ICN Nodus

Introduction:

- Introduction:
- Problem statement:
- Architecture:
- Features:
    - InFinite Network Resources:
        - Virtual Networks:
        - Provider Networks:
        - Service Function Chaining:
    - Finite network Resources:
        - SRIOV Overlay Networks:
            - Required features in SRIOV Overlay networking:
                - SRIOV Type Virtual network
                - SRIOV Type provider network
                - SRIOV Type Direct network
                - Parameter definition:

Comprehensive documentation:
Presentation:

- Updated 05/18/2022
- LFN Developer Forum - January 13th 2022
- Open vSwitch and OVN 2021 Fall conference - Dec 7th & 8th 2021
- October 11th 2021
    - Nodus slide deck:
- July 19th 2021
    - Nodus slide deck:
- March 9th 2021
    - OVN4NFV Slide deck:
    - OVN4NFV recorded Video:
- October 8th 2020
    - OVN4NFV slide deck:
    - OVN4NFV Prerecorded Demo:

Nodus is the Network controller designed based on the K8s controller framework and provides Open flow control based on OVN. This address OVN based Multiple Network creation, support Multiple network interfaces and support Virtual networking and Provider networkings.

Problem statement:

For 5 years, there has been quite the development in Cloud Native development to develop infrastructure to build and run the application that is fully automated, secure, flexible, performant, and resilient. This gives an opportunity in the evolution journey of the Monolith application to be decomposed into VMs and they are further decomposed into Microservices. This led to the path of Kubernetes to become a de-facto orchestration engine that facilitates this journey. During these courses, it was identified that Kubernetes networking and Container Networking are not addressing the need for Network evolutions in multiples cases starting from creating and managing the network infrastructure and automating it, and in general, it lacks Software Defined Networking (SDN) concepts such as programmable networks, decoupling the control and data planes, and scalability. Kubernetes networking has evolved so far only for the data ware centric cloud networking and it lacks network distribution to be suitable for Edge Cloud Networking

Nodus is the network controller in Kubernetes that addresses the need for Software Defined Networking that simplifies the network operations, builds programmable networks for greater flexibility and scalability.

Application transformation is one of the major objectives in the edge computing in the cloud-native evolution. Taking a PNF(Physical Network Function) or a VNF(Virtual Network Functions) to be ready to deploy in the edge is as challenging because the NFs(Network Functions) are composited into smaller microservices and these microservers will be deployed in the multiple edge location. Controlling the network traffics such as both control plane and data plane traffics in the scenarios is required to achieve low latency and multiple clusters networking

Architecture:

Nodus is a network controller and is designed to deliver Cloud-Native programmable networks that enable centralized control and network policy enforcement across physical and virtual networks.

Nodus support Kubernetes Networking, Open Virtual Network (OVN) based networking features to support virtual switching, virtual network abstractions and Service Function chaining. Nodus is planned to deliver features on Network Multi-tenancy, Network OoS, Network based Multicore Utilization, and TCFlower based network access.

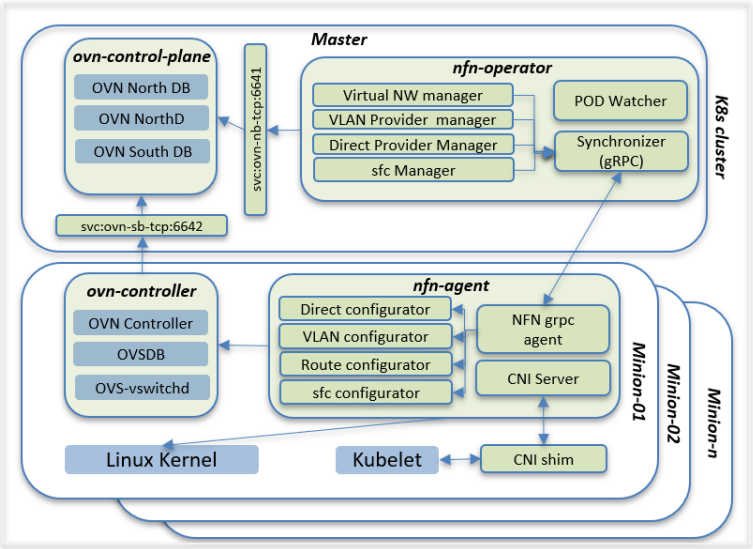
Adding Multi cluster networking is a challenging requirement for Edge networking in the cloud-native world. As Kubernetes delegates all the networking features to CNI(Container Network Interfaces), and right now we have 16+ CNI types that offer various networking features starting from localhost to BGP networking. Having a single network controller for the Multiple Cluster within an edge and also across geo-distributed edge location is a requirement to create a virtual network, provider networks across the edges, and apply the same tuning parameter for the network resources in the edges.

Nodus is decomposed into 4 microservices, Network Function (NFN) Operators, Network Function (NFN) agent, OVN Control plane, and OVN controller.

Network Function (NFN) Operator is a Kubernetes Operator framework that hosts multiple Kubernetes Custom resource definition controller, and Kubernetes core API watchers to monitor and predicate the events to synchronize the network operations. It gets the input from the user, and it acts as an orchestration engine for Networking to create, manage virtual and physical network support for both VLAN and direct networking, Service Function chaining using OVN control plane and NFN agent.

Network Function (NFN) agent is a Kubernetes daemon set to get the command from Network Function (NFN) operator to do network plumbing works such as network configuration and managing the workloads network namespace in each host using Container Network Interfaces specs.

OVN control-plane hosts all the OVN Northbound API as the microservices and the OVN controller host all the OVN Southbound API and OVS virtual switch as the microservices. OVN control plane gets the network abstraction configuration from NFN operators and translates network configuration into Openflow to implement distributed logical switches, Access control lists for network policy implementations, Service function chaining, and distributed virtual routers.



NFN Operator:

- Exposes virtual, provider, chaining CRDs to external world.
- Programs OVN to create L2 switches.
- Watches for PODs being coming up
    - Assigns IP addresses for every network of the deployment.
    - Looks for replicas and auto create routes for chaining to work.
    - Create LBs for distributing the load across CNF replicas.

NFN agent:

- Performs CNI operations.
- Configures VLAN and Routes in Linux kernel (in case of routes, it could do it in both root and network namespaces)
- Communicates with OVSDB to inform of provider interfaces. (creates ovs bridge and creates external-ids:ovn-bridge-mappings)

Features:

InFinite Network Resources:

Virtual Networks:

Nodus uses the NFN operator to define the virtual network CRs that will create a OVN networking for virtual networking as defined in the CR.

Virutal network

```
apiVersion: k8splugin.opnfv.org/v1alpha1
kind: Network
metadata:
  name: ovn-priv-net
spec:
  cniType: ovn4nfv
  ipv4subnets:
  - subnet: 172.16.33.0/24
    name: subnet1
    gateway: 172.16.33.1/24
    excludeIps: 172.16.33.2 172.16.33.5..172.16.33.10
```

This CR defines the OVN networking and provides the gateway and exclude IPs to be reserved for any internal static IP address assignment.

Provider Networks:

Provider network supports both VLAN and direct provider networking

Virutal network

```
apiVersion: k8s.plugin.opnfv.org/v1alpha1
kind: ProviderNetwork
metadata:
  name: pnetwork
spec:
  cniType: ovn4nfv
  ipv4Subnets:
  - subnet: 172.16.33.0/24
    name: subnet1
    gateway: 172.16.33.1/24
    excludeIps: 172.16.33.2 172.16.33.5..172.16.33.10
  providerNetType: VLAN
  vlan:
    vlanId: "100"
    providerInterfaceName: eth0
    logicalInterfaceName: eth0.100
    vlanNodeSelector: specific
    nodeLabelList:
    - kubernetes.io/hostname=ubuntu18
```

The major change between the VLAN provider network and direct provide networks is the VLAN information is provided in the VLAN CR and they are excluded in the direct provider

Virutal network

```
apiVersion: k8s.plugin.opnfv.org/v1alpha1
kind: ProviderNetwork
metadata:
  name: directpnetwork
spec:
  cniType: ovn4nfv
  ipv4Subnets:
  - subnet: 172.16.34.0/24
    name: subnet2
    gateway: 172.16.34.1/24
    excludeIps: 172.16.34.2 172.16.34.5..172.16.34.10
  providerNetType: DIRECT
  direct:
    providerInterfaceName: eth1.
    directNodeSelector: specific
    nodeLabelList:
    - kubernetes.io/hostname=ubuntu18
```

Service Function Chaining:

Virutal network

```
apiVersion:  k8splugin.opnfv.org/v1alpha1
kind: NetworkChaining
metadata:
  name: chain1
  namespace: vFW
spec:
  type: Routing
  routingSpec:
    leftNetwork:
      - networkName: ovn-provider1
        gatewayIP: 10.1.5.1
        subnet: 10.1.5.0/24
    rightNetwork:
      - networkName: ovn-provider1
        gatewayIP: 10.1.10.1
        subnet: default
    networkChain: app=slb, ovn-net1, app=ngfw, ovn-net2, app=sdwancnf
```

Finite network Resources:

SRIOV Overlay Networks:

Required features in SRIOV Overlay networking:

- Currently, OVN4NFV by default create the Veth pair interfaces for all interfaces.
- SRIOV Overlay networks introduce a feature to include the interfaceType in the OVN networking and provide the deviceplugin sock name and targets on the devices only having SRIOV hardware-enabled labels

SRIOV Type Virtual network

Virutal network

```
apiVersion: k8splugin.opnfv.org/v1alpha1
kind: Network
metadata:
  name: ovn-sriov-net
spec:
  cniType: ovn4nfv
  ipv4subnets:
  - subnet: 172.16.33.0/24
    name: subnet1
    gateway: 172.16.33.1/24
    excludeIps: 172.16.33.2 172.16.33.5..172.16.33.10
    NodeSelector: specific
    nodeLabelList:
    - feature.node.kubernetes.io/network-sriov.capable=true
    - feature.node.kubernetes.io/custom-xl710.present=true
```

SRIOV Type provider network

Virutal network

```
apiVersion: k8s.plugin.opnfv.org/v1alpha1
kind: ProviderNetwork
metadata:
  name: ovn-sriov-vlan-pnetwork
spec:
  cniType: ovn4nfv
  interface:
  - Type:sriov
    deviceName: intel.com/intel_sriov_700
  ipv4Subnets:
  - subnet: 172.16.33.0/24
    name: subnet1
    gateway: 172.16.33.1/24
    excludeIps: 172.16.33.2 172.16.33.5..172.16.33.10
  providerNetType: VLAN
  vlan:
    vlanId: "100"
    providerInterfaceName: eth0
    logicalInterfaceName: eth0.100
    vlanNodeSelector: specific
    nodeLabelList:
    - feature.node.kubernetes.io/network-sriov.capable=true
    - feature.node.kubernetes.io/custom-xl710.present=true
```

SRIOV Type Direct network

Virutal network

```
apiVersion: k8s.plugin.opnfv.org/v1alpha1
kind: ProviderNetwork
metadata:
  name: ovn-sriov-direct-pnetwork
spec:
  cniType: ovn4nfv
  interface:
  - Type:sriov
    deviceName: intel.com/intel_sriov_700
  ipv4Subnets:
  - subnet: 172.16.34.0/24
    name: subnet2
    gateway: 172.16.34.1/24
    excludeIps: 172.16.34.2 172.16.34.5..172.16.34.10
  providerNetType: DIRECT
  direct:
    providerInterfaceName: enp
    directNodeSelector: specific
    nodeLabelList:
    - feature.node.kubernetes.io/network-sriov.capable=true
    - feature.node.kubernetes.io/custom-xl710.present=true
```

Parameter definition:

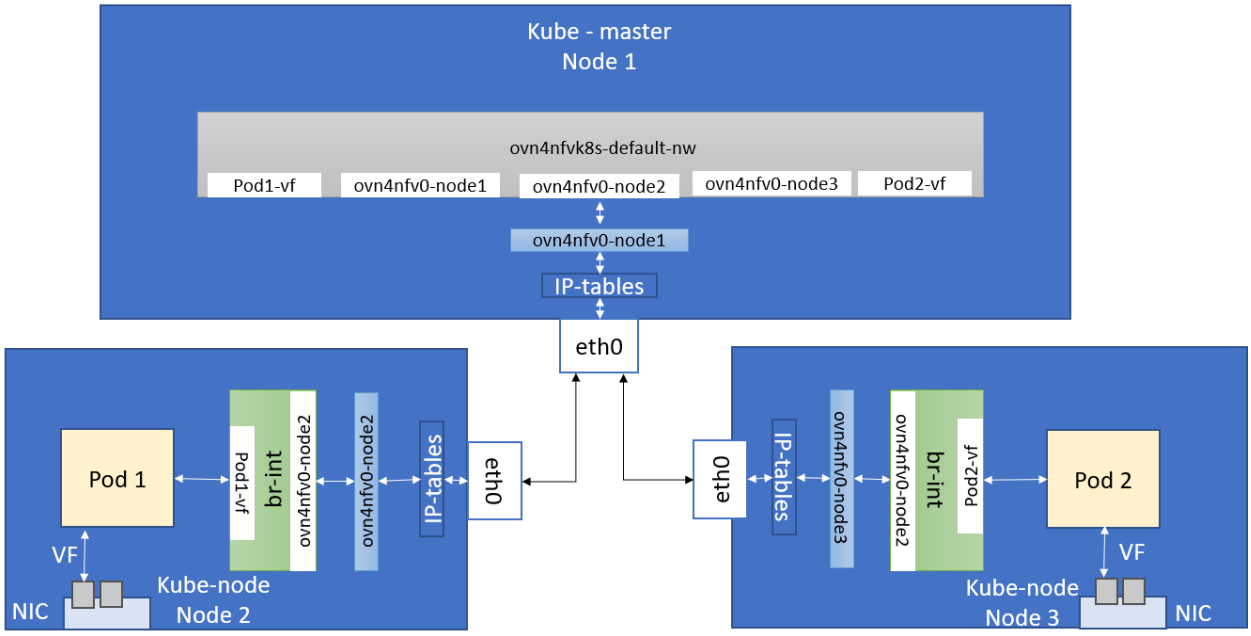
interface - Define the type of sriov interface to be created.

deviceName - Define device plugin to be targeted to get the pod resource information from the kubelet api - For more information refer here -

https://github.com/kubernetes/kubernetes/blob/master/test/e2e_node/util.go

sriov single pod spec

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-case-01
  annotations:
    k8s.plugin.opnfv.org/nfn-network: '{ "type": "ovn4nfv", "interface": [{ "name": "ovn-sriov-virutal-net",
"interface": "net0", Type: sriov, Attribute: [{type: bonding, mode: roundrobin}, {type:tuning, bandwidth:”
2GB”}}]}'
spec:
  containers:
  - name: test-pod
    image: docker.io/centos/tools:latest
    command:
    - /sbin/init
    resources:
      requests:
        intel.com/intel_sriov_700: '1'
      limits:
        intel.com/intel_sriov_700: '1'
```



1. Admission controller should be part of NFN operator that insert the request and limit to pod spec by reading the OVN4NFV net CR.
2. This design adds the SRIOV directly into the OVN overlay for both primary and secondary networking. The development should also address the SNAT for all the interfaces

Comprehensive documentation:

Nodus

How to use?

Development

Configuration

Presentation:

Updated 05/18/2022

LFN Developer Forum - January 13th 2022

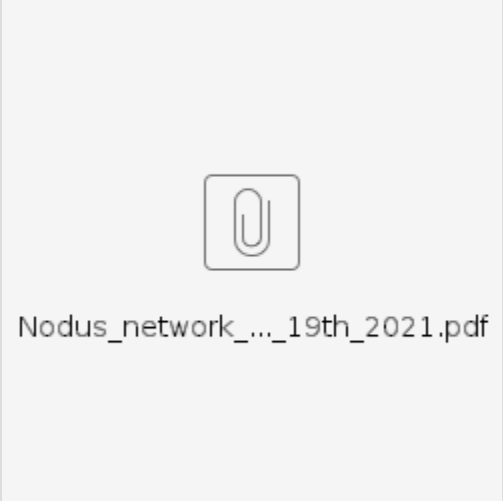
Open vSwitch and OVN 2021 Fall conference - Dec 7th & 8th 2021

October 11th 2021

Nodus slide deck:

July 19th 2021

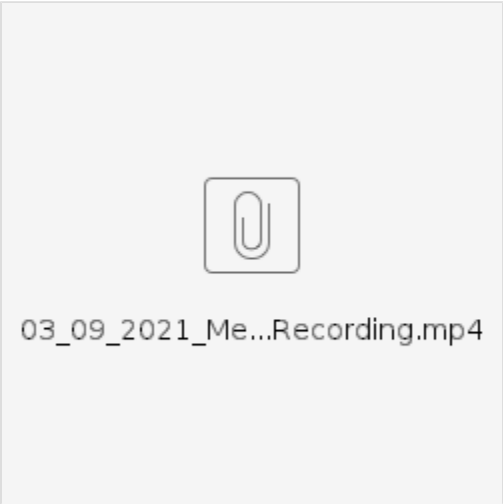
Nodus slide deck:



March 9th 2021

OVN4NFV Slide deck:

OVN4NFV recorded Video:



October 8th 2020

OVN4NFV slide deck:

OVN4NFV Prerecorded Demo: