OSM MR Hackfest – Hack 3
Automating VNF Day-1 & 2 operations

David Garcia (Canonical)
Dominik Fleischmann (Canonical)
Dmitrii Shcherbakov (Canonical)
Introduction to VNF operations
Operating workloads

Operator instance

VNFD

OSM

Workload

How can we **operate** this workload?
VNFD

Metadata

Operations package - “Charm”
- Lifecycle
- Configuration
- Operation
- Integration

DECLARATIVE

CODE
Lifecycle

• Install
• Scale out
• Upgrade
• Report status
Configuration

- Initial configuration
- Change configuration
- YAML

```
options:
  auth-type:
    type: string
    description: Sets the authentication type
    default: keystone
  port:
    type: int
    description: Sets the port
```
Operations… for example

- Backup
- Monitor
- Debug
- Add users, policies, rules, etc.
- Manage certificates, keys, etc.
- Rotate logs
- **Custom operations**

Each ‘operations primitive’ is a function call that takes parameters and produces a result.
Integration
Native and Proxy Charms

Operator Code “Charm”

Workload Code

OSM
“Native” Charm

Charm

+ 

Workload

“Proxy” Charm

Charm

+ 

Workload
Kubernetes “operators” are charms

Workload containers

Operator container

Charm
Summary of charms

- Part of VNFD
- Manage:
  - Lifecycle
  - Configuration
  - Operations
  - Integration
- Machine and kubernetes workloads
- Charms are **code**
Python Operator Framework
“Make it easy to write a Kubernetes operator in pure Python”
Python Operator Framework

- Class hierarchy
  - for modeling services and integrations;
- Event system;
- Persistence layer;
- Minimal dependencies.
Why Pure Python?

- High-level
- Widely known
- Low entry barrier
- Good for writing integration code
- Simple debugging & testing
- Cross-architecture
Unified operator framework for K8s and legacy workloads
Event Driven Programming Model
class MyCharm(CharmBase):

def __init__(self, *args):
    super().__init__(*args)
    self.framework.observe(self.on.install, self.on_install)
    self.framework.observe(self.on.config_changed, self.on_config_changed)
    self.framework.observe(self.on.start, self.on_start)
    self.framework.observe(self.on.leader_elected, self.on_leader_elected)
    self.framework.observe(self.on.db_relationJoined, self.on_db_relationJoined)
    self.framework.observe(self.on.db_relationChanged, self.on_db_relationChanged)

def on_install(self, event):
    packages = self.model.config[‘packages’]
    # Install relevant packages...

def on_config_changed(self, event):
    # Re-render a template here.
Graph change events

“A multi-VDU VNF integrated with a single-VDU VNF”
Graph change events

Event 1: Deploy 1 unit of app A
Event 2: Deploy 1 unit of app B
Event 3: Relate app A and app B
Event 4: appA/0 and appB/0 observe each other
Event 5: Scale app A: add 1 unit
Event 6: Scale app A: add 1 unit
Event 7: appA/1 and appB/0 observe each other
Event Horizon
Event Horizon
How to write a charm?
Charm Declaration

my-charm/
  metadata.yaml
  config.yaml
  ...
src/charm.py
lib/
  ...python modules

from ops.main import main
from ops.charm import CharmBase

class MyCharm(CharmBase):
    ...

if __name__ == '__main__':
    main(MyCharm)
Installing a Workload on a Machine

class LBCharm(CharmBase):

    def __init__(self, *args):
        super().__init__(*args)
        self.framework.observe(self.on.install, self.on_install)

    def on_install(self, event):
        packages = self.model.config['packages']
        # install the specified packages...
        apt_install(packages)
def on_install(self, event):

    self.model.pod.set_spec({
        'containers': [{
            'name': self.meta.name,
            'ports': [{
                'containerPort': self.model.config['http_port'],
                'protocol': 'TCP',
            }],
            'imageDetails': {
                'imagePath': self.model.config['image-path'],
                'username': self.model.config['registry-user'],
                'password': self.model.config['registry-password']
            },
            'config': {
                'CONFIG_KEY': 'CONFIG_VALUE',  # ...
            }
        }
    })
class LBCharm(CharmBase):

    # web/x unit added to the model and seen by a load-balancer
    def on_backend_relation_joined(self, event):
        relation_data = event.relation.data[event.unit]
        backend_addr = relation_data.get('ingress-address')

        # Trigger load-balancer re-configuration
        self.on.backend_added.emit(backend_addr)
## Relation Data

<table>
<thead>
<tr>
<th>App A Data</th>
<th>App B Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/0 Data</td>
<td>B/0 Data</td>
</tr>
<tr>
<td>A/1 Data</td>
<td>B/1 Data</td>
</tr>
<tr>
<td>A/2 Data</td>
<td></td>
</tr>
</tbody>
</table>

© ETSI 2020
Relation Data Change Event

Writing to db/0's data dictionary (bag) notifies all remote units.

- Relation set: $x=y$
- db/0
  - code
  - data
- client/0
  - code
  - data
- db/1
  - code
  - data
- client/1
  - code
  - data
- db/2
  - code
  - data
- client/2
  - code
  - data
Configuration Changes

class LBCharm(CharmBase):
    def on_config_changed(self, event):
        ctxt = {
            'request_timeout': self.model.config['request-timeout'],
            # Other template context values...
        }
        # Render a jinja2 template with a provided context.
        env = Environment(loader=FileSystemLoader('templates'))
        template = env.get_template('haproxy.conf.j2')
        rendered_content = template.render(ctxt)
Charm-defined Events

```python
class LBStarted(EventBase): ...

class LBCharmEvents(EventsBase):
    lb_started = EventSource(LBStarted)

class LBCharm(CharmBase):
    on = LBCharmEvents()

    def on_start(self, event):
        # start the lb service ...
        self.on.lb_started.emit()
```

Actions

- Admin-triggered one-time events;
- Action event names have an "_action" suffix:
  ```python
  self.framework.observe(self.on.upgrade_action, self)
  ```
- **ActionEvent** type exposes a useful API to work with actions:
  ```python
  def on_upgrade_action(self, event):
    my_param = event.params["my_param"]

    event.log("Upgrade progress: 42\%")

    event.set_results({"success": "false"})
    event.fail("Almost got it but not quite.")
  ```
- ActionEvents cannot be deferred.
class DBCharm(CharmBase):

    def on_cluster_initialized(self, event):
        if not self.model.unit.is_leader():
            raise RuntimeError('initial unit isn’t a leader')

        cluster_relation = self.model.get_relation('cluster')
        cluster_app_data = cluster_relation.data[self.model.app]
        cluster_app_data['initial_unit'] = self.model.unit.name
        cluster_app_data['cluster_id'] = event.cluster_id
Application Relation Data: Example
Application Relation Data

- Relation data of an app per relation;
- Only modifiable by a leader unit;
- Each side of a relation has its own leader unit and app relation data modifiable by that unit;
- App data on a peer relation allows data written by a leader unit to be exposed to peers;
- Model representation:

```python
rel = self.model.get_relation('cluster')
app_data = rel.data[rel.app]
cluster_id = app_data.get("cluster-id")
```
class LBCharm(CharmBase):

    state = StoredState()

    def __init__(self, *args):
        self.framework.observe(self.on.start, self)
        self.framework.observe(self.on.config_changed, self)
        self.state.set_default(started=False)

    def on_start(self, event):
        # start the LB service...
        self.state.started = True

    def on_config_changed(self, event):
        if self.state.started:
            # write config files and reload...
Stored State: Summary

- Each framework **Object** has a **Handle** as its persistent ID;
- Object data is persisted via attributes of type StoredState;
- Only basic Python types may be stored (lists, sets, dicts, ...);
- **StoredState.set_default** is available to initialize the state once.
class LBCharm(CharmBase):

    state = StoredState()

    def backend_relation_changed(self, event):
        if self.state.tls_configured:
            # write config files and reload...
        else:
            # will fire on the next hook execution
            event.defer()
        return
Model Representation

Model
  unit
  app
  config
  relations[name]
    relation
      name
      id
      data[unit or app]
    units
  storages
  resources
Convenience Methods of a Model

Easy lookup of relevant objects:

def on_config_changed(self, event):
    # raises an exception if multiple relations exist
    db_relation = self.model.get_relation("db")

    binding = self.model.get_binding(db_relation)
    addr = binding.network.ingress_address
mkdir charm-hello-world && cd charm-hello-world && git init
mkdir hooks src lib mod
git submodule add https://github.com/canonical/operator mod/operator
touch metadata.yaml config.yaml src/charm.py
chmod +x src/charm.py

ln -s ../mod/operator/ops lib/ops
ln -s ../src/charm.py hooks/install
"Hello World" Charm Metadata

metadata.yaml:

name: hello-world
summary: a "Hello, World" charm.
description: "Hello, World."
series:
  - bionic
"Hello World" Charm Code


# ...

def __init__(self, *args):
    super().__init__(*args)
    self.state.set_default(logged_hello=False)
    self.framework.observe(self.on.install, self.on_install)

def on_install(self, event):
    logger.info('Hello, world')
    self.state.logged_hello = True
Question

If you have an HTTP service scaled to 1 VDU, and then it’s scaled out to 3 VDUs, does OSM balance the load between the workloads?
Get ready for hands-on

Ensure you have an Ubuntu installed

Options:

- AWS instance
- Multipass
- VM in Virtualbox

Track your progress here:

https://docs.google.com/spreadsheets/d/1dXpPJP6XfWg8GHfQyNyT_49HSh28dvgVas0e62RGU8o/edit?usp=sharing
Operations in workloads
Let’s get hands-on: Initial setup

```
sudo snap install juju --classic
sudo snap install microk8s --classic
sudo adduser ubuntu microk8s
newgrp microk8s
microk8s.status --wait-ready
microk8s.enable storage dns
juju bootstrap microk8s
juju add-model osm
juju deploy osm
```
Operating workloads

# Lifecycle
juju scale-application nbi-k8s 3

# Configuration
juju config nbi-k8s image=opensourcemano/nbi:7.0.1

# Operation
juju run-action mongodb-k8s/0 backup

# Integration
juju deploy cs:~charmed-osm/keystone-k8s
juju relate nbi-k8s keystone-k8s
juju relate keystone-k8s mariadb-k8s
Charms walkthrough

- Zookeeper charm:
  - https://github.com/charmed-osm/charm-k8s-zookeeper
- Kafka charm:
  - https://github.com/charmed-osm/charm-k8s-kafka
- Zookeeper interface
  - https://github.com/charmed-osm/interface-zookeeper
VNFD walkthrough

- K8s charm
  - Base code

```
vnfd-catalog:
vnfd:
  - id: hackfest-simple-k8s-vnfd
    name: hackfest-simple-k8s-vnfd
    connection-point:
      - name: mgmtnet
        mgmt-interface:
          cp: mgmt
        kdu:
          - name: mykdu
            juju-bundle: cs:~dominik.f/bundle/hf-k8s-bundle-0
        k8s-cluster:
          nets:
            - id: mgmtnet
              external-connection-point-ref: mgmt
```
VNFD walkthrough

- Proxy charm
  - **Base code**

---

Charm

+ Workload

---

```
vnfd:vnfd-catalog:
  vnfd:
    - ...
      mgmt-interface:
        cp: vnf-mgmt
    ...
  vnf-configuration:
    juju:
      charm: simple-proxy
    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"
  config-primitive:
  - name: touch
    parameter:
      - name: filename
        default-value: "/home/ubuntu/touched"
        data-type: STRING
```
VNFD walkthrough

- Native charm
  - **Base code**

Charm +

Workload

vnfd:vnfd-catalog:
  vnfd:
    - ...
      mgmt-interface:
        cp: vnf-mgmt
      vdu:
        - id: mgmtVM
          ...
        cloud-init-file: cloud-config.txt
    vdu-configuration:
      juju:
        charm: simple-native
        proxy: False
        config-access:
          ssh-access:
            required: True
            default-user: ubuntu
        initial-config-primitive:
          - seq: "1"
            name: touch
            parameter:
              - name: filename
                value: "/home/ubuntu/first-touch"
        config-primitive:
          - name: touch
            parameter:
              - name: filename
                data-type: STRING
                default-value: "/home/ubuntu/touch"