OSM Hackfest – Session 7a
Adding day-1/day-2 configuration to your VNF
Creating your first proxy charm
Adam Israel (Canonical) and Eduardo Sousa
(Whitestack)
Preparing your development environment
Install the charm tools

Install charm tools via snap:

$ sudo snap install charm
charm 2.4.5+git-1-gd62c072 from 'charms' installed

$ charm version

charmstore-client 2.3.0+snap-298+git-47-g44bc628
charm-tools 2.4.5+snap-298+git-1-gd62c072
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
```
Speed up local deployments

We'll make a few changes to your installed Juju that will speed up the deployment of charms.

# Disable automatic OS update and upgrade of every new container
$ juju model-config enable-os-refresh-update=false enable-os-upgrade=false

# Get a copy of my hackfest tools and examples
$ git clone https://github.com/AdamIsrael/osm-hackfest.git

# Cache the LXD image used by Juju for new containers
$ osm-hackfest/bin/update-juju-lxc-images
Understanding charms
Reactive Framework

• The *Reactive* programming pattern that allows a charm to respond to flags that represent a change 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 providing 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.
Anatomy of a layer

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

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

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

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

© ETSI 2017
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: ['bionic']
```

```
$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
```
• Actions are functions that can be called automatically when a VNF is initialized (day-1 configuration) or on-demand by the operator (day-2 configuration).

• 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 helper

$ mkdir actions

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

When you’re done, mark the script executable:

$ chmod +x actions/touch

This sets the "actions.touch" flag when the primitive is invoked.

```
#!/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))
```

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

Edit
`reactive/simple.py`.

This is where all reactive states are handled.

```python
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
```
Your first Reactive function

Edit
`
reactive/simple.py`

This is where all reactive states are handled.

```python
@when('sshproxy.configured')
@when_not('simple.installed')
def install_simple_proxy_charm():
    """Set the status to active when ssh configured.""

    # This sets the "simple.installed" flag so this function
    # only runs once.
    set_flag('simple.installed')

    # Tell's the VCA that the charm is ready to be used.
    status_set('active', 'Ready!')
```
React to the action

Edit `reactive/simple.py`.

This is where all reactive states are handled.

```python
@when('actions.touch')
def touch():
    """Touch a file.""
    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:
        clear_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
Examining the compiled charm again

If you examine the compiled charm now, you should see the `touch` action is now declared in `actions.yaml`, along with the actions provided by the vnfproxy layer.

```
$ 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
```
Find us at:

osm.etsi.org
osm.etsi.org/wikipub