VNF Life Cycle Management with EMCO and KubeVirt

Kuralamudhan Ramakrishnan,
kuralamudhan.ramakrishnan@intel.com
Todd Malsbary, todd.malsbary@intel.com
01/13/2022
Cloud Native Evolution

- **Today**
  - Virtualize Physical Appliances
  - Monolith
  - VM

- **Emerging**
  - Containers
  - Containers Baremetal Deployment Model
  - Automated; Secure; Flexible; Performant; Resilient;
  - Micro-services

**Hindsight** What Happened
Cloud Native Evolution

Today

Monolith

VM

Monolith & Container coexist

Hindsight
What Happened

Automated; Secure; Flexible; Performant; Resilient;

Containers

Micro-services

Emerging
# Integrated Cloud Native stack

## Use cases/Apps
- **PaaS (MEX)**
  - Multi-Tenant Mgr
  - Edge Label Mgr
  - Slice Mgr
  - K8S HPA
  - MC – K8S Plugin Service
    (Instantiation, Day0, Day2 config)

## CNFs
- **Analytics framework**
  - Data Lake
  - Training
  - Model Repo
  - Messaging

## VNFs
- **Host Operating System**
  - Ubuntu

## Hardware
- S1
- S2
- S3

## EmCO
- **K8S App Components**
  - ISTIO
  - KATA
  - CollectD
  - OpenEBS
  - KubeVirt
  - Rook
  - MetalLB
  - Flannel, OVN
  - Prometheus
  - knative
  - Inferencing

## Kubernetes
- **NFV Specific components**
  - Multus
  - PMEM CSI
  - DPK
  - NFD
  - Numa Mgr
  - OVS DPDK

## NFV SDN specific components
- **Nodus**
- SFC Mgr
- Network Mgr
- Route Mgr

## Host Operating System
- Tuned for eBPF and XDP

## Local Controller
- **Infrastructure Provisioning & Configuration**
  - **ClusterAPI**
  - Fluxcd
  - Metal3
  - Ironic

## Upstream communities:
- Akraino LF Edge, CNCF projects, LF Networking, FD.IO, DPDK, Linux, OVS, Many ASF projects, Intel Open Source

- **Existing Open source, Major enhancement, work with upstream**
- **Open source projects with ICN Enhancement**
- **Open source projects in ICN roadmap for the inclusion**

## Edge location
- Central/Regional location
  - Multi-Site scheduler

## Use cases/Apps
- **Use cases/Apps**
  - Infrastructure Provisioning & Configuration
  - Cluster API
  - PaaS (MEX)
  - Analytics framework
  - VNFs
  - CNFs

## Edge location
- Edge location
  - K8S App Components
  - Kubernetes
  - Host Operating System
  - Hardware

## https://github.com/akraino-edge-stack/icn
VNF deployment in ICN Stack

Deploy & Manage Apps - EMCO
- Cluster Management
- Scheduler
- Placement Controller
- Resource Synchronizer

Distributed Applications
- SD-EWAN CNF
- VNFs
- K8s plugins add-ons
- K8s
- Nodus
- K8s plugins add-ons
- Linux
- Private Cloud K8s cluster
- Possibly in hundreds

ICN
- SD-EWAN CNF
- VNFs
- K8s plugins add-ons
- K8s
- Nodus
- K8s plugins add-ons
- Linux
- Public Cloud K8s Cluster
- Edge K8s Cluster

Infrastructure Orchestrator
- SD-EWAN traffic Hub

Internet

https://github.com/akraino-edge-stack/icn
KubeVirt – What and Why

• KubeVirt provides the components needed to build, modify, and deploy virtual machines in Kubernetes

• It targets teams that already have or are wanting to adopt Kubernetes and have existing virtual-machine based workloads that cannot be easily containerized

• The existing virtual-machine based workloads can continue to be used while transitioning incrementally to containerized workloads
KubeVirt Architecture
Differences Between Container and VNF Deployments
Deployment vs. VirtualMachine

- Deployments and VirtualMachine resources describe a template for creating instances of each: Pods and VirtualMachineInstances respectively.
- Pods describe running containers, and VirtualMachineInstances describe running VMs.
VirtualMachine CR Overview

- Spec template describes the VirtualMachineInstance, like Pod spec
- Domain contains the guest parts of the VM
- Networks connects the guest interfaces to the host networks
- Volumes populates the guest disks from host data

```yaml
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: vm
spec:
template:
  metadata: ...
  spec:
    domain:
      devices:
        disks:
          - name: containerdisk
disk:
            bus: virtio
          - name: cloudinitdisk
disk:
            bus: virtio
      networks:
        - name: default
          pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: ...
          - name: cloudinitdisk
            cloudInitNoCloud: ...
```
Images

- Pods specify container images holding the application and its dependencies
- VMIs provide similar functionality by specifying the guest disks and host volumes

```yaml
spec:
template:
spec:
domain:
devices:
disks:
  - name: containerdisk
disk:
    bus: virtio
volumes:
  - name: containerdisk
    containerDisk:
      image: integratedcloudnative/ubuntu:16.04
```
Environment

- Pods can accept environment values to be set in the running container inline or from ConfigMaps

- The comparable mechanism with VMIs is to use cloud-init data sources

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          disks:
            - name: cloudinitdisk
disk:
              bus: virtio

volumes:
  - name: cloudinitdisk
cloudInitNoCloud:
  networkData: |
v    version: 2
...

userData: |
  #cloud-config
  ssh_pwauth: True
  ...
```
Networks (1/2)

- Pods using only the default Pod network generally require no special configuration
- A VMI requires describing the guest and host parts

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              bridge: {}
        networks:
          - name: default
            pod: {}
```
Using multiple networks is accomplished by using Multus.

With Pods, the `k8s.v1.cni.cncf.io/networks` annotation is added to the spec containing the name of `NetworkAttachmentDefinition` describing the additional network.

With VMIs, a guest and host part are again added.

KubeVirt translates the VMI spec for the additional network to a `k8s.v1.cni.cncf.io/networks` annotation on the KubeVirt managed Pod.

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: additional-network
          bridge: {}
        networks:
          - name: additional-network
            multus:
              networkName: additional-network-attachment-definition-name
```
admin@fw0-firewall:~$ ip a
2: enp1s0: ...
   link/ether 2a:81:33:d6:c6:a2 ...
   inet 10.244.65.57/24 brd 10.244.65.255 ...
3: eth1: ...
   link/ether 52:57:2b:7b:e4:27 ...
   inet 192.168.10.3/24 brd 192.168.10.255 ...
bash-5.0# ip a
3: eth0-nic@if80: ... master k6t-eth0 ...
   link/ether 2a:81:33:f7:6a:fe ...
5: net1-nic@if81: ... master k6t-net1 ...
   link/ether 52:57:2b:77:54:6a ...
10: eth0: ...
   link/ether c2:d9:9f:41:01:e9 ...
11: k6t-eth0: ...
   link/ether 2a:81:33:f7:6a:fe ...
   inet 169.254.75.10/32 ...
12: tap0: ... master k6t-eth0 ...
   link/ether ce:4d:7c:23:35:9d ...
13: net1: ...
   link/ether ae:8e:8b:bd:8c:41 ...
14: k6t-net1: ...
   link/ether 52:57:2b:77:54:6a ...
   inet 169.254.75.11/32 ...
15: tap1: ... master k6t-net1 ...
   link/ether 72:54:79:0b:b0:f7 ...
KubeVirt Network Architecture (3/3)

root@machine-2:~$ ip a
81: 6ff46d99091fb91@if5: ...
   link/ether 2a:43:76:9b:20:dc ...
82: 6ff46d99091fb92@if7: ...
   link/ether ca:14:f3:a9:5a:8b ...
Experiences and Lessons Learned Enabling a Commercial VNF with KubeVirt
Persistent Volumes (1/2)

- A licensed VM will require that the disk image be persistent, unlike the ephemeral container disk approach shown earlier
- The first step is set the host volume in the VM spec to the name of a persistent volume claim

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          disks:
            - name: pvcdisk
              disk:
                bus: virtio
          volumes:
            - name: pvcdisk
              persistentVolumeClaim:
                claimName: fedora-pvc
```
Persistent Volumes (2/2)

- The next step uses the Containerized Data Importer (CDI) project of KubeVirt to import a QEMU disk image into a persistent volume
- When this resource is created, the CDI controllers will fetch the image and create the PV
- Lastly, CDI requires that the K8s cluster is configured with a dynamic storage provisioner
- In our case, we used OpenEBS with the cStor backend
- For further reference, refer to the KubeVirt blog entry Building a VM Image Repository

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: "fedora-pvc"
labels:
  app: containerized-data-importer
annotations:
  cdi.kubevirt.io/storage.import.endpoint: "https://.../Fedora.qcow2"
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
      storageClassName: "cstor-csi-disk"
```

Enabling High Performance

CPU Pinning

• First, enable the `CPUManager` feature gate in KubeVirt's configuration

• Second, request a number of CPUs and specify dedicated CPU placement in the VMI spec

• To further improve latency the emulator event loop can also be specified to run in a dedicated vCPU

```yaml
apiVersion: kubevirt.io/v1
kind: KubeVirt
metadata:
  name: kubevirt
  namespace: kubevirt
spec:
  configuration:
    developerConfiguration:
      featureGates:
      - CPUManager
---
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
spec:
  template:
    spec:
      domain:
        cpu:
          dedicatedCpuPlacement: true
          isolateEmulatorThread: true
      resources:
        requests:
          cpu: 8
        limits:
          cpu: 8
```
Enabling High Performance

AES Feature Detection

- VMIs may also request that they only be scheduled on Nodes supporting specific CPU features
- For example, encryption performance may be improved on Nodes with AES-NI instruction support

```
spec:
template:
  spec:
    domain:
      cpu:
        features:
          - name: "aes"
            policy: "require"
```
Enabling High Performance

SR-IOV (1/4)

• Using SR-IOV capable NICs in a VM is similar to using multiple networks shown earlier

• However instead of instructing KubeVirt to use type bridge in the guest specification, use sriov instead

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: ingress
              sriov: {}
        networks:
          - name: ingress
            multus:
              networkName: sriov-intel
```
SR-IOV (2/4)

- The SR-IOV Network Operator is deployed to create the sriov-intel NetworkAttachmentDefinition

- The operator's SriovNetworkNodePolicy CR selects and configures SR-IOV drivers

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-xl710
spec:
  deviceType: vfio-pci
  nicSelector:
    deviceID: "1583"
    vendor: "8086"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
    feature.node.kubernetes.io/pci-0200_8086_1583.present: "true"
  numVFs: 8
  resourceName: intel_sriov_nic
```
Enabling High Performance

SR-IOV (3/4)
- The operator's SriovNetwork CR then creates a NetworkAttachmentDefinition composed of the drivers configured from the policy

```yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-intel
spec:
  ipam:
  - type: "host-local",
    subnet: "10.56.206.0/24",
    routes: [{
      dst: "0.0.0.0/0"
    }],
    gateway: "10.56.206.1"

networkNamespace: default
resourceName: intel_sriov_nic
```
SR-IOV (4/4)

- The resulting NetworkAttachmentDefinition is what will be referenced in the VMI *networks* section.

```yaml
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/intel_sriov_nic
  name: sriov-intel
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "sriov-intel",
    "type": "sriov",
    "vlan": 0,
    "vlanQoS": 0,
    "ipam": {
      "type": "host-local",
      "subnet": "10.56.206.0/24",
      "routes": [{"dst": "0.0.0.0/0"},
                  {"dst": "0.0.0.0/0"}],
      "gateway": "10.56.206.1"
    }
  }'
```
Interface Naming

- In one case we encountered, a VM image required specific interface names.
- While Multus allows for naming the interfaces used in additional networks, KubeVirt does not currently expose this in the VM specification.
- Instead, each additional network is named net1, net2, etc.
- The workaround employed is to provide a MAC address in the guest devices section and use the networkData of cloud-init to match the MAC address and set the interface name.

```yaml
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              bridge: {}
            - name: additional-network
              macAddress: ee:f0:75:e0:b6:26
              bridge: {}
        volumes:
          - name: cloudinitdisk
            cloudInitNoCloud:
              networkData:
                version: 2
                ethernets:
                  enp1s0:
                    dhcp4: true
                  eth1:
                    match:
                      macaddress: "ee:f0:75:e0:b6:26"
                    set-name: eth1
                    dhcp4: true
```
How We Have Used EMCO
The first step to using KubeVirt with EMCO is to package the KubeVirt resources into a Helm chart.

For the initial effort creating a chart containing only the `VirtualMachine` YAML with no templated values may be sufficient.

For more complicated use cases, EMCO provides mechanisms for overriding Helm chart values and patching resources at various points:

- Profiles
- Deployment intent groups
- **Generic action controller**

Note that both virtual and containerized applications may be mixed freely together in the definition of an EMCO composite application.
**Network Controller Intent (1/2)**

- EMCO's network controllers provide the ability to define additional networks and the application interfaces connected to those networks.
- The Nodus project provides the mechanisms to create the networks and interfaces.
- References:
  - [Network Configuration Management](#)
  - [OVN Action Controller](#)
• When using network controller intents together with the interface naming workaround shown earlier, some care is needed in providing the interface name

• Recall that *net1*, *net2*, etc. are the interface names created by Multus

• These names must be provided to EMCO, **not** the renamed guest interfaces (e.g. *eth1*, *eth2*, etc.)
Multi-cluster Deployment of VMs

- No special consideration is needed to use the multi-cluster orchestration of EMCO together with VNFs
- Define the logical cloud and deployment intents as you would for any EMCO project
Changes Made to EMCO
Deploying VirtualMachines

- Prior to this work, EMCO made some assumptions that the application contained only Deployment resources.
- Those assumptions were removed to support KubeVirt's custom resources, i.e. VirtualMachine.
Network Controller Additions (1/2)

- Changes were required in Nodus to better support the use of Multus by KubeVirt
- Previously Nodus expected only one invocation of the CNI during Pod creation; this resulted in attempting to create the same NIC multiple times and failing
- Nodus now correctly handles multiple invocations, one per additional network
An additional change was made to EMCO to automatically create NetworkAttachmentDefinitions of the *provider-networks* and *networks now* required by Nodus.

The NetworkAttachmentDefinition contains the information needed to determine which interface to create in the Pod’s namespace.

An example from the vFW project is shown to the right.
VNF Life Cycle Management with EMCO and KubeVirt

Kuralamudhan Ramakrishnan, kuralamudhan.ramakrishnan@intel.com
Todd Malsbary, todd.malsbary@intel.com

01/13/2022