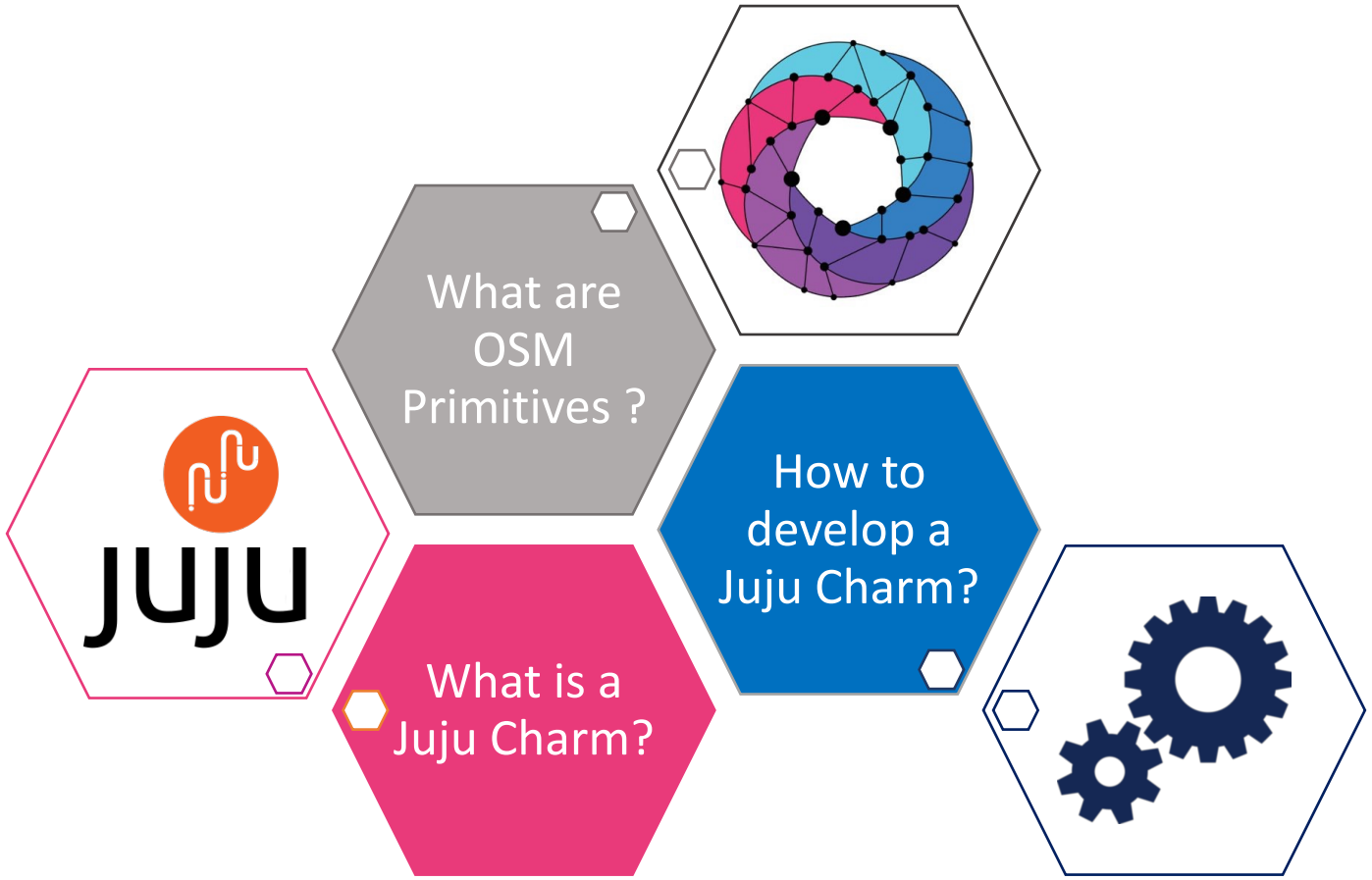
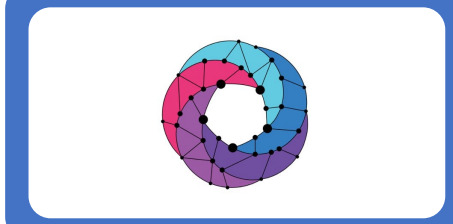
Introducing OSM Primitives and Juju Charms

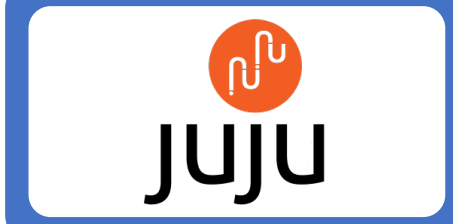


OSM Primitives



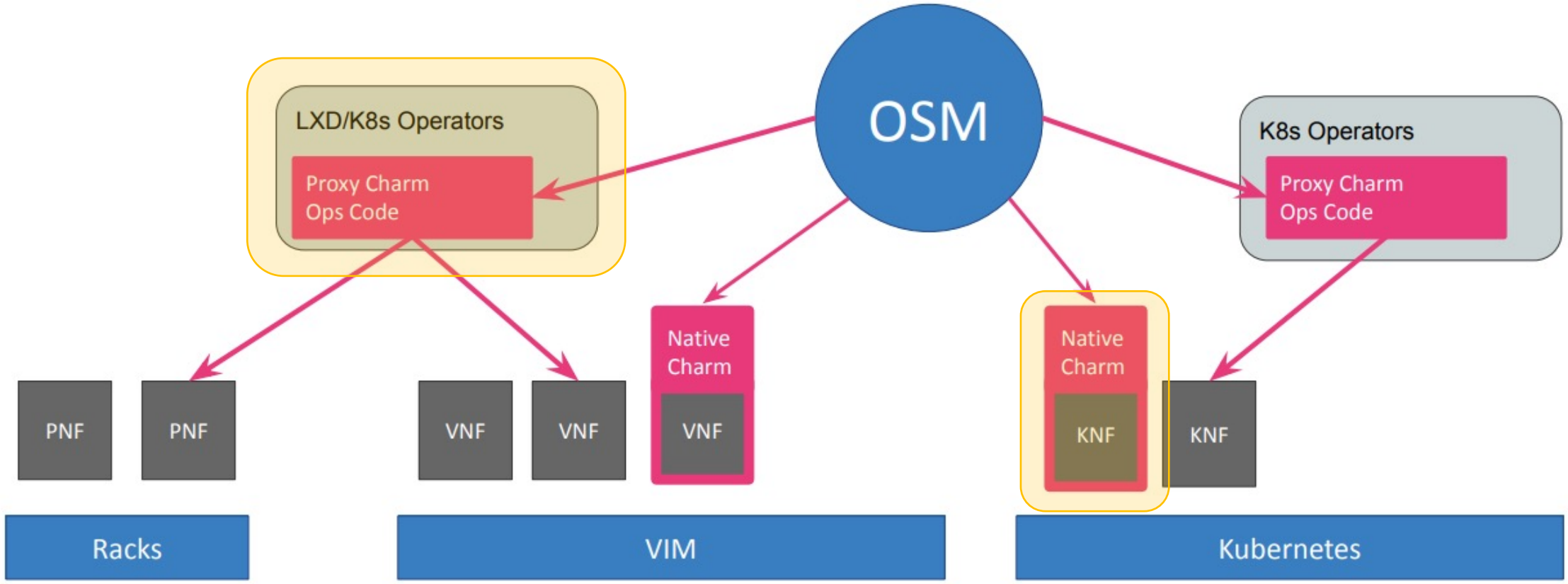
Actions exposed by the operators to perform VNF operations

Provide day-1 and day-2 operations



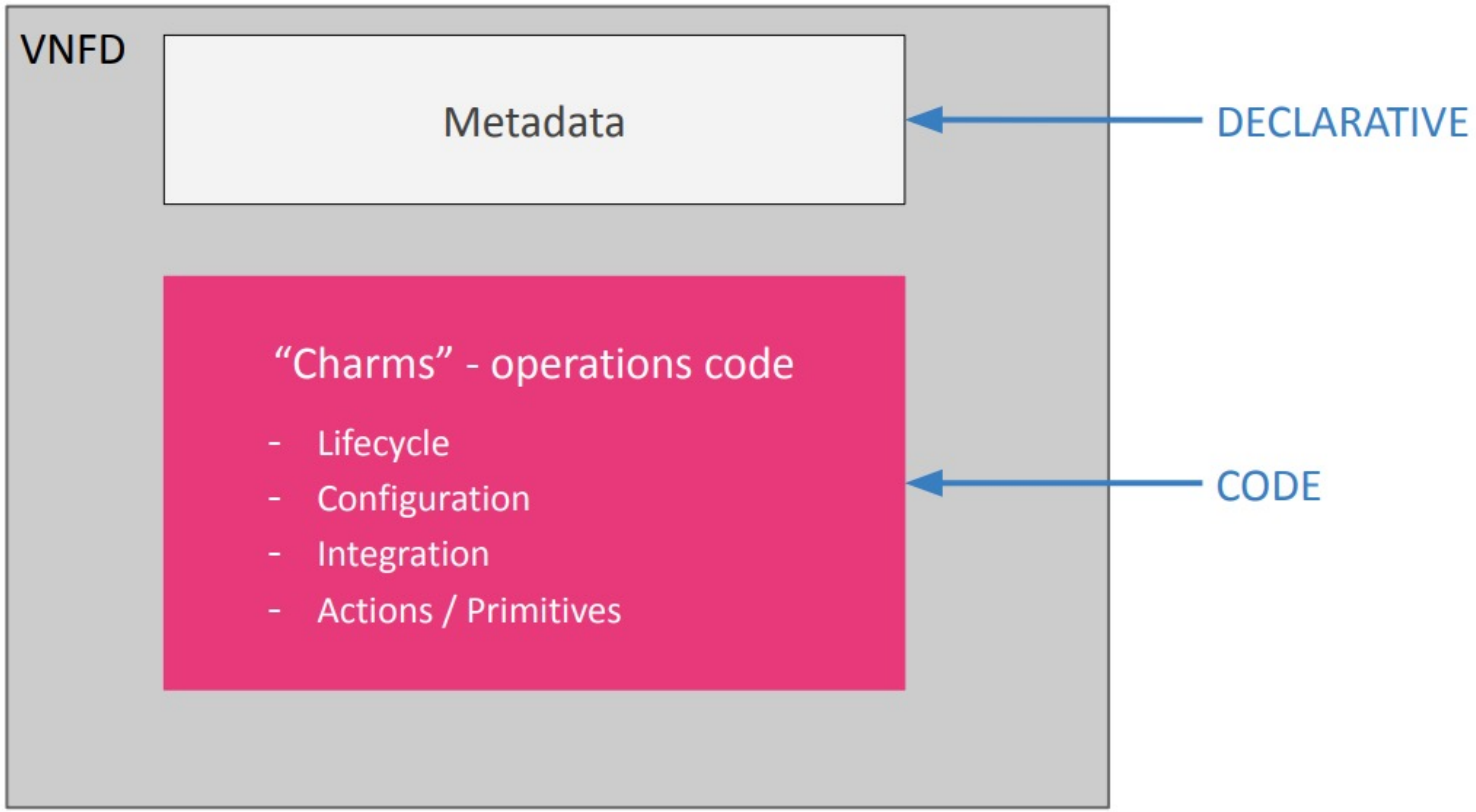
Are offered via charms’ actions

OSM Primitives (Charms)



Source: OSM MR10 Hackfest

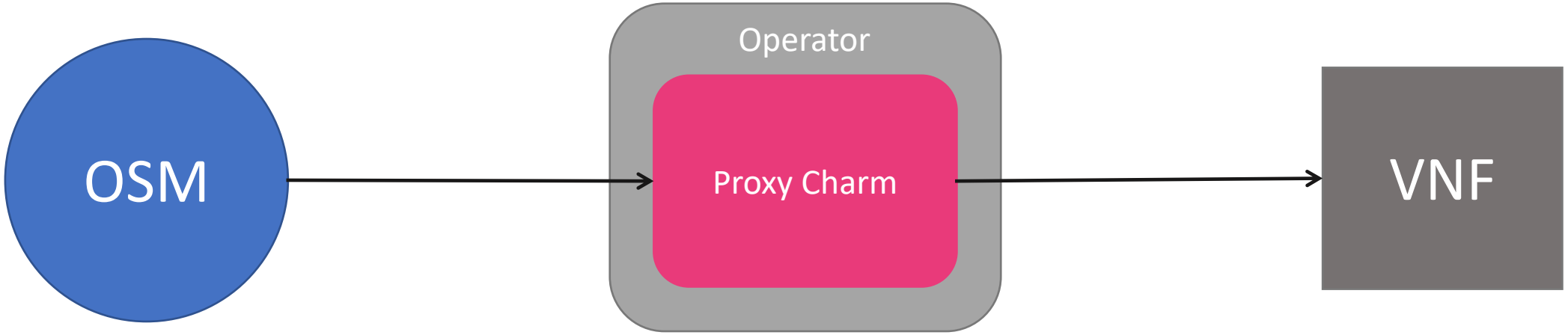
How to use Charms to perform VNF operations?



Source: OSM MR10 Hackfest

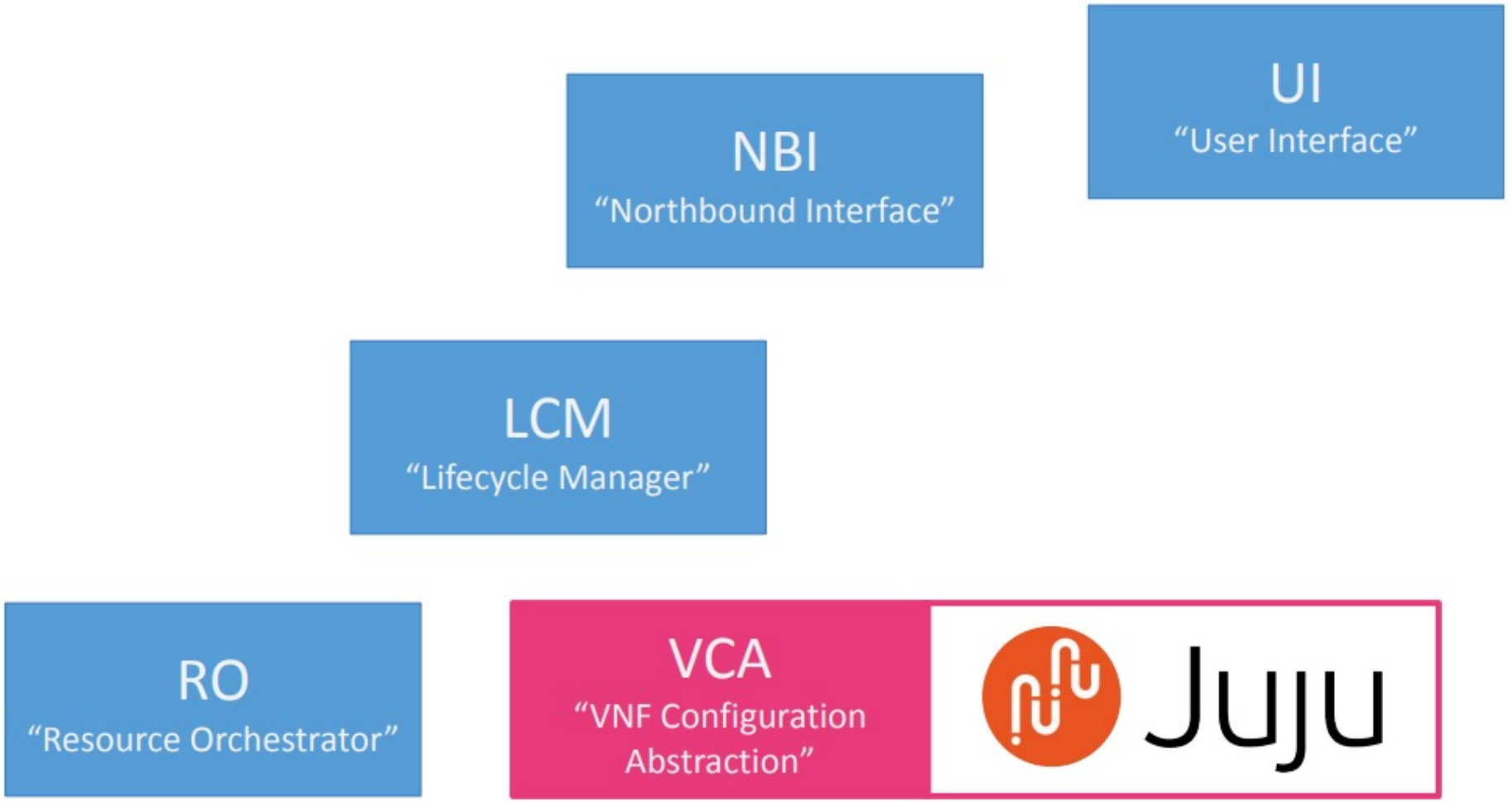
How to use Charms to perform VNF operations?

We will use proxy Charms to perform operations on the VNFs



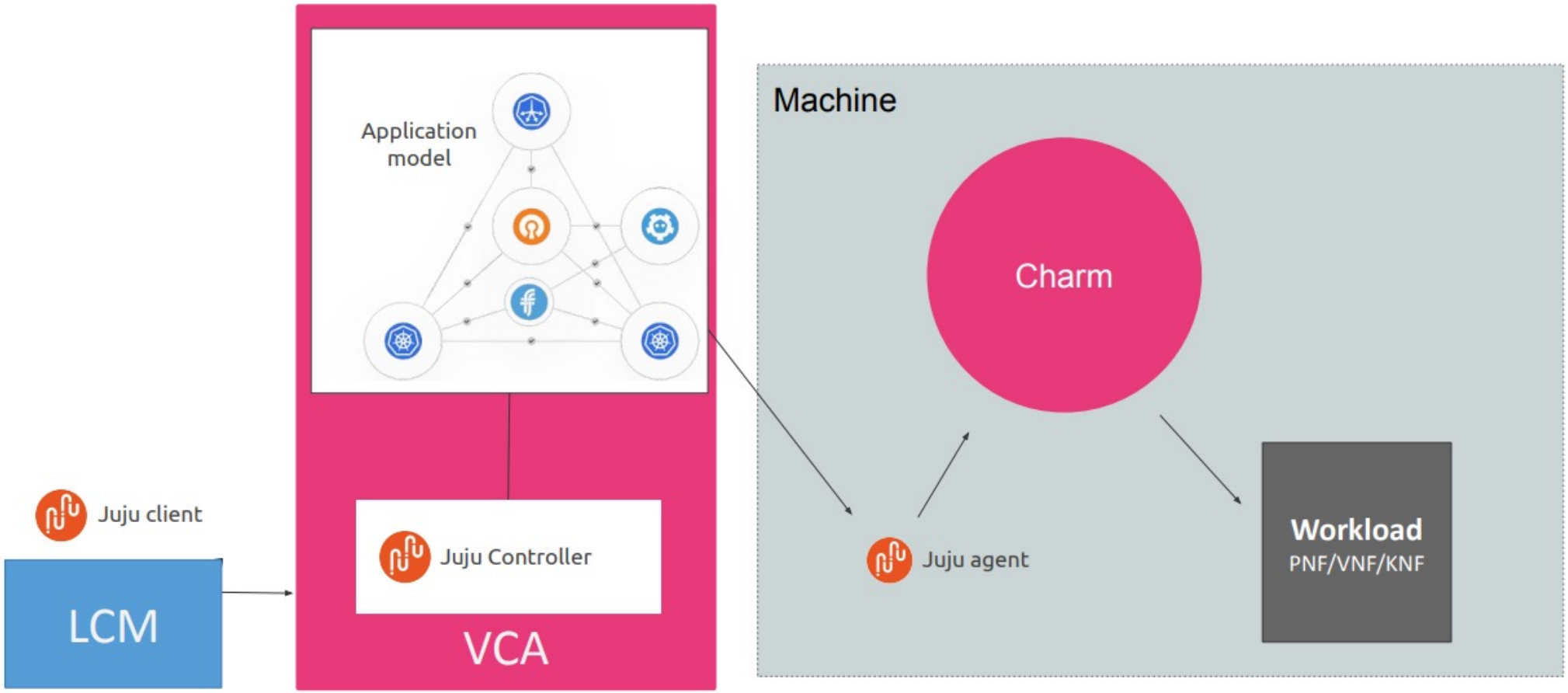
Adapted from OSM MR10 Hackfest

OSM Architecture and Charms Integration



Source: OSM MR10 Hackfest

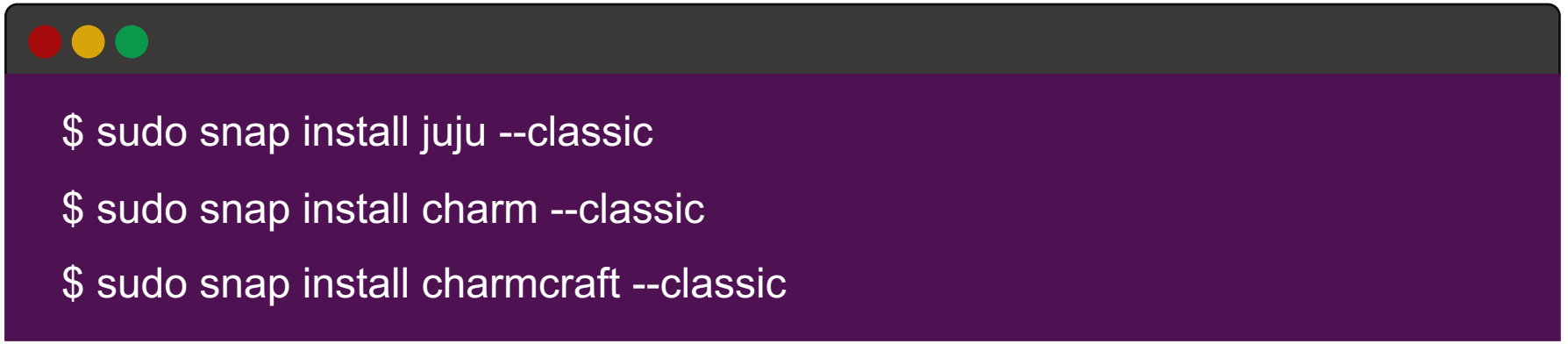
OSM Architecture and Charms Integration



Source: OSM MR10 Hackfest

How to create a Juju Charm ?

First of all, install all the needed software…



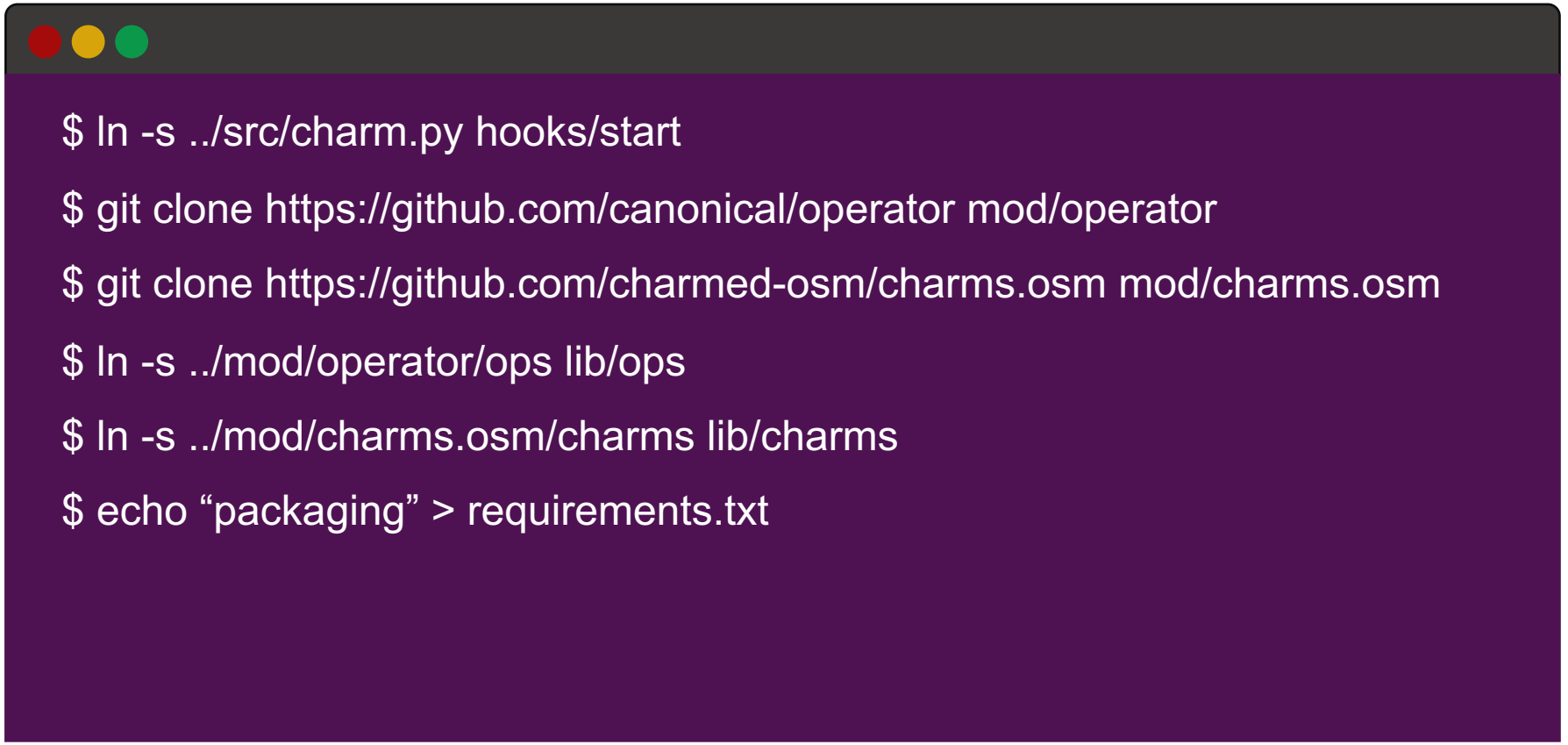
Now, let’s start the development of the Charm…

Let’s create the base structure of our charm and add all the libraries required by OSM

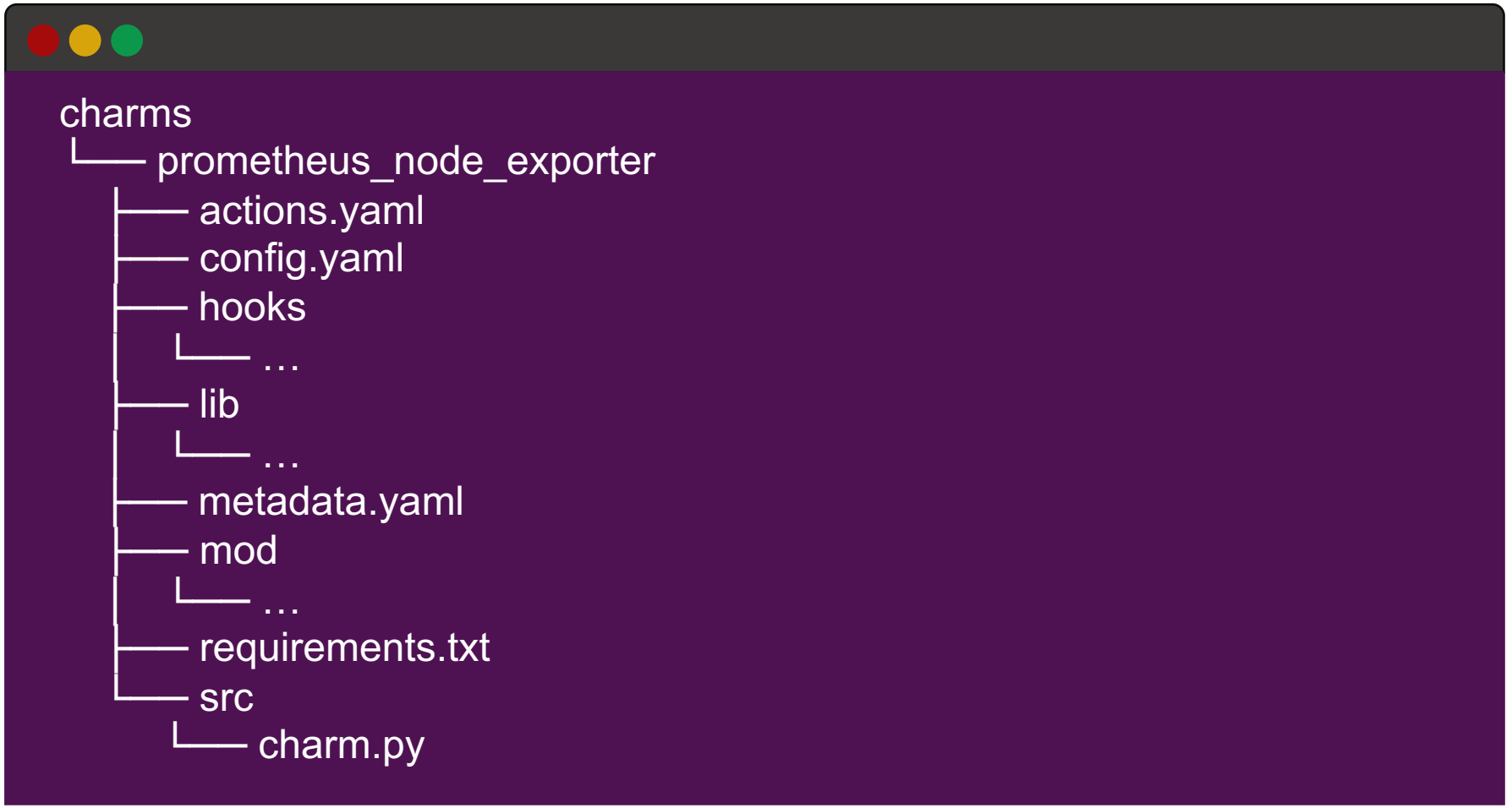
```
$ mkdir -p charms/prometheus_node_exporter/
$ cd charms/prometheus_node_exporter
$ mkdir hooks lib mod src
$ touch src/charm.py
$ touch actions.yaml metadata.yaml config.yaml
$ chmod +x src/charm.py
$ ln -s ../src/charm.py hooks/upgrade-charm
$ ln -s ../src/charm.py hooks/install
```

Now, let’s start the development of the Charm…

Let’s create the base structure of our charm and add all the libraries required by OSM



After running these commands, you should have the following file structure



File Content
metadata.yaml

```yaml
name: prometheus-node-exporter
summary: This charm enables the deployment of a prometheus exporter
maintainer: Rafael Direito <rdireito@av.it.pt>
description: |
  This charm is part of a juju charms development tutorial.
tags:
  - nfv
subordinate: false
series:
  - bionic
  - xenial
  - focal
peers: # This will give HA capabilities to your Proxy Charm
  proxypeer:
    interface: proxypeer
```

File Content
config.yaml

```yaml
options:
  ssh-hostname:
    type: string
    default: ""
    description: "The hostname or IP address of the machine to"
  ssh-username:
    type: string
    default: ""
    description: "The username to login as."
  ssh-password:
    type: string
    default: ""
    description: "The password used to authenticate."
  ssh-public-key:
    type: string
    default: ""
    description: "The public key of this unit."
  ssh-key-type:
    type: string
    default: "rsa"
    description: "The type of encryption to use for the SSH key."
  ssh-key-bits:
    type: int
    default: 4096
    description: "The number of bits to use for the SSH key."
```

File Content
actions.yaml

```yaml
# Actions to be implemented in src/charm.py
configure-remote:
  description: "Configures the remote server"
  params:
    destination_ip:
      description: "IP of the remote server"
      type: string
      default: ""
  required:
    - destination_ip
start-service:
  description: "Starts the service of the VNF"

# Required by charms.osm.sshproxy
run:
  description: "Run an arbitrary command"
  params:
    command:
      description: "The command to execute."
      type: string
      default: ""
  required:
    - command
generate-ssh-key:
  description: "Generate a new SSH keypair for this unit. This will replace any existing previously generated keypair."
verify-ssh-credentials:
  description: "Verify that this unit can authenticate with server specified by ssh-hostname and ssh-username."
get-ssh-public-key:
  description: "Get the public SSH key for this unit."
```

File Content
src/charm.py

```
#!/usr/bin/env python3
import sys

sys.path.append("lib")

from charms.osm.sshproxy import SSHProxyCharm
from ops.main import main


class SampleProxyCharm(SSHProxyCharm):
    def __init__(self, framework, key):
        super().__init__(framework, key)

        # Listen to charm events
        self.framework.observe(self.on.config_changed, self.on_config_changed)
        self.framework.observe(self.on.install, self.on_install)
        self.framework.observe(self.on.start, self.on_start)
        # self.framework.observe(self.on.upgrade_charm, self.on_upgrade_charm)

        # Listen to the touch action event
        self.framework.observe(self.on.configure_remote_action, self.configure_remote)
        self.framework.observe(self.on.start_service_action, self.start_service)

    def on_config_changed(self, event):
        """Handle changes in configuration"""
        super().on_config_changed(event)
```

File Content
src/charm.py

```
    def on_install(self, event):
        """Called when the charm is being installed"""
        super().on_install(event)

    def on_start(self, event):
        """Called when the charm is being started"""
        super().on_start(event)

    def configure_remote(self, event):
        """Configure remote action."""

        if self.model.unit.is_leader():
            stderr = None
            try:
                mgmt_ip = self.model.config["ssh-hostname"]
                destination_ip = event.params["destination_ip"]
                cmd = "vnfcli set license {} server {}".format(
                    mgmt_ip,
                    destination_ip
                )
                proxy = self.get_ssh_proxy()
                stdout, stderr = proxy.run(cmd)
                event.set_results({"output": stdout})
            except Exception as e:
                event.fail("Action failed {}. Stderr: {}".format(e, stderr))
        else:
            event.fail("Unit is not leader")
```

File Content
src/charm.py

```
    def start_service(self, event):
        """Start service action."""

        if self.model.unit.is_leader():
            stderr = None
            try:
                cmd = "sudo service vnfoper start"
                proxy = self.get_ssh_proxy()
                stdout, stderr = proxy.run(cmd)
                event.set_results({"output": stdout})
            except Exception as e:
                event.fail("Action failed {}. Stderr: {}".format(e, stderr))
        else:
            event.fail("Unit is not leader")


if __name__ == "__main__":
    main(SampleProxyCharm)
```

How to test our base-Charm?

From the *config.yaml file*, we can see that there are variables required to use this charm. We will create a *local-config.yaml* file and define all these variables.

```
# this parameters should be adapted to
# the charateristics of your destination VM
prometheus-node-exporter:
        ssh-hostname: 10.0.12.95
        ssh-username: ubuntu
        ssh-password: password
```

Since we are using a proxy Charm, the values of each variable must address the VM/VNF where we will perform operations.

How to test our base-Charm?

Now, we will test if we can perform the desired operations using this Juju Charm

```
# let's start by building the charm
$ charmcraft build
# in case of build error: ($ lxd init –auto) and ($ lxc network set lxdbr0 ipv6.address none)
# before deploying, you might need to choose your controller (select localhost):
$ juju bootstrap
# after this, we will deploy the Charm using the configurations defined in local-config.yaml
$ juju deploy ./prometheus-node-exporter_ubuntu-20.04-amd64.charm  --config local-config.yaml
# we can now check the status of our Charm's deployment
$ juju status
```

How to test our base-Charm?

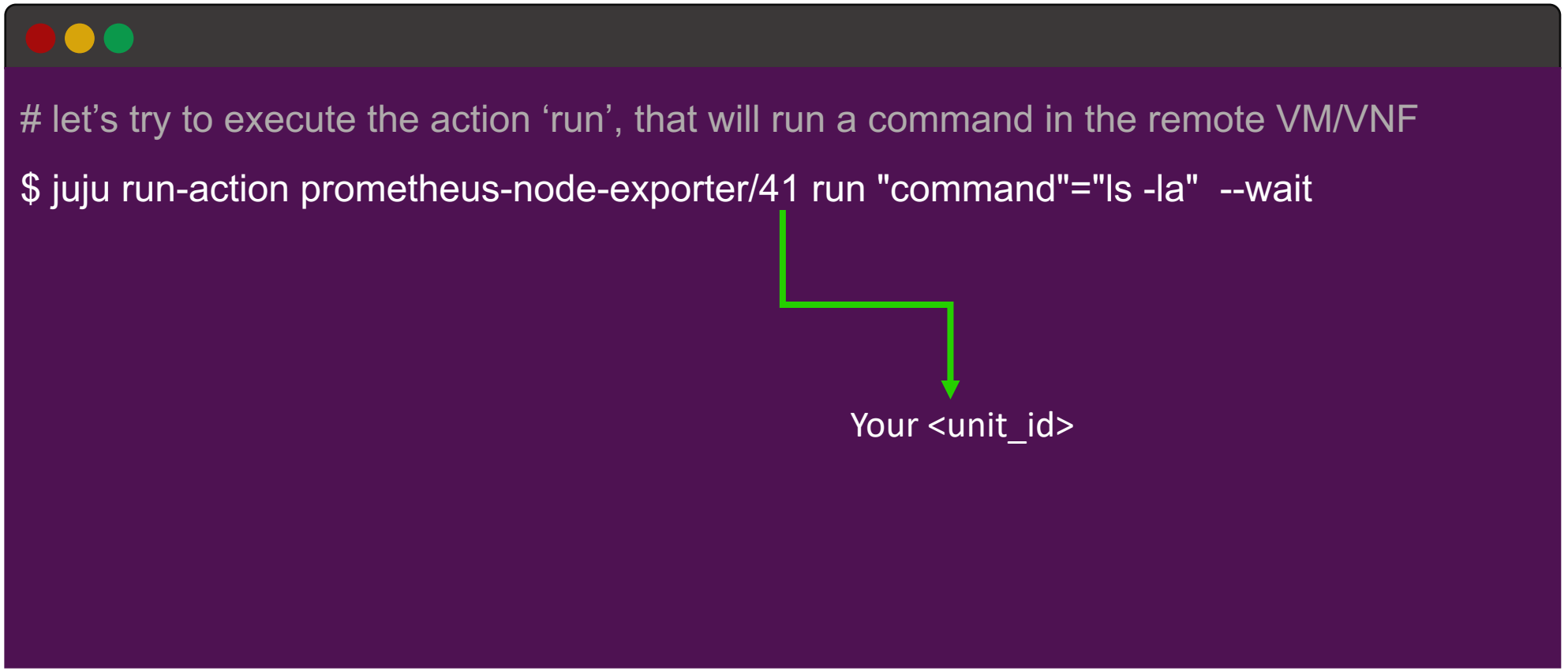
After a few seconds, who should see the following output:

```
ubuntu@juju-server:~/charms/prometheus_node_exporter$ juju status
Model  Controller  Cloud/Region         Version  SLA          Timestamp
dev    lxd         localhost/localhost  2.9.22   unsupported  10:41:55Z

App                       Version  Status  Scale  Charm                     Store  Channel  Rev  OS      Message
prometheus-node-exporter           active      1  prometheus-node-exporter  local            2  ubuntu

Unit                         Workload  Agent  Machine  Public address  Ports  Message
prometheus-node-exporter/41*  active    idle   63       10.71.122.164

Machine  State    DNS            Inst id         Series  AZ  Message
63       started  10.71.122.164  juju-285551-63  bionic      Running
```

Your **<unit_id>** that you will need later to execute actions

How to test our base-Charm?

Now, we are ready to execute the action “run”



How to test our base-Charm?

If all goes accordingly, you should have a similar output:

```
ubuntu@juju-server:~/charms/prometheus_node_exporter$ juju run-action prometheus-node-exporter/41 run "command"="ls -la"  --wait
unit-prometheus-node-exporter-41:
  UnitId: prometheus-node-exporter/41
  id: "197"
  results:
    output: |-
      total 120
      drwxr-xr-x 13 ubuntu ubuntu  4096 Jan 13 10:39 .
      drwxr-xr-x  6 root   root    4096 Jun  7  2021 ..
      -rw-------  1 ubuntu ubuntu 19597 Jan 12 16:14 .bash_history
      -rw-r--r--  1 ubuntu ubuntu     0 Feb 25  2020 .bash_logout
      -rw-r--r--  1 ubuntu ubuntu    88 Jan  6 11:23 .bashrc
      drwx------  5 ubuntu ubuntu  4096 Jan 10 10:34 .cache
      drwxrwxr-x  2 ubuntu lxd     4096 Jan  6 11:27 .config
      -rw-rw-r--  1 ubuntu ubuntu    34 Jan 10 15:03 .gitconfig
      drwx------  3 ubuntu lxd     4096 Jan  6 11:19 .local
```

Adding the logic we want to our charm

The initial goal was to create a Juju Charm that makes available a Prometheus Node Metrics Exporter. To do so, we will have to:

- Add the Prometheus Node Metrics Exporter's logic to *src/charms.py*
- Define two new actions in *actions.yaml:*
  - *start-prometheus-exporter*
  - *stop-prometheus-exporter*

Add the following to *actions.yaml*

```
# Custom actions
start-prometheus-exporter:
  description: "Start the Prometheus Node Exporter"
stop-prometheus-exporter:
  description: "Stop the Prometheus Node Exporter"
```

Add the following to *src/charm.py*

Add the needed imports to handle operations

```
from ops.model import (
    ActiveStatus,
    MaintenanceStatus,
    BlockedStatus,
    WaitingStatus,
    ModelError,
)
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
class SampleProxyCharm(SSHProxyCharm):
    def __init__(self, framework, key):
        super().__init__(framework, key)
        # ...
        # Custom actions
        self.framework.observe(self.on.start_prometheus_exporter_action, self.on_start_prometheus_exporter)
        self.framework.observe(self.on.stop_prometheus_exporter_action, self.on_stop_prometheus_exporter)
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
    ############################
    #     Custom Actions       #
    ############################
    def on_start_prometheus_exporter(self, event):
        self._get_prometheus_exporter(event)
        self._create_prometheus_exporter_service(event)
        self._run_prometheus_exporter(event)


    def on_stop_prometheus_exporter(self, event):
        self._stop_prometheus_exporter(event)
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
########################
#        Functions     #
########################

def _get_prometheus_exporter(self, event):
    commands = [
        {
            "command": "wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz",
            "initial_status": "Getting Prometheus Exporter Bin...",
            "ok_status": "Obtained Prometheus Exporter Tar",
            "error_status": "Couldn't Obtain Prometheus Exporter Tar"
        },
        {
            "command": "tar xvfz node_exporter-1.3.1.linux-amd64.tar.gz",
            "initial_status": "Decompressing Prometheus Exporter Tar...",
            "ok_status": "decompressed Prometheus Exporter Tar",
            "error_status": "Couldn't Decompress Prometheus Exporter Tar"
        },
        {
            "command": "sudo mv node_exporter-1.3.1.linux-amd64/node_exporter /usr/local/bin/ && rm -rf node_exporter-1.3.1.linux-amd64",
            "initial_status": "Moving Prometheus Exporter Bin to the Correct Location...",
            "ok_status": "Moved Prometheus Exporter Bin to /usr/local/bin/",
            "error_status": "Couldn't Move the Prometheus Exporter Bin to /usr/local/bin/"
        },
    ]

    for i in range(len(commands)):
        self.unit.status = MaintenanceStatus(commands[i]["initial_status"])
        try:
            proxy = self.get_ssh_proxy()
            result, error = proxy.run(commands[i]["command"])
            event.set_results({"output": result, "errors": error})
            event.log(commands[i]["ok_status"])
            self.unit.status = MaintenanceStatus(commands[i]["ok_status"])
        except Exception as e:
            event.fail("[Unable to Get the Prometheus Exporter Binary] Action failed {}. Stderr: {}".format(e, error))
            self.unit.status = BlockedStatus(commands[i]["error_status"])
            return False

    self.unit.status = ActiveStatus("Prometheus Exporter Binary Obtained With Success")
    return True
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
    def _create_prometheus_exporter_service(self, event):
        commands = [
            {
                "command": 'sudo useradd -rs /bin/false node_exporter || true',
                "initial_status": "Adding node_exporter User...",
                "ok_status": "Added node_exporter User...",
                "error_status": "Couldn't Add  node_exporter User...",
            },
            {
                "command": '\"echo -e \\"{}\\" | sudo tee {}\"'.format(
                "[Unit]\nDescription=Node Exporter\nAfter=network.target\n\n[Service]\nUser=node_exporter\nGroup=node_exporter\nType=simple\nExecStart=/usr/local/bin/node_exporter\n\n[Install]\nWantedBy=multi-user.target",
                "/etc/systemd/system/node_exporter.service"),
                "initial_status": "Adding Prometheus Exporter Service to Systemd...",
                "ok_status": "Added Prometheus Exporter Service to Systemd",
                "error_status": "Couldn't Add Prometheus Exporter Service to Systemd",
            },
            {
                "command": 'sudo systemctl daemon-reload && sudo systemctl enable node_exporter',
                "initial_status": "Enabling Prometheus Exporter Service...",
                "ok_status": "Enabled Prometheus Exporter Service...",
                "error_status": "Couldn't Enable Prometheus Exporter Service...",
            },

        ]

        for i in range(len(commands)):
            self.unit.status = MaintenanceStatus(commands[i]["initial_status"])
            try:
                proxy = self.get_ssh_proxy()
                result, error = proxy.run(commands[i]["command"])
                event.set_results({"output": result, "errors": error})
                event.log(commands[i]["ok_status"])
                self.unit.status = MaintenanceStatus(commands[i]["ok_status"])
            except Exception as e:
                event.fail("[Unable to Create Prometheus Exporter Service With Success] Action failed {}. Stderr: {}".format(e, error))
                self.unit.status = BlockedStatus(commands[i]["error_status"])
                return False

        self.unit.status = ActiveStatus("Created Prometheus Exporter Service With Success")
        return True
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
def _run_prometheus_exporter(self, event):
    self.unit.status = MaintenanceStatus("Starting Prometheus Exporter Service...")
    try:
        proxy = self.get_ssh_proxy()
        result, error = proxy.run("sudo service node_exporter start")
        event.set_results({"output": result, "errors": error})
        event.log("Started Prometheus Exporter Service...")
        self.unit.status = MaintenanceStatus("Started Prometheus Exporter Service...")
    except Exception as e:
        event.fail("[Couldn't Start Prometheus Exporter Service] Action failed {}. Stderr: {}".format(e, error))
        self.unit.status = BlockedStatus("Couldn't Start Prometheus Exporter Service...")
        return False

    self.unit.status = MaintenanceStatus("Checking if Prometheus Exporter Service is Running")
    try:
        proxy = self.get_ssh_proxy()
        result, error = proxy.run("curl -Is http://127.0.0.1:9100/metrics | head -1")
        event.set_results({"output": result, "errors": error})
        if "200" not in result:
            event.fail("Prometheus Exporter Service is not Running...")
            self.unit.status = BlockedStatus("Prometheus Exporter Service is not Running")
            return False
    except Exception as e:
        event.fail("[Prometheus Exporter Service is not Running] Action failed {}. Stderr: {}".format(e, error))
        self.unit.status = BlockedStatus("Prometheus Exporter Service is not Running")
        return False

    event.log("Prometheus Exporter Service is Running")
    self.unit.status = ActiveStatus("Prometheus Exporter Service is Running")
    return True
```

Add the following to *src/charm.py* inside the *SampleProxyCharm* class

```
    def _stop_prometheus_exporter(self, event):
        self.unit.status = MaintenanceStatus("Stopping Prometheus Exporter Service...")
        try:
            proxy = self.get_ssh_proxy()
            result, error = proxy.run("sudo service node_exporter stop")
            event.set_results({"output": result, "errors": error})
            event.log("Stopped Prometheus Exporter Service")
            self.unit.status = MaintenanceStatus("Stopped Prometheus Exporter Service")
        except Exception as e:
            event.fail("[Couldn't Stop Prometheus Exporter Service] Action failed {}. Stderr: {}".format(e, error))
            self.unit.status = BlockedStatus("Couldn't Stop Prometheus Exporter Service")
            return False

        self.unit.status = ActiveStatus("Prometheus Exporter Service was Stopped")
        return True
```

How to test the Charm?

Now, we will test if we can perform the desired operations using this Juju Charm

```
$ # let's start by building the charm
$ charmcraft build
$ # after this, we will deploy the Charm using the configurations defined in
local-config.yaml
$ juju deploy ./prometheus-node-exporter_ubuntu-20.04-amd64.charm  --config
local-config.yaml
$ # we can now check the status of our Charm's deployment
$ juju status
```

How to test our base-Charm?

Now, we are ready to execute the action “start-prometheus-exporter”

```
$ # let’s try to execute the action ‘start-prometehus-exporter’, that will run a command in the remote VM/VNF
$ juju run-action prometheus-node-exporter/42 start-prometheus-exporter --wait
```

How to test our base-Charm?

Desired output:

```
ubuntu@juju-server:~/charms/prometheus_node_exporter$ juju run-action prometheus-node-exporter/42 start-prometheus-exporter --wait
unit-prometheus-node-exporter-42:
  UnitId: prometheus-node-exporter/42
  id: "200"
  log:
  - 2022-01-13 11:27:09 +0000 WET Obtained Prometheus Exporter Tar
  - 2022-01-13 11:27:10 +0000 WET decompressed Prometheus Exporter Tar
  - 2022-01-13 11:27:11 +0000 WET Moved Prometheus Exporter Bin to /usr/local/bin/
  - 2022-01-13 11:27:13 +0000 WET Added node_exporter User...
  - 2022-01-13 11:27:14 +0000 WET Added Prometheus Exporter Service to Systemd
  - 2022-01-13 11:27:16 +0000 WET Enabled Prometheus Exporter Service...
  - 2022-01-13 11:27:19 +0000 WET Started Prometheus Exporter Service...
  - 2022-01-13 11:27:21 +0000 WET Prometheus Exporter Service is Running
  results:
    errors: ""
    output: HTTP/1.1 200 OK
  status: completed
  timing:
    completed: 2022-01-13 11:27:21 +0000 UTC
    enqueued: 2022-01-13 11:27:05 +0000 UTC
    started: 2022-01-13 11:27:06 +0000 UTC
```

How to test our base-Charm?

To test if the Prometheus metrics exporter is working, we will curl the metrics endpoint and verify if we can access the desired metrics.

```
ubuntu@juju-server:~/charms/prometheus_node_exporter$ curl http://10.0.12.95:9100/metrics | head -n 15
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0.000245571
go_gc_duration_seconds{quantile="0.25"} 0.000422831
go_gc_duration_seconds{quantile="0.5"} 0.000477091
go_gc_duration_seconds{quantile="0.75"} 0.000515006
go_gc_duration_seconds{quantile="1"} 0.004005789
go_gc_duration_seconds_sum 0.657628109
go_gc_duration_seconds_count 1377
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.17.3"} 1
```

Success!

Useful commands

```
$ # delete an application
$ juju remove-application <application_id>

$ # delete a machine
$ juju remove-machine <machine_id>

$ # if a deployment fails, we can view its logs
$ juju debug-log

$ # it is possible to continuously observe the status of the deployment, using the –watch flag
$ juju status –watch 5s

$ # it is also possible to access the charm’s machine terminal
$ juju ssh <machine_id>
```

The code developed during this tutorial is available at

https://github.com/5gasp/tutorials

If you have any questions regarding the contents addressed in this tutorial you can send an e-mail to Rafael Direito rdireito@av.it.pt, or contact us via contact@5gasp.eu

You can access more information at community.5gasp.eu