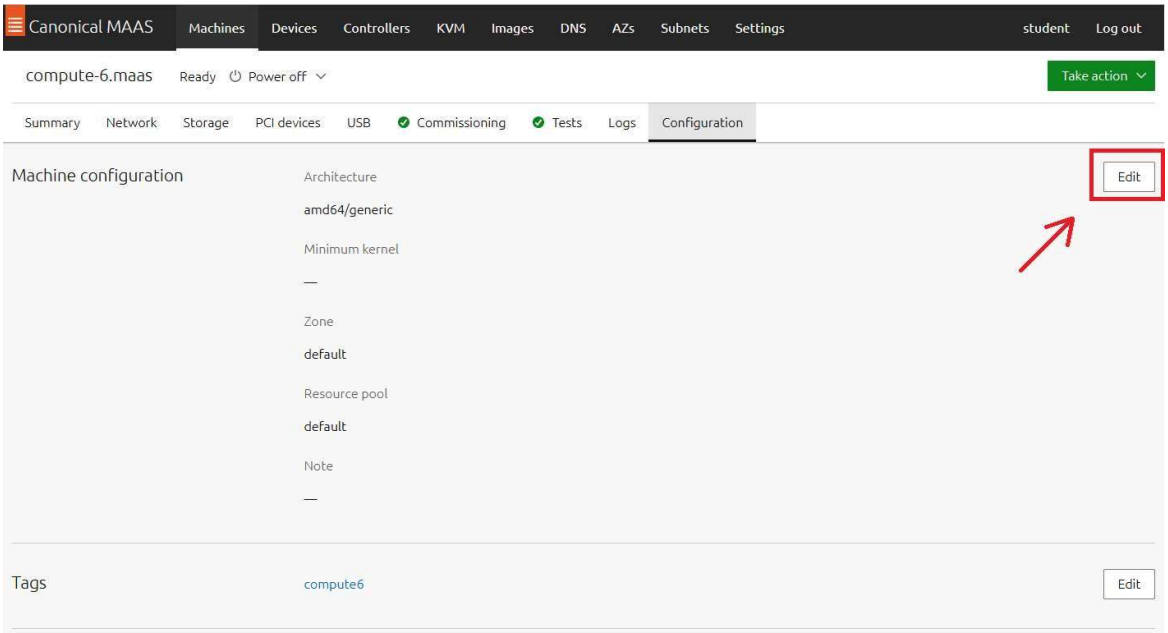


Figure 15 - Setting the minimum kernel in the web interface of MAAS

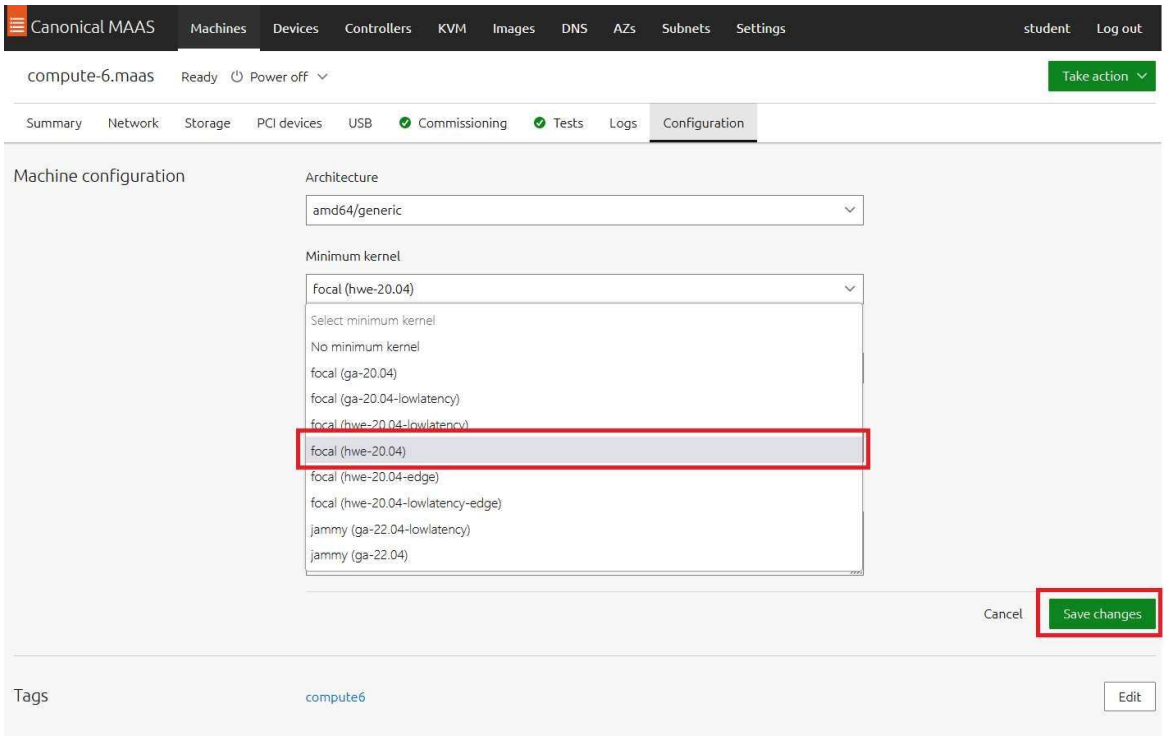


Figure 16 - Setting the minimum kernel in the web interface of MAAS

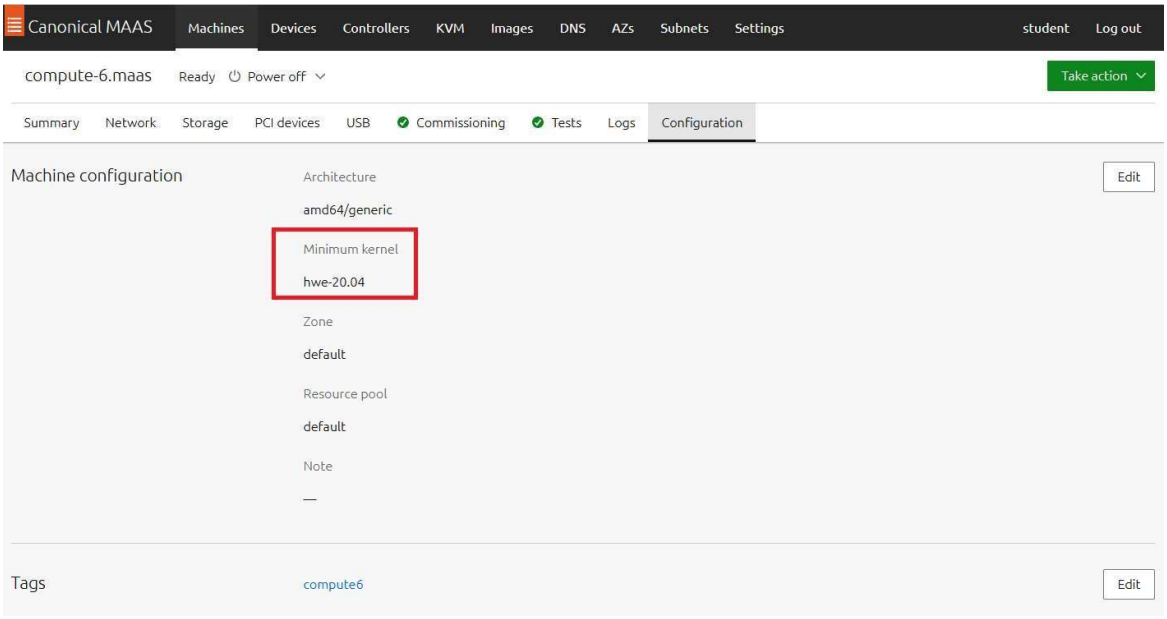


Figure 17 - Setting the minimum kernel in the web interface of MAAS

1.2.14 Installation package (bundle) for OpenStack

As already mentioned, for the installation of OpenStack itself we will use an installation package, a so-called bundle, which will be included in the annexes. Compared with manual installation, the main advantages are that it is not necessary to copy every command or create configuration files for the components, and

everything is neatly, clearly and transparently contained in one file. This means that after opening and reviewing such a file, the reader will find out which components will be installed, which relations will be created or which network interfaces will be used, and more. The bundle consists of five parts: the header, variables, machines (servers), relations and applications. The header specifies the name and series; in our case, the name is “openstack-base” and we will use the “focal” series for the Linux Ubuntu operating systems:

```
name: openstack-base
series: focal
```

Next is the variables section, in which the “focal” series is again specified first, used to determine the operating system on the servers together with the OpenStack version. Data-port specifies which network bridges will be mapped to which network interfaces. In our case, this section will also contain the variables for the Ceph Osd component, namely:

- hard disks that will be used for this component
- the number of servers on which this component will be installed
- the number of servers on which the monitoring component will be installed

Apart from the header section, each section has its own name, and the information in it is indented with a tab or, in the case of the relations section, with a hyphen.

```
variables:
  openstack-origin: &openstack-origin cloud:focal-yoga
  data-port: &data-port brEX:bond.153 brEX2:bond.154 brVXLAN:bond.500
  osd-devices: &osd-devices /dev/sdb /dev/sdc
  expected-osd-count: &expected-osd-count 5
  expected-mon-count: &expected-mon-count 5
```

In the machines (servers) section, it is possible to number the machines according to one's own convention and, if necessary, assign them a tag from the MAAS system.

```
machines:
  '11':
    constraints: tags=controller
  '23':
    constraints: tags=compute3
  '24':
    constraints: tags=compute4
```

In relations, hyphens are used for differentiation, namely as follows:

- the first component with which a relation is to be created is marked with two hyphens and a space
- the second component, with which a relation is to be created, is first indented with a tab and only then followed by a single hyphen

```
relations:
- - nova-compute:amqp
  - rabbitmq-server:amqp
- - nova-cloud-controller:identity-service
  - keystone:identity-service
- - glance:identity-service
  - keystone:identity-service
```

In the last section, applications (components), this varies between applications, because not all of them have the same options, but they share the same basis. Specifically, we will examine the Nova component. First come the bindings, which are an optional element and specify which data from the given component will use the given space from MAAS. Annotations specify a point by X and Y coordinates where the component's icon will be placed in the Juju web graphical interface representing the given model. Charm and channel determine which version of the given component will be installed; a complete list of versions and channels can be found at charmhub.io, e.g. https://charmhub.io/nova-compute. The number of units (num_units) indicates how many times the component will be installed. The options differ from component to component; a list and the meanings of the individual options can also be found on the charmhub.io website. Last is again an optional setting, where we can specify on which machines the component will be installed; otherwise, Juju does this automatically. For components that are installed in containers, it is necessary to specify “lxd:” before the machine number.

```
applications:
  nova-compute:
    bindings:
      "": openstack-mgmt
      internal: openstack-mgmt
      migration: openstack-mgmt
    annotations:
      gui-x: '190'
      gui-y: '890'
    charm: ch:nova-compute
    channel: yoga/stable
    num_units: 5
    options:
      config-flags: default_ephemeral_format=ext4
      enable-live-migration: true
      enable-resize: true
      migration-auth-type: ssh
      openstack-origin: *openstack-origin
    to:
    - '23'
    - '24'
    - '25'
    - '26'
```

```
    - '27'
  mysql-innodb-cluster:
    bindings:
      "": openstack-mgmt
      shared-db: openstack-mgmt
    annotations:
      gui-x: '535'
      gui-y: '1550'
    charm: ch:mysql-innodb-cluster
    channel: 8.0/stable
    num_units: 3
    to:
    - lxd:11
    - lxd:11
    - lxd:11
```

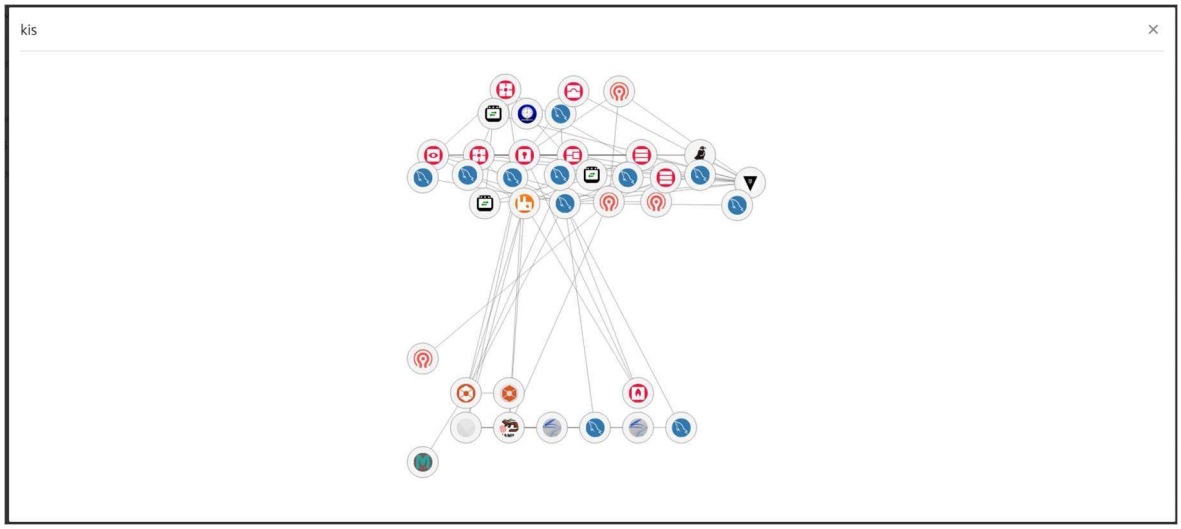


Figure 18 - Example of the display of the kis model in the Juju web interface using the icons of components and the connections (relations) between them

1.2.15 Installation of OpenStack Yoga

At this point, we have everything necessary prepared and configured to start the installation of the OpenStack Yoga platform using the bundle. We begin by uploading this file in YAML format to the MAAS server. We will continue in the CLI on the MAAS server with the command to create a model named “kis”, owned by the account “student”. Then all that remains is to start the installation using the “deploy” command and the path to the bundle.

```
juju add-model kis

juju deploy ./bundleKIS.yaml
```

After some time (15-20 minutes), we can check the installation status using the command:

```
juju status
```

If the component named Vault is in the “Idle” state and its status is “Vault needs to be initialized”, it will be necessary to perform the last step, namely to unseal it. Before we begin, we must also install the software required for this.

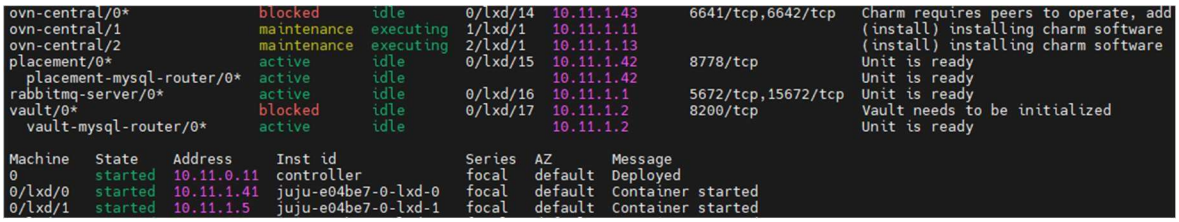


Figure 19 – Sealed and uninitialized vault (Vault) during installation

```
sudo snap install vault
```

We export the IP address of the container in which the Vault module is installed (juju status).

```
export VAULT_ADDR=http://10.11.1.30:8200
```

We start the initialization using the “init” command and specify the number of keys to be printed and the number of keys that will be required for unsealing. From the output, it is best to keep all the keys, or at least the number needed for unsealing. We also save the “Initial Root Token”.

```
vault operator init -key-shares=5 -key-threshold=3
```

Output:

```
Unseal Key 1: 9xCIrnapVFY5QKWPx6UcrZAL0f2I7GI2cZ2y2/ydZA+u Unseal Key
2: Ahs6PAEwSosNki7W0+idDCaD4rnUy+FmzGQsN/0yopLB
Unseal Key 3: Ac0hWoHGqxRev9SWIvzrNIgUxtng3YHbqFWQTJJWr2CP
Unseal Key 4: mGVNmEW7MvBJAqP4VJ8wv452Ls949TI9V+/6gMXI15IB
Unseal Key 5: yALNYeOBIlNrg77Ws67xKznmA9myIxeGJ/aNXQExpzGF

Initial Root Token: s.Un84Rz15EjUmfb9uk3uQRhqb

Vault initialized with 5 key shares and a key threshold of 3. Please securely distribute the key
shares printed above. When the Vault is re-sealed, restarted, or stopped, you must supply at least 3
of these keys to unseal it before it can start servicing requests.

Vault does not store the generated master key. Without at least 3 key to

reconstruct the master key, Vault will remain permanently sealed!

It is possible to generate new unseal keys, provided you have a quorum of
existing unseal keys shares. See "vault operator rekey" for more information.
```

We take the number of keys from “-key-threshold=” and use them in the “vault operator unseal” commands.

```
vault operator unseal 9xCIrnapVFY5QKWPx6UcrZAL0f2l7Gl2cZ2y2/ydZA+u vault
operator unseal Ahs6PAEwSosNki7W0+idDCaD4rnUy+FmzGQsN/0yopLB
vault operator unseal Ac0hWoHGqxRev9SWlvzrNlgUxtng3YHbqFWQTJJWr2CP
```

In the export VAULT_TOKEN command, we use the Initial Root Token from the previous output.

```
export VAULT_TOKEN=s.Un84Rz15EjUmfb9uk3uQRhqb
vault token create -ttl=10m
```

Output:

```
Key                  Value
---                  -----
token                s.KFrGIuaSB9JuioHaCttgFRAN
token_accessor       gkG8YIMYg6j2DYjWgAxCADnC
token_duration       10m
token_renewable      true
token_policies       ["root"]
identity_policies    []
policies             ["root"]
root@openstackmaas:~#
```

We use the token in the following command:

```
 juju run-action --wait vault/leader authorize-charm
token=s.KFrGIuaSB9JuioHaCttgFRAN
```

The last command for generating the root certificates:

```
juju run-action --wait vault/leader generate-root-ca
```

After the certificates have been generated, it is still necessary to wait for some time until all components are in the “active” and “idle” states. We can then display the password for the administrator account using the command:

```
juju run --unit keystone/leader leader-get admin_passwd
```

The web interface is accessed at the IP address of the Openstack-dashboard component: https://<IP_ADDRESS>/horizon. The login credentials are:

User Name: admin
Password: <PASSWORD>
Domain: admin_domain

To allow access from anywhere to the console of virtual instances in OpenStack, since private IP addresses are used everywhere, we configure a proxy IP address.

```
juju config nova-cloud-controller console-proxy-ip=158.193.152.130
```

And to set the default domain for logging in, so that it does not have to be typed every time.

```
juju config openstack-dashboard default-domain="admin_domain"
```

1.2.16 Configuration of the MAAS server for automatic authentication for the OpenStack CLI client

After a successful installation of OpenStack, this platform can be configured either via the web graphical interface provided by the Horizon component, or using the OpenStack client in the form of a CLI. For some tasks the web graphical interface is better, for others the CLI, for example creating external networks, which will be less time-consuming. We install the OpenStack client on the MAAS server using a simple command:

```
sudo snap install openstackclients
```

Next, authentication needs to be performed using exports, and so that this authentication does not have to be done manually every time, we add code for automatic authentication to the ".bashrc" file. This file is executed at every login to the server. In the first part, the certificates are copied to the correct locations; then the IP address of the Keystone component and the administrator password are stored in the variables _keystone_ip and _password using Juju commands. The required variables are set using exports, and as verification a table of the OpenStack endpoints is printed [10].

```
#Login OpenStack CLI Client
JUJU_MODEL=kis
if [ -d ~/snap/openstackclients/common/ ]; then
  # When using the openstackclients confined snap the certificate has to be
  # placed in a location reachable by the clients in the snap.
  _root_ca=~/snap/openstackclients/common/${JUJU_MODEL-""}root-ca.crt
else
  _root_ca=/tmp/${JUJU_MODEL-""}root-ca.crt
fi
```

```
juju run $_juju_model_arg --unit vault/leader 'leader-get root-ca' | tee
$_root_ca >/dev/null 2>&1

_keystone_ip=$(juju run -m kis --unit keystone/leader -- 'network-get --bind-
address public')
_password=$(juju run -m kis --unit keystone/leader 'leader-get admin_passwd')

export OS_REGION_NAME=RegionOne
export OS_AUTH_VERSION=3
export OS_CACERT=/home/student/snap/openstackclients/common/root-ca.crt
export OS_AUTH_URL=https://${_keystone_ip}:5000/v3
export OS_PROJECT_DOMAIN_NAME=admin_domain
export OS_AUTH_PROTOCOL=https
export OS_USERNAME=admin
export OS_AUTH_TYPE=password
export OS_USER_DOMAIN_NAME=admin_domain
export OS_PROJECT_NAME=admin
export OS_PASSWORD=${_password}
export OS_IDENTITY_API_VERSION=3

echo "--- Testing exports to initialize OpenStack CLI Client ---"
echo "--- Printing endpoints of OpenStack ---"
openstack endpoint list --interface admin
#End of login OpenStack CLI Client
```

```
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-53-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Fri Dec  2 08:50:26 AM UTC 2022

  System load:  0.0029296875      Processes:               287
  Usage of /:   67.6% of 23.45GB  Users logged in:         0
  Memory usage: 27%               IPv4 address for ens160: 10.11.0.2
  Swap usage:   0%

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

18 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


Last login: Fri Dec  2 08:31:27 2022 from 10.11.0.1
--- Testing exports to initialize OpenStack CLI Client ---
--- Printing endpoints of OpenStack ---
+----------------------------------+-----------+--------------+--------------+---------+-----------+-----------------------------------------+
| ID                               | Region    | Service Name | Service Type | Enabled | Interface | URL                                     |
+----------------------------------+-----------+--------------+--------------+---------+-----------+-----------------------------------------+
| 02baecf0cd314b4aa1bbbfce4efc1790 | RegionOne | placement    | placement    | True    | admin     | https://10.11.1.30:8778                 |
| 339f4b046c554dd3ad6c1591837fb02b | RegionOne | keystone     | identity     | True    | admin     | https://10.11.1.20:35357/v3             |
| 3d59654f31be4fdfa71d5263c4bb8817 | RegionOne | neutron      | network      | True    | admin     | https://10.11.1.22:9696                 |
| 3ed243be17df4e64a49f43e0092db1ea | RegionOne | glance       | image        | True    | admin     | https://10.11.1.33:9292                 |
| 86a9db4b118b4460855e828c14a3bd26 | RegionOne | swift        | object-store | True    | admin     | https://10.11.1.26:443/swift            |
| 9a9c237931154479a3bc1badc0524e79 | RegionOne | nova         | compute      | True    | admin     | https://10.11.1.31:8774/v2.1            |
| be42141354344daaa28f9d537b37f3d8 | RegionOne | cinderv3     | volumev3     | True    | admin     | https://10.11.1.27:8776/v3/$(tenant_id)s |
| eaa421d7ce0349d8b6c43cc6b499b6fb | RegionOne | s3           | s3           | True    | admin     | https://10.11.1.26:443/                 |
+----------------------------------+-----------+--------------+--------------+---------+-----------+-----------------------------------------+
student@maas:~$
```

Figure 20 – Demonstration of authentication after logging in to the MAAS server

1.2.17 Creating external networks for OpenStack

The OpenStack platform uses a so-called external network for communication between virtual instances and outside networks. In our case, we currently have two IP address ranges available that we can use to allocate public IP addresses to virtual instances. Such an external network can be created using the OpenStack web interface, or, as we chose, using the CLI OpenStack client. Since we have two address ranges, namely 158.193.153.0/24 and 158.193.154.0/24, we need to create two external networks. We create the first external network named “ext-net-153”, and as “provider-network- type” we choose the “flat” option so that packets are sent unmodified. As “provider-physical-network” we specify “physnet1”, which we specified

in the installation package as “flat- network-providers” and subsequently mapped to the brEX bridge, which in turn is mapped to the physical interface bond.153. For this external network “ext-net-153”, we also create a subnet allocating IP addresses in the range 158.193.153.2 – 158.193.153.254, with the department's DNS server with IP address 158.193.152.4, the default gateway 158.193.153.1 and the name EXT1.

```
openstack network create --external --share \
   --provider-network-type flat --provider-physical-network physnet1 \
   ext-net-153

openstack subnet create --network ext-net-153 \
  --allocation-pool start=158.193.153.2,end=158.193.153.254 \
  --dns-nameserver 158.193.152.4 --gateway 158.193.153.1 \
  --subnet-range 158.193.153.0/24 EXT1
```

For the second external network, “ext-net-154”, we will proceed in the same way, except for the change of name, and for “provider-physical-network” we will specify “physnet2”, which is further mapped in the installation bundle to the bridge brEX2 and the physical interface bond.154. When creating the subnet, we will of course also change the IP address range in the third octet to 154 and name it EXT2.

```
openstack network create --external --share \
   --provider-network-type flat --provider-physical-network physnet2 \
   ext-net-154

openstack subnet create --network ext-net-154 \
  --allocation-pool start=158.193.154.2,end=158.193.154.254 \
  --dns-nameserver 158.193.152.4 --gateway 158.193.154.1 \
  --subnet-range 158.193.154.0/24 EXT2
```

For verification, we can use commands to list networks and subnets in OpenStack.

```
openstack network list
openstack subnet list
```

```
student@maas:~$ openstack network list
+--------------------------------------+-------------+--------------------------------------+
| ID                                   | Name        | Subnets                              |
+--------------------------------------+-------------+--------------------------------------+
| 3e7591c8-c85f-4447-a7ba-0deb2cbe8860 | ext-net-153 | 86e73eb3-2743-48b7-9f84-a75e2634b009 |
| c8bed010-e42e-4f98-9651-6aa7562aff85 | ext-net-154 | 87d41c69-9eb5-4ce6-af29-7791eeda550c |
+--------------------------------------+-------------+--------------------------------------+
student@maas:~$ openstack subnet list
+--------------------------------------+------+--------------------------------------+------------------+
| ID                                   | Name | Network                              | Subnet           |
+--------------------------------------+------+--------------------------------------+------------------+
| 86e73eb3-2743-48b7-9f84-a75e2634b009 | EXT1 | 3e7591c8-c85f-4447-a7ba-0deb2cbe8860 | 158.193.153.0/24 |
| 87d41c69-9eb5-4ce6-af29-7791eeda550c | EXT2 | c8bed010-e42e-4f98-9651-6aa7562aff85 | 158.193.154.0/24 |
+--------------------------------------+------+--------------------------------------+------------------+
student@maas:~$
```

Figure 21 – List of networks and subnets in OpenStack on the MAAS server

1.2.18 Deleting component IP addresses in MAAS

The MAAS system also provides network discovery functionality, i.e. at certain intervals it can map the subnets to which it is connected. This functionality can be disabled globally or specifically for individual subnets.

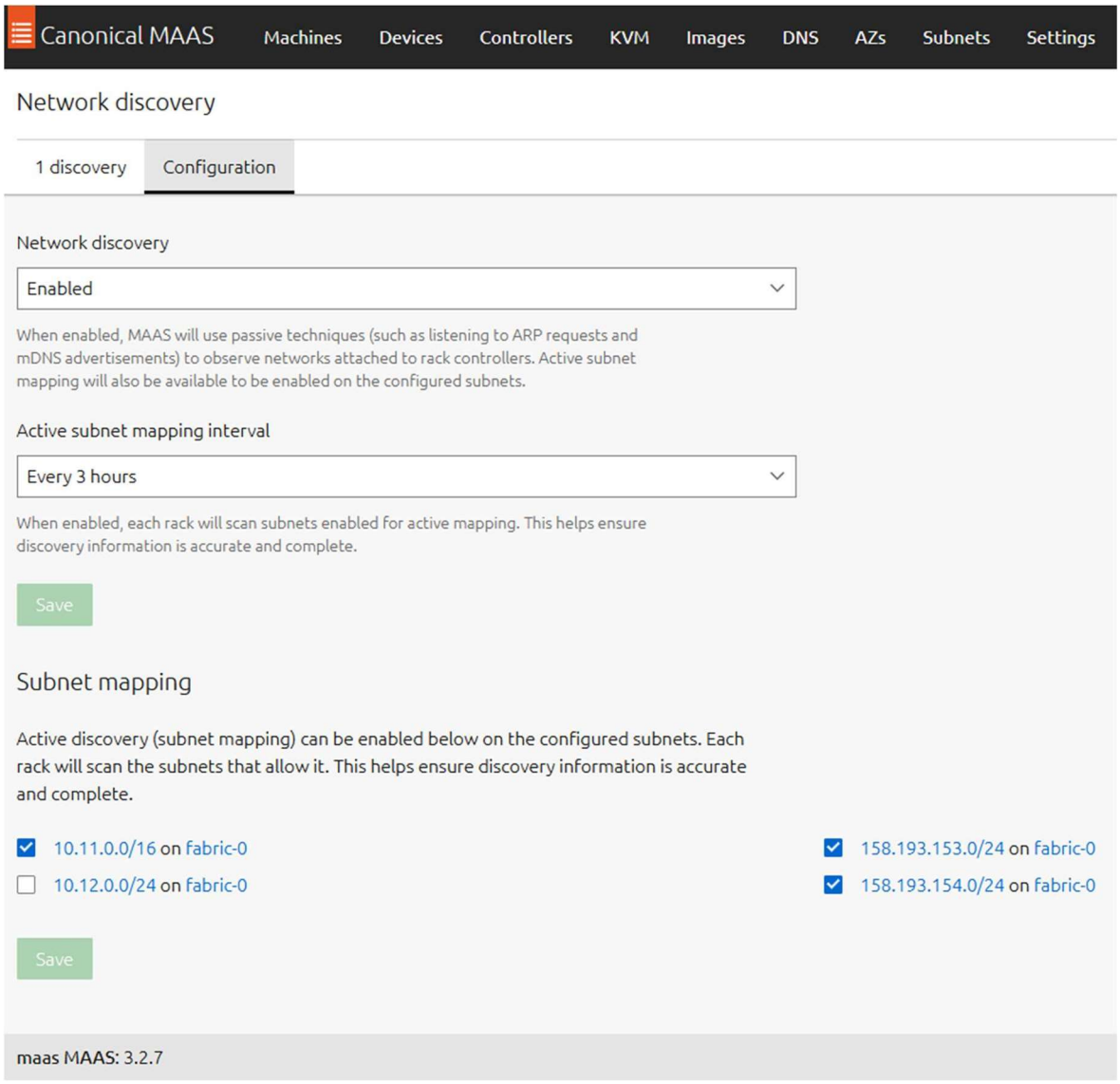


Figure 22 – Example of network mapping settings in the MAAS web interface

If we need to delete the model and use the same range of IP addresses that we allocated for the OpenStack components, it is necessary to delete the discovered IP addresses in this list. This action can be performed both in the GUI and using the CLI. In the web interface, we first click on the name “Canonical MAAS” in the upper left corner, after which the “Network discovery” menu is displayed, where in the upper right part we click on “Clear all discoveries”, i.e. clearing all discovered devices.

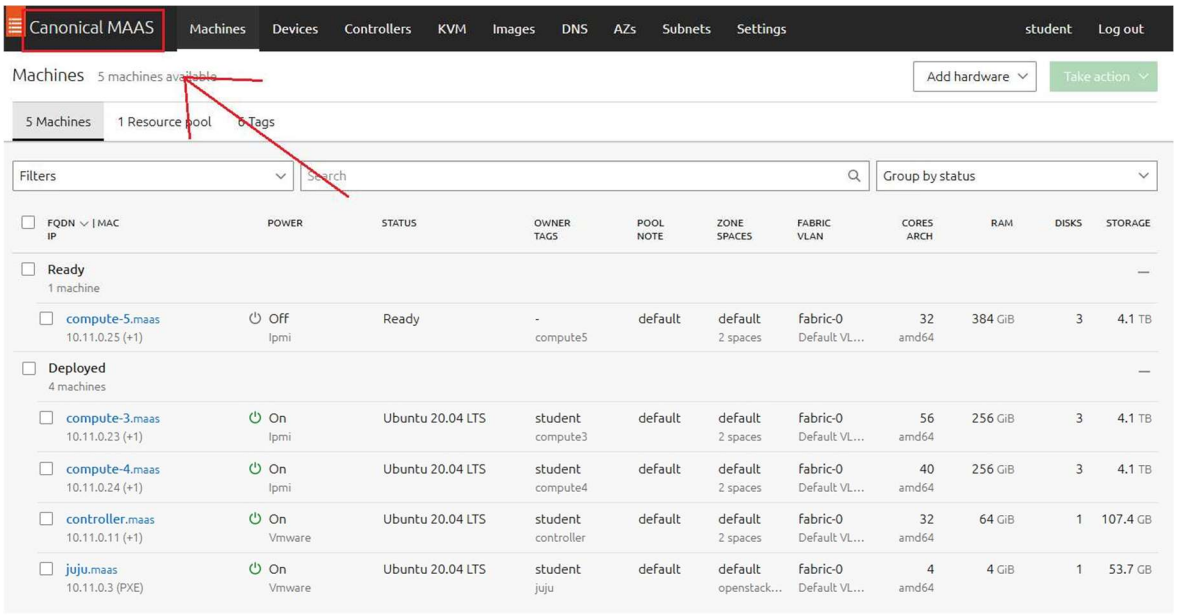


Figure 23 – Navigation in the MAAS web interface to the Network discovery menu

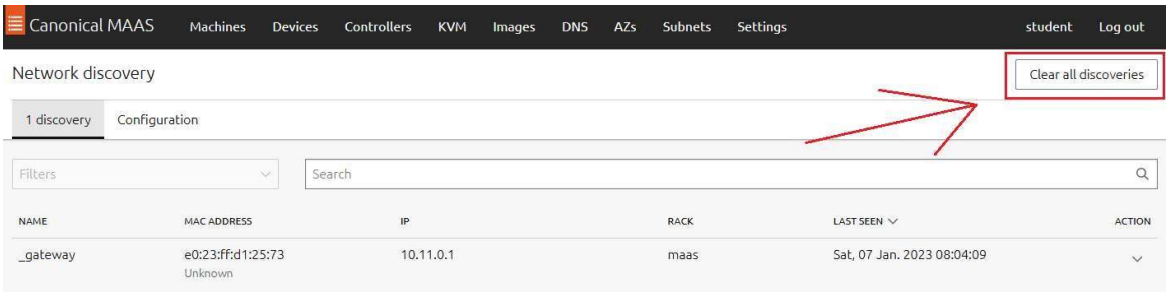


Figure 24 – Clearing the list of discovered devices in Network discovery

For the CLI, it again applies that if we are not logged in to MAAS, we first log in to MAAS under the user account student.

```
maas login student http://10.11.0.2:5240/MAAS - <  maas-api-key
```

Then, using a single command, we likewise delete the entire list of discovered devices. The “all” option can also be replaced with “mdns”, which deletes all mDNS discoveries, or “neighbours”, which deletes all entries from neighbours.

```
maas student discoveries clear all=True
```

2 Verification

Verification was carried out by creating elements and thereby also verifying their functionality:

- creation of instances
- console in the web interface
- connecting to the console using a terminal
- uploading an image for booting
- creating flavors (resources for instances)
- creation of networks
- creation and uploading of SSH keys

Furthermore, after installation, all components were in the active state without error messages, as shown by the “juju status” output.

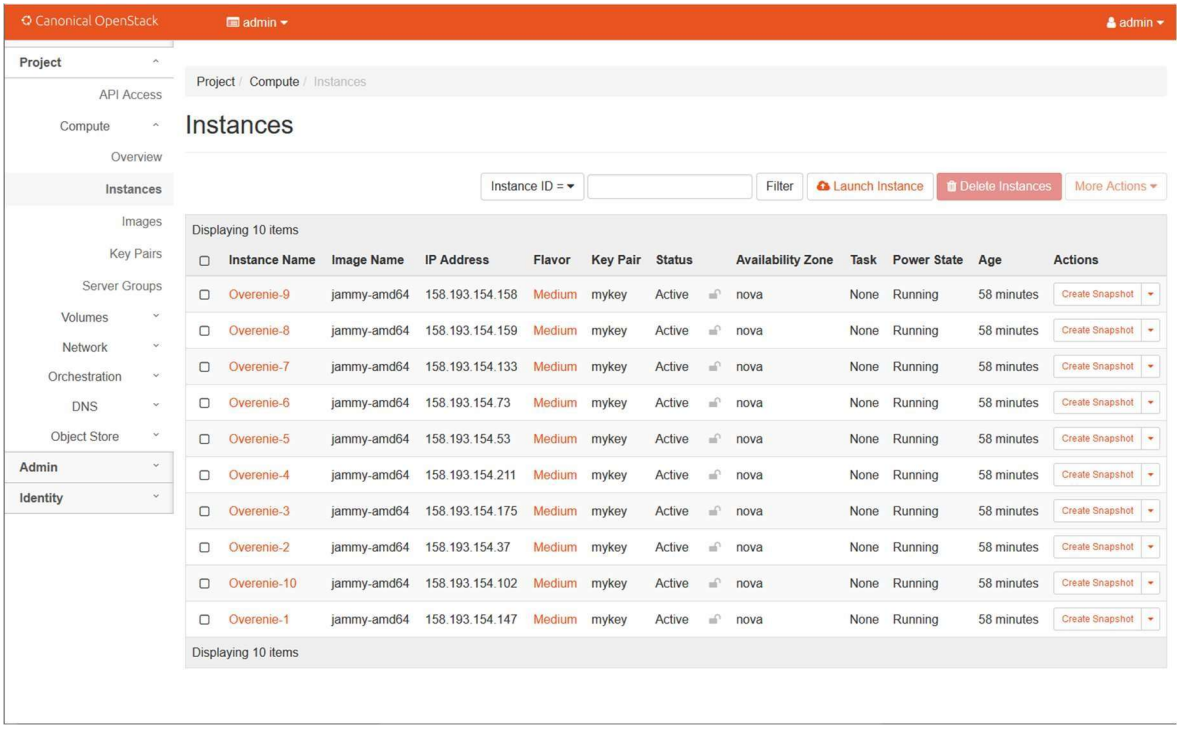


Figure 29 - Virtual instances for functionality verification

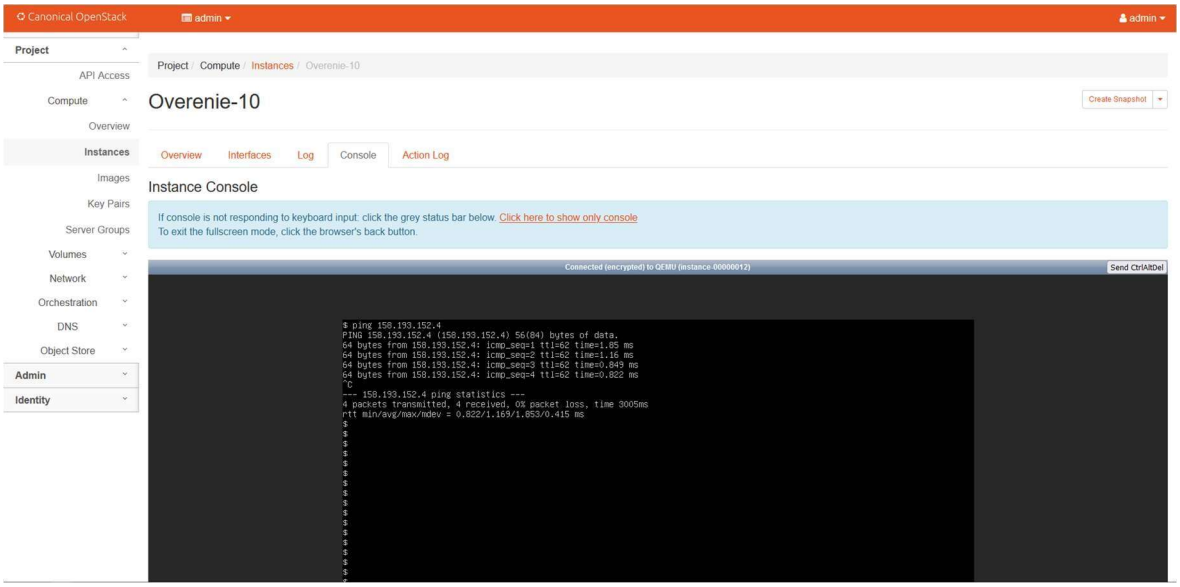


Figure 30 - Console in the web interface

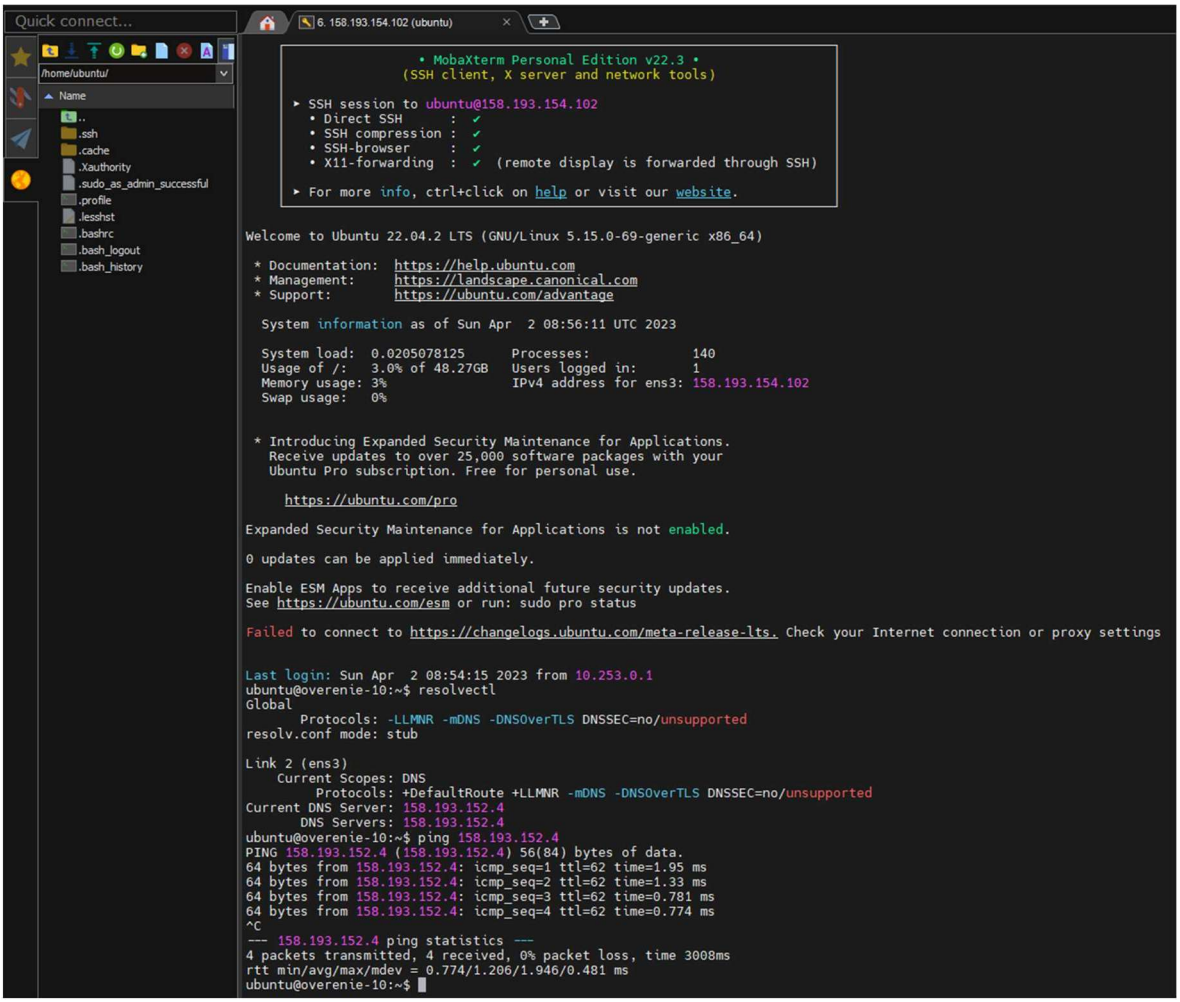


Figure 31 - Connecting to the console via the terminal

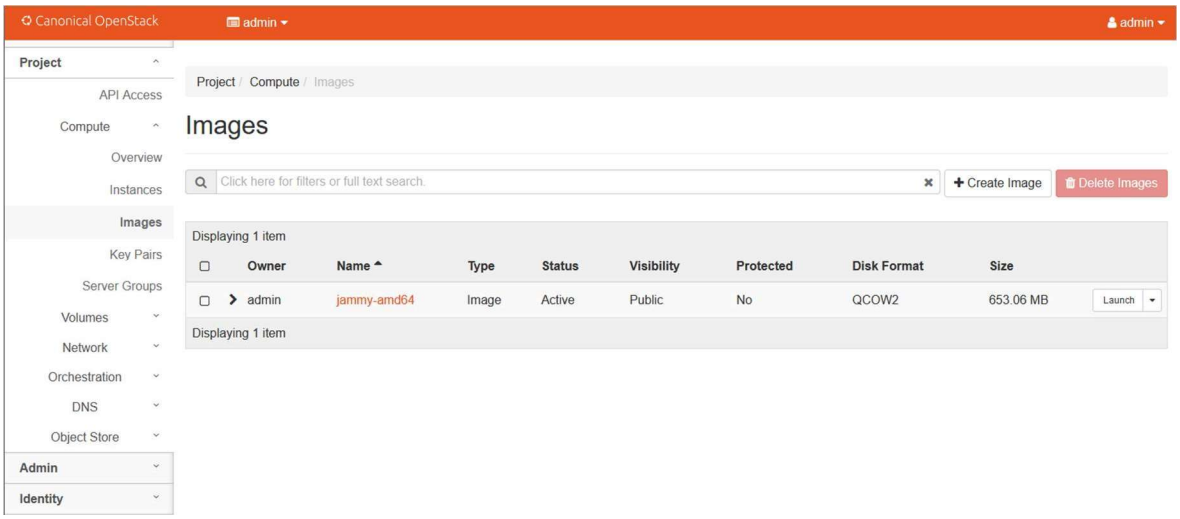


Figure 32 - List of uploaded images for booting

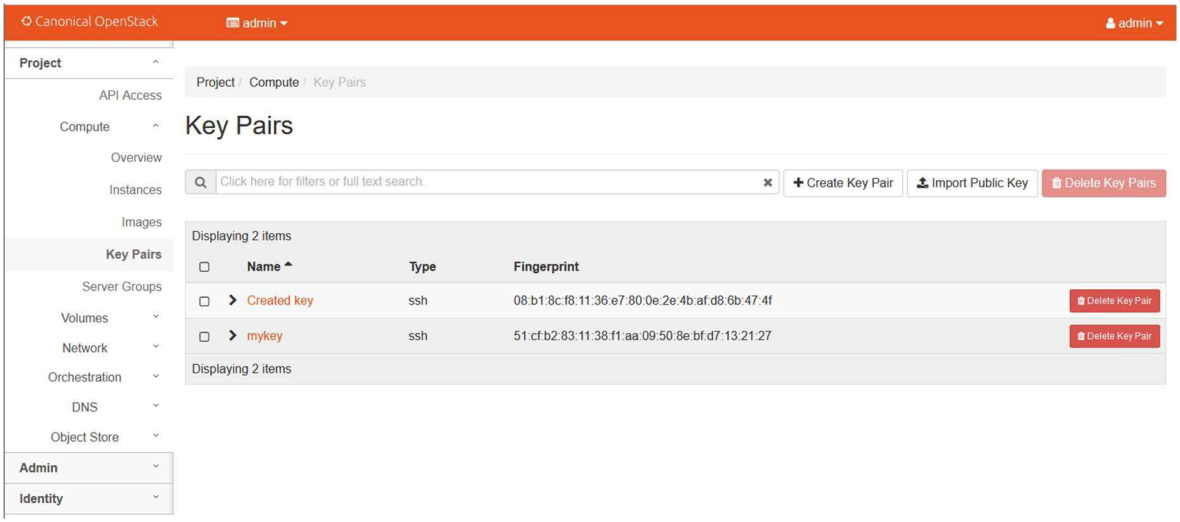


Figure 33 - SSH keys created/uploaded

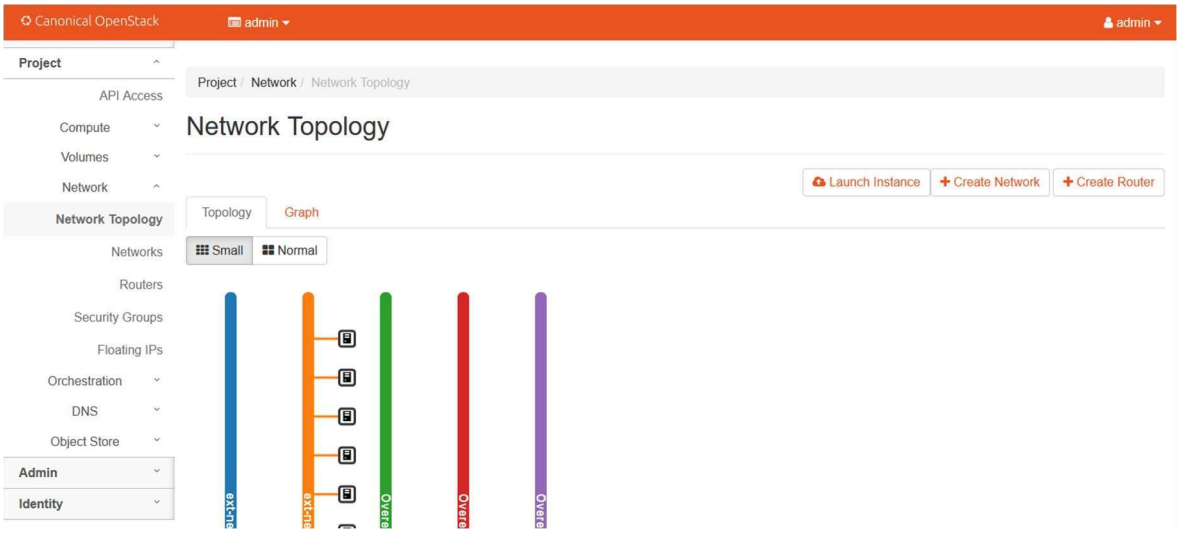


Figure 34 - Network topology

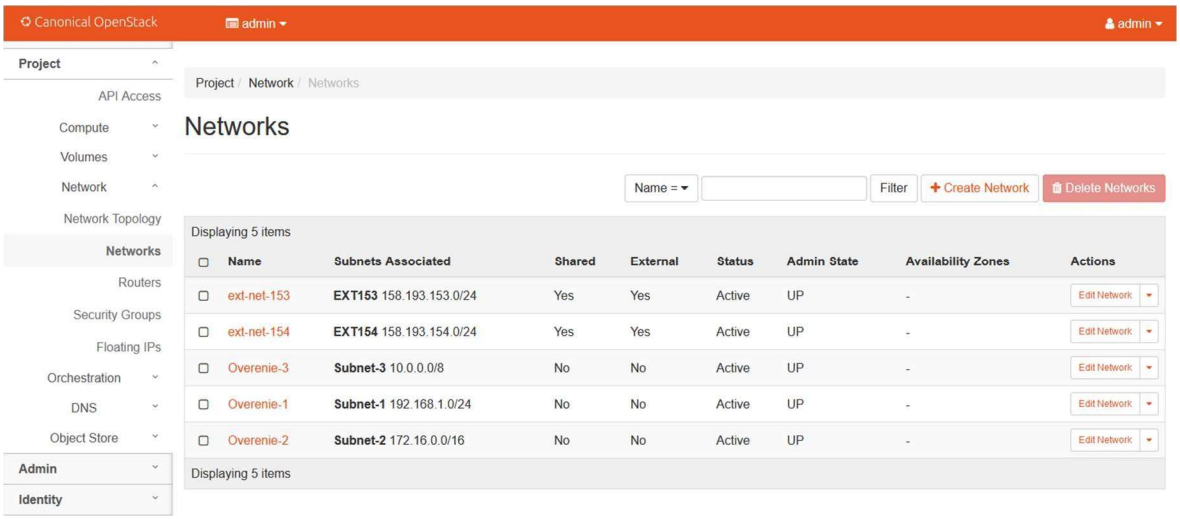


Figure 35 - List of created networks for verification

Conclusion

The main objective of the implementation was the deployment of the OpenStack cloud platform using Charms tools on physical and virtual infrastructure. In addition to deploying OpenStack on physical servers, the preparation and installation of virtual servers intended for running the MAAS and Juju systems were also carried out, while the remaining computing capacity was used to deploy the cloud platform controller.

After the technical requirements had been defined, the components needed to operate the platform were identified. The implementation then proceeded in a predefined sequence – initially the deployment of the MAAS system, which ensures efficient and automated provisioning of further servers. Within MAAS, the Juju server was subsequently also installed, which constitutes a framework for deploying application services by means of Charms.

The physical servers were configured in the MAAS environment, including their network topology and tags, which are essential for the correct distribution of components. Based on the list of required services, an installation bundle was created, in which the assignment of the individual components was defined. The control components were allocated to the controller and the compute components to the compute nodes. During configuration, further servers were also gradually integrated, with the necessary adjustments to network communication and testing of service availability being carried out.

To make the process more efficient, scripts were created to automate the installation and management of the environment. The technical documentation includes an extensive administrator's guide containing detailed instructions for the individual parts of the system, including the deployment of the Skyline web interface. The user documentation provides basic guides for interacting with the OpenStack environment, including procedures for creating and configuring:

- a virtual instance,
- a boot image,
- an SSH key,
- a network,
- a router.

List of references

[1] **vexxhost.com**. 2023. Cloud Orchestration with OpenStack Heat - An Introduction. 2023. Available on the Internet: https://vexxhost.com/blog/cloud-orchestration-with-openstack-heat/

[2] **openstack.org**. 2022. What is OpenStack?. [online]. 2022. Available on the Internet: https://www.openstack.org/software/

[3] **openstack.org**. 2023. OpenStack Deployment Tools. [online]. 2023. Available on the Internet: https://www.openstack.org/software/project-navigator/deployment-tools

[4] **openstack.org**. 2022. OpenStack Components. [online]. 2022. Available on the Internet: https://www.openstack.org/software/project-navigator/openstack-components

[5] **Juju Charm – RabbitMQ**. 2022. charm-rabbitmq-server. [online]. 2022. Available on the Internet: https://github.com/openstack/charm-rabbitmq-server

[6] **OpenStack Charmers.** 2022. Vault. [online]. 2022. Available on the Internet: https://charmhub.io/vault

[7] **openstack.org**. 2022. OpenStack Charms Deployment Guide. [online]. 2022. Available on the Internet: https://docs.openstack.org/project-deploy-guide/charm-deployment- guide/yoga/index.html

[8] **maas.io**. 2022. Canonical MAAS. [online]. 2022. Available on the Internet: https://maas.io/

[9] **juju.is**. 2022. Take control of.....[online]. 2022. Available on the Internet: https://juju.is/

[10] **github.com**. 2022. openstack-charmers / openstack-bundles. [online]. 2022. Available on the Internet: https://github.com/openstack-charmers/openstack- bundles/tree/master/stable/shared

[11] **Zaira Hira**. 2023. Shell Scripting for Beginners – How to Write Bash Scripts in Linux. [online]. 2023. Available on the Internet: https://www.freecodecamp.org/news/shell-scripting- crash-course-how-to-write-bash-scripts-in-linux/