

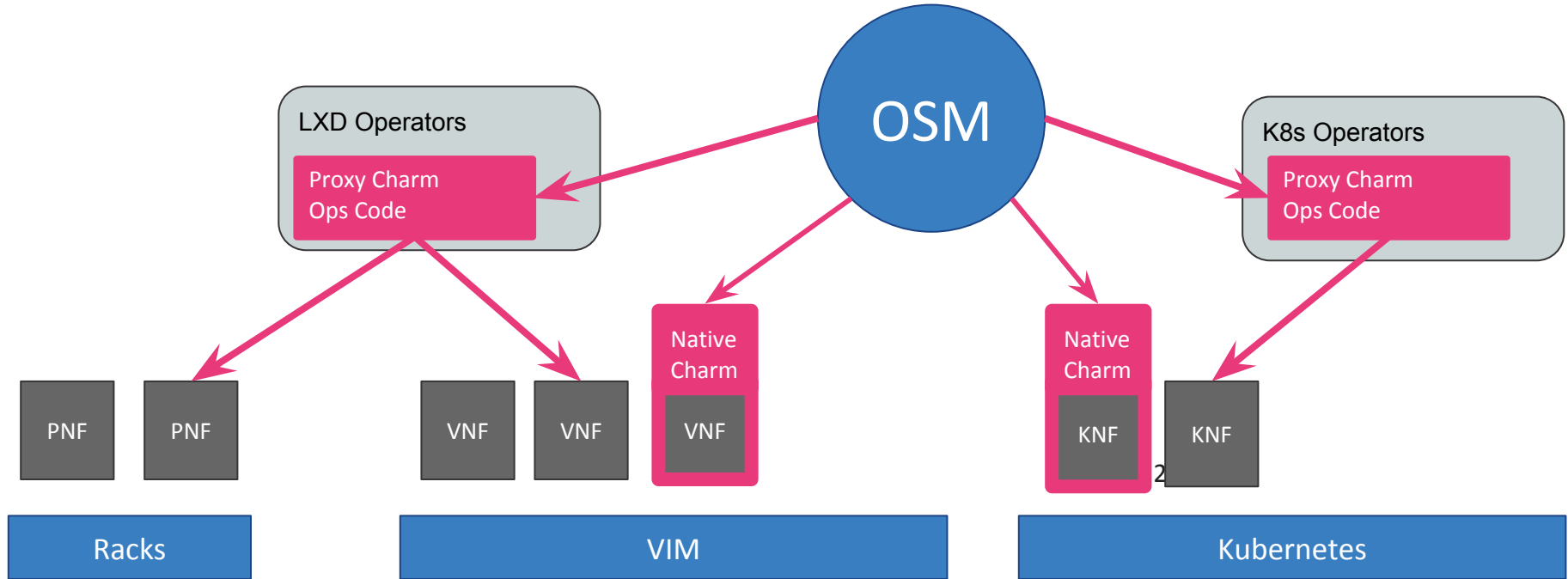
Open Source
MANO

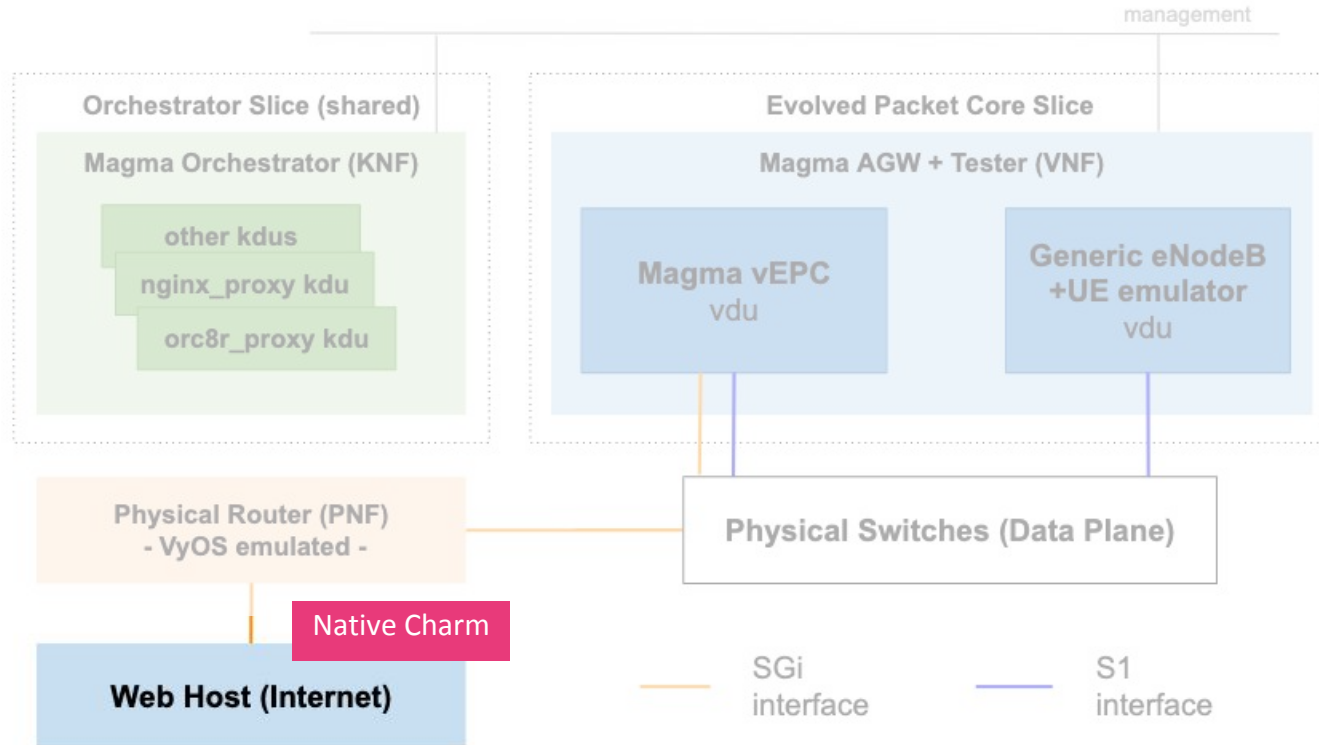
OSM-MR#9 Hackfest

Day 1-2 CNF Operations

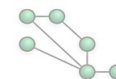
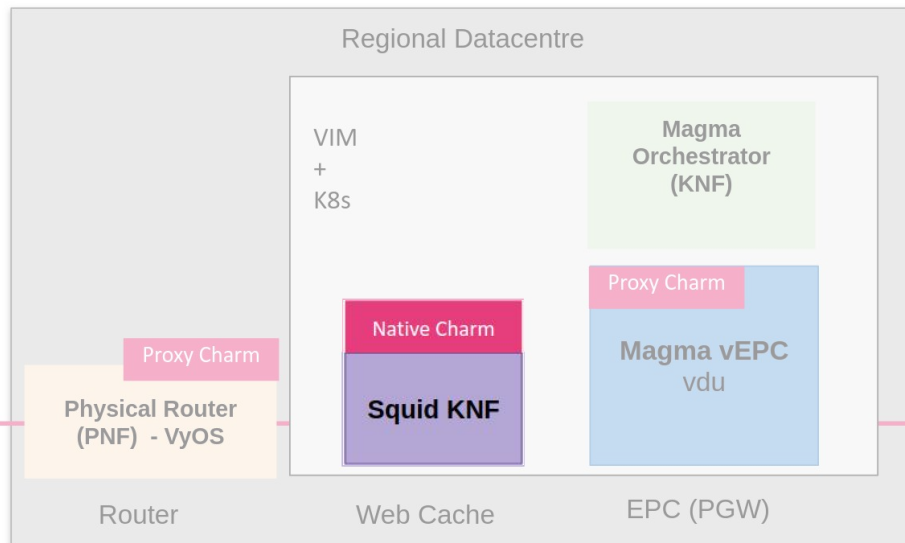
Dominik Fleischmann, David Garcia, Mark Beierl (Canonical)

OSM drives NFs through Charm Ops Code

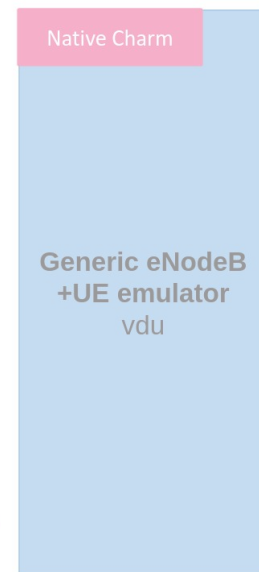




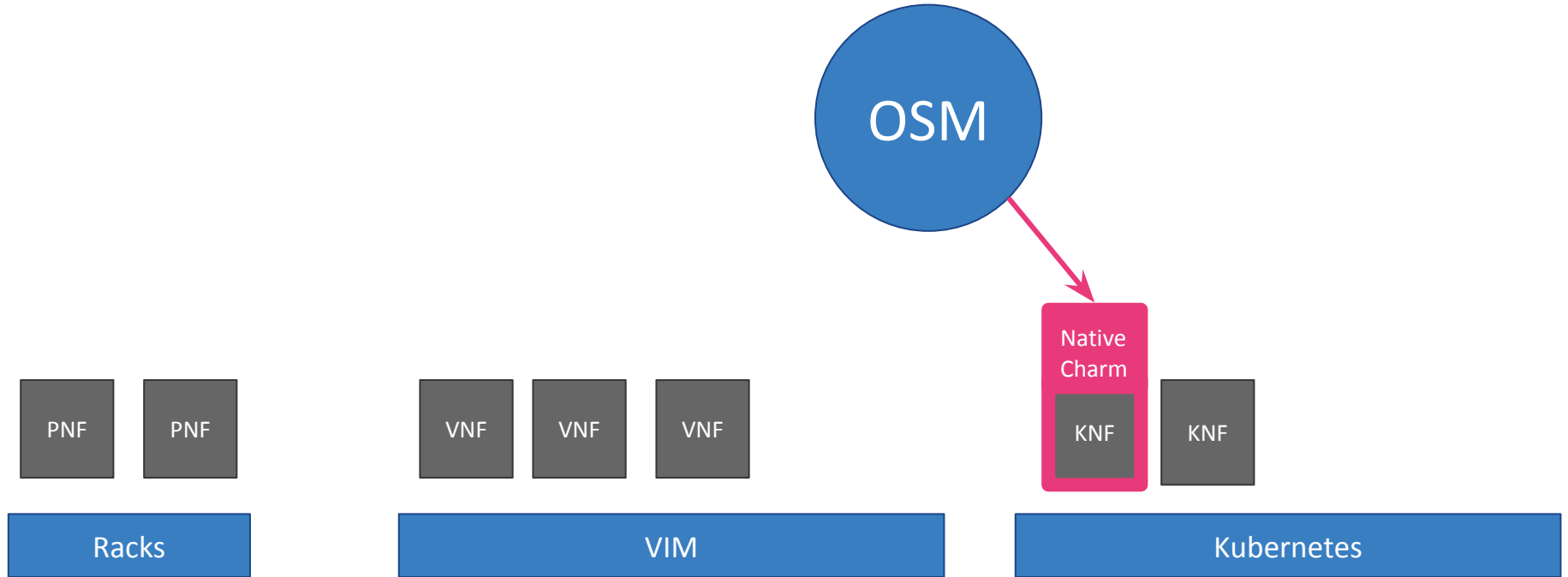
Cellular Data Path



SDN



OSM drives NFs through Charm Ops Code



Application Model in YAML

The Juju VCA in OSM provides a universal modelling language.

A YAML 'bundle' describes the app architecture of the KNF.

```
bundle: kubernetes
applications:
  mariadb-k8s:
    charm: cs:~juju/mariadb-k8s-2
    scale: 1
  mediawiki-k8s:
    charm: cs:~juju/mediawiki-k8s-3
    scale: 1
relations:
- - mariadb-k8s:server
  - mediawiki-k8s:db
```

Charms are ‘operations packages’ for apps

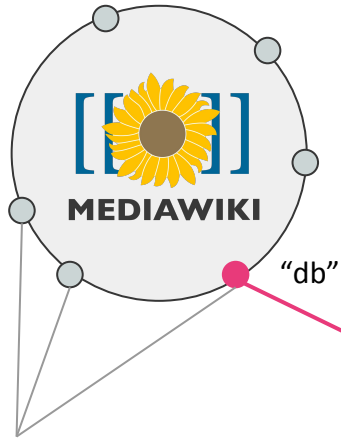
A charm is a package of ops code for a KNF app component.

Charms are reusable and composable. So the ‘mariadb’ charm might be used in many CNFs and many VNFs.

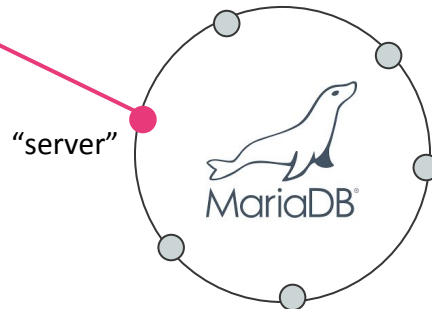
Charms are written in Python for best results.

```
bundle: kubernetes
applications:
  mariadb-k8s:
    charm: cs:~juju/mariadb-k8s-2
    scale: 1
  mediawiki-k8s:
    charm: cs:~juju/mediawiki-k8s-3
    scale: 1
relations:
- - mariadb-k8s:server
- - mediawiki-k8s:db
```

Model includes integration between charms



Named points
of integration



```
bundle: kubernetes
applications:
  mariadb-k8s:
    charm: cs:~juju/mariadb-k8s-2
    scale: 1
  mediawiki-k8s:
    charm: cs:~juju/mediawiki-k8s-3
    scale: 1
relations:
- - mariadb-k8s:server
- - mediawiki-k8s:db
```


Charms package lifecycle and action scripts

Lifecycle scripts

- install
- config
- update
- remove
- scale

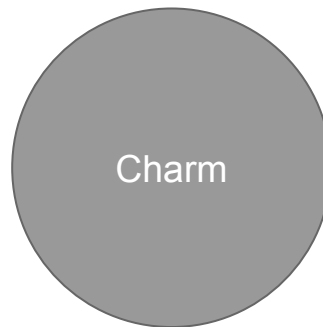
Integration scripts

- relate-mysql
- relate-ldap
- relate-proxy
- relate-...

“Action” scripts are OSM Primitives

“action: backup”
“action: restore”
“action: scan-viruses”
“action: health-check”
“action: add-repo”
“action: ...”
“action: ...”
“action: ...”

These are your operations primitives.



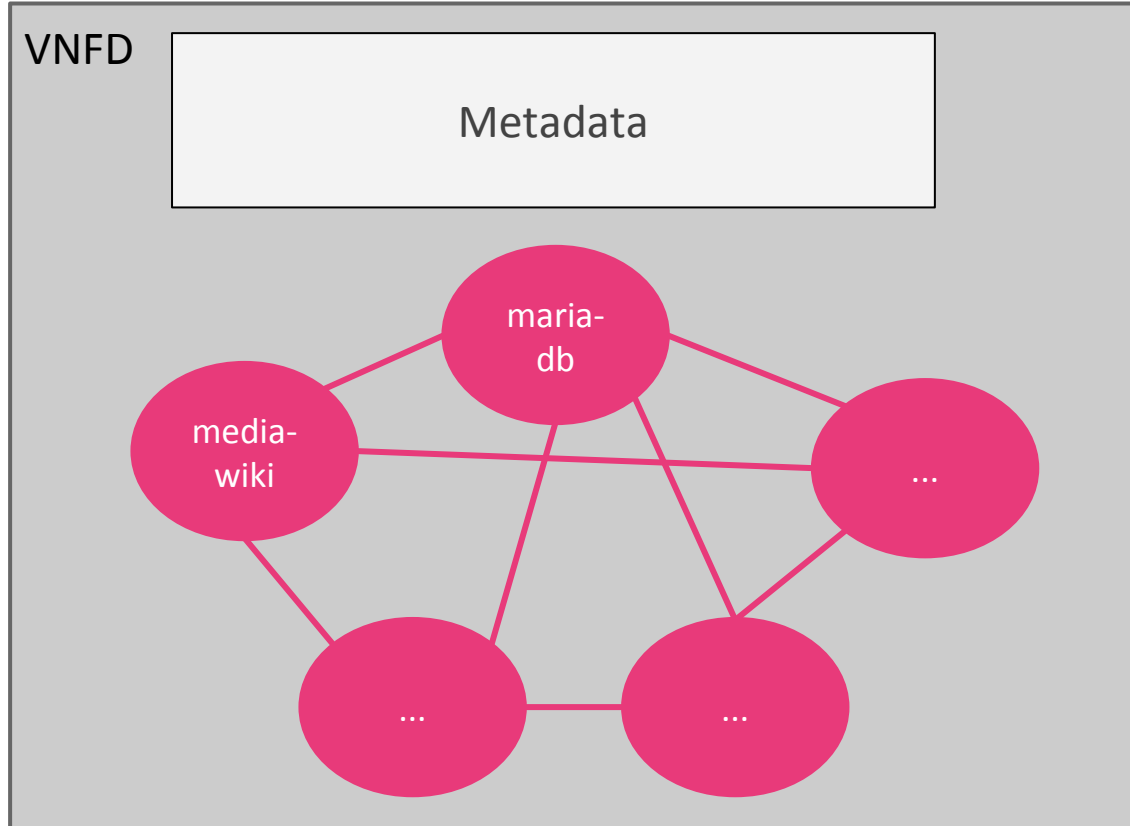
Charms describe **Action Parameters**

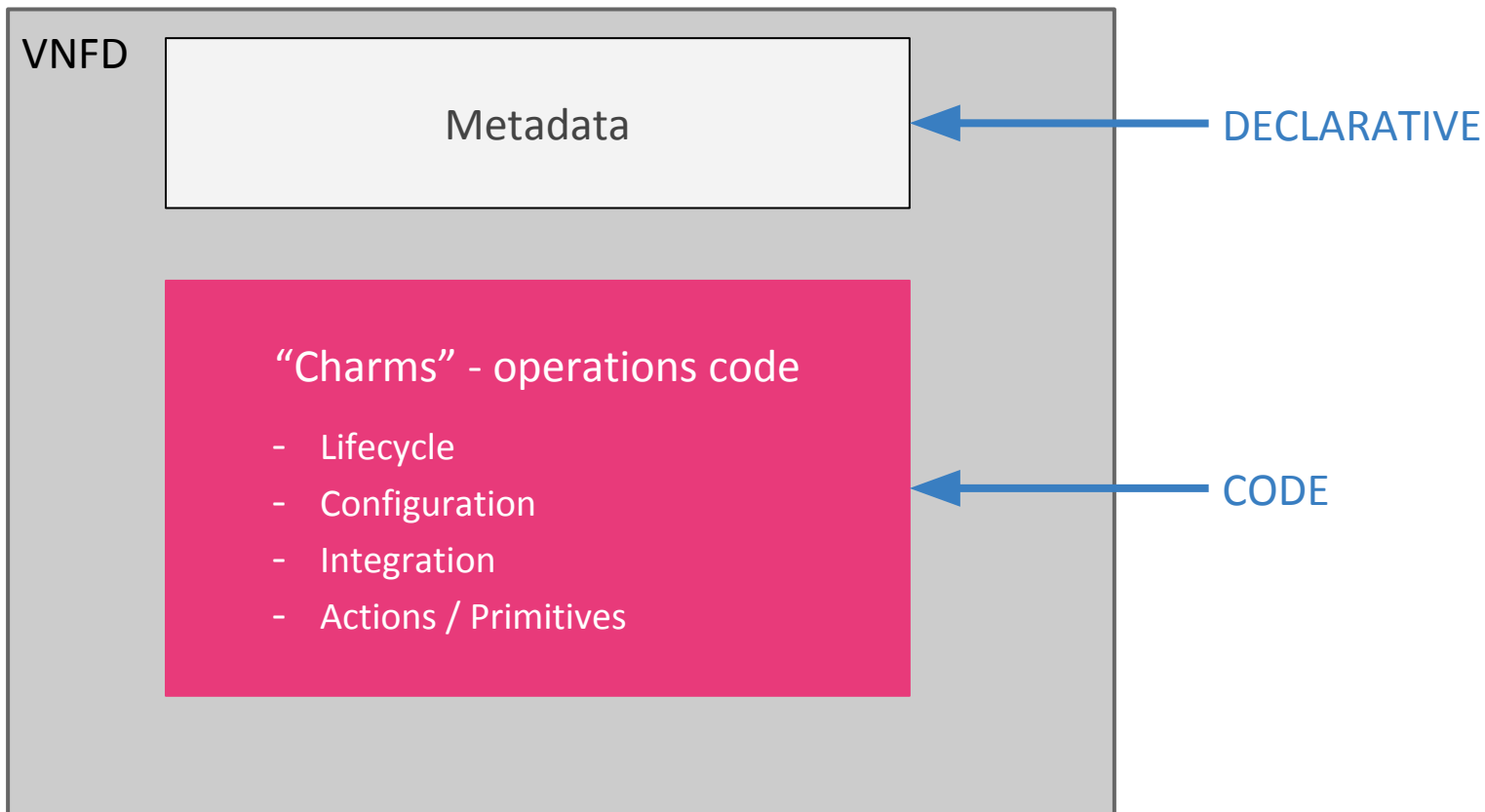
Each Action is a script, usually in Python or Bash.

Charm metadata describes the action parameters.

```
backup:
  description: "Do a mariadb backup"
  params:
    target:
      description: "The unit in which it should be
performed the action. (ANY, PRIMARY, SECONDARY)"
      type: string
      default: "ANY"
    path:
      description: "Path for the backup inside the unit"
      type: string
      default: "/data"
restore:
  description: "Restore from a MariaDB Backup"
  params:
    path:
      description: "Path for the backup inside the unit"
      type: string
      default: "/data"
```

One VNF is many apps and integration



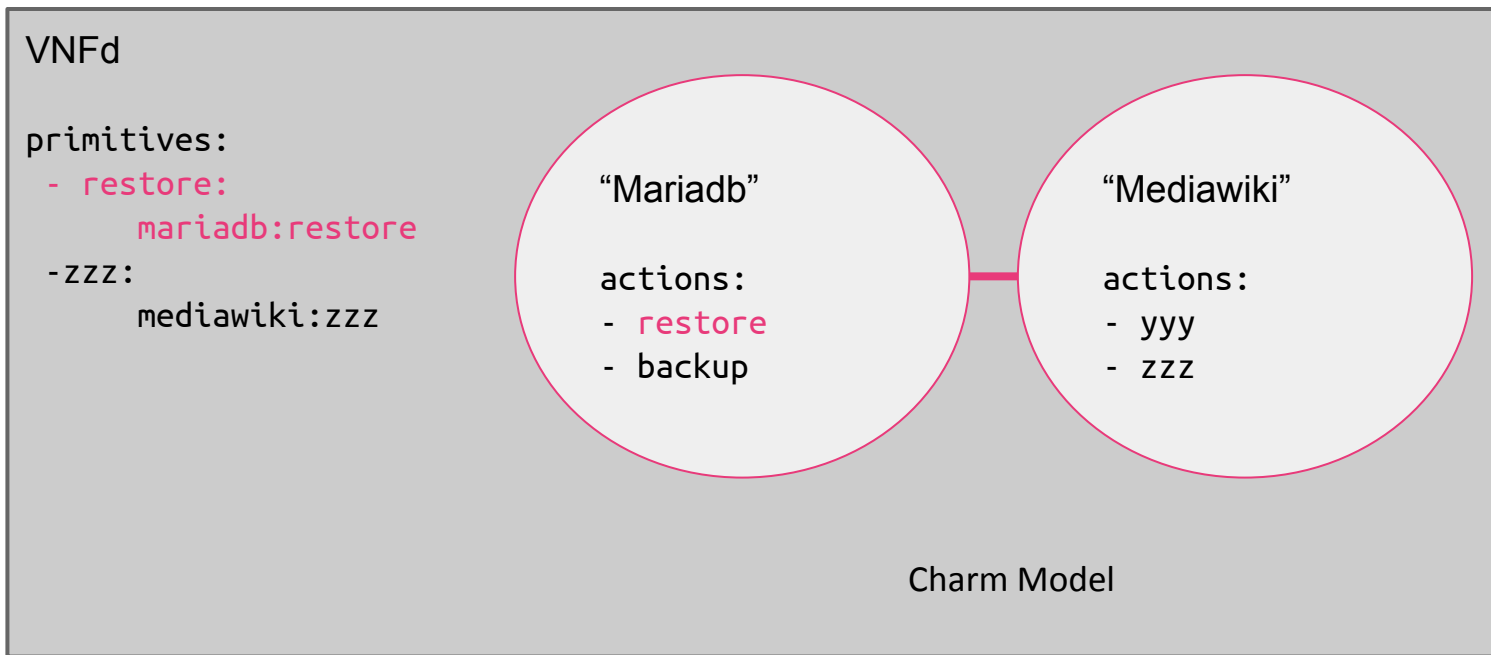


VNFD wraps up a bundle of charms

The Kubernetes Deployment Unit is defined with a Bundle that can include many charms for different application components.

```
vnfd-catalog:
  vnfd:
  - id: squid-vnf
    name: squid-vnf
    connection-point:
      - name: mgmtnet
    mgmt-interface:
      cp: mgmt
    kdu:
      - name: mediawiki-kdu
        juju-bundle: bundle.yaml
        kdu-configuration:
          config-primitive:
            - name: backup
    ...
```

VNFs map OSM Primitives to Charm Actions



VNFds map OSM Primitives to Charm Actions

Primitives are declared as a list
in the VNFd kdu-configuration
section

```
kdu:  
- name: mediawiki-kdu  
  juju-bundle: bundle.yaml  
  kdu-configuration:  
    config-primitive:  
      - name: restore  
        parameter:  
          - name: application-name  
            data-type: STRING  
            default-value: mariadb  
          - name: path  
            data-type: STRING  
            default-value: ""
```

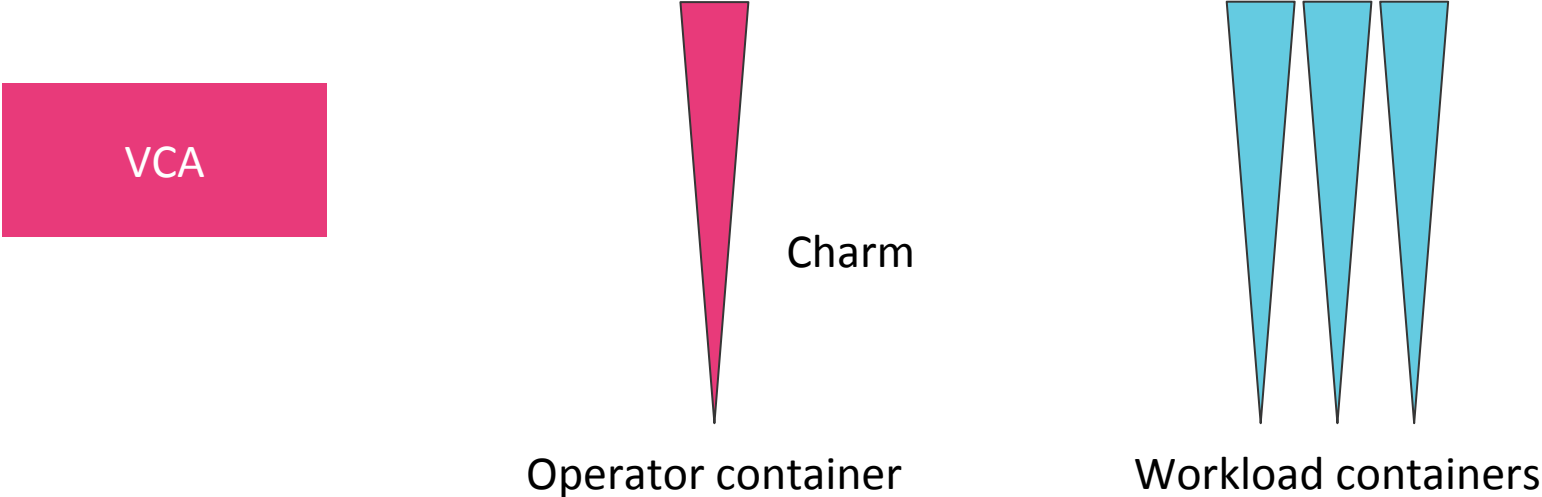
VNFds map OSM Primitives to Charm Actions

The application-name specifies which charm is providing the actual action for the primitive.

The VNFd can provide default action parameter values.

```
kdu:
- name: mediawiki-kdu
  juju-bundle: bundle.yaml
  kdu-configuration:
    config-primitive:
      - name: restore
        parameter:
          - name: application-name
            data-type: STRING
            default-value: mariadb
          - name: path
            data-type: STRING
            default-value: ""
```


KNF charms run in their own container on K8s



So, you need to provide...

VNFD Metadata

- Bundle
- Primitive mapping

Bundle

- Charms
- List of integrations



Charm



Charm



Charm

- Action scripts
- Metadata describing action parameters



Open Source
MANO

Example KNF

A Web Proxy using Squid

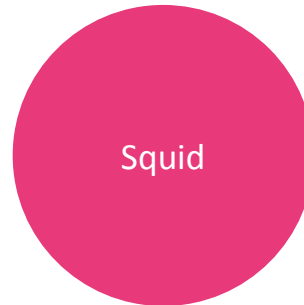


YAML Bundle of Squid

The model bundle will always be located in the juju-bundles folder of our package.

The Web Proxy model is very simple, it has a single charm, which is **squid**.

```
description: Squid Bundle
bundle: kubernetes
applications:
  squid:
    charm: '../charms/squid'
    scale: 1
```



Charms on Kubernetes

A CNF Charm is almost identical to the charms for VNF and PNF workloads.

`series: kubernetes`
in charm metadata says that it runs on K8s.

```
name: squid
summary: Charm for Squid on k8s
maintainers:
  - Dominik Fleischmann <dominik.fleischmann@canonical.com>
description: |
  A charm that will deploy Squid on Kubernetes
tags:
  - misc
series:
  - kubernetes
deployment:
  type: stateful
  service: loadbalancer
storage:
  docker:
    type: filesystem
    location: /srv/docker/squid
  spool:
    type: filesystem
    location: /var/spool/squid
```

The ‘install’ hook on K8s uses pod spec

Charms running on Kubernetes generate a pod spec to run their application.

```
def make_pod_spec(self):
    config = self.framework.model.config
    ports = [{"name": "squid", "containerPort": config["port"],
              "protocol": "TCP"}]

    spec = {
        "containers": [{
            "name": self.framework.model.app.name,
            "image": config["image"],
            "ports": ports,
        }],
    }
    return spec

def _apply_spec(self, spec):
    # Only apply the spec if this unit is a leader
    if self.framework.model.unit.is_leader():
        self.framework.model.pod.set_spec(spec)
        self.state.spec = spec
```

Squid charm has two Actions

Charm metadata describes the action parameters.

```
addurl:
  description: "Add squid config"
  params:
    url:
      description: "URL that will be allowed"
      type: string
      default: ""

deleteurl:
  description: "Delete allowed URL squid config"
  params:
    url:
      description: "URL that will stop to be allowed"
      type: string
      default: ""
```

Operations code (bash)

Actions can be written in bash for very simple cases.

```
#!/bin/bash
URL=`action-get url`

if ! grep -Fxq "http_access allow allowedurls"
/etc/squid/squid.conf
then
    sed -i '/^# And finally deny all ./i http_access allow
allowedurls\n' /etc/squid/squid.conf
fi

sed -i "/^http_access allow allowedurls.*/i acl allowedurls
dstdomain \.${URL}" /etc/squid/squid.conf

kill -HUP `cat /var/run/squid.pid`
```


Operations code (python)

It is common to write actions in Python using the standard Operator Framework.

```
def on_deleteurl_action(self, event):
    """Handle the deleteurl action."""
    url = event.params["url"]

    line_to_delete = "acl allowedurls dstdomain .{}".format(url)
    line_deleted = False

    with open("/etc/squid/squid.conf", "r") as f:
        lines = f.readlines()
    with open("/etc/squid/squid.conf", "w") as f:
        for line in lines:
            if line_to_delete not in line:
                f.write(line)
            else:
                line_deleted = True

    if line_deleted:
        event.set_results({"output": "URL deleted succesfully"})
        subprocess.check_output("kill -HUP `cat /var/run/squid.pid`",
                                shell=True)
    else:
        event.fail("No URL was deleted")
```

VNFs for Squid

This will be the kdu section

For our Squid KNF.

It states our bundle and the primitives that will be exposed to OSM.

```
kdu:  
- name: squid-kdu  
  juju-bundle: bundle.yaml  
  kdu-configuration:  
    config-primitive:  
      - name: addurl  
        parameter:  
          - name: application-name  
            data-type: STRING  
            default-value: squid  
          - name: url  
            data-type: STRING  
            default-value: ""  
      - name: deleteurl  
        parameter:  
          - name: application-name  
            data-type: STRING  
            default-value: squid  
          - name: url  
            data-type: STRING  
            default-value: ""
```

KNF Network Service descriptor

- `hackfest_squid_cnf_ns`
- Same structure as a descriptor for a VNF based NS.

Getting our KNF Package

Execute the following commands:

```
cp -r /home/ubuntu/examples/squid-knf/ .  
cd squid-knf  
tar -czf hackfest_squid_cnf.tar.gz hackfest_squid_cnf  
tar -czf hackfest_squid_cnf_ns.tar.gz hackfest_squid_cnf_ns
```

Instantiate our KNF

Execute the following commands:

```
osm upload-package hackfest_squid_cnf.tar.gz
```

```
osm upload-package hackfest_squid_cnf_ns.tar.gz
```

```
osm ns-create --ns_name webcache --nsd_name squid-cnf-ns --vim_account  
etsi-openstack-x --config '{vld: [ {name: mgmtnet, vim-network-name: sgi}  
]}'
```

Testing our KNF

Check the status of our deployment with `osm ns-list`

If it is in a READY state execute the following commands:

```
NSID=`osm ns-list | grep webcache | awk '{ print $4 }'`  
PROXY_IP=`kubectl --kubeconfig ~/kube.yaml get services -n  
squid-kdu-$NSID | grep 'squid' | awk '{print $4}'`  
export https_proxy=http://$PROXY_IP:3128  
curl https://google.com
```

The curl command should fail with the “HTTP code 403” error.

Use a separate Session for the actions!!

To execute the actions use a different session than the one where you set the squid proxy

To reset the proxy execute the following:

```
export https_proxy=
```

Execute Operations

With the CNF being deployed, execute the following command:

```
osm ns-action webcache --vnf_name squid-vnf --kdu_name squid-kdu  
--action_name addurl --params '{application-name: squid, url:  
google.com}'
```

Check the status of the execution with:

```
osm ns-op-list webcache
```

Now this curl command should return the correct output:

```
curl https://google.com
```


Execute Operations

With the CNF being deployed, execute the following command:

```
osm ns-action webcache --vnf_name squid-vnf --kdu_name squid-kdu  
--action_name deleteurl --params '{application-name: squid, url:  
google.com}'
```

Check the status of the execution with:

```
osm ns-op-list webcache
```

Now this curl command should give you a “HTTP code 403” error again:

```
curl https://google.com
```