S16&S17: ETSI-OSM RELEASE FOUR: Controlling OSM with Juju Charms	uc3m | Universidad Carlos III de Madrid
Network Function Virtualisation(NFV) Last update: February 25, 2019	2018-2019

Introduction and Objectives

The goal of this laboratory is to show how the ETSI Open Source MANO (OSM) framework is controlled with Juju Charms. This lab will review the main concepts and workflows to setup a Network Service (NS) and to control it using Juju Charms. This lab covers all the configuration phases of NS. It is tightly coupled and builds upon the experience *and work* you did in the first Network Function Virtualisation (NFV) session.

1. Requirements

As before any laboratory session, read the short guide (presentation) describing the initial steps to configure the laptop and the multiport for your group. It also contains a table with the IP addresses you will be using.

This lab builds on the knowledge your acquired in the first NFV laboratory **as well as on the materials you created there**. Refer to the guide provided for that lab as a reference as well as to the guide for the previous NFV laboratory for **Juju** concepts.

Basic knowledge of the Linux OS (commands and environment) is needed to accomplish this laboratory. We will be using Ubuntu Linux 16.04 LTS in all the environments. [1] or [2] provide a good reference for the different tasks you must accomplish.

Hint : Be prepared to create Bash shell scripts to automate some of the processes described in the guide. This will speed up the process and help you avoid boring, repetitive and error-prone tasks.

2. Guidelines

Read this guide carefully and completely before starting the laboratory. This laboratory is intended to be done in groups of 2 students.

If you have questions, please ask them to the lab teachers.

This laboratory guide includes *milestones* that will help you monitor your progress and **checkpoints** that will be evaluated by the lab professor. Collect evidence you reached the milestones (*e.g.* , files you used, screenshots, *etc.*) and call the lab teacher once you reach a checkpoint. Be ready to answer questions about the milestones you have accomplished to reach the checkpoint.

3. Work environment

The work environment is composed by a laptop, a multiport and an Ethernet switch to connect them to the laboratory network. The laptop and the multiport have the basic environment ready to work. Additional files are posted in the laboratory file server, which can be accessed at `http://172.16.24.3`. Download them to the `$HOME/Downloads` in the laptop (a good time to do it is while the Openstack environment is booting).

4. Extras

Bring a USB storage device and keep a backup of all files you create, as you did in the first NFV laboratory session (S12&S13). Use the files you created in that session as the starting point for your work in this session.

Check-point	MS	Task	Est. time	Mark
	MS1	Devstack as an OSM Virtual Infrastructure Manager (VIM)		
Chpkt1	MS2	OSM Juju base arch working	30-45 min	2 pts
	MS3	Day-0 conf for simple charm		
	MS4	Simple charm		
	MS5	Simple charm VNFD		
Chkpt 2		Simple charm deployed and working	1 h	3 pts
	MS 6	CLI commands for the video server		
Chkpt 3		ffmpeg charm ready		1 pt
Chkpt 4		ffmpeg VNFD		1 pt
Chkpt 5		ffmpeg NSD		1 pt
Chkpt 6		ffmpeg NS demonstarted	2h 15min	2 pts

Table 1.: Laboratory structure, timings and evaluation

Estimated duration of the laboratory: 4 hours.

1. OSM RELEASE FOUR: a reminder

Open Source MANO (OSM) implements the ETSI NFV Management and Orchestration (MANO) specification. As mentioned in the first NFV laboratory session, we use Release FOUR [3], which is more mature than the latest Release FIVE [4]. It is open sourced and can be downloaded from `https://osm.etsi.org/wikipub/index.php/OSM_Release_FOUR`. A technical white paper describing some of the internals of OSM Release FOUR is available at [5].

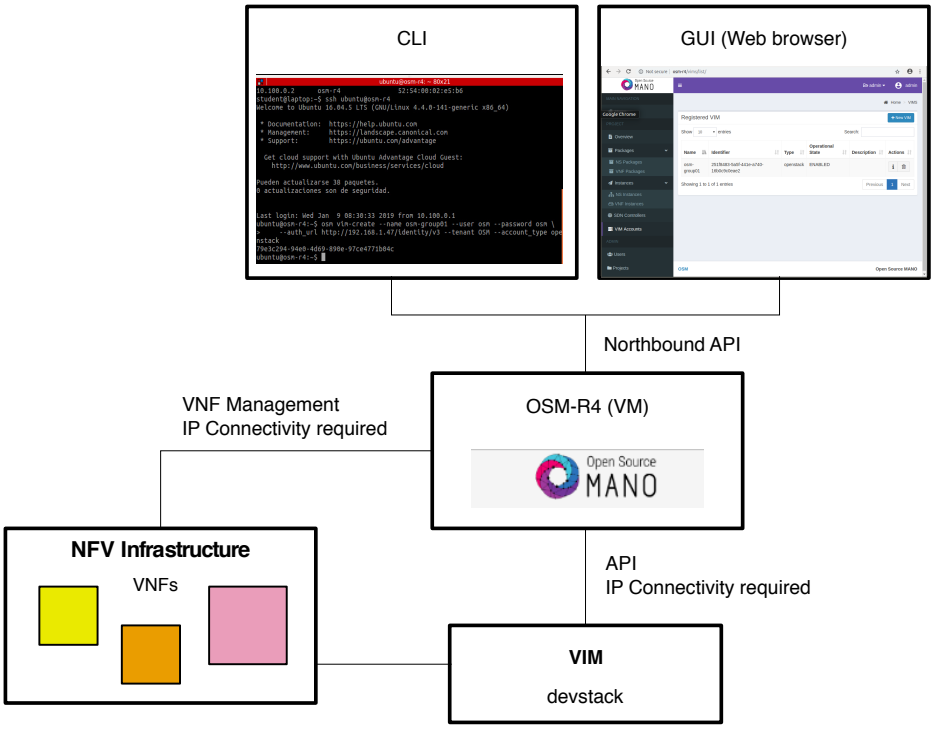


Figure 1.1.: The ETSI-OSMR4 architecture

Figure 1.1 shows the interactions between OSM, the VIM and the NFV Infrastructure (NFVI). In this lab, we will be using `devstack` as the VIM.

Accessing OSM through a Web browser based Graphical User Interface (GUI) or using a Command Line Interface (CLI) as covered in the first NFV laboratory. As mentioned in that laboratory, OSM also needs IP connectivity to the deployed Virtual Network Functions (VNFs) when (Day-0, 1 and 2) configuration primitives are used to setup the deployed NSes using Juju charms.

In this lab, we will concentrate on configuration automation, *i.e.* the interaction between OSM and the VNFs is based on **Juju**, as shown in Figure 1.2. OSM R4 uses *proxy charms* that are installed in the same machine that executes OSM (*i.e.* in our case the `osm-r4` Virtual Machine (VM)) as an LinuX Containers (LXC) container) that executes commands *remotely* using SSH. In OSM-R4, the charm is responsible for Day-1 and Day-2 configuration.

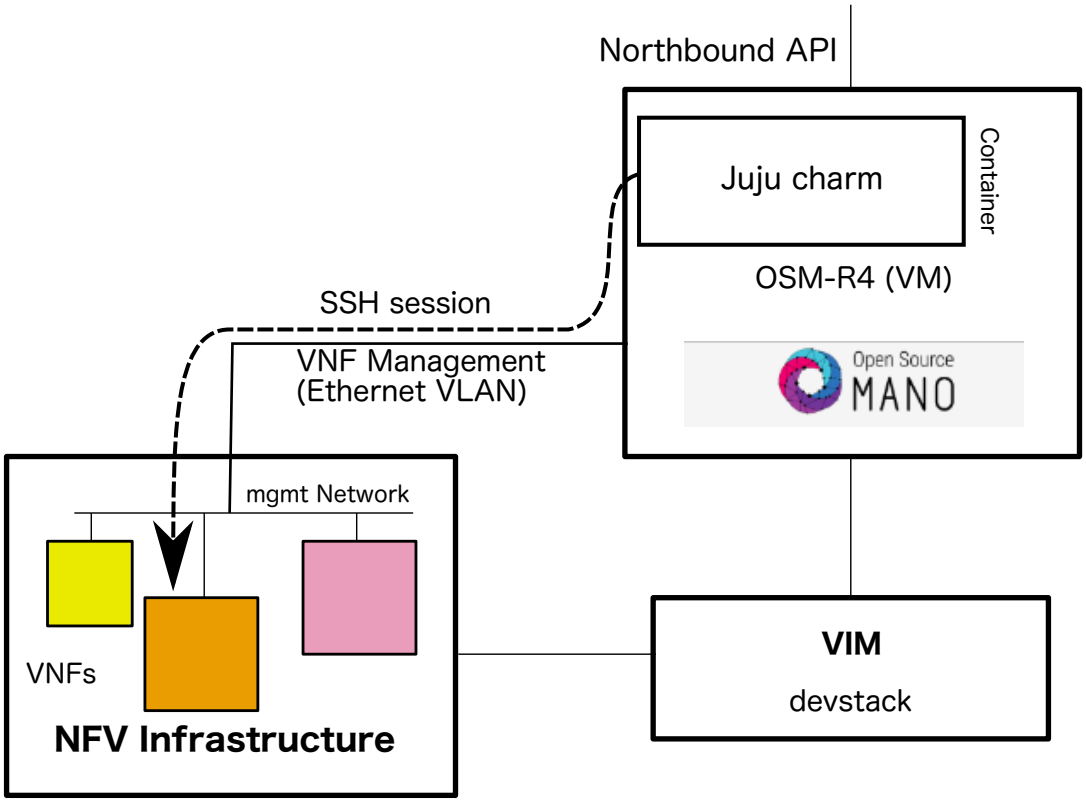


Figure 1.2.: Interaction between the OSM components when Juju charms are used

2. Setting up the OSM/OpenStack environment

estimated duration: 30min-45min

In order to start working with OSM, you will need to setup the devstack environment. The objective is to setup the environment shown in Figure 1.2. A more detailed view of all the components and technologies involved, including the networking details is the following:

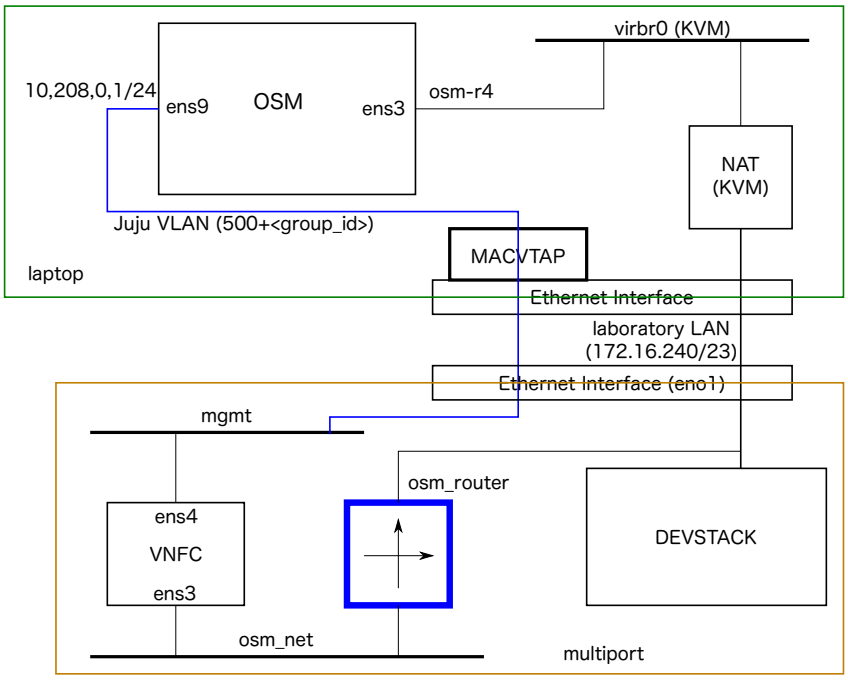


Figure 2.1.: Detailed OSM/Juju architecture

2.1. Booting the OSM VM and setting up the OpenStack VIM

For this first part, you need the laboratory guide and the guide you used for the first session lab. Boot the multiport and laptop as described in the guide. Configure them and start devstack on the multiport.

Remember that OSM is provided as a Virtual Machine. The VM name is `osm-r4` and you will be able to access it using the VM name as host name. Name resolution in the laptop is configured to allow this. As in the Kubernetes lab, the user to access the VM is `ubuntu` and **great care should be taken not to tamper with the SSH keys, since VM access is based on a pre-shared SSH key.**

While devstack boots, we will check the `osm` VM's configuration and start it on the laptop.

1. The first step is to check that the VM has two Ethernet interfaces and that they are correctly configured. Use `virt-manager` and check that
   - the first interface is connected to `virbr0: NAT`
   - the second interface is connected to a MACVTAP interface that is connected to the laptop's Ethernet interface (as shown in Figure 2.2)

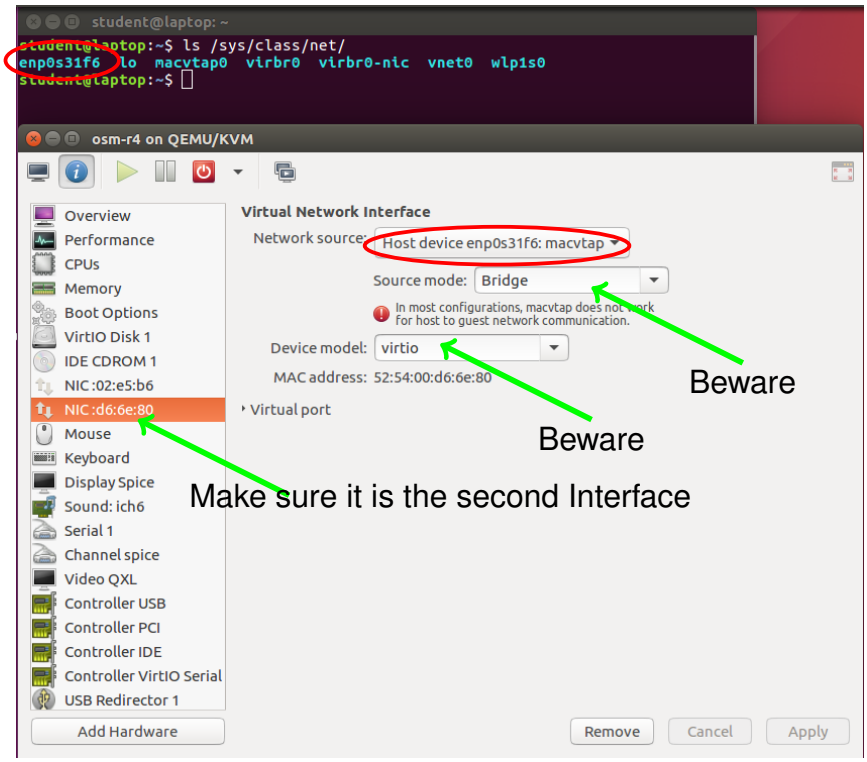


Figure 2.2.: VLAN interface settings

2. Start the `osm-r4` VM (using `virsh` or `virt-manager`).
3. Open a terminal (pane) and check that the OSM VM has started. Access it and setup the VLAN for the management "backplane"

```
virt-hosts
(osm-r4 should have MAC and IP addresses assigned)

ssh ubuntu@osm-r4
sudo bash osm-r4-lan.sh <group>
```

Hint : keep the VLAN id assigned to the management backplane for reference when you configure the devstack VIM and exit the VM. <

Download the contents of the OSM-Lab3 directory in the lab file server to the `${HOME}/Downloads` directory in your laptop.

Setting up the devstack VIM

Following the steps in the devstack guide, access the devstack GUI with Google Chrome.

1. As devstack admin:
   - Create a devstack user `osm` (password: `osm`)
   - Create a tenant `OSM`[1] with user `osm` as admininstrator
2. For the `OSM` tenant
   - Create a network and a subnet with the following names: `osm-net`, `osm-subnet` and IP addressing on prefix `10.10.100+<group>.0/24`
   - Create a router called `osm-router`, that connects networks `osm-subnet` and `public`
   - Create a key pair called `osm-key`
   - Download the Openstack RC v3 for the `OSM` tenant to the laptop
     HINT: browse through the file and identify the authorisation URL. You will use it when you set up your devstack deployment as a VIM using the OSM CLI.
   - Modify the default security group to allow ICMP and ssh
3. On the laptop, open a terminal and source the RC file you just downloaded.
4. Using the VLAN ID you got when configuring the `osm-r4` VM, execute the following commands to create the 'mgmt' network needed by the Juju charms:

```
neutron net-create mgmt --provider:network_type=vlan \
  --provider:physical_network=public \
  --provider:segmentation_id=<VLAN from osm-r4-juju.sh> \
  --shared

neutron subnet-create --name subnet-mgmt mgmt 10.208.0.0/24 \
  --allocation-pool start=10.208.0.2,end=10.208.0.254
```

5. Register the devstack VIM from the same console you interact with devstack to reuse the environment set by `OSM_openrc.sh`:

```
source ~/Downloads/OSM-openrc.sh
osm vim-create --name osm-group<XY> \
  --user osm --password osm --tenant OSM \
```

[1] upper and lower case in user, password and tenant are **significant**

```
  --auth_url $OS_AUTH_URL --account_type openstack \
  --config='{security_groups: default,
    keypair: osm-key}'
osm vim-list
```

Note that you should not enable floating IP address assignment by default. When needed, floating IP addresses will be assigned to *specific* VMs.

6. Open a new Google Chrome browser tab and log into the OSM GUI using the following URL and credentials

URL: `http://osm-r4/` account `*admin/admin*`

7. Go the the *VIM Accounts* tab and check that your devstack VIM appears.

Milestone 1: devstack is prepared to interact with OSM as a VIM.

2.2. Checking that the Juju backplane for OSM is ready

In order to isolate the different groups in the lab, each group is using a different VLAN for the Juju backplane. To check that it is correctly configured, launch Wireshark and start a capture on the laptop's Ethernet interface, filtering for the `arp` packets. Enter the OSM VM and ping any address in the range 10.208.0.3 to 10.208.0.254. You should see `arp` packets as shown in Figure 2.3.

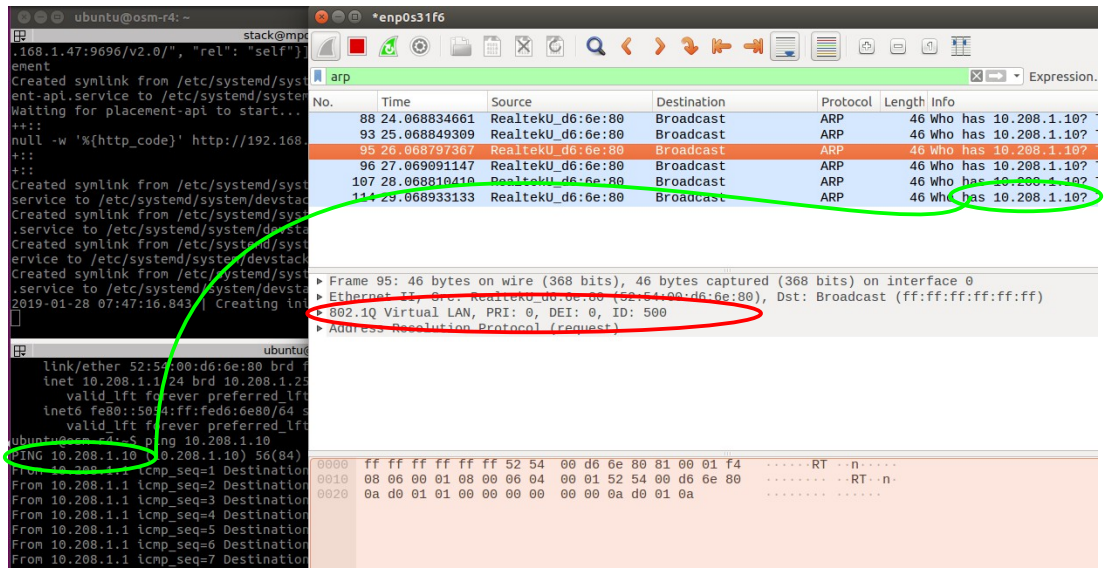


Figure 2.3.: Checking the Juju VLAN

Note that the VLAN ID that appears in your capture should be the VLAN id you indicated when you created the `mgmt` network. The VLAN_ID should be

$$VLAN\_ID = 500 + group\_id$$

If other groups are performing this test at the same time as you, you will also capture their traffic. Check that **your** traffic appears in the capture.

Milestone 2: The `mgmt` network is visible at the Ethernet port of the laptop, it uses a 802.1q VLAN, and the VLAN ID corresponds to your group.

Checkpoint 1: (2 points) The setup is ready: devstack is configured as the VIM for OSM and the Juju backplane is ready.

3. Creating a Network Service with a simple Juju charm

estimated duration: 1h

The objective of this chapter is to create a simple Network Service comprised by one Virtual Network Function with one VNF Component. We use charms to configure the Network Service at the Virtual Network Function level. VNFs composed by more than one VNF Component (VNFC) are out-of-scope in this laboratory.

We will cover the three configuration phases for VNFs:

Phase	When	Where
Day-0	Static configuration	In the VNF Descriptor (VNFD) using, for example, `cloud-config`
Day-1	Initial configuration	In the VNFD using a charm marked as **initial** configuration charms
Day-2	During the life-time of the NS	In the VNFD using charms marked as configuration charms

Table 3.1.: Configuration phases

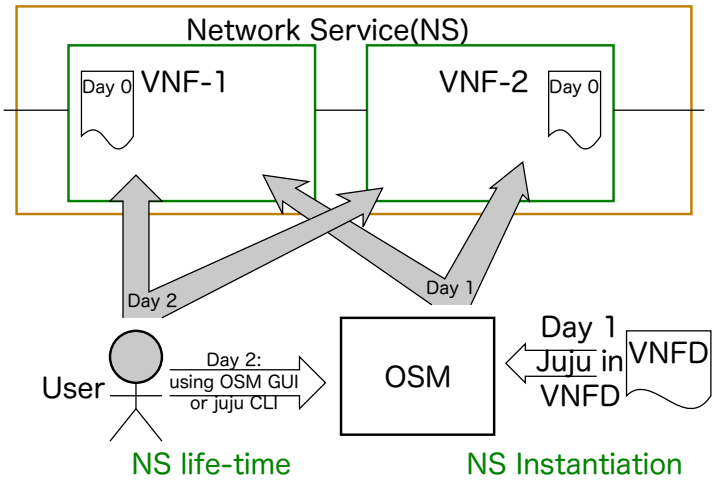


Figure 3.1.: *Day-X* configurations and actors

This part of the session is based on the introductory slides to the Juju session in the third OSM Hackfest [6] and [7]. However, use the slides with care, because the procedures described there need some changes.

3.1. The workflow for VNFs with proxy charms

Developing a NS Descriptor (NSD) with VNFs that use proxy charms always requires the following sequence of actions:

1. Create the day-0 configuration for the VNF(s) in form of a minimal cloud-init configuration that:
   - enables password access for SSH and sets a password for the default user
   - creates a second Ethernet (for the control) and enables DHCP on it
   - creates extra configuration files that will be used by the srvice if needed
2. Develop the proxy charm(s) (see below)
3. Integrate the proxy charm(s) and their actions into the VNF Descriptor(s).

The workflow for creating the proxy charms requires the following steps:

1. plan actions
   - define command(s) to be executed for each action and parameter(s) that may be eventually passed them
2. create clean charm
3. create the `actions/*` files and make sure they are executable
4. create the `actions.yaml` file
5. create the `reactive/<charm_name>.py` file implementing the actions
6. successfully compile charm

3.2. The first charm

The first charm you will implement creates a single VM VNF for a Network Service with a single action: to create an empty file in the VM using the `touch` command. The action will be used in the initial configuration step (day-1 configuration) as well as during the life-time of the NS (day-2 configuration).

3.2.1. Day-0 configuration

For this lab, we will reuse the work we did in the first NFV laboratory, in order to speed up the process and concentrate on the additional steps introduced by the use of proxy

charms.

1. On a clean `$HOME/Devel` directory, decompress the `ubuntu_vnf` VNFD bundle you created in the first NFV labortory session.
2. Open the cloud-config file and remove the `packages:` entry.
3. Now enable password driven SSH sessions and define the password "`osm-2018`" for the default user *i.e.* `ubuntu`, since we are using a Ubuntu Xenial image, at the beginning of the file: The cloud config file should look like this:

```
#cloud-config
password: osm-2018
chpasswd: {expire: False}
ssh_pwauth: True

package_update: true

write_files :
- content : |
    auto ens4
    iface ens4 inet dhcp
  path: /etc/network/interfaces.d/51-ifcfg-ens4.cfg
  permissions: '0644'

runcmd :
  - [ ifup , ens4 ]

manage_etc_hosts: True
```

   Note : the last line in the `cloud-config` file is mandatory to avoid problems with remote `sudo` commands.

Milestone 3: The day-0 configuration is ready

4. Juju Charms communicate with OSM through one of the Ethernet interfaces of the VNFs that compose a Network Service. This interfaces needs to be properly specified in the VNFD file. Check that the `mgmt-interface` section:

```
# Management interface
mgmt-interface:
    cp: eth1
```

   corresponds with your VNFD interface assigments:

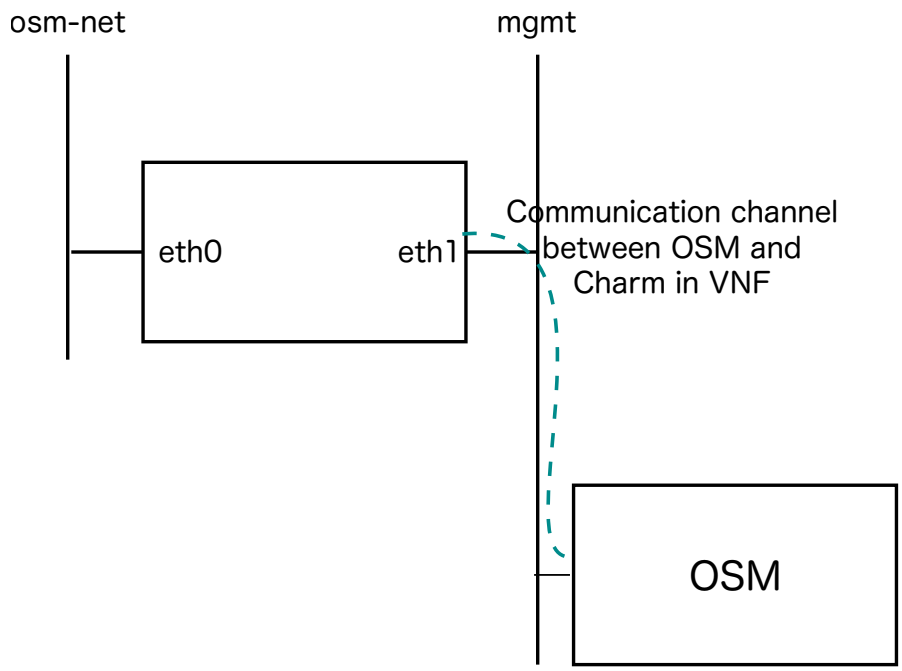


Figure 3.2.: Rationale for the mgmt-interface definition

3.2.2. Developing the Juju proxy charm

The laptop has the basic Juju charm development tools installed. Go to your `$HOME` directory and create the charm development directory tree if it is not present and go to that directory

```
mkdir -p $HOME/charms/layers
cd $HOME/charms/layers
```

Check that the environment variable `JUJU_REPOSITORY` is defined in your shell (*e.g.* the shell command `set  grep JUJU|` should show you the variable and its value). If not, include the line:

```
export JUJU_REPOSITORY=$HOME/charms
```

in your `$HOME/bash_aliases` file.

Now you are ready to create the skeleton and start developing the charm components.

4. Use `charm` to create the skeleton, check what was created and go to the created infrastructure:

   ```
   cd $JUJU_REPOSITORY/layers
   charm create simple
   cd simple
   ```

   The process of creating a charm involves editing three files within this project:

- `layer.yaml`
- `metadata.yaml`
- `reactive/simple.py`

and creating an action handler that we will replicate for all actions defined in the charm and the action definition file.

5. The `layer.yaml` file defines the layers used for VNF ssh proxy charms. It should look like this in any OSM Release FOUR charm definition:

```
includes: ['layer:basic', 'layer:vnfproxy']
options:
    basic:
        use_venv: false
```

6. The `metadata.yaml` file needs to define the Juju charm name. You may also change the summary and maintainer, but they are not essential for the charm to work:

```
name: simple
summary: A simple VNF proxy charm
maintainer: Name <user@domain.tld>
subordinate: false
series: ['xenial']
```

7. Now you have to decide what actions your charm will implement. In this first example, we are only going to implement one action called `touch`. We first need to create an action handler:

```
mkdir actions && cd actions
textadept touch
```

putting the following in the actions/touch file:

```
#!/usr/bin/env python3
import sys
sys.path.append('lib')
from charms.reactive import main, set_flag
from charmhelpers.core.hookenv import action_fail, action_name

set_flag('actions.{}'.format(action_name()))
try:
    main()
except Exception as e:
    action_fail(repr(e))
```

and making the file executable:

```
chmod +x touch
```

8. The next step is to create a YAML file to expose the actions and their parameters. This file is called `actions.yaml` and shoud be located in the `simple/` directory. We need to define an action called `touch` that take a filename as parameter:

```
touch:
    description: "Touch a file on the VNF."
    params:
        filename:
            description: "The name of the file to touch."
            type: string
            default: ""
    required:
        - filename
```

Hint : Using textadept will help you get the indenting right. for example, it will make a quick check to try to find errors before saving YAML files.

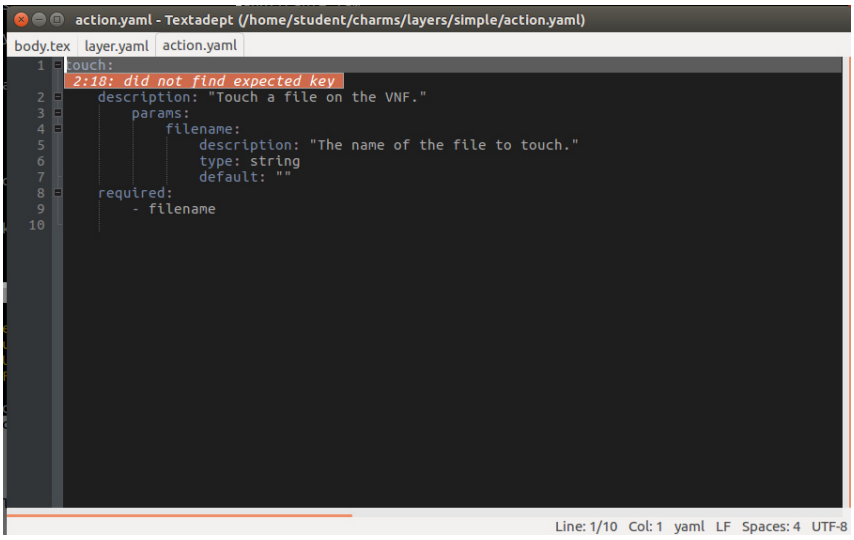


Figure 3.3.: textadept checks when saving a YAML file

9. Finally, we need to create the handler for the action in `reactive/simple.py`:
   - Make sure the beginning of the file looks like this:

```
from charmhelpers.core.hookenv import (
 action_get,
 action_fail,
 action_set,
 status_set,
)
from charms.reactive import (
 clear_flag,
 set_flag,
 when,
 when_not,
)
import charms.sshproxy
```

   - Then define what needs to be done for the `touch` action at the end of this file:

```
# Define what to do when the 'touch' primitive is invoked.
@when('actions.touch')
def touch():
    err = ''
    try:
        filename = action_get('filename')
        cmd = ['touch {}'.format(filename)]
        result, err = charms.sshproxy._run(cmd)
    except:
        action_fail('command failed:' + err)
    else:
        action_set({'outout': result}) # ?? output ??
    finally:
        clear_flag('actions.touch')
```

- Once all the files are edited/created, you can proceed to compile the charm:

```
cd $JUJU_REPOSITORY/layers/simple
charm build
```

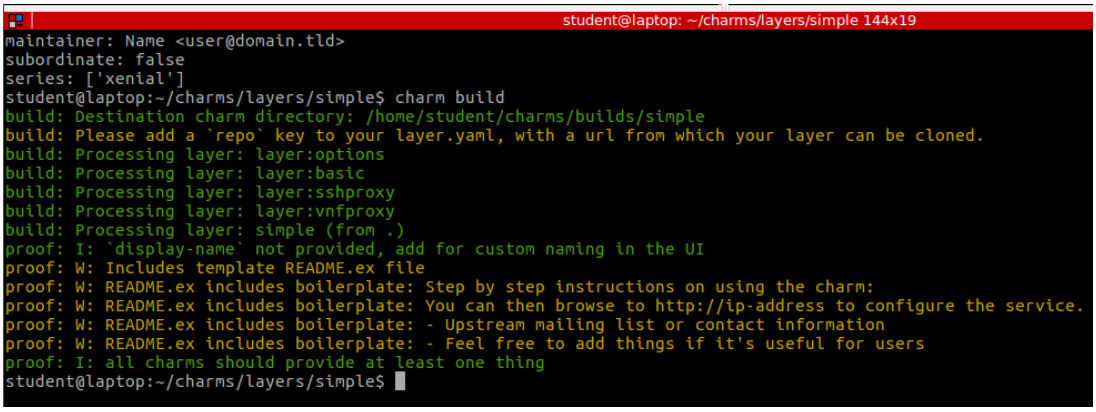


Figure 3.4.: Building the charm

Milestone 4: You have created a simple charm with one action

3.2.3. Integrating the charm into the VNFD

Integrating a charm into a VNFD basically implies three steps:

1. Copy the successfully compiled Charm into the VNFD infrastructure:

```
cd $HOME/Devel/ubuntu_vnf/charms
cp -rv $JUJU_REPOSITORY/builds/simple .
```

2. Include the charm bindings in the VNFD YAML file. This involves adding the following configuration definition at the end of the VNFD YAML file:

```
vnf-configuration:
    juju:
        charm: simple
    initial-config-primitive:
```

```
-   seq: '1'
    name: config
    parameter:
    -   name: ssh-hostname
        value: <rw_mgmt_ip>
    -   name: ssh-username
        value: ubuntu
    -   name: ssh-password
        value: osm-2018
-   seq: '2'
    name: touch
    parameter:
    -   name: filename
        value: '/home/ubuntu/touch-day1.txt'
config-primitive:
-   name: touch
    parameter:
    -   name: filename
        data-type: STRING
        default-value: '/home/ubuntu/touch-day2.txt'
```

As you can see, the `juju:` definition references the charm we just copied into the VNFD infrastructure. The `initial-config-primitive:` defines the sequence of commands that are executed to provide the Day-1 configuration. The first element **must** define the parameters for the Charm to remotely access the VNFs.

Note : The user and password definitions are taken from the Day-0 configuration defined in Section 3.2.1. The IP address must be spelt out as shown in the listing. This instructs OSM to take it from the management information provided by the VIM.

The second element in the sequence then uses the `touch` primitive to create a first file in the default user's home directory.

Finally, you have the `config-primitive:` field to define Day-2 configuration primitives. In our case, we use the `touch` primitive again to create other files in the VM.

Regarding the indentation, make sure that this configuration block is at the same level as the `connection-point` block:

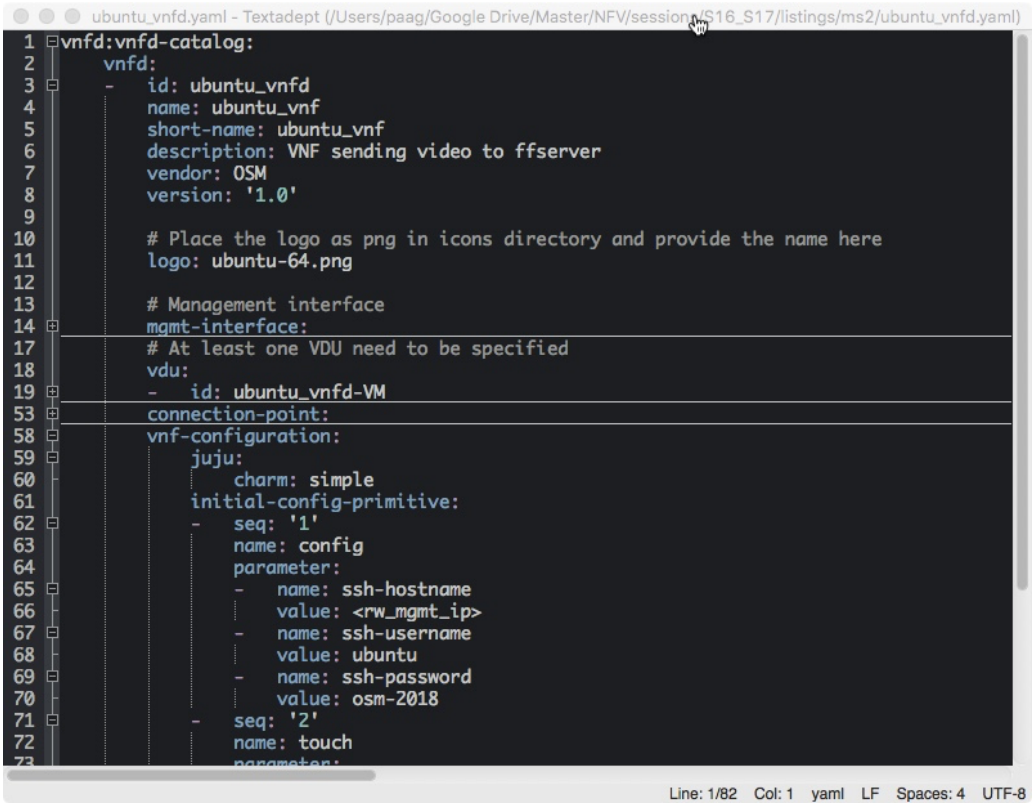


Figure 3.5.: The correct intentation for the `vnf-configuration` block

3. Generate the VNFD bundle

```
cd $HOME/Devel
validate_descriptor.py -V ubuntu_vnf/ubuntu_vnfd.yaml
generate_descriptor_pkg.sh -N -v ubuntu_vnf
```

Milestone 5: You have created a VNF Descriptor that uses the simple Juju charm created previously

3.2.4. Deploying and testing the VNFD with its NSD

Copy the NSD using the `ubuntu_vnf` VNF you created in the first NFV laboratory session. Deploy the VNFD bundle you just created and this NSD bundle. Instantiate the Network Service and follow the deployment procedure in the devstack GUI.

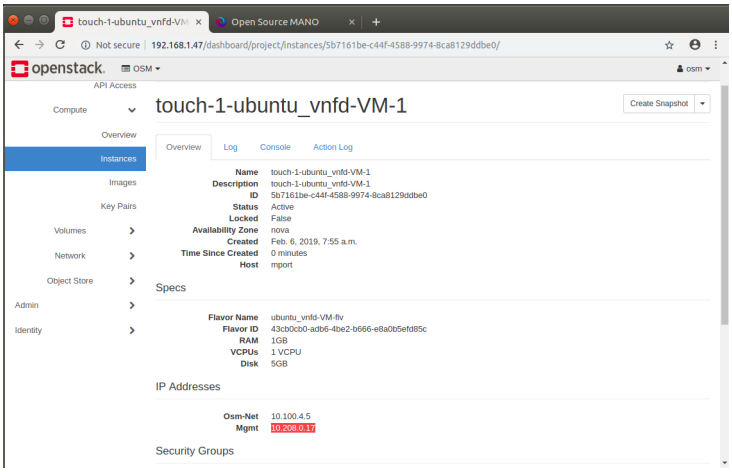
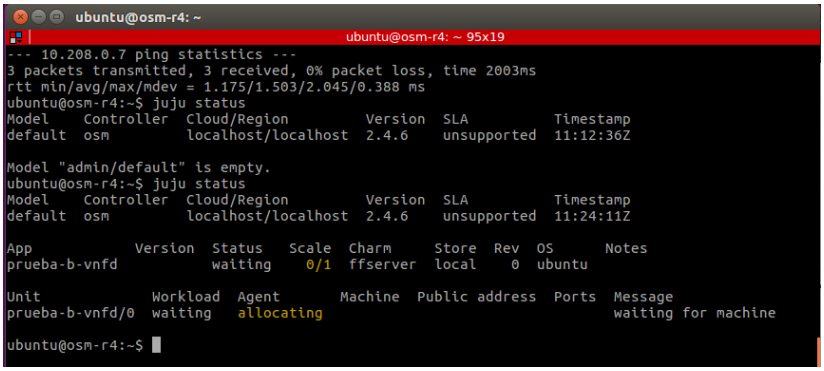


Figure 3.6.: Addresses assigned to the VNF

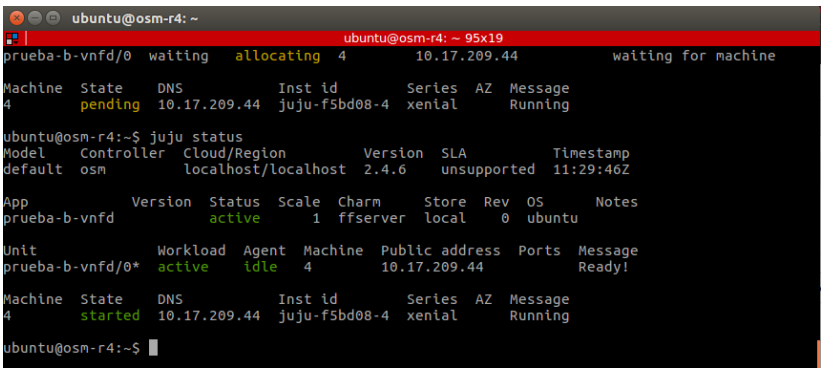
Once the VM is active, note the IP addresses it was assigned and start the checking procedure:

1. Enter the `osm-r4` VM and ping the IP address the VNF received in the `mgmt` network.
2. Enter the VNF console from the devstack GUI. Check that the default user can use the password it was assigned in the Day-0 configuration file.
3. In the `osm-r4` console, follow the charm creation process with the command:

```
juju status
```



(a) `juju status` when the VM has started



(b) `juju status` when the charm was launched correctly

Figure 3.7.: Starting a VNF with charms

4. Once the charm has started, you can follow the instantiation process on the OSM GUI:

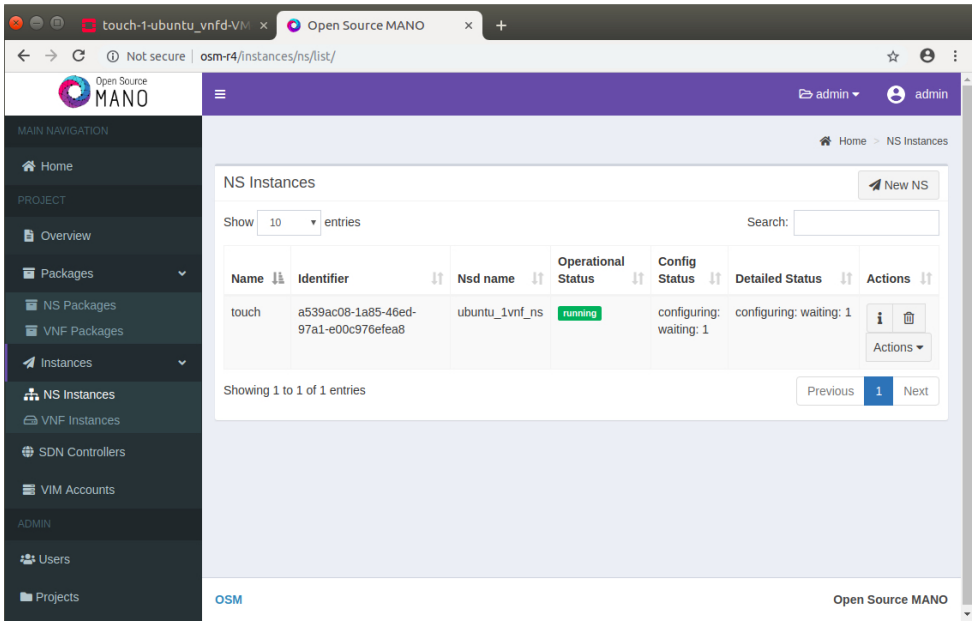


Figure 3.8.: OSM is starting the VM and the charm to configure it

After refreshing the GUI for a certain time, the VNF will have been configured:

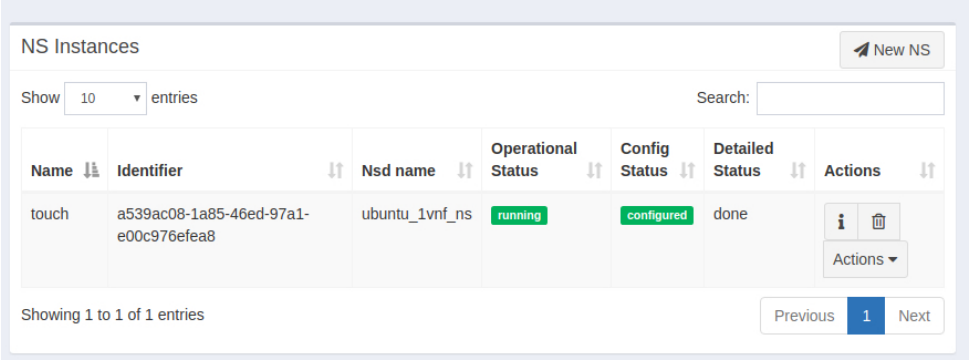


Figure 3.9.: OSM has finished the Day 1 configuration (GUI detail)

Using the VM console provided by devstack, check that the “Day-1 configuration file” was created.

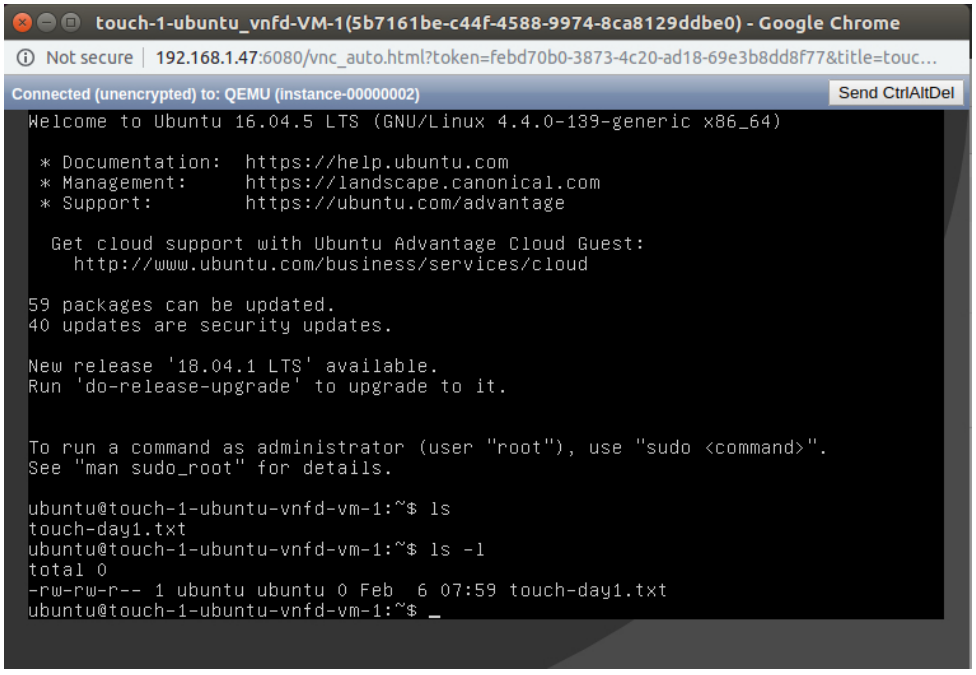


Figure 3.10.: VNF console showing that the Day-1 configuration was completed

5. Now, we are ready to trigger the “Day-2 configuration primitives”using the OSM GUI. In this example, we use the `touch` action we also used for the “day-1” configuration:
   - In the NS instance view of the OSM GUI, select the “Actions” button
   - select the “Exec NS Primitives” item

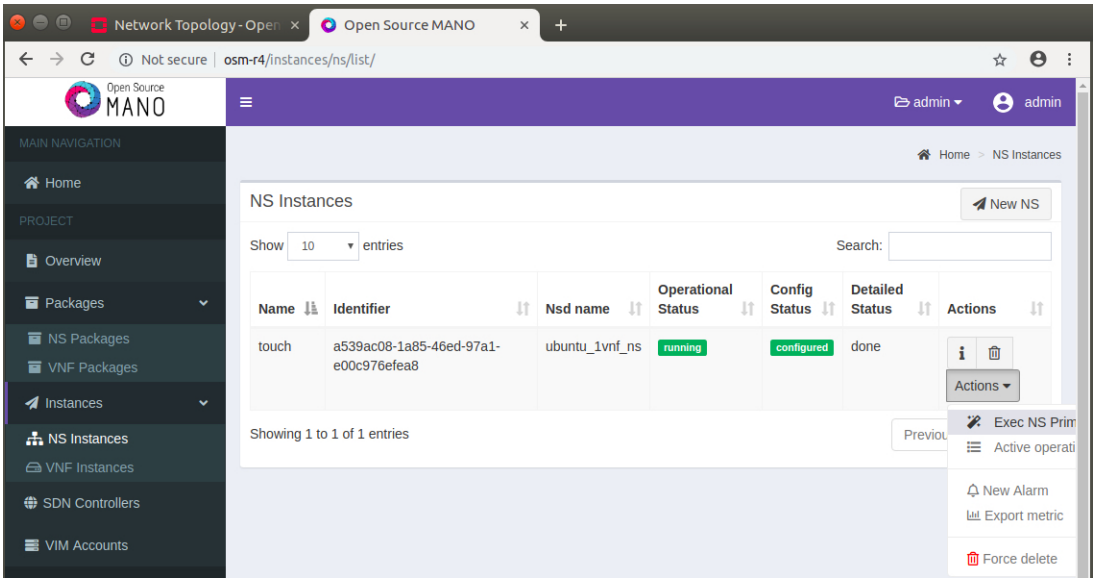


Figure 3.11.: Note that the dropdown menu is cut

a dialog will appear:

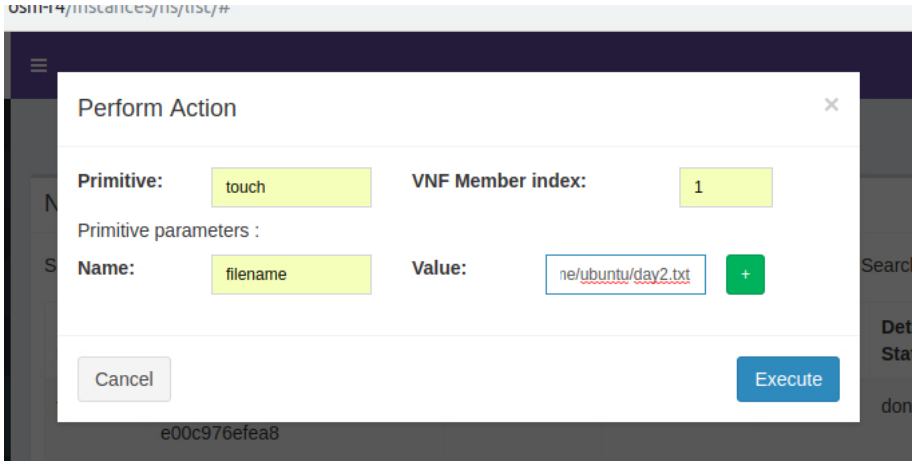


Figure 3.12.: Instance actions dialog

- Specify the `touch` primitive for VNF member index `1`, primitive parameter name `filename` and Value `/home/ubuntu/day2.txt`
- Using the devstack VM console, check that the file was created.

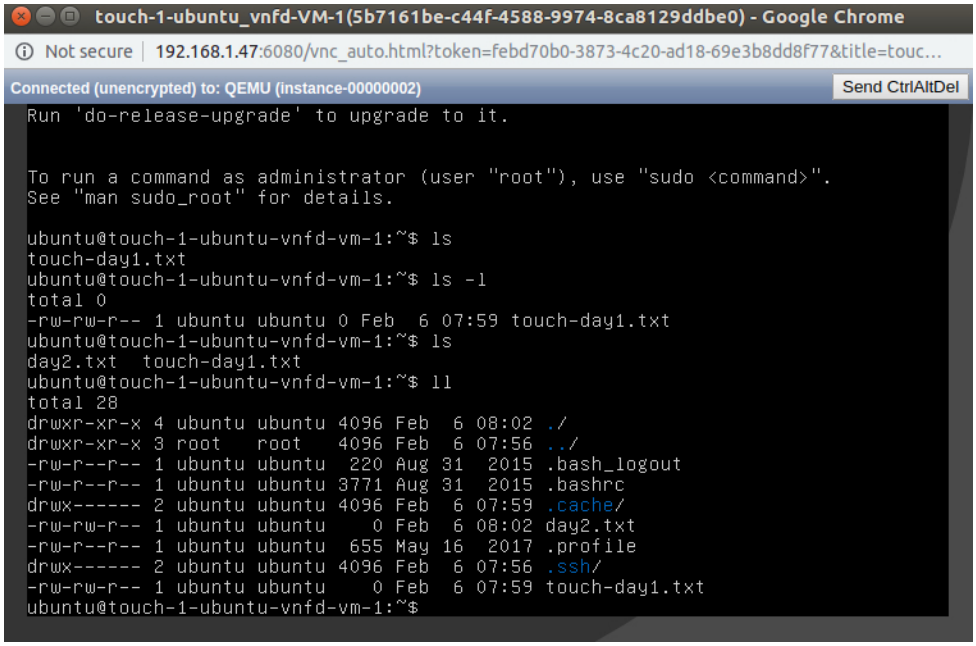


Figure 3.13.: ”Day 2” configuration done

6. Finally, using the “Active Operations” item in the “Actions” dropdown menu of the `NS Instances` view, check the actions that have been performed on the VNF:

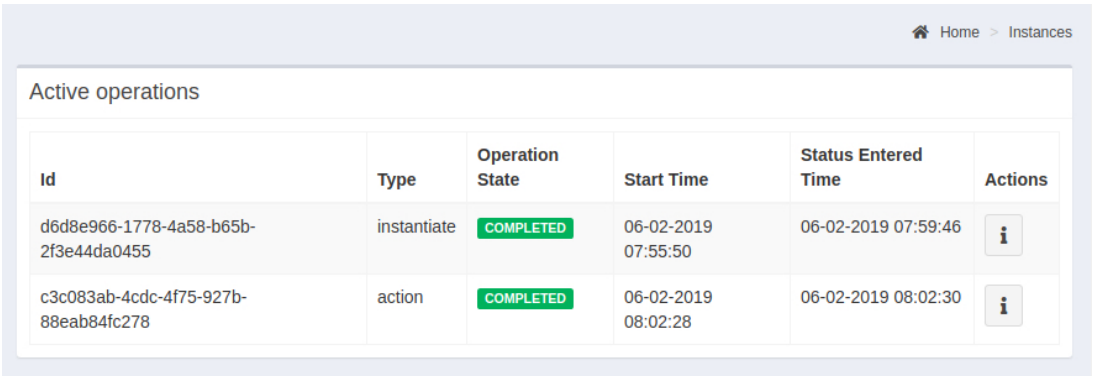


Figure 3.14.: Actions performed on the VNF

Checkpoint 2: (3 points) You have created a simple network service that uses cloud-init for Day0 and a simple Juju charm for Day-1/Day-2 configuration.

After your checkpoint has been evaluated, delete the Network Service instance, the NS and VNF descriptors.

Milestone 6: You have a clean OSM to start developing your own Network Services.

4. Developing a video transcoding NS

(estimated duration: 2h-2.5h)

Revisiting the service we implemented in the first NFV laboratory session, we see that there are four basic actions we implemented and coordinated (as far as the setup allowed):

- providing the required software
- getting the source video
- starting the video server (*i.e.* `ffserver`)
- starting the video feed (using `ffmpeg`)

We checked the service using `ffplay`. The process can be represented in a small flow diagram:

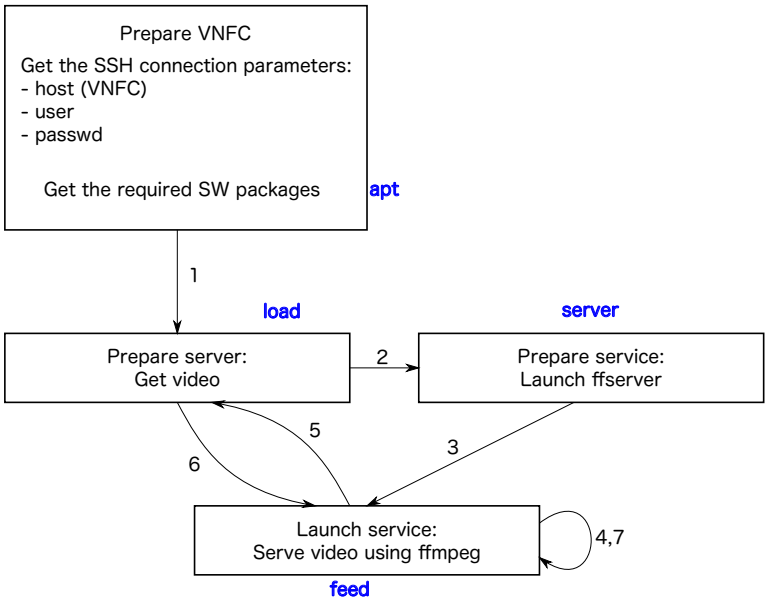


Figure 4.1.: Service logic we implement in the video server

where the action names we are going to use are highlighted in blue.

4.1. Developing the video server as a single VM NS

The principles of developing a charm were covered in the previous checkpoint as well as in the second NFV laboratory session. We are just making things more elaborate by introducing a multi-action charm. Once the charm is working at the end of this section, the service will be split in two distinct Virtual Machines interacting through the management Virtual LAN (VLAN).

4.1.1. Developing the Juju Charm: identifying the commands executed in each action

We start by identifying the CLI commands each charm will implement:

1. In order to install packages, since we are in a Ubuntu environment, we will use the '`apt install -y`' command, raising privileges using the `sudo` command. Additional magic is needed as described in [8].
2. In order to get the video(s) from the lab server, we will use the `wget` command, using the charm to provide as much information as possible and keeping the action parameter(s) as short as possible.
3. The `ffserver` program needs a helper to run effectively: GNU `screen` is used to run the program in the background and eventually getting access to the console. It is advisable to create a wrapper script in the `/usr/local/bin` directory using the cloud-config file and execute `screen` and `ffserver` from it.

Milestone 7: You have the CLI commands to implement the four actions. YOu can now start coding the Juju charm

This may be a trial and error process and it is advisable that you get familiar with the Juju CLI tools in the `osm-r4` Virtual Machine (cf.Annex A) since they will speed up the process.

4.1.2. Developing the Juju charm: coding the initial version

Repeat the process for creating a Juju charm you followed for the previous checkpoint. Sketched out, the steps you must follow are the following:

1. start by creating the basic code infrastructure for the charm:

```
cd $HOME/charms/layers
charm create ffmpeg
```

2. copy the basic action code from your `simple` charm to the new charm:

```
for action in apt load server feed; do
    cp simple/actions/touch ffmpeg/actions/$action
done
```

Hint : Look for other files you may reuse

3. register the actions in the `actions.yaml` file. As an example, the `apt` action should look like:

```
apt:
    description: "Install the debian packages"
    params:
        packages:
            description: "The list of packages"
            type: string
            default: "wget ffmpeg screen"
    required: [ packages ]
```

The action to start `ffserver`, which does not require parameters, should look like this:

```
server:
    description: "Start ffserver"
```

Customise `metadata.yaml` to match your charm.

4. Copy the Python implementation code from the `simple` charm:

```
cd $HOME/charms/layers
cp simple/reactive/simple.py ffmpeg/reactive/ffmpeg.py
```

Hint : Use the <Tab> key to complete file names and avoid typing errors

5. customise the Python implementation and include the hooks for the different actions.

Remember to rename the `simple.installed` hook to `ffmpeg.installed` and provide meaningful variable names there:

```
# Set the charm's state to active so the LCM knows
# it is ready to work.
@when_not('ffmpeg.installed')
def install_ffmpeg_proxy_charm():
    set_flag('ffmpeg.installed')
    status_set('active', 'Ready!')
```

Take care with the `server` action. Remember it is parameterless:

```
@when('actions.server')
def server():
    err = ''
    try:
        cmd = '/usr/local/bin/start-ffserver.sh'
        result, err = charms.sshproxy._run(cmd)
    except:
        action_fail('command␣failed:' + err)
    else:
        action_set({'output': cmd})
    finally:
        clear_flag('actions.server')
```

Hint : Note that the charm invokes a shell script in the `/usr/local/bin` directory. You will have to provide the shell script in the cloud-config file.

6. Compile your charm and make sure it compiles successfully.

Checkpoint 3: (1 point) The Juju charm driving the video Network Service is ready.

4.1.3. Creating the VNFD and integrating the charm

Using the VNF Descriptor you created for the previous checkpoint as a starting point, create the VNFD for the single VM video server. A suggested name in `ubuntu_ffmpeg_vnf`. In the VNFD YAML file:

1. take into account that the application needs a VM with enough resources:
   - 2 VCPUs
   - 2048 MB memory
   - 5 GB of storage
2. integrate the `apt` action as the second action in the `initial-config-primitive` sequence
3. put the other actions (*i.e.* `load`, `server` and `feed`) as `config-primitive`s. Remember that the `server` action has no parameters:

```
config-primitive:
-   name: server
-   name: load
    parameter:
    -   name: filename
        data-type: STRING
```

4. customise the cloud-config file. There are a couple of modifications needed:
   - create a specific configuration file for `ffserver` with the contents of the `ffserver` configuration file you used in the first NFV laboratory session. Make sure it doesn't collide with the default configuration file (`/etc/ffserver.conf`) because this will hamper the package installation process.
   - create script in `/usr/local/bin` to lauch `ffserver` using the file you create in the Day-0 configuration. Use the `screen` command to run `ffserver` in the background. Experiment with launching **screen** with `nohup` if the server is not kept running after the script exits. An example for the cloud-config configuration that creates this script is:

```
- content : |
    #!/bin/bash
    screen -S ffserver -dm ffserver -f /etc/ffserver-flv.conf
  path: /usr/local/bin/start-ffserver.sh
  permissions: '0755'
```

- create a script to launch `ffmpeg` serving the file in a `screen` session.

Hint : Creating the scripts in `/usr/local/bin` allows a better debug strategy: You can try to launch the action and if it fails, modify the script on the VM you are testing.

- compile the VNF Descriptor

Checkpoint 4: (1 point) You have a viable VNF Descriptor.

4.1.4. Creating the NS Descriptor and launching the Network Service

Starting with the NS Descriptor you created for the `simple` charm, create a new NSD that used the VNFD you just created. The objective is that you deploy the following on your multiport:

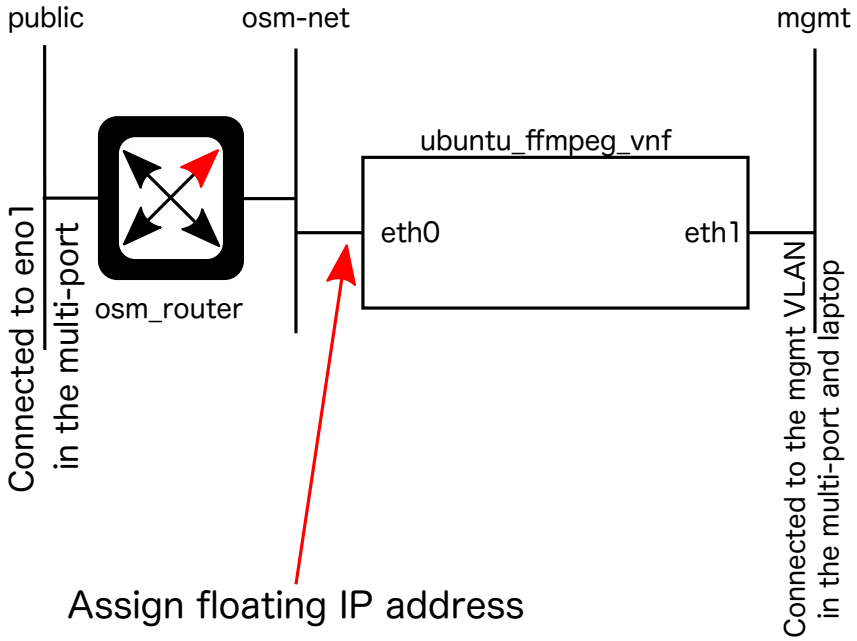


Figure 4.2.: The video transcoder Network Service

Checkpoint 5: (1 point) You have a viable Network Service

Now instantiate the Network Service and show that it is running correctly. Use the `ffplay` program:

```
ffplay http://<floating IP assgined to NS>:8090/live.flv
```

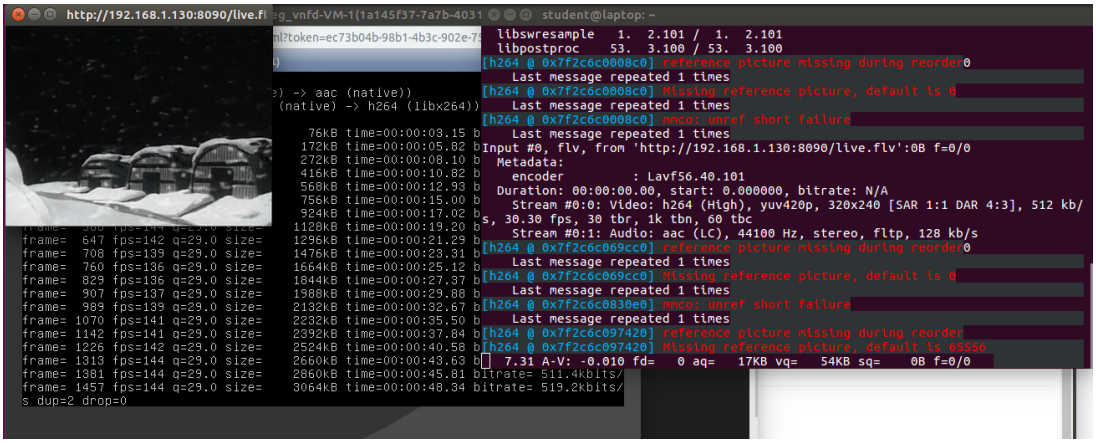


Figure 4.3.: The video and the ffmpeg screen

Checkpoint 6: (2 points) Demonstrate the deployment process using the different NS primitives, and that your video is playing.

A. Debugging the charms: interacting with a charm from the CLI

The charm creation and activation processes are complex and sometimes cumbersome. However, there is full access to the Juju environment in the OSM VM. The default `ubuntu` user has access to the Juju CLI and can monitor the whole process and interact with the deployed VNFs directly.

A.1. Checking the status of a deployed charm

The command `juju status` provides information about deployed charms:

```
        ubuntu@osm-r4:~$ juju status
Model    Controller  Cloud/Region         Version  SLA
   Timestamp
default  osm         localhost/localhost  2.4.6    unsupported  09:51:56
   Z

App           Version  Status  Scale  Charm   Store  Rev  OS      Notes
otra-b-vnfd            active      1  simple  local    6  ubuntu

Unit             Workload  Agent  Machine  Public address  Ports  Message
otra-b-vnfd/0*   active    idle   9        10.17.209.24           Ready!

Machine  State    DNS           Inst id        Series  AZ  Message
9        started  10.17.209.24  juju-f5bd08-9  xenial      Running
```

A.2. Following the deployment and action execution processes

The command `juju debug-log` will give you access to the log console of Juju. You can start it at any point and it is recommended that you start it ***before*** you deploy a Network Service for the first time.

You can use the terminal's buffer to scroll up and check for potential errors.

To exit, press `Control-C`. This will stop the output, but will not harm the Juju processes running.

A.3. CLI interaction with charms

There are four commands to interact with deployed charms from the CLI:

- `juju list-actions <app>` to show a list actions provided by a charm
- `juju run-action <unit> <action> [param="value"]` to execute a specific action
- `juju show-action-status <action-id>` to show the status
- `juju show-action-output <action-id>` to show the output produced by an action

All the process starts with examining the output of the `juju status` command:

```
        ubuntu@osm-r4:~$ juju status
Model    Controller  Cloud/Region         Version  SLA
   Timestamp
default  osm         localhost/localhost  2.4.6    unsupported  09:51:56
   Z

App           Version  Status  Scale  Charm   Store  Rev  OS      Notes
otra-b-vnfd            active      1  simple  local    6  ubuntu

Unit             Workload  Agent  Machine  Public address  Ports  Message
otra-b-vnfd/0*   active    idle   9        10.17.209.24           Ready!

Machine  State    DNS           Inst id        Series  AZ  Message
9        started  10.17.209.24  juju-f5bd08-9  xenial      Running
```

Note that the charm is called App and that the instance deployed in a VNF is called Unit

A.3.1. Listing the actions implemented in a VNF

You can use the `juju list-actions <app>` to show a list actions provided by a charm:

```
ubuntu@osm-r4:~$ juju list-actions otra-b-vnfd
Action                  Description
apt                     Install the debian packages
feed                    Send the video to ffserver using ffmpeg
generate-ssh-key        Generate a new SSH keypair for this unit. This
   will replace any existing previously generated keypair.
get-ssh-public-key      Get the public SSH key for this unit.
load                    Get a video from the lab server
reboot                  Reboot the VNF virtual machine.
restart                 Stop the service on the VNF.
run                     Run an arbitrary command
server                  Start ffserver
start                   Stop the service on the VNF.
stop                    Stop the service on the VNF.
upgrade                 Upgrade the software on the VNF.
```

```
verify-ssh-credentials  Verify that this unit can authenticate with
    server specified by ssh-hostname and ssh-username.
```

Note how to use the application name from the `juju status command`

A.3.2. Executing an action from the CLI

To execute an action from the CLI, you need to include (at least) the required parameters. Failing to do so will generate an error message:

```
ubuntu@osm-r4:~$ juju run-action otra-b-vnfd/0 touch
ERROR validation failed: (root) : "filename" property is missing and
    required, given {}
```

When all parameters are correctly included, the action is triggered.

```
ubuntu@osm-r4:~$ juju run-action otra-b-vnfd/0 touch filename="/home/
    ubuntu/cli.txt"
Action queued with id: 3984c8d7-8cd0-4f3b-8e07-35bb94cfdf76
```

The action identifier can then be used to monitor the progress and check the output.

A.3.3. Monitoring an action from the CLI

Using the action identifier provided by the `juju run-action` command, you can follow the progress and check when the action has been completed:

```
ubuntu@osm-r4:~$ juju show-action-status fd2ba092-d98d-4fbe-8f2d-91050
    b22b8a6
actions:
- action: load
  completed at: n/a
  id: fd2ba092-d98d-4fbe-8f2d-91050b22b8a6
  status: running
  unit: ultimo-b-vnfd/0
ubuntu@osm-r4:~$ juju show-action-status fd2ba092-d98d-4fbe-8f2d-91050
    b22b8a6
actions:
- action: load
  completed at: n/a
  id: fd2ba092-d98d-4fbe-8f2d-91050b22b8a6
  status: running
  unit: ultimo-b-vnfd/0
ubuntu@osm-r4:~$ juju show-action-status fd2ba092-d98d-4fbe-8f2d-91050
    b22b8a6
actions:
- action: load
  completed at: "2019-02-08 12:08:54"
  id: fd2ba092-d98d-4fbe-8f2d-91050b22b8a6
  status: completed
  unit: ultimo-b-vnfd/0
```

A.3.4. Getting the output produced by an action

Once the action is completed, you can use the action identifier to check the output generated by the action:

```
ubuntu@osm-r4:~$ juju run-action otra-b-vnfd/0 server
Action queued with id: 58ffffc3-4695-4c42-899a-710a8260de71
ubuntu@osm-r4:~$ juju show-action-status 58ffffc3-4695-4c42-899a-710
    a8260de71
actions:
- action: server
  completed at: "2019-02-08 12:10:24"
  id: 58ffffc3-4695-4c42-899a-710a8260de71
  status: completed
  unit: ultimo-b-vnfd/0
ubuntu@osm-r4:~$ juju show-action-output 58ffffc3-4695-4c42-899a-710
    a8260de71
results:
  output: nohup screen -dmS ffserver ffserver -f /etc/ffserver-flv.conf
status: completed
timing:
  completed: 2019-02-08 12:10:24 +0000 UTC
  enqueued: 2019-02-08 12:10:20 +0000 UTC
  started: 2019-02-08 12:10:21 +0000 UTC
```

B. Interacting with programs through `screen` command

The easiest way use screen with name:

```
screen -S <name> <application>
```

If the application needs to run in detached mode (*i.e.* when you need the program to run in the background)

```
screen -S <name> -dm <application>
```

Return to screen a the session you created before:

```
screen -r 'name'
```

To exit the screen without killing the program use the key combination `Ctrl+a d`.

List the active `screen`s:

```
screen -ls
```

Kill screens marked as inactive, dead, etc.

```
screep -wipe
```

C. Suggested commands for the service

This is just a suggestion of possible ways to implement the commands you need for the video server. Experiment on your own.

Install software

Using the package management software in your VM you can install new software with the following command:

```
sudo DEBIAN_FRONTEND=noninteractive apt-get -y install <pkg> ...
```

Suggested packages are `ffmpeg`, `screen` and `wget`. SOme may be already installed. In that case, `apt` will silenty ignore the request to install the requested packages.

Downloading the video from the lab server

Using `wget`, you can download the videos from the lab server. Use the command:

```
wget -o /home/ubuntu/wget.log <url>
```

This will leave a log in the hoem directory of the `ubuntu` user. You can use the command '`tail -f /home/ubuntu/wget.log`' to follow the downloading process. Quit with `Ctrl+C` once the file has been downloaded.

Launching ffserver

A possible way to launch `ffserver` and keep it in the background is to create a script that executes the following command:

```
nohup screen -S ffserver -dm ffserver f <ffserver configuration>
```

Follow the instructions in Annex B to access the screen and check that `ffserver` is running correctly.

Launching ffmpeg

A possible way to launch `ffmpeg` and keep it in the background while it sends the video to the server is to create a script that executes the following command:

```
nohup screen -S ffmpeg -dm ffmpeg -re -i <video> http://localhost:8090/feed1.ffm
```

Follow the instructions in Annex B to access the screen and check that `ffmpeg` is running correctly.

Acronyms

CLI Command Line Interface
ETSI European Technical Standards Institute
GUI Graphical User Interface
LAN Local Area Network
LXC LinuX Containers
MANO Management and Orchestration
NFV Network Function Virtualisation
NFVI NFV Infrastructure
NS Network Service
NSD NS Descriptor
OSM Open Source MANO
SSH Secure SHell
URL Universal Resource Locator
VDU Virtual Device Unit
VIM Virtual Infrastructure Manager
VLAN Virtual LAN
VLD Virtual Line Descriptor
VM Virtual Machine
VNF Virtual Network Function
VNFC VNF Component
VNFD VNF Descriptor

Bibliography

[1] Keir Thomas. Ubuntu Pocket Guide and Reference. http://www.ubuntupocketguide.com/index_main.html.

[2] Ubuntu handbook. http://ubuntuhandbook.org.

[3] ETSI. OSM Release FOUR. https://osm.etsi.org/wikipub/index.php/OSM_Release_FOUR, May 2018.

[4] ETSI. OSM Release FIVE. https://osm.etsi.org/wikipub/index.php/OSM_Release_FIVE, Dec 2018.

[5] ETSI. OSM Release FOUR Technical Overview. https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFOUR-FINAL.pdf, May 2018.

[6] Adam Israel. OSM Hackfest Session 7a - Creating your first proxy charm. https://osm-download.etsi.org/ftp/osm-4.0-four/3rd-hackfest/presentations/20180628%20OSM%20Hackfest%20-%20Session%207a%20-%20Adding%20day-1%20and%20day-2%20configuration%20to%20your%20VNF%20-%20Creating%20your%20first%20proxy%20charm.pdf, 2017.

[7] Adam Israel. OSM Hackfest - Session 7b: Adding charms to your VNF descriptor. https://osm-download.etsi.org/ftp/osm-4.0-four/3rd-hackfest/presentations/20180628%20OSM%20Hackfest%20-%20Session%207b%20-%20Adding%20Charms%20to%20your%20VNF%20Descriptor.pdf, 2017.

[8] https://serverfault.com/questions/227190/how-do-i-ask-apt-get-to-skip-any-interactive-post-install-configuration-steps, Jan 2012.