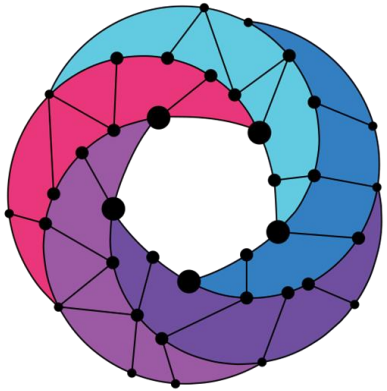


Open Source
MANO

OSM Hackfest - Session 5
Adding day-1/day-2 configuration to your VNF
Creating your first proxy charm

Adam Israel, Canonical
Gerardo García, Telefónica

ETSI

What is Juju?

- Juju is an open source modeling tool, composed of a controller, models, and charms, for operating software in the cloud.
- Juju can handle configuration, relationships between services, lifecycle and scaling.
- This ensures that common elements such as databases, messaging systems, key value stores, logging infrastructure and other ‘glue’ functions are available as charms for automatic integration, reducing the burden on vendors and integrators.

What is a Charm?

- A charm is a collection of software containing all of the logic to install, configure, and scale cloud-based applications in a repeatable and reliable way.
- Charms are installed on a machine, running a cloud image, and handle the full lifecycle of an application, including day-0, day-1, and day-2 config.
- But...

Proxy Charms

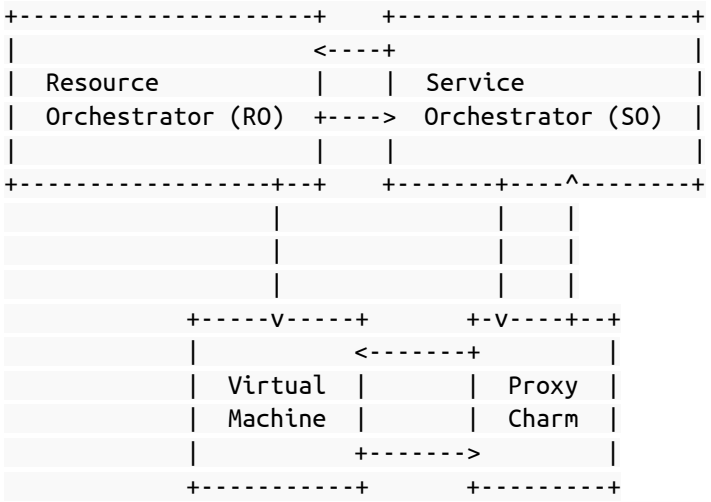
- OSM Release THREE* uses “proxy charms”, where the charm is installed into an LXD container, and is only responsible for day-1 and day-2 configuration, executed remotely (typically via ssh).

- Don’t worry! Proxy charm support is being expanded to support more features of full charms, and will still be supported in future releases.

* Full charm support is a feature targeted at R4.

Proxy Charms

Here is a simple diagram showing how a proxy charm fits into the OSM workflow:



- A VNF package is instantiated via the SO
- The SO requests a virtual machine from the RO
- The RO instantiates a VM with your VNF image
- The SO instructs the VCA to deploy a VNF proxy charm, and tells it how to access your VM (hostname, user name, and password)

Open Source
MANO

Preparing your development environment

ETSI

Install the charm tools

Install charm tools via snap:

```
$ sudo snap install charm
charm 2.2.3 from 'charms' installed

$ charm version

charm 2.2.2
charm-tools 2.2.3
```

Setup your Charming environment

Create the directories we'll use for our charm:

```
mkdir -p ~/charms/layers
```

Tell the charm command where our workspace is (for best results, add this to ~/.bashrc):

```
export JUJU_REPOSITORY=~/charms
```

Open Source
MANO

Understanding charms

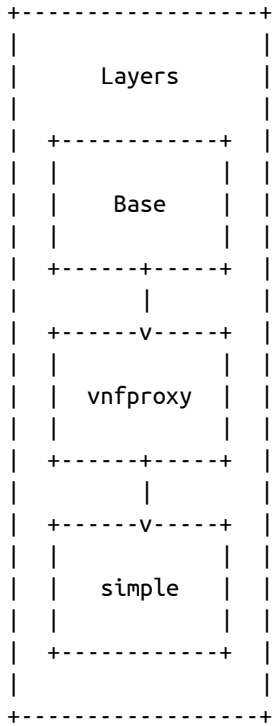
ETSI

Reactive Framework

- The *Reactive* programming pattern that allows a charm to respond to changes in state, including lifecycle events, in an asynchronous way.
- Lifecycle events may tell the charm to install, start, or stop an application, to perform leadership election, to collect metrics, or to upgrade the charm itself.

Layers

- Layers are encapsulations of charm code that lend themselves to being reused across charms.
- The Base layer contains the core code needed for other layers to function.
- Vnfproxy is a runtime layer which provides common functionality to interoperate with a VNF.
- Simple is the charm layer containing code to manage *your* vnf.



Creating a VNF Proxy charm

```
# Change into the layers folder

$ cd $JUJU_REPOSITORY/layers

# Invoke the charm command to create a charm layer
called ‘simple’

$ charm create simple

$ cd simple
```

Anatomy of a charm layer

To the right is the contents of your simple charm.

For the purposes of this example, we will ignore the struck-through files.

```
$JUJU_REPOSITORY/layers
└── simple
    ├── ~~config.yaml~~
    ├── ~~icon.svg~~
    ├── layer.yaml
    ├── metadata.yaml
    ├── reactive
    │   └── simple.py
    ├── ~~README.ex~~
    └── ~~tests~~
        ├── ~~00-setup~~
        └── ~~10-deploy~~
```

Anatomy of a layer

`layer.yaml` defines which base and runtime layers your charm depends on.

Edit `layer.yaml` to include the vnfproxy layer:

```
includes: ['layer:basic', 'layer:vnfproxy']
```

```
$JUJU_REPOSITORY/layers
└── simple
    ├── config.yaml
    ├── icon.svg
    ├── layer.yaml
    ├── metadata.yaml
    ├── reactive
    │   └── simple.py
    ├── README.ex
    └── tests
        ├── 00-setup
        └── 10-deploy
```

Anatomy of a layer

Edit `metadata.yaml` with the name and description of your charm:

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

```
$JUJU_REPOSITORY/layers
└── simple
    ├── ~~config.yaml~~
    ├── ~~icon.svg~~
    ├── layer.yaml
    ├── metadata.yaml
    ├── reactive
    │   └── simple.py
    ├── ~~README.ex~~
    └── ~~tests~~
        ├── ~~00-setup~~
        └── ~~10-deploy~~
```

Building your first charm

```
$ charm build
build: Destination charm directory: ~/charms/builds/simple
build: Please add a `repo` key to your layer.yaml, with a url from which your layer can be
cloned.
build: Processing layer: layer:basic
build: Processing layer: layer:sshproxy
build: Processing layer: layer:vnfproxy
build: Processing layer: simple (from .)
proof: W: Includes template README.ex file
proof: W: README.ex includes boilerplate: Step by step instructions on using the charm:
proof: W: README.ex includes boilerplate: You can then browse to http://ip-address to configure
the service.
proof: W: README.ex includes boilerplate: - Upstream mailing list or contact information
proof: W: README.ex includes boilerplate: - Feel free to add things if it's useful for users
proof: I: all charms should provide at least one thing
```

Examining the compiled charm

The `charm build` command takes all of the layers defined in layer.yaml, combines them into a single charm, and caches the dependencies in the `wheelhouse` directory for faster installation.

```
$ ls $JUJU_REPOSITORY/builds/simple

actions     bin           copyright  hooks      layer.yaml    Makefile
reactive    README.md     simple     tox.ini    actions.yaml  config.yaml
deps        icon.svg      lib        README.ex  metadata.yaml tests
requirements.txt          wheelhouse
```

Adding an action

Actions are functions that can be called automatically when a VNF is initialized or on-demand by the operator. In OSM terminology, we know these as config primitives.

Define an action

Let’s create `actions.yaml` in the root of the simple charm:

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

Create the action handler

$ mkdir actions

Create `actions/touch`, with the contents to the right.

When you're done, mark the script executable:

$ chmod +x actions/touch

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

set_state('actions.{}'.format(action_name()))

try:
    main()
except Exception as e:
    action_fail(repr(e))
```

***Note**: The same content has to be used for every action in the charm layer. It is only a boilerplate script to invoke the reactive framework*

Handle the action

Edit `reactive/simple.py`.

This is where all reactive states are handled.

```
from charmhelpers.core.hookenv import (
    action_get,
    action_fail,
    action_set,
    status_set,
)

from charms.reactive import (
    remove_state as remove_flag,
    set_state as set_flag,
    when,
    when_not,
)
import charms.sshproxy
```

Handle the action

Edit `reactive/simple.py`.

This is where all reactive states are handled.

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

Handle the action

Edit `reactive/simple.py`.

This is where all reactive states are handled.

```
# 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({'output': result})
    finally:
        remove_flag('actions.touch')
```

***Note**: For every action in the charm layer you need to add a @when decorator and the function to be run*

That’s it!

We’re ready to compile the charm with our new action:

$ charm build

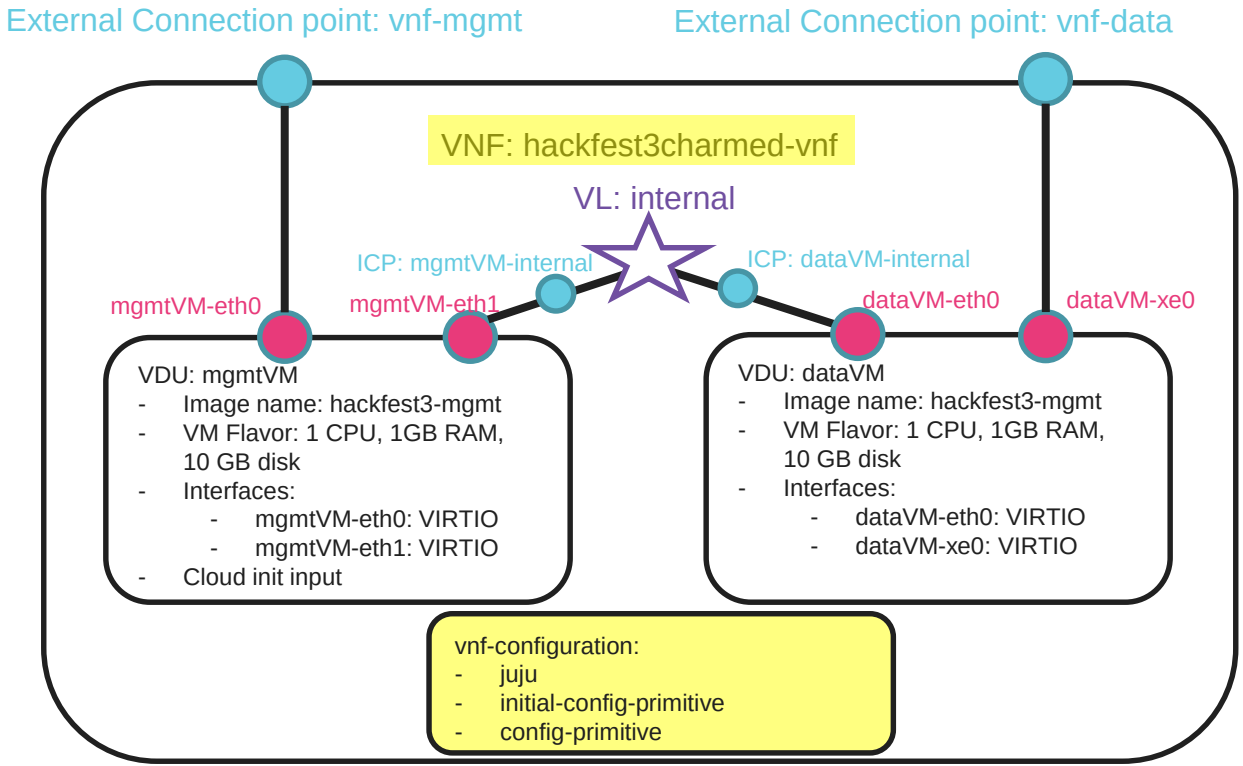
Open Source
MANO

Adding Charms to your VNF Descriptor

With subtitle

ETSI

VNF diagram
Changes with respect to ‘hackfest3-vnf’ highlighted in yellow



Generate the skeleton of the VNF Package and write the VNF descriptor

Create a skeleton folder with all the files required for a single-VM VNF package

./devops/descriptor-packages/tools/generate_descriptor_pkg.sh -t vnfd --image hackfest3-mgmt -c hackfest_3charmed

Go into the VNF folder and write the VNF descriptor 'hackfest_3charmed_vnfd.yaml'. To save time, replace the auto-generated descriptor by this one:

https://osm-download.etsi.org/ftp/osm-3.0-three/2nd-hackfest/other/hackfest_3charmed_vnfd.yaml

Charms and Descriptors

Add the vnf-configuration section, as seen to the right, to the end of your descriptor, with the same level of indentation as the name of the VNF.

`initial-config-primitive` defines the primitives run at day-1, when the charm is instantiated.

`config-primitive` defines the primitives available to run as day-2 configuration.

```
name: ‘myvnf’
...
vnf-configuration:
    initial-config-primitive:
    config-primitive:
    juju:
        charm: simple
```

Charms and Descriptors

Fill in the `initial-config-primitive` section. The <rw_mgmt_ip> token will be replaced with the IP address of your VM, allowing the charm to ssh to it.

```
initial-config-primitive:
-   seq: '1'
    name: config
    parameter:
    -   name: ssh-hostname
        value: <rw_mgmt_ip>
    -   name: ssh-username
        value: ubuntu
    -   name: ssh-password
        value: osm4u
-   seq: ‘2'
    name: touch
    parameter:
    -   name: filename
    -   value: '/home/ubuntu/first-touch'
```

Charms and Descriptors

Fill in the `config-primitive` section. This defines the primitive(s) available to run by the operator.

```
config-primitive:
-   name: touch
    parameter:
    -   name: filename
        data-type: STRING
        default-value: '/home/ubuntu/touched'
```

Validate your VNF descriptor

Use the validation tool to check that the descriptor is syntactically correct:

```
./devops/descriptor-packages/tools/upgrade_descriptor_version.py --test
<VNF_DESCRIPTOR_FILE>
```

If an error message is shown, review the descriptor and validate again. Otherwise, the descriptor is syntactically correct.

Complete your VNF Package with the charm, the cloud-init file and the logo

- Copy your compiled charm to descriptor folder (e.g. ~/hackfest_3charmed_vnfd)
  - cp -r ~/charms/builds/simple ~/hackfest_3charmed_vnfd/charms
- Download the logo and copy it into the ‘icons’ folder:
  - *https://osm-download.etsi.org/ftp/osm-3.0-three/2nd-hackfest/other/osm.png*
- Download the cloud-config file and copy it into the ‘cloud_init’ folder:
  - *https://osm-download.etsi.org/ftp/osm-3.0-three/2nd-hackfest/other/cloud-config.txt*

Generate the VNF package and upload it to the UI

- Generate the VNF Package .tar.gz
  - ~/devops/descriptor-packages/tools/generate_descriptor_pkg.sh -t vnfd -N hackfest_3charmed_vnfd

  Note: the argument -N is important if you want to keep the package folder and files after creating the package.

- Upload hackfest_3charmed_vnfd.tar.gz to OSM UI

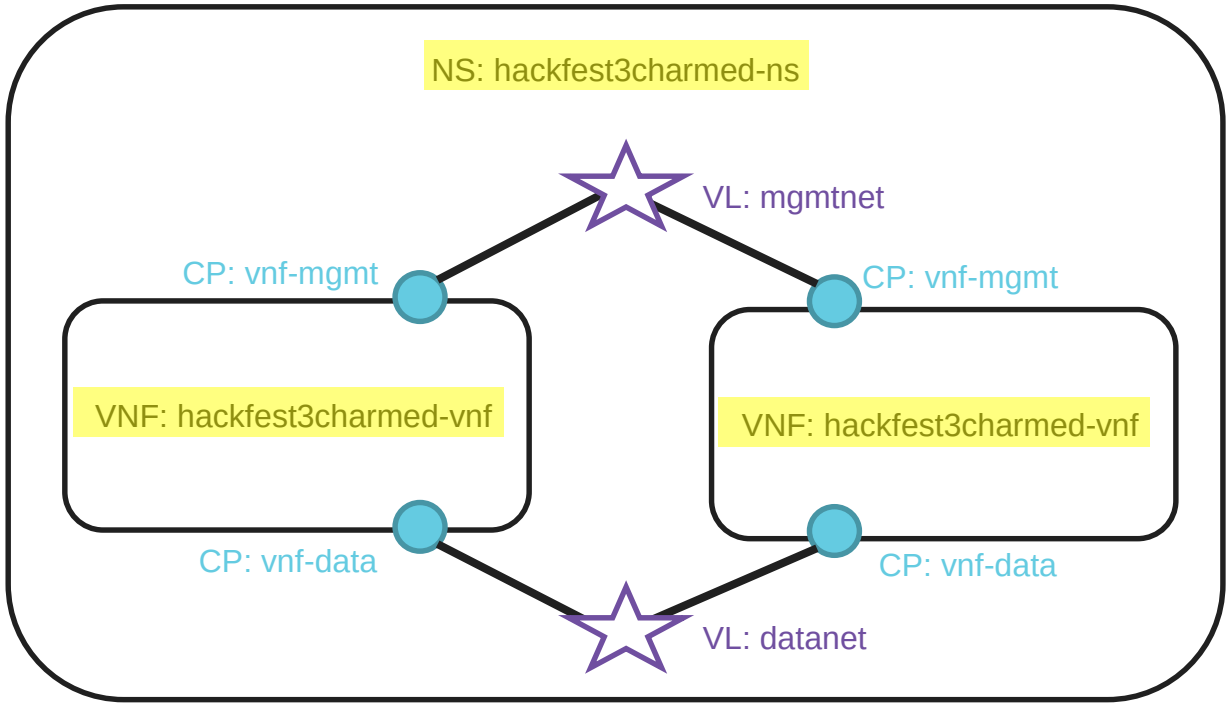
Open Source
MANO

Create NS, instantiate and run config primitives

With subtitle

ETSI

NS diagram
Changes highlighted in yellow



Deploying NS in the UI

- Go to Launchpad > Instantiate
- Select hackfest3charmed-ns and click Next
- Complete the form
  - Add a name to the NS
  - Select the Datacenter where the NS will be deployed
  - Add SSH key
- Go to the dashboard to see the instance and get the mgmt IP address of the VNF
- Connect to each VNF:
  - ssh ubuntu@<IP>
- Check that the cloud-config file was executed;
  - The file '/home/ubuntu/first-touch' should exist

Testing VNF primitives

- Check that the initial-config-primitive was executed
  - File '/home/ubuntu/first-touch' should have been created
- Go to Launchpad -> Dashboard, and open the NS instance.
- Click on a VNF, run the VNF config primitive 'touch' from the dashboard, and check that the corresponding file is created.

Relevant links

- Juju
  - https://jujucharms.com/
- Charm Developers Guide
  - https://jujucharms.com/docs/2.3/developer-getting-started
- Creating a VNF Charm
  - https://osm.etsi.org/wikipub/index.php/Creating_your_own_VNF_charm_(Release_THREE)
- Creating a VNF Package
  - https://osm.etsi.org/wikipub/index.php/Creating_your_own_VNF_package_(Release_THREE)
- Session 5 charm and descriptors
  - https://github.com/AdamIsrael/osm-hackfest

Example VNF Charms

- Using Ansible
  - https://github.com/5GinFIRE/mano/tree/master/charms/ansible-charm
- vpe-router, demoed at MWC 2016
  - https://github.com/AdamIsrael/vpe-router
- Hackfest examples
  - https://github.com/AdamIsrael/osm-hackfest

Open Source
MANO

The End

La fin