Juju Charms tutorial

Software installation

Installation of juju and charmcraft

```
$ sudo snap install juju --classic
$ sudo snap install charm --classic
$ sudo snap install charmcraft --classic
```

Creation of the Charm structure

```
$ 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

$ ln -s ../src/charm.py hooks/start
$ git clone https://github.com/canonical/operator mod/operator
$ git clone https://github.com/charmed-osm/charms.osm mod/charms.osm
$ ln -s ../mod/operator/ops lib/ops
$ ln -s ../mod/charms.osm/charms lib/charms
$ echo “packaging” > requirements.txt
```

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

```
charms
└── prometheus_node_exporter
    ├── actions.yaml
    ├── config.yaml
    ├── hooks
    │   └── …
    ├── lib
    │   └── …
    ├── metadata.yaml
    ├── mod
    │   └── …
    ├── requirements.txt
    └── src
        └── charm.py
```

Changing files

Changing files to create a basic charm

metadata.yaml

You should replace the maintainer

```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
```

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."
```

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."
```

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)

    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")

    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 loader")

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

local-config.yaml

Keep in mind that you should replace the hostname with your machine, as well as the username and password that you have.

```
# this parameters should be adapted to
# the characteristics of your destination VM
prometheus-node-exporter:
        ssh-hostname: 10.0.12.224
        ssh-username: ubuntu
        ssh-password: password
```

Notes

Before building the charm you might need to execute the following commands:

```
$ lxd init --auto
$ lxc network set lxdbr0 ipv6.address none
```

Build the charm:

```
$ charmcraft build
```

Before deploying you might need to choose your controller (In your case should be localhost):

```
$ juju bootstrap
```

Deploying the charm using the configurations in the local-config file:

```
$ juju deploy ./prometheus-node-exporter_ubuntu-20.04-amd64.charm  --config local-config.yaml
```

Check status:

```
$ juju status
```

Execute the **'run'** action to run a command inside the VM/VNF. You can check your *unit_id* with juju status.

```
$ juju run-action prometheus-node-exporter/<unit_id> run "command"="ls -la" --wait
```

Adding the Prometheus Node Metrics Exporter's logic

Update actions.yaml

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

Update src/charm.py

Add needed imports

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

Inside the SampleProxyCharm class add the custom actions

```
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)
```

Define the 2 functions inside the 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 functions to handle the prometheus

```
#########################
#        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


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/lo
cal/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


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


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
```

Check if charm is working

Build and deploy the charm again

First, remove the first charm that we deployed:

```
$ juju remove-application prometheus-node-exporter
```

Using the commands in the first charm, build and deploy.

Run the action for the prometheus that we created

```
$ juju run-action prometheus-node-exporter/<unit_id> start-prometheus-exporter --wait
```

Useful juju commands

Delete an application

```
$ juju remove-application <application_id>
```

Delete a machine

```
$ juju remove-machine <machine_id>
```

Check deployment logs

```
$ juju debug-log
```

Continuously observe deployment status

```
$ juju status –watch 5s
```

Access machine terminal

```
$ juju ssh <machine_id>
```

Video

For a better description of the tutorial, you can check our video here