

CNF Conformance v0.3

Dan Kohn, Executive Director
dan@linuxfoundation.org
or Slack to Dan Kohn at slack.cncf.io

Summary

- Many of the largest telecom operators have expressed interest in evolving their Virtual Network Function (VNF) infrastructure to enable Cloud native Network Functions (CNFs) running on top of Kubernetes
- CNCF has run an extremely successful conformance program called Certified Kubernetes that has achieved adoption by over 90 organizations, including all cloud and enterprise software providers
- Operators want to create a conformance program to enable interoperability of CNFs from multiple vendors running on top of Kubernetes supplied by a different vendor



Background

Cloud Native Computing Foundation

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

Graduated



kubernetes
Orchestration



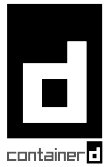
Prometheus
Monitoring



envoy
Network Proxy



CoreDNS
Service
Discovery



Container
Runtime



fluentd
Logging



JAEGER
Distributed
Tracing



Vitess
Storage



Software
Update Spec



OPENTRACING

Distributed
Tracing API



ROOK
Storage



HARBOR
Registry



etcd
Key/Value
Store



Open Policy Agent
Policy



cri-o
Container
Runtime



Key/Value
Store



cloudevents
Serverless



Falco
Container
Security

Incubating



OPENTRACING

Distributed
Tracing API



gRPC
Remote
Procedure Call



CNI
Networking
API



Notary
Security



NATS
Messaging



LINKERD
Service
Mesh



HELM
Package
Management

- Platinum members:

Alibaba Cloud



arm



DELL Technologies



IBM Cloud



NetApp

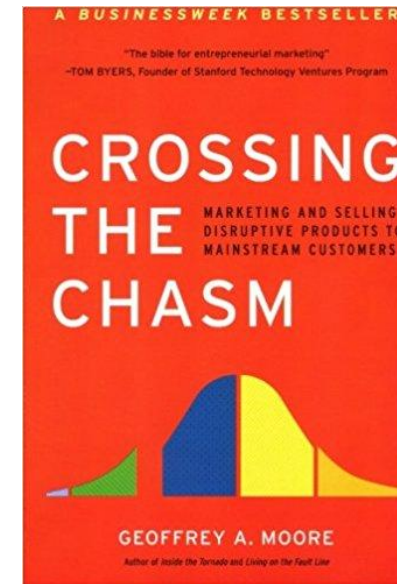
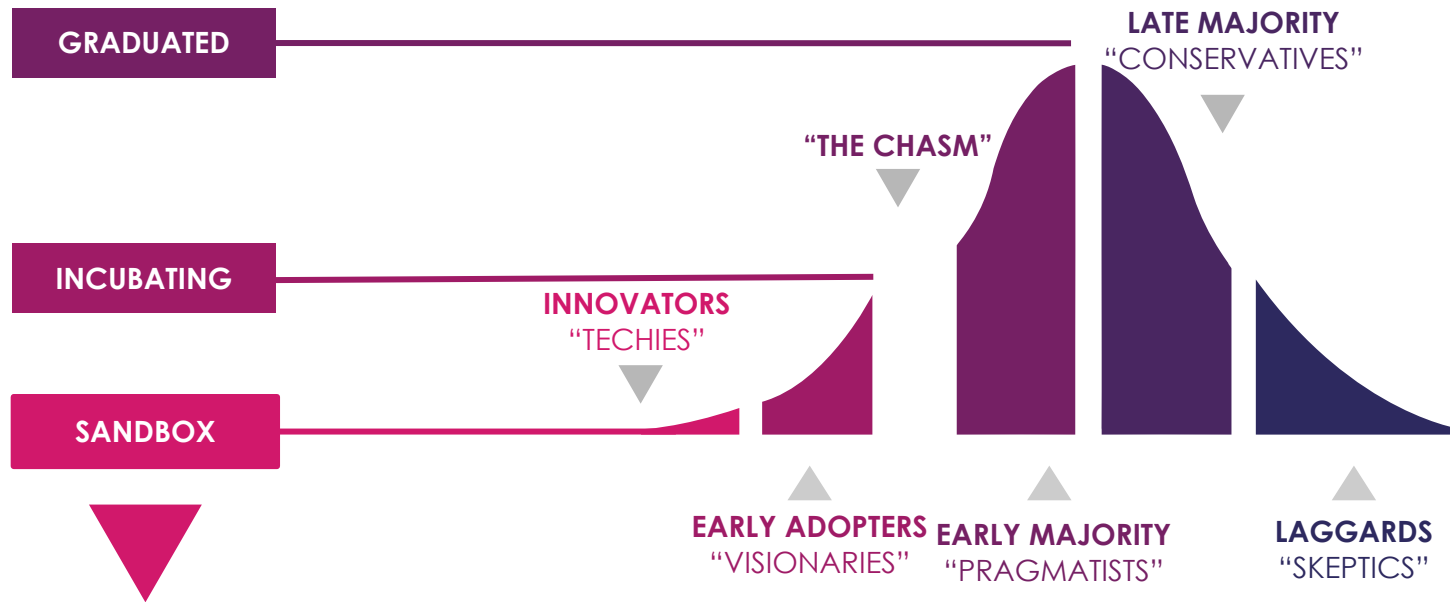
ORACLE























vmware



CNCF Project Maturities



SANDBOX

 spiffe Identity Spec	 SPiRE Identity	 TELEPRESENCE Tooling	 OPENMETRICS Metrics Spec	 cortex Monitoring	 Buildpacks.io Packaging Spec	 Dragonfly Image Distribution	 Virtual Kubelet Nodeless	 KubeEdge Edge	 BRIGADE Scripting
 Network Service Mesh Networking	 OpenTelemetry Telemetry Spec	 OpenEBS Storage	 Thanos Monitoring	 flux GitOps	 STRIMZI Kafka Operator	 in-toto Security	 KubeVirt VM Operator	 LONGHORN Storage	 ChubaoFS Storage



KubeCon + CloudNativeCon



 | 

KubeCon | **CloudNativeCon**

Europe 2020

March 30 – April 2, 2020
Amsterdam, The Netherlands

[LEARN MORE](#)



 | 

KubeCon | **CloudNativeCon**

 **OPEN SOURCE SUMMIT**

China 2020

July 28 – 30, 2020
Shanghai, China

[LEARN MORE](#)



 | 

KubeCon | **CloudNativeCon**

North America 2020

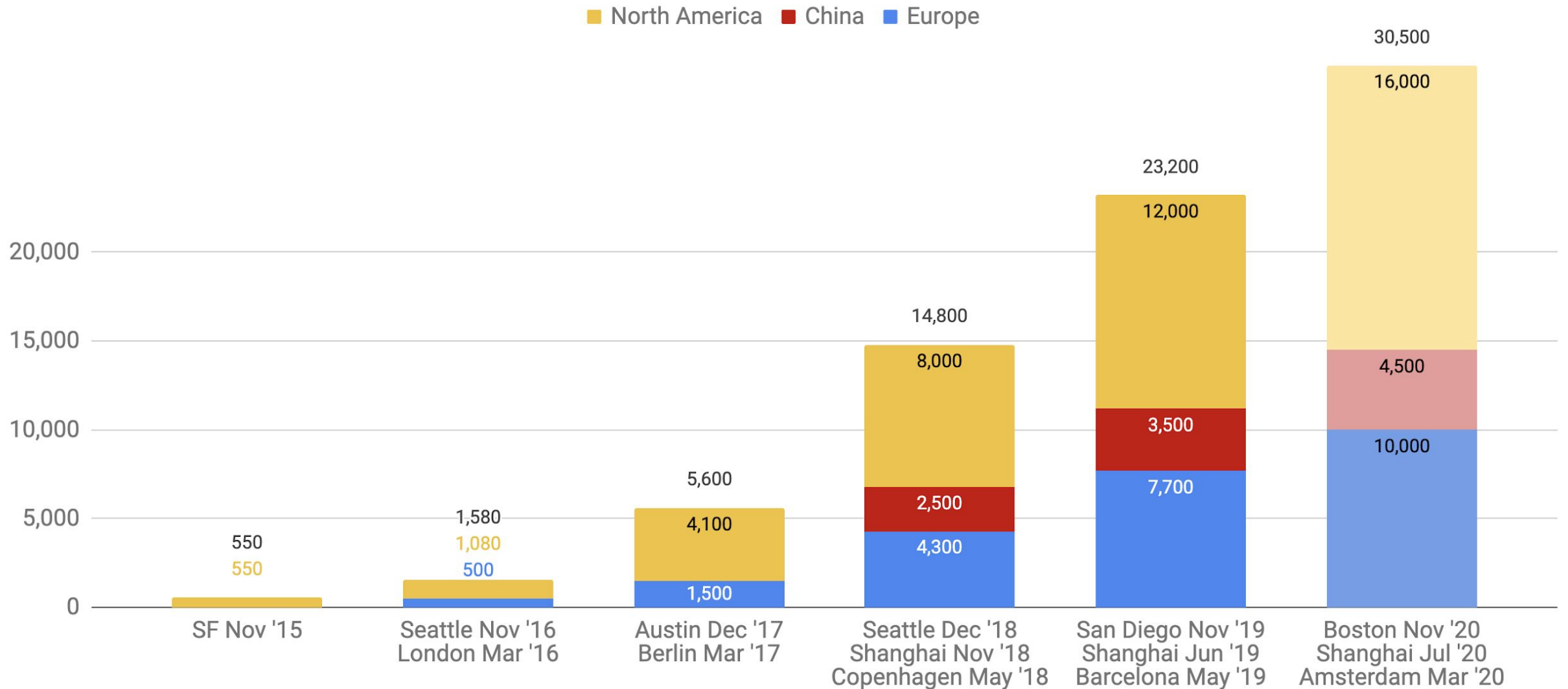
November 17 – 20, 2020
Boston, Massachusetts

[LEARN MORE](#)

kubcon.io



KubeCon + CloudNativeCon Attendance



Certified Kubernetes Conformance

- CNCF runs a software conformance program for Kubernetes
 - Implementations run conformance tests and upload results
 - Enables use of mark and more flexible use of Kubernetes trademark for conformant implementations
 - cncf.io/ck



96 Certified Kubernetes Partners



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 emeritus, SIG-Architecture



Why Organizations Are Adopting Cloud Native

1. Better resource efficiency lets you to run the same number of services on less servers
2. Improved resiliency and availability: despite failures of individual CNFs, machines, and even data centers
3. Cloud native allows multi-cloud (switching between public clouds or running on multiple ones) and hybrid cloud (moving workloads between your data center and the public cloud)
4. Cloud native infrastructure enables higher development velocity – improving your services faster – with lower risk



Cloud native Network Functions (CNFs)

Cloud native Network Function (CNF) Definition

A cloud native network function (CNF) is a cloud native application that implements or facilitates network functionality. A cloud native network function consists of one or more microservices, and has been developed using [Cloud Native Principles](#) including immutable infrastructure, declarative APIs, and a “repeatable deployment process.”



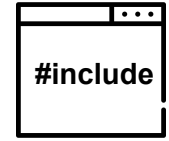
Evolution

- Physical Network Functions (PNFs) and Virtual Network Functions (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

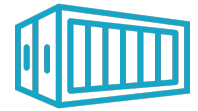


CNF Testbed

- What it is
 - Open source [initiative](#) from CNCF
 - Creates and deploys a complete telecom-ready Kubernetes stack along with several open source CNFs
 - Collaborating with CNCF Telecom User Group
 - Runs on top of on-demand hardware from the bare metal hosting company, Packet, but can be ported to other environments
- Goals
 - 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



NETWORK FUNCTIONS



CONTAINERS



kubernetes

KUBERNETES



BARE-METAL
SERVER

HARDWARE

packet



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 that can and should be checked into source control and replicated at any time
- Network servers should be treated like cattle, not pets



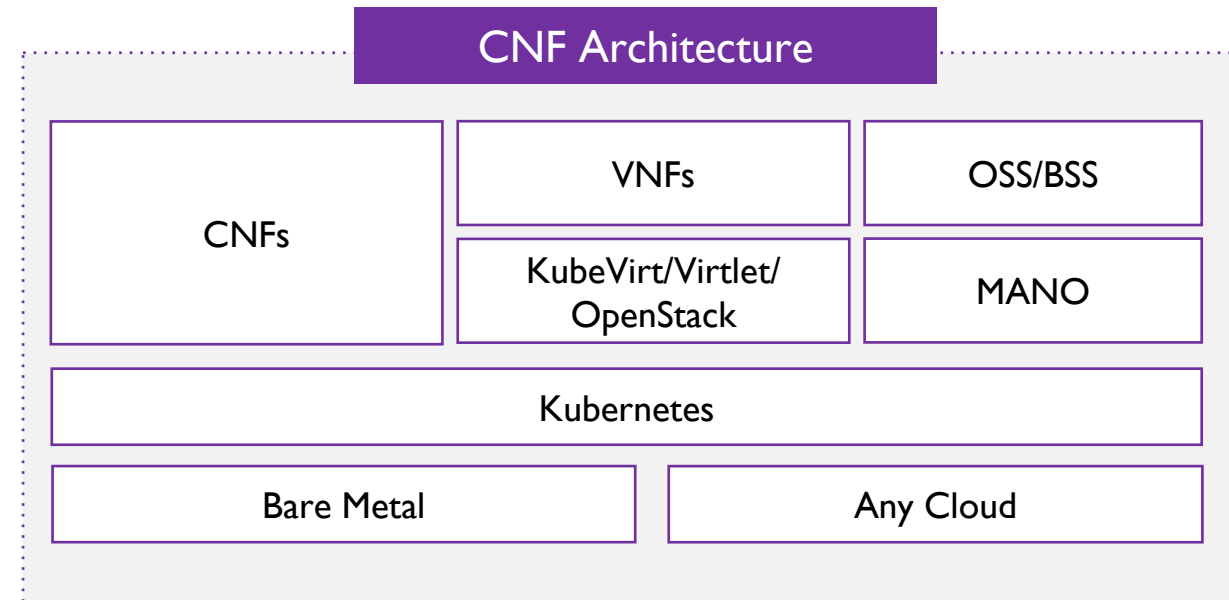
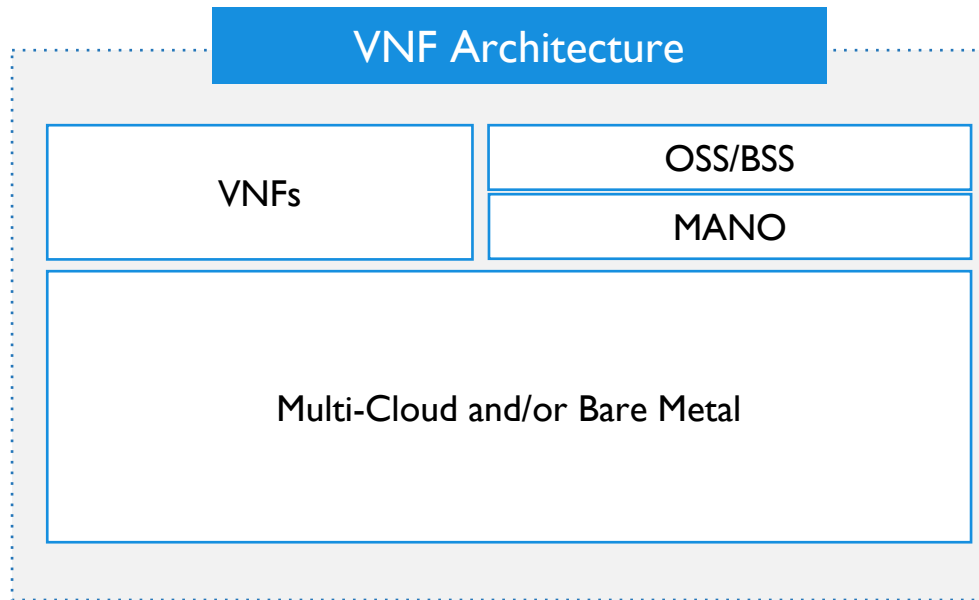
The Importance of a Repeatable Testbed

- 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 \$9M [grant](#) of cloud credits to cover 3 years of testing and AWS has granted over \$200k a year
- The CNF Testbed is a completely replicable platform for doing apples-to-apples networking comparisons of CNFs



CNF Best Practices Ideas

Evolving from VNFs to CNFs



Not-So-Great 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 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:** They should support horizontal scaling (across multiple machines) and vertical scaling (between sizes of machines)
 - **Configuration and Lifecycle:** via ConfigMaps/CRDs, [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
- Not-So-Great 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 “bronze” or “silver” CNF may be critical
- Operators may write into request for proposals (RFPs) a requirement for silver CNFs and/or specific aspects of gold



CNF Conformance Plan

Goals

- Enable a thriving ecosystem of cloud native telecoms where Cloud-native Network Functions (CNFs) from different vendors can interoperate and run on top of the same cloud native infrastructure (i.e., Kubernetes)
- Provide an open source test suite to enable both open and closed source CNFs to demonstrate conformance and implementation of best practices



Planned Implementation

- Develop an open source [CNF Conformance Test Suite](#)
 - Initial work will be seeded by Vulk Coop and ii.coop, CNCF contractors who developed the CNF Testbed, APInoop, and are working on K8s conformance tests
 - Engagement needed from developers from telecom operators and vendors
 - Planning to apply to become a CNCF sandbox project
 - Aspiration is to leverage existing upstream work as much as possible
- Static tests will run on the CNF artifact, outside of K8s
- Runtime tests will demonstrate that the CNF installs in a K8s cluster and successfully responds to tests provided by the CNF, upstream tests from other conformance projects, or a packet generator
- A basic principle of the test suite is that it will leverage upstream test suites, linters, and validations as much as possible, and any bug fixes or enhancements will be submitted upstream first



Components of CNF Conformance Test Suite

- [APIsnoop](#): Use Kubernetes audit logs to track every K8s API call used by a CNF and therefore specify what version of K8s is required and any beta or alpha API usage
- Helm v3 chart testing ([ct](#)): linting and testing
- Helm v3 installation: Functional test on CNF Testbed
- Dual Stack (IPv4 & v6): Functional test on CNF Testbed
- Validate Prometheus metrics using [promtool](#)
- Other tests (need plans)
 - Validate OpenTelemetry metrics
 - Validate Fluentd-compatible logging
 - Demonstrate that CNF is stateless and/or relies on separate service to store state
 - Demonstrate that any requirement for custom hardware uses [device plugin](#) API
- Performance testing
 - Demonstrate that the CNF delivers a set amount of traffic throughput and supports a set amount of sessions, when running on standardized hardware



Lessons from Certified Kubernetes

- Certified Kubernetes is based on the open source conformance test suite developed in parallel with new versions of Kubernetes
- It includes some unique properties for a certification platform:
 - Organizations self-certify by running the test suite against their Kubernetes platform
 - Output log results are submitted via a GitHub pull request
 - These results are reviewed by CNCF staff, and then the certification is approved
 - Any user can later verify that the platform remains conformant, which has the effect of crowd-sourcing the validation
 - Certified implementations can then use the Certified Kubernetes logo and add Kubernetes to their product name (e.g., Acme Kubernetes Engine)
- Decision-making is done by the certification working group that operates in conjunction with Kubernetes's SIG Architecture



From Conformance to Certification

- If the CNF Conformance Test Suite proves useful, it would be natural to build out a certification program for CNFs
- This could replicate the self-certification process used by the Certified Kubernetes program
- It can be done in conjunction with LF Networking's OPNFV Verification Program ([OVP](#)) and the Common NFVI Telco Task Force ([CNTT](#))
- One option would be to emulate [LEED Certification](#) and the Core Infrastructure Initiative [Best Practices Badge](#) by having the test suite specify scoring ranges as bronze, silver, etc.
 - We also want probably want to explicitly include a “not-so-great” result meaning that that the CNF does not implement some base level of best practices
- It is not clear that there is any need for platform conformance and certification beyond what is already provided by Certified Kubernetes



Open Questions

The CNI Trap

- The Container Network Interface ([CNI](#)) is a CNCF-hosted project that provides network interfaces for plugins to Kubernetes. There are several dozen CNI [plugins](#) today
- One concern that operators have expressed is that if a vendor's CNFs require the use of that vendor's CNI plugin – and especially if it is the only CNI plugin that can be used – then “conformant” Kubernetes implementations could wind up as single-vendor walled gardens
- On the other hand, if CNFs can only rely on the lowest common denominator of CNI functionality, then they will not be able to take full advantage of the underlying hardware
- The solution may be a set of CNI profiles that define sufficient performance without requiring use of specific CNI plugins



Conformance Profiles

- One of the reasons that Certified Kubernetes has been so successful is that it has limited the number of conformance profiles – so far, to only 1
- There are not, for example, different profiles for public and private clouds
- At some point, a profile may be added to support Windows workloads
- One fear of a CNF Conformance program is that it will support so many profiles – such as different CNI plugins, or implementation of [Network Service Mesh](#), or availability of certain hardware – that interoperability will be far less likely



Privileged Pods

- Best practice is to follow the [Principle of Least Privilege](#)
 - Unprivileged pods are also less likely to conflict
- One pattern that is emerging for cloud native telecoms is to use a [privileged](#) pod to initialize nodes with networking functionality such as [Network Service Mesh](#), [SR-IOV](#), and/or [eBPF/Cilium](#)
 - This would enable CNFs to utilize advanced networking without needing to run in privileged mode
 - This is somewhat analogous to Unix daemons that initially run with privilege and then use [setuid](#) to drop privileges



MANO Integration

- One open question on CNF Conformance is whether and how to test for Management and Orchestration (MANO) integration
- The leading MANO offerings are [ONAP](#) from the Linux Foundation and [OSM](#) from ETSI
- Some operators have expressed an interest in deploying CNFs that are **not** managed by their existing MANO systems



What about CNTT RA2?

- The Common NFVI Telco Task Force ([CNTT](#)) is a joint effort of the Linux Foundation's [LF Networking](#) and the [GSM Association](#)
- CNTT has an active effort underway to define a reference architecture ([RA2](#)) for Kubernetes-based CNFs
- The CNF Conformance test suite envisioned in this proposal would likely be able to validate conformance with RA2
- The expectation is that this test suite would feed into the RA2 process by encoding best practices into software tests



Outline of Some Potential Tests

Compatibility Tests

CNFs should work with any Certified Kubernetes product and any CNI-compatible network that meet their functionality requirements

- Does the vendor's CNI plugin conform to the [CNI specification](#)?
- Does the CNF use Alpha Endpoints (e.g. using [APIsnoop](#))?
- Does the CNF use Beta Endpoints (e.g. using [APIsnoop](#))?
- Does the CNF use only K8s Conformance Tested / Generally Available Endpoints (e.g. using [APIsnoop](#))?



Statelessness Tests

State should be stored in a [Custom Resource Definition](#) or a separate database (e.g. [etcd](#)) rather than requiring local storage. The state should be resilient to node failure

- Can we reset the container and verify that the CNF comes back up (e.g. using [Litmus](#) chaos testing)?
- Can we reset any child processes that the parent process started, and see that those child processes are reaped (e.g. using [Falco](#) or [Sysdig Inspect](#))?



Scalability Tests

CNFs should support horizontal scaling (across multiple machines) and vertical scaling (between sizes of machines) using the native K8s [kubectl](#) command.

- Can we increase/decrease capacity (without signing physical contracts or calling anyone)?
- Can a CNF be scaled automatically based on load (e.g using [CNF Testbed](#) load test use case)?
- Does the CNF control layer respond to retries for failed communication (e.g. using [Pumba](#) or [Blockade](#) for network chaos and using [Envoy](#) for retries)?



Security Tests

CNF containers should be isolated from one another and the host. They can be hardened using tools like [OPA Gatekeeper](#), [Falco](#), [Sysdig Inspect](#) and [gVisor](#)

- Are there any containers running in [privileged mode](#) (e.g. using [OPA Gatekeeper](#))?
- Are there containers accessing sensitive files, paths or writing files to sensitive directories on the host (e.g. using [Falco](#))?
- Are there any shells running inside a container?
- Is there a server process spawning a child process of an unexpected type?
- Is there a standard system binary, such as “`ls`”, that is making an outbound network connection?



Configuration and Lifecycle Tests

Configuration and lifecycle should be managed using [ConfigMaps](#), [Operators](#), or other [declarative interfaces](#).

- Is the CNF installed using a [versioned](#) Helm v3 chart?
- Is there a liveness entry in the helm chart and is the container responsive to it after a reset (e.g. by checking the [helm chart entry](#))?
- Is there a readiness entry in the helm chart and is the container responsive to it after a reset?
- Can we start the pod/container without mounting a volume (e.g. using [helm configuration](#)) that has configuration files?
- Can we stop pods/containers and see that the application continues to perform (e.g. using [Litmus](#) or [Kube-monkey](#))?
- Can we reset any child processes that the parent process started, and see that those child processes are reaped (ie. monitoring processes with [Falco](#) or [sysdig-inspect](#))?
- Can the CNF perform a rolling update (i.e. [kubectl rolling update](#))?



Observability Tests

CNFs must externalize their internal states in a way that supports metrics, tracing, and logging in order to maintain, debug, and give insight into their protected environments.

- **Logging:** The CNF supports Fluentd-compatible logging
 - Is there traffic to Fluentd?
- **Tracing:** The CNF supports [OpenTelemetry](#)-compatible tracing
 - Does the CNF generate Open Telemetry compliant data?
 - Is there traffic to Jaeger?
- **Monitoring:** The CNF supports an [OpenMetrics](#) interface that Prometheus and other monitoring tools can use.
 - Is there traffic to [Prometheus](#)?



Installable and Upgradeable Tests

The CNF Conformance suite will check for usage of standard, in-band deployment tools such as [Helm](#) (version 3) charts

- Is the Helm chart valid (e.g. using the [helm linter](#))?
- Does the install script use Helm v3?
- Can the CNF perform a rolling update (i.e. [kubectl rolling update](#))?



Hardware Resources and Scheduling Tests

The CNF container should access all hardware and schedule to specific worker nodes by using a [device plugin](#).

- Is the CNF accessing hardware in its configuration files?
- Does the CNF access hardware directly during run-time (e.g. accessing the host /dev or /proc from a mount)?
- Does the CNF access hugepages directly instead of via [Kubernetes resources](#)?
- Does the CNF Testbed performance output show increased throughput and sessions when the scale is increased (e.g. using the [CNF Testbed](#) (vendor neutral) hardware environment)?



Tools

- Current work taking place at <https://github.com/cncf/cnf-conformance>
- Using [k-rail](#) to validate that CNFs continue to carry traffic when appropriate policies are enforced
- Using [NFVBench](#) and [TRex](#) to generate network traffic



Where to Discuss

Get Connected with the Telecom User Group

- [Book](#) a time to meet with Dan Kohn or email your comments to dan@linuxfoundation.org or Slack me at slack.cncf.io
- Join the #tug 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



Events Where the TUG Will Be Meeting

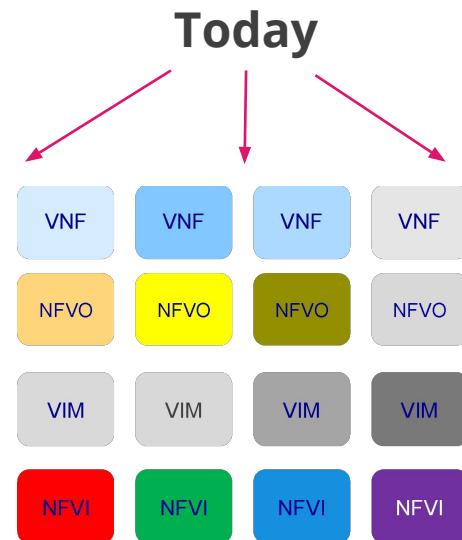
- [Linux Foundation Member Summit](#)
 - Lake Tahoe: March 10–12, 2020
- [KubeCon + CloudNativeCon Europe](#)
 - Amsterdam: March 30-April 2, 2020
- [Open Networking & Edge Summit](#)
 - Los Angeles: April 20-21, 2020



Appendix

LF Networking OVP: End to End Compliance & Verification Program

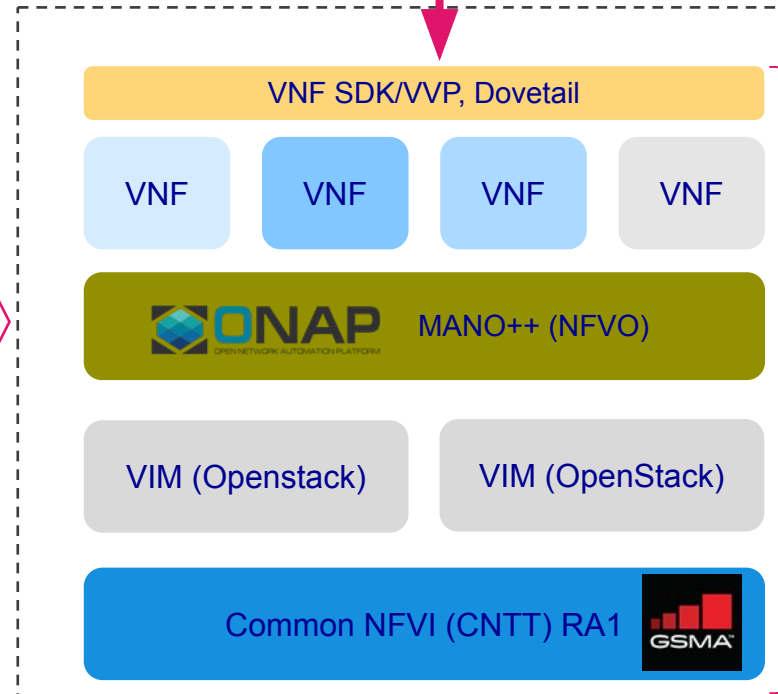
Accelerating Deployment – Reduce Operator & Supplier Integration/Interop Testing Intervals By 50%



Separate solutions for VNF, MANO, NFVi lifecycle management & onboarding

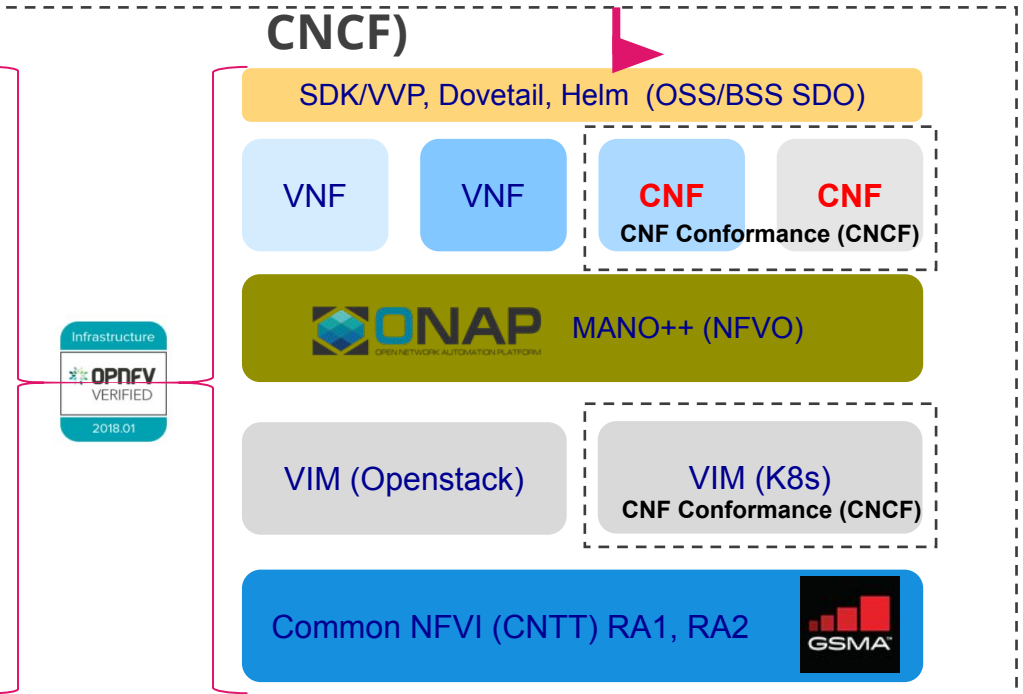
Operator Services on Boarding

LFN OVP Phase 1 (in collaboration with GSMA/CNTT)



Simplified VNF on-boarding, Reduced time and cost of operationalizing a VNF through a common NFV Framework*

LFN OVP Phase 2 (additional collaboration with CNCF)



Hybrid (VNF & CNF) Compliance and Verification with OVP Ph2 in partnership with CNCF Conformance tests for CNFs and K8s in CNCF

Current Approaches for CNF Modeling

#	CNF Modeling Approach	Reference implementation	Onboarding	VNFC Model	Design Output	Runtime
1	Heat + Helm + TOSCA	ONAP K8s Cloud Regions (Reference : link)	Dummy VNF Heat template	Helm Chart for Pod-based VNF Component Heat/TOSCA for VM based VNFC	TOSCA CSAR with Heat & Helm chart as artifacts	Helm chart consumed by K8s plugin for Pod-based VNFC, Rest consumed by Orchestrator
2	Extended TOSCA Types	Cloudify (Reference: link , link)	TOSCA	TOSCA	TOSCA blueprint	TOSCA blueprint processed by K8s plugin or Infra plugin
3	TOSCA Kubernetes profile	Puccini (Reference : link)	NA (Not an orchestrator) , input can be TOSCA	TOSCA	Clout – Can generate specific CNF or VNF specs	Clout to respective Infra-specific template
4	TOSCA + Helm chart as artifact	NA, See Note 1, Note 2	Dummy VNFD TOSCA template	Helm chart for Pod-based VNF Component, TOSCA for VM based VNFC	TOSCA CSAR with Helm chart as artifact	TOSCA template consumed by Orchestrator and Helm chart consumed by the VNFM/CISM/PaaS
5	Extended TOSCA types+ K8s Custom Resources/Operators	ONAP K8s Network CRDs (Reference : link)	TOSCA	TOSCA	TOSCA	Plugins that leverage TOSCA model to invoke Custom Resources implemented in K8s + Controllers for Custom Resource processing

- Note 1: IFA011 Support for Pods Contribution ([link](#)) , VDU extension to OsContainerDesc. Helm Chart is being refred as one of the potential deployment method in ETSI IFA029
- Note 2: May be a recommended approach in ONAP



Current Approaches: Pros and Cons

#	Approach	Pros	Cons
1	Heat + Helm + TOSCA	<ul style="list-style-type: none"> Accommodate VNF/CNF modeling requirements No cross dependency , can independently describe the NF in respective modeling format of choice 	<ul style="list-style-type: none"> Customized approach for ONAP Complexity of managing multiple formats of descriptors requires additional skills Currently based on Helm 2 Complexity to pass CNF instantiation inputs
2	Extended TOSCA Types	<ul style="list-style-type: none"> Logical extension to the existing VNF modeling approach Supports multiple mechanisms for attaching the CNF-specific K8s resource artifacts 	<ul style="list-style-type: none"> Require additional plugins to interpret and orchestrate for specific infrastructure. No consensus with SDOs yet
3	TOSCA Kubernetes Profile	<ul style="list-style-type: none"> Supports K8s and Openstack infra profiles Can work with any orchestrator with available toolsets and programmable interface 	<ul style="list-style-type: none"> Design time integration challenges Redundant parsers (existing + Puccini) Managing intermediate format and associated catalog operations
4	TOSCA + Helm chart as artifact	<ul style="list-style-type: none"> Logical extension to the current TOSCA-based VNF modeling with Helm as additional artifact 	<ul style="list-style-type: none"> Switching back and forth between TOSCA and Helm, across Helm charts might be overhead for existing Orchestration Solution Helm templating and dynamic value management, repo management overheads Additional tooling to be integrated in Orchestrator
5	Extended TOSCA types+ K8s Custom Resources/Operators	<ul style="list-style-type: none"> Minimum changes for the existing TOSCA-based orchestration Balanced approach to solve challenges of each 	<ul style="list-style-type: none"> Additional consensus and customizations Possibility of specializations if not standardized. which may lead to maintenance overhead



Alternatives Considered

- Using TOSCA or HEAT
- [Cloudify](#) or Terraform Orchestration
- [TOSCA on Kubernetes](#) by Tal Liron



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

