

CNCF TUG Progress

Lei WANG



White Papers

TUG white paper:

- [Prologue: Cloud Native Thinking for Telecommunications](#), final version is in progress

Cloud Native Thinking for Telecommunications

Basic tenets and end users' suggestions on how cloud native design and principles can be applied to mission critical telecommunications functions.

Table of Contents

- [1.1 Introduction](#)
- [1.2 What is Cloud Native?](#)
- [1.3 Defining Cloud Native Systems](#)
 - [1.3.1 What "loosely coupled" means in cloud native systems](#)
 - [1.3.2 Cloud native applications require orchestration](#)
 - [1.3.3 Infrastructure, Deployment and Configuration of Cloud Native Systems](#)
- [1.4 Cloud Native Network Functions](#)
- [1.5 Cloud Native for Telcos](#)
 - [1.5.1 A Brief History of Virtualisation for Telcos](#)
 - [1.5.2 Software Defined Networking And The Emergence of VNFs](#)
 - [1.5.3 Principles for Cloud Native Telco Infrastructure and Applications](#)
 - [1.5.4 Evolving the Stack From VNFs to CNFs](#)
- [1.6 Cloud Native for Telcos in Practice](#)
- [1.7 Conclusion](#)



White Papers – cont.

- WIP: Deploying Cloud Native Network Functions in a telecom service provider ecosystem ([google docs](#))

- CNCF TUG White Paper
- UPDATE 2019-06-19
- UPDATE 2019-06-14
- UPDATE 2019-06-07
- LIST OF CONTRIBUTORS
- TOC:
- **INTRO**
- CHAPTER OVERVIEW
 - Introduction [Ian Wells]
 - Characteristics of a telecom de...
 - Concept of a vanilla K8s-based ...
 - The ideal CNF
 - Best Practices
- Contributions
 - Definition of CNF [Tamas Zsiros...
 - Target Audience [Tamas Zsiros ...
 - License [Tamas Zsiros - Ready ...
 - ETSI NFV/MANO [Gergely Csata...

Outlining contents of individual chapters.

CHAPTER OVERVIEW

Please do not modify this chapter directly (except for adding your name). Comment or add suggestions instead.

Introduction [Ian Wells]

- Background & brief look back at telecom clouds (incl. typical implementations)
- Motivation [Jeffrey Saelens]
- Scope
- Definition of a CNF [Tamas Zsiros - done]
- Target audience [Tamas Zsiros- done]
- License [Tamas Zsiros - done]

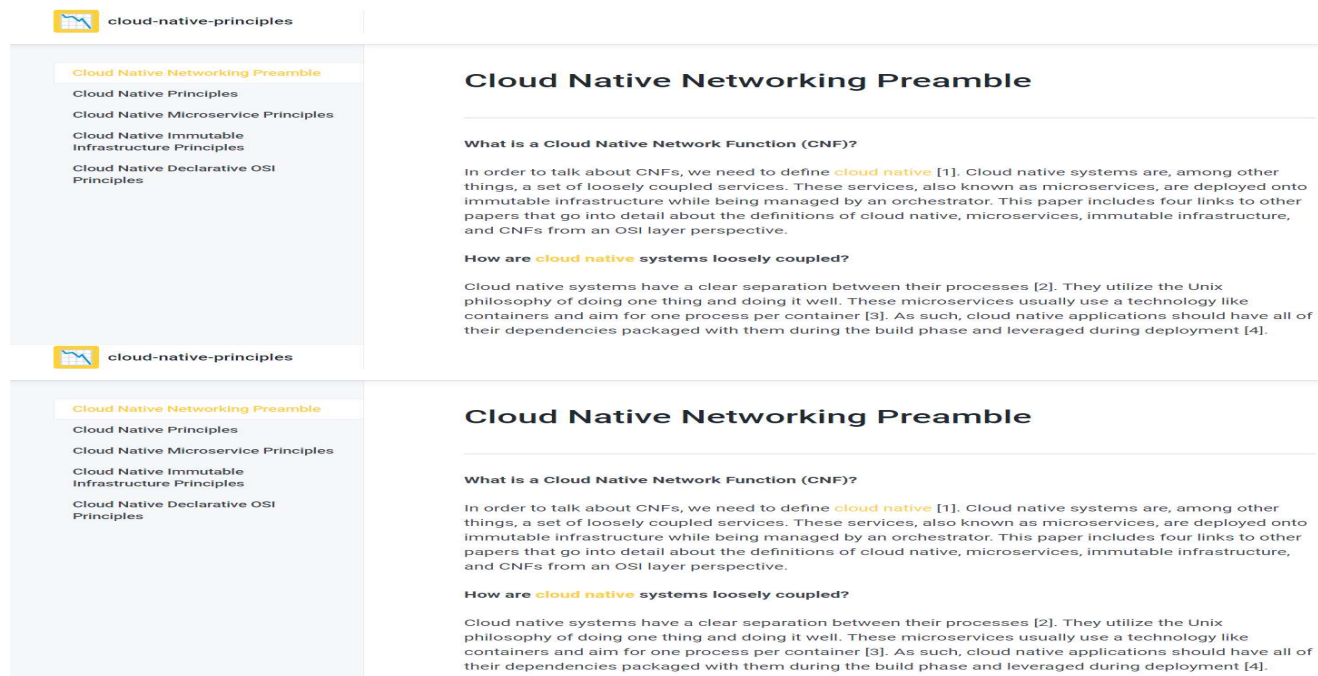
Characteristics of a telecom deployment

- Aim statement [Ian Wells]
- Scenarios: "on-premises" and "as a service"
- Hybrid cloud and cloud-bursting
- Homogenous and heterogenous workloads
- Similarities and differences compared to other cloud native applications (e.g. enterprise & web) [Gergely Csatari]
- Support for the ARM architecture [Lei Wang]
- Multi-vendor aspect
 - Vertical integration (cloud infra / CNF)



White Papers – cont.

- Cloud Native Principles gitbook ([github markdown](#))
- New paper regarding CNF Conformance



The image shows two identical screenshots of a web page from the 'cloud-native-principles' gitbook. The page title is 'Cloud Native Networking Preamble'. On the left side, there is a navigation menu with the following items: 'Cloud Native Networking Preamble' (highlighted), 'Cloud Native Principles', 'Cloud Native Microservice Principles', 'Cloud Native Immutable Infrastructure Principles', and 'Cloud Native Declarative OSI Principles'. The main content area has the following structure:

Cloud Native Networking Preamble

What is a Cloud Native Network Function (CNF)?

In order to talk about CNFs, we need to define **cloud native** [1]. Cloud native systems are, among other things, a set of loosely coupled services. These services, also known as microservices, are deployed onto immutable infrastructure while being managed by an orchestrator. This paper includes four links to other papers that go into detail about the definitions of cloud native, microservices, immutable infrastructure, and CNFs from an OSI layer perspective.

How are **cloud native** systems loosely coupled?

Cloud native systems have a clear separation between their processes [2]. They utilize the Unix philosophy of doing one thing and doing it well. These microservices usually use a technology like containers and aim for one process per container [3]. As such, cloud native applications should have all of their dependencies packaged with them during the build phase and leveraged during deployment [4].



CNF conformance

- The goal is to provide an open source test suite to demonstrate conformance and implementation of best practices for both open and closed source Cloud native Network Functions.
- The test suite will be categorized by following aspects:
 - **Compatibility** - CNFs should work with any Certified Kubernetes product and any CNI-compatible network that meet their functionality requirements.
 - **Statelessness** - The CNF's state should be stored in a custom resource definition or a separate database (e.g. etcd) rather than requiring local storage. The CNF should also be resilient to node failure.
 - **Security** - CNF containers should be isolated from one another and the host.
 - **Scalability** - CNFs should support horizontal scaling (across multiple machines) and vertical scaling (between sizes of machines).
 - **Configuration and Lifecycle** - The CNF's configuration and lifecycle should be managed in a declarative manner, using ConfigMaps, Operators, or other declarative interfaces.
 - **Observability** - CNFs should externalize their internal states in a way that supports metrics, tracing, and logging.
 - **Installable and Upgradeable** - CNFs should use standard, in-band deployment tools such as Helm (version 3) charts.
 - **Hardware Resources and Scheduling** - The CNF container should access all hardware and schedule to specific worker nodes by using a device plugin.
- <https://github.com/cncf/cnf-conformance/blob/master/TEST-CATEGORIES.md>



Platform conformance test

- Review and assess CNTT RA-2 platform requirements
 - provide feedback on how CNF Conformance can support RA-2 requirements for Platform Conformance Tests
- Assessment of K8s test coverage for CNTT RA2 requirements
 - Map CNTT RA-2 requirements to Conformance and e2e tests
- Create PoC scoring system for Platform Conformance (RA2) tests
 - to create an initial scoring system/points for the RA-2-based Platform Conformance tests
- Current work taking place at <https://github.com/cncf/cnf-conformance>



Open Questions

- CNI issue - enhance the CNI profiles to adopt more performance requirements
- Conformance profiles – less is better
- Privileged pods – least privilege and advanced networking
- MANO Integration – ONAP can be used
- CNF Modeling



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 referred 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



Thank you