CNCF TUG Progress

Lei WANG
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](#))
White Papers – cont.

- Cloud Native Principles gitbook ([github markdown](https://github.com/mdk-mllar/Cloud-Native-Principles))
- New paper regarding CNF Conformance

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 OSM 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.
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](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

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

- **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

<table>
<thead>
<tr>
<th>#</th>
<th>Approach</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Heat + Helm + TOSCA</td>
<td>• Accommodate VNF/CNF modeling requirements</td>
<td>• Customized approach for ONAP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No cross dependency, can independently describe the NF in respective modeling format of choice</td>
<td>• Complexity of managing multiple formats of descriptors requires additional skills</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Currently based on Helm 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Complexity to pass CNF instantiation inputs</td>
</tr>
<tr>
<td>2</td>
<td>Extended TOSCA Types</td>
<td>• Logical extension to the existing VNF modeling approach</td>
<td>• Require additional plugins to interpret and orchestrate for specific infrastructure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Supports multiple mechanisms for attaching the CNF-specific K8s resource artifacts</td>
<td>• No consensus with SDOs yet</td>
</tr>
<tr>
<td>3</td>
<td>TOSCA Kubernetes Profile</td>
<td>• Supports K8s and Openstack infra profiles</td>
<td>• Design time integration challenges</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Can work with any orchestrator with available toolsets and programmable interface</td>
<td>• Redundant parsers (existing + Puccini)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Managing intermediate format and associated catalog operations</td>
</tr>
<tr>
<td>4</td>
<td>TOSCA + Helm chart as artifact</td>
<td>• Logical extension to the current TOSCA-based VNF modeling with Helm as additional artifact</td>
<td>• Switching back and forth between TOSCA and Helm, across Helm charts might be overhead for existing Orchestration Solution</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Helm templating and dynamic value management, repo management overheads</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Additional tooling to be integrated in Orchestrator</td>
</tr>
<tr>
<td>5</td>
<td>Extended TOSCA types+ K8s Custom Resources/Operators</td>
<td>• Minimum changes for the existing TOSCA-based orchestration</td>
<td>• Additional consensus and customizations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Balanced approach to solve challenges of each</td>
<td>• Possibility of specializations if not standardized, which may lead to maintenance overhead</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank you