ubuntu

Generic VNF Configuration Management and Orchestration

Why telco's are finally taking the cloud seriously

CANONICAL

Why things are changing

The telecommunications industry is in a state of disruption.

The old way of doing business doesn't work as well in a cloud-based ecosystem.

Network appliances are going the way of the non-avian dinosaur.

It's time to evolve.

Welcome to Network Function Virtualization (NFV)

NFV management and organization (MANO)

An ETSI working group

NFV Orchestrator	VNF Manager	Virtualized Infrastructure Manager
Manage Network Services, VNF packages, and global resources.	Lifecycle and configuration management of VNF instances.	Manage compute, network, and storage resources. This is the cloud layer.

NFV Mano Implementations

Generic VNF Managers (VNFM)

Tacker

Virtualization Infrastructure Managers

Virtualization and hypervisors

OpenVIM

Carriers using Juju for generic VNF manager

Diving in to Network Function Virtualization (NFV)

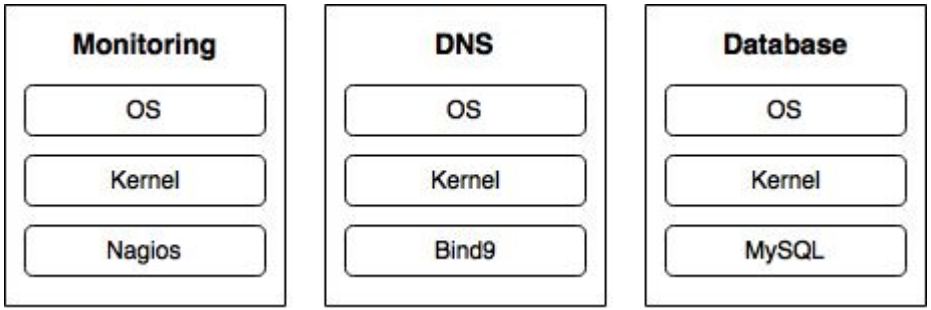
Network Appliances dissected

Load Balancer

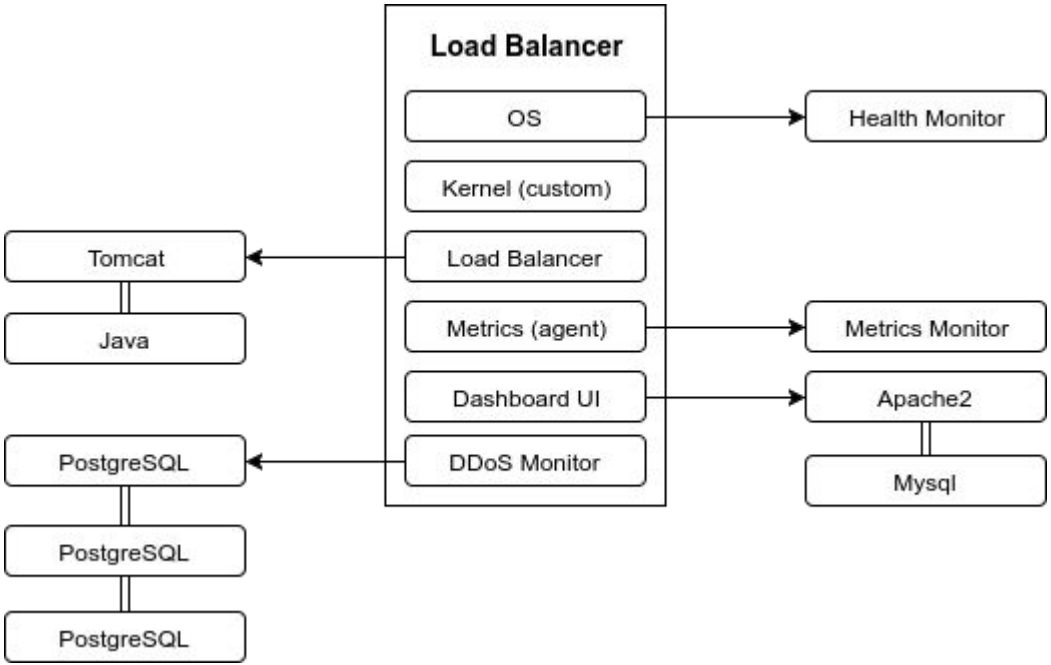
- Big, beefy hardware
- Expensive
- It’s a black box
- Only scales horizontally
- Vendor lock-in
- Monolithic

Network Function Virtualization explained

Network Function Virtualization (NFV) virtualizes specific classes of network functions into reusable components.



Network Function Virtualization executed

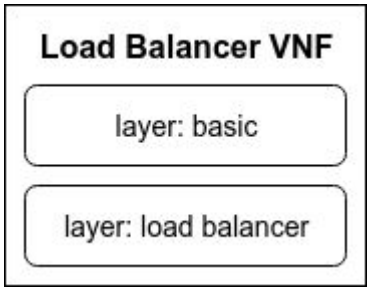


How VNFs are created

Breaking it down

VNF Packages and Descriptors

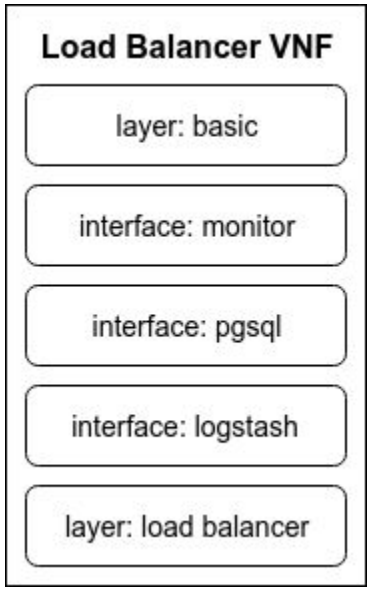
- A series of scripts that react to a lifecycle event, such as install, start, and stop
- Compatible with existing config management, such as puppet, chef, ansible, docker, etc.
- Built in “layers” so common functionality can be shared and reused.



Breaking it down

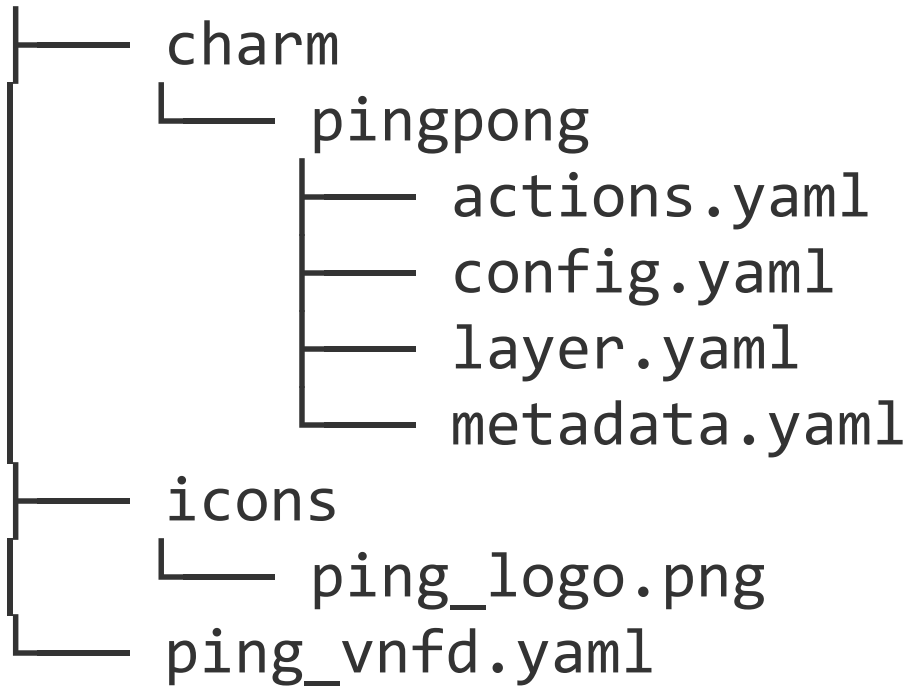
Integrations

- Integrate with things you need to work with
- Encapsulate protocol, reducing the pain of interoperability
- Take advantage of a vendor’s operational knowledge of their application



Breaking it down

The VNF package



Breaking it down

Event Driven

Lifecycle:

- config-changed
- install
- leader-elected
- leader-settings-changed
- start
- stop
- upgrade-charm
- update-status

Relation:

- [name]-relation-joined
- [name]-relation-changed
- [name]-relation-departed
- [name]-relation-broken

Storage:

- [name]-storage-attached
- [name]-storage-detached

Metrics:

- collect-metrics

Breaking it down

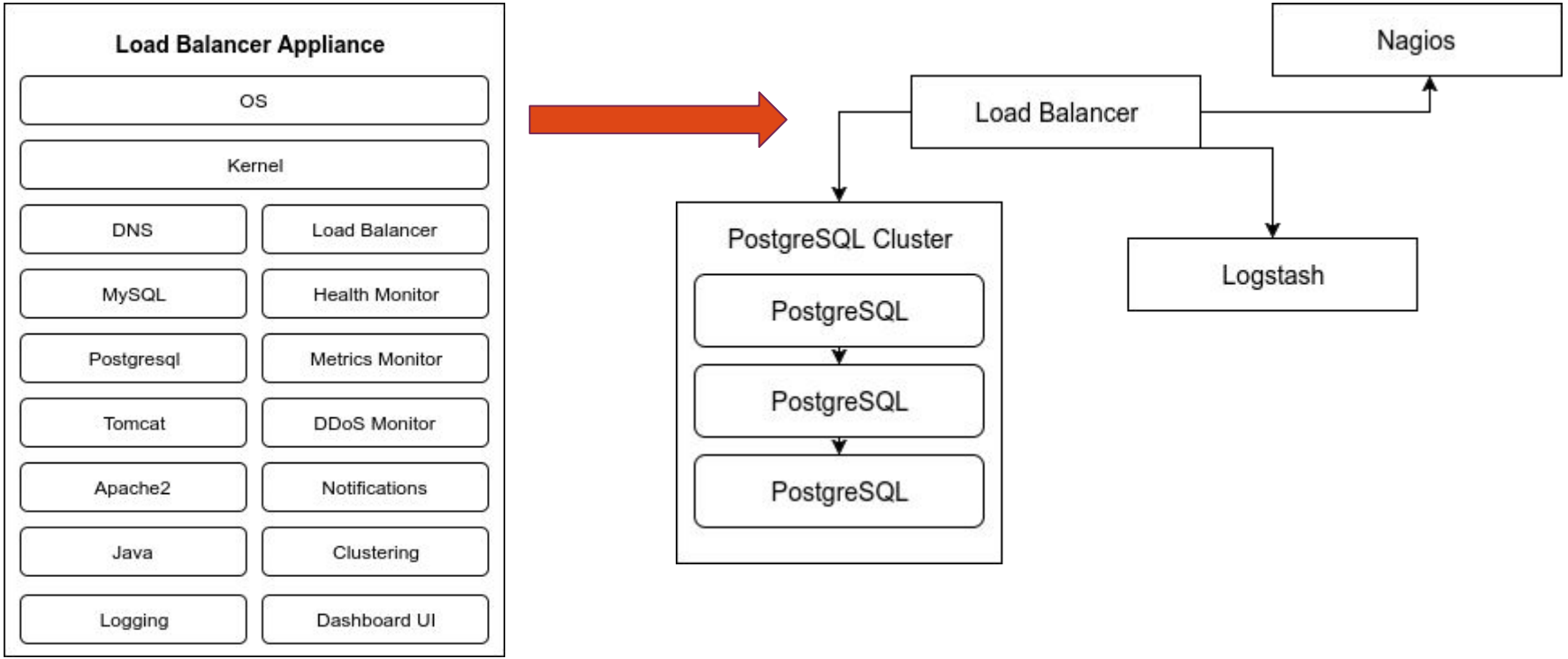
Implementation - Python example

```
@when_not('load-balancer.installed')
def install_load_balancer():
    # Validate license & install
    license = config[‘license-key’]
    if valid_license():
        download_packages()
        verify_packages()
        install_packages()
        set_flag('load-balancer.installed')
```

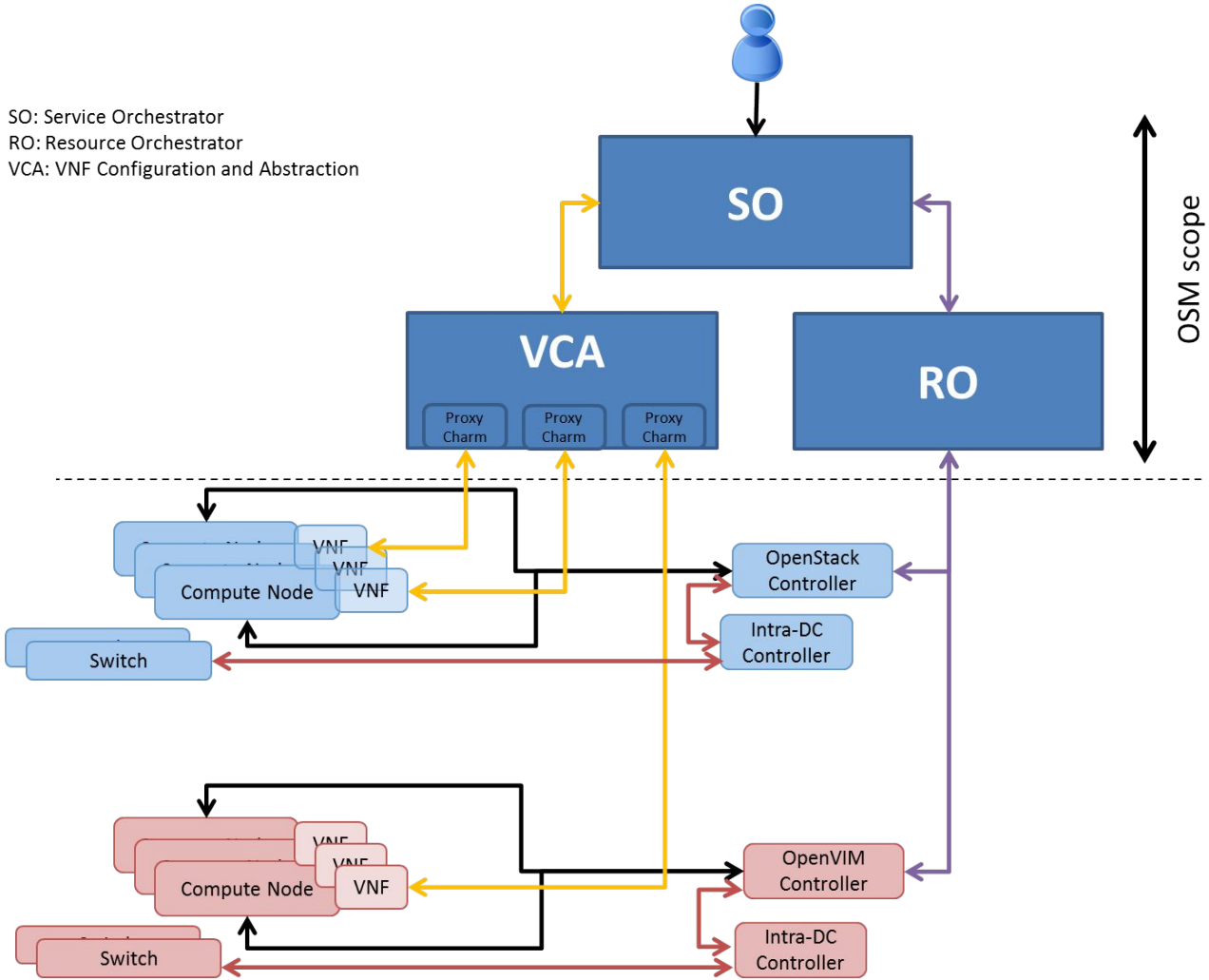
Orchestration

Putting it together

Transform a monolithic appliance into a scalable application.



SO: Service Orchestrator
RO: Resource Orchestrator
VCA: VNF Configuration and Abstraction



Putting it together

Advanced Features

libjuju

- asynchronous Python library,
- drives Juju via it’s websocket API.
- (example) how to dynamically auto-scale up or down based on the performance metrics of your application, and auto-healing.

Libjuju talk today!,

- Tim Van Steenburgh: Operator Track: Driving Juju with Python Monday, February 6, 17:00 in room B4.039

https://github.com/juju/python-libjuju

Putting it together

Deploying via CLI

```
juju deploy load-balancer
juju deploy postgresql -n 3 --constraints mem=8G
juju deploy nagios
juju deploy logstash

juju add-relation load-balancer postgresql
juju add-relation load-balancer nagios
juju add-relation load-balancer logstash
juju add-relation postgresql nagios
juju add-relation postgresql logstash
```

Putting it together

Scaling

```
Horizontally:
$ juju add-units load-balancer -n 2

Vertically:
$ juju deploy postgresql -n 3 --constraints mem=16G

Remove 8GB Nodes:
$ juju remove-unit postgresql/0 postgresql/1 postgresql/2
```

Managing complex configurations

Application-level configuration

VNF Configuration

```
juju config set load-balancer max_connections=2048

or

juju config set load-balancer --file=lbconfig.yaml
```

Summary

Open Source carrier-grade tools you can use.

- Scale horizontally and/or vertically
- Auto-scaling
- Auto-healing
- Drive by CLI, GUI or API
- Cross-platform - Ubuntu, Centos, Windows, OS X
- Cross-architecture: i386, AMD64, ARM64, Power, S390X
- Used by many MANO implementations

Open Source carrier-grade tools you can use.

Get started:

```
juju deploy cs:~aisrael/netutils/
```

Or try out my work-in-progress Asterisk VNF:

```
juju deploy cs:~aisrael/asterisk-0
```

```
https://github.com/AdamIsrael/layer-netutils
```

http://github.com/AdamIsrael/layer-asterisk

ubuntu

Thank you!

Adam Israel
adam.israel@canonical.com
http://github.com/AdamIsrael

CANONICAL

Demo?

Afterthoughts

How could I have been a better presenter?

- Speak louder, annunciate more.
- Stick to the script. Every time I went off-script I ended up mumbling or rambling.
-

Breaking it down

actions.yaml

```
deploy-service:
    description: "Deploy a service to be load-balanced."
    params:
        service-template:
            type: string
            description: "The template for the service."
        service-name:
            type: string
            description: "The name of the service."
```

Breaking it down

config.yaml

```
options:
  max_connections:
    default: 4096
    type: int
    description: |
        Sets the maximum per-process number of concurrent
connections to
        <number>.
```

Breaking it down

layer.yaml

```
includes:

    - 'layer:basic'

    - 'layer:sshproxy'

options:

  basic:

    packages: ['python-heatclient', 'python-glanceclient']
```

Breaking it down

metadata.yaml

```
name: load-balancer

summary: A mock load-balancer application, for demonstration
purposes only.

maintainer: Adam Israel <adam.israel@canonical.com>

Description: “This is an example of packaging a VNF

tags:

  - cache-proxy

subordinate: false

series:

  - centos7

  - xenial
```

Breaking it down

Implementation - Bash

```
#!/bin/bash
set -eux
license = $(config-get license-key)
if [ validate_license() ]
then
   download_packages() && validate_packages() && install_packages()
   exit 0
fi
exit 1
```

Putting it together

Scaling

Horizontally:
$ juju add-units load-balancer -n 2

Vertically:
$ juju deploy postgresql -n 3 --constraints mem=16G

Remove 8GB Nodes:
$ juju remove-unit postgresql/0 postgresql/1 postgresql/2

Bringing it together

Bundles

- are YAML-formatted
- defines the integration between applications
- defines machine placement
- defines initial resource requirements

Deploying a bundle of applications is easy:

```
juju deploy mybundle.yaml
```