Haaga-Helia
University of Applied Sciences

Deploying Openstack with Juju and MAAS

Masih Shekarak Ghashghaiee

Table of contents

1 Introduction

This project is a continuation of my Cloud Services project, where I had the opportunity to learn and practice the fundamentals of cloud computing, focusing mostly on IaaS. However, the scope and resources for the cloud project were limited, and I barely managed to scratch the surface with Openstack. The result of the project was a proof-of-concept deployment of Openstack using RDO's Packstack installer.

Packstack installation is a fast way to install Openstack. However, it is only for testing and therefore limited. Besides that, because I was running my hypervisor inside Oracle's VirtualBox, I could only run light Linux instance called Cirros, which is only for pinging other machines.

The main objective of this project remains the same as the previous project, that is to build and operate a private cloud infrastructure using Openstack. But this time on real hardware. In this project, the infrastructure has four physical nodes which will host Openstack services and two virtual nodes. One will run MAAS (region and rack) controllers and the other one runs the Juju controller. After the Openstack deployment, hosting a web application or building a complex infrastructure was also part of the original plan. Unfortunately, several big obstacles came across this project and shifted the path completely.

The plan is to first create a small cluster which will be controlled by MAAS. Then use Juju as the configuration management tool to deploy applications to the nodes of the cluster.

Initially, I hoped to write this report in a way that could be picked up and followed by anyone. But because there was a long gap between the deployment and writing of this report, that is not possible. New versions of the tools have been released and they no longer use the same processes as here and in some cases, their commands have changed. The project started with MAAS version 2.4 but upgraded to 2.6 and 2.8 along the way. However, the report will describe the journey and should give a understanding of all the steps and requirements.

Here, I will more focus on the concepts related to the installation processes. Full review of the tools used is out of the scope of this report.

2 Background

There are many ways to install Openstack. One can install Openstack from scratch by installing and configuring each Openstack service separately from their project repositories. That is a very complex process which requires a thorough understanding of various concepts

and takes the longest time to implement. Enterprises that want to build large production environment suitable for large projects, or commercial use, usually install Openstack from scratch.

Another way to deploy Openstack is to use deployment and lifecycle management tools. These tools are generally two categories. Configuration management is the core concepts of these tools.

1- Frameworks for lifecycle management

- TRIPLEO deploys Openstack using Openstack itself
- Openstack-HELM deploys Openstack in containers using Helm
- KOLLA-ANSIBLE deploys Openstack in containers using Ansible
- KAYOBE, deployment of containerized Openstack to bare metal
- Openstack-ANSIBLE, Ansible playbooks to deploy Openstack
- Openstack-CHARMS deploys Openstack in containers using Charms and Juju
- BIFROST, Ansible playbooks using ironic
- Openstack-CHEF, Chef cookbooks to build, operate and consume Openstack

2- Packaging recipes for popular frameworks

- LOCI, Lightweight OCI containers
- PUPPET-Openstack, Puppet modules to deploy Openstack
- RPM-PACKAGING, RPM package specs to deploy Openstack

In this project, I build a four-node Openstack cloud using Charms with the help of MAAS and Juju. While this is a minimal cloud, it can do some real work and, it should be easy to scale up the project for more ambiguous needs thanks to the unique capabilities and flexibility of Juju. With the Openstack charms, it is also possible to deploy the cloud on LXD containers, all on a single machine. That is called “Openstack on LXD”, which calls for Juju to be backed by the LXD cloud type as opposed to the standard MAAS cloud type.

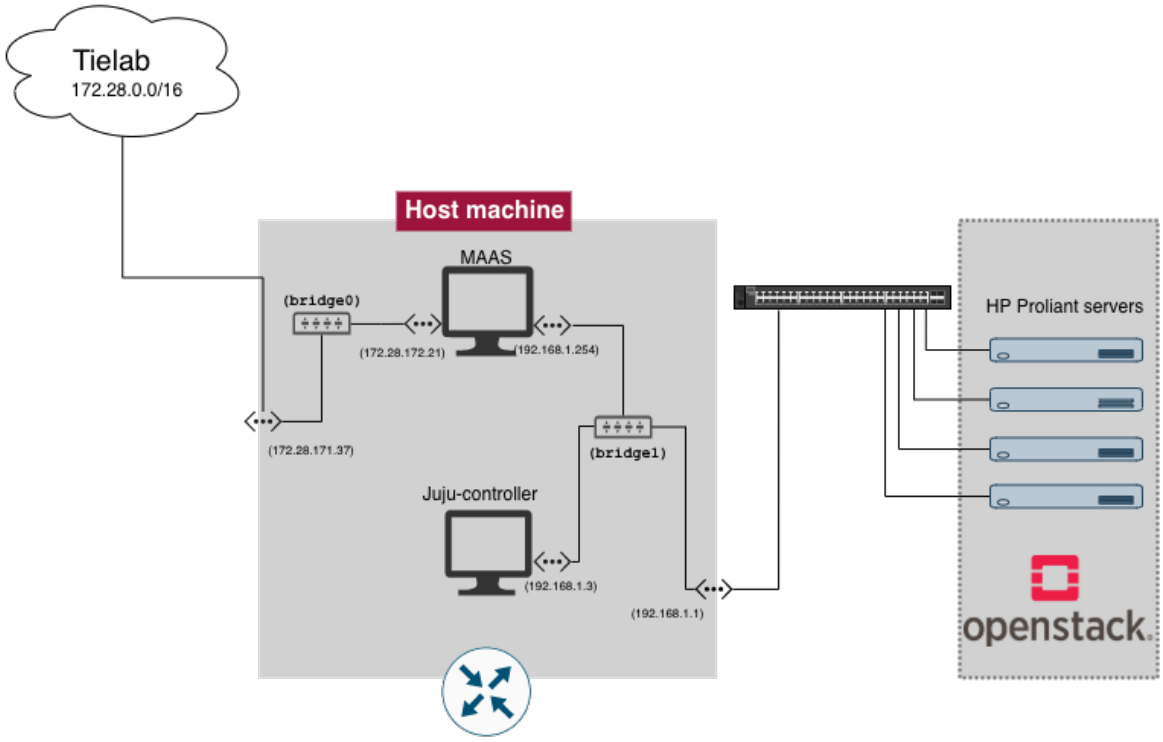
The process, in short, is as follows:

- Preparing the environment
- Installing MAAS
- Giving the controller of HP Proliant servers to MAAS
- Building the Juju controller
- Deploying Openstack charms with Juju

All the four servers were HP ProLiant DL360p Gen8. The firmware, ROM and iLO (HP's remote server management software) were updated as the first step of the project. Even though the HP servers are powerful machines, they are old and for some reason had not been updated for a long time. I didn't have HP licence key which would allow a full update of firmware, drivers and software on the machine. Some of the components such as storage and memory had already failed before and during this project which could be the result of the old firmware.

All the machines that will host Openstack, must have at least two Network Interface Card (NIC) and two disks for storage. One disk will be used for the Operation system and the other one is used by Openstack.

Below is the logical view of the environment.



Environment diagram

Why did I choose Juju and charms? Me and my teammate, who was part of this project in the beginning, spent two weeks researching and reviewing all the deployment options. As we went through different options, we found out that learning to use at least one more tool, such as Ansible, Puppet, Kolla, Salt, Docker, Helm, Chef, Juju, MAAS, was a prerequisite regardless of which framework we choose.

Because we were somewhat familiar with Openstack terminologies and had even experience with its environment from our previous project, we started studying the documentations.

Openstack has guides for all the possible deployments. Doing that, helped us to estimate the level of complexity of our options. We based our evaluation on the following criteria:

- The complexity of hardware preparation
- Number of steps before the installation began
- Number of programming language and third-party technologies involved
- Familiarity with the terminologies
- Automation level
- Availability of external resources
- Fixed bugs and open bugs

After reviewing all the scores, the Openstack Charms Deployment had the highest score. Also, we found good reviews about Juju and MAAS and that made us a bit biased and we chose Charms deployment. While I struggled a lot in the environment preparation phase, the deployment itself is not difficult and the process is smooth if the documentations are followed carefully.

3 Preparing the infrastructure

The minimum requirement of this method was to have four nodes and two servers to run MAAS and Juju controllers. Besides the four HP Proliant servers, I had also one workstation for this project. Because I was missing one machine, I decided to run virtual Juju and MAAS on the workstation.

The virtual machines are very convenient and helped me in the previous project as well. For example, taking backup is very easy with the cloning and snapshots and it was used countless times during this project. The second advantage of using a virtual machine was that the hardware configurations could be changed easily. For example, attaching more network interfaces or storage when needed.

But using a virtual machine had its downsides too. For example, it brought complexity and uncertainty to the networking process. When a problem occurred, for example, if MAAS couldn’t communicate with the nodes, it was hard to identify the source of the problem with certainty. But because a virtual machine should have theoretically worked fine, the benefits won the disadvantages and I took the risk.

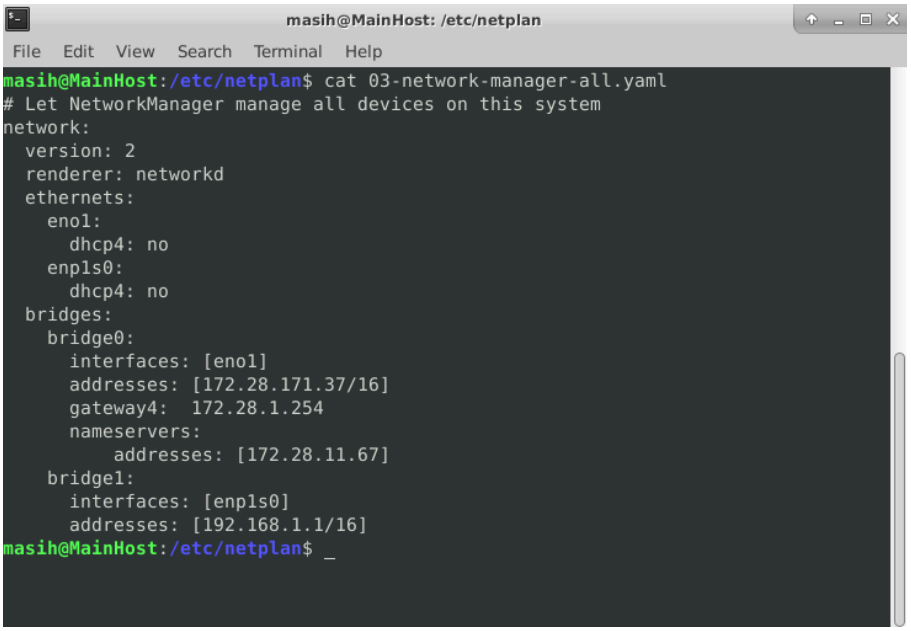
3.1 Creating virtual machines

The workstation where MAAS and Juju VMs are hosted is named `MainHost` in this report. `MainHost` has an Intel i5 CPU (3.4Ghz and 4 cores) and 32GB RAM. Therefore it has enough power to easily host the VMs. The relevant specification of the host is that it has two network interface controllers (NIC). One is connected to Tielab and one is dedicated to this project. Of course, it was also possible to implement this project with one NIC. For example by using VLAN on the L2 switch, but unfortunately at that time, I didn't have sufficient knowledge in that area. Because the MainHost also acts as a router, using two NICs was simpler and gave more freedom and flexibility. The Host and the four HP Proliant servers are connected to an HP ProCurve switch.

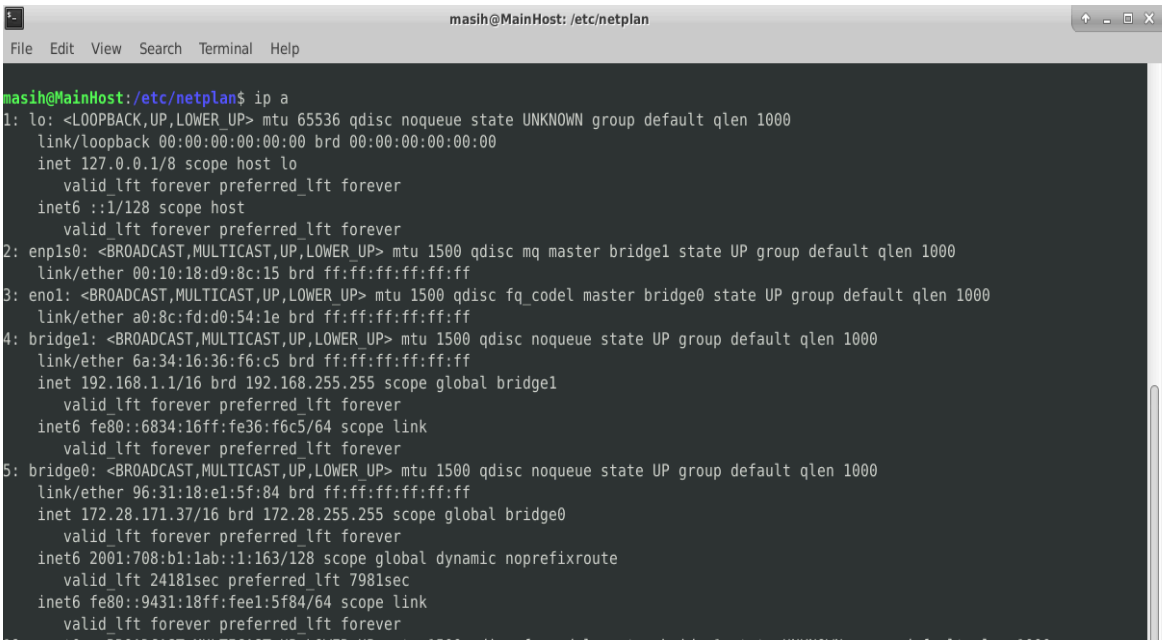
3.2 Configuring Host's bridge interfaces

Because MAAS and Juju must have direct access to the LAN, a bridge interface is used. A Linux bridge is build using Netplan. Netplan is the new default utility used as Ubuntu network configuration backend. Netplan reads network configuration from **`/etc/netplan/*.yaml`**. During early boot, Netplan generates backend-specific configuration files in **`/run`** to hand off control of devices to a particular networking daemon. Which in Ubuntu servers is **`Systemd-networkd`** daemon**.** YAML uses precise indentation and must be followed otherwise the configuration will fail.

Bridge interfaces can be created by adding the `bridges` keyword to the YAML file and adding the interfaces which will be the master of the bridge interfaces. Only the interface connected to the Tielab network has a default gateway.

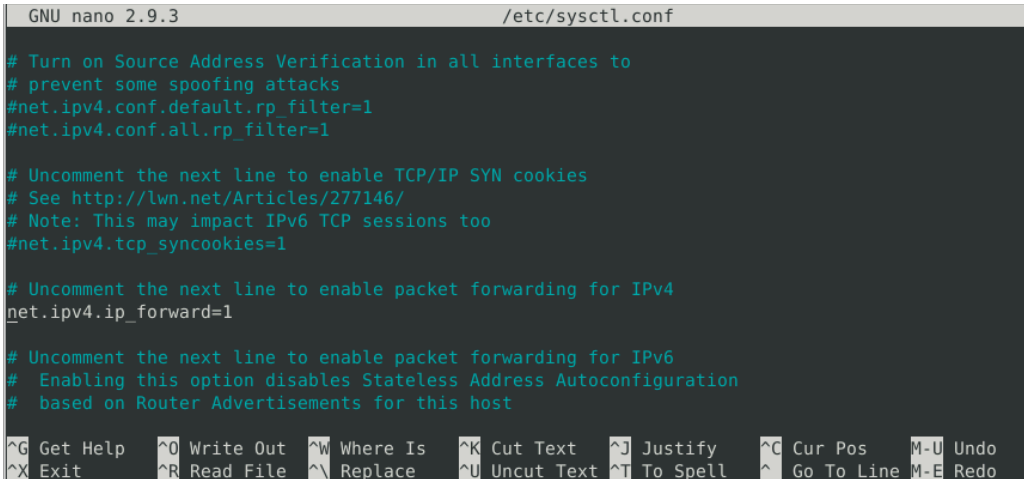


Interface configuration shows that bridges are successfully created and active. The physical network interface controllers (`enp1s0` and `eno1`) are now the master of the bridges.



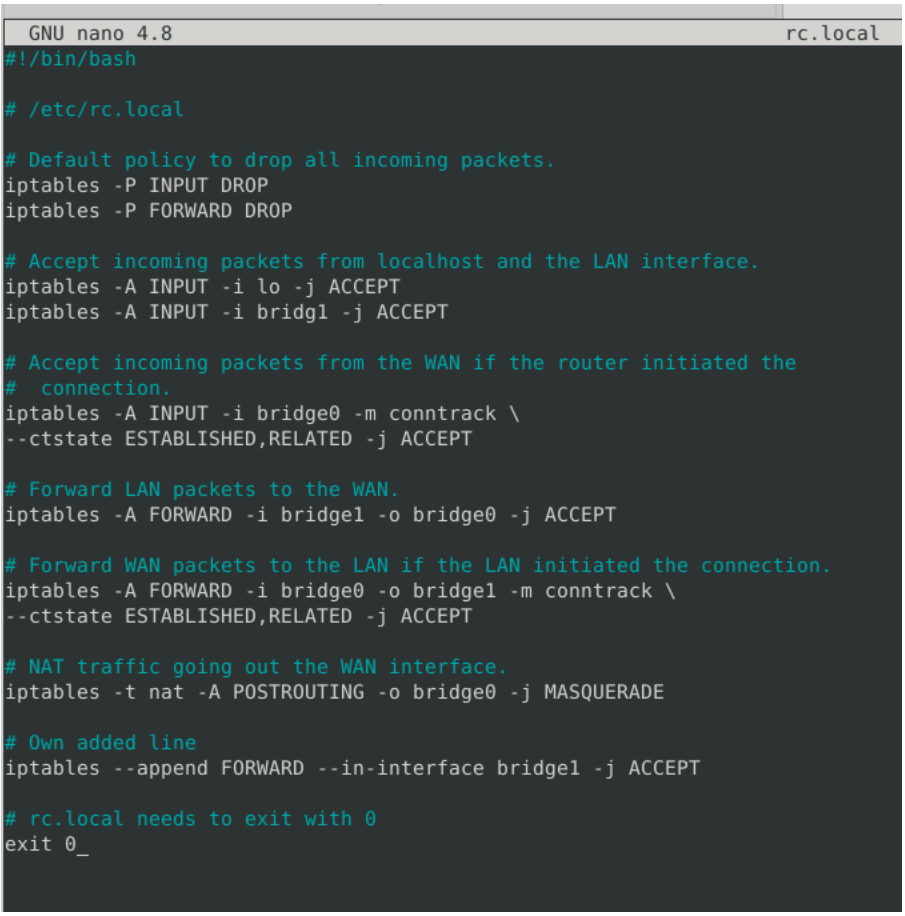
3.3 Routing and NAT service

As shown in the diagram, `MainHost` machine also acts as a router. Interface `enp1s0` (bridge1) routes traffic between 172.28.0.0/16 and 192.168.0.0/16. To configure that, I first enable packet forwarding by uncommenting the line below in **`sysctl.conf`** in **`/etc`** and setting its value to **`1`**.



The command **`sysctl -p`** activates the new configuration. NAT service for the network 192.168.0.0 is build using `iptables`. To have persistent connectivity, the NAT configuration must load with booting. For that, I placed the `iptables` rules necessary for the NAT and

routing, inside a bash script file. This file is saved as **`/etc/rc.local`** file and will be executed during the boot. The comments above the command should help in explaining the parameters better.



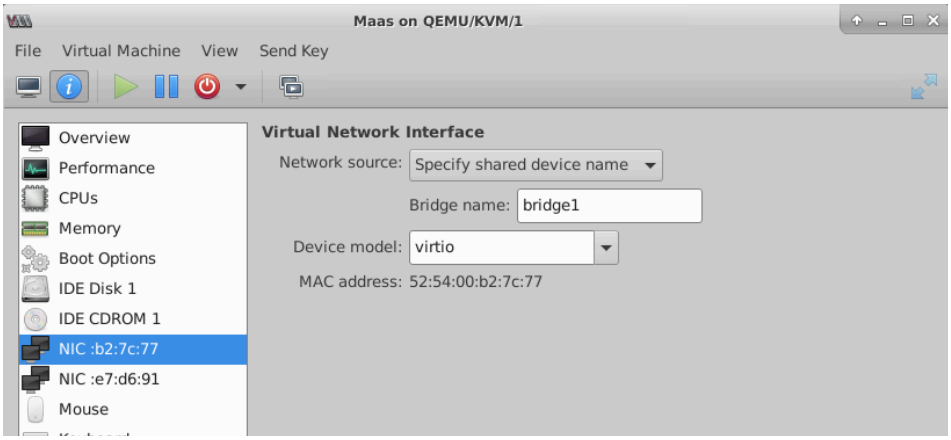
After this, machines connected to the HP switch should reach the internet.

3.4 Starting the VM

The virtual machine built here will be called `maas` machine from here onwards. Virtual Machine Manager (VMM with bash name `virt-manager`) is used to control the VMs. The VMM is a desktop user interface for managing `libvirt` daemon. Libvirt is a powerful tool for managing virtualisation platforms and work extremely well with QEMU and KVM technologies. Therefore it is much better suited for this project compared to VirtualBox. Besides the possibility of ISO image, Libvirt also supports installation from other technologies Network Boot (e.g. PXE) or Network Install using HTTP, FTP or NFS which I will use later in this project.

Next, I start a new VM with 2-core CPU, 8GB RAM and 50 GB storage and an Ubuntu server 20.04 image. Additional NIC was attached to the machine as well. One interface will be con-

nected to `bridge0` and one to `bridge1` of the Host. As shown below, the name of the interface must be typed in the field manually. Device model can be anything but virtio has the best performance.



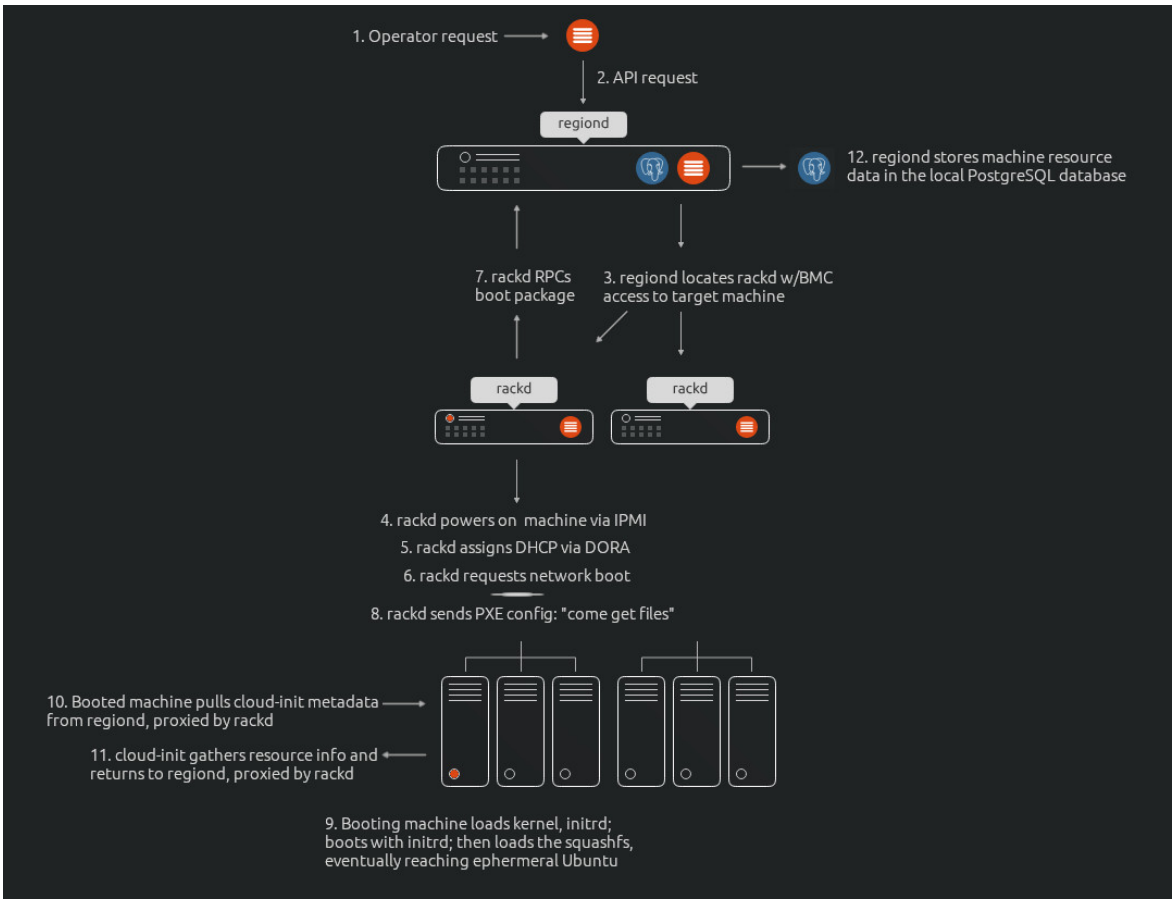
Now I can start the VM and follow the basic Ubuntu server installation. The interface attached to `bridge1` will not get an IP address during the installation because there is no DHCP server available for that network just yet. After the installation is ready, the NICs must be configured. Below is the configuration of static IPs to the interfaces.

```
masih@maas: ~
masih@maas:/etc/netplan$ cat 00-installer-config.yaml

network:
    version: 2
    renderer: networkd
    ethernets:
        ens3:
            addresses:
              - 192.168.1.254/16
            gateway4: 192.168.1.1
            nameservers:
              search: []
              addresses: [8.8.8.8]

        ens4:
            addresses:
              - 172.28.172.21/16
```

Now, I only set the gateway for the 192.168.0.0/16 subnet. 192.168.1.1 is the MainHost IP which I configured to route traffic in previous steps.

Next, I'll install MAAS on the `maas` server. Below, I explain the process and at the same time cover the related theory.

4 MAAS



Canonical MAAS communication 2020

MAAS (metal-as-a-service) is Canonical's open-source tool for building datacenter from bare-metal servers. It converts bare-metal servers into cloud instances of virtual machines. MAAS has a tiered architecture with a central Postgres database backing a "Region Controller (`regiond`)" that deals with operator requests. I will do all configuration in the region controller. Rack Controllers (`rackd`) provide services to the racks. The controller itself is stateless and horizontally scalable, presenting only a REST API. They cache large items like instance images at the rack level for performance but maintain no exclusive state other than credentials to talk to the controller. [Canonical 2020]

MAAS uses standard server BMC and NIC services such as IPMI and PXE to control the machines in the data centre. For converged infrastructure, MAAS talks to the chassis controller of the rack or chassis such as Intel RSD, Cisco UCS or HP Moonshot. Custom plugins extend MAAS for alternative BMC protocols.

Initial machine inventory and commissioning are done from an ephemeral Ubuntu image that works across all major servers from all major vendors. It is possible to add custom scripts for firmware updates and reporting. [Canonical 2020]

The image above shows MAAS' architecture and the communication between the region and rack controller and how they control the nodes. Before starting the installation, there are few key terms used in MAAS which needs to be clarified.

- **Node** is a machine or device that have been added to MAAS thus have become an object. MAAS manages nodes through a life cycle. These are three types of objects in MAAS, Controller, Machines and Devices.
- **Controllers** are two types, region and rack controller. Region controller deals with operator requests. In this project, I only log into the region controller and operate there. A rack controller provides services to racks of servers.

  A region controller consists of five components:

  1. REST API server
  2. PostgreSQL database
  3. DNS
  4. Caching HTTP proxy
  5. Web UI

  A rack controller provides four services:

  1. DHCP
  2. TFTP
  3. HTTP (for images)
  4. Power management

- **Machine** is a node ready to be deployed.
- **Device** is a node that can't be deployed. Like switches, hubs or routers.
- **Space** is a collection of subnets that we create in MAAS. A subnet can belong to one space only.
- **Zone** is an organisational unit containing nodes where each node can in one zone only. These can be created and configured freely by the user.
- **Fabrics** are a lot like VLANs term in networking**.** The main difference is that Fabrics make VLAN-to-VLAN connection possible.
- **Tags** are different than VLAN tags in MAAS. They are simply for identification of nodes and targeted deployment of services.
- **Subnet** holds its traditional definition in MAAS.
- **IP range** are two types
  - **Reserved range** Mode operates differently depending on whether the subnet is managed or unmanaged:

        - **Managed (subnet)**: MAAS will never assign IP addresses inside this range. These can be used for anything, such as infrastructure systems, network hardware, external DHCP, or an OpenStack namespace.
        - **Unmanaged (subnet)**: MAAS will only assign IP addresses inside this range.
    - **Reserved dynamic range** An IP range that MAAS will use for enlisting, commissioning and, if enabled, MAAS-managed DHCP on the node's VLAN during commissioning, deploying. An initial range is created as part of the DHCP enablement process if done with the web UI. MAAS never uses IP addresses from this range for an unmanaged subnet.

There are many more term and concepts but because they are not directly affecting this project I don't mention them here.

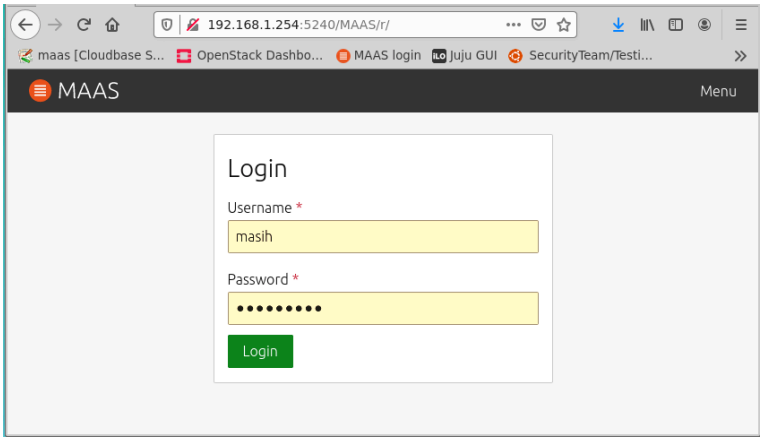
4.1 Installation

For small infrastructure like this in this project, there is no need for a separate region controller. The commands below will install rack and region controllers at once. The first command enables the repository and the second one installs MAAS:

```
sudo apt-add-repository ppa:maas/stable
sudo apt install maas
```

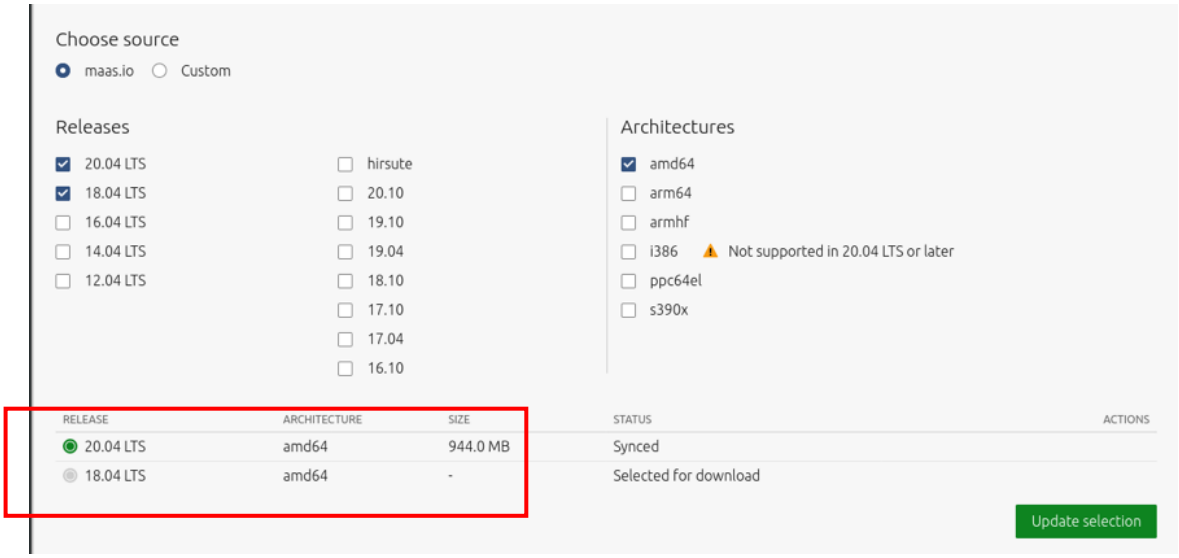
After a successful installation, an admin user must be created with **`sudo maas create-admin`** command. Now MAAS is set up and running.

The GUI dashboard can be accessed from **`http://${API_HOST}:5240/MAAS.`** The `API_HOST` address is the interface IP address. If there are several interfaces available, like in my case, both addresses will provide access to GUI. In this report, I use **`http://192.168.1.254:5240/MAAS`** URL. When logged in for the first time, the configuration page opens.

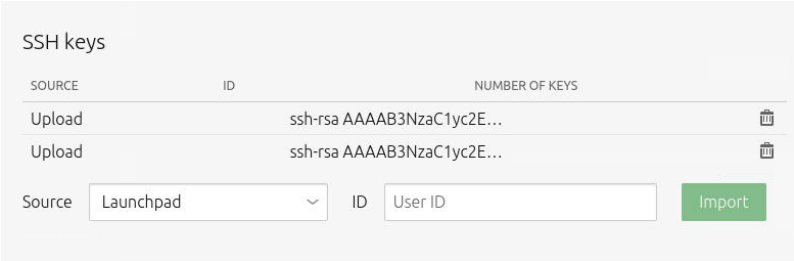


4.2 Configuration

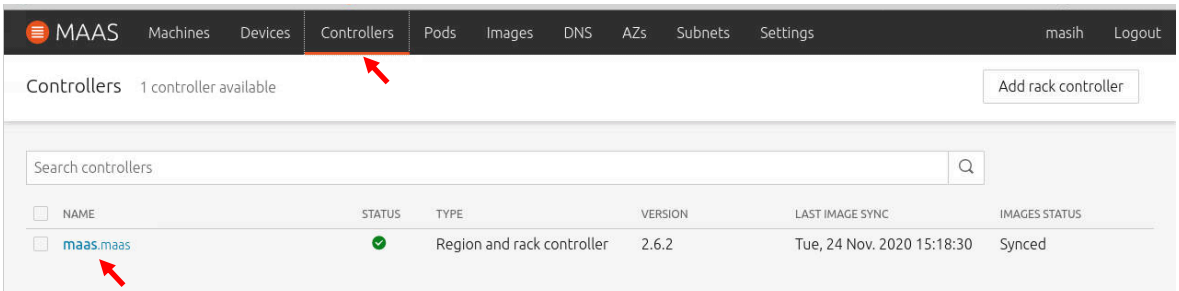
After a fresh MAAS installation, the web UI presents a couple of welcome screens. From these screens, many system-wide options, including connectivity, image downloads, and. Authentication keys can be set. The DNS forwarder is configured to google's DNS server. For the image, at least one LTS image should be selected.



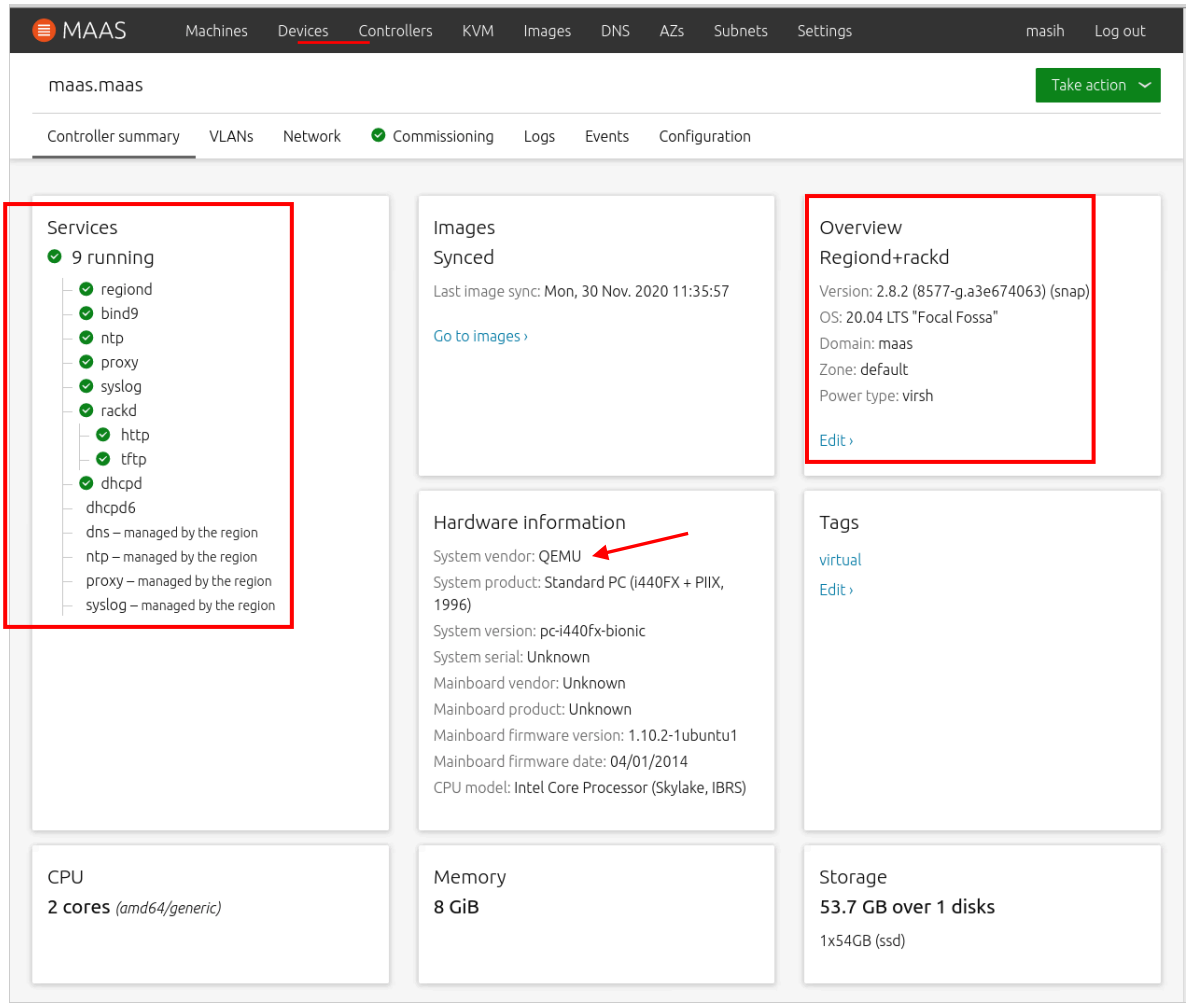
A "Launchpad" or "GitHub" keys or user's public key should be added in the SSH Key section. These keys will be automatically added to the machines deployed with MAAS and Juju which makes the remote login to the machines easier.



After the basic configurations are done, it is a good idea to check if MAAS services are all installed and running without errors. This can be checked by going to the Controllers tab.



The image below shows the MAAS version as 2.8.2 while the above is 2.6.2. That is because the previous deployment crashed in the middle of this report.

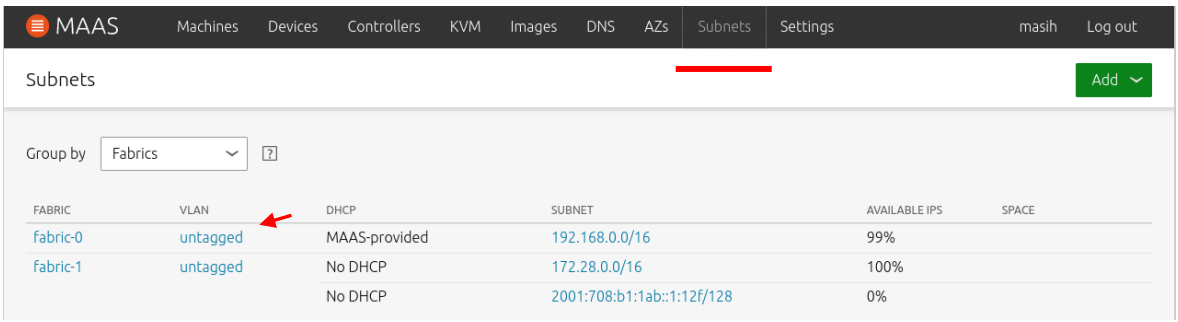


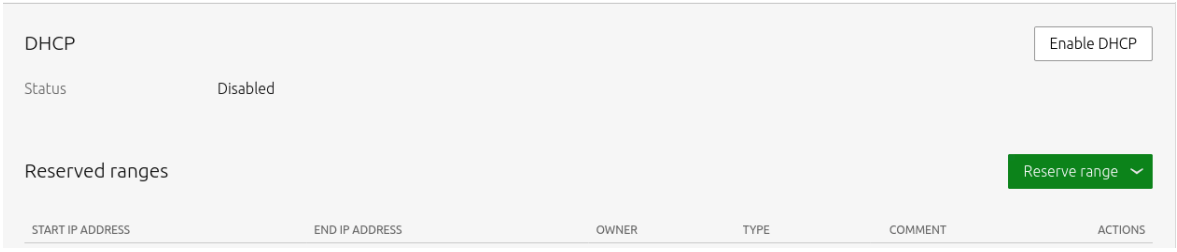
The left section with green indicators shows that installed services are running and healthy.

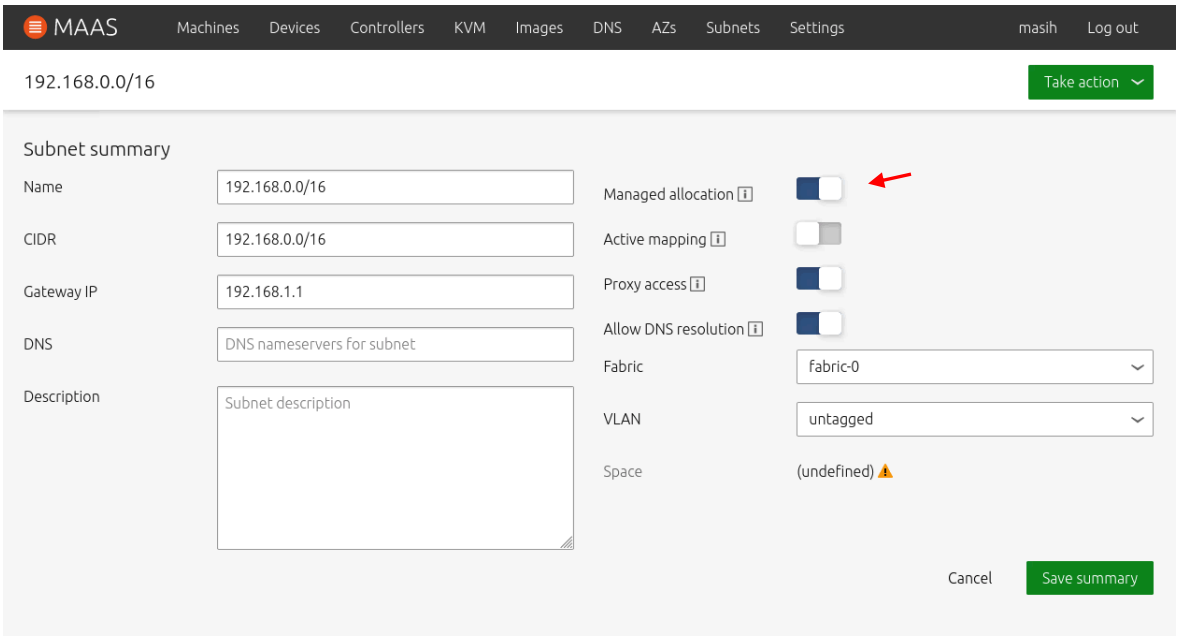
4.3 Subnet

The topic of subnet management relates to whether or not MAAS is in full control of a subnet. When a subnet is managed, MAAS handles all aspects of IP address allocation. This process includes DHCP leases and assigned static addresses.

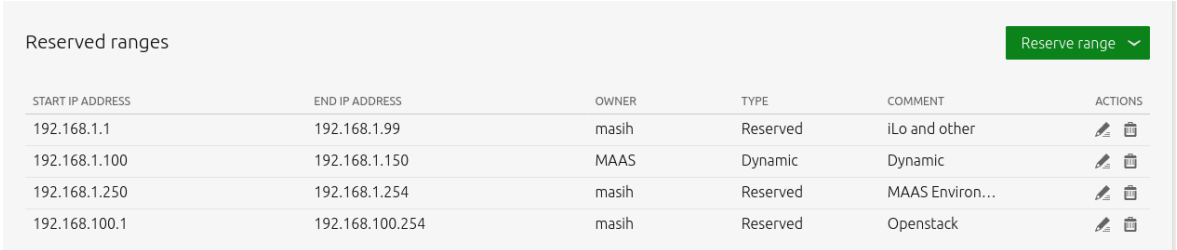
MAAS will automatically find the LAN(s) it is connected to. These are called Fabrics in MAAS environment. Two Fabrics are identified in my case. Therefore I have to specify which network MAAS should control. That can be configured by clicking in the subnet IP address. Next, I have to enable DHCP on my private network because with it MAAS can't control the environment easily. DHCP can be enabled by clicking on the *untagged* link under VLAN.







On the same page, I can also configure Reserved and Dynamic IPs. Reserved IP is used for machines which I want to control myself in this network. As explained this before in the 'Managed' network, this means MAAS won't use the IP addresses specified in the 'Reserved section'. When a machine netboot to MAAS, an address from Dynamic IP range will be assigned to it.



5 Adding Machines to MAAS

The most important role of MAAS is to control datacenters' resources, be it servers, racks, devices or even virtual machines. Machines can be viewed in the Machine tab. Of course, this page is empty in the beginning.

Each machine managed by MAAS goes through a series of processes called lifecycle. These are commission, deploy, acquire, release. For the Openstack project, I need to only commission the machines. Juju will take care of deploying process

5.1 Enlisting

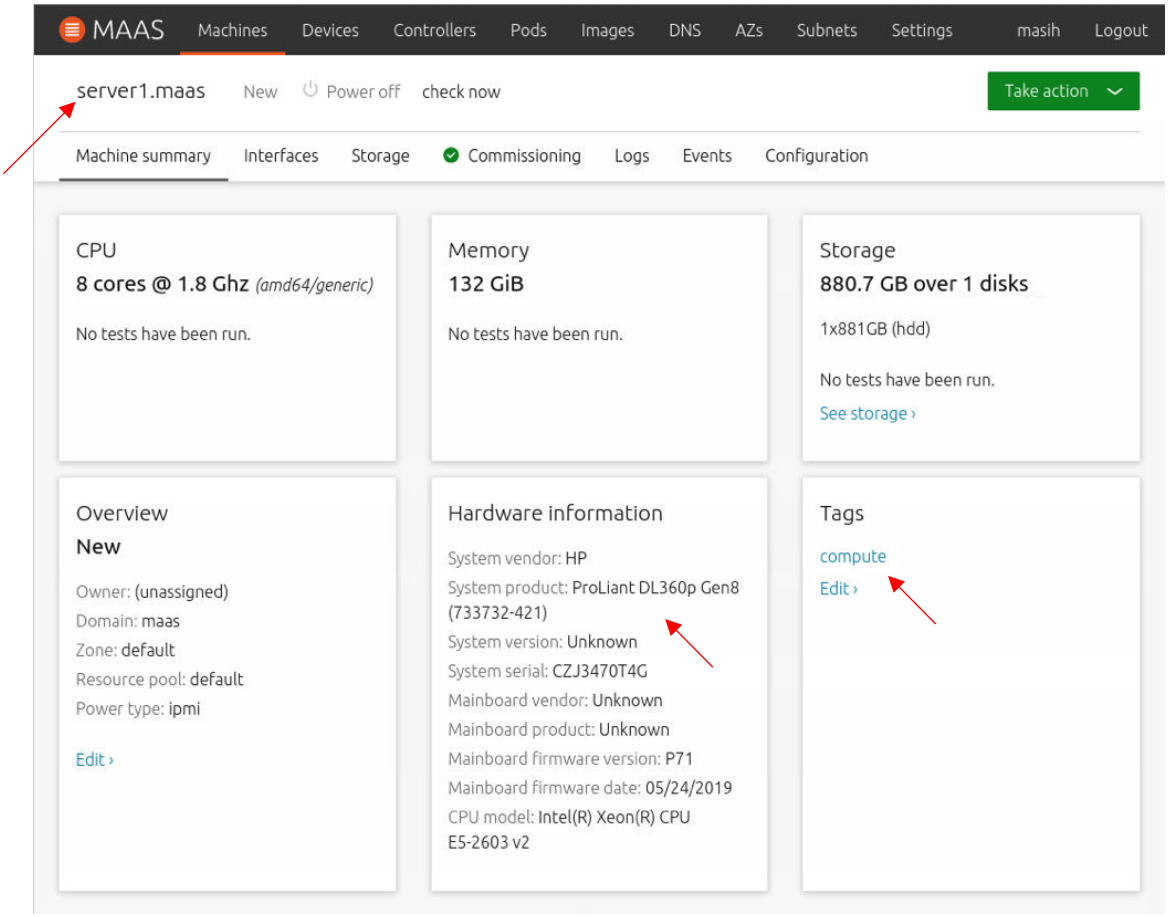
When machines netboot (e.g. PXE-boot) on a MAAS network, they will be automatically enlisted, if MAAS can detect their Baseboard Management Controller (BMC) parameters. This is one of the reasons why I had to separate MAAS from Tielab so I don't interfere with other students' project accidentally.

For a machine to be controlled by MAAS, they should always have netboot as their first booting method. After configuring the booting method and turning them on, they go through the enlisting process. The DHCP is reached and the ephemeral software and configurations will start executing.

The enlisting machine will appear in the MAAS UI as well and the 'Status' updates to each process that MAAS executes on the machine. MAAS assigns a random name to the machine while enlisting.

HP Proliant are IPMI enabled which lets their power to be controlled by MAAS. In the image below, the enlisting is completed and the status updated to 'New'. Basic hardware information is displayed as well. More details are provided when clicking on the machine name.

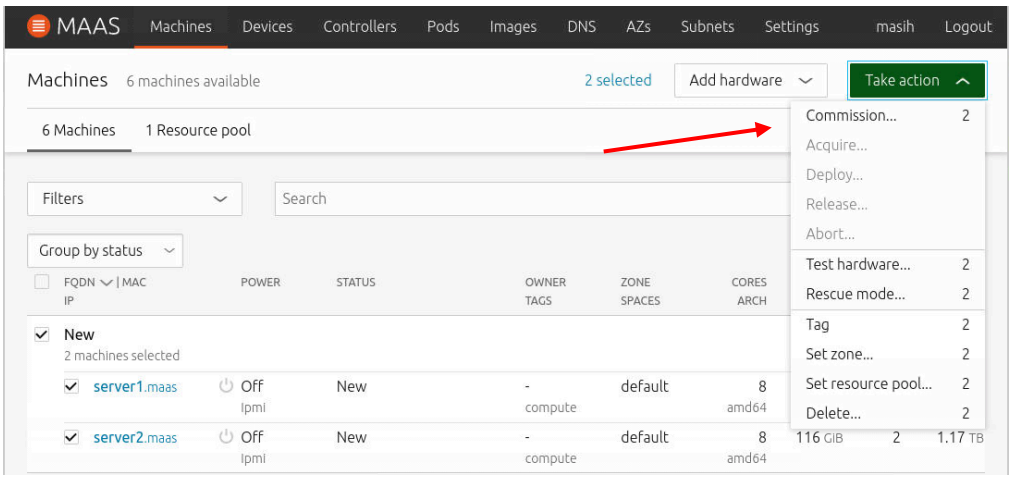
Name of the machine can be changed from the top left corner. I also edit the tag to ‘compute’. Later with Juju, I use tags to target the configuration to a specific machine.



5.2 Commissioning

Next, I have to *Commission* the machines. In this phase, several scripts are executed one after another automatically. To summaries the script process, when MAAS commissions a machine, the following sequence of events takes place:

- The DHCP server is contacted
- kernel and `initrd` are received over TFTP
- machine boots
- `initrd` mounts a `Squashfs` image ephemerally over HTTP
- `cloud-init` runs built-in and custom commissioning scripts
- machine shuts down



The first time I tried to commission the HP server, MAAS would fail. My first suspicion was that communication was failing because MAAS was a virtual machine. Something I was worried about from beginning when deciding to run MAAS inside a virtual machine. After spending many hours, the problem was simply a bug and was resolved by upgrading MAAS. Should I have read the documentation thoroughly, I would have saved those hours. To upgrade first, the repository is added to `apt` and the normal system upgrade.

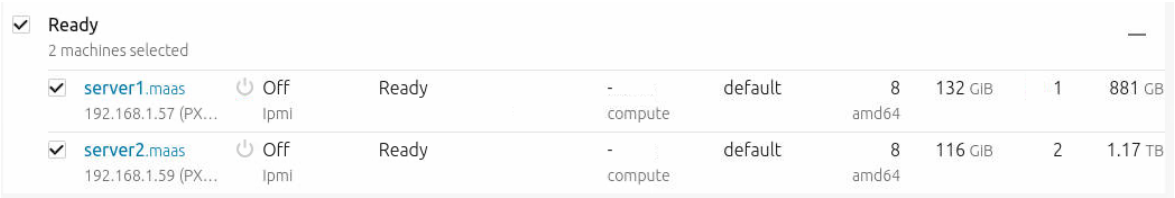
```
apt-add-repository ppa:maas/2.6
```

After upgrading, HP servers were enlisted successfully. Images below show a few phases of commissioning of two servers. Below are a few of the stages that the machines go through.

1) Powering on the machines

2) PXE booting and loading ephemeral

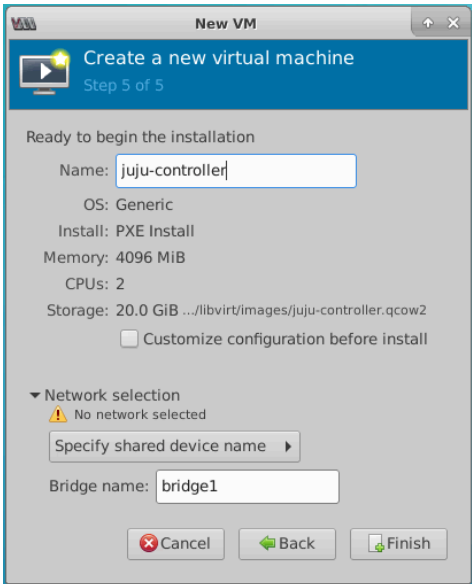
3) Commissioning is ready and the machine turned off.



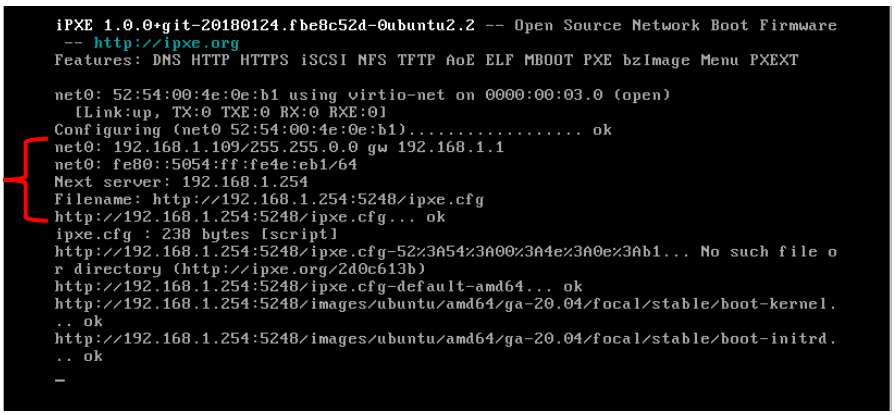
After successful commissioning, the status is changed to 'Ready' which indicates that MAAS has gained full control of the machine.

5.3 Commissioning a virtual node to MAAS

If MAAS cannot find BMC parameters, the machine can be added manually. the Juju is a virtual machine and must be added manually. For that, I start a new virtual machine but instead of starting it from ISO image, I choose, Network Boot (PXE).

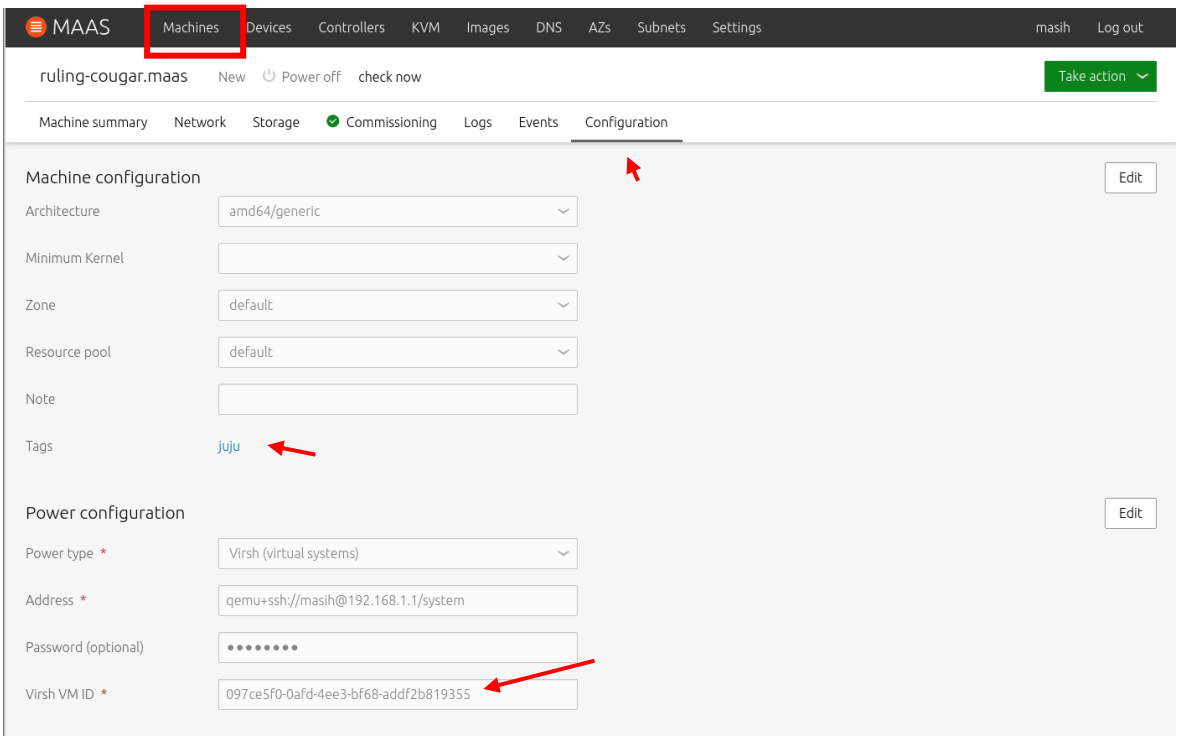


After choosing PXE, all the following options are the same as creating a virtual machine. Such as adding RAM, CPU and Storage. I chose 8GB RAM, 2CPU and 40GB storage. Juju provides services on the 192.168.0.0 network. Therefore it should be attached to the shared device `bridge1` as shown above. After configurations are ready, the machine will boot and should find the DHCP server and enlisting starts.



The terminal above shows that MAAS has been reached and an IP address from the dynamic range I specified earlier, has been assigned to this machine. Like with the HP nodes, a 'New' machine with a random name added to the MAAS. But unlike physical hardware, MAAS can't read all the hardware configuration of the virtual machine. This will be done in the commissioning process.

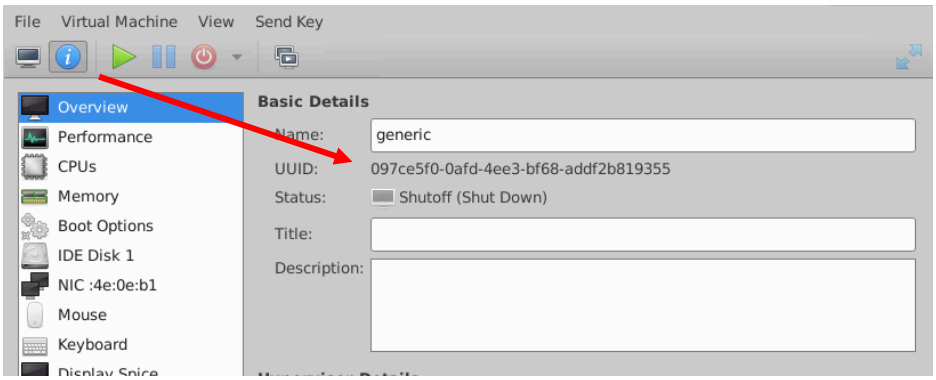
As we can see above the power option is 'Unknown'. Meaning that MAAS can't control this machine just yet. To fix that, I have to add the power parameters in the machine's configuration tab.



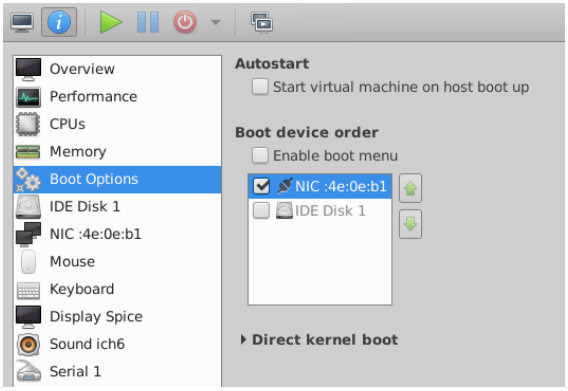
From the ‘Power configuration’ drop-down list, the Virsh is chosen. The virsh address is the host’s address. In this case, MainHost (192.168.1.1) and the user masih is added. So the pattern is always like this:

```
qemu+ssh://<user>@<host IP>/system
```

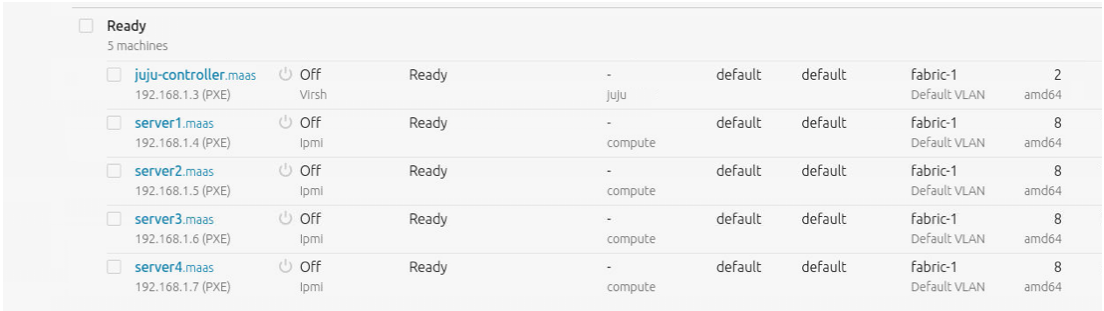
The virsh password is the masih’s password. Even if MAAS says this password is optional, for me it didn’t work without inserting it. The last section, ‘Virsh VM ID’ is a virtual machine’s UUID which can be found from the virt-manager info tab.



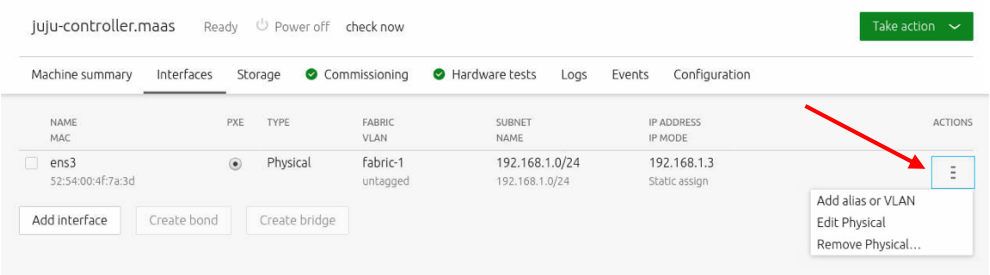
Again like the physical nodes, the first Boot option must be set to netboot.



Now that MAAS knows the power parameter of the virtual machine, I can commission in the VM in the same way I commissioned the HP servers. Finally, I have four ‘Ready’ nodes for Openstack and one node for Juju controller.



Some of the hardware configurations of a machine can be changed while they are in the ‘Ready’ state. The image above shows that I have assigned a static IP address to machines. These are from my ‘Reserved’ IP range. Below is how that can be done.



6 Juju

Juju is an open-source modelling tool for operating software in the cloud. Juju allows to deploy, configure, manage, maintain, and scale cloud applications quickly on public clouds, as well as on physical servers, OpenStack, and containers.

Juju uses Operator Lifecycle Manager (OLM) that provides model-driven application management and infrastructure-as-code. Operators are an automation approach for configuration management. A charm is a compressed archive of the operator code and metadata, which can also be shared directly along with applications.

At the lowest level, traditional configuration management tools like Chef and Puppet, or even general scripting languages such as Python or bash, automate the configuration of machines to a particular specification. But Juju also creates a model of the relationships between applications that make up a solution and gives users the mapping of all parts of that model to machines. Juju then applies the necessary configuration management scripts to each machine in the model.

6.1 Terms and Concepts

There are a few key concepts that I’ll be using in this deployment. These are:

- **Cloud** to Juju, is a resource which provides machines (instances), and possibly storage, in order for application units to be deployed upon them.
- This can be public or private cloud like AWS or Openstack but it can also be a cluster of machines like MAAS or LXD.
- **Controller** is the initial cloud instance which is created in order for Juju to gain access to a cloud using cloud’s API. The controller is a central management node for the chosen cloud, taking care of all operations requested by the Juju client.

- **Client** is the software that connects to Juju controllers and is used to issue commands that deploy and manage application units running on cloud instances. I will install this software on the `maas` machine and with it send commands to `Juju-controller.`
- **Charms** contain all the instructions necessary for deploying and configuring application units. They can be downloaded from the Charm Store.
- **Subordinate charm** is a charm that supplements the functionality of another charm. Subordinates don't create a new unit and can be added only when a relationship has been created.
- **Container is a** generic term used for LXD and KVM machines
- **Unit and application** is the software that we deploy to a machine. An application might require only one unit but the complex application might have multiple units running in different machines. All units for a given application will share the same charm, the same relations, and the same user-provided configuration.
- **Leader** is the application unit that is the authoritative source for an application's status and configuration is the leader application. There is always at most one leader at any given time. Unit agents will each seek to obtain leadership, and maintain it while they have it or wait for the current leader to drop out. The leader is denoted by an asterisk in the output to **`Juju status.`**
- **Relation** the inter-application connections are called relations, and they are formed by connecting the applications' endpoints.
- **Endpoint** is defined in a charm's **`metadata.yaml`** by the collection of three properties: a *role*, a *name*, and an *interface*.

  There are three types of roles:

  - **`requires`**: The endpoint might use services represented by another charm's endpoint over the given interface.
  - **`provides`**: The endpoint provides a service that another charm's endpoint can use over the given interface.
  - **`peers`**: The endpoint can coexist with another charm's endpoint in a peer-to-peer manner.
- **Agent** is the main Juju software that runs on each machine that it deploys. There are machine agent and a unit agent. A machine agent manages its own unit agents and any containers if such exists. The machine agent creates the unit agent. The unit agents are responsible for all charm related tasks. Agents track state changes, respond to those changes, and pass updated information back to the controller.

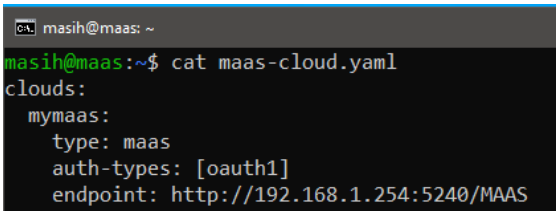
6.2 Setting up Juju Client

Before I can orchestrate Openstack on my MAAS nodes with Juju, I have to create the controller. But controller itself is just an administrative node of the MAAS cloud and to manage it, I need the client first.

```
sudo snap install juju --classic
```

6.3 Adding MAAS to Juju

To add MAAS to Juju I must add the following parameter to a file and then read those to Juju.



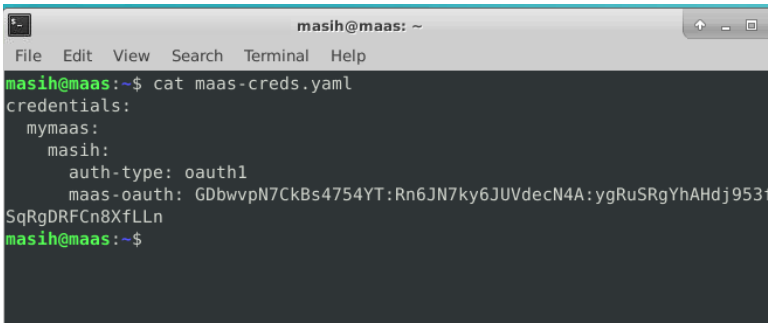
In the configuration above, the name '`mymaas`' is given to the cloud. The endpoint of the cloud is the IP address of my MAAS region controller. Next, `mymaas` cloud is added to Juju with:

```
juju add-cloud --client -f maas-cloud.yaml mymaas
```

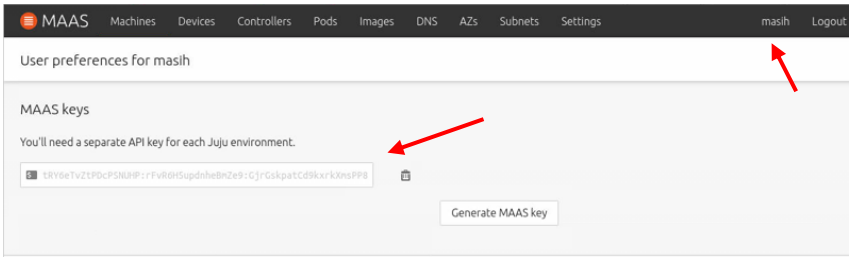
The **`--client`** tells juju that the cloud is on the same machine that the client is running on.

6.4 Adding the MAAS credentials to cloud

Next, I have to add my MAAS credentials so Juju can interact with it. I `create mymaas-creds.yaml` file and add the following to it.



In the configuration above, I specify the cloud name that the credential belongs to, the name of the user and the `maas-oauth` key which can be viewed from the user's profile in the web UI.



Credentials are passed to Juju by reading the Yaml file.

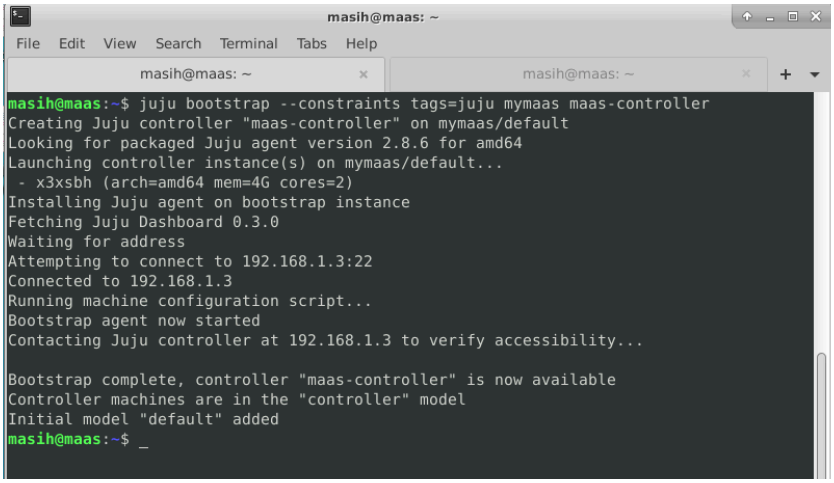
```
juju add-credential --client -f maas-creds.yaml mymaas
```

6.5 Creating the Juju controller

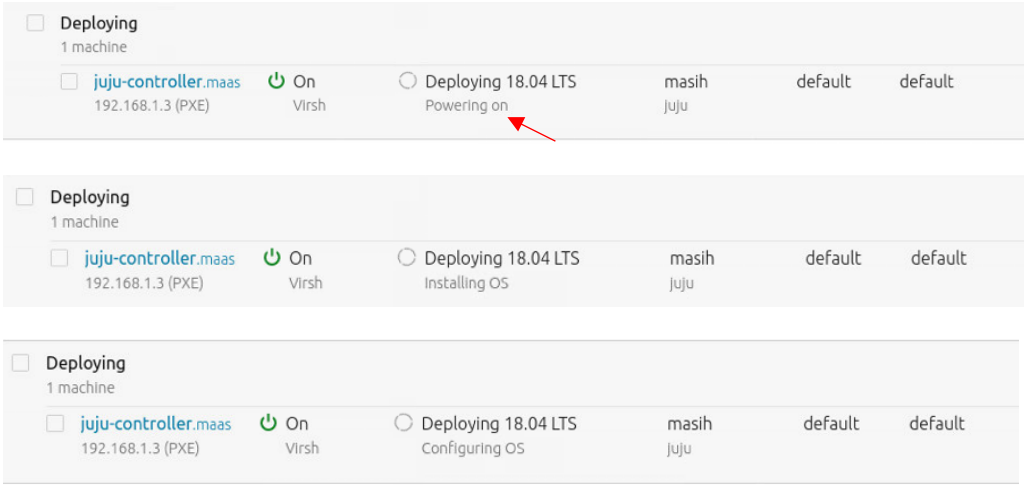
In the previous section, I added a cloud ‘mymaas’ to Juju client. Next, I create the controller for that cloud, and call it ‘maas-controller’:

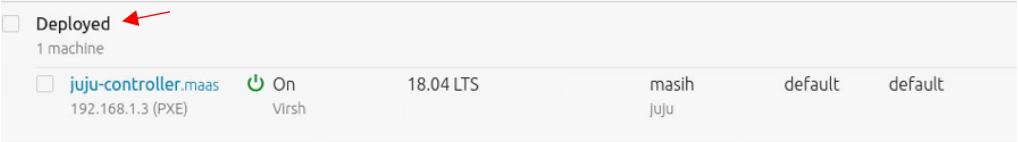
```
juju bootstrap --constraints tags=juju mymaas maas-controller
```

In the command above, I use **`--constraints tags`** to target the Juju node I created in the MAAS cluster. After executing the command, MAAS turns the juju-controller node on and Juju starts deploying an Ubuntu image to the node and it also installs and configures all the necessary software. Meanwhile, the Juju client on maas also updates the status to the terminal as shown below.



MAAS UI also shows the processes. The final result shows that 18.04LTS has been deployed on the Juju machine. This who process takes about 5 minutes.





6.6 Creating the Openstack model

The Openstack deployment will be placed in its own Juju model for organisational purposes. I create the model 'Openstack' and specify the desired series of 'focal' as below:

```
juju add-model --config default-series=focal Openstack
```

Choosing the correct series that matches the Openstack release is very important. The list is provided in Openstack charms release note. Also, all the machines must be running on the same series (bionic, focal etc) but not the mix. My nodes don't have an operating system but I will command Juju to deploy Focal series (Ubuntu 20.04 LTS) to all of them.

```
masih@maas:~$ juju add-model --config default-series=focal openstack
Added 'openstack' model on mymaas/default with credential 'masih' for user 'admin'
masih@maas:~$ _
```

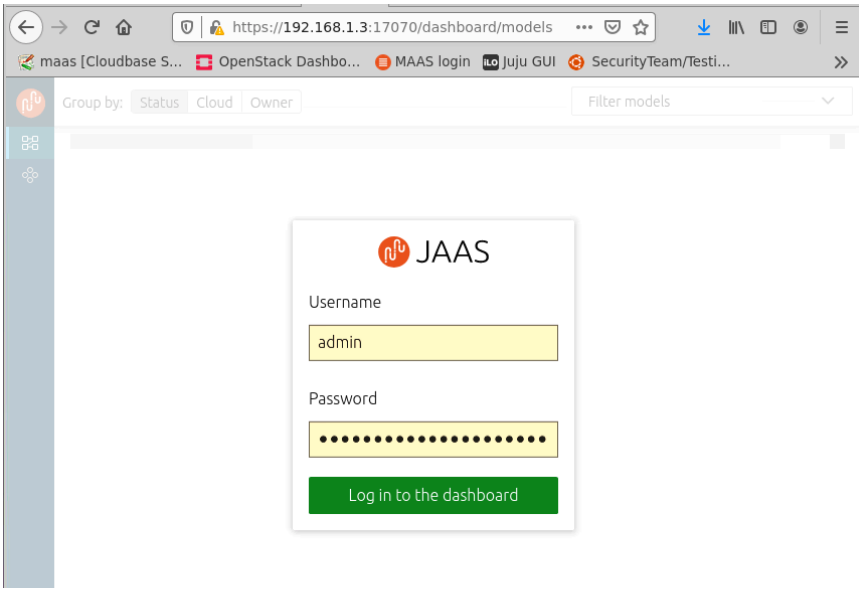
Running `Juju status` command will show the model, controller and the cloud that I just created.

```
masih@maas:~$ juju status
Model      Controller       Cloud/Region    Version  SLA          Timestamp
openstack  maas-controller  mymaas/default  2.8.6    unsupported  10:49:27Z

Model "admin/openstack" is empty.
```

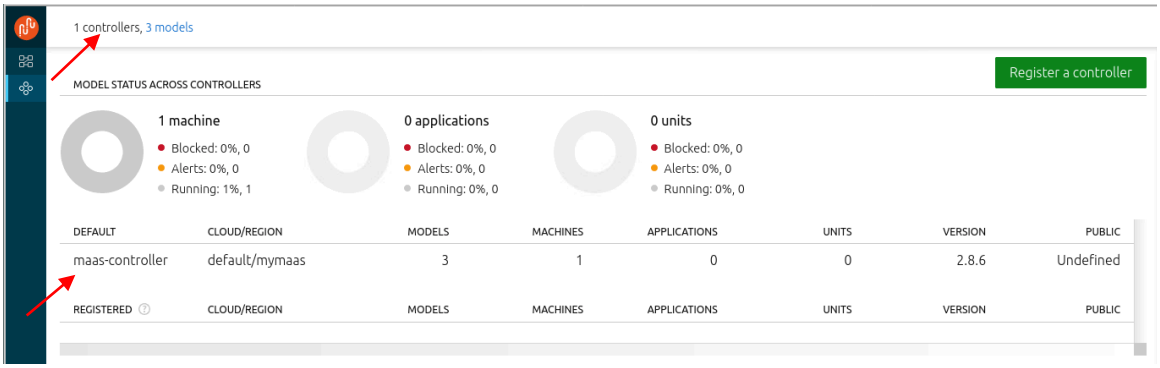
Now I can also access Juju UI. By executing the `Juju dashboard` command, all the necessary information is provided.

```
masih@maas:~$ juju dashboard
Dashboard 0.3.0 for controller "maas-controller" is enabled at:
  https://192.168.1.3:17070/dashboard
Your login credential is:
  username: admin
  password: f632199605f605125a5e336fbc7bea20
```

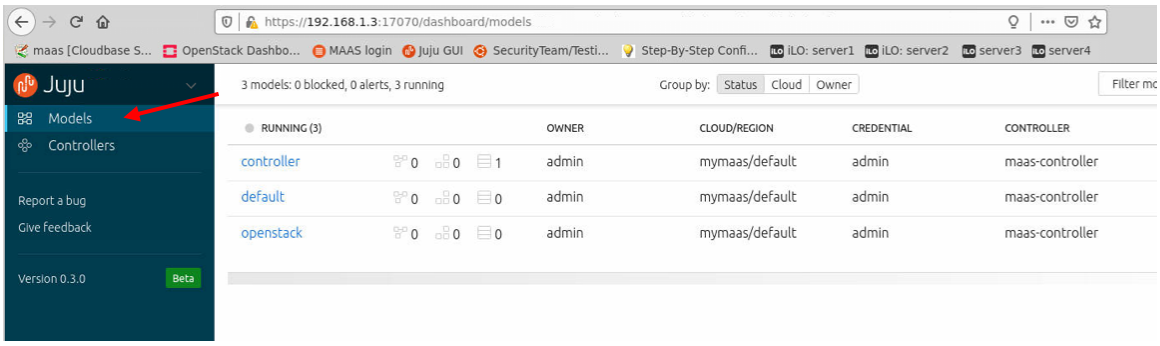
I can now log into JAAS service which stands for Juju as a Service.



There isn't much here just yet. The controller shows all the configurations that I have specified so far.



An empty model named 'default' is deployed automatically with Juju bootstrap.



7 Openstack

Installing Openstack with from Charms is done with two methods.

1. By individual charm. In this method, all configurations, installations and making relations are done manually. This gives a good insight into how juju works and also how

openstack components are related to each other. Below I will demonstrate this method.

2. By charm bundle. This method provides an automated means to install Openstack. All necessary commands and configurations are read from a YAML file.

Openstack provides a detailed guide for both methods and they constantly update their documentation as new recommendations, based on the reported bugs arrive. Therefore the instructions must be followed carefully. I will not write all the codes and the explanation provided in the Openstack guide. There are a few reasons for that. One is that the guide is very long but essentially follows these three concepts.

1- Deploying an application
2- Deploying into a specific machine
3- Managing relations

Another reason is that as new versions arrive, the commands and configurations might change which makes the current state obsolete.

7.1 OpenStack release

At the beginning of this project, I deployed the Openstack Train. But later when I was forced to do everything again, I installed a newer version, Openstack Victoria release atop Ubuntu 20.04 LTS (Focal) cloud nodes. Because of that, in the configuration I have to specify '**`cloud:focal-victoria`**' to access the right cloud archive. Some of the application are not part of the Openstack project and therefore don't need that configuration.

7.2 Installation process

As I will demonstrate soon, there are many moving parts involved in this deployment. Many of the application or units that I install require other condition to be satisfied first and therefore might throw an error or warning. But this is normal. I learn this in a hard way. Meaning that I would spend time on debugging while the situation was completely normal. Two main errors that will come up a lot are *missing relation* and *blocked* messages. But as said these will resolve later on when correct unit and relation arrive.

7.3 Deploy OpenStack

To start the deployment, first I switch to the correct controller with:

```
juju switch maas-controller:openstack
```

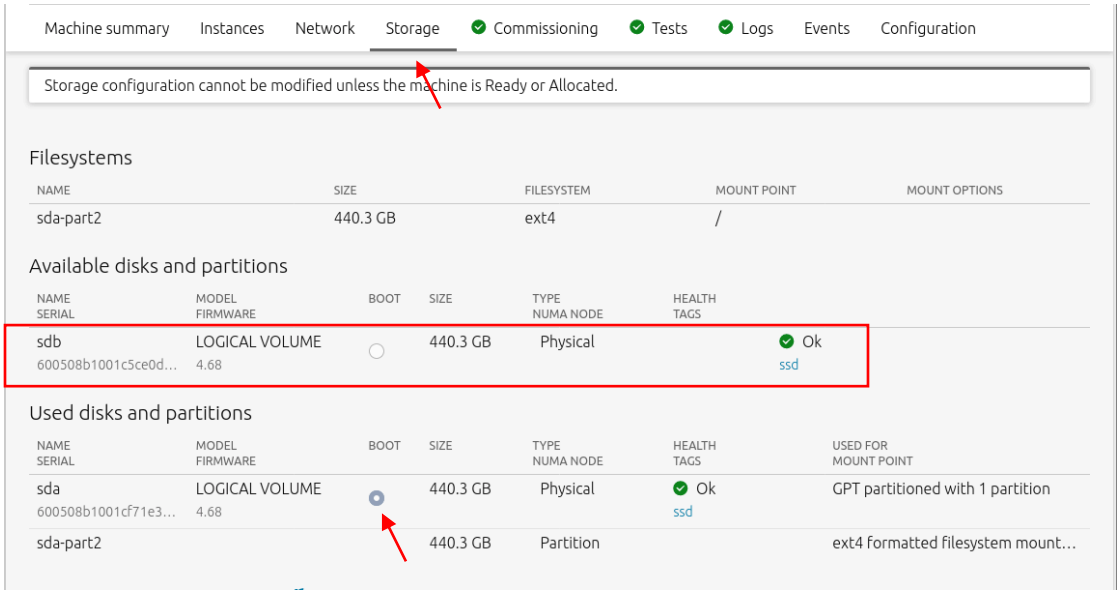
Now I will start adding OpenStack components to my 'openstack' model. They will be installed from the online Charm store and many will have configuration options specified via a YAML file.

7.3.1 Ceph OSD

This application will be installed on all the four nodes with the `ceph-osd` charm. For the configuration file, I create a **`ceph-osd.yaml`** file and add the following configuration to it.

```
ceph-osd:
  osd-devices: /dev/sdb
  source: cloud:focal-victoria
```

The name of the device must match that on the node. For example, all my nodes have two disks. **`/dev/sda`** and **`dev/sdb`**. MAAS has automatically assigned **`sda`** as a boot disk and has formatted the disk. Therefore, I will insert **`sdb`** into the configuration. If these names vary between the nodes all of the names should be specified in the configuration file.



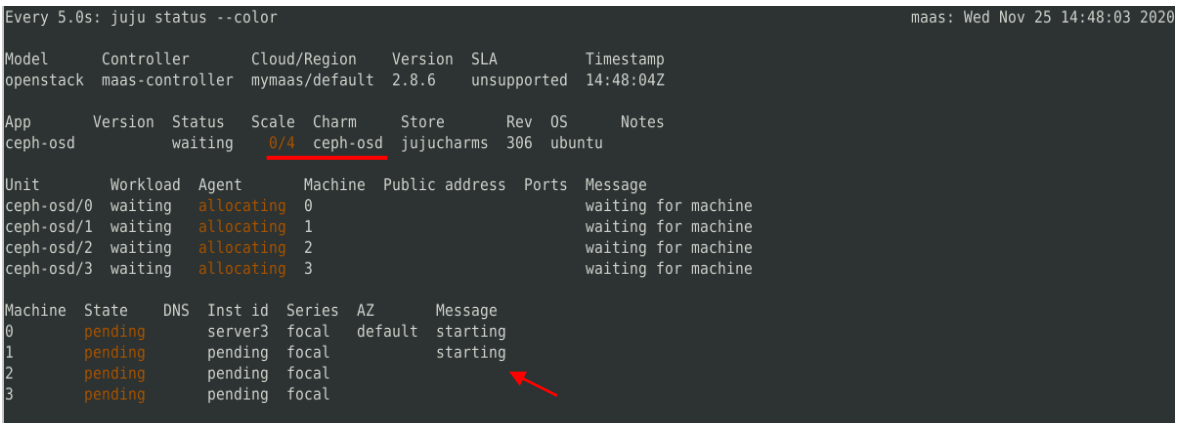
To deploy the application I use the compute tag to tell Juju which machines I want the application to be installed on.

```
juju deploy -n 4 --config ceph-osd.yaml --constraints tags=compute
ceph-osd
```

I can watch the whole process by running the Linux watch command on Juju status with:

```
watch -n 5 -c juju status --color
```

Which will show the status as the application are allocated, configured and deployed on the machines.



Juju turns on the 'compute' nodes and first installs an Ubuntu 20.04 on them and then deploys the Ceph unit to each one. There is no need to wait for each deployment and I can run the next command while other resources are being allocated and deployed. However, if there is any message in juju's status other than a missing relation, or a workload error, the situation must be investigated.

7.3.2 Nova compute

For nova-compute application, I first deploy it to one node with the `nova-compute` charm. Then I'll scale it up to two other machines. First I create the **`nova-compute.yaml`** with the following configuration:

```
nova-compute:
  enable-live-migration: true
  enable-resize: true
  migration-auth-type: ssh
  openstack-origin: cloud:focal-victoria
```

Because there are no more free Juju machines (MAAS nodes) available, the initial node will be targeted by machine. This means multiple services are placed on one node. The machines 1, 2, and 3 are chosen here:

```
juju deploy -n 3 --to 1,2,3 --config nova-compute.yaml nova-compute
```

7.3.3 MySQL InnoDB Cluster

MySQL InnoDB Cluster always requires at least three database units. They will be containerized on machines 0, 1, and 2:

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 mysql-innodb-cluster
```

7.3.4 Vault

Vault is necessary for managing the TLS certificates that will enable encrypted communication between cloud applications. Vault requires further configuration and without them, the process will not continue. First I will containerised Vault on machine 3:

```
juju deploy --to lxd:3 vault
```

This is the first application that I will relate to another application. In this case, I will make a relation between Vault and mysql.

The process is as follows:

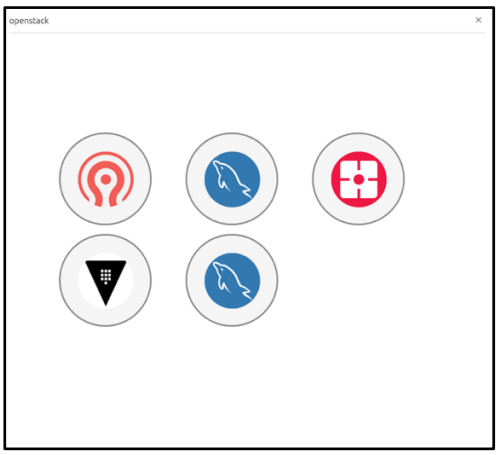
1. create an application-specific instance of mysql-router (a subordinate)
2. add a relation between that mysql-router instance and the database
3. add a relation between the application and the mysql-router instance

The combination of steps 2 and 3 joins the application to the cloud database.

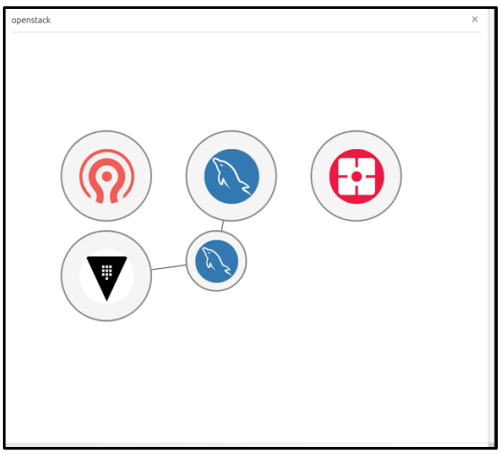
Corresponding commands for Vault are:

```
juju deploy mysql-router vault-mysql-router
juju add-relation vault-mysql-router:db-router mysql-innodb-clus-
ter:db-router
juju add-relation vault-mysql-router:shared-db vault:shared-db
```

The whole status is also reported and visualized in the Juju UI.



The applications added before any relation



The view after the relation is made

As shown below the Vault needs to be initialized message is blocking the unit. Other blocked statuses are normal because they are simply just missing relation.

```
App                   Version  Status   Scale  Charm                Store       Rev  OS      Notes
ceph-osd              15.2.5   blocked      4  ceph-osd             jujucharms  306  ubuntu
mysql-innodb-cluster  8.0.22   active       3  mysql-innodb-cluster jujucharms    3  ubuntu
nova-compute          22.0.0   blocked      3  nova-compute         jujucharms  323  ubuntu
vault                 1.5.4    blocked      1  vault                jujucharms   41  ubuntu
vault-mysql-router    8.0.22   active       1  mysql-router         jujucharms    4  ubuntu

Unit                     Workload  Agent  Machine  Public address  Ports     Message
ceph-osd/0*              blocked   idle   0        192.168.1.6               Missing relation: monitor
ceph-osd/1               blocked   idle   1        192.168.1.4               Missing relation: monitor
ceph-osd/2               blocked   idle   2        192.168.1.7               Missing relation: monitor
ceph-osd/3               blocked   idle   3        192.168.1.5               Missing relation: monitor
mysql-innodb-cluster/0*  active    idle   0/lxd/0  192.168.1.58              Unit is ready: Mode: R/W
mysql-innodb-cluster/1   active    idle   1/lxd/0  192.168.1.56              Unit is ready: Mode: R/O
mysql-innodb-cluster/2   active    idle   2/lxd/0  192.168.1.254             Unit is ready: Mode: R/O
nova-compute/0*          blocked   idle   1        192.168.1.4               Missing relations: image, messaging
nova-compute/1           blocked   idle   2        192.168.1.7               Missing relations: image, messaging
nova-compute/2           blocked   idle   3        192.168.1.5               Missing relations: messaging, image
vault/0*                 blocked   idle   3/lxd/0  192.168.1.71   8200/tcp  Vault needs to be initialized
  vault-mysql-router/0*  active    idle            192.168.1.71              Unit is ready
```

Juju status (The IP addresses here are from the previous deployment and will change in future images)

I will skip the initialization and unsealing process of Vaul. The steps are explained clearly here. After the initialization the vault becomes active.

```
Unit                     Workload  Agent  Machine  Public address  Ports     Message
ceph-osd/0*              blocked   idle   0        192.168.1.5               Missing relation: monitor
ceph-osd/1               blocked   idle   1        192.168.1.6               Missing relation: monitor
ceph-osd/2               blocked   idle   2        192.168.1.7               Missing relation: monitor
ceph-osd/3               blocked   idle   3        192.168.1.4               Missing relation: monitor
mysql-innodb-cluster/0   active    idle   0/lxd/0  192.168.1.201             Unit is ready: Mode: R/O
mysql-innodb-cluster/1*  active    idle   1/lxd/0  192.168.1.202             Unit is ready: Mode: R/W
mysql-innodb-cluster/2   active    idle   2/lxd/0  192.168.1.203             Unit is ready: Mode: R/O
nova-compute/0*          blocked   idle   1        192.168.1.6               Missing relations: image, messaging
nova-compute/1           blocked   idle   2        192.168.1.7               Missing relations: messaging, image
nova-compute/2           blocked   idle   3        192.168.1.4               Missing relations: messaging, image
vault/0*                 active    idle   3/lxd/0  192.168.1.204   8200/tcp  Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*  active    idle            192.168.1.204             Unit is ready
```

7.3.5 Neutron networking

Neutron networking is implemented with four applications:

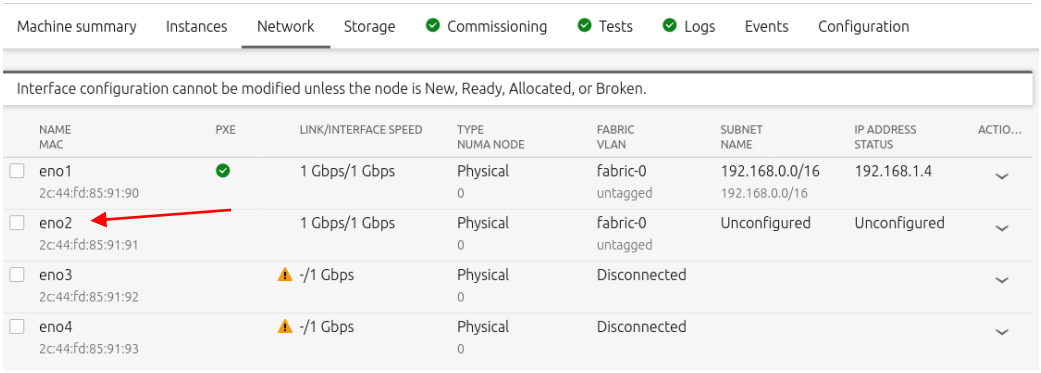
- neutron-api

- neutron-api-plugin-ovn (subordinate)
- ovn-central
- ovn-chassis (subordinate)

File **neutron.yaml** contains the configuration necessary for three of them:

```
ovn-chassis:
  bridge-interface-mappings: br-ex:eno2
  ovn-bridge-mappings: physnet1:br-ex
neutron-api:
  neutron-security-groups: true
  flat-network-providers: physnet1
  openstack-origin: cloud:focal-victoria
ovn-central:
  source: cloud:focal-victoria
```

The `bridge-interface-mapping` must point to the second interface of the node which in my case is `eno2`.



The HP Proliant have four NICs of which two I have used in this project. The first NIC is for the node's normal network processes and the second one will be dedicated to Openstack environment. The interface details are giving in Network tab of the node. Another important parameter in the configuration above is the `flat-network-providers: physnet1.` I need the name of the flat network when I set up the public network for Openstack.

The main OVN application is `ovn-central` and I need three units of it containerised, on machines 0, 1, and 2:

```
juju deploy -n 3 --to lxd:0,lxd:1,lxd:2 --config neutron.yaml ovn-central
```

The neutron-api application will be containerised on machine 1:

```
juju deploy --to lxd:1 --config neutron.yaml neutron-api
```

Deploy the subordinate charm applications:

```
juju deploy neutron-api-plugin-ovn
juju deploy --config neutron.yaml ovn-chassis
```

Adding the necessary relations with:

```
juju add-relation neutron-api-plugin-ovn:neutron-plugin neutron-api:neutron-plugin-api-subordinate
juju add-relation neutron-api-plugin-ovn:ovsdb-cms ovn-central:ovsdb-cms
juju add-relation ovn-chassis:ovsdb ovn-central:ovsdb
juju add-relation ovn-chassis:nova-compute nova-compute:neutron-plugin
juju add-relation neutron-api:certificates vault:certificates
juju add-relation neutron-api-plugin-ovn:certificates vault:certificates
juju add-relation ovn-central:certificates vault:certificates
juju add-relation ovn-chassis:certificates vault:certificates
```

Join neutron-api to the cloud database:

```
juju deploy mysql-router neutron-api-mysql-router
juju add-relation neutron-api-mysql-router:db-router mysql-innodb-cluster:db-router
juju add-relation neutron-api-mysql-router:shared-db neutron-api:shared-db
```

After Neutron, the following components will be added which I will skip because the same code provided in Openstack guide will be used.

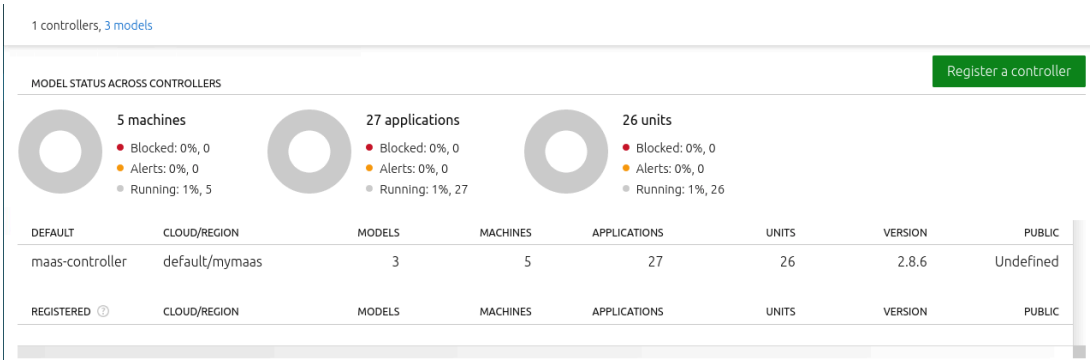
- Keystone
- RabbitMQ
- Nova cloud controller
- Placement
- OpenStack dashboard
- Glance
- Ceph monitor
- Cinder
- Ceph RADOS Gateway
- NTP

Once all commands are sent to the controller, it takes quite some time before everything

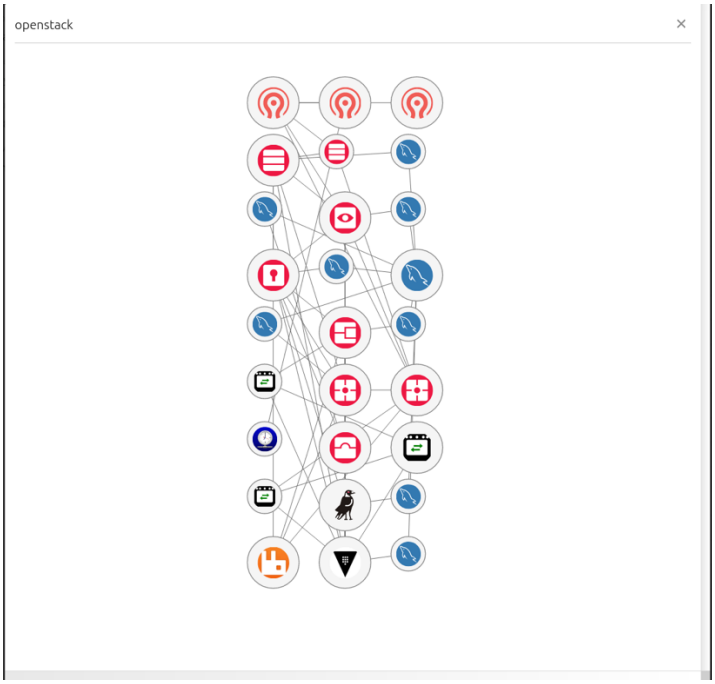
is ready. Sometimes it feels that nothing is happening and everything has stopped, but eventually, Juju will deploy applications and statuses will be updated. When Juju has deployed the last application and configurations are ready, the status shows all workloads as active without any red blocked messages.

```
Unit                          Workload  Agent  Machine  Public address  Ports              Message
ceph-mon/0*                   active    idle   0/lxd/3  192.168.1.215                     Unit is ready and clustered
ceph-mon/1                    active    idle   1/lxd/3  192.168.1.217                     Unit is ready and clustered
ceph-mon/2                    active    idle   2/lxd/4  192.168.1.216                     Unit is ready and clustered
ceph-osd/0*                   active    idle   0        192.168.1.5                       Unit is ready (1 OSD)
  ntp/1*                      active    idle            192.168.1.5      123/udp          chrony: Ready
ceph-osd/1                    active    idle   1        192.168.1.6                       Unit is ready (1 OSD)
  ntp/2                       active    idle            192.168.1.6      123/udp          chrony: Ready
ceph-osd/2                    active    idle   2        192.168.1.7                       Unit is ready (1 OSD)
  ntp/0                       active    idle            192.168.1.7      123/udp          chrony: Ready
ceph-osd/3                    active    idle   3        192.168.1.4                       Unit is ready (1 OSD)
  ntp/3                       active    idle            192.168.1.4      123/udp          chrony: Ready
ceph-radosgw/0*               active    idle   0/lxd/4  192.168.1.219    80/tcp           Unit is ready
cinder/0*                     active    idle   1/lxd/4  192.168.1.218    8776/tcp         Unit is ready
  cinder-ceph/0*              active    idle            192.168.1.218                     Unit is ready
  cinder-mysql-router/0*      active    idle            192.168.1.218                     Unit is ready
glance/0*                     active    idle   3/lxd/3  192.168.1.214    9292/tcp         Unit is ready
  glance-mysql-router/0*      active    idle            192.168.1.214                     Unit is ready
keystone/0*                   active    idle   0/lxd/2  192.168.1.209    5000/tcp         Unit is ready
  keystone-mysql-router/0*    active    idle            192.168.1.209                     Unit is ready
mysql-innodb-cluster/0        active    idle   0/lxd/0  192.168.1.201                     Unit is ready: Mode: R/O
mysql-innodb-cluster/1*       active    idle   1/lxd/0  192.168.1.202                     Unit is ready: Mode: R/W
mysql-innodb-cluster/2        active    idle   2/lxd/0  192.168.1.203                     Unit is ready: Mode: R/O
neutron-api/0*                active    idle   1/lxd/2  192.168.1.208    9696/tcp         Unit is ready
  neutron-api-mysql-router/0* active    idle            192.168.1.208                     Unit is ready
  neutron-api-plugin-ovn/0*   active    idle            192.168.1.208                     Unit is ready
nova-cloud-controller/0*      active    idle   3/lxd/1  192.168.1.211    8774/tcp,8775/tcp  Unit is ready
  ncc-mysql-router/0*         active    idle            192.168.1.211                     Unit is ready
nova-compute/0*               active    idle   1        192.168.1.6                       Unit is ready
  ovn-chassis/1               active    idle            192.168.1.6                       Unit is ready
nova-compute/1                active    idle   2        192.168.1.7                       Unit is ready
  ovn-chassis/2*              active    idle            192.168.1.7                       Unit is ready
nova-compute/2                active    idle   3        192.168.1.4                       Unit is ready
  ovn-chassis/0               active    idle            192.168.1.4                       Unit is ready
openstack-dashboard/0*        active    idle   2/lxd/3  192.168.1.213    80/tcp,443/tcp   Unit is ready
  dashboard-mysql-router/0*   active    idle            192.168.1.213                     Unit is ready
ovn-central/0*                active    idle   0/lxd/1  192.168.1.205    6641/tcp,6642/tcp  Unit is ready (leader: ovnnb_db, ovnsb_db northd: active)
ovn-central/1                 active    idle   1/lxd/1  192.168.1.207    6641/tcp,6642/tcp  Unit is ready
ovn-central/2                 active    idle   2/lxd/1  192.168.1.206    6641/tcp,6642/tcp  Unit is ready
placement/0*                  active    idle   3/lxd/2  192.168.1.212    8778/tcp         Unit is ready
  placement-mysql-router/0*   active    idle            192.168.1.212                     Unit is ready
rabbitmq-server/0*            active    idle   2/lxd/2  192.168.1.210    5672/tcp         Unit is ready
vault/0*                      active    idle   3/lxd/0  192.168.1.204    8200/tcp         Unit is ready (active: true, mlock: disabled)
  vault-mysql-router/0*       active    idle            192.168.1.204                     Unit is ready
```

The controller UI populates cloud the model as new applications are installed. The image below shows the details of the controller.



The image of the relations is built as well and the final result looks like this.



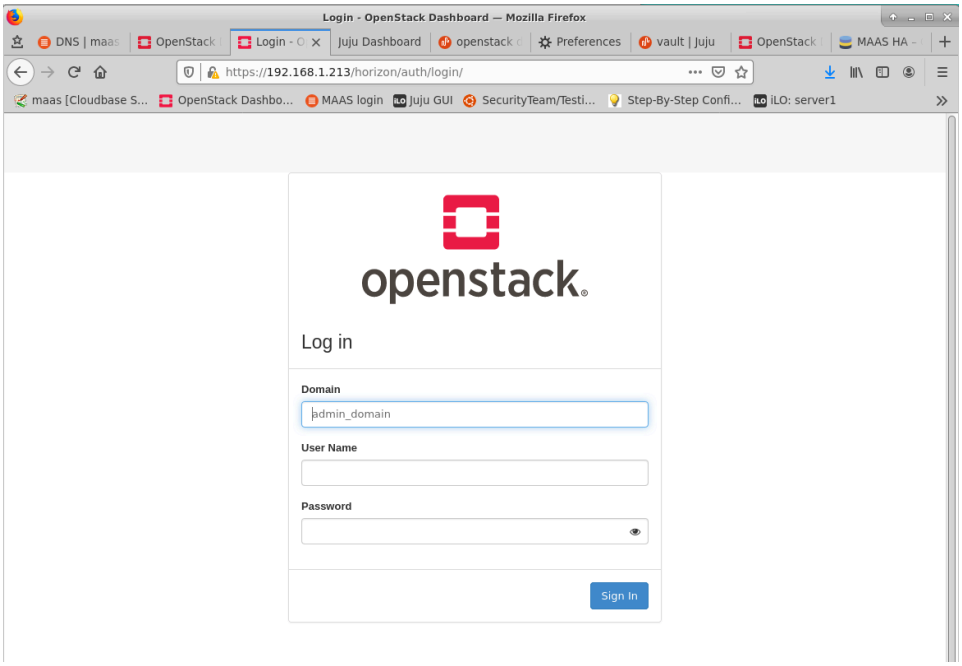
The URL of the Openstack dashboard is openstack-dashboard unit's IP address which can also be extracted with:

```
juju status --format=yaml openstack-dashboard | grep public-address | awk\
'{print $2}' | head -1
```

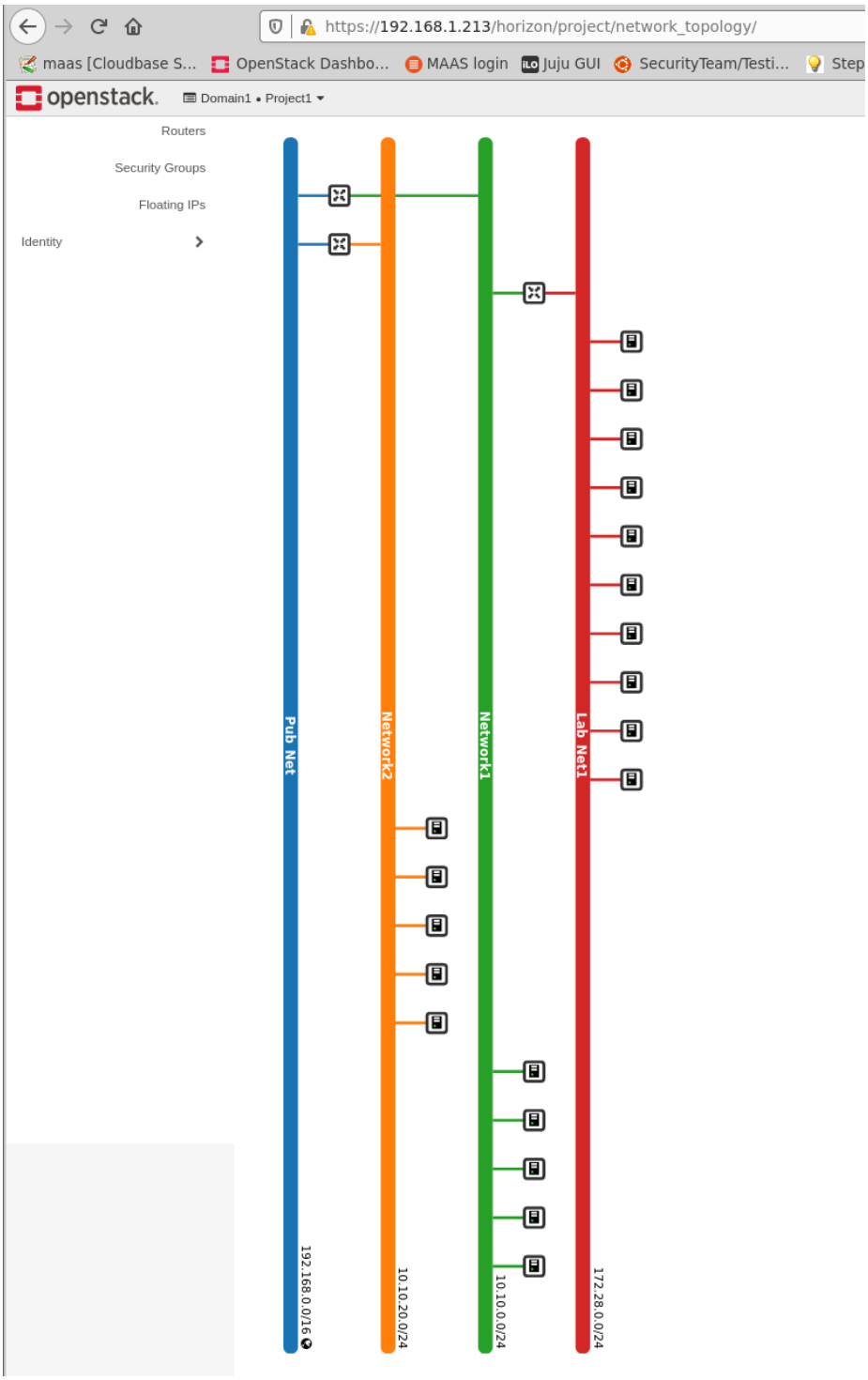
```
masih@maas:~$ juju status --format=yaml openstack-dashboard | grep public-address | awk '{print $2}' | head -1
192.168.1.213
```

And the password with this:

```
juju run --unit keystone/0 leader-get admin_passwd
```



Now that enough resources are available, I can create networks with multiple subnets and instances running on them. The diagram below shows two subnets Network1 and 2 and a Lab subnet for Network1.



8 Conclusions

I’m glad I chose this deployment path because I learned two new tools, MAAS and Juju, besides Openstack. I learned very well how to operate MAAS. I find MAAS to be very practical for data centers and cloud projects. Working with it requires a good understanding of computer infrastructure and networking. Canonical documentation for MAAS covers basic configurations but for a more demanding scenarios, a bit of investigation and imagination is required.

Juju is very powerful and much more complex software. It is believed that it will change the way configuration management is done in the future. I had elementary experience with Salt which is also believed to be one of the fastest and a simple tool for configuration management. But Juju changed my perspective on this completely. Juju does multiple provisioning, configuration management and deployment, all at once. I learned the main concept of Juju as well, but because configuration management and Juju itself are vast topics, a lot left to be learned.

Most of the challenges I faced in this project came from the preparation of the environment which required thorough planning and a decent understanding of virtualisation and networking. The deployment of Openstack with Juju was not difficult per se, but it requires a good understanding of all the moving parts from hardware to networking, virtualisation, cloud computing and configuration management. But this project indeed helped me to improve my knowledge on all those topics.

Openstack foundation and Openstack-charmers community provide excellent documentation and without reading them, it is very difficult if not impossible to succeed.

This was a large and complex project but at the same time extremely educational. It took my knowledge of cloud infrastructure and configuration management to a new level. It was very unfortunate that I did not have the time to explore the capabilities of Openstack as it was initially set to be the main goal of the project. Nonetheless, because I worked with MAAS and Juju and had the opportunity to learn two new platforms and many new concepts, I’m happy the turns this project took.

Reference:

1. https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/install-openstack.html
2. https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/install-Juju.html
3. https://Juju.is/docs/concepts-and-terms
4. https://maas.io/how-it-works
5. https://maas.io/docs/snap/2.7/ui/maas-documentation
6. https://docs.openstack.org/project-deploy-guide/charm-deployment-guide/latest/app-vault.html