LF NETWORKING

LFN Developer & Testing Forum

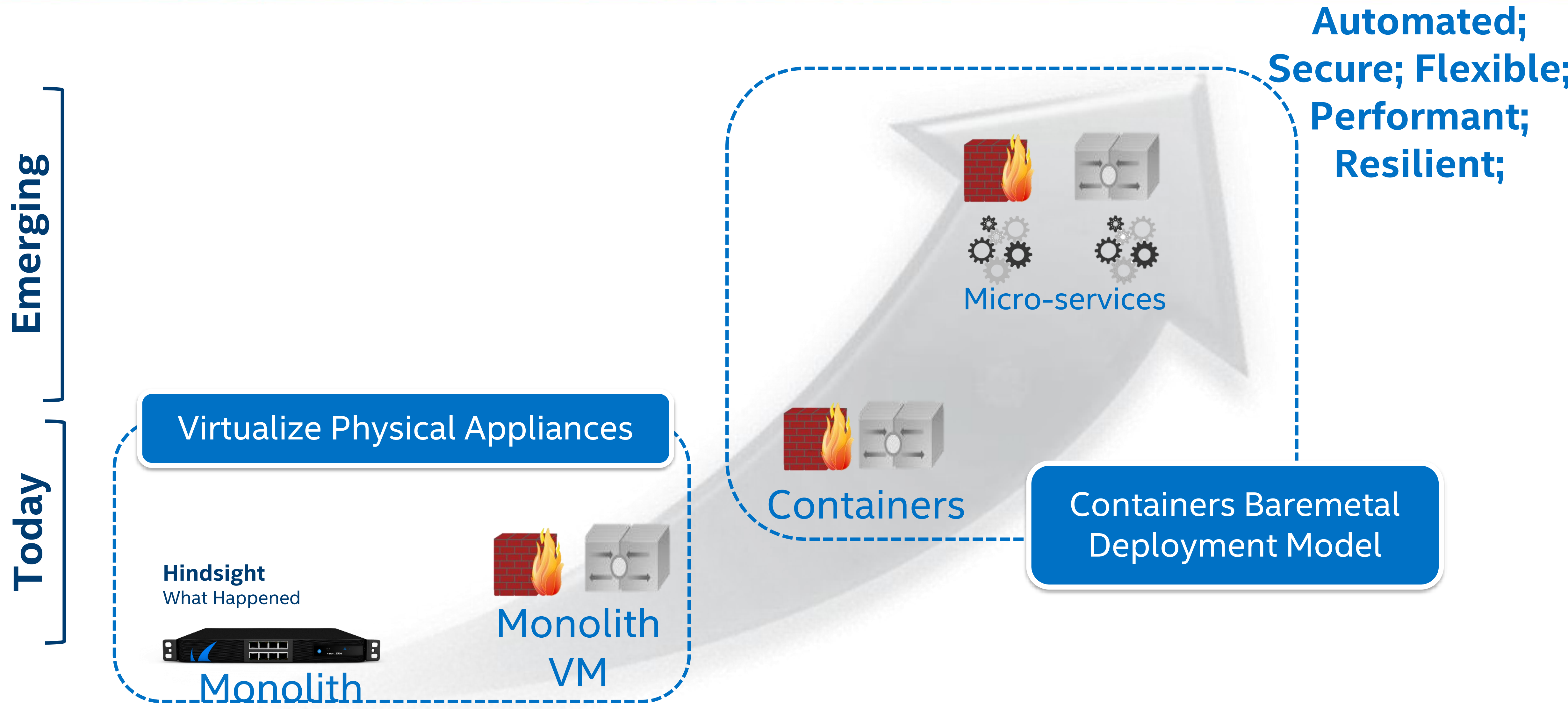
VNF Life Cycle Management with EMCO and KubeVirt

Kuralamudhan Ramakrishnan, kuralamudhan.ramakrishnan@intel.com

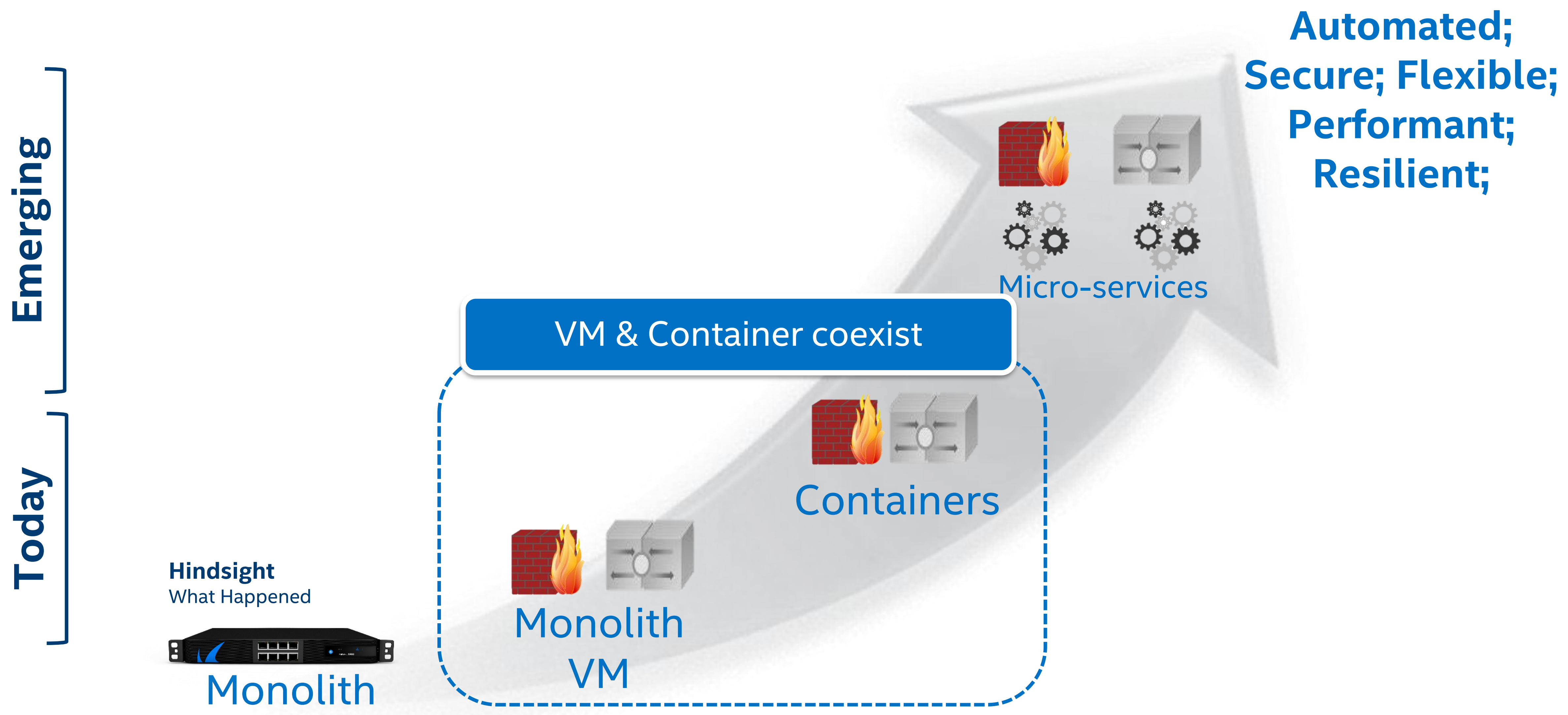
Todd Malsbary, todd.malsbary@intel.com

01/13/2022

Cloud Native Evolution

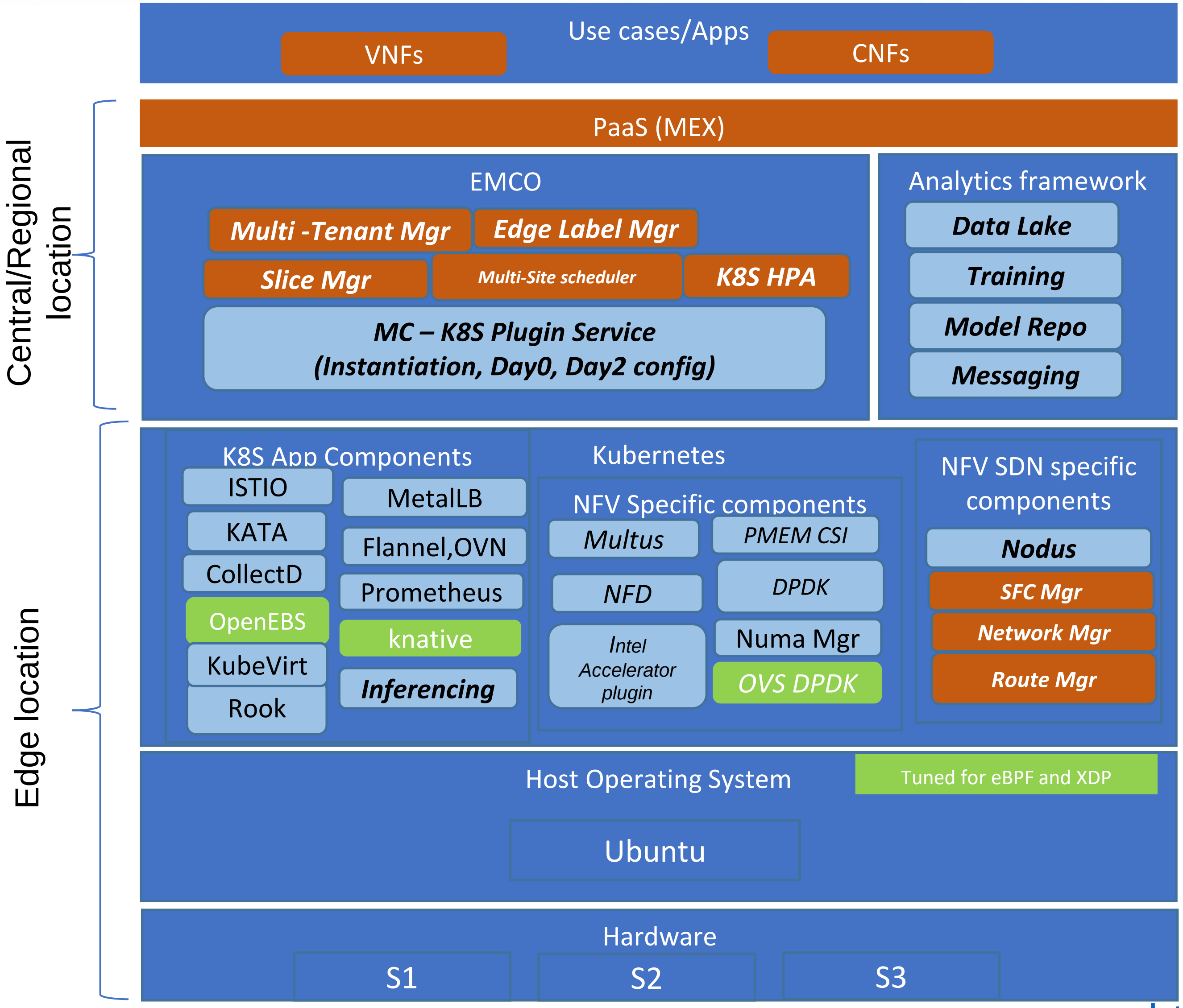


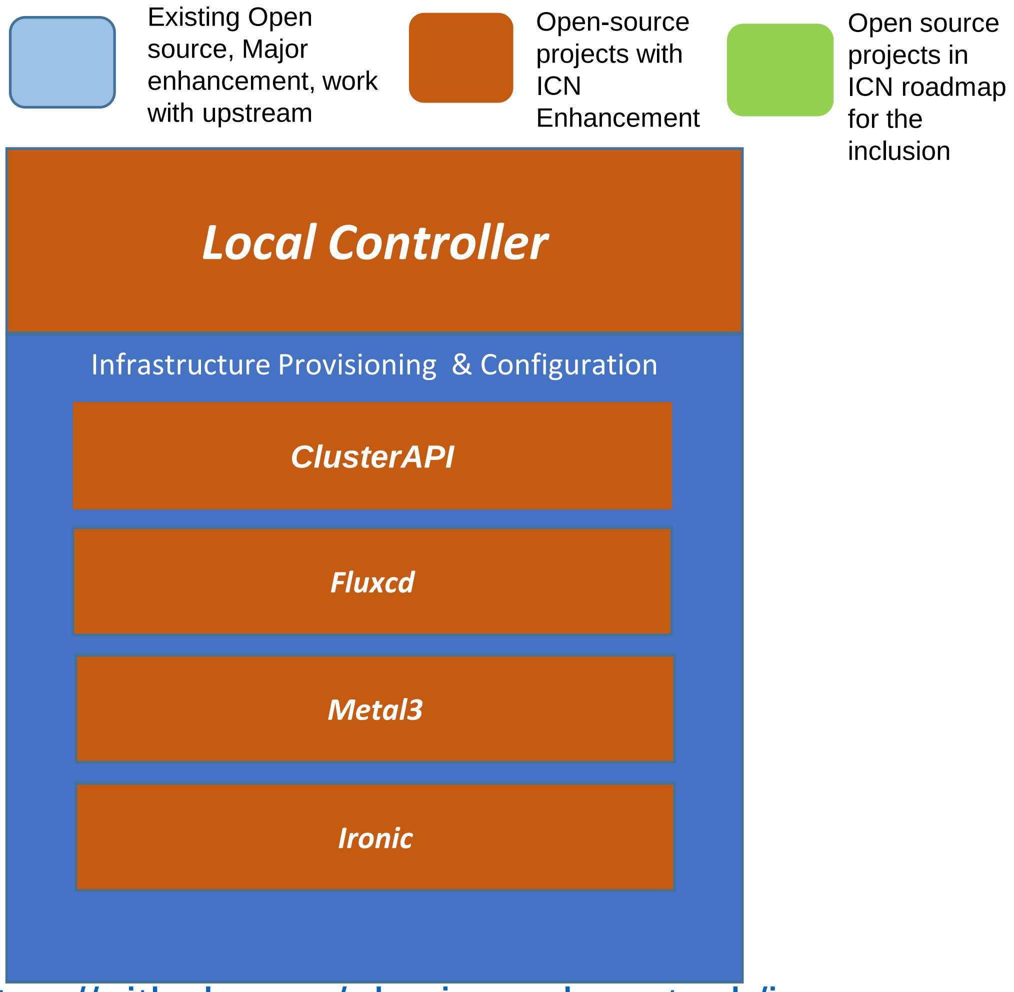
Cloud Native Evolution



Integrated Cloud Native stack

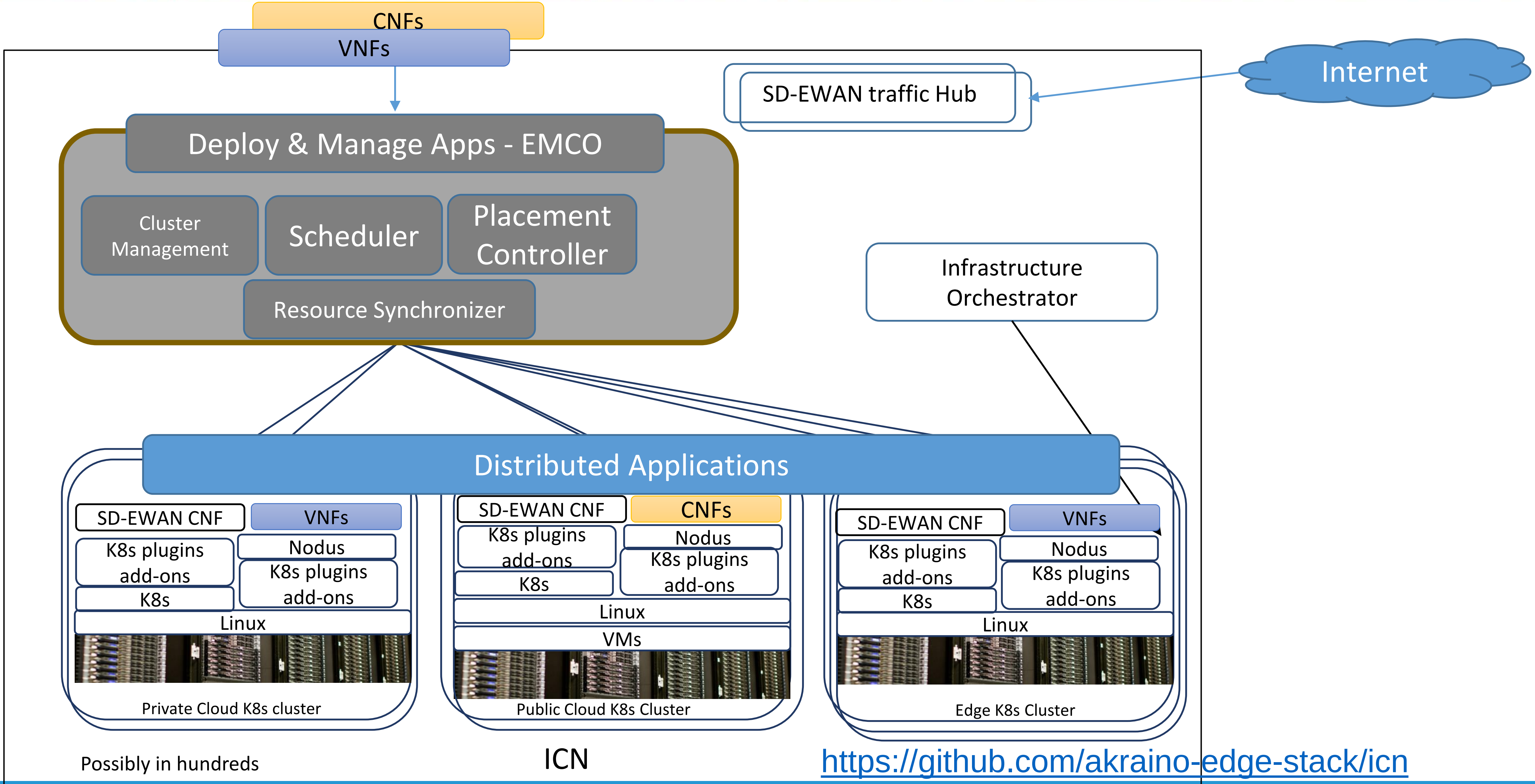
Upstream communities: Akraino LF Edge, CNCF projects ,LF Networking, FD.IO, DPDK, Linux, OVS, Many ASF projects, Intel Open Source





https://github.com/akraino-edge-stack/icn

VNF deployment in ICN Stack

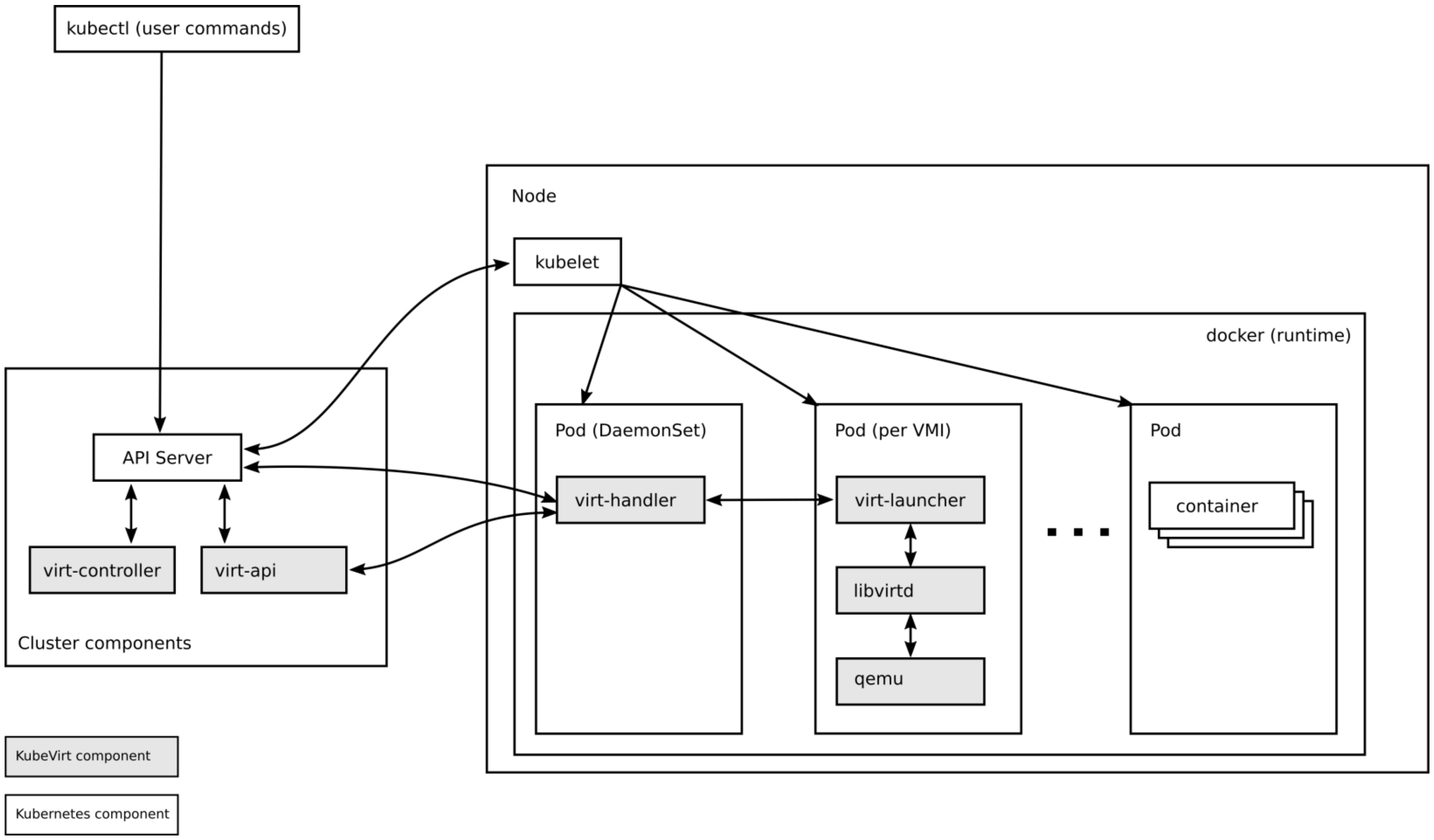


https://github.com/akraino-edge-stack/icn

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

LF NETWORKING

LFN Developer & Testing Forum

- 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

```
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
          interfaces:
          - name: default
            bridge: {}
      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

```
spec:
  template:
    spec:
      domain:
        devices:
          disks:
            - name: cloudinitdisk
              disk:
                bus: virtio
      volumes:
        - name: cloudinitdisk
          cloudInitNoCloud:
            networkData: |
              version: 2
              ...
            userData: |
              #cloud-config
              ssh_pwauth: True
              ...
```

Networks (1/2)

LF NETWORKING
LFN Developer & Testing Forum

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

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - name: default
            bridge: {}
      networks:
      - name: default
        pod: {}
```

Networks (2/2)

LF NETWORKING
LFN Developer & Testing Forum

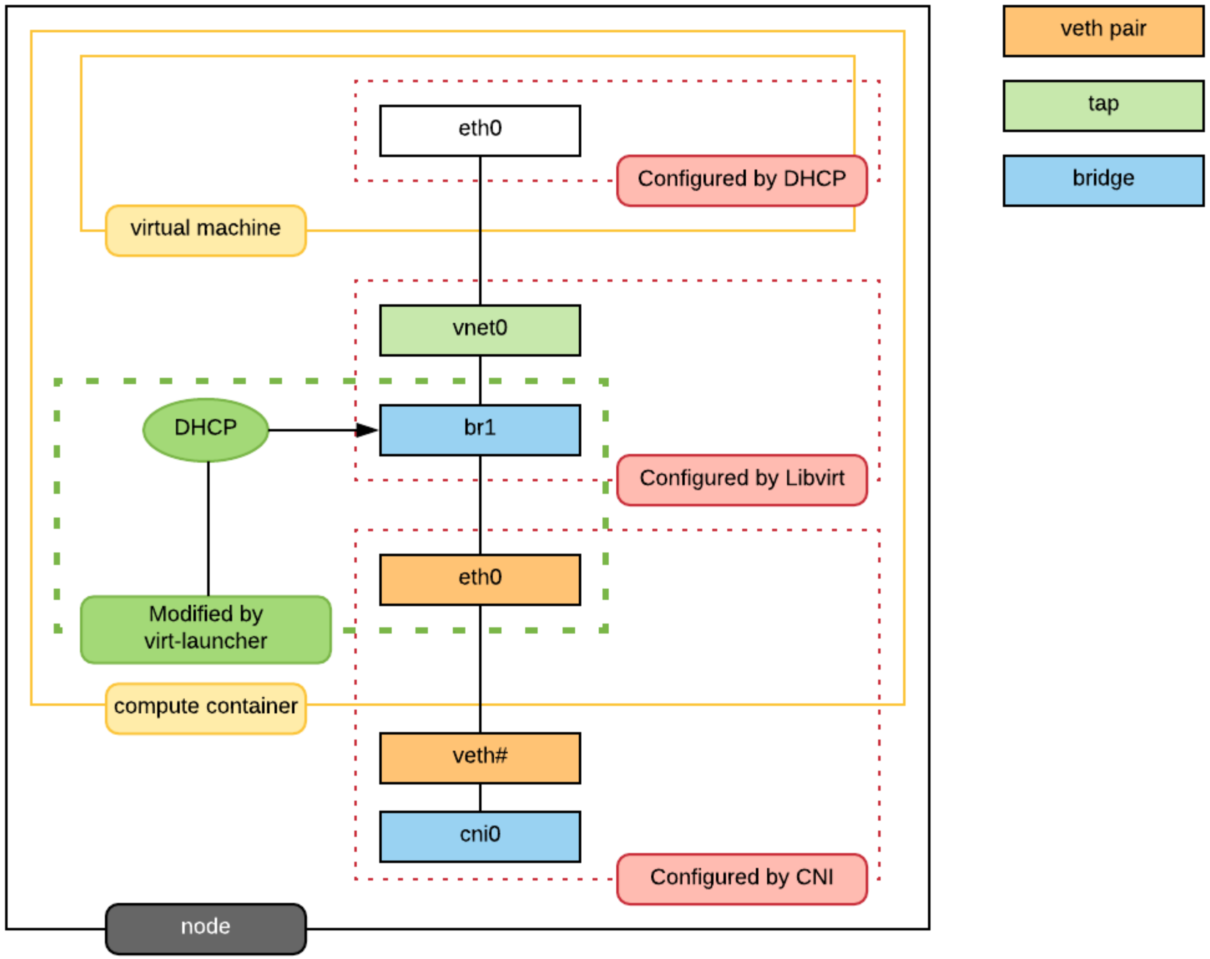
- Using multiple networks is accomplished by using Multus
- With Pods, the *k8s.v1.cni.cncf.io/networks* annotation is added to the spec containing the name of *NetworkAttachmentDefinition* describing the additional network
- With VMIs, a guest and host part are again added
- KubeVirt translates the VMI spec for the additional network to a *k8s.v1.cni.cncf.io/networks* annotation on the KubeVirt managed Pod

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - name: additional-network
            bridge: {}
      networks:
      - name: additional-network
        multus:
          networkName: additional-network-attachment-definition-name
```

KubeVirt Network Architecture (1/3)

LF NETWORKING
LFN Developer & Testing Forum

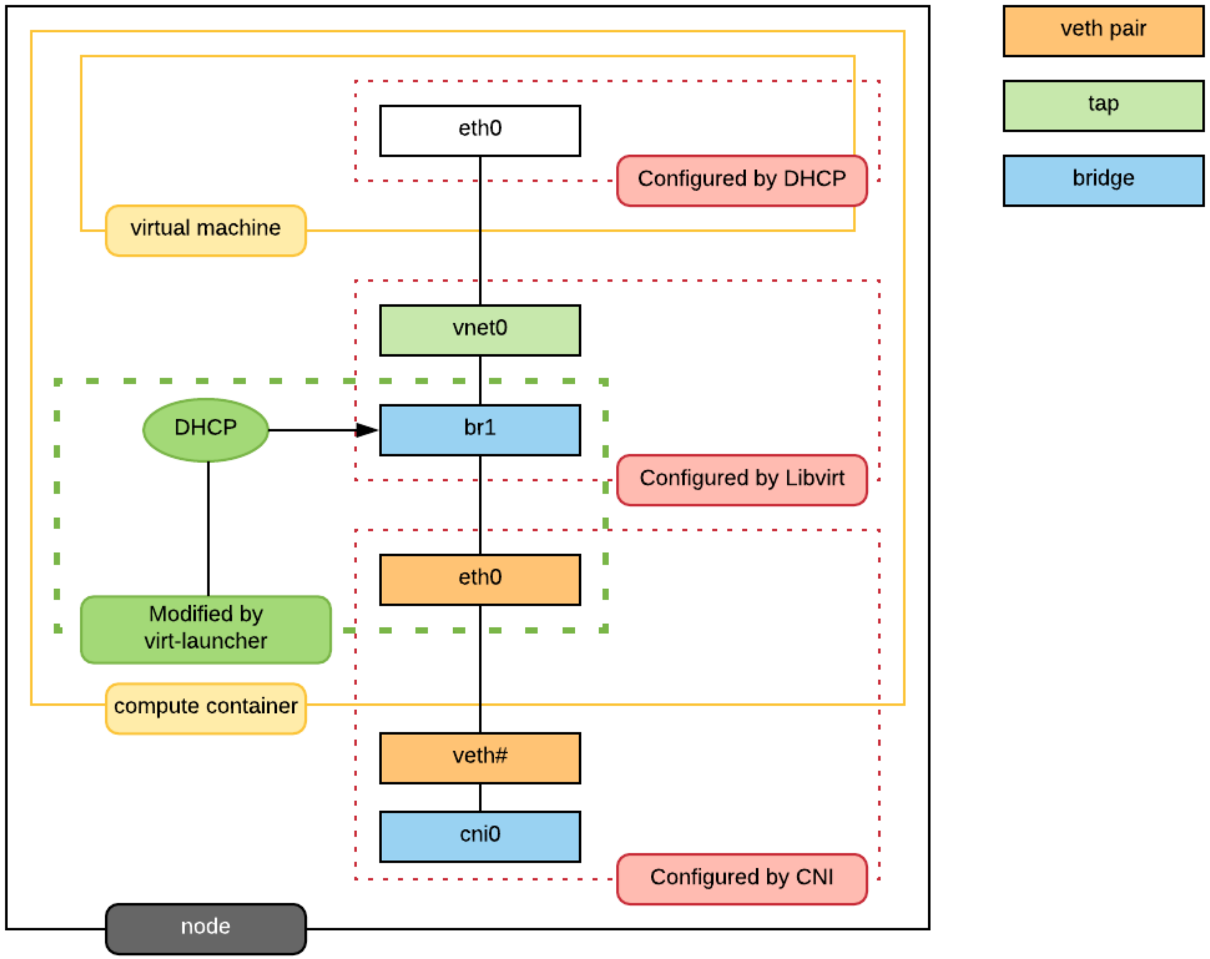
```
admin@fw0-firewall:~$ ip a
2: enp1s0: ...
    link/ether 2a:81:33:d6:c6:a2 ...
    inet 10.244.65.57/24 brd 10.244.65.255 ...
3: eth1: ...
    link/ether 52:57:2b:7b:e4:27 ...
    inet 192.168.10.3/24 brd 192.168.10.255 ...
```



KubeVirt Network Architecture (2/3)

LF NETWORKING
LFN Developer & Testing Forum

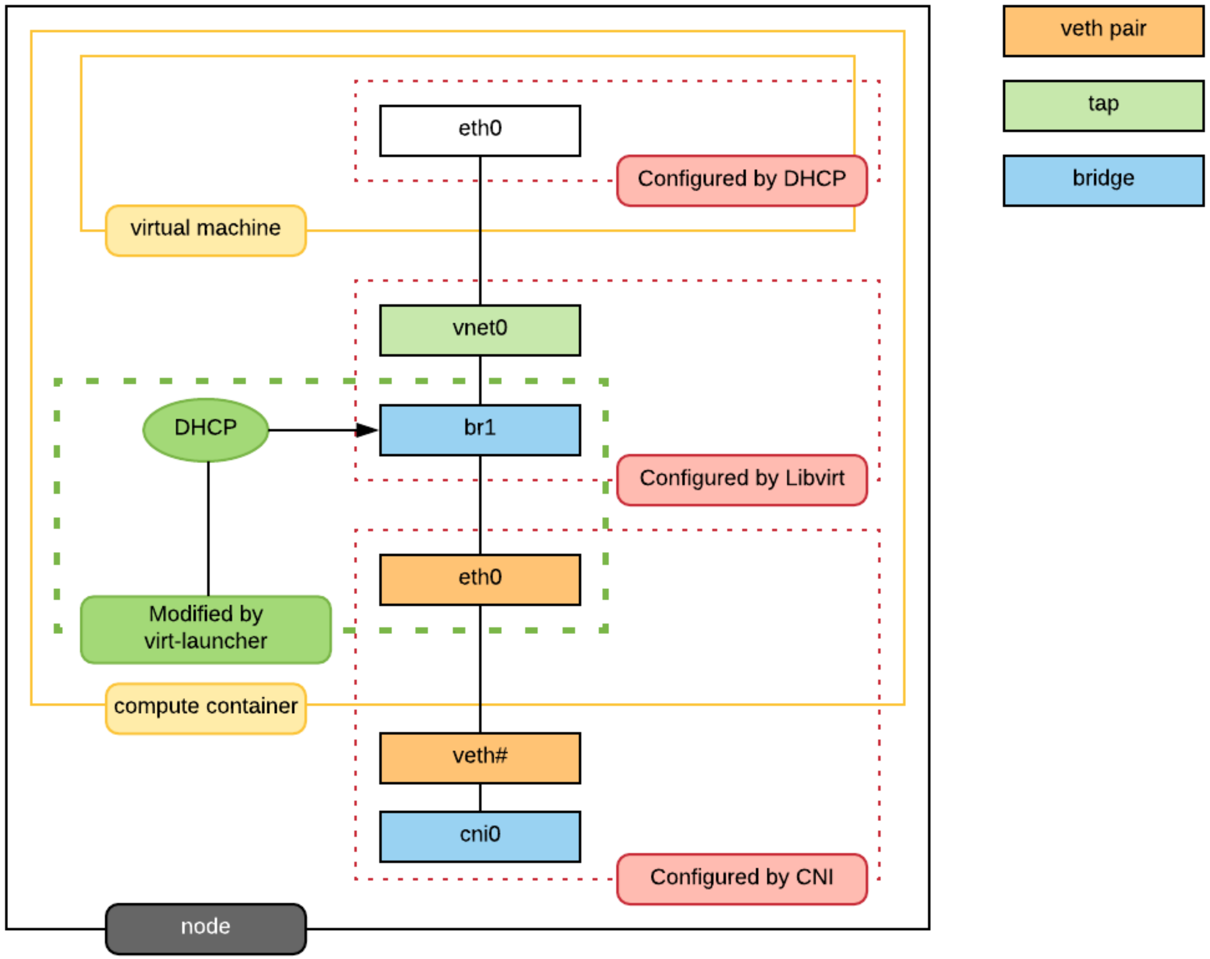
```
bash-5.0# ip a
3: eth0-nic@if80: ... master k6t-eth0 ...
    link/ether 2a:81:33:f7:6a:fe ...
5: net1-nic@if81: ... master k6t-net1 ...
    link/ether 52:57:2b:77:54:6a ...
10: eth0: ...
    link/ether c2:d9:9f:41:01:e9 ...
11: k6t-eth0: ...
    link/ether 2a:81:33:f7:6a:fe ...
    inet 169.254.75.10/32 ...
12: tap0: ... master k6t-eth0 ...
    link/ether ce:4d:7c:c2:35:9d ...
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)

LF NETWORKING
LFN Developer & Testing Forum

```
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

```
spec:
  template:
    spec:
      domain:
        devices:
          disks:
            - name: pvcdisk
              disk:
                bus: virtio
      volumes:
        - name: pvcdisk
          persistentVolumeClaim:
            claimName: fedora-pvc
```

Persistent Volumes (2/2)

LF NETWORKING
LFN Developer & Testing Forum

- 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

```
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

LF NETWORKING
LFN Developer & Testing Forum

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

```
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

LF NETWORKING
LFN Developer & Testing Forum

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

LF NETWORKING
LFN Developer & Testing Forum

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
```

Enabling High Performance

LF NETWORKING
LFN Developer & Testing Forum

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

```
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
```

Enabling High Performance

LF NETWORKING
LFN Developer & Testing Forum

SR-IOV (4/4)

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

```
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"}],
        "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

```
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

Helm Charts

- 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 Intents (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

Network Controller Intents (2/2)

- 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.)

```
version: emco/v2
resourceContext:
  anchor: projects/P/composite-apps/CA/V/deployment-intent-groups/DIG/network-
controller-intent/NCI/workload-intents/WI/interfaces
metadata:
  name: NAME
spec:
  interface: net1
  name: PROVIDER-NETWORK
  defaultGateway: "false"
  ipAddress: 192.168.10.2
  macAddress: ee:f0:75:e0:b6:26
```

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

Network Controller Additions (2/2)

LF NETWORKING
LFN Developer & Testing Forum

- 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

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: unprotected-private-net
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "type": "ovn4nfvk8s-cni",
      "nfn-network": "unprotected-private-net"
    }
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.plugin.opnfv.org/nfn-network: '{"type":"ovn4nfv","interface":[
      {"interface":"net1","name":"unprotected-private-net",
       "defaultGateway":"false","ipAddress":"192.168.10.3",
       "macAddress":"52:57:2b:7b:e4:27"},
      ...
    k8s.plugin.opnfv.org/ovnInterfaces: '[
      {"ip_address":"192.168.10.3/24","mac_address":"52:57:2b:7b:e4:27",
       "gateway_ip":"10.154.142.20","defaultGateway":"false",
       "interface":"net1"},
      ...
  name: virt-launcher-fw0-firewall-t6ld6
```

LF NETWORKING

LFN Developer & Testing Forum

VNF Life Cycle Management with EMCO and KubeVirt

Kuralamudhan Ramakrishnan, kuralamudhan.ramakrishnan@intel.com

Todd Malsbary, todd.malsbary@intel.com

01/13/2022