

Grok 3

I have a Harvester Cluster, RKE2 up and running and accessible under ip address 10.123.235.200/22. I have also Rancher standalone deployed as well as Argo stack. Everything is running on Intel based CPU (Xeon) with VT-d enabled and ready to use Intel E810 network cards in each node with SRIOV support enabled. I want to deploy a 3 node ephemeral Kubernetes cluster on Virtual Machines provisioned on demand, that uses custom build of DPDK and Ice drivers for E810 hardware network interfaces from host I a way that I could utilize it as Virtual NIC and/or VF driver inside the VM. I need the best known method to do it in an automated way, for example using ansible, helm or just argo. The purpose of this is to test Intel ICE drivers on top of witch MTL and MCM from <https://www.github.com/OpenVisualCloud> are being run.

Deploying an ephemeral 3-node Kubernetes cluster on virtual machines (VMs) within a Harvester cluster, leveraging Intel E810 network cards with SR-IOV and custom DPDK/ICE drivers, is a complex but achievable task. The goal is to automate the provisioning and configuration process using tools like Ansible, Helm, or ArgoCD, ensuring the VMs can utilize the E810's virtual functions (VFs) as virtual NICs for high-performance networking, specifically to test Intel ICE drivers with OpenVisualCloud's MTL and MCM components. Below is a detailed, step-by-step approach to accomplish this, focusing on automation and best practices.

### Overview of the Solution

- Environment:** Harvester (RKE2-based) cluster at `10.123.235.200/22`, Rancher for management, Argo stack for GitOps, and Intel Xeon nodes with E810 NICs (SR-IOV enabled).
- Objective:** Deploy 3 ephemeral Kubernetes nodes on VMs, each utilizing E810 VFs with custom DPDK/ICE drivers for high-performance networking, to test MTL/MCM from OpenVisualCloud.
- Tools:**
  - Harvester:** To provision VMs.
  - Rancher:** For Kubernetes cluster management and Helm chart deployments.
  - ArgoCD:** For GitOps-driven configuration and deployment.
  - Ansible:** For initial VM provisioning and host configuration.
  - Helm:** For deploying Kubernetes components and custom configurations.
  - Custom DPDK/ICE Drivers:** Built from source to support E810 NICs.

Grok 3

- Key Components:**
  - SR-IOV configuration on E810 NICs to expose VFs to VMs.
  - Custom DPDK/ICE drivers installed in VMs for high-performance networking.
  - Kubernetes cluster setup with Multus CNI for multiple network interfaces.
  - OpenVisualCloud MTL/MCM for testing.

### Prerequisites

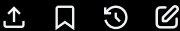
- Hardware:**
  - Intel Xeon-based Harvester nodes with VT-d enabled.
  - Intel E810 NICs with SR-IOV support (firmware updated to the latest version).
  - Ensure E810 NICs are detected (`lspci | grep E810`) and SR-IOV is enabled in BIOS and NIC firmware.
- Software:**
  - Harvester cluster running RKE2, accessible at `10.123.235.200/22`.
  - Rancher standalone deployed for cluster management.
  - ArgoCD installed for GitOps workflows.
  - Ansible installed on a control node (e.g., a local machine or a VM).
  - Helm CLI installed for chart management.
  - Git repository for storing manifests and playbooks.
- Network:**
  - Network range for VMs (e.g., `10.123.236.0/24` within `10.123.235.200/22`).
  - SR-IOV VFs configured on E810 NICs (at least 3 VFs per node for 3 VMs).
  - VLANs configured if needed for isolation.
- Custom Drivers:**
  - Clone the Intel ICE driver repository: `git clone https://github.com/intel/ethernet-linux-ice`.
  - Clone the DPDK repository: `git clone https://github.com/DPDK/dpdk`.
  - Clone OpenVisualCloud repository: `git clone https://github.com/OpenVisualCloud/Media-Transport-Library` (for MTL/MCM).

### Step-by-Step Deployment

Step 1: Configure SR-IOV on E810 NICs

✕ Enable SR-IOV on E810 NICs:

Grok 3



- On each Harvester node, verify SR-IOV support:

```
lspci | grep E810
bash
```

- Enable SR-IOV in the NIC firmware (using `ethtool` or Intel's `ice` tools):

```
echo 8 > /sys/class/net/<nic>/device/sriov_numvfs
bash
```

Replace `<nic>` with the E810 interface (e.g., `enp1s0f0`). This creates 8 VFs per NIC (adjust as needed).

• Bind VFs to VFIO-PCI:

- Identify VFs:

```
lspci | grep Virtual
bash
```

- Bind VFs to `vfio-pci` for passthrough to VMs:

```
modprobe vfio-pci
for vf in $(ls /sys/bus/pci/devices/<nic-pci-addr>/virtfn*); do
    echo $(basename $(readlink $vf)) > /sys/bus/pci/drivers/vfio-pci/new_id
done
bash
```

Replace `<nic-pci-addr>` with the PCI address of the E810 NIC (e.g., `0000:01:00.0`).

• Automate SR-IOV Configuration with Ansible:

- Create an Ansible playbook ( `sriov_setup.yml` ) to configure SR-IOV on all Harvester nodes:

```
- name: Configure SR-IOV on E810 NICs
  hosts: harvester_nodes
  tasks:
    - name: Enable SR-IOV VFs
      shell: echo 8 > /sys/class/net/{{ nic }}/device/sriov_numvfs
      vars:
        nic: enp1s0f0
      ignore_errors: yes
    - name: Load vfio-pci module
```



modprobe:

Grok 3



```
name: vfio-pci
- name: Bind VFs to vfio-pci
  shell: |
    for vf in $(ls /sys/bus/pci/devices/{{ nic_pci_addr }}/virtfn*); do
        echo $(basename $(readlink $vf)) > /sys/bus/pci/drivers/vfio-pci/new_id
    done
  vars:
    nic_pci_addr: 0000:01:00.0
```

- Run the playbook:

```
ansible-playbook -i inventory sriov_setup.yml
bash
```

Step 2: Provision VMs on Harvester

• Create VM Templates in Harvester:

- Use Harvester's UI or API to create a VM template (e.g., Ubuntu 22.04):
  - CPU: 4 vCPUs
  - Memory: 8GB
  - Disk: 40GB
  - Network: 1 management NIC (for SSH/control) + 1 SR-IOV VF (passed through).
- Attach an E810 VF to each VM using Harvester's PCI passthrough feature:
  - In the Harvester UI, add a PCI device (select the VF's PCI address).
  - Alternatively, use a Harvester `VirtualMachine` manifest:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: k8s-node-{{ item }}
  namespace: default
spec:
  running: true
  template:
    spec:
      domain:
        cpu:
          cores: 4
        memory:
          guest: 8Gi
        devices:
          disks:
            - disk:
                bus: virtio
                name: rootdisk
```



```
- disk:
  bus: virtio
  name: cloudinitdisk
  interfaces:
    - name: mgmt
      masquerade: {}
    - name: sriov
      sriov: {}
  hostDevices:
    - deviceName: e810-vf
      name: e810-vf
  volumes:
    - name: rootdisk
      containerDisk:
        image: ubuntu:22.04
    - name: cloudinitdisk
      cloudInitNoCloud:
        userData: |
          #cloud-config
          password: ubuntu
          chpasswd: { expire: False }
          ssh_pwauth: True
```



- Save as `vm-template.yaml` and apply for each node:

```
for i in {1..3}; do sed "s/{{ item }}/$i/g" vm-template.yaml | kubectl apply -f -;
bash
```

#### • Automate VM Provisioning with Ansible:

yaml

- Use Ansible to interact with Harvester's API (via `kubevirt` module or `uri` for REST API) to deploy 3 VMs.
- Example playbook (`provision_vms.yaml`):

```
- name: Provision VMs on Harvester
  hosts: localhost
  tasks:
    - name: Create Kubernetes VMs
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('template', 'vm-template.yaml') | from_yaml }}"
        loop: "{{ range(1,4) | list }}"
      vars:
        item: "{{ item }}"
```

- Run:

yaml



ansible-playbook provision\_vms.yaml

Grok 3

bash



### Step 3: Install Custom DPDK/ICE Drivers in VMs

#### • Build Custom ICE Driver:

- On a build machine, clone and compile the ICE driver with ADQ support:

```
git clone https://github.com/intel/ethernet-linux-ice
cd ethernet-linux-ice
make ADQ=1
```

bash

- Package the driver (`ice.ko`) and dependencies into a tarball:

```
tar -czf ice-driver.tar.gz out/ice.ko
```

bash

#### • Build Custom DPDK:

- Clone and build DPDK with E810 support:

```
git clone https://github.com/DPDK/dpdk
cd dpdk
meson build
ninja -C build
```

bash

- Package the DPDK libraries and binaries:

```
tar -czf dpdk.tar.gz build/
```

bash

#### • Install Drivers in VMs:

- Create an Ansible playbook (`install_drivers.yaml`) to copy and install drivers:

```
- name: Install custom DPDK/ICE drivers in VMs
  hosts: k8s_nodes
  tasks:
    - name: Install dependencies
      apt:
        name: "{{ packages }}"
        state: present
      vars:
        packages:
```



```
- build-essential
- linux-headers-{{ ansible_kernel }}
- libnuma-dev

- name: Copy ICE driver
  copy:
    src: ice-driver.tar.gz
    dest: /tmp/ice-driver.tar.gz
- name: Extract and install ICE driver
  shell: |
    tar -xzf /tmp/ice-driver.tar.gz -C /tmp
    insmod /tmp/out/ice.ko
- name: Copy DPDK
  copy:
    src: dpdk.tar.gz
    dest: /tmp/dpdk.tar.gz
- name: Extract and install DPDK
  shell: |
    tar -xzf /tmp/dpdk.tar.gz -C /opt
    ldconfig
- name: Bind E810 VF to DPDK
  shell: |
    /opt/dpdk/build/bin/dpdk-devbind.py --bind=vfio-pci <vf-pci-addr>
vars:
  vf_pci_addr: "{{ ansible_facts['pci']['e810_vf'] }}"
```



- Inventory file (`inventory`):

```
[k8s_nodes]
k8s-node-1 ansible_host=10.123.236.1
k8s-node-2 ansible_host=10.123.236.2
k8s-node-3 ansible_host=10.123.236.3
```

```
ini
```



- Run:

```
ansible-playbook -i inventory install_drivers.yml
bash
```



#### Step 4: Deploy Kubernetes Cluster with Multus CNI

- Bootstrap Kubernetes with Kubeadm:

- Use Ansible to install `kubeadm`, `kubectl`, and `kubelet` on VMs and initialize the cluster:

```
- name: Bootstrap Kubernetes cluster
  hosts: k8s_nodes
  tasks:
    - name: Install Kubernetes components
      apt:
```



```
name: "{{ packages }}"
state: present
vars:
  packages:
    - kubeadm
    - kubectl
    - kubelet
- name: Initialize control plane (on first node)
  shell: |
    kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=10
  when: inventory_hostname == 'k8s-node-1'
- name: Copy kubeconfig
  fetch:
    src: /etc/kubernetes/admin.conf
    dest: ./kubeconfig
    flat: yes
  when: inventory_hostname == 'k8s-node-1'
- name: Join worker nodes
  shell: |
    {{ hostvars['k8s-node-1']['kubeadm_join_command'] }}
  when: inventory_hostname != 'k8s-node-1'
```

Grok 3



- Run:

```
ansible-playbook bootstrap_k8s.yml
bash
```



- Deploy Multus CNI with SR-IOV:

- Install Multus CNI to support multiple network interfaces, including SR-IOV VFs:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master
bash
```



- Deploy SR-IOV CNI and device plugin:

```
kubectl apply -f https://raw.githubusercontent.com/intel/sriov-cni/master/images/sriov
kubectl apply -f https://raw.githubusercontent.com/intel/sriov-network-device-plugin/m
bash
```



- Configure Multus to use SR-IOV VFs:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
```



```
metadata:
  name: sriov-net
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "sriov",
    "deviceID": "<vf-pci-addr>",
    "vf": 0,
    "ipam": {
      "type": "host-local",
      "subnet": "10.123.237.0/24",
      "rangeStart": "10.123.237.10",
      "rangeEnd": "10.123.237.100"
    }
  }'
```

Grok 3



Replace `<vf-pci-addr>` with the VF's PCI address. Apply:

```
yaml
```

```
kubectl apply -f sriov-net.yaml
```

```
bash
```

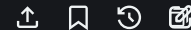
#### • Automate with ArgoCD:

- Create a Git repository with the above manifests (e.g., `multus.yaml`, `sriov-cni.yaml`, `sriovdp.yaml`, `sriov-net.yaml`).
- Define an ArgoCD `Application` to deploy the CNI stack:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: k8s-cni
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/your-repo/k8s-config
    targetRevision: HEAD
    path: cni
  destination:
    server: https://10.123.236.1:6443
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```



• Apply:  
yaml Grok 3



```
kubectl apply -f argocd-app.yaml
bash
```

#### Step 5: Deploy OpenVisualCloud MTL/MCM

##### • Install MTL/MCM:

- Clone the OpenVisualCloud Media-Transport-Library repository in each VM:

```
git clone https://github.com/OpenVisualCloud/Media-Transport-Library
cd Media-Transport-Library
./build.sh
./install.sh
```

```
bash
```

##### • Automate with Ansible:

```
- name: Install OpenVisualCloud MTL/MCM
  hosts: k8s_nodes
  tasks:
    - name: Clone MTL repository
      git:
        repo: https://github.com/OpenVisualCloud/Media-Transport-Library
        dest: /opt/mtl
    - name: Build and install MTL
      shell: |
        cd /opt/mtl
        ./build.sh
        ./install.sh
```

• Run:  
yaml

```
ansible-playbook install_mtl.yml
bash
```

##### • Deploy Test Workloads:

- Create a Helm chart for MTL/MCM workloads, specifying SR-IOV network interfaces:

```
apiVersion: v1
kind: Pod
metadata:
  name: mtl-test
  annotations:
```



- ✕

• Break playbooks into roles (e.g., `sriov`, `vm_provision`, `k8s_bootstrap`, `drivers`, `mtl`) for reusability.

Grok 3

↑

🔖

↺

🔗
- **Immutable Infrastructure:**

• Treat VMs as ephemeral and recreate them for each test cycle to ensure consistency.
- **Monitoring:**

• Deploy Prometheus/Grafana via Rancher to monitor NIC performance and MTL/MCM metrics.
- **Security:**

• Use RBAC in Rancher/ArgoCD to restrict access.

• Isolate SR-IOV networks using VLANs.

## Challenges and Mitigations

- **SR-IOV VF Passthrough:**

• **Challenge:** Harvester’s PCI passthrough may require manual VF assignment.

• **Mitigation:** Use Ansible to automate VF binding and verify passthrough in Harvester.
- **Custom Driver Compatibility:**

• **Challenge:** Custom ICE/DPDK builds may conflict with VM kernel versions.

• **Mitigation:** Use a consistent Ubuntu version (e.g., 22.04) and test drivers in a staging VM first.
- **Ephemeral Cluster Teardown:**

• **Challenge:** Residual configurations may persist.

• **Mitigation:** Use Ansible’s `reset-cluster.yml` to clean up VMs and Kubernetes.

## References

- Intel ICE Driver: <https://github.com/intel/ethernet-linux-ice>(<https://github.com/intel/ethernet-linux-ice>)

• DPDK: <https://github.com/DPDK/dpdk>

• OpenVisualCloud MTL: <https://github.com/OpenVisualCloud/Media-Transport-Library>

• Multus CNI: <https://github.com/k8snetworkplumbingwg/multus-cni>([https://github.com/intel/container-experience-kits-demo-area/blob/master/docs/nfv-](https://github.com/intel/container-experience-kits-demo-area/blob/master/docs/nfv-features-in-k8s/README.md)

- features-in-k8s/README.md)

✕

SR-IOV CNI: <https://github.com/intel/sriov-cni>(<https://github.com/intel/container-experience-kits-demo-area/blob/master/docs/nfv-features-in-k8s/README.md>)

Grok 3

↑

🔖

↺

🔗
- Harvester Documentation: <https://docs.harvesterhci.io/>

• ArgoCD Documentation: <https://argoproj.github.io/argo-cd/>

This approach leverages ArgoCD for GitOps, Ansible for initial provisioning, and Helm for Kubernetes deployments, ensuring an automated, reproducible pipeline for testing Intel ICE drivers with MTL/MCM. Let me know if you need further clarification or assistance with specific components!

15 web pages



Reiterate above, but focus on NIC part utilizing Intel Ethernet Operator and/or SR-IOV Network Device Plugin for Kubernetes.

Below is a focused reiteration of the deployment process for an ephemeral 3-node Kubernetes cluster on Harvester-provisioned VMs, emphasizing the configuration and utilization of Intel E810 network interface cards (NICs) with SR-IOV, leveraging the **Intel Ethernet Operator** and **SR-IOV Network Device Plugin for Kubernetes**. The goal remains to test Intel ICE drivers with OpenVisualCloud’s MTL and MCM components, using automation tools like Ansible, Helm, and ArgoCD. This response prioritizes the NIC-related setup while summarizing other steps for brevity.

## Overview

- **Environment:** Harvester (RKE2) cluster at `10.123.235.200/22` , Rancher standalone, ArgoCD, Intel Xeon nodes with E810 NICs (SR-IOV enabled).

• **Objective:** Deploy a 3-node ephemeral Kubernetes cluster on VMs, using E810 NICs’ virtual functions (VFs) with custom DPDK/ICE drivers, managed via Intel Ethernet Operator and SR-IOV Network Device Plugin, to test MTL/MCM.

• **Focus:** Automate NIC configuration, VF passthrough, and Kubernetes networking for high-performance workloads.

• **Tools:**

- ✕

Grok 3

↑

🔖

🔄

📄
- **Intel Ethernet Operator:** Manages E810 NIC firmware, drivers, and SR-IOV configurations.
  - **SR-IOV Network Device Plugin:** Exposes VFs as Kubernetes resources.
  - **Ansible:** For host and VM setup.
  - **Helm/ArgoCD:** For Kubernetes deployments and GitOps.
  - **Multus CNI:** For multiple network interfaces.

## Prerequisites

- **Hardware:**
  - Intel E810 NICs with SR-IOV enabled in firmware and BIOS (VT-d enabled).
  - Verify NICs: `lspci | grep E810`.
- **Software:**
  - Harvester cluster (RKE2) with Rancher and ArgoCD.
  - Ansible, Helm, and Git for automation.
  - Custom ICE driver: `git clone https://github.com/intel/ethernet-linux-ice`.
  - Custom DPDK: `git clone https://github.com/DPDK/dpdk`.
  - OpenVisualCloud MTL: `git clone https://github.com/OpenVisualCloud/Media-Transport-Library`.
- **Network:**
  - VM subnet (e.g., `10.123.236.0/24`).
  - At least 3 VFs per E810 NIC for 3 VMs.

## Step-by-Step Deployment

### Step 1: Configure E810 NICs with Intel Ethernet Operator

- **Enable SR-IOV on E810 NICs:**
  - On Harvester nodes, enable SR-IOV (creates 8 VFs per NIC):

```
echo 8 > /sys/class/net/<nic>/device/sriov_numvfs
```

```
bash
```

Replace `<nic>` (e.g., `enp1s0f0`). Verify VFs:

- ✕

bash

Grok 3

↑

🔖

🔄

📄
- **Deploy Intel Ethernet Operator:**
    - The Intel Ethernet Operator automates E810 NIC configuration, including firmware updates, driver management, and SR-IOV VF allocation.
    - Clone the operator repository:

```
git clone https://github.com/intel/ethernet-operator
```

```
cd ethernet-operator
```

```
bash
```

- Deploy the operator using Helm (preferred for Kubernetes integration):

```
helm install ethernet-operator ./helm/ethernet-operator \
```

```
--namespace ethernet-operator --create-namespace \
```

```
--set e810Support=true
```

```
bash
```

- Alternatively, use ArgoCD to deploy via a Git repository:

```
apiVersion: argoproj.io/v1alpha1
```

```
kind: Application
```

```
metadata:
```

```
  name: ethernet-operator
```

```
  namespace: argocd
```

```
spec:
```

```
  project: default
```

```
  source:
```

```
    repoURL: https://github.com/your-repo/ethernet-operator
```

```
    targetRevision: HEAD
```

```
    path: helm/ethernet-operator
```

```
    helm:
```

```
      parameters:
```

```
        - name: e810Support
```

```
          value: "true"
```

```
  destination:
```

```
    server: https://kubernetes.default.svc
```

```
    namespace: ethernet-operator
```

```
syncPolicy:
```

```
  automated:
```

```
    prune: true
```

```
    selfHeal: true
```

Apply: `kubectl apply -f ethernet-operator-app.yaml`.



## • Configure E810 NICs via Ethernet Operator:

- Create a `NicClusterPolicy` to configure SR-IOV and VFs:

```
apiVersion: ethernet.intel.com/v1
kind: NicClusterPolicy
metadata:
  name: e810-policy
  namespace: ethernet-operator
spec:
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  deviceType: e810
  sriov:
    enabled: true
    numVfs: 8
  firmware:
    update: true
    url: "http://<firmware-repo>/e810_firmware.bin"
```

- Replace `url` with a valid firmware binary location (host firmware on a local server if needed).

- Apply: `kubectl apply -f nic-policy.yaml`.

- The operator will configure SR-IOV, bind VFs to `vfio-pci`, and update firmware as needed.

## • Automate Host Setup with Ansible:

- Create a playbook (`e810_setup.yml`) to ensure SR-IOV is enabled and VFs are bound:

```
- name: Configure E810 NICs on Harvester nodes
  hosts: harvester_nodes
  tasks:
    - name: Enable SR-IOV VFs
      shell: echo 8 > /sys/class/net/{{ nic }}/device/sriov_numvfs
      vars:
        nic: enp1s0f0
        ignore_errors: yes
    - name: Load vfio-pci module
      modprobe:
        name: vfio-pci
    - name: Bind VFs to vfio-pci
      shell: |
        for vf in $(ls /sys/bus/pci/devices/{{ nic_pci_addr }}/virtfn*); do
          echo $(basename $(readlink $vf)) > /sys/bus/pci/drivers/vfio-pci/new_id
        done
      vars:
        nic_pci_addr: 0000:01:00.0
```





Run: `ansible-playbook -i inventory e810_setup.yml`.

## Step 2: Provision VMs with VF Passthrough

### • Create VM Templates in Harvester:

- Define a VM template (Ubuntu 22.04) with an E810 VF passed through:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: k8s-node-{{ item }}
  namespace: default
spec:
  running: true
  template:
    spec:
      domain:
        cpu:
          cores: 4
        memory:
          guest: 8Gi
        devices:
          disks:
            - disk:
                bus: virtio
                name: rootdisk
            - disk:
                bus: virtio
                name: cloudinitdisk
          interfaces:
            - name: mgmt
              masquerade: {}
            - name: sriov
              sriov: {}
          hostDevices:
            - deviceName: e810-vf
              name: e810-vf
      volumes:
        - name: rootdisk
          containerDisk:
            image: ubuntu:22.04
        - name: cloudinitdisk
          cloudInitNoCloud:
            userData: |
              #cloud-config
              password: ubuntu
              chpasswd: { expire: False }
              ssh_pwauth: True
```

Apply for 3 nodes: `for i in {1..3}; do sed "s/{{ item }}/$i/g" vm-template.yaml | kubectl apply -f -; done.` **Grok 3**    

- Automate with Ansible:

- Playbook ( `provision_vms.yaml` ):

```
yaml
- name: Provision VMs on Harvester
  hosts: localhost
  tasks:
    - name: Create Kubernetes VMs
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('template', 'vm-template.yaml') | from_yaml }}"
        loop: "{{ range(1,4) | list }}"
      vars:
        item: "{{ item }}"
```

Run: `ansible-playbook provision_vms.yaml`.

### Step 3: Install Custom DPDK/ICE Drivers in VMs

- Build Drivers:

- ICE driver:

```
git clone https://github.com/intel/ethernet-linux-ice
cd ethernet-linux-ice
make ADQ=1
tar -czf ice-driver.tar.gz out/ice.ko
```

bash

- DPDK:

```
git clone https://github.com/DPDK/dpdk
cd dpdk
meson build
ninja -C build
tar -czf dpdk.tar.gz build/
```

bash

- Install in VMs with Ansible:

- Playbook ( `install_drivers.yaml` ):

```
- name: Install DPDK/ICE drivers in VMs
  hosts: k8s_nodes
  tasks:
```

×

```
- name: Install dependencies
  apt:
    name: "{{ packages }}"
    state: present
  vars:
    packages:
      - build-essential
      - linux-headers-{{ ansible_kernel }}
      - libnuma-dev
- name: Copy ICE driver
  copy:
    src: ice-driver.tar.gz
    dest: /tmp/ice-driver.tar.gz
- name: Install ICE driver
  shell: |
    tar -xzf /tmp/ice-driver.tar.gz -C /tmp
    insmod /tmp/out/ice.ko
- name: Copy DPDK
  copy:
    src: dpdk.tar.gz
    dest: /tmp/dpdk.tar.gz
- name: Install DPDK
  shell: |
    tar -xzf /tmp/dpdk.tar.gz -C /opt
    ldconfig
- name: Bind VF to DPDK
  shell: |
    /opt/dpdk/build/bin/dpdk-devbind.py --bind=vfio-pci <vf-pci-addr>
  vars:
    vf_pci_addr: "{{ ansible_facts['pci']['e810_vf'] }}"
```

Run: `ansible-playbook -i inventory install_drivers.yaml`.

### Step 4: Deploy Kubernetes with SR-IOV Network Device Plugin

- Bootstrap Kubernetes:

- Use Ansible to install `kubeadm` and initialize the cluster:

yaml

```
- name: Bootstrap Kubernetes
  hosts: k8s_nodes
  tasks:
    - name: Install Kubernetes
      apt:
        name: "{{ packages }}"
        state: present
      vars:
        packages:
          - kubeadm
          - kubect1
          - kubelet
```



```
- name: Initialize control plane
  shell: kubeadm init --pod-network-cidr=10.244.0.0/16
  when: inventory_hostname == 'k8s-node-1'
```



Run: `ansible-playbook bootstrap_k8s.yml`.

#### • Deploy Multus CNI:

- Install Multus for multiple network interfaces:

```
kubectl apply -f https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-cni/master/
```

#### • Deploy SR-IOV Network Device Plugin:

- The SR-IOV Network Device Plugin exposes E810 VFs as Kubernetes resources, allowing pods to request them.
- Deploy the plugin:

```
kubectl apply -f https://raw.githubusercontent.com/intel/sriov-network-device-plugin/master/
```

- Configure the plugin to detect E810 VFs:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
      "resourceList": [
        {
          "resourceName": "intel_sriov_e810",
          "resourcePrefix": "intel.com",
          "selectors": {
            "vendors": ["8086"],
            "devices": ["1592", "1889"],
            "drivers": ["vfio-pci"],
            "isSriov": true
          }
        }
      ]
    }
```



Apply: `kubectl apply -f sriovdp-config.yaml`.

Grok 3



#### • Configure SR-IOV CNI:

- Deploy SR-IOV CNI for VF networking:

```
kubectl apply -f https://raw.githubusercontent.com/intel/sriov-cni/master/images/sriov/
```

- Create a `NetworkAttachmentDefinition`:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "sriov",
    "deviceID": "<vf-pci-addr>",
    "vf": 0,
    "ipam": {
      "type": "host-local",
      "subnet": "10.123.237.0/24",
      "rangeStart": "10.123.237.10",
      "rangeEnd": "10.123.237.100"
    }
  }'
```

Apply: `kubectl apply -f sriov-net.yaml`.

#### • Automate with ArgoCD:

- Store manifests in a Git repository and deploy via ArgoCD:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: k8s-cni
  namespace: argocd
spec:
  project: default
  source:
    repoURL: https://github.com/your-repo/k8s-config
    targetRevision: HEAD
    path: cni
```



```
destination:
  server: https://10.123.236.1:6443
  namespace: default
syncPolicy:
  automated:
    prune: true
    selfHeal: true
```



## Step 5: Deploy and Test MTL/MCM

### • Install MTL/MCM:



- Ansible playbook ( `install_mtl.yml` ):

```
- name: Install MTL/MCM
  hosts: k8s_nodes
  tasks:
    - name: Clone MTL repository
      git:
        repo: https://github.com/OpenVisualCloud/Media-Transport-Library
        dest: /opt/mtl
    - name: Build and install
      shell: cd /opt/mtl && ./build.sh && ./install.sh
```

```
Run: ansible-playbook install_mtl.yml.
```



### • Deploy Test Workload:

- Helm chart for MTL test pod using SR-IOV:

```
apiVersion: v1
kind: Pod
metadata:
  name: mtl-test
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net
    intel.com/sriov_e810: "1"
spec:
  containers:
    - name: mtl
      image: ubuntu:22.04
      command: ["/opt/mtl/bin/mtl test"]
  limits:
    intel.com/sriov_e810: 1
```



- Deploy via ArgoCD (similar to Step 4).

### • Validate:



Grok 3



- Check VF allocation: `kubectl describe pod mtl-test`.
- Verify DPDK binding: `/opt/dpdk/build/bin/dpdk-devbind.py --status`.
- Monitor MTL performance via logs: `kubectl logs mtl-test`.

## Step 6: Teardown

- Delete VMs: `kubectl delete -f vm-template.yaml`.
- Reset Kubernetes: Use Ansible to run `kubeadm reset`.

## Best Practices

### • Intel Ethernet Operator:

- Regularly update the operator for E810 firmware and driver patches.
- Monitor `NicClusterPolicy` status: `kubectl get nicclusterpolicy`.

### • SR-IOV Network Device Plugin:

- Tune `resourceList` in `sriovdp-config` to match E810 device IDs.
- Use taints to restrict SR-IOV pods to specific nodes.

### • Automation:

- Centralize configurations in Git for ArgoCD.
- Modularize Ansible roles (e.g., `nic_setup`, `vm_provision`, `cni`).

### • Monitoring:

- Use Rancher to deploy Prometheus for VF and MTL metrics.

## References

- Intel Ethernet Operator: <https://github.com/intel/ethernet-operator>
- SR-IOV Network Device Plugin: <https://github.com/intel/sriov-network-device-plugin>
- SR-IOV CNI: <https://github.com/intel/sriov-cni>
- Multus CNI: <https://github.com/k8snetworkplumbingwg/multus-cni>
- OpenVisualCloud MTL: <https://github.com/OpenVisualCloud/Media-Transport-Library>
- Harvester Docs: <https://docs.harvesterhci.io/>

This approach ensures robust NIC management with the Intel Ethernet Operator and SR-IOV Network Device Plugin, automating E810 VF allocation for high-performance MTLMCI testing. Let me know if you need further details!

Ask anything



DeepSearch 

Think 

Edit Image 



