Building container-based NFV solutions with OPNFV, ONAP and VPP on Arm platform

Tina Tsou <tina.tsou@arm.com>
Trevor Tao <trevor.tao@arm.com>
Agenda

• Background
• Project Auto
• Compass4nfv on Arm
• Container4nfv on Arm
• OPNFV CI/CD for Arm
• VPP and Auto on Arm
Background
Projects in Linux Foundation

- ONAP
  - OOM
  - SDC
  - VID
  - SO
  - AAI/ESR
  - Policy Framework
  - CLAMP
  - DCAE
  - APP-C
  - SDN-C
  - AAF
  - MSB

- OPNFV
  - Armband
  - Compass4nf
  - Container4nf
  - Apex
  - FUEL
  - JOID

- OpenDaylight
  - Auto

- Kubernetes
  - FuncTest
  - YardStick

- FD.io
  - Releng

- DPDK
  - VPP

- Linux Foundation
  - Edge Cloud
  - Resiliency Improvements
  - Enterprise vCPE
  - Enterprise vCPE
  - Enterprise vCPE
  - Enterprise vCPE
  - Enterprise vCPE
OPNFV Projects with Arm

• Project Armband
  • The purpose of this project is simply to integrate and test all aspects of OPNFV releases on ARM-based servers.

• Project Yardstick
  • A test framework with test cases and test stimuli to enable NFV-I performance verification

• Project Auto
  • This project focuses on ONAP component integration and verification with OPNFV reference platforms/scenarios

• Project Compass4nfv
  • An installer project based on open source project Compass, which provides automated deployment and management of OpenStack and other distributed systems
  • Ansible is used by default.
  • Our main installer for OPNFV Container4NFV project

• Project Container4NFV
  • Provide a container full-stack environment where VNF can run, including data plane VNF and control plane VNF. Let the platform support container and virtualization technology. Collect requirement for containerized NFVs.
  • Previously named as OpenRetriever
  • What are we focusing on for building Arm’s containerized NFV infrastructure now
Project Auto
OPNFV Projects – Auto Project

This project focuses on **ONAP component integration and verification with OPNFV reference platforms/scenarios**, through primarily a post-install process in order to avoid impact to OPNFV installer projects.

Related Project Opera: developing OPNFV-installer supported scenarios that can deploy and verify ONAP as a whole.
Auto (ONAP-Automated OPNFV)

Validate ONAP (Open Network Automation Platform) as NFV Orchestrator and VNF Manager in OPNFV ecosystem; Auto project home page

Show added value of:

• Automation using closed loops (defined in CLAMP), policies (defined in Policy Framework), and DCAE (real-time monitoring, execution of closed loops and policies; also alarm correlation)

• Design-time portal-based (as well as API-based control) streamlined VNF lifecycle management: Onboarding (with SDC, to define VSPs with VLMs, and end-to-end Services), Deployment (with VID and MSO), and Operations (with persistent inventory data in AAI)

CLAMP Closed-Loop Automation Management Platform
DCAE Data Collection Analytics and Events

SDC Service Design and Creation
VSP Vendor Software Product
VLM Vendor License Model
VID Virtual Infrastructure Deployment
MSO Master Service Orchestration
AAI Active and Available Inventory
Auto (ONAP-Automated OPNFV): use cases

Three specific use cases for Auto:

1. **Edge Cloud** - autonomy of Edge Cloud management
   - Autonomy enabled by **systematic** catalog-based VNF deployment through SDC/VID/MSO, **automated** monitoring and management through MSO, DCAE, CLAMP, Policies, and an array of controllers

2. **Resilience** - improvements through ONAP
   - Failure recovery time reduction with ONAP, thanks to automated monitoring and management

3. **Enterprise vCPE** - ensure high performance, enterprise-grade vCPEs in Edge Cloud
   - Rely on **enterprise-grade vCPE VNFs**, properly onboarded (tested, certified, approved: multiple Roles in onboarding process), and properly monitored and managed for **performance assurance** (SLAs and High Availability: redundancy, recovery)
Auto Project – **ONAP on Kubernetes Architecture**

**Kubernetes Client**
Could be a k8s node or collocated server

- (2) createAll.bash
- (1) Onap-parameters.yaml
- kubectl client

**OpenStack**
Rackspace or vanilla

- DCAE VM
- vFirewall VM

**Kubernetes Host(s)**

- config container (stopped)
- kubectl server

**Rancher Container Management System (optional)**

- rancher agent container
- rancher server container

**DCAE VM**
- docker container (1)

**vFirewall VM**
- docker containers (70+16 filebeat)
Auto (ONAP-Automated OPNFV): status

Test cases:

- Top-down definition (methodology like FuncTest)
- For initial Auto project purposes, test environment may not need to be as complete and systematic as YardStick, FuncTest, or Robot; later, alignment with FuncTest and YardStick will be sought
- Script development environment target: Python3, maximize similarity between 3 use cases

Dependency on DCAE, deployment on K8S (hopefully from OOM, for ONAP Beijing release, end March 2018; external dependency: containerized Cloudify)

Infrastructure deployment: two CPU architectures (x86 and Arm)

- 2 pods (6 physical servers each, one x86, one Arm) at UNH IOL (University New Hampshire, Interoperability Lab)
- x86 pod: ONAP on Kubernetes on Bare Metal; no DCAE yet;
- Arm pod: target is ONAP on Kubernetes on OpenStack VMs; also no DCAE yet;
- installation tools:
  - x86: (TBC)
  - Arm: MCP for OpenStack, new tests for K8S (existing methods don’t work)
- Note: goal is to have ONAP on Kubernetes in both pods
Use Case Specification and Development Methodology

1. Test Overall Objectives
   - DependsOn: Test Strategy

2. Test Strategy
   - DependsOn: Test Design Documentation

3. Test Design Documentation
   - DependsOn: Test Analysis, Conclusions

4. Test Script Implementation
   - Enables: Test Script Language (Python, bash, ...); Comments in script capture the test documentation

5. Test Runner (Calling script)
   - Executes: Test Script Implementation

6. Test Analysis, Conclusions
   - Implements: Test Analysis, Conclusions

Use Case Description

- Follows (sequence)
- "section" or "chapter" of a Test; could be just 1 UC

Pre-test State Description
- Description of required configuration, environment variables, ...

Action (Test Step)
- e.g.: setup, start, use, stop, (functions in a script language); execute planned VNF LC events

Assertion Step
- Evaluation of current state, application of assertion criteria, determination of Pass/Fail result

Post-test State Capture
- Capture/log of significant state data (environment variables, ...)

Cleanup Step
- Function to return system to its Pre-test State

Physical Resources (unless Test as a Service)
- Physical servers (CPU, RAM), disks (HD, SSD), OS; pod organization;

Cloud & Virtual Resources
- Manager: Openstack, K8S, AWS, ...
- Resources: compute (OS), storage, network, ...; VMs or Containers;

VNF Manager
- ONAP, Cloudify, Tacker, ...

VNFs
- vXYZ (CPE, FW, Switch, HSS, V/S/P-SCF, ...);
- Source (OPNFV, ClearWater, ETSI, ...);
- Format (JSON, TOSCA, ...)

VNF Lifecycle Events
- Onboard, Remove, Deploy/Activate, Terminate, Scale-In/Out, Monitor, ...

Environment Description
- DependsOn: Environment Variant

Environment Variant
- IsVariantOf: Test Specific Objectives

Test Specific Objectives
- DependsOn: Environment Description

Follows (sequence)

1 2 3 4 5 6

<table>
<thead>
<tr>
<th>IsVariantOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

IsVariantOf:

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
</table>

DependsOn:

<table>
<thead>
<tr>
<th>Test Overall Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Strategy</td>
</tr>
<tr>
<td>Test Design Documentation</td>
</tr>
<tr>
<td>Test Analysis, Conclusions</td>
</tr>
<tr>
<td>Test Script Implementation</td>
</tr>
<tr>
<td>Test Runner (Calling script)</td>
</tr>
<tr>
<td>Use Case Description</td>
</tr>
<tr>
<td>Pre-test State Description</td>
</tr>
<tr>
<td>Action (Test Step)</td>
</tr>
<tr>
<td>Assertion Step</td>
</tr>
<tr>
<td>Post-test State Capture</td>
</tr>
<tr>
<td>Cleanup Step</td>
</tr>
<tr>
<td>Physical Resources</td>
</tr>
<tr>
<td>Cloud &amp; Virtual Resources</td>
</tr>
<tr>
<td>VNF Manager</td>
</tr>
<tr>
<td>VNFs</td>
</tr>
<tr>
<td>VNF Lifecycle Events</td>
</tr>
</tbody>
</table>

Follows (sequence)

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;section&quot; or &quot;chapter&quot; of a Test; could be just 1 UC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description of required configuration, environment variables, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g.: setup, start, use, stop, (functions in a script language); execute planned VNF LC events</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation of current state, application of assertion criteria, determination of Pass/Fail result</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture/log of significant state data (environment variables, ...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function to return system to its Pre-test State</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical servers (CPU, RAM), disks (HD, SSD), OS; pod organization;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical servers (CPU, RAM), disks (HD, SSD), OS; pod organization;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager: Openstack, K8S, AWS, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manager: Openstack, K8S, AWS, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources: compute (OS), storage, network, ...; VMs or Containers;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources: compute (OS), storage, network, ...; VMs or Containers;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONAP, Cloudify, Tacker, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONAP, Cloudify, Tacker, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vXYZ (CPE, FW, Switch, HSS, V/S/P-SCF, ...); Source (OPNFV, ClearWater, ETSI, ...); Format (JSON, TOSCA, ...);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vXYZ (CPE, FW, Switch, HSS, V/S/P-SCF, ...); Source (OPNFV, ClearWater, ETSI, ...); Format (JSON, TOSCA, ...);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onboard, Remove, Deploy/Activate, Terminate, Scale-In/Out, Monitor, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Follows (sequence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onboard, Remove, Deploy/Activate, Terminate, Scale-In/Out, Monitor, ...</td>
</tr>
</tbody>
</table>

© 2018 Arm Limited
Compass4NFV on Arm
Project Compass4nfv for Kubernetes -- Installer now Arm Support

Compass4NFV on Arm (Yibo Cai, Di Xu):
What We Have Done:

- Ported Compass4NFV docker images for AArch64 and uploaded to dockerhub Linaro repo.
  - Compass-tasks-k8s
  - Compass-deck
  - Compass-mq
  - Compass-cobbler
  - Compass-db
- Supported AArch64 baremetal deployment (CentOS7)
- Supported deploying Kubernetes cluster on AArch64 virtual and bare-metal nodes.

Compass4NFV repo

Our ‘F’ release scenarios for Container4NFV would be based on the work in Compass4NFV
Container4NFV on Arm
ONAP supports multiple VNF environments by integrating with multiple VIMs, VNFMs, SDN Controllers, and even legacy equipment.

Kubernetes as COE

Multus plugin for Kubernetes as CNI

Flannel/DPDK/Vhost user CNI plugins integrated

Ref: https://wiki.opnfv.org/pages/viewpage.action?spaceKey=OpenRetriever&title=Container%27s+Architecture+for+Cloud+Native+NFV
Container Networking Acceleration with DPDK

K8s MASTER

API SERVER
etcd
SCHEDULER
NODE1
NODE2
NODE3
CONTROLLER
MANAGER

K8s Node

KUBELET
KUBE-PROXY
Flannel CNI

Containers

Docker VNF

K8s Node

KUBELET
KUBE-PROXY
Multus CNI
Flannel CNI
SR-IOV CNI

Containers

Docker VNF

Kernel

docker0 bridge
Flannel0 bridge

VFIO/UIO

Kernel

PF
VF
VF

Flannel0 bridge

OPNFV CI/CD for Arm
OPNFV ‘F’ Release Scenarios for Arm

Fraser Scenario Status:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Installer</th>
<th>Owner</th>
<th>Jenkins Job Created (Y/N)</th>
<th>Intent to release 6.0 (Y/N) (1,2)</th>
<th>Intent to release 6.1 (Y/N) (1,2)</th>
<th>Intent to release 6.2 (Y/N) (1,2)</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8-multus-nofeature-noha</td>
<td>Compass</td>
<td>@ Trevor Tao</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>k8-sriov-cni-nofeature-noha</td>
<td>Compass</td>
<td>@ Phoenix Striker</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Jenkins projects:

- **k8-multus-nofeature-noha**
  - k8-multus-nofeature-noha with Compass
- **k8-sriov-cni-nofeature-noha-arm**
  - k8-sriov-cni-nofeature-noha with Compass

Contributor: Yibo Cai, Di Xu @arm
VPP and Auto on Arm
Linking VPP and Auto in Linux Foundation, for Arm architecture

- **VPP data-plane optimized VNFs** (FD.io, DPDK), validated for Arm architecture
- the VPP-optimized VNFs are **containerized** (Kubernetes/Docker)
- the VPP-optimized VNFs can be **managed by ONAP** (onboarded and deployed as end-to-end Services)
- ONAP itself (each component: SDC, CLAMP, DCAE, etc.) is also **containerized** and validated for Arm architecture
- ONAP installer (OOM) and ONAP components leverage OPNFV installers such as Compass4nfv and Container4nfv
- **Auto use cases** (current three, and future) will be tested on these VPP-optimized VNFs
- Auto use cases will eventually align with OPNFV test frameworks (**FuncTest**, **YardStick**)
- overall governance of Test-Frameworks/Auto-tests/ONAP/VPP-optimized-VNFs on Arm architecture: **Armband**
ONAP installation target on Arm pod: ONAP/K8S/VM

- ONAP on K8S
- Container-ized Test Case

Don’t rely on QEMU: instead, ensure binary-level compatibility
For all:
- runtime components (ONAP modules, K8S, Docker, OpenStack)
- offline installers (Cloudify, Rancher)
- VNFs
- possibly also Test Case implementations (scripts)

if implemented in interpreted languages:
- Ensure runtime environments are compiled for Arm architecture
if implemented in compiled languages:
- Ensure source codes are compiled for Arm architecture

K8S/OpenStack: just to avoid tearing down current OpenStack installation on resource-limited pod (6 servers). Once tested on VMs, ONAP/K8S should also work on bare metal.

Ultimate VNF installation: via ONAP Portal (onboard, use in service, deploy), via SDC, VID, SO; setup Policies, closed-loops; monitor with DCAE

Ultimate ONAP install method: OOM (ONAP Operations Manager)

Cloud VM supported by ONAP:
- Openstack
- Microsoft Azure
- Amazon AWS
- Google GCD
- VMware VIO

Cloud VIM supported by ONAP:
- Openstack
- Microsoft Azure
- Amazon AWS
- Google GCD
- VMware VIO

VM-based (nova) VNF

Guest OS

VM (nova instance)

OpenStack
- K8S Pod
- Container
- Container (app + libs)
- Container (app + libs)

COE (Container Orchestration Engine) (Kubernetes)

K8S Cluster

K8S Pod

Container Runtime (Docker)

VM (nova instance)

Guest OS

VM (nova instance)

Guest OS

Don’t rely on QEMU: instead, ensure binary-level compatibility

For all:
- runtime components (ONAP modules, K8S, Docker, OpenStack)
- offline installers (Cloudify, Rancher)
- VNFs
- possibly also Test Case implementations (scripts)

if implemented in interpreted languages:
- Ensure runtime environments are compiled for Arm architecture
if implemented in compiled languages:
- Ensure source codes are compiled for Arm architecture

K8S/OpenStack: just to avoid tearing down current OpenStack installation on resource-limited pod (6 servers). Once tested on VMs, ONAP/K8S should also work on bare metal.

Ultimate VNF installation: via ONAP Portal (onboard, use in service, deploy), via SDC, VID, SO; setup Policies, closed-loops; monitor with DCAE

Ultimate ONAP install method: OOM (ONAP Operations Manager)

Cloud VM supported by ONAP:
- Openstack
- Microsoft Azure
- Amazon AWS
- Google GCD
- VMware VIO

Cloud VIM supported by ONAP:
- Openstack
- Microsoft Azure
- Amazon AWS
- Google GCD
- VMware VIO
Thank You!
Danke!
Merci!
谢谢!
ありがとうございます!
Gracias!
Kiitos!
감사합니다
धन्यवाद
Backups