CNCF Telecom Users Group (TUG)

Cheryl Hung, Director of Ecosystem
Dan Kohn, Executive Director

CLOUD NATIVE
COMPUTING FOUNDATION

CNCF Telecom User Group Meeting Times

- Twice-a-month calls
  - First and third Mondays of the month at 08:00 PT
  - https://github.com/cncf/cnf-testbed#meeting-time
- Mailing list
  - telecom-user-group@lists.cncf.io
- Past kick-off meetings
  - May 23 - KubeCon + CloudNativeCon Barcelona
  - June 25 - KubeCon + CloudNativeCon China

Cloud Native Computing Foundation

- Nonprofit, part of the Linux Foundation; founded Dec 2015

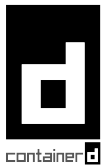
Graduated

Orchestration

Monitoring

Network Proxy

Service Discovery



Container Runtime

Logging

Distributed Tracing

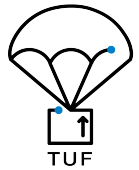
Storage

Incubating

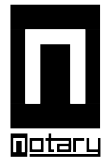
Distributed Tracing API

Remote Procedure Call

Networking API



Software Update Spec



Security

Messaging

Service Mesh

Package Management

Storage

Registry

Key/Value Store

Policy

Container Runtime

Key/Value Store

Serverless

- Platinum members:

KubeCon + CloudNativeCon + Open Networking Summit

- Linux Foundation Member Summit
  - Lake Tahoe: March 10–12, 2020
- Mobile World Congress
  - Barcelona: February 24-27, 2020
- KubeCon + CloudNativeCon Europe
  - Amsterdam: March 30-April 2, 2020

CNCF Telecom User Group (TUG)

- CNCF is launching the Telecom User Group (TUG) for operators and their vendors who are using or aiming to use cloud native technologies in their networks.
- The TUG will operate in a similar capacity to CNCF's End User Community (since operators have never been included in CNCF's definition of end users). Unlike the End User Community, telecom vendors are also encouraged to participate in the TUG.
- The TUG is not expected to do software development, but may write up requirements, best practices, gap analysis, or similar documents.
- We kicked off the TUG in a birds-of-a-feather (BoF) in Barcelona and Shanghai and have moved to twice-a-month Zoom calls.

Efforts To Follow

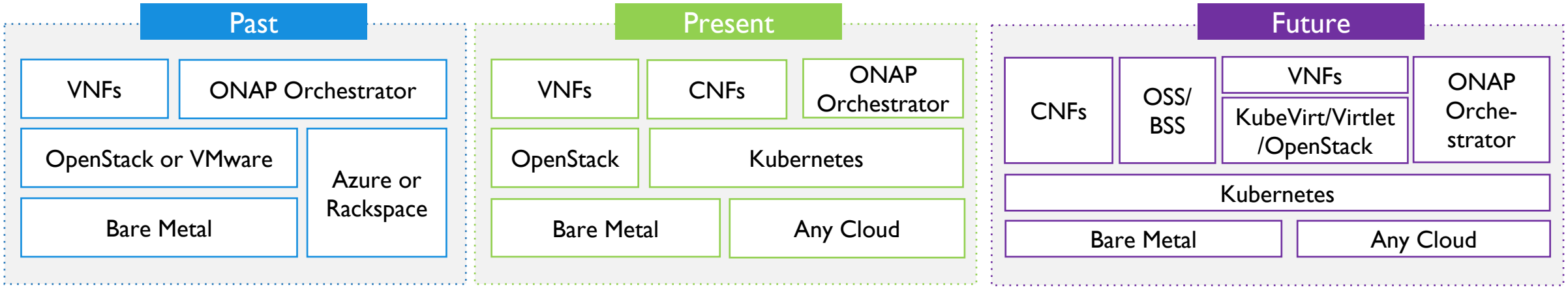
- The TUG is not expected to develop code directly but will want to closely follow upstream projects and may make recommendations for operator use cases. Projects include:
  - Kubernetes Federation v2 (KubeFed)
  - Helm package manager and/or Kustomize plugin
  - Envoy service proxy
  - CNF Testbed
  - Open Policy Agent
  - Observability tools like Prometheus, Fluentd, Jaeger, and OpenTelemetry
  - Service meshes like Linkerd and Istio (the latter is not a CNCF-hosted project)
  - Network Service Mesh (sandbox project)
  - Operators
  - K8s IoT Edge Working Group
  - Using Kubernetes as an Inventory Manager

Initial Possible Work Items

- Gap analysis
- Connecting VNFs and CNFs using Multus, DANM, and/or Network Service Mesh
- Cloud Native Telecoms Best Practices
- CNF Best Practices
- Demonstrating different approaches using the CNF Testbed

Overview

Evolving from VNFs to CNFs



Kubernetes Architecture

"The entire system can now be described as an unbounded number of independent asynchronous control loops reading and writing from/to a schematized resource store as the source of truth. This model has proven to be very resilient, evolvable, and extensible."

- Brian Grant, co-chair, SIG-Architecture

Evolution

- PNFs and VNFs are likely to be with us for at least another decade
- The only feasible approach for cloud native telecom is to offer an evolution of PNFs and VNFs to become CNFs
- This mirrors how enterprises are moving their monoliths to Kubernetes and then (often slowly) refactoring them into microservices
- For this to be economic, there need to be incremental gains in resiliency, bin packing, and development velocity as more network functions become cloud native

CNF Best Practices Ideas

CLOUD NATIVE
COMPUTING FOUNDATION

Bronze CNFs

- Consider a physical firewall device that was ported to a VM to become a VNF, but with no other changes
- When that firewall VNF is ported to become a Cloud native Network Function (CNF), it can no longer carry custom kernel patches or kernel modules and must be compatible with any kernel version 3.10 or higher (the minimum to run Docker)
- This is a "lift-and-shift"
- But it can still include a number of sub-optimal patterns such as:
  - Continued reliance on proprietary management interface
  - Requires stateful storage and writes using a proprietary opaque format
  - No support for horizontal scalability (i.e., multiple instances)
  - No support for ConfigMaps and environment variables
  - Proprietary installer rather than offering a Helm chart

Gold CNFs

- We would like to work with operators and their vendors to define a set of best practices around CNFs, which we could call gold CNFs. These might include:
  - **Compatible**: They should work with any Certified Kubernetes product and any CNI-compatible network that meet their functionality requirements
  - **Stateless**: State should be stored in a Custom Resource Definition or a separate database rather than requiring local storage
  - **Security**: Run unprivileged
  - **Scaling**: It should support horizontal scaling (across multiple machines) and vertical scaling (between sizes of machines)
  - **Configuration and Lifecycle**: via ConfigMaps, Operators, or other declarative interface
  - **Observability**:
    - **Monitoring**: All performance metrics previously available via a proprietary interface should be shared via an OpenMetrics interface that Prometheus and other monitoring tools can use
    - **Tracing**: Support OpenTelemetry-compatible tracing
    - **Logging**: Support Fluentd-compatible logging
  - **Installable and Upgradeable**: Such as via a Helm chart and/or Kustomize plugin
  - **Hardware support**: Via device plugin

CNF Best Practices

- CNCF could offer a self-testing platform to demonstrate conformance with best practices
- Bronze CNFs likely have relatively few, if any, benefits over VNFs
- Gold CNFs will often require a complete re-architecture and so may not be immediately available
- That would mean that the definition of a silver CNF may be critical
- Operators may write into request for proposals (RFPs) a requirement for silver CNFs and/or specific aspects of gold

Cloud Native and Telecom

CLOUD NATIVE
COMPUTING FOUNDATION

Networking

- Kubernetes already offers a myriad of options for networking and with the adoption of Network Service Mesh as a CNCF sandbox project, is poised to support nearly all other use cases
- The CNCF-hosted project Container Network Interface (CNI) supports over a dozen networking technologies, including Multus and DANM, which has seen use in operator applications
- Network Service Mesh (a new sandbox project) flexibly creates layer 2 or 3 network endpoints ("virtual tunnels") similarly to how Envoy/Istio work with TCP and HTTP
  - It requires no changes to upstream Kubernetes and implements cluster functionality as a Custom Resource Definition (CRD)
  - It supports a number of operator use cases such as bridging VPNs, high performance vSwitch, and connecting to PNFs
- Kubernetes supports IPv6 with dual-stack support coming this summer

Security

- Like all systems, Kubernetes needs to be configured appropriately to provide the necessary security
- High-security applications are running on Kubernetes in enterprises around the world, including applications meeting PCI Level 1, HIPAA, and ISO-27001
- Documenting best practices:
  - Containers enable passive patching and a better model for supply chain security
  - Security best practices (and more) including RBAC and namespaces
  - What Kubernetes Does and Doesn't do for Security
  - Disallow containers running as root

Combating FUD Around MicroVMs

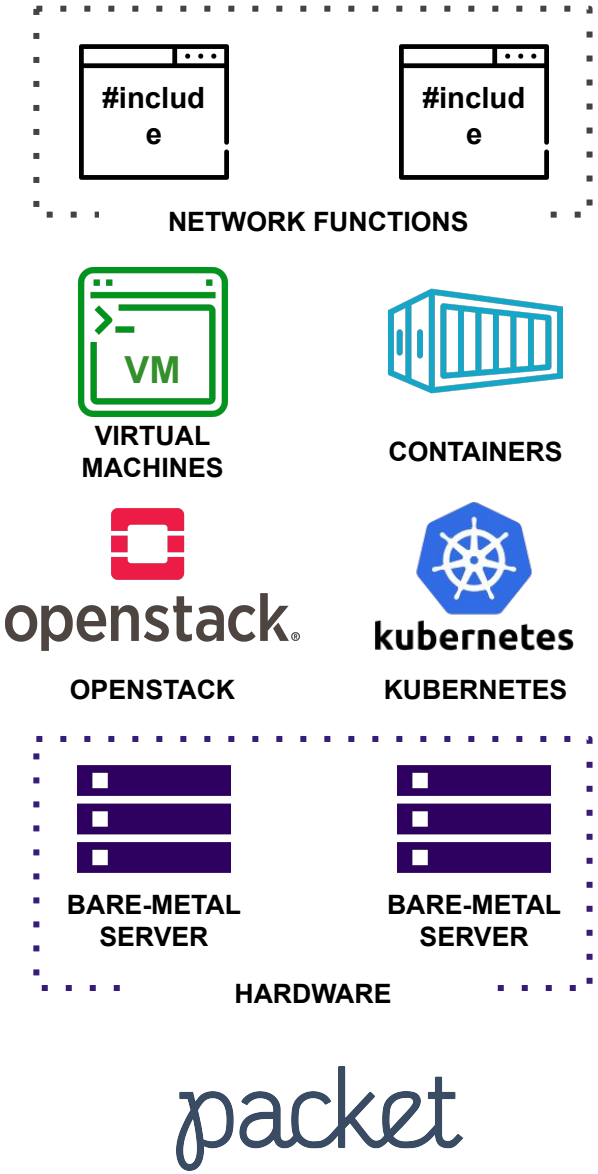
- There has been a lot of Fear, Uncertainty, and Doubt (FUD) about the value of MicroVMs and similar sandbox technology
- Micro virtual machine and sandbox technologies – including Firecracker, gVisor, Kata, Nabla, Singularity, and Unik – are promising options to run **untrusted** code securely on a cluster
- MicroVMs are not necessary to address the **noisy neighbor** issue; that's what the core Kubernetes features **resource limits** (for CPU and memory) and **QoS** (for networking) are for
- More generally, operators are running their own (1st party) code or trusted vendor (2nd party) code, not untrusted 3rd party code

CNF Testbed

CLOUD NATIVE
COMPUTING FOUNDATION

CNF Testbed

- Open source initiative from CNCF
- Collaborating with CNCF Telecom User Group
- Testing and reviewing emerging cloud native technologies in the Telecom domain
- Funneling the new technology to early adopters
- Providing fully reproducible use cases and examples
- Running on top of on-demand hardware from the bare metal hosting company, Packet



CNF Testbed Contributors

Dan Kohn
@dankohn

Ed Warnicke
@edwarnicke

Taylor Carpenter
@taylor

Denver Williams
@denverwilliams

W.Watson
@wavell

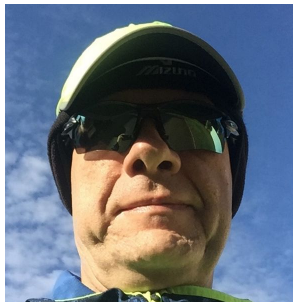
Lucina Stricko
@lixuna

Michael S. Pedersen
@michaels pedersen

Robert Starmer
@robertstarmer

Fred Sharp
@linkous8

Peter Mikus
@rpmikus



Maciek Konstantynowicz
@maciekatbgpnu

Nikolay Nikolaev
@nickolaev

CNF Testbed Contributors

Network Service Mesh

We Welcome Your Participation

- Replicate our results from github.com/cncf/cnf-testbed with an API key from packet.com/cnf
- Package your internal network functions in containers (ideally following cloud native principles) and run on your instance of the testbed
  - We don't need to see the code but would love to see the results
- Create pull requests to have the CNF Testbed run on your bare metal servers or other cloud bare metal servers like AWS i3.metal

Contribute Use Cases and Enhancements

- Contribute new use cases to the CNF Testbed (issues or spec board)
- Create pull requests to improve Kubernetes or OpenStack deployments

Get Connected with the CNF Testbed

- Join the #cnf-testbed channel on CNCF slack
  - slack.cncf.io

- Subscribe to the CNCF Telecom User Group mailing list:
  - telecom-user-group@lists.cncf.io

- Attend CNCF Telecom User Group meetings:
  - https://github.com/cncf/telecom-user-group
  - 1st Mondays at 5pm CET / 8am Pacific Time (US & Canada)
  - 3rd Mondays at 1pm CET / 7pm China Standard Time

Review & Roadmap

Review of CNF Testbed v1 - It Begins

- Initiative started at ONS NA 2018 in Los Angeles
  - Apples-to-apples comparison of CNFs and VNFs
  - What can we re-use from ONAP and other projects?
  - What gaps are missing on the path to cloud native?
  - What is a POC to assist with discussions?

ONAP Demo to Ansible-based v1 CNF Testbed

- Started with onap-demo
- Pivot to building blocks: Docker + Vagrant first
- Next: OpenStack and K8s workload platforms
- VPP based vSwitch for both platforms
- Ansible for additional hardware, host and network provisioning
- Custom use cases with Ansible, scripts and HEAT templates

General Goals - Technology Innovation Review Tool

- Support changing and trying different technology options
- Keep things as simple as reasonable
- Use upstream community tooling
- Use cloud native principles where possible

Use Cloud Native Principles

- Where possible use cloud native principles for all levels (hardware to use case)
  - Immutable hardware
  - Version control all configuration including underlay networking
  - Workload bootstrapping repeatable by automation/pipeline
- Highlight where gaps are missing and out-of-band procedures are used
- Bring focus to technology which is attempting to provide solutions to meet cloud native principles

Key Features of CNF Testbed v2

- Using more in-band components
  - refactor using Helm or kubectl for K8s use cases
  - replace cross-cloud provisioner with Terraform + Kubespray
  - more K8s-native replacements for out-of-band host setup
- Adding support for emerging technology including NSM, DANM, SRIOV device plugins
- Adding new examples:
  - SR-IOV
  - Hybrid K8s + OpenStack service chains
  - workload configs (eg. Nokia CPU Pooler + NSM)

CNF Testbed Roadmap | Nov 2019 to Jan 2020

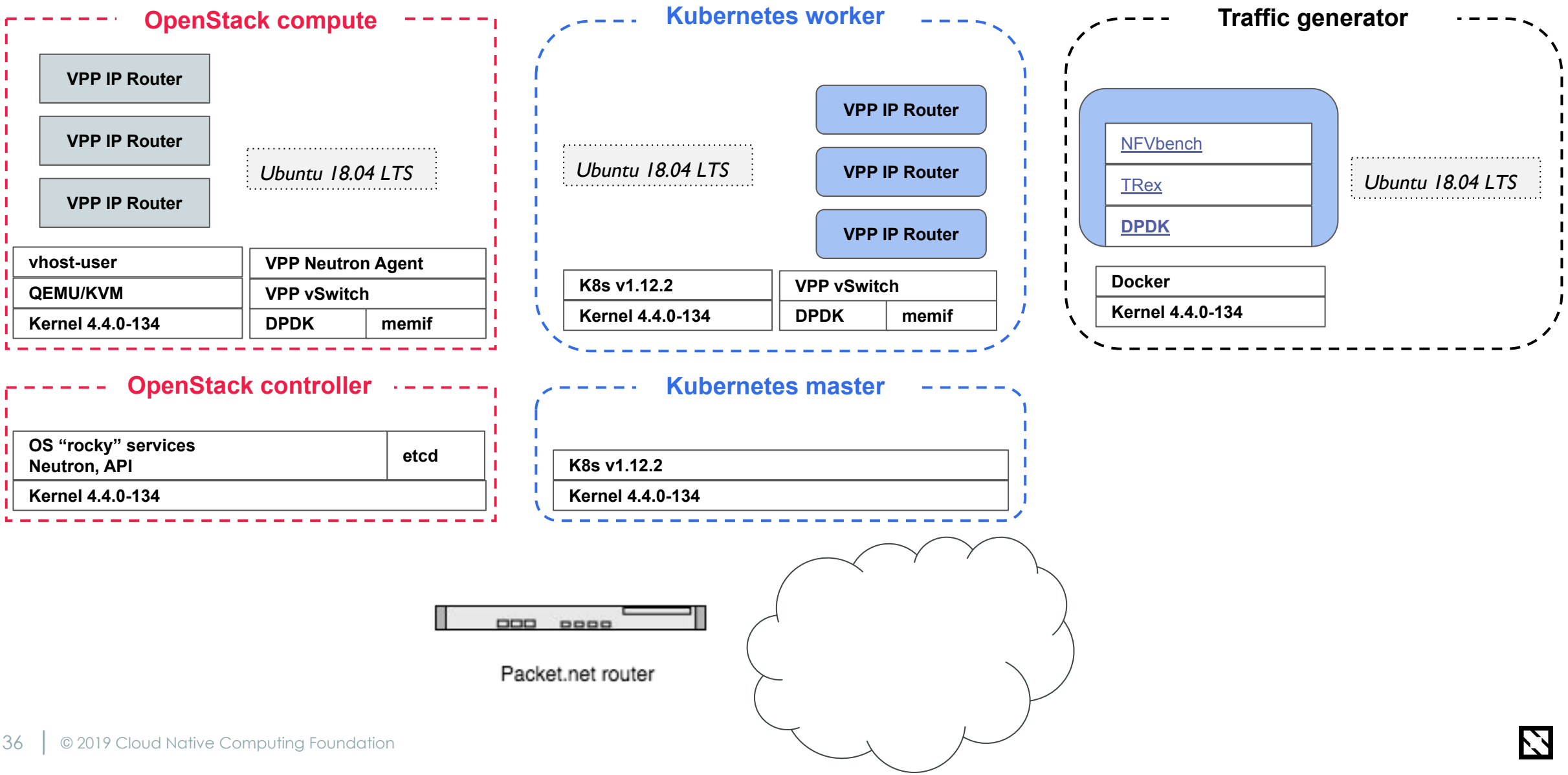
Nov 2019	❑ Clients using different external gateways ❑ NSM SR-IOV Use Case ❑ Separate hardware and workload provisioning stages + Kubespray for K8s	NSMCon, KubeCon NA (Nov 18-21)
Dec 2019	❑ DANM SR-IOV use case ❑ NSM multi-cluster IPsec use case ❑ Multus + CPU Manager use case	[TBD]
Jan 2020	❑ NSM 5G use case ❑ NSM Hybrid K8s+Openstack use case ❑ Kolla/Openstack-helm (TBD)	[TBD]

Overview of Components and Stages

Components of the CNF Testbed

- Hardware provisioning
- Workload provisioning (eg. K8s, OpenStack, SRIOV, VLANs)
- Use Cases and Examples
- Network Functions (eg. Packet Filter, NIC Gateway)
- Testing tools (eg. NFVbench)

CNF Testbed Software Components



 |

CNF Testbed Appendix

CLOUD NATIVE
COMPUTING FOUNDATION

Combating FUD Around MicroVMs

- There has been a lot of Fear, Uncertainty, and Doubt (FUD) about the value of MicroVMs and similar sandbox technology
- Micro virtual machine and sandbox technologies – including Firecracker, gVisor, Kata, Nabla, Singularity, and Unik – are promising options to run **untrusted** code securely on a cluster
- MicroVMs are not necessary to address the **noisy neighbor** issue; that's what the core Kubernetes features **resource limits** (for CPU and memory) and **QoS** (for networking) are for
- More generally, operators are running their own (1st party) code or trusted vendor (2nd party) code, not untrusted 3rd party code

Network Labs (pets) vs. Repeatable Testbed (cattle)

- Networking equipment used to be separate hardware boxes that needed to be integrated in a lab for testing
- Most network labs today are still a group of carefully tended pets whose results cannot be reliably reproduced
- Modern networking is mainly done in software which can and should be checked into source control and replicated at any time
- Network servers should be treated like cattle, not pets

The Importance of a Repeatable Testbed

- A key driver of the Kubernetes project's robustness has been the significant investment in continuous integration (CI) resources
  - Every pull request runs a large automated test suite
  - On any given weekday, we run 10,000 CI jobs
  - Every 2 days, we run a new scalability test of 150,000 containers across 5,000 virtual machines
  - Google provided CNCF a $9 M grant of cloud credits to cover 3 years of testing
- The CNF Testbed is a completely replicable platform for doing apples-to-apples networking comparisons of CNFs and VNFs

Three Major Benefits

1. Cost savings
2. Improved resiliency (to failures of individual CNFs, machines, and even data centers)
3. Higher development velocity

Server Specifications: compute/worker nodes

Packet's M2.xlarge (currently available)

- CPU: 2 x Intel® Xeon® Gold 5120 Processor (28 physical cores)
- RAM: 384 GB of DDR4 ECC RAM
- Storage: 3.2 TB of NVMe Flash and 2 × 120 GB SSD
- **NIC: 10GB dual-port Mellanox ConnectX-4**

packet

Packet's N2.xlarge (available March 2019)

- CPU: 2 x Intel® Xeon® Gold 5120 Processors (28 physical cores)
- Same RAM and storage as the M2-xlarge
- **NIC: 10GB quad-port Intel X710**

Why This Was a Challenging Project: OpenStack

- No existing 100% open source OpenStack installer w/baked-in high-performance dataplane
- Limited choices for high-performance Layer-2 dataplane: SR-IOV, VPP, OVS+DPDK
- OpenStack VPP-networking setup was not well documented
- VPP-Neutron plugin did not support standard OVS setup and configuration (eg. multiple port creation)
- VNF test case deployment configuration with OpenStack-VPP
- Apples-to-apples layer-2 underlay network for K8s
- Physical hardware - Mellanox NICs and proprietary drivers
- Provider limitations - no spanning tree support

Why This Was a Challenging Project: Kubernetes

- Support for dropping in different data plane solutions
- No Kubernetes installer w/baked-in high-performance data plane underlay was available
- No CNI plugins which provide a high-performance layer-2 underlay were available
- Network Service Mesh is a promising approach to dynamically configure the layer 2 network that is currently being manually configured, but it doesn't yet meet our needs
- Physical hardware - Mellanox NICs and proprietary drivers
- Provider limitations - no spanning tree support

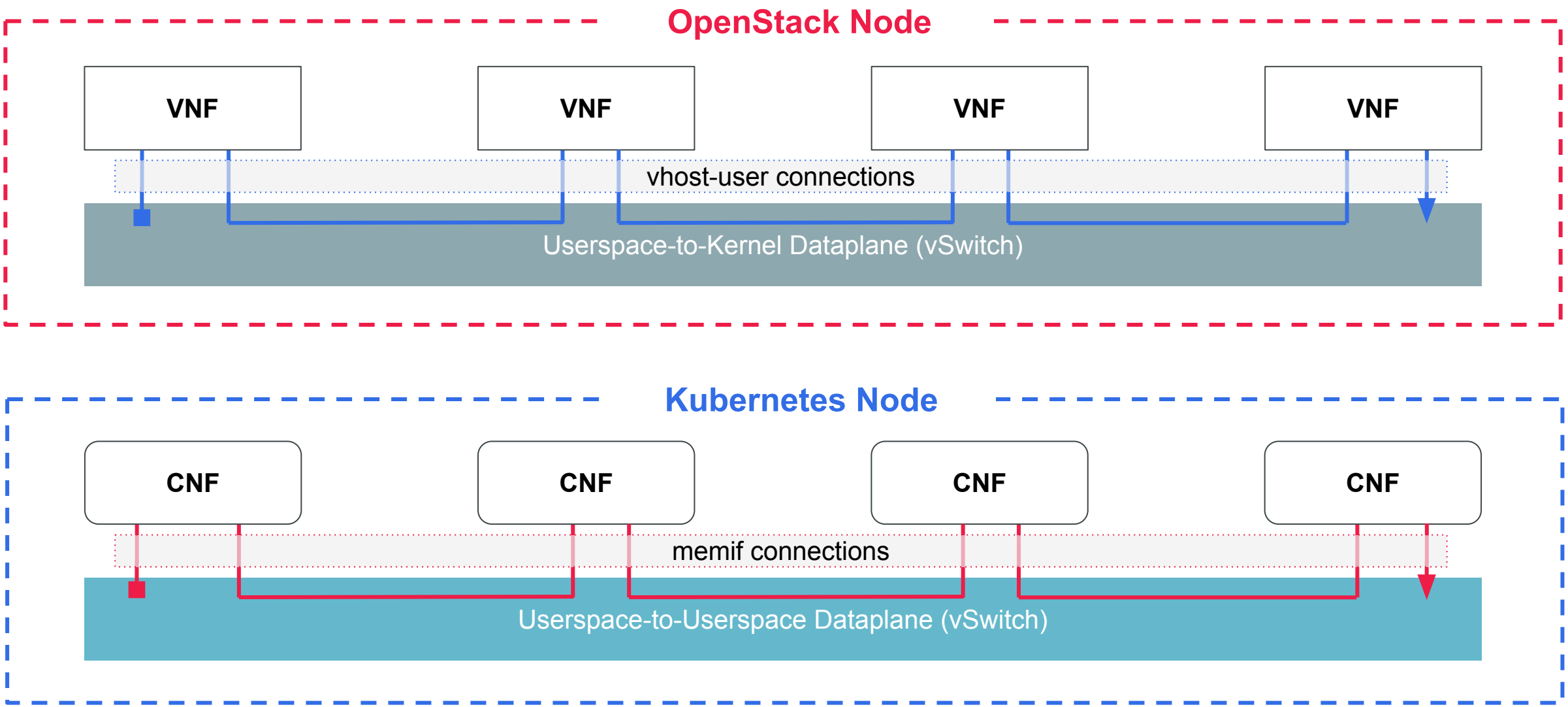
The challenge of transitioning VNFs to CNFs

- Moving from network functionality from *physical* hardware to encapsulating the software in a *virtual* machine (P2V) is generally easier than *containerizing* the software (P2C or V2C)
- Many network function virtualization VMs rely on kernel hacks or otherwise do not restrict themselves to just the stable Linux kernel userspace ABI
  - They also often need to use DPDK or SR-IOV to achieve sufficient performance
- Containers provide nearly direct access to the hardware with little or no virtualization overhead
  - But they expect containerized applications to use the stable userspace Linux kernel ABI, not to bypass it

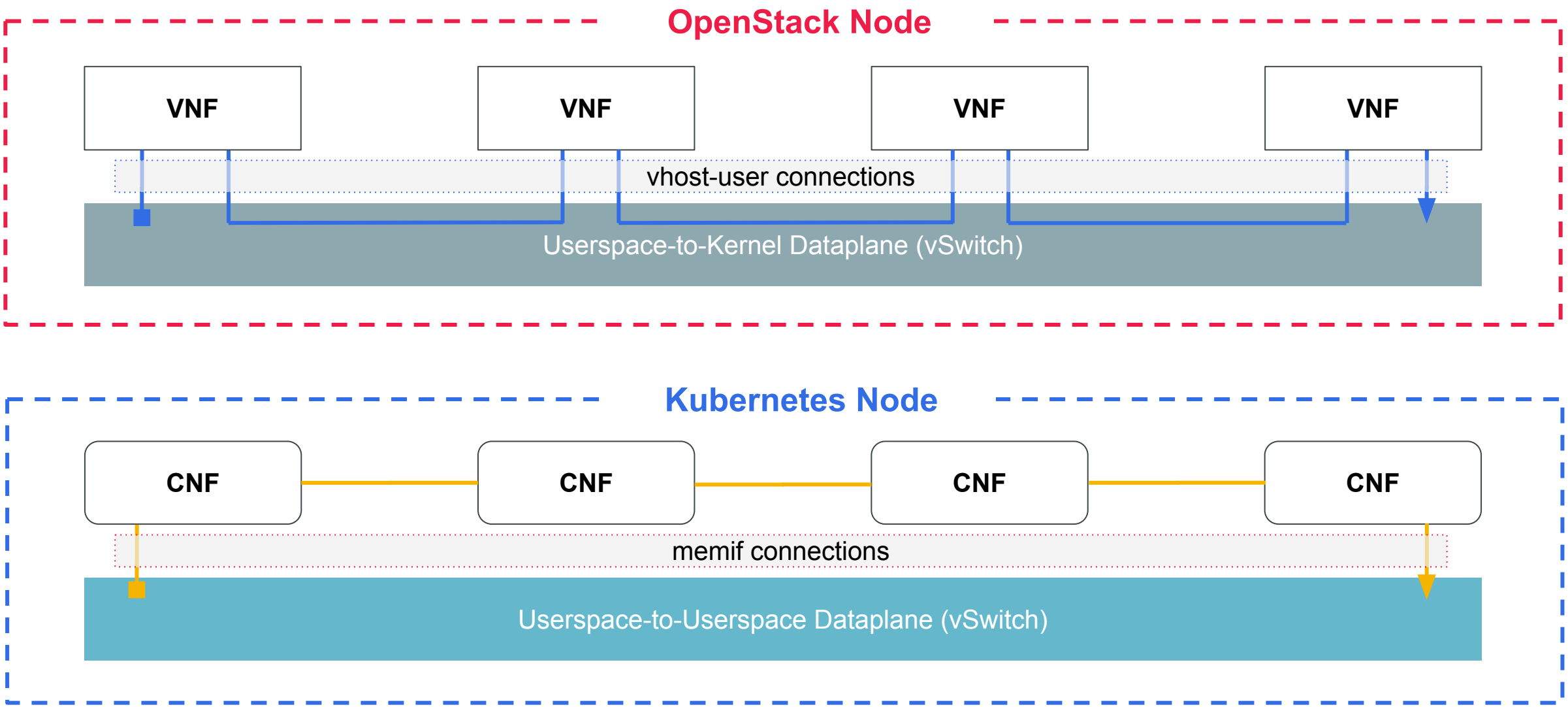
Areas for More Discussion

- The strength of no longer being locked into specific OSs
  - Any version of Linux >3.10 is acceptable
- Multi-interface pods vs. Network Service Mesh
- Complete parity for IPv6 functionality and dual-stack support in K8s
- Security, and specifically recommendations from Google and Jess that come into play when hosting untrusted, user-provided code
  - Possible use of isolation layers such as Firecracker, gVisor, or Kata
- Scheduling container workloads with network-related hardware constraints (similar to what's been done for GPUs)
  - Network-specific functionality like traffic-shaping

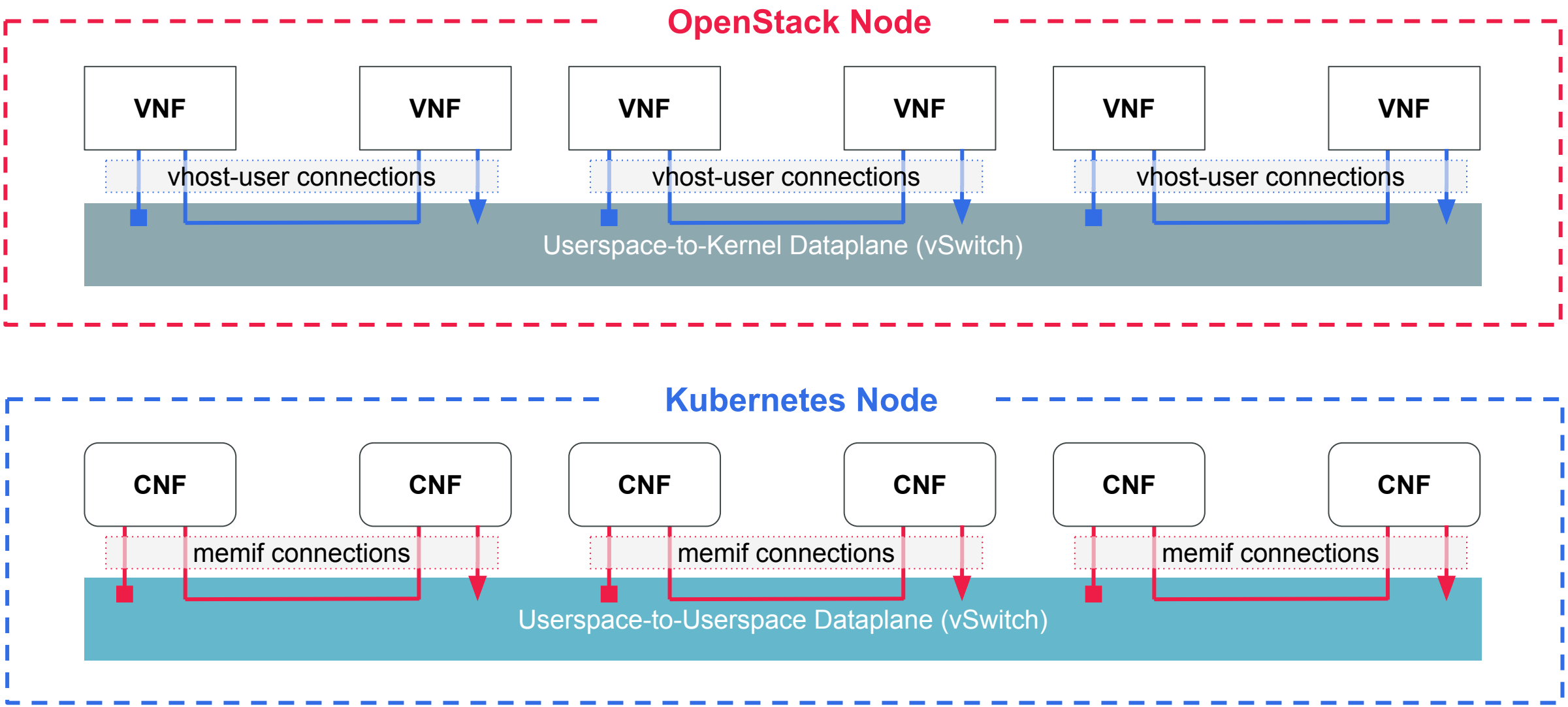
A Service Function Chain: Snake Case



A Service Function Chain: Pipeline Case

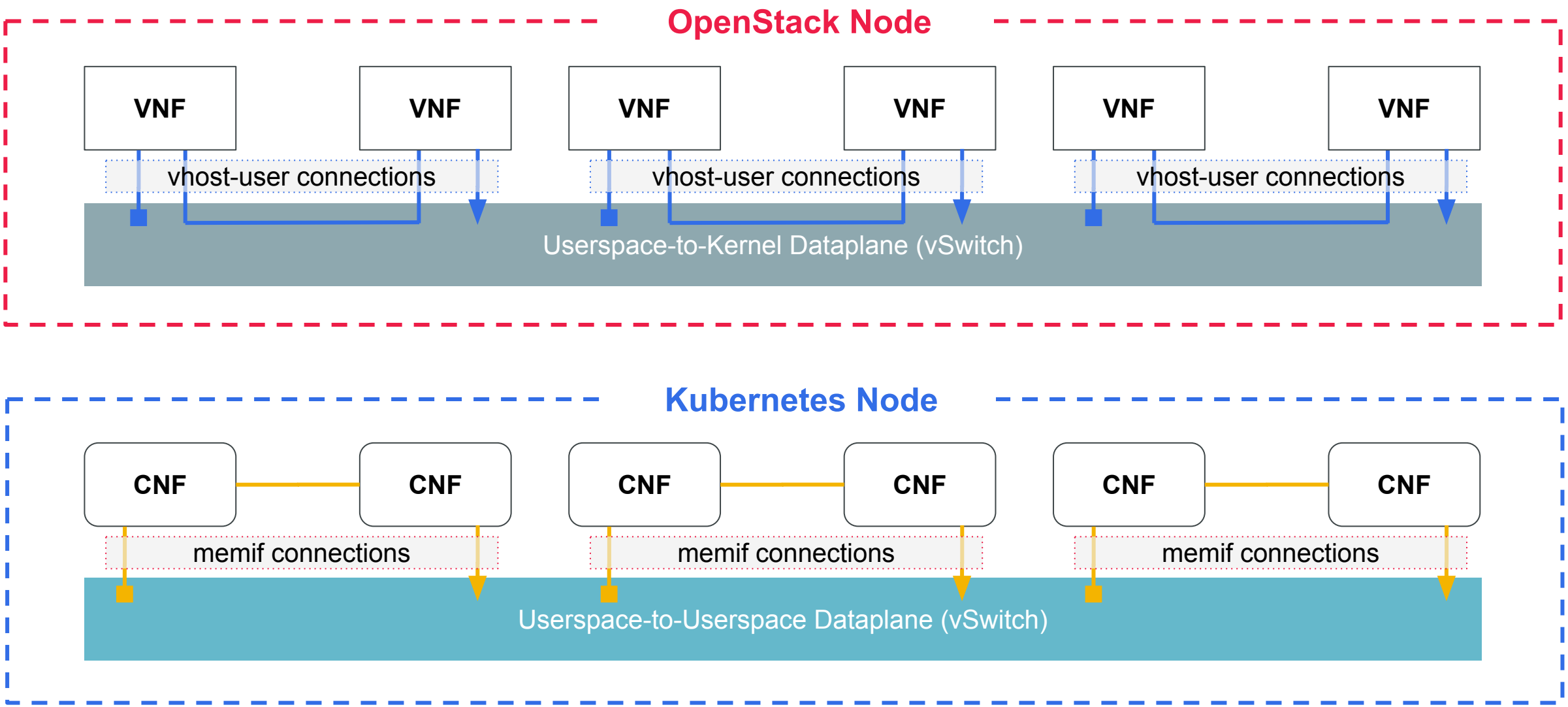


Multiple Service Function Chains: Snake Case



 |

Multiple Service Function Chains: Pipeline Case



 |

CNF Testbed Technical Appendix

CLOUD NATIVE
COMPUTING FOUNDATION

CNF Testbed Deployment stages

Common steps	
Clone https://github.com/cncf/cnf-testbed and install any pre-requites listed in the README	
Create configuration with Packet API, number of nodes, etc (k8s example)	
Run the (k8s or openstack) deploy cluster script which **provisions the Packet machines with Terraform**	
OpenStack	**Kubernetes**
Terraform starts **Ansible which pre-configures the Packet machines (using the openstack infrastructure playbook**) including installing network drivers, optimizing grub and rebooting the compute nodes.	**Cloud-init bootstraps the Kubernetes cluster** on the Packet nodes. *(Note: next release will use kubeadm for bootstrapping k8s)*
Ansible then runs the openstack install playbook, which **configures the Packet switch and VLANs** and then **deploys OpenStack using Chef to the Packet nodes**	The **k8s vpp vswitch installer** script runs the Ansible k8s vpp vswitch playbook which **configures the Packet switch and VLANs**
Ansible then installs & configures VPP as a vSwitch using the Openstack vpp-networking plugin to all compute nodes in the cluster	**Ansible then optimizes the system configuration, installs & configures the VPP vSwitch** and reboots the worker nodes

CNF vs. VNF Performance Comparison

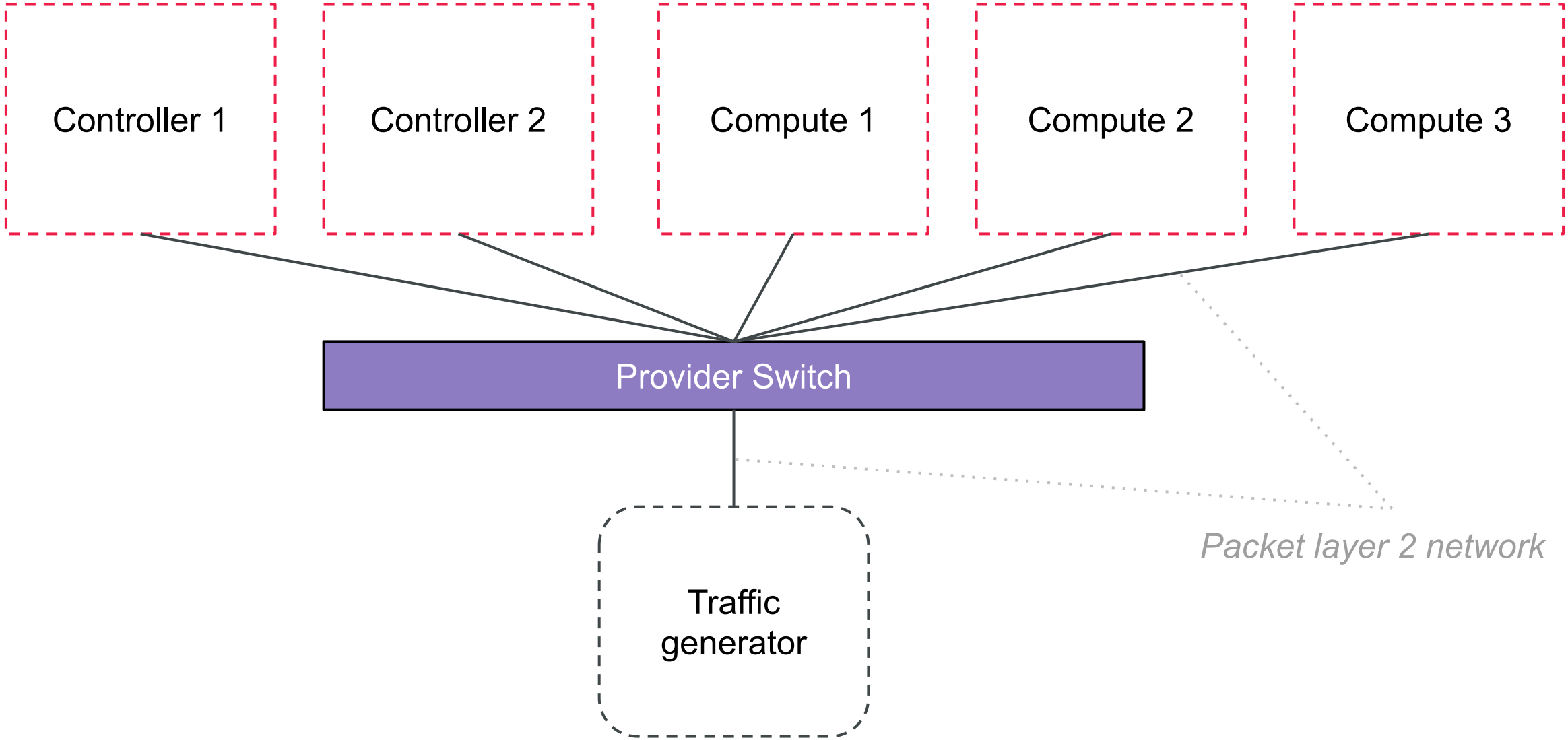
The comparison test bed includes multi-node HA clusters for Kubernetes and OpenStack running chained dataplane CNF and VNFs for performance comparison testing. All software is open source. The entire test bed and comparison results can be recreated by following step-by-step documentation on the CLI with a Packet.net account.

Each test bed will consist of 6 physical machines for each platform - OpenStack and Kubernetes.

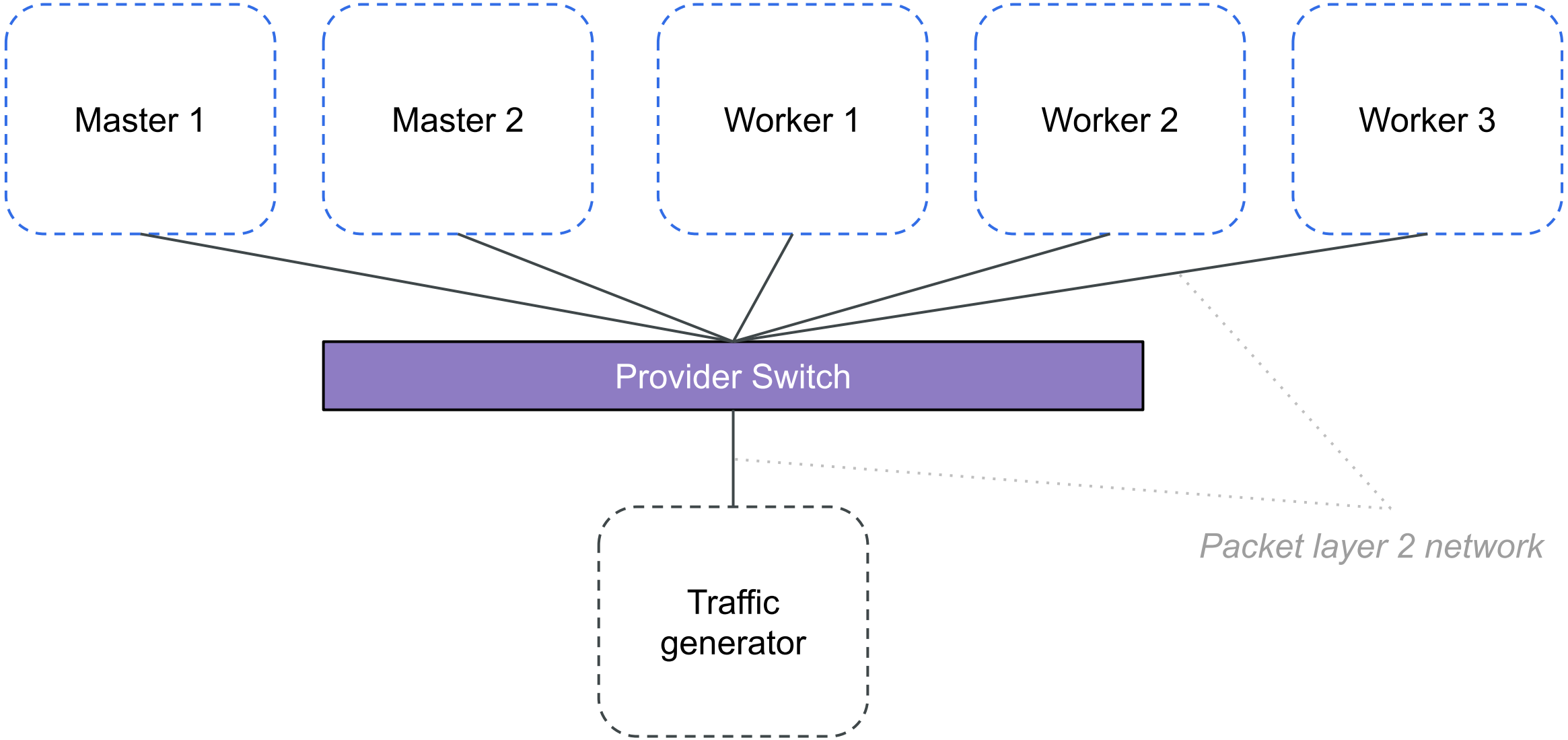
- OpenStack - 2 controllers and 3 compute nodes
- Kubernetes - 2 masters and 3 worker nodes
- Traffic generator - 1 NFVbench system

Provisioning and deployment of K8s and OpenStack clusters includes use of Terraform, Ansible, and Kitchen/Chef. Network functions primarily use VPP and performance testing is done with NFVbench with TRex as the traffic generator.

OpenStack Cluster + Traffic generator

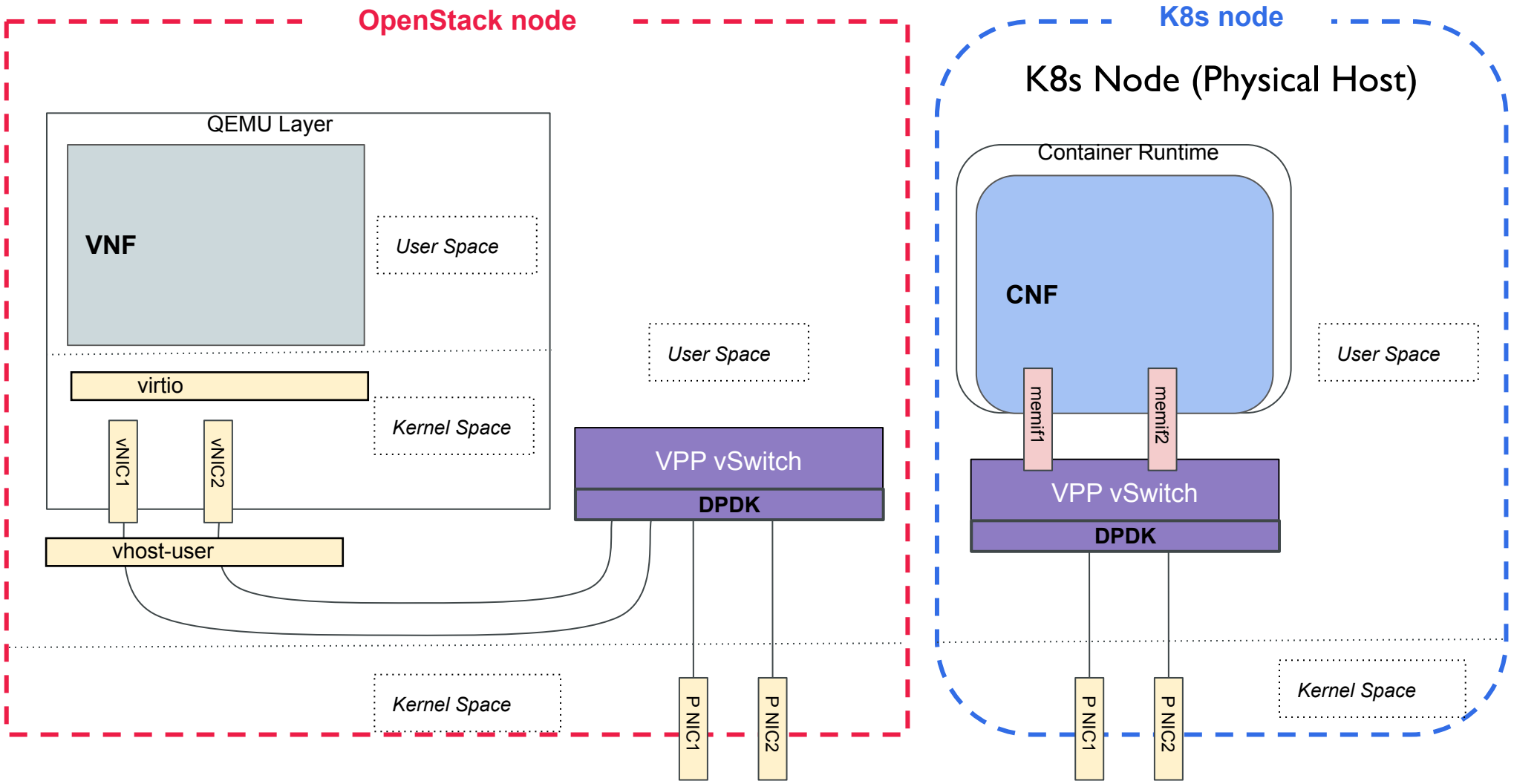


Kubernetes Cluster + Traffic generator

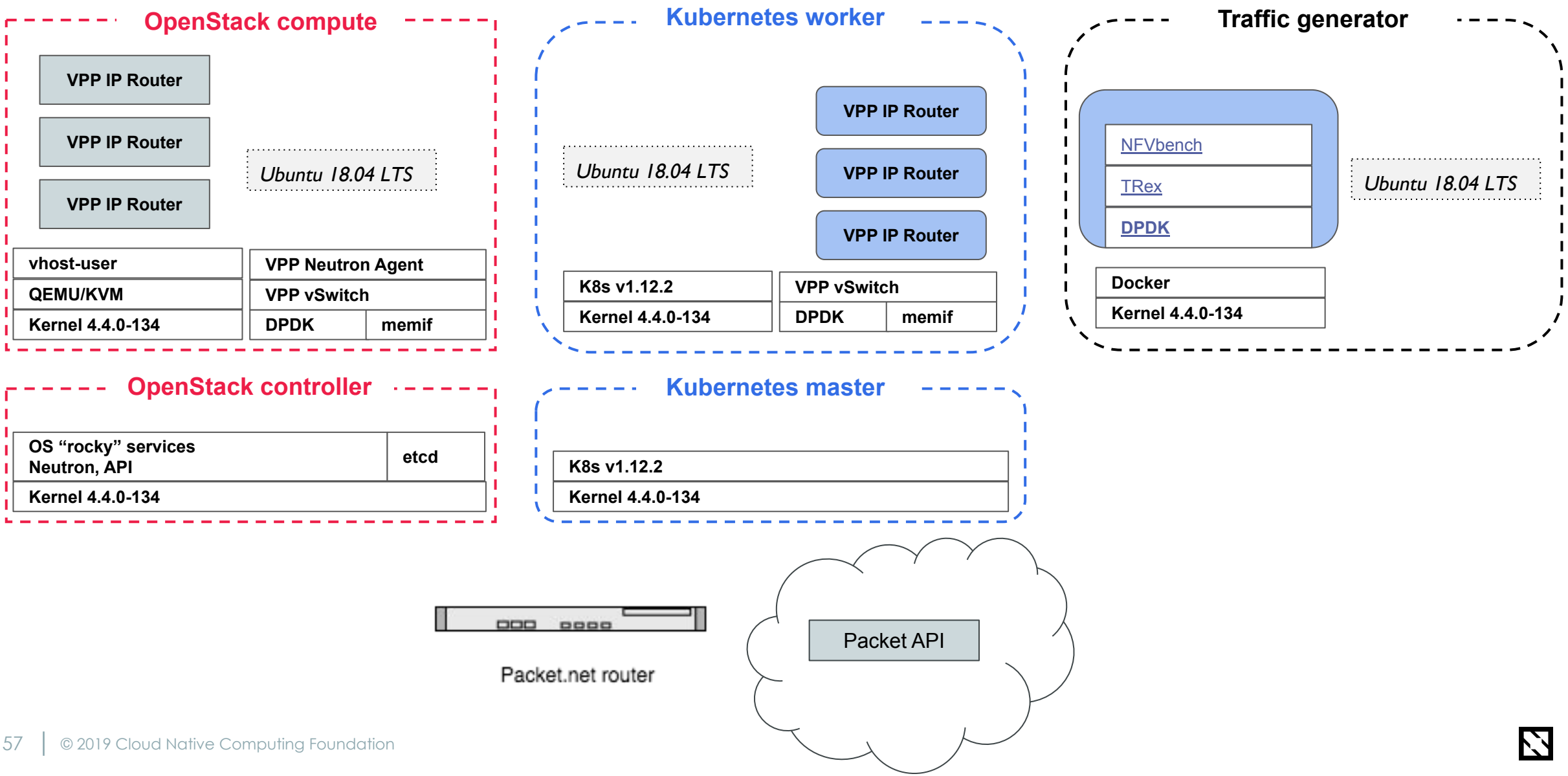


Vhost-user vs memif

Stay in memory & stay in user space!

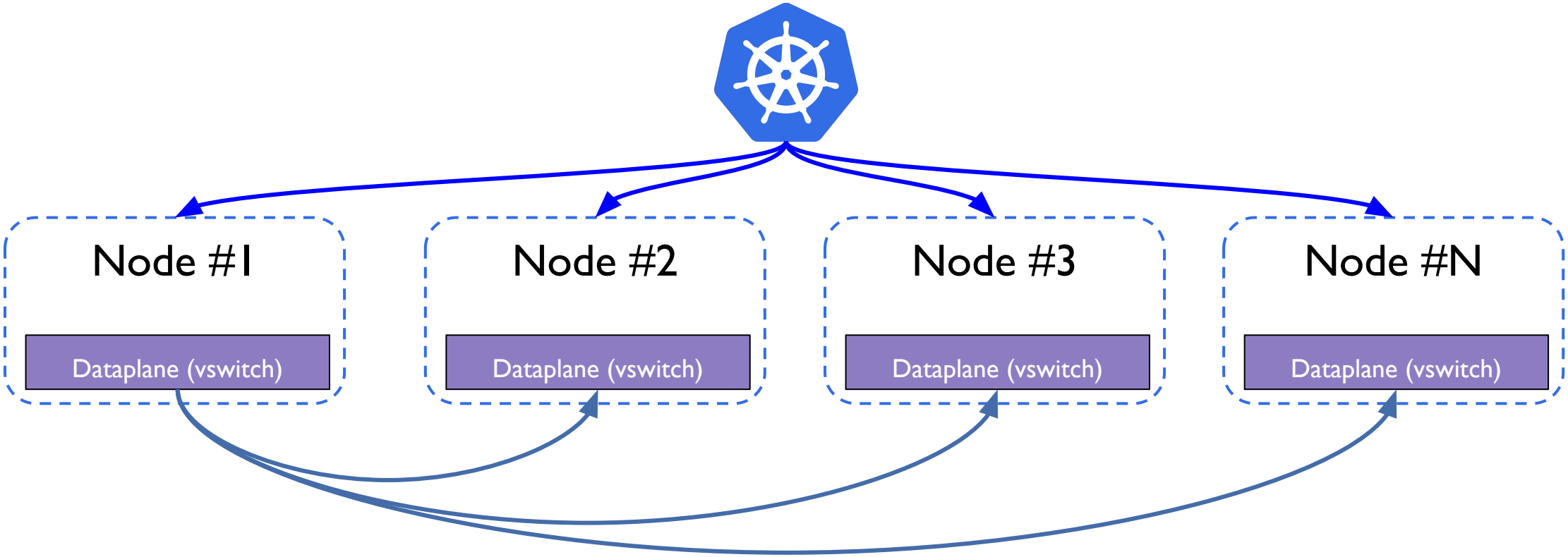


CNF Testbed Software components

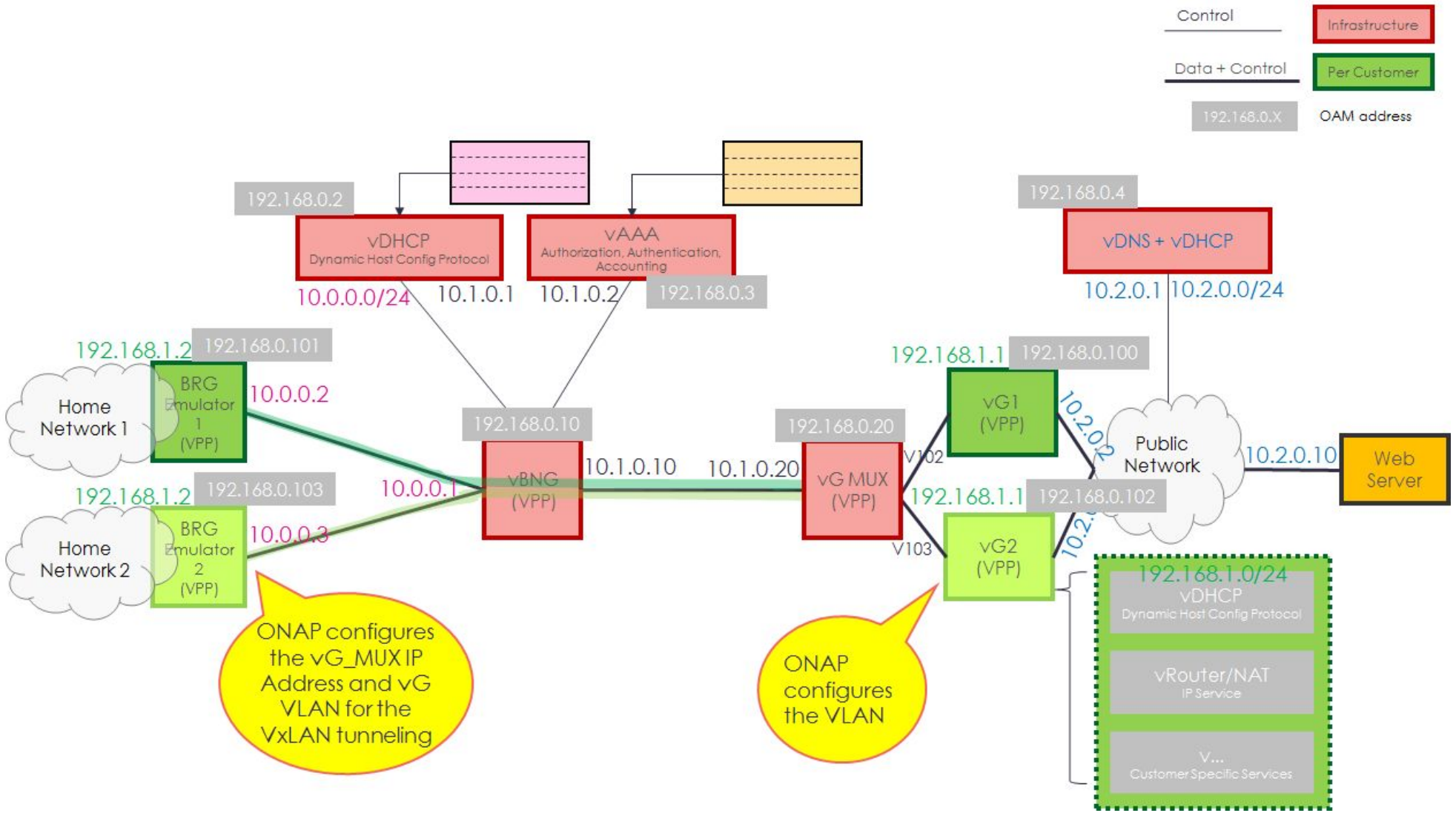


 |

What about inter-node connectivity?



vCPE Use Case



More [details on ONAP wiki]

Project links

- Repo: https://github.com/cncf/cnf-testbed
  - Comparison test case summary and overview
- Comparison and code examples:
  - CNF Edge Throughput and Baseline Single Chain
    - CNF and VNF
    - VPP vSwitch
  - Baseline NF Performance Test
  - In progress: Baseline K8s Chained NF Test
  - Github Projects and Issue

Endnotes

- Let's Move Everything to Kubernetes! by Tal Liron, Jan 2019
- Next-Generation NFV Orchestration by Tal Liron, April 2019
- Cloud Native Edge App & NFV Stack by Srinivasa Addepalli (Intel) and Ravi Chunduru (Verizon), April 2019