kubernetes-nmstate

Apply complex network configuration to your Kubernetes cluster hosts by declaring it

Petr Horáček
phoracek@redhat.com
github.com/phoracek

Agenda

- **nmstate**
- **kubernetes-nmstate**
- **Demo**

CONFIGURE YOUR HOSTS

Configuring networking is tedious

- Sometimes the **default Kubernetes network is not enough**
- **SR-IOV** and **L2** requirements
- **Dynamic** node network **configuration**

nmstate applies your desired state

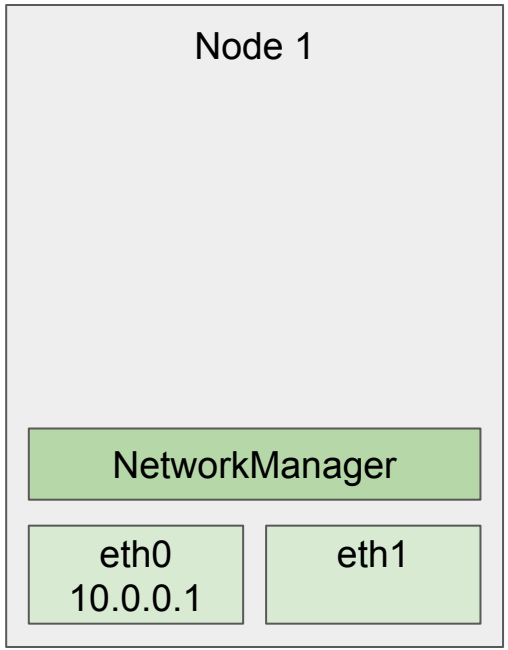
- **Declarative** network management

- Configuration tool on top of **NetworkManager**

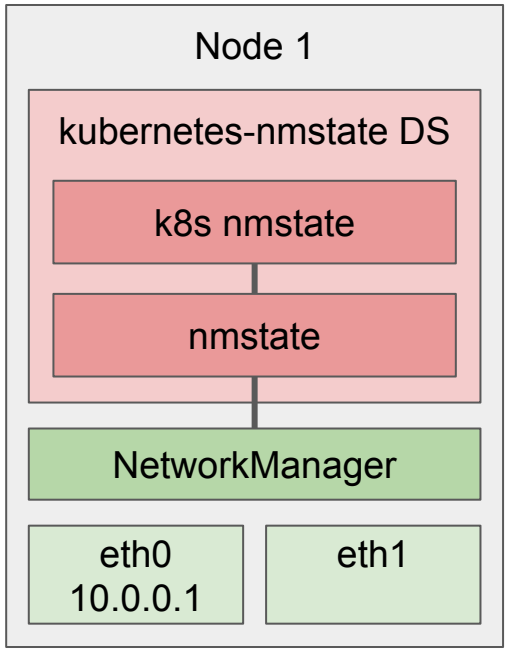
kubernetes-nmstate

- **Host-networking** driven by kubernetes and executed by nmstate
- **NodeNetworkState**
- **NodeNetworkConfigurationPolicy**

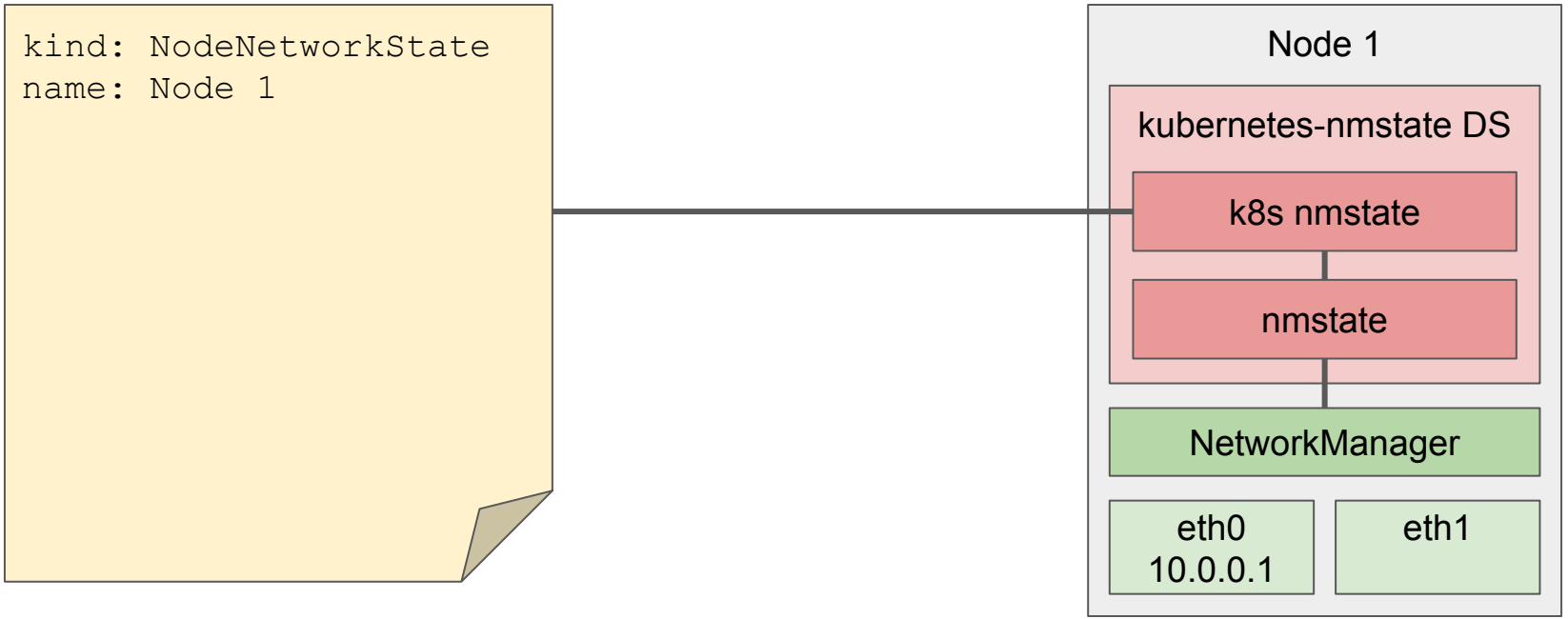
kubernetes-nmstate



kubernetes-nmstate

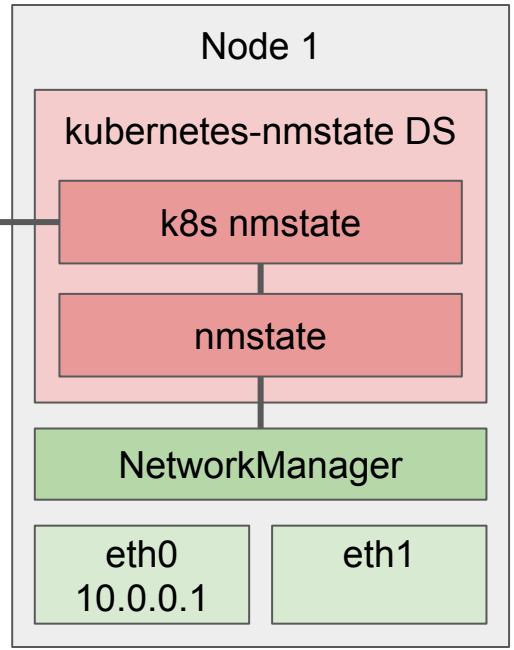


kubernetes-nmstate



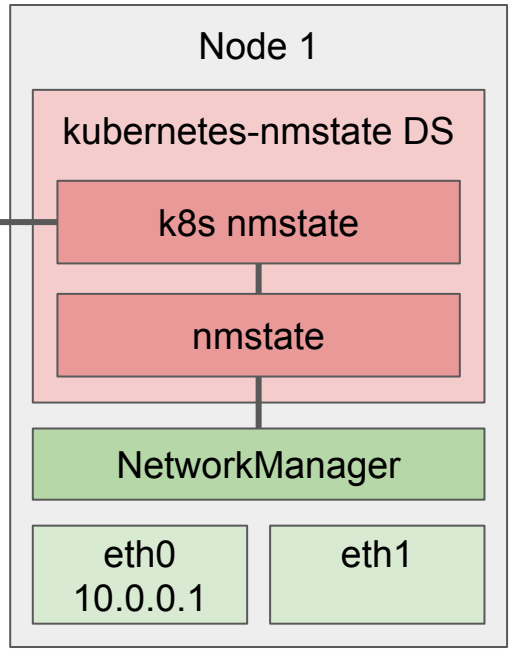
kubernetes-nmstate

```
kind: NodeNetworkState
name: Node 1
state:
  eth0: 10.0.0.1
  eth1: None
```



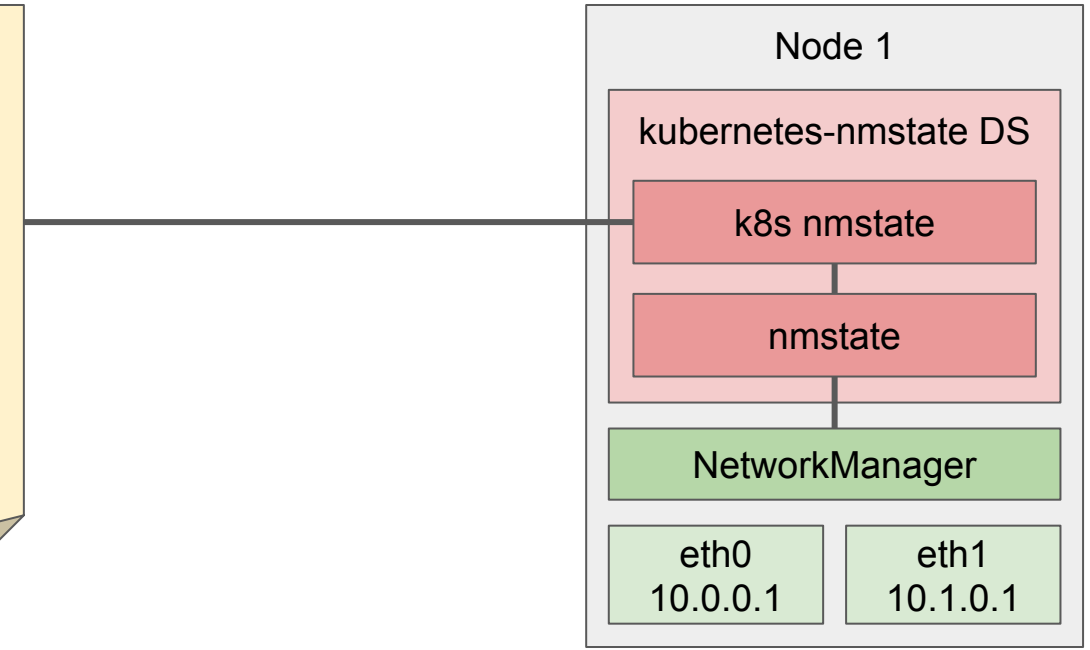
kubernetes-nmstate

```
kind: NodeNetworkState
name: Node 1
state:
  eth0: 10.0.0.1
  eth1: None
spec:
  eth1: 10.1.0.1
```



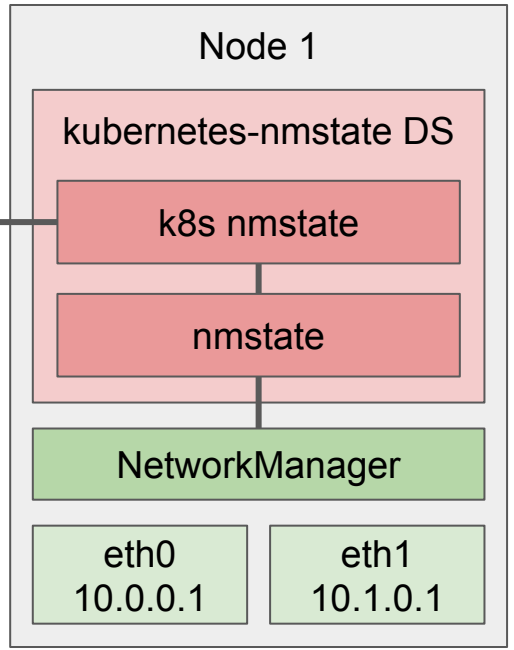
kubernetes-nmstate

```
kind: NodeNetworkState
name: Node 1
state:
  eth0: 10.0.0.1
  eth1: None
spec:
  eth1: 10.1.0.1
```

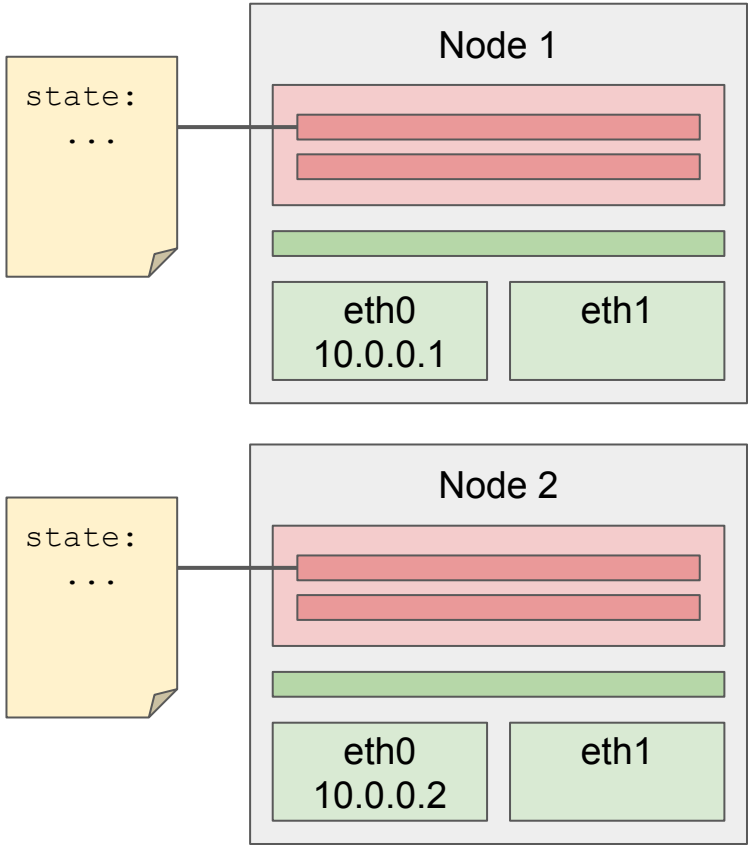


kubernetes-nmstate

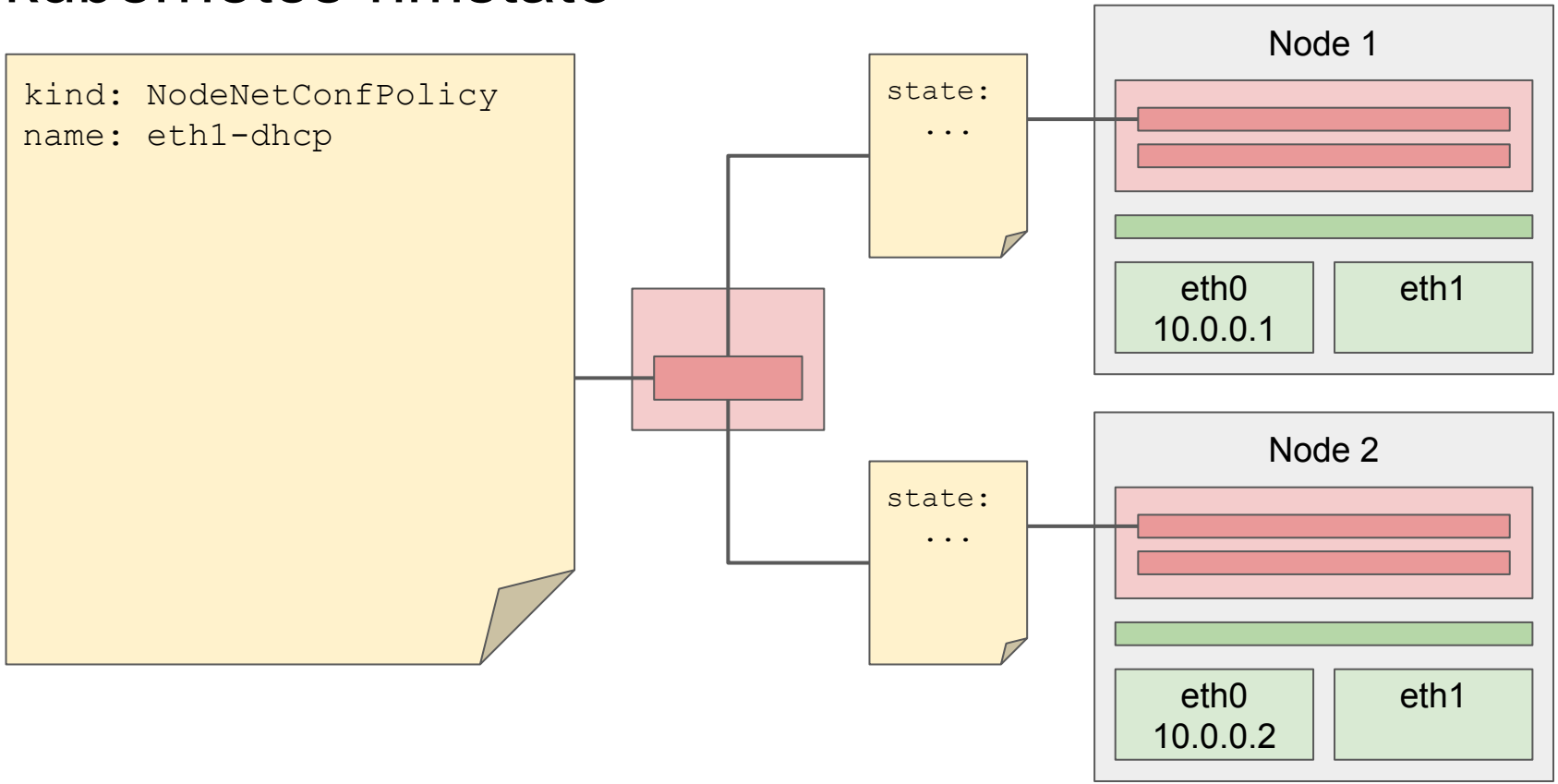
```
kind: NodeNetworkState
name: Node 1
state:
  eth0: 10.0.0.1
  eth1: 10.1.0.1
spec:
  eth1: 10.1.0.1
```



kubernetes-nmstate

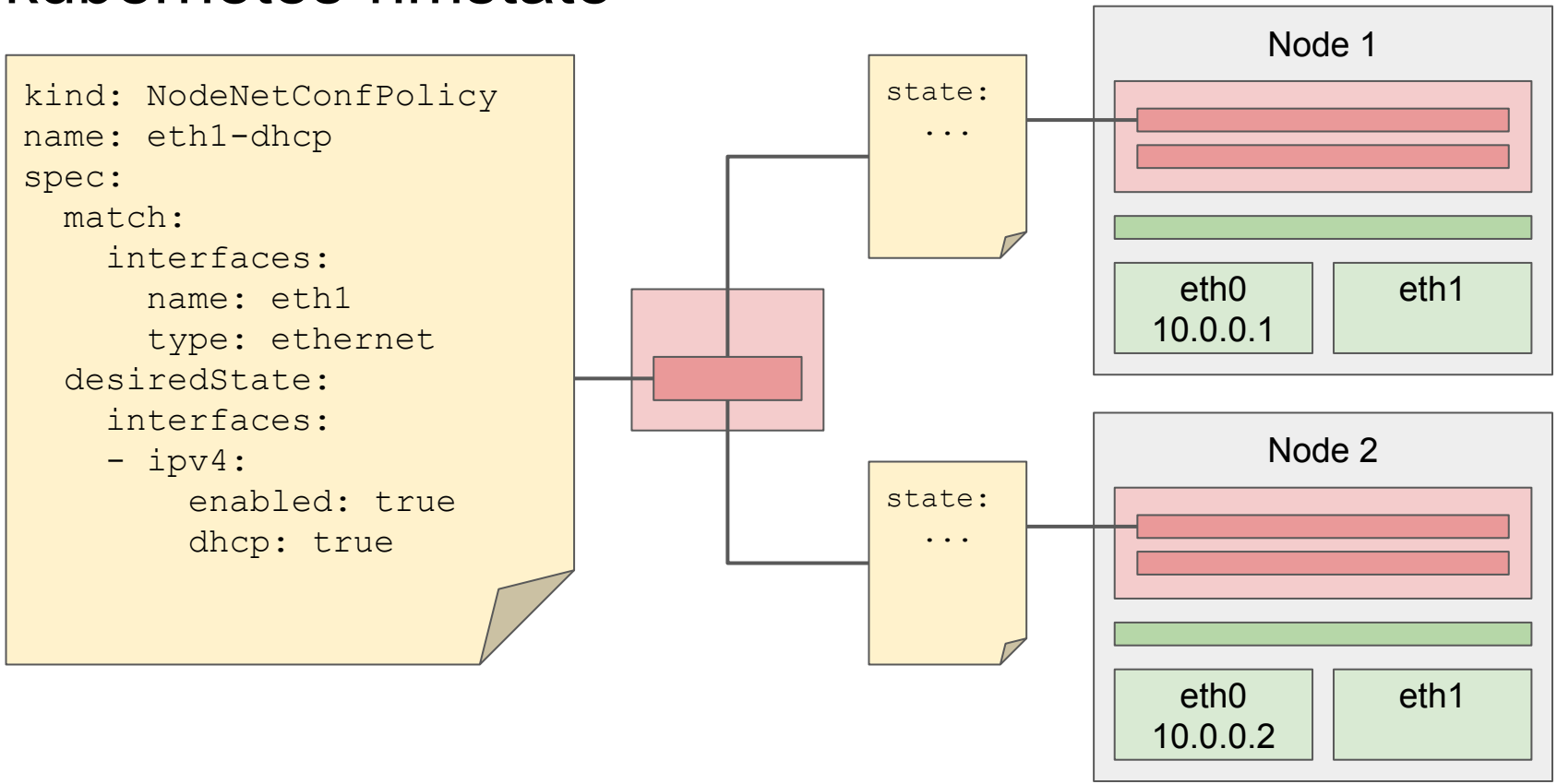


kubernetes-nmstate

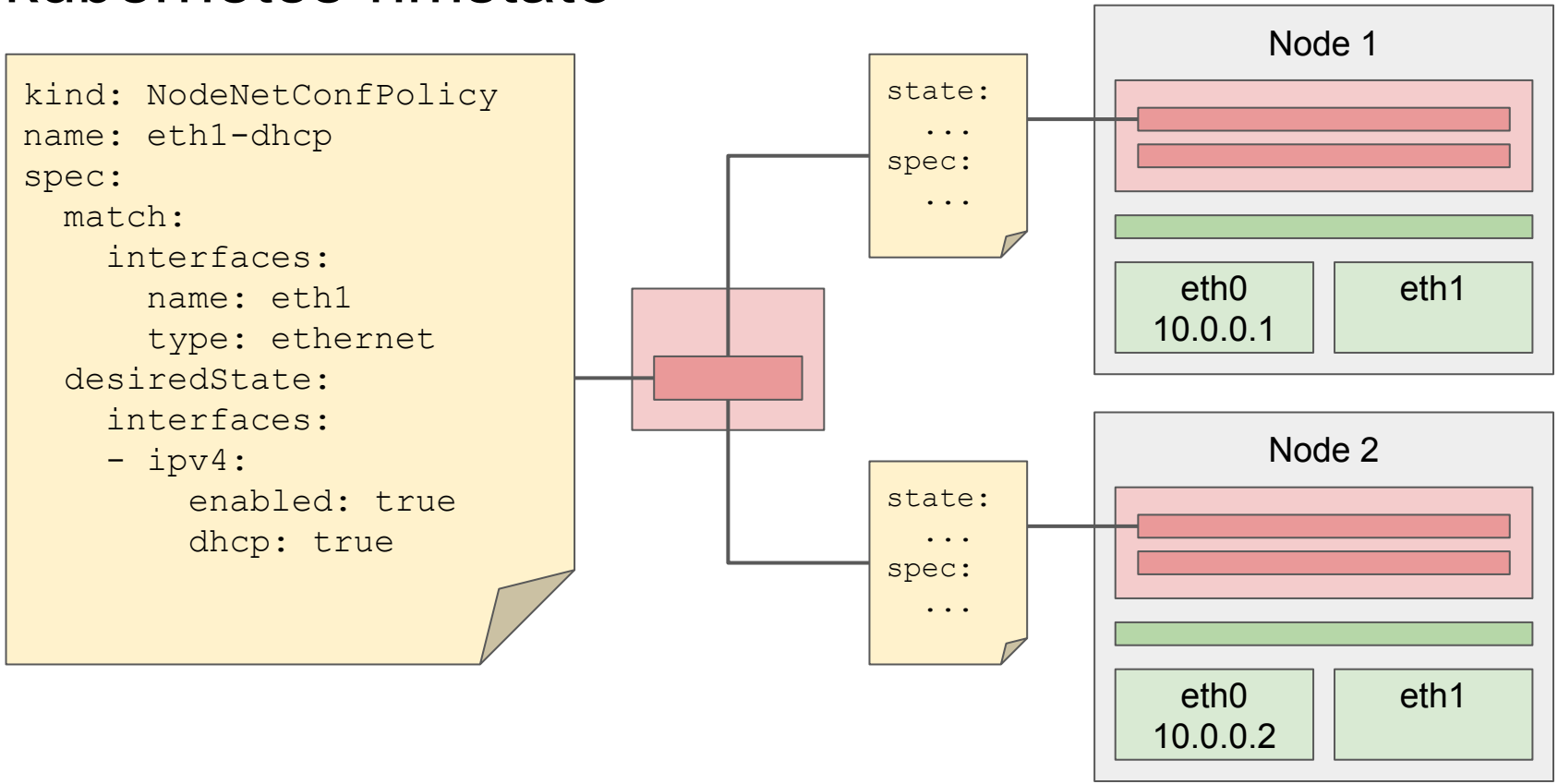


kubernetes-nmstate

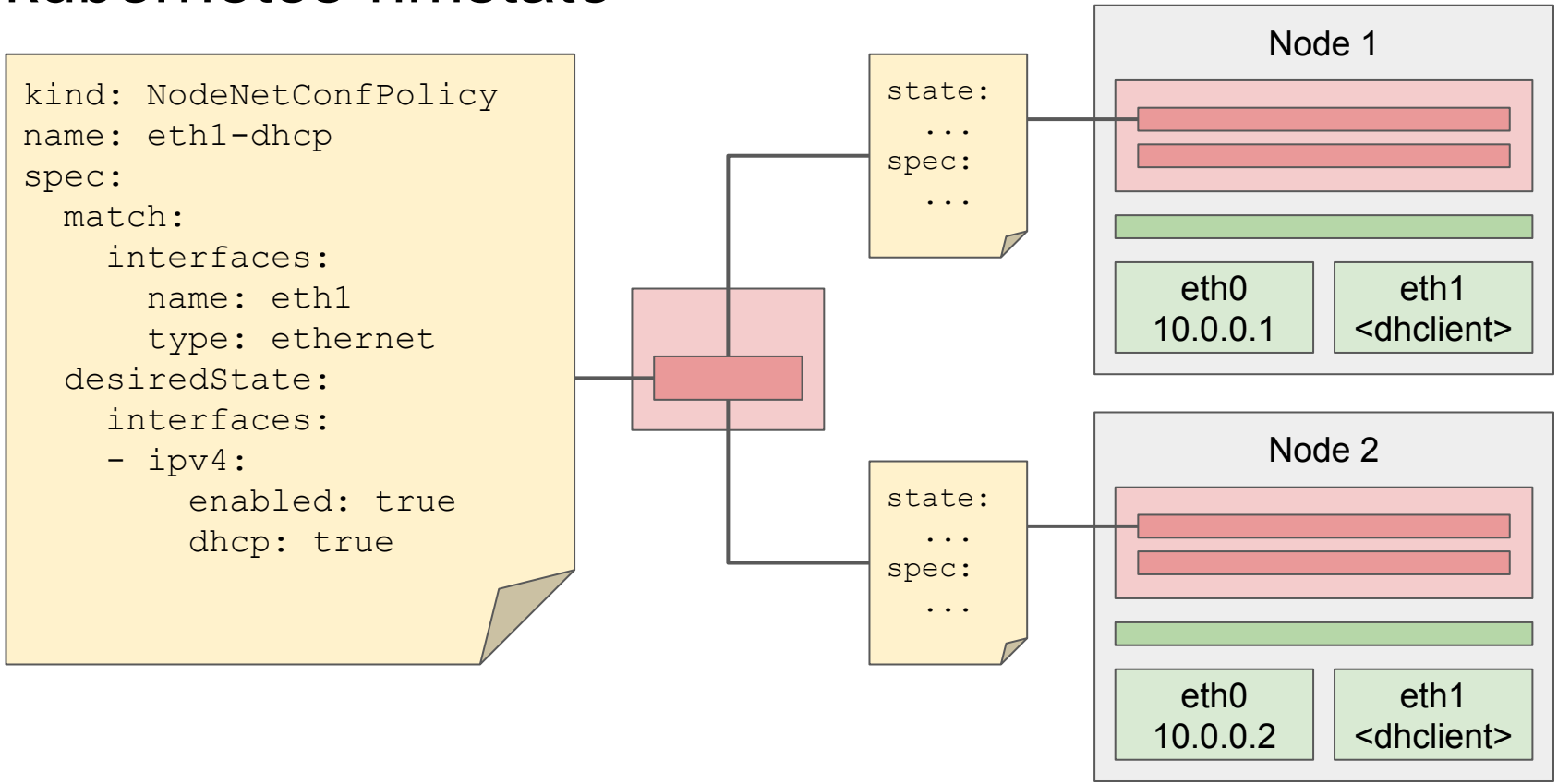
```
kind: NodeNetConfPolicy
name: eth1-dhcp
spec:
  match:
    interfaces:
      name: eth1
      type: ethernet
  desiredState:
    interfaces:
    - ipv4:
        enabled: true
        dhcp: true
```



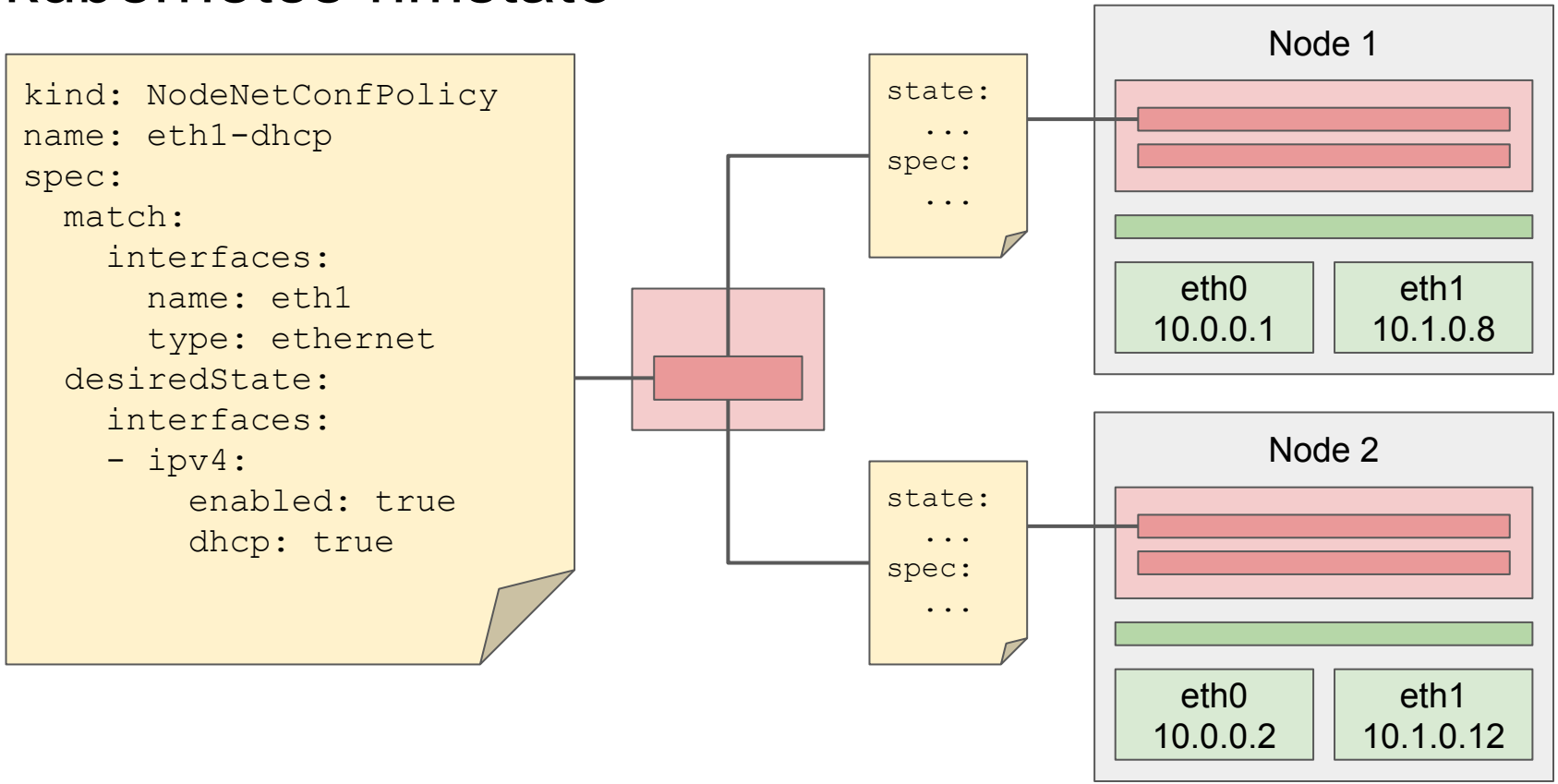
kubernetes-nmstate



kubernetes-nmstate



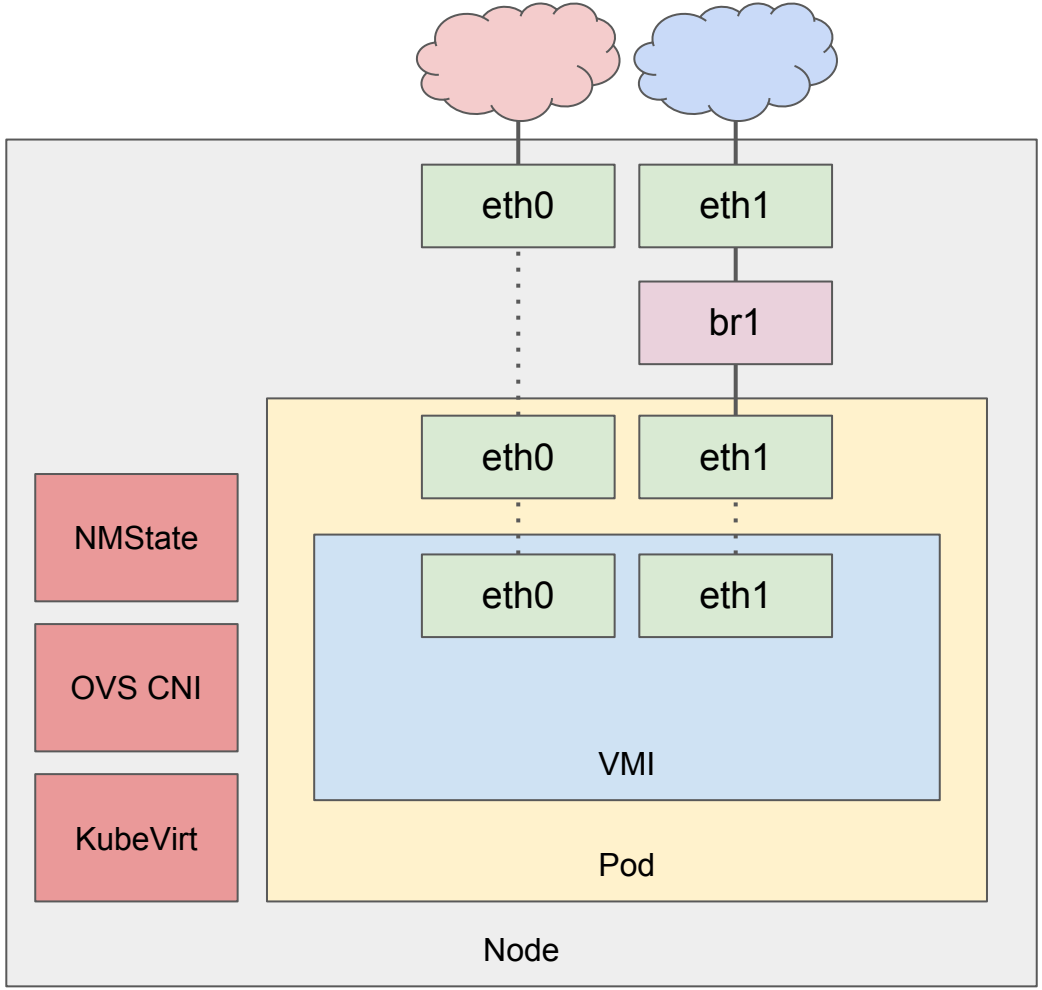
kubernetes-nmstate

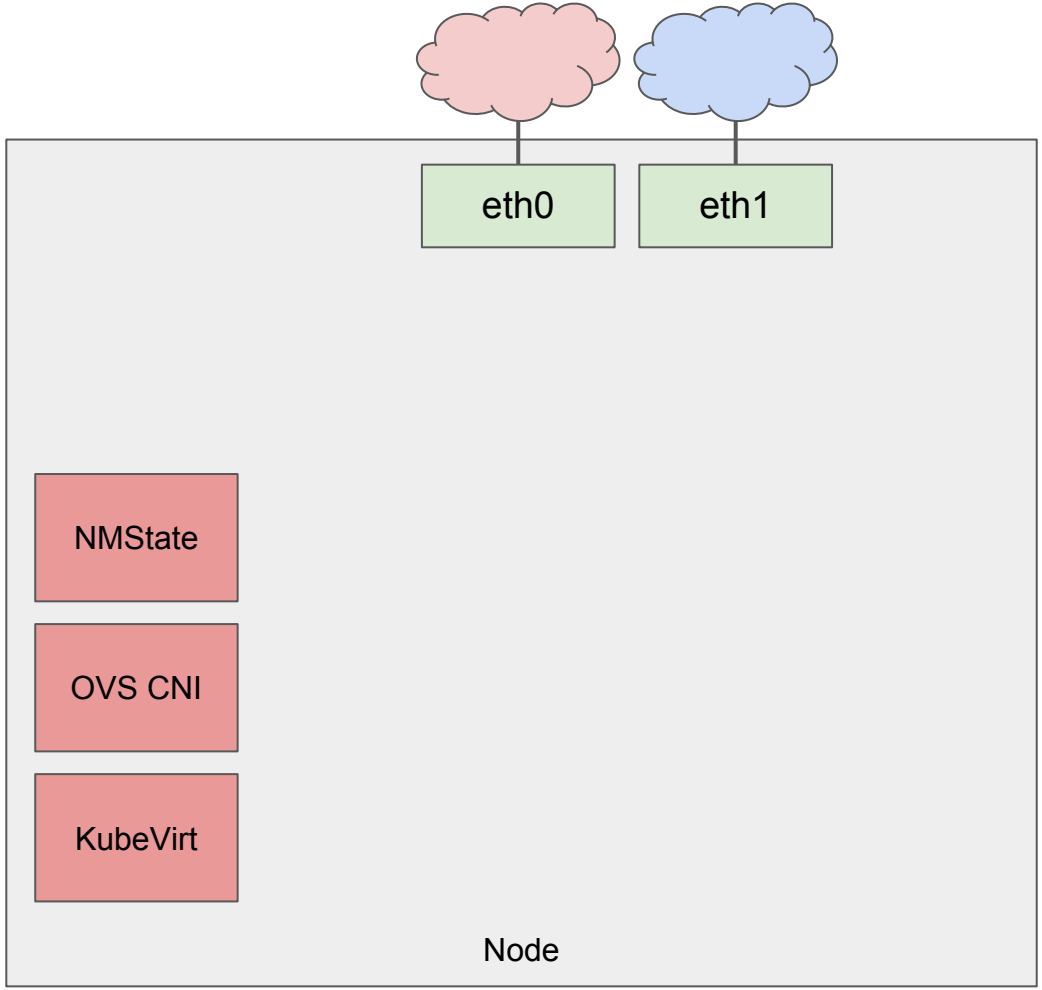


DEMO

Demo

- **Report network interfaces** using kubernetes-nmstate
- **Configure network interfaces** using kubernetes-nmstate
- **Create a VM** requesting secondary network connection using KubeVirt, Multus and OVS CNI


eth0
eth1
br1
eth0
eth1
eth0
eth1
NMState
OVS CNI
KubeVirt
VMI
Pod
Node


eth0
eth1
NMState
OVS CNI
KubeVirt
Node

```
#
```

```
# # check current node network status
```

```
# # check current node network status
# 
```

```
# # check current node network status
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 1
    name: 192.168.122.101
    namespace: nmstate-default
    resourceVersion: "1553"
    selfLink: /apis/nmstate.io/v1/namespaces/nmstate-default/nodenetworkstates/19216
8.122.101
    uid: 4c14cd37-150c-11e9-be67-5254002ef48e
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
:
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 1
    name: 192.168.122.101
    namespace: nmstate-default
    resourceVersion: "1553"
    selfLink: /apis/nmstate.io/v1/namespaces/nmstate-default/nodenetworkstates/19216
8.122.101
    uid: 4c14cd37-150c-11e9-be67-5254002ef48e
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
/spec
```

```
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
    currentState:
      capabilities: null
      interfaces:
      - ifIndex: 0
        ipv4:
          address:
          - ip: 10.233.64.1
            prefix-length: 24
          enabled: true
        ipv6:
          address:
          - ip: fe80::3889:6aff:fe63:c613
            prefix-length: 64
:
```

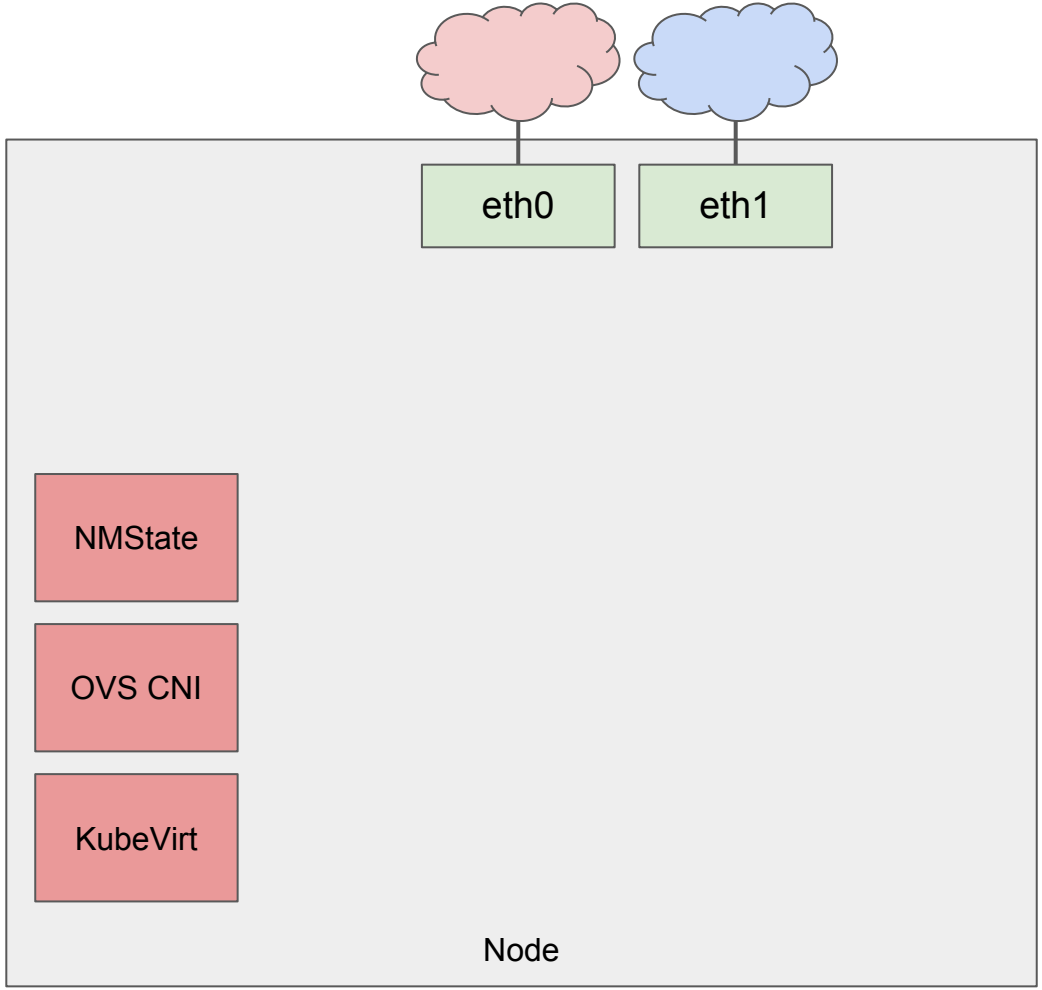
```
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
    currentState:
      capabilities: null
      interfaces:
      - ifIndex: 0
        ipv4:
          address:
          - ip: 10.233.64.1
            prefix-length: 24
          enabled: true
        ipv6:
          address:
          - ip: fe80::3889:6aff:fe63:c613
            prefix-length: 64
/status
```

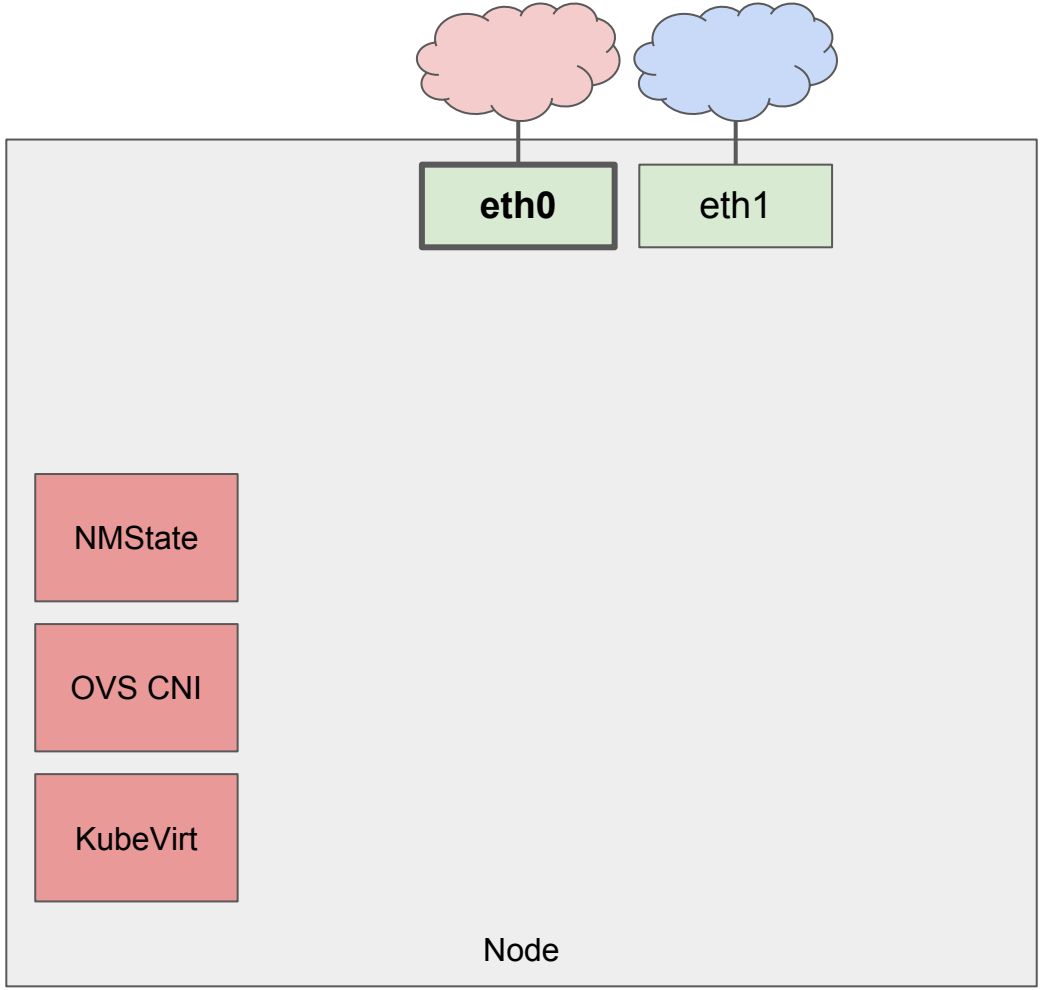
```
  status:
    currentState:
      capabilities: null
      interfaces:
      - ifIndex: 0
        ipv4:
          address:
          - ip: 10.233.64.1
            prefix-length: 24
          enabled: true
        ipv6:
          address:
          - ip: fe80::3889:6aff:fe63:c613
            prefix-length: 64
          enabled: true
        mtu: 1450
        name: cni0
        state: up
        type: unknown
:
```

```
  status:
    currentState:
      capabilities: null
      interfaces:
      - ifIndex: 0
        ipv4:
          address:
          - ip: 10.233.64.1
            prefix-length: 24
          enabled: true
        ipv6:
          address:
          - ip: fe80::3889:6aff:fe63:c613
            prefix-length: 64
          enabled: true
        mtu: 1450
        name: cni0
        state: up
        type: unknown
/eth0
```

```
    name: eth0
    state: up
    type: ethernet
  - ifIndex: 0
    ipv4:
      address:
      - ip: 192.168.122.101
        prefix-length: 24
      enabled: true
    ipv6:
      address:
      - ip: fe80::5054:ff:feab:7a23
        prefix-length: 64
      enabled: true
    mtu: 1500
    name: eth1
    state: up
    type: ethernet
  - ifIndex: 0
:
```

```
    - ifIndex: 0
      ipv4:
        address:
        - ip: 192.168.121.99
          prefix-length: 24
        enabled: true
      ipv6:
        address:
        - ip: fe80::5054:ff:fe2e:f48e
          prefix-length: 64
        enabled: true
      mtu: 1500
      name: eth0
      state: up
      type: ethernet
    - ifIndex: 0
      ipv4:
        address:
        - ip: 192.168.122.101
:
```


eth0
eth1
NMState
OVS CNI
KubeVirt
Node


eth0
eth1
NMState
OVS CNI
KubeVirt
Node

```
# # check current node network status
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
#
```

```
#
```

```
# # configure network attachment definition for secondary pod/vm networks
```

```
# # configure network attachment definition for secondary pod/vm networks
# 
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
#
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: blue-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: ovs-cni.network.kubevirt.io/br1
spec:
  config: '{
      "cniVersion": "0.3.1",
      "type": "ovs",
      "bridge": "br1"
    }'
examples/network-attachment.yml (END)
```

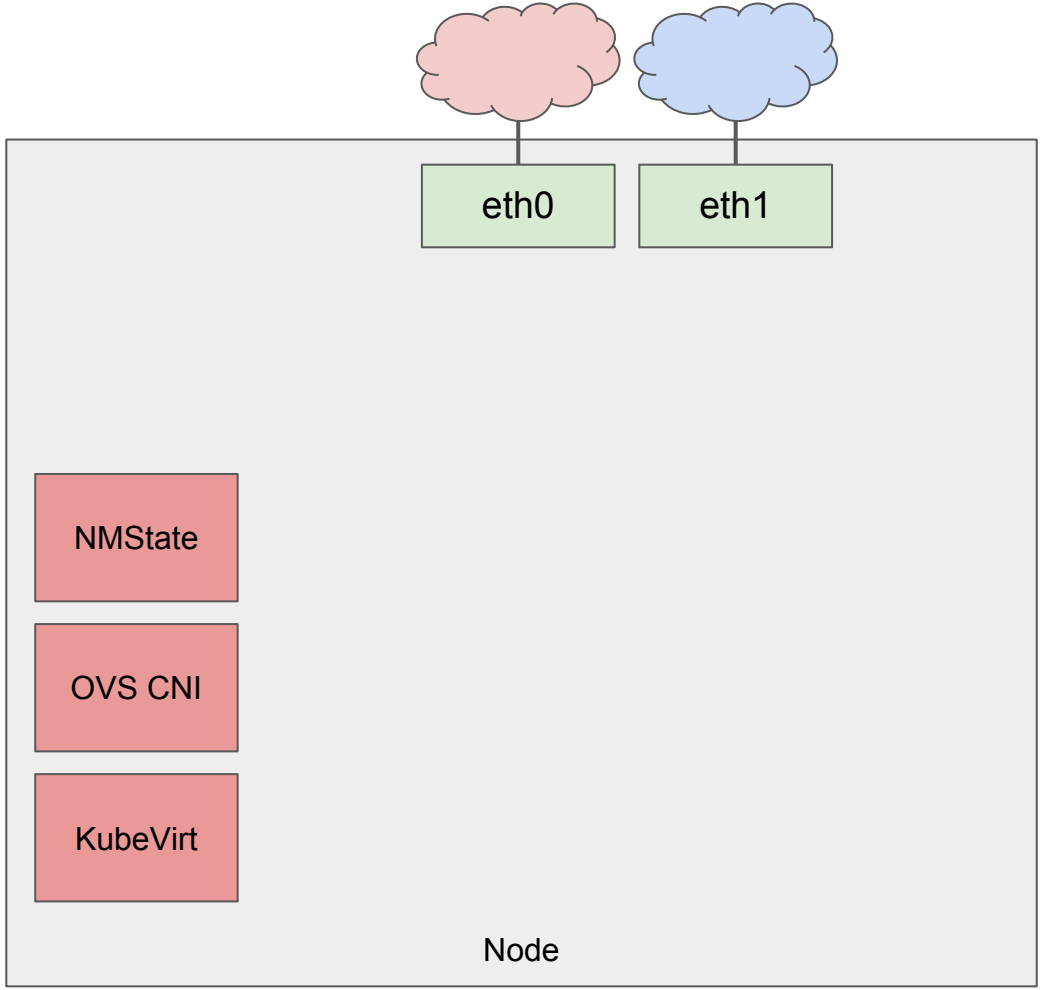
```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
#
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
# kubectl create -f examples/network-attachment.yml
```

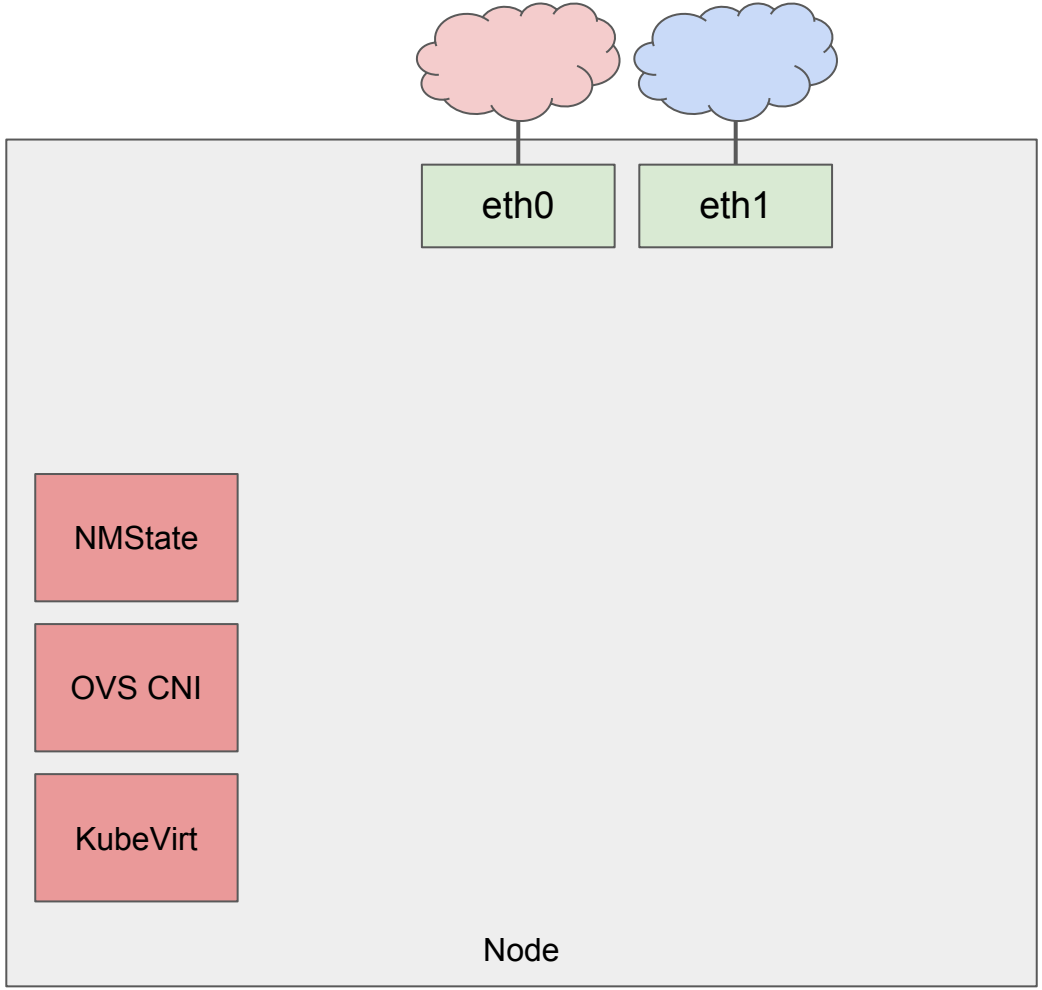
```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
# kubectl create -f examples/network-attachment.yml
networkattachmentdefinition.k8s.cni.cncf.io/blue-network created
#
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
# kubectl create -f examples/network-attachment.yml
networkattachmentdefinition.k8s.cni.cncf.io/blue-network created
# kubectl get network-attachment-definitions
```

```
# # configure network attachment definition for secondary pod/vm networks
# # this attachment will connect pod/vm to OVS bridge br1
# less examples/network-attachment.yml
# kubectl create -f examples/network-attachment.yml
networkattachmentdefinition.k8s.cni.cncf.io/blue-network created
# kubectl get network-attachment-definitions
NAME           AGE
blue-network   6s
#
```


eth0
eth1
NMState
OVS CNI
KubeVirt
Node

```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```



```
#
```

```
# # create a VM connected to the secondary network
```

```
# # create a VM connected to the secondary network
# 
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
```

```
apiVersion: kubevirt.io/v1alpha2
kind: VirtualMachineInstance
metadata:
  name: test-vmi
spec:
  terminationGracePeriodSeconds: 30
  domain:
    resources:
      requests:
        memory: 1024M
    devices:
      disks:
      - name: containerdisk
        volumeName: registryvolume
        disk:
          bus: virtio
      - name: emptydisk
        volumeName: emptydiskvolume
        disk:
```

examples/vmi.yml

```
apiVersion: kubevirt.io/v1alpha2
kind: VirtualMachineInstance
metadata:
  name: test-vmi
spec:
  terminationGracePeriodSeconds: 30
  domain:
    resources:
      requests:
        memory: 1024M
    devices:
      disks:
      - name: containerdisk
        volumeName: registryvolume
        disk:
          bus: virtio
      - name: emptydisk
        volumeName: emptydiskvolume
        disk:
/networks
```

```
  networks:
  - name: default
    pod: {}
  - name: blue-network
    multus:
      networkName: blue-network
  volumes:
  - name: registryvolume
    containerDisk:
      image: kubevirt/fedora-cloud-container-disk-demo:latest
  - name: emptydiskvolume
    emptyDisk:
      capacity: "2Gi"
  - name: cloudinitvolume
    cloudInitNoCloud:
      userData: |-
        #cloud-config
        password: fedora
        chpasswd: { expire: False }
(END)
```

```
  networks:
  - name: default
    pod: {}
  - name: blue-network
    multus:
      networkName: blue-network
  volumes:
  - name: registryvolume
    containerDisk:
      image: kubevirt/fedora-cloud-container-disk-demo:latest
  - name: emptydiskvolume
    emptyDisk:
      capacity: "2Gi"
  - name: cloudinitvolume
    cloudInitNoCloud:
      userData: |-
        #cloud-config
        password: fedora
        chpasswd: { expire: False }
?interfaces
```

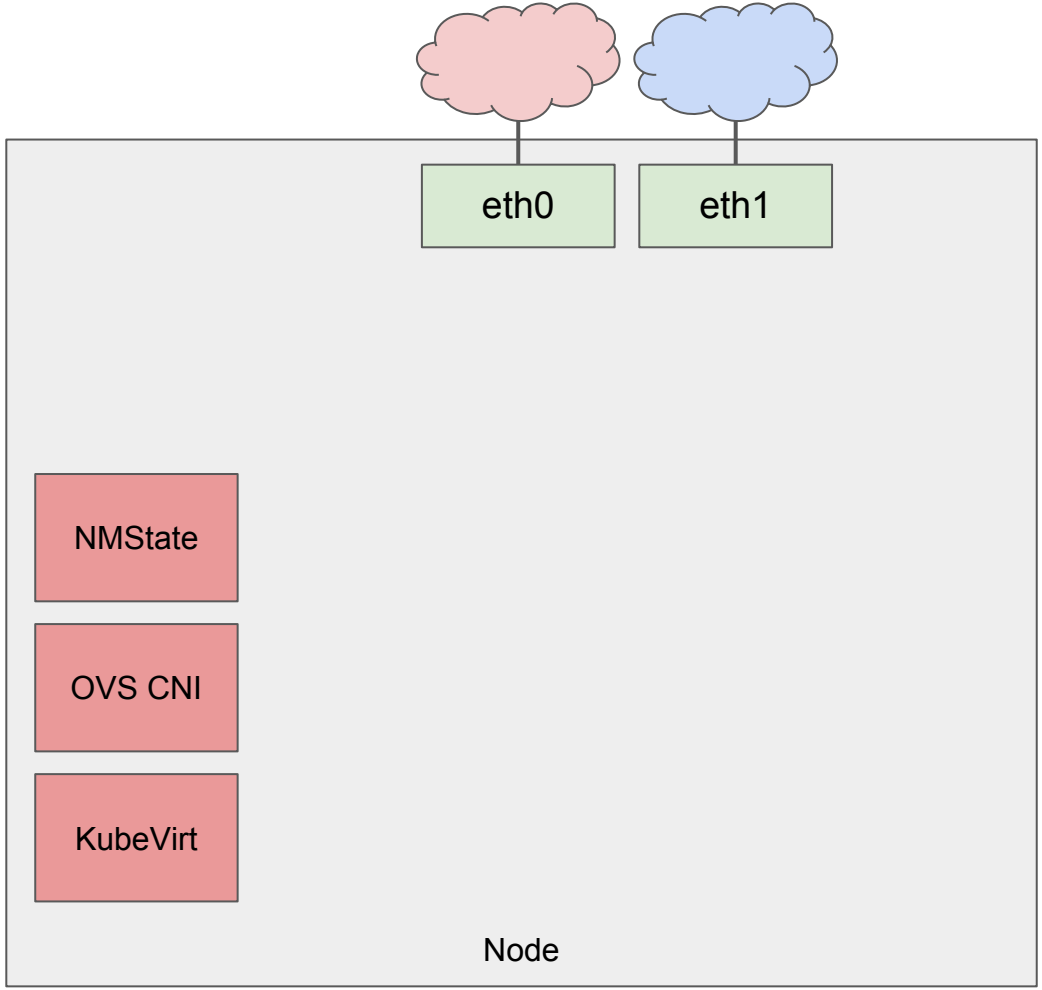
```
      interfaces:
      - bridge: {}
        name: default
      - bridge: {}
        name: blue-network
  networks:
  - name: default
    pod: {}
  - name: blue-network
    multus:
      networkName: blue-network
  volumes:
  - name: registryvolume
    containerDisk:
      image: kubevirt/fedora-cloud-container-disk-demo:latest
  - name: emptydiskvolume
    emptyDisk:
      capacity: "2Gi"
  - name: cloudinitvolume
:
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
#
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
```

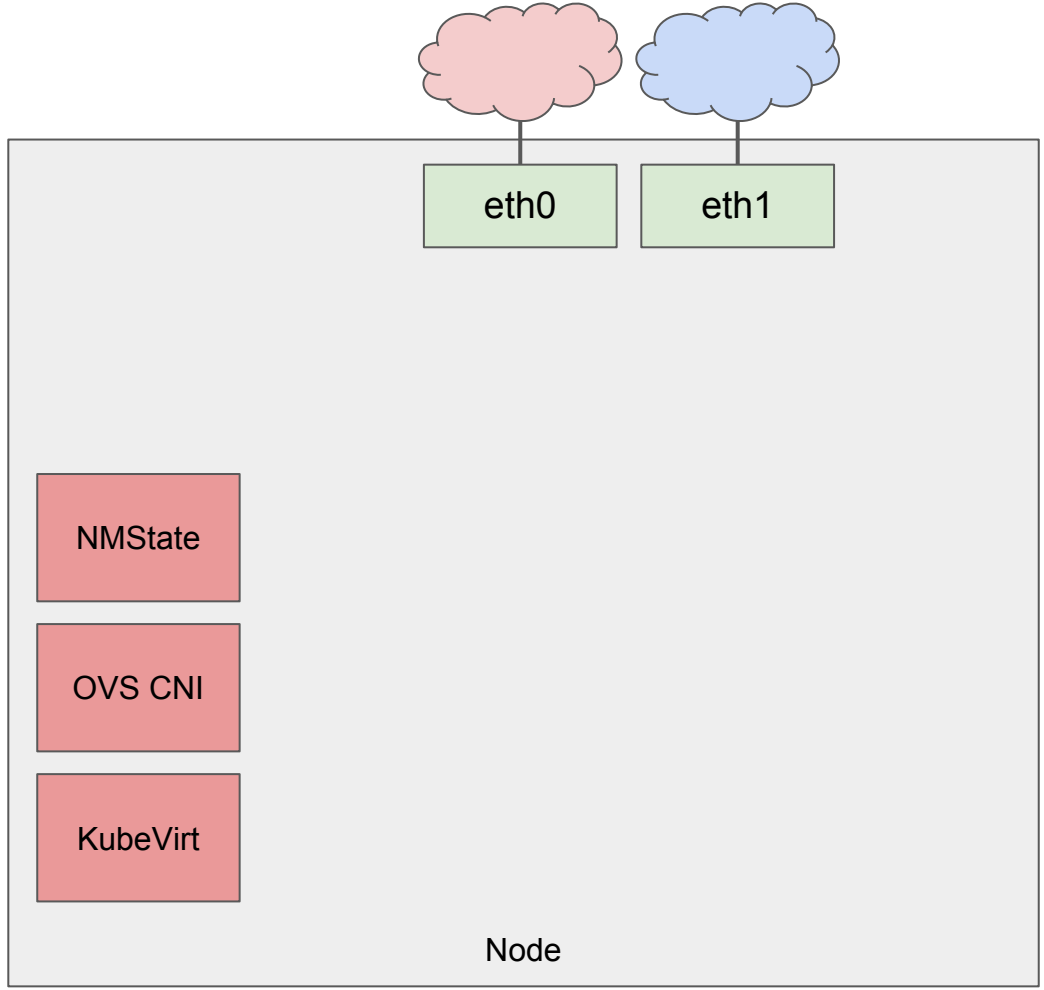
```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
#
```

```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```



```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# 
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   0/2     Pending   0          10s
#
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   0/2     Pending   0          10s
# kubectl describe pod virt-launcher-test-vmi-nqvqs | less
```

```
Name:                virt-launcher-test-vmi-nqvqs
Namespace:           default
Priority:            0
PriorityClassName:   <none>
Node:                <none>
Labels:              kubevirt.io=virt-launcher
                     kubevirt.io/created-by=10e3ee0d-150d-11e9-be67-5254002ef48e
Annotations:         k8s.v1.cni.cncf.io/networks: blue-network
                     kubevirt.io/domain: test-vmi
Status:              Pending
IP:
Controlled By:       VirtualMachineInstance/test-vmi
Containers:
  volumeregistryvolume:
    Image:       kubevirt/fedora-cloud-container-disk-demo:latest
    Port:        <none>
    Host Port:   <none>
    Command:
      /entry-point.sh
:
```

```
  virt-share-dir:
    Type:          HostPath (bare host directory volume)
    Path:          /var/run/kubevirt
    HostPathType:
  libvirt-runtime:
    Type:    EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
  ephemeral-disks:
    Type:        EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
QoS Class:       Burstable
Node-Selectors:  kubevirt.io/schedulable=true
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason            Age                  From               Message
  ----     ------            ----                 ----               -------
  Warning  FailedScheduling  110m (x2 over 110m)  default-scheduler  0/1 nodes areav
ailable: 1 Insufficient ovs-cni.network.kubevirt.io/br1.
(END)
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   0/2     Pending   0          10s
# kubectl describe pod virt-launcher-test-vmi-nqvqs | less
#
```

```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   0/2     Pending   0          10s
# kubectl describe pod virt-launcher-test-vmi-nqvqs | less
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
```

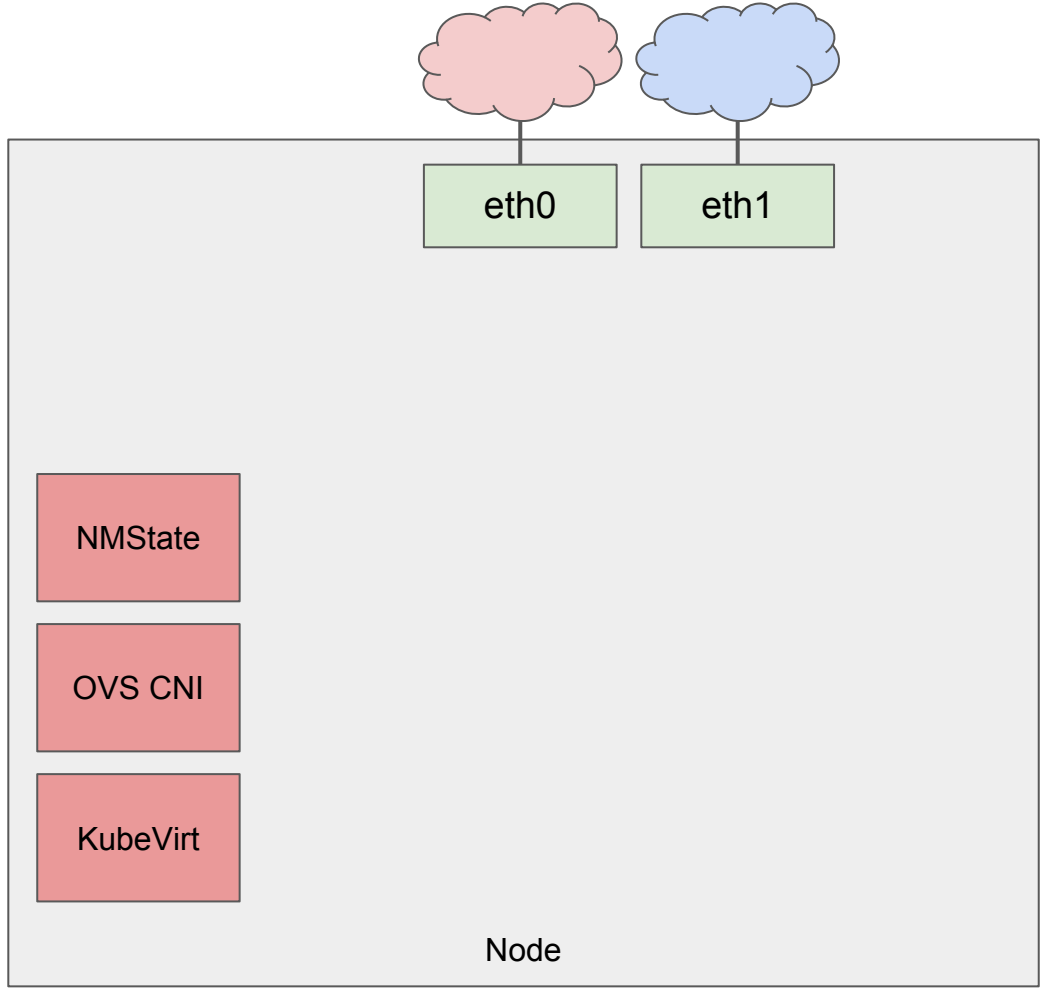
```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 1
    name: 192.168.122.101
    namespace: nmstate-default
    resourceVersion: "1553"
    selfLink: /apis/nmstate.io/v1/namespaces/nmstate-default/nodenetworkstates/19216
8.122.101
    uid: 4c14cd37-150c-11e9-be67-5254002ef48e
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
:
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 1
    name: 192.168.122.101
    namespace: nmstate-default
    resourceVersion: "1553"
    selfLink: /apis/nmstate.io/v1/namespaces/nmstate-default/nodenetworkstates/19216
8.122.101
    uid: 4c14cd37-150c-11e9-be67-5254002ef48e
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
/br1
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 1
    name: 192.168.122.101
    namespace: nmstate-default
    resourceVersion: "1553"
    selfLink: /apis/nmstate.io/v1/namespaces/nmstate-default/nodenetworkstates/192.
168.122.101
    uid: 4c14cd37-150c-11e9-be67-5254002ef48e
  spec:
    desiredState:
      interfaces: null
    managed: true
    nodeName: 192.168.122.101
  status:
Pattern not found  (press RETURN)
```

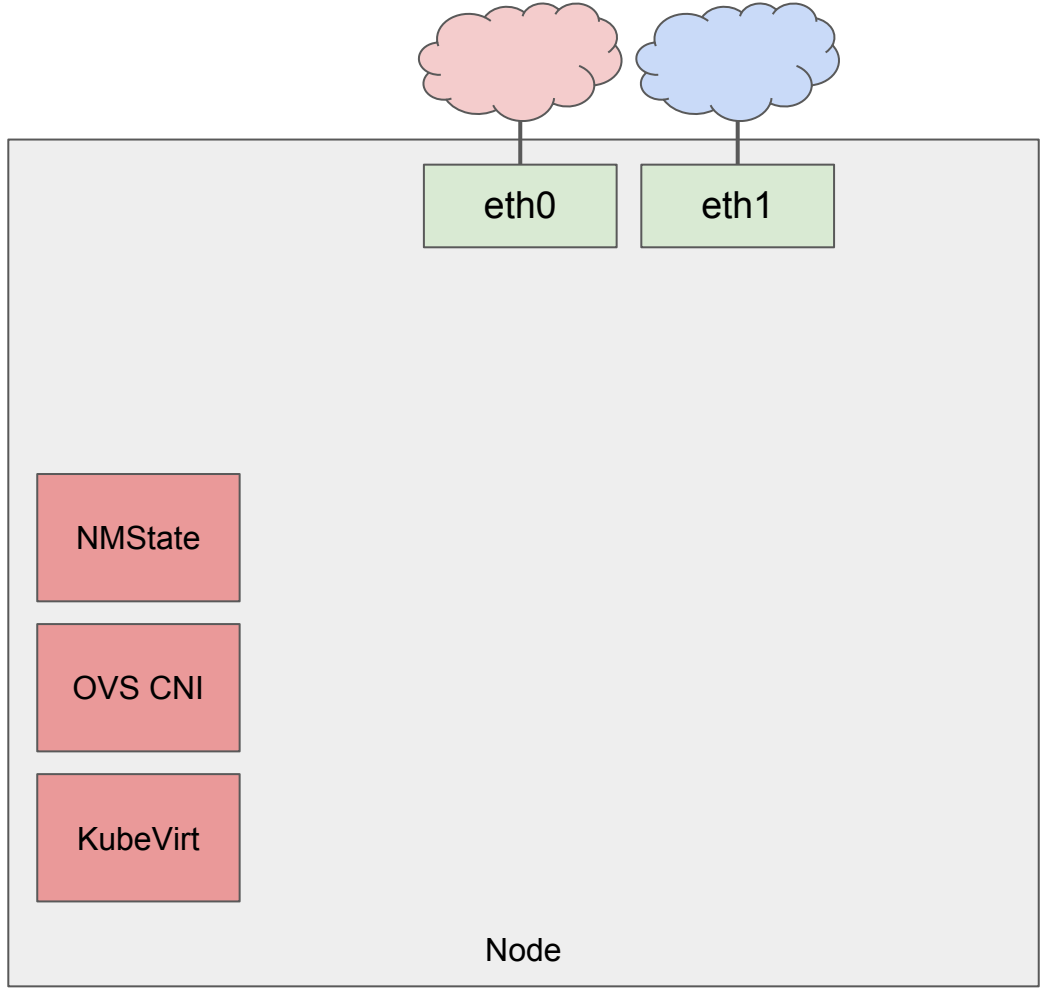
```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```



```
# # create a VM connected to the secondary network
# less examples/vmi.yml
# kubectl create -f examples/vmi.yml
virtualmachineinstance.kubevirt.io/test-vmi created
# kubectl get vmi
NAME       AGE   PHASE        IP    NODENAME
test-vmi   4s    Scheduling
# kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   0/2     Pending   0          10s
# kubectl describe pod virt-launcher-test-vmi-nqvqs | less
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
#
```

```
#
```

```
# # configure required bridge on the node
```

```
# # configure required bridge on the node
#
```

```
# # configure required bridge on the node
# less examples/node-network.yml
```

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  creationTimestamp: null
  name: 192.168.122.101
  namespace: nmstate-default
spec:
  desiredState:
    interfaces:
    - description: OVS bridge for secondary pod networking
      name: br1
      type: ovs-bridge
      state: up
      bridge:
        options:
          fail-mode: ''
          mcast-snooping-enable: false
          rstp: false
          stp: true
```

examples/node-network.yml

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  creationTimestamp: null
  name: 192.168.122.101
  namespace: nmstate-default
spec:
  desiredState:
    interfaces:
    - description: OVS bridge for secondary pod networking
      name: br1
      type: ovs-bridge
      state: up
      bridge:
        options:
          fail-mode: ''
          mcast-snooping-enable: false
          rstp: false
          stp: true
/spec
```

```
spec:
  desiredState:
    interfaces:
    - description: OVS bridge for secondary pod networking
      name: br1
      type: ovs-bridge
      state: up
      bridge:
        options:
          fail-mode: ''
          mcast-snooping-enable: false
          rstp: false
          stp: true
        port:
        - name: eth2
          type: system
      ipv4:
        enabled: false
      ipv6:
:
```

```
        mcast-snooping-enable: false
        rstp: false
        stp: true
      port:
      - name: eth2
        type: system
    ipv4:
      enabled: false
    ipv6:
      enabled: false
  - description: Bridge physical interface
    name: eth2
    type: ethernet
    state: up
    ipv4:
      enabled: false
    ipv6:
      enabled: false
managed: true
:
```

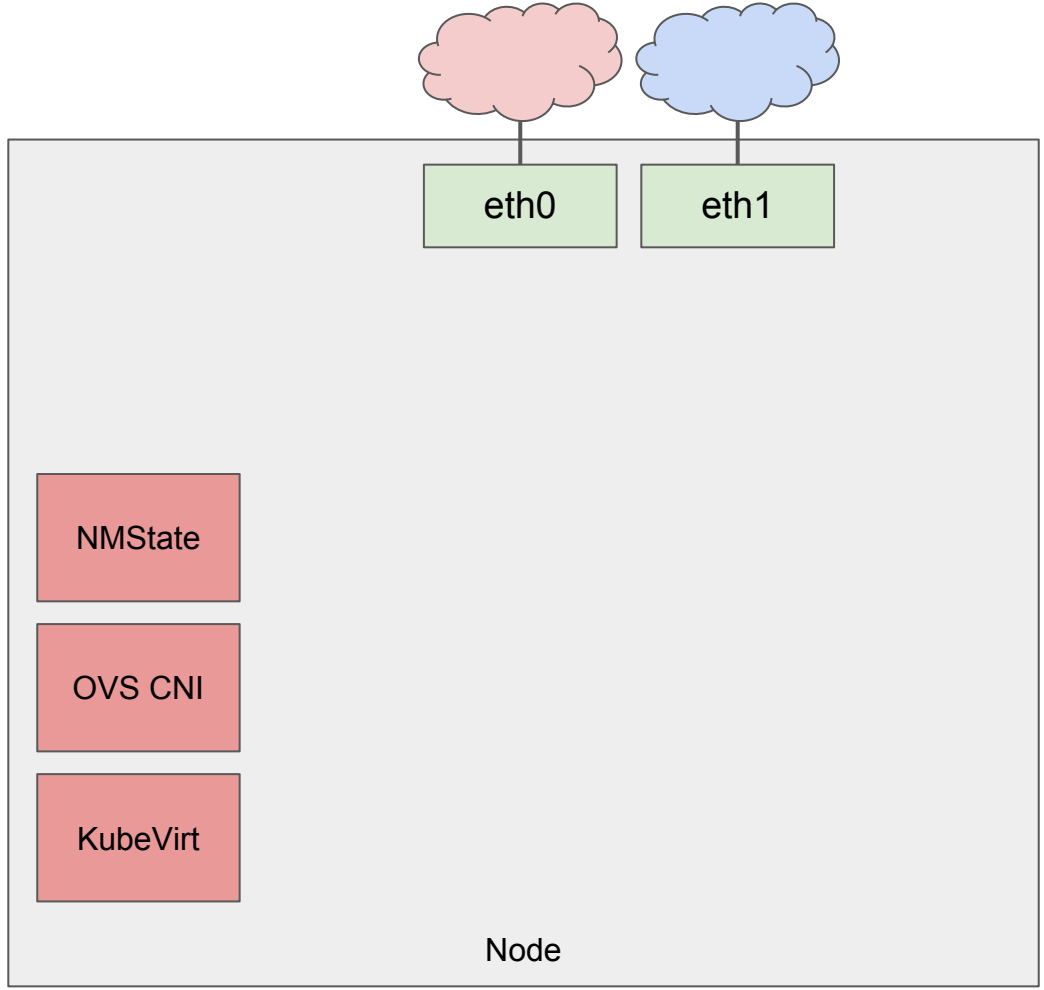
```
# # configure required bridge on the node
# less examples/node-network.yml
#
```

```
# # configure required bridge on the node
# less examples/node-network.yml
# kubectl apply -f examples/node-network.yml
```

```
# # configure required bridge on the node
# less examples/node-network.yml
# kubectl apply -f examples/node-network.yml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
nodenetworkstate.nmstate.io/192.168.122.101 configured
#
```

```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

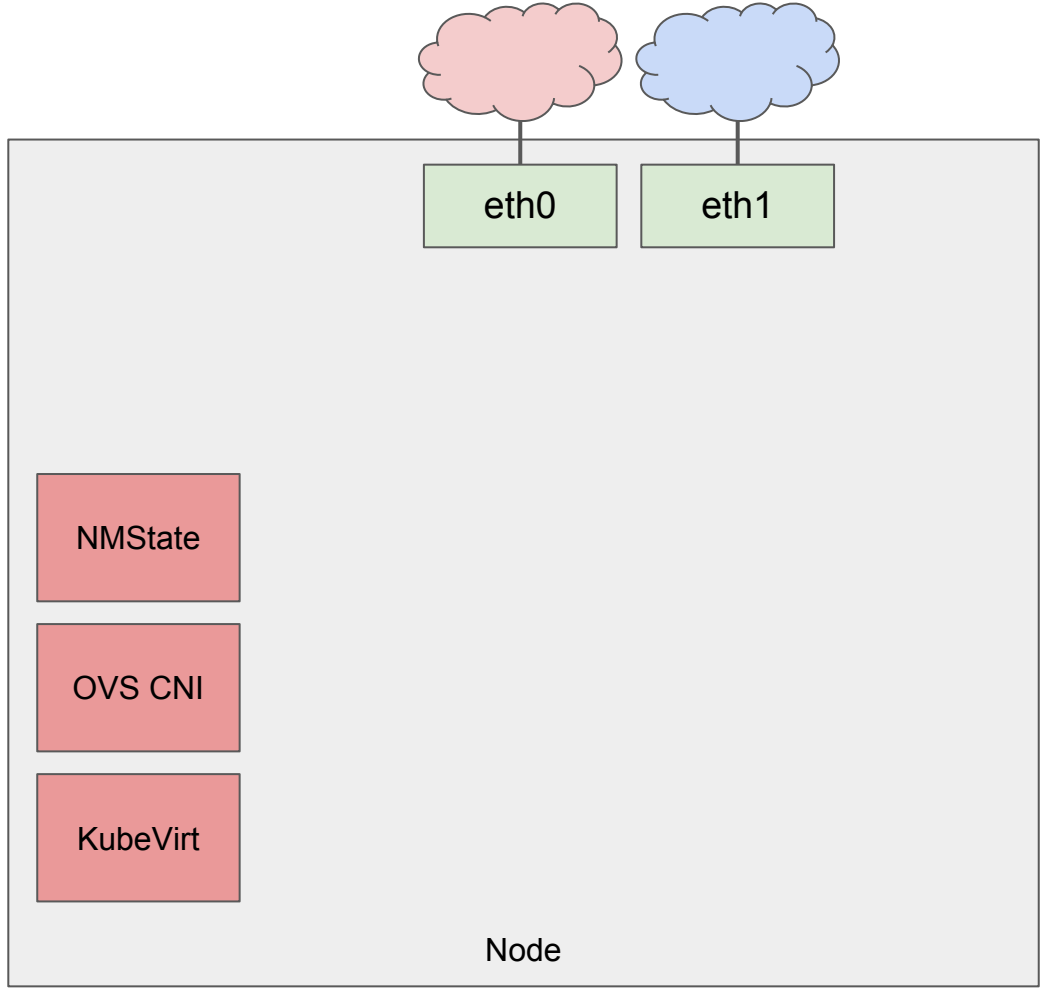
```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
#
```

```
# # see whether the bridge was successfully configured
```

```
# # see whether the bridge was successfully configured
# 
```

```
# # see whether the bridge was successfully configured
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"nmstate.io/v1","kind":"NodeNetworkState","metadata":{"annotti
ons":{},"creationTimestamp":null,"name":"192.168.122.101","namespace":"nmstate-defau
lt"},"spec":{"desiredState":{"interfaces":[{"bridge":{"options":{"fail-mode":"","mca
st-snooping-enable":false,"rstp":false,"stp":true},"port":[{"name":"eth2","type":"sy
stem"}]},"description":"OVS bridge for secondary pod networking","ipv4":{"enabled":f
alse},"ipv6":{"enabled":false},"name":"br1","state":"up","type":"ovs-bridge"},{"desc
ription":"Bridge physical interface","ipv4":{"enabled":false},"ipv6":{"enabled":fals
e},"name":"eth2","state":"up","type":"ethernet"}]},"managed":true,"nodeName":"node1"
},"status":{"currentState":{"capabilities":null,"interfaces":null}}}
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 3
    name: 192.168.122.101
:
```

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkState
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
        {"apiVersion":"nmstate.io/v1","kind":"NodeNetworkState","metadata":{"annotti
ons":{},"creationTimestamp":null,"name":"192.168.122.101","namespace":"nmstate-defau
lt"},"spec":{"desiredState":{"interfaces":[{"bridge":{"options":{"fail-mode":"","mca
st-snooping-enable":false,"rstp":false,"stp":true},"port":[{"name":"eth2","type":"sy
stem"}]},"description":"OVS bridge for secondary pod networking","ipv4":{"enabled":f
alse},"ipv6":{"enabled":false},"name":"br1","state":"up","type":"ovs-bridge"},{"desc
ription":"Bridge physical interface","ipv4":{"enabled":false},"ipv6":{"enabled":fals
e},"name":"eth2","state":"up","type":"ethernet"}]},"managed":true,"nodeName":"node1"
},"status":{"currentState":{"capabilities":null,"interfaces":null}}}
    creationTimestamp: "2019-01-10T19:16:57Z"
    generation: 3
    name: 192.168.122.101
/spec
```

```
spec:
  desiredState:
    interfaces:
    - bridge:
        port:
        - name: eth2
          type: system
      description: OVS bridge for secondary pod networking
      ipv4:
        enabled: false
      ipv6:
        enabled: false
      name: br1
      state: up
      type: ovs-bridge
    - description: Bridge physical interface
      ipv4:
        enabled: false
      ipv6:
:
```

```
  spec:
    desiredState:
      interfaces:
      - bridge:
          port:
          - name: eth2
            type: system
        description: OVS bridge for secondary pod networking
        ipv4:
          enabled: false
        ipv6:
          enabled: false
        name: br1
        state: up
        type: ovs-bridge
      - description: Bridge physical interface
        ipv4:
          enabled: false
        ipv6:
/status
```

```
  status:
    currentState:
      capabilities: null
      interfaces:
      - bridge:
          port:
          - name: eth2
            type: system
        description: OVS bridge for secondary pod networking
        ifIndex: 0
        ipv4:
          enabled: false
        ipv6:
          enabled: false
        mtu: 0
        name: br1
        state: up
        type: ovs-bridge
      - ifIndex: 0
:
```

```
# # see whether the bridge was successfully configured
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
#
```

```
# # see whether the bridge was successfully configured
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
# kubectl get node node1 -o yaml | less
```

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    flannel.alpha.coreos.com/backend-data: '{"VtepMAC":"ca:2d:17:cf:0f:ff"}'
    flannel.alpha.coreos.com/backend-type: vxlan
    flannel.alpha.coreos.com/kube-subnet-manager: "true"
    flannel.alpha.coreos.com/public-ip: 192.168.121.99
    kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
    kubevirt.io/heartbeat: "2019-01-10T19:24:36Z"
    node.alpha.kubernetes.io/ttl: "0"
    volumes.kubernetes.io/controller-managed-attach-detach: "true"
  creationTimestamp: "2019-01-10T19:07:56Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node1
    kubevirt.io/schedulable: "true"
    node-role.kubernetes.io/master: ""
:
```

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    flannel.alpha.coreos.com/backend-data: '{"VtepMAC":"ca:2d:17:cf:0f:ff"}'
    flannel.alpha.coreos.com/backend-type: vxlan
    flannel.alpha.coreos.com/kube-subnet-manager: "true"
    flannel.alpha.coreos.com/public-ip: 192.168.121.99
    kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
    kubevirt.io/heartbeat: "2019-01-10T19:24:36Z"
    node.alpha.kubernetes.io/ttl: "0"
    volumes.kubernetes.io/controller-managed-attach-detach: "true"
  creationTimestamp: "2019-01-10T19:07:56Z"
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/os: linux
    kubernetes.io/hostname: node1
    kubevirt.io/schedulable: "true"
    node-role.kubernetes.io/master: ""
/allocatable
```

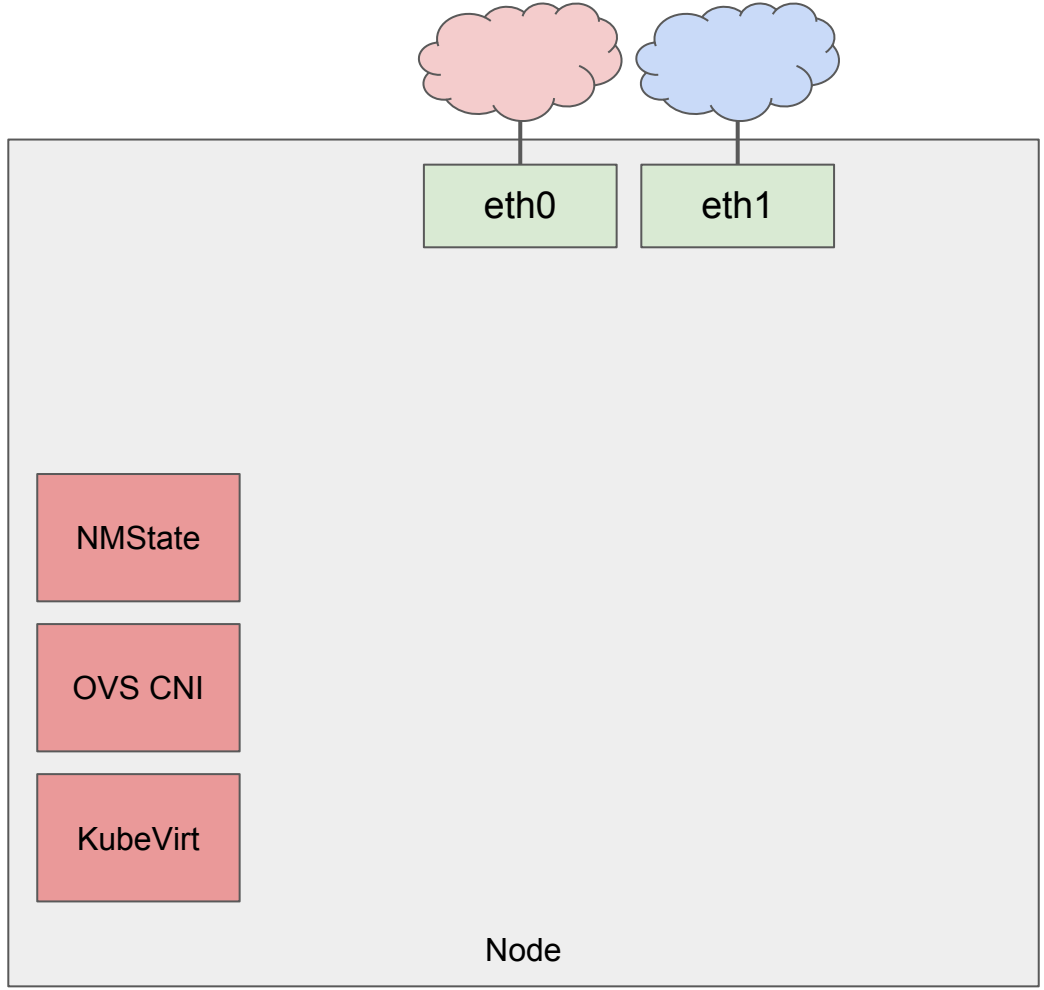
```
  allocatable:
    cpu: 3800m
    devices.kubevirt.io/tun: "110"
    devices.kubevirt.io/vhost-net: "110"
    ephemeral-storage: "37926361435"
    hugepages-1Gi: "0"
    hugepages-2Mi: "0"
    memory: 3278232Ki
    ovs-cni.network.kubevirt.io/: 1k
    ovs-cni.network.kubevirt.io/br1: 1k
    pods: "110"
  capacity:
    cpu: "4"
    devices.kubevirt.io/tun: "110"
    devices.kubevirt.io/vhost-net: "110"
    ephemeral-storage: 41152736Ki
    hugepages-1Gi: "0"
    hugepages-2Mi: "0"
    memory: 3880632Ki
:
```

```
# # see whether the bridge was successfully configured
# kubectl get nodenetworkstates -n nmstate-default -o yaml | less
# kubectl get node node1 -o yaml | less
#
```

```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

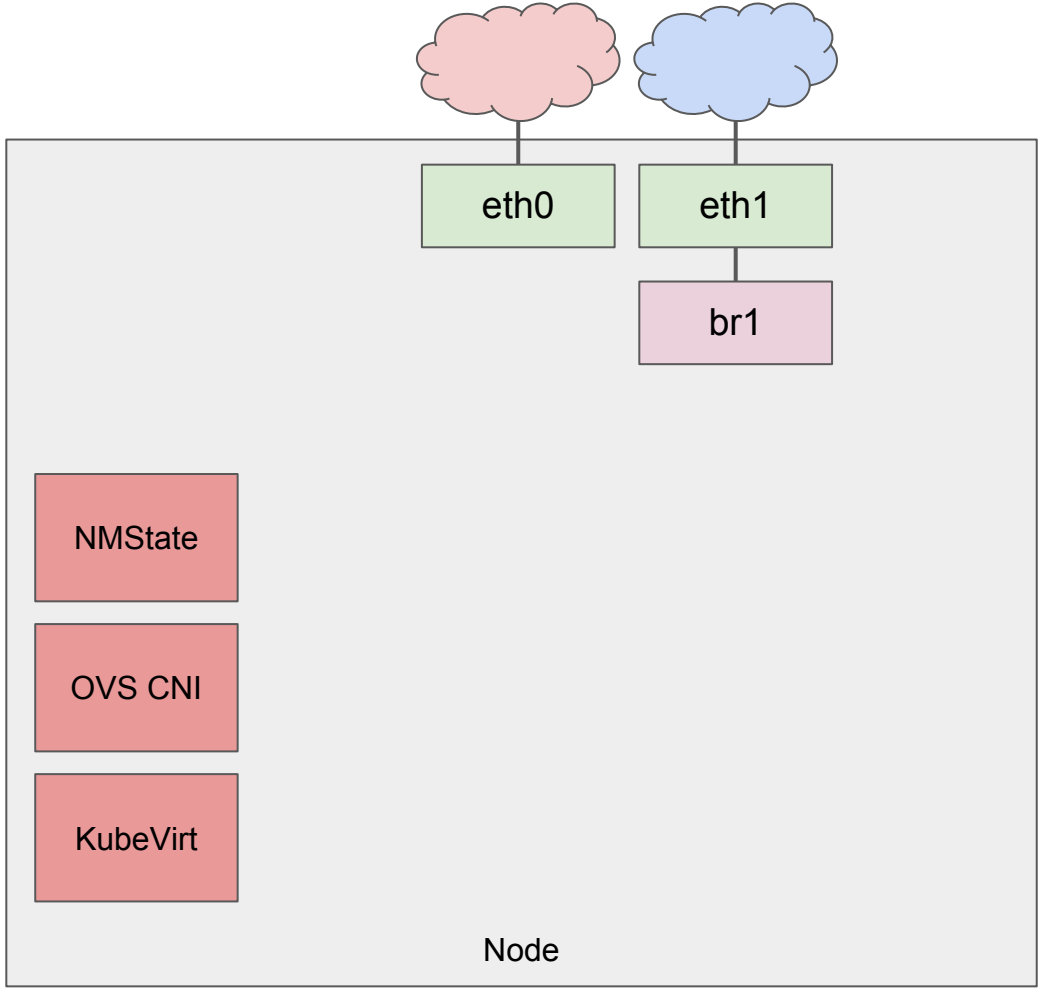
```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
#
```

```
# # see whether VM launcher pod was finally scheduled
```

```
# # see whether VM launcher pod was finally scheduled
# 
```

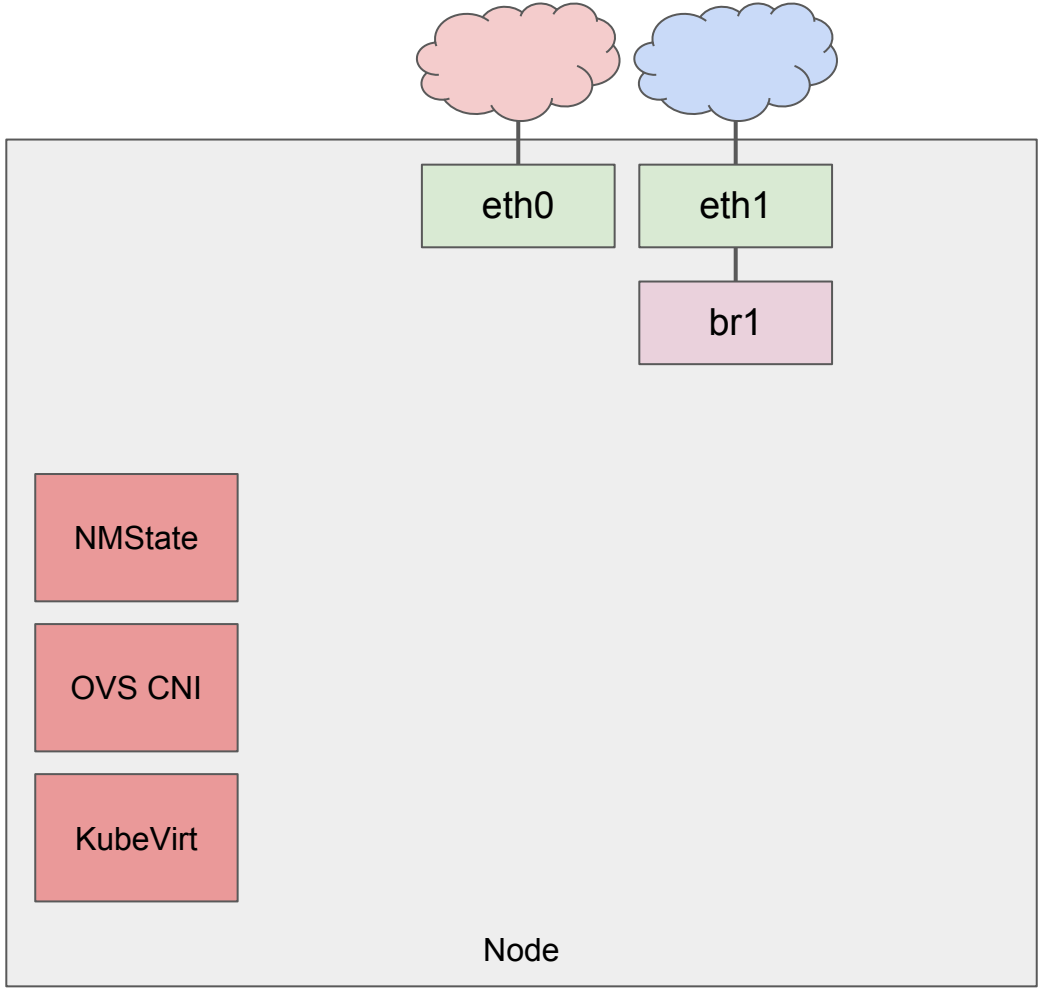
```
# # see whether VM launcher pod was finally scheduled
# kubectl get pods
```

```
# # see whether VM launcher pod was finally scheduled
# kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
virt-launcher-test-vmi-nqvqs   2/2     Running   0          4m24s
#
```

```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

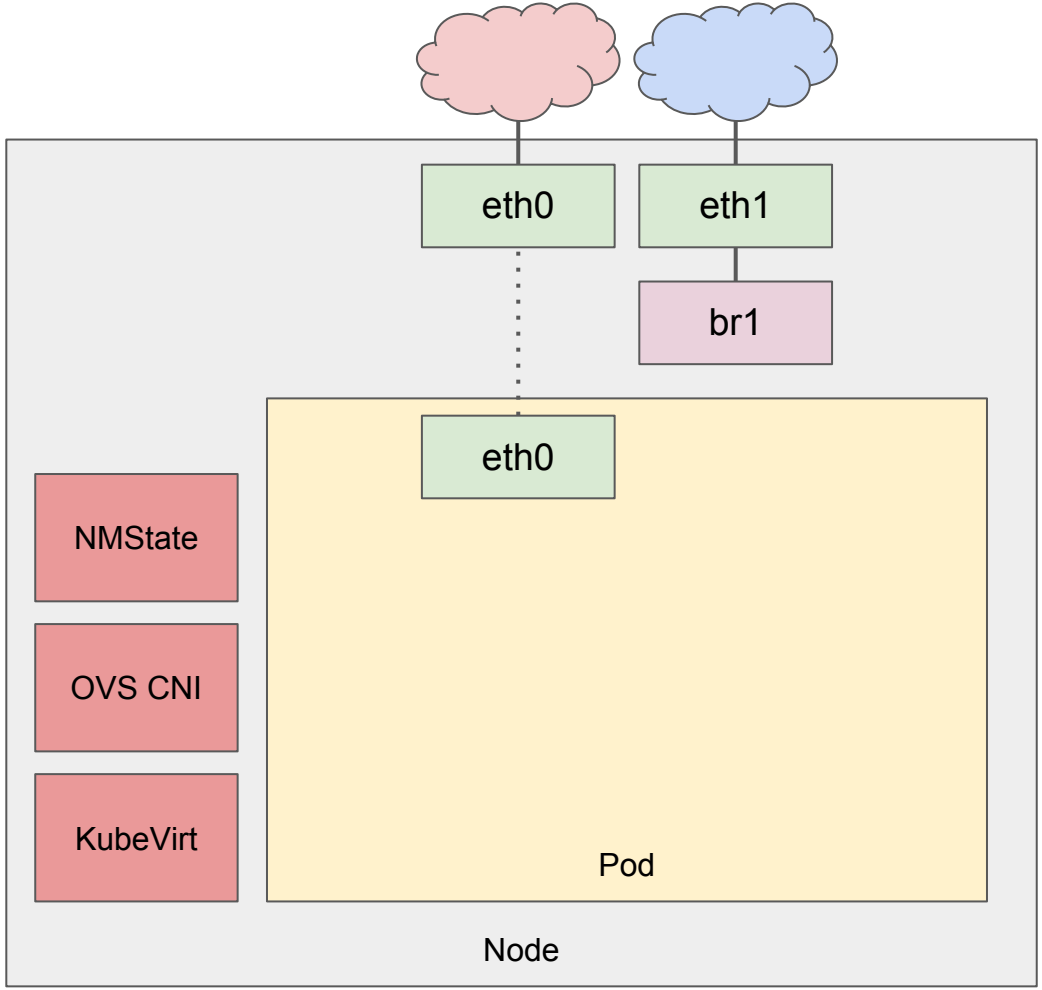
```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

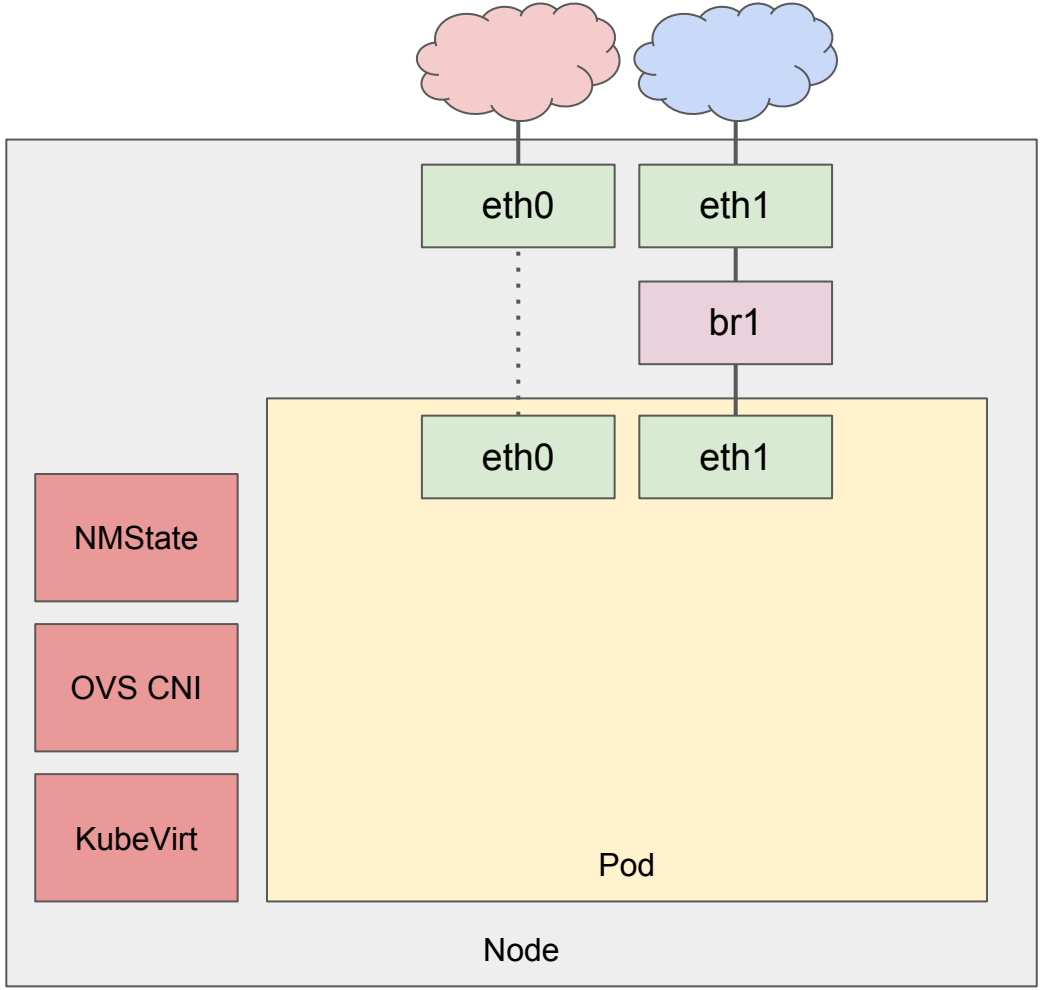
```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

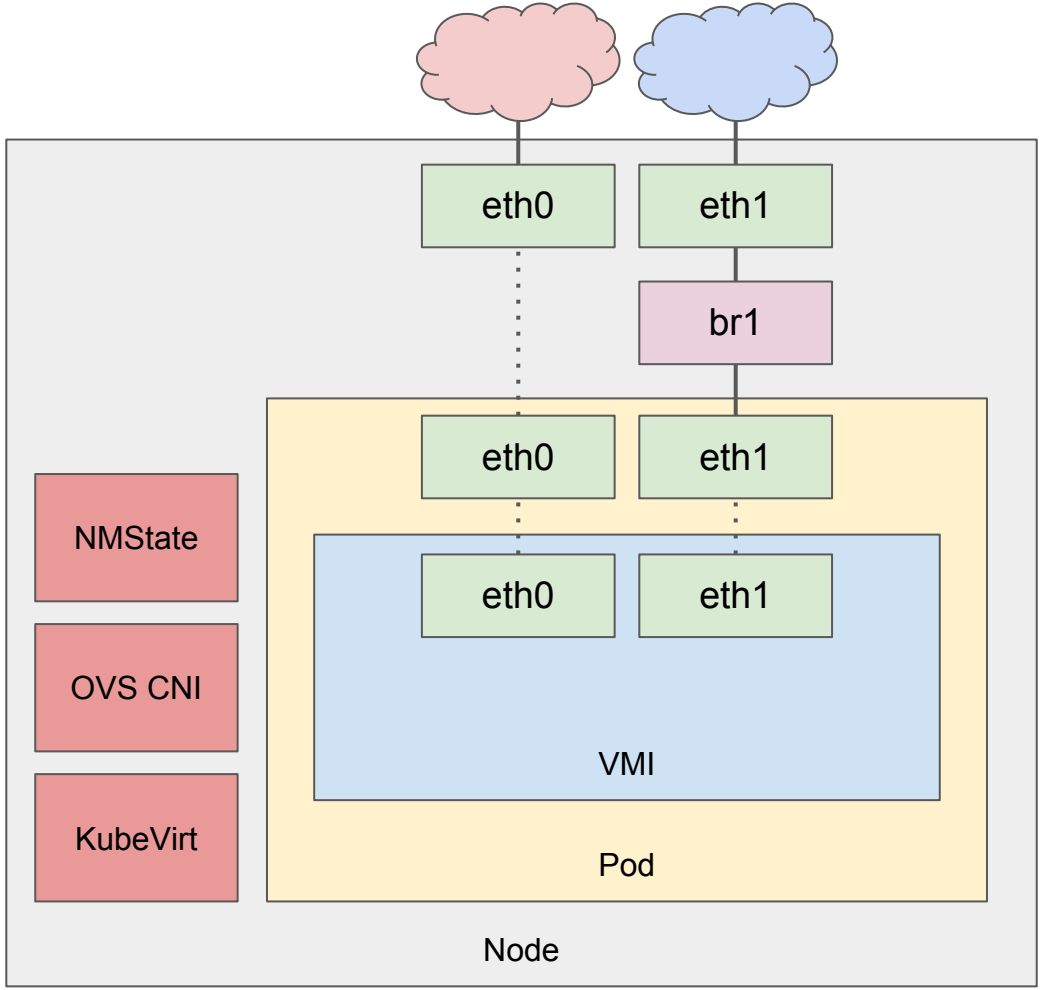
```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
kind:
NetworkAttachmentDefinition
name: blue-network
bridge: br1
```

```
kind: VirtualMachineInstance
name: test-vmi
networks:
- default network
- blue-network
```

```
kind: NodeNetworkState
name: Node
interfaces:
- eth1 as downlink
- br1 as a bridge on top of eth1
```



```
#
```

```
# # check whether the VM was started with secondary interface available
```

```
# # check whether the VM was started with secondary interface available
# 
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
NAME       AGE   PHASE     IP             NODENAME
test-vmi   4m    Running   10.233.64.3    node1
#
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
NAME       AGE   PHASE     IP             NODENAME
test-vmi   4m    Running   10.233.64.3    node1
# virtctl console test-vmi
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
NAME       AGE   PHASE     IP             NODENAME
test-vmi   4m    Running   10.233.64.3   node1
# virtctl console test-vmi
Successfully connected to test-vmi console. The escape sequence is ^]

Fedora 29 (Cloud Edition)
Kernel 4.18.16-300.fc29.x86_64 on an x86_64 (ttyS0)

test-vmi login:
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
NAME       AGE   PHASE     IP             NODENAME
test-vmi   4m    Running   10.233.64.3   node1
# virtctl console test-vmi
Successfully connected to test-vmi console. The escape sequence is ^]

Fedora 29 (Cloud Edition)
Kernel 4.18.16-300.fc29.x86_64 on an x86_64 (ttyS0)

test-vmi login: fedora
```

```
# # check whether the VM was started with secondary interface available
# kubectl get vmi test-vmi
NAME       AGE   PHASE     IP             NODENAME
test-vmi   4m    Running   10.233.64.3   node1
# virtctl console test-vmi
Successfully connected to test-vmi console. The escape sequence is ^]

Fedora 29 (Cloud Edition)
Kernel 4.18.16-300.fc29.x86_64 on an x86_64 (ttyS0)

test-vmi login: fedora
Password:
```

```
[fedora@test-vmi ~]$
```

```
[fedora@test-vmi ~]$ ip address
```

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel state UP group de
fault qlen 1000
    link/ether 0a:58:0a:e9:40:03 brd ff:ff:ff:ff:ff:ff
    inet 10.233.64.3/24 brd 10.233.64.255 scope global dynamic noprefixroute eth0
       valid_lft 86313490sec preferred_lft 86313490sec
    inet6 fe80::858:aff:fee9:4003/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group de
fault qlen 1000
    link/ether 02:00:00:c4:78:95 brd ff:ff:ff:ff:ff:ff
    inet 192.168.123.164/24 brd 192.168.123.255 scope global dynamic noprefixroute e
th1
       valid_lft 3494sec preferred_lft 3494sec
    inet6 fe80::e77a:af1c:7793:7dea/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
[fedora@test-vmi ~]$
```

Takeaways

- Some network workloads might require node to be **configured before the Pod is scheduled** (bridges, SR-IOV, …)

- **Requirements change** and all nodes are not the same

- **kubernetes-nmstate** allows administrator to do day 2 network configuration in declarative and centralized manner

So long
and
thanks for listening!

KubeVirt https://kubevirt.io/

Multus CNI https://github.com/intel/multus-cni

Open vSwitch CNI https://github.com/kubevirt/ovs-cni

nmstate http://nmstate.io

Kubernetes nmstate https://github.com/nmstate/kubernetes-nmstate

Kubernetes Node Network Configuration design document https://tinyurl.com/y7otybg9

Kubernetes Network Custom Resource Definition De-facto Standard design document https://tinyurl.com/y9fxdkfp