



KubeCon



CloudNativeCon

Europe 2026

#KubeCon #CloudNativeCon

Multi-Network Step-by-Step: Enabling SR-IOV Support From Kubernetes Network (DRA) Drivers

Masaharu Kanda - NTT, Inc.
Lionel Jouin - Red Hat





KubeCon



CloudNativeCon

Europe 2026



Masaharu Kanda
Senior Research Engineer
NTT, Inc.



Lionel Jouin
Software Engineer
Red Hat

SR-IOV / Virtual Function



KubeCon



CloudNativeCon

Europe 2026

Single Root I/O Virtualization (SR-IOV)

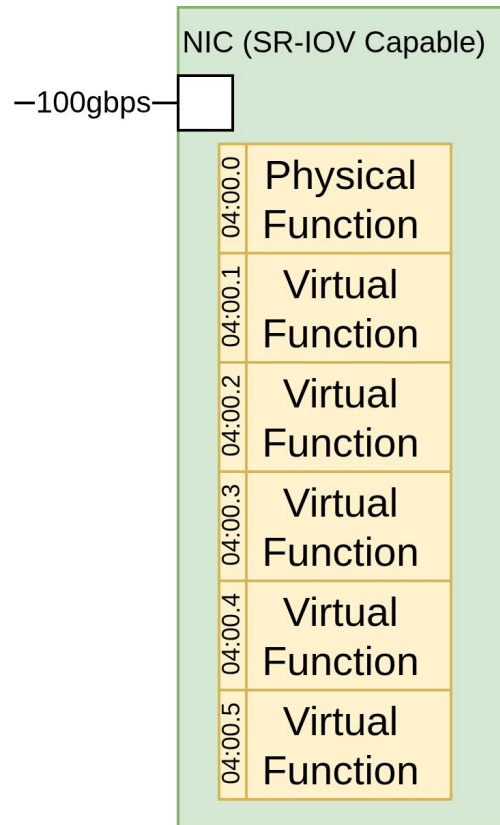
- PCI Express standard allowing a single a network card (NIC) to expose multiple “slices” called Virtual Functions (VFs).
- VFs appear as independent PCIe devices to the system.

Hardware-Level Separation

- Each VF has its own PCI configuration space, MAC address, queues, and memory regions...
- Isolation limits: VFs share NIC resources (bandwidth).

Hardware & Driver Constraints

- Hardware limits: max number of VFs is fixed by NIC.
- Hardware support required.



Virtual Function - Use Cases



KubeCon



CloudNativeCon

Europe 2026

High-Performance I/O at Scale

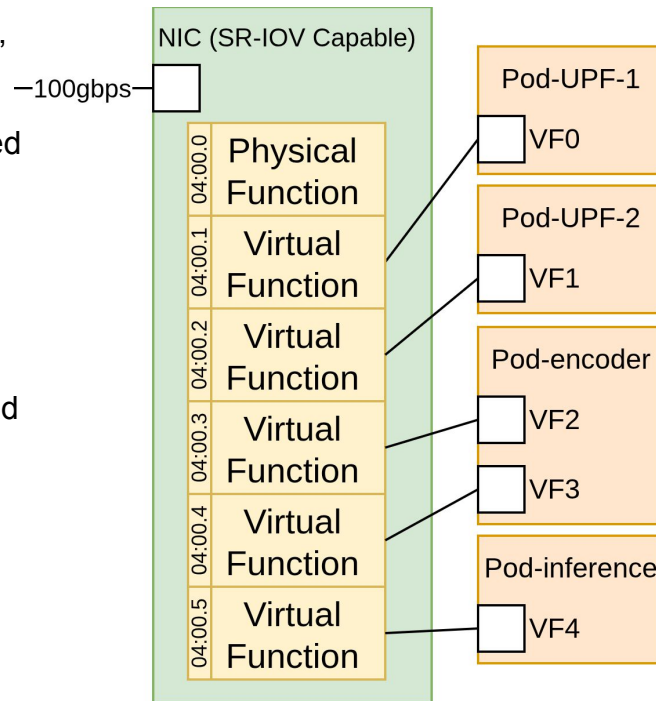
- Common on high-end NICs (often SmartNICs) with advanced features (e.g., RDMA).
- Effectively “duplicates” a NIC into multiple Virtual Functions (VFs) for isolated workloads.

Deterministic, Low-Latency Networking

- Near line-rate performance with predictable latency.
- Kernel bypass (e.g., DPDK) and hardware offloading reduces CPU overhead for packet processing.

Typical Use Cases

- Cloud-Native Network Functions (CNFs) in Telco (5G UPF, RAN...).
- HPC and scientific computing.
- AI/ML training and inference.



A Real-World Use Case at Expo 2025 Osaka, Japan



KubeCon



CloudNativeCon

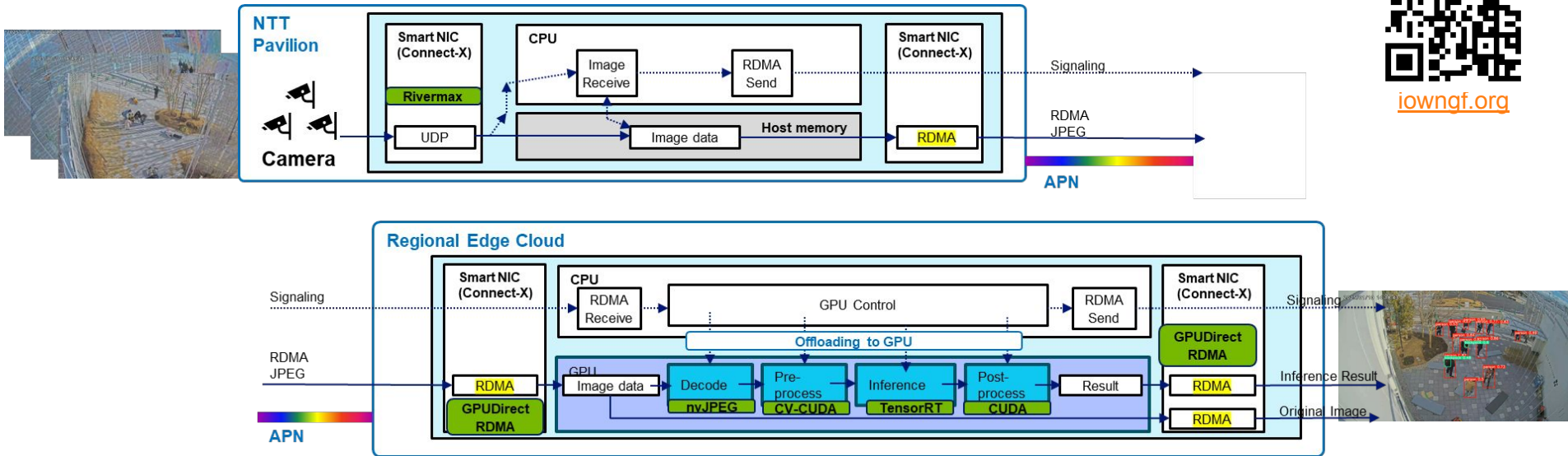
Europe 2026

RDMA over SR-IOV VFs for a Video AI Analysis

SR-IOV VFs (SmartNIC) are used for RDMA connection between each pipeline stage.



iowngf.org



* APN: All Photonics Network, which connects SmartNICs via a PEC (Photonic-Electronic Convergence) switch.

Multi-Network



KubeCon



CloudNativeCon

Europe 2026

Multi-Network, a SIG-Network Subproject

KEP-3698 - Multi-Network

- Closed, but the objectives are not abandoned

DRA Enhancements to Support Networking

- KEP-4817: ResourceClaim Device Status
- KEP-5075: Consumable Capacity

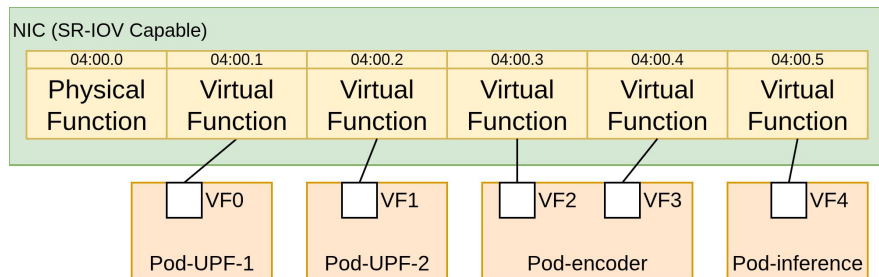
Kubernetes Network API Integration

- First-class multi-network support
- Integration with the rest of the Kubernetes networking ecosystem

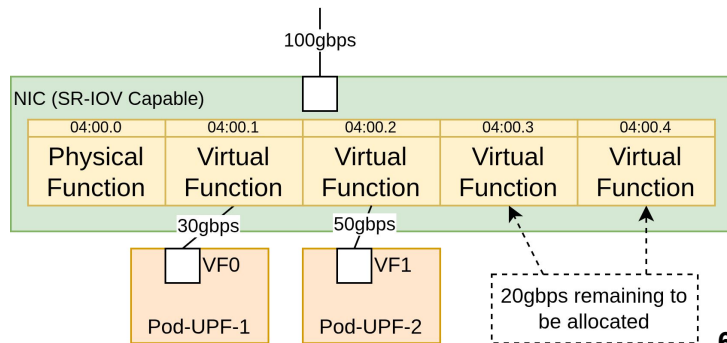
Today

- Many original KEP-3698 goals implemented via DRA
- The remaining work is implementation & ecosystem adoption

KEP-3698 - User Story 2 - VF



KEP-3698 - User Story 8 - Bandwidth



Kubernetes Network Drivers (KND)



KubeCon



CloudNativeCon

Europe 2026

Modernize Kubernetes network model *(CNI / Device Plugin)*

- DRA: Request resources for scheduling
- NRI: Hook into the pod lifecycle for configuration

Network as a First-Class Resource

Enables and Simplify Complex Use-Cases

- Topology-Aware Scheduling
- High-Performance / Hardware Offloading

Simplified Architecture

- Independent / Composable drivers
- Cleaner lifecycle

TODO

DRA applied to Virtual Functions



KubeCon



CloudNativeCon

Europe 2026

DRA is an evolution API of the legacy Device Plugin

DRA enables more flexible device assignment to Pods.

GA on Kubernetes v1.34.

The useful DRA APIs for designing network interface bandwidth

- Consumable Capacity
- Partitionable Devices

Consumable Capacity DRA API



KubeCon



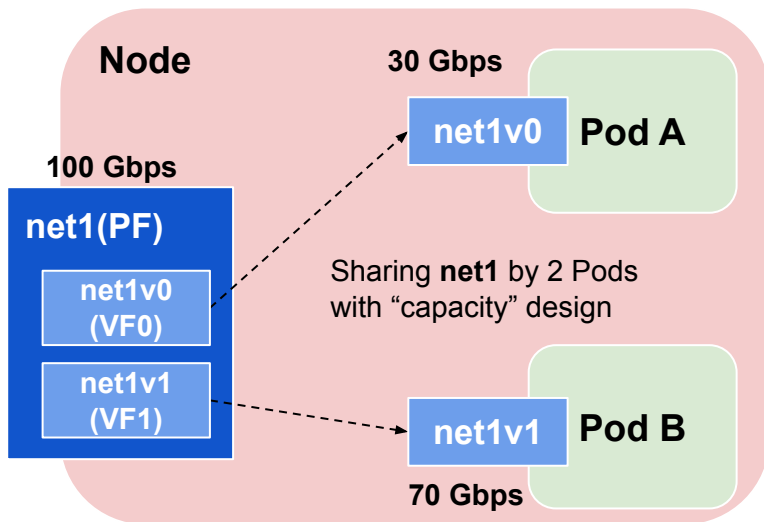
CloudNativeCon

Europe 2026

Sharing one device and indicate the capacity consumed for each Pod

The Consumable Capacity feature enables a model in which a single device can be shared by multiple Pods by defining a capacity for that device and allowing each Pod to consume a portion of it.

```
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
...
spec:
  devices:
  - name: net1
    allowMultipleAllocations: true
    capacity:
      bandwidth:
        value: 100Gi
        requestPolicy:
          default: 5Gi
    numOfVF;
    value: 2
    requestPolicy:
      default: 1
```



```
apiVersion: resource.k8s.io/v1
kind: ResourceClaim
...
spec:
  devices:
    requests:
    - name: req-a
      exactly:
        deviceClassName: resource.example.com
        capacity:
          requests:
            bandwidth: 30Gi
            numOfVF: 1
```

```
kind: ResourceClaim
...
- name: req-b
  exactly:
    deviceClassName: resource.example.com
    capacity:
      requests:
        bandwidth: 70Gi
        numOfVF: 1
```

* This is a conceptual figure for explaining *Consumable Capacity*; in reality, each VF appears on the node as an independent device.

Consumable Capacity DRA API



KubeCon



CloudNativeCon

Europe 2026

Sharing one device and indicate its capacity

The Consumer

Pods by

Multiple

When deploying Pod A, which VF is selected is non-deterministic.

Impossible to handle cases where different configurations are applied to individual VFs.

(for example, enabling or disabling spoof check or using different trust modes)

* This is a conceptual figure for explaining *Capacity*; in reality, each VF appears on the node as an independent device.

```
- name: req-b
  exactly:
    deviceClassName: resource.example.com
  capacity:
    requests:
      bandwidth: 70Gi
      numOfVF: 1
```

Partitionable Devices DRA API



KubeCon



CloudNativeCon

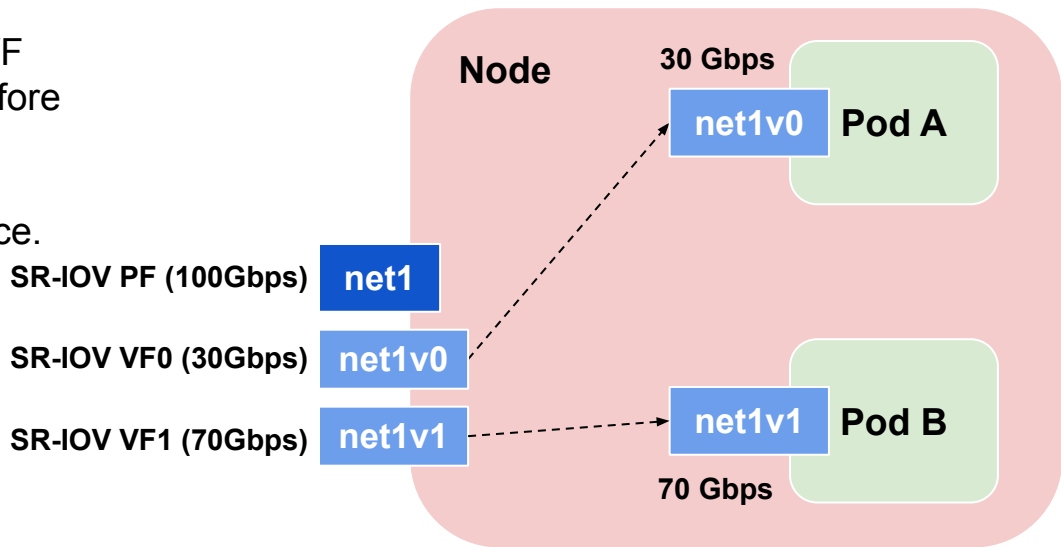
Europe 2026

Sharing a parent (whole) device as partitioned child devices

Primary designed for NVIDIA GPU MIG.

Can be treated SR-IOV VFs as partitions of their parent PF.

- Create VFs and **pre-configure** per-VF bandwidth (e.g., via a ConfigMap) before deploying Pods.
- Each VF should be advertised as an independent device in a ResourceSlice.
- Users can select the VF in a ResourceClaim.



Bandwidth design for SR-IOV VF with Partitionable Devices DRA API



KubeCon



CloudNativeCon

Europe 2026

Assigning bandwidth to individual VF as partition of PF

Sample implementation on top of the CNI DRA Driver.

Users can configure bandwidth for each VF with a ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: vf-bandwidth
  namespace: default
data:
  vf-bandwidth.yaml: |
    nodes:
      node1:
        parents:
          # Use the parent PF identifier (clean PCI address)
          0000:a8:00.0:
            # Key: VF PCIe address, Value: Percentage of bandwidth
            0000:a8:01.0: 80
            0000:a8:01.1: 20
```

- Two VFs
- Assign 80% bandwidth to one VF and 20% to the other

Bandwidth design for SR-IOV VF with Partitionable Devices DRA API



KubeCon



CloudNativeCon

Europe 2026

You can confirm allocated bandwidth for each VF in ResourceSlice

Advertised SR-IOV VFs as partitions of PF in a ResourceSlice

```
apiVersion: resource.k8s.io/v1
kind: ResourceSlice
...
spec:
  devices:
    - allowMultipleAllocations: false
      attributes:
        cni.dra.networking.x-k8s.io/allocatedBandwidth:
          int: 80000
        cni.dra.networking.x-k8s.io/pciAddress:
          string: 0000:a8:01.0
      ...
    capacity:
      cni.dra.networking.x-k8s.io/maxBandwidth:
        value: 100Ki
      consumesCounters:
        - counterSet: 0000-a8-00-0-bandwidth
          counters:
            bandwidth:
              value: 80Ki
          name: net1v0
      ...
```

```
...
driver: cni.dra.networking.x-k8s.io
nodeName: node1
pool:
  generation: 1
  name: node1
  resourceSliceCount: 1
sharedCounters:
- counters:
    bandwidth:
      value: 100Ki
    name: 0000-a8-00-0-bandwidth
```

This VF has **80%** bandwidth of PF's that. counterset **0000-a8-00-0-bandwidth** is shared with VFs which has the same PF.



KubeCon



CloudNativeCon

Europe 2026

DEMO



github.com/kanikan/cni-dra-driver/tree/feature/sriov-nic-partitionable-devices/docs/demo-sriovnetdev

Demo - Environment



KubeCon



CloudNativeCon

Europe 2026

```
$ # Shorthand: v0 means VF0, v1 means VF1. I'll refer to them as 'eagle04 v0/v1' and 'eagle05 v0/v1'.
$ # Check Intel E810 VFs on eagle04
$ ssh eagle04 'lspci -nn |grep -i 8086:1889'
a8:01.0 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual Function [8086:1889] (rev 02)
a8:01.1 Ethernet controller [0200]: Intel Corporation Ethernet Adaptive Virtual Function [8086:1889] (rev 02)
$ # Check Mellanox ConnectX-6 VFs on eagle05
$ ssh eagle05 'lspci -nn |grep -i 15b3:101c'
27:00.2 Ethernet controller [0200]: Mellanox Technologies MT28908 Family [ConnectX-6 Virtual Function] [15b3:101c]
27:00.3 Ethernet controller [0200]: Mellanox Technologies MT28908 Family [ConnectX-6 Virtual Function] [15b3:101c]
$ # Bandwidth plan for each VF (ConfigMap)
$ cat vf-bandwidth-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: vf-bandwidth
  namespace: default
data:
  vf-bandwidth.yaml: |
    nodes:
      eagle04:
        parents:
          # Use the parent PF identifier (clean PCI address)
          0000:a8:00.0:
            # Key: VF PCIe address, Value: Percentage of bandwidth
            0000:a8:01.0: 80
            0000:a8:01.1: 20
      eagle05:
        parents:
          0000:27:00.0:
            0000:27:00.2: 40
            0000:27:00.3: 60
```

Cluster

mbs08: control plane

eagle04: node

Intel E810

a8:00.0 PF

enp168s0f0

a8:01.0 VF0

enp168s0f0v0

80 Gbps

a8:01.1 VF1

enp168s0f0v1

20 Gbps

eagle05: node

Mellanox
ConnectX-6

27:00.0 PF

ens1f0np0

27:00.2 VF0

ens1f0v0

40 Gbps

27:00.3 VF1

ens1f0v1

60 Gbps

Demo - Deploy CNI DRA Driver



KubeCon



CloudNativeCon

Europe 2026

```
$ # Create DeviceClass for the CNI DRA Driver
$ cat deviceclasses.yaml
---
apiVersion: resource.k8s.io/v1
kind: DeviceClass
metadata:
  name: network-interface
spec:
  selectors:
  - cel:
      expression: device.driver == "cni.dra.networking.x-k8s.io"
$ kubectl apply -f deviceclasses.yaml
deviceclass.resource.k8s.io/network-interface created
$ # Deploy the CNI DRA Driver
$ kubectl apply -f ../../deployments/cni-dra-driver_sriov.yaml
clusterrole.rbac.authorization.k8s.io/cni-dra-driver-cluster-role unchanged
serviceaccount/cni-dra-driver-service-account unchanged
clusterrolebinding.rbac.authorization.k8s.io/cni-dra-driver-role-binding unchanged
role.rbac.authorization.k8s.io/cni-dra-driver-configmap-reader unchanged
rolebinding.rbac.authorization.k8s.io/cni-dra-driver-configmap-role-binding unchanged
role.rbac.authorization.k8s.io/cni-dra-driver-resourceclaim-reader unchanged
rolebinding.rbac.authorization.k8s.io/cni-dra-driver-resourceclaim-role-binding unchanged
daemonset.apps/cni-dra-driver created
$ kubectl set image daemonset/cni-dra-driver cni-dra-driver=localhost:5000/kubernetes
-sigs/cni-dra-driver:latest
daemonset.apps/cni-dra-driver image updated
$ # Confirm the CNI DRA Driver DaemonSet/Pods
$ kubectl get pod -o wide |grep cni-dra-driver
cni-dra-driver-ks7dj 1/1 Running 0 20s 192.168.210.125 eagle05
<none> <none>
cni-dra-driver-m98d6 1/1 Running 0 19s 192.168.210.124 eagle04
<none> <none>
cni-dra-driver-wsn65 1/1 Running 0 20s 192.168.200.28 mbs08
<none> <none>
```

Create DeviceClass.
Deploy CNI DRA Driver.

Demo - Confirm ResourceSlice: eagle04



KubeCon



CloudNativeCon

Europe 2026

```
$ # Check ResourceSlice for eagle04
$ # ResourceSlice: eagle04 v0 (expected 80G)
$ kubectl get resourceslice eagle04-cni.dra.networking.x-k8s.io-pkmqq
-o yaml | grep -e 'name: enp168s0f0v0' -B 20
- allowMultipleAllocations: false
  attributes:
    cni.dra.networking.x-k8s.io/allocatedBandwidth:
      int: 80000
    cni.dra.networking.x-k8s.io/interfaceName:
      string: enp168s0f0v0
    cni.dra.networking.x-k8s.io/interfaceType:
      string: SrioV
    cni.dra.networking.x-k8s.io/pciAddress:
      string: 0000:a8:01.0
    resource.kubernetes.io/pciRoot:
      string: 0000:a8
  capacity:
    cni.dra.networking.x-k8s.io/maxBandwidth:
      value: 100k
  consumesCounters:
    - counterSet: 0000-a8-00-0-bandwidth
      counters:
        bandwidth:
          value: 80k
  name: enp168s0f0v0
```

eagle04 v0: 80K [Mbps]

```
$ # ResourceSlice: eagle04 v1 (expected 20G)
$ kubectl get resourceslice eagle04-cni.dra.networking.x-k8s.io-pkmqq
-o yaml | grep -e 'name: enp168s0f0v1' -B 20
- allowMultipleAllocations: false
  attributes:
    cni.dra.networking.x-k8s.io/allocatedBandwidth:
      int: 20000
    cni.dra.networking.x-k8s.io/interfaceName:
      string: enp168s0f0v1
    cni.dra.networking.x-k8s.io/interfaceType:
      string: SrioV
    cni.dra.networking.x-k8s.io/pciAddress:
      string: 0000:a8:01.1
    resource.kubernetes.io/pciRoot:
      string: 0000:a8
  capacity:
    cni.dra.networking.x-k8s.io/maxBandwidth:
      value: 100k
  consumesCounters:
    - counterSet: 0000-a8-00-0-bandwidth
      counters:
        bandwidth:
          value: 20k
  name: enp168s0f0v1
```

eagle04 v1: 20K [Mbps]

Demo - Confirm ResourceSlice: eagle05



KubeCon



CloudNativeCon

Europe 2026

```
$ # Check ResourceSlice for eagle05
$ # ResourceSlice: eagle05 v0 (expected 40G)
$ kubectl get resourceslice eagle05-cni.dra.networking.x-k8s.io-dgn4h
-o yaml | grep -e 'name: ens1f0v0' -B 20
- allowMultipleAllocations: false
attributes:
  cni.dra.networking.x-k8s.io/allocatedBandwidth:
    int: 40000
  cni.dra.networking.x-k8s.io/interfaceName:
    string: ens1f0v0
  cni.dra.networking.x-k8s.io/interfaceType:
    string: SrioV
  cni.dra.networking.x-k8s.io/pciAddress:
    string: "0000:27:00.2"
  resource.kubernetes.io/pciRoot:
    string: "0000:27"
capacity:
  cni.dra.networking.x-k8s.io/maxBandwidth:
    value: 100k
consumesCounters:
- counterSet: 0000-27-00-0-bandwidth
  counters:
    bandwidth:
      value: 40k
name: ens1f0v0
```

eagle05 v0: 40K [Mbps]

```
$ # ResourceSlice: eagle05 v1 (expected 60G)
$ kubectl get resourceslice eagle05-cni.dra.networking.x-k8s.io-dgn4h
-o yaml | grep -e 'name: ens1f0v1' -B 20
- allowMultipleAllocations: false
attributes:
  cni.dra.networking.x-k8s.io/allocatedBandwidth:
    int: 60000
  cni.dra.networking.x-k8s.io/interfaceName:
    string: ens1f0v1
  cni.dra.networking.x-k8s.io/interfaceType:
    string: SrioV
  cni.dra.networking.x-k8s.io/pciAddress:
    string: "0000:27:00.3"
  resource.kubernetes.io/pciRoot:
    string: "0000:27"
capacity:
  cni.dra.networking.x-k8s.io/maxBandwidth:
    value: 100k
consumesCounters:
- counterSet: 0000-27-00-0-bandwidth
  counters:
    bandwidth:
      value: 60k
name: ens1f0v1
```

eagle05 v1: 60K [Mbps]

Demo - ResourceClaim for Pod A



KubeCon



CloudNativeCon

Europe 2026

```
$ # Deploy Pod A: request a VF with bandwidth > 50Gbps
$ # (CEL filter in ResourceClaimTemplate)
$ yq 'select(.kind == "ResourceClaimTemplate")' pod-a.yaml
---
apiVersion: resource.k8s.io/v1
kind: ResourceClaimTemplate
metadata:
  name: sriov-net0
spec:
  spec:
    devices:
      requests:
        - name: sriov-net0
          exactly:
            deviceClassName: network-interface
            allocationMode: ExactCount
            count: 1
            selectors:
              - cel:
                  expression: >
                    "interfaceType" in device.attributes["cni.dra.networking.x-k8s.io"] && device.attributes["cni.dra.networking.x-k8s.io"].interfaceType == "Sriov" && "allocatedBandwidth" in device.attributes["cni.dra.networking.x-k8s.io"] && device.attributes["cni.dra.networking.x-k8s.io"].allocatedBandwidth > 50000
            config:
              - requests:
                  - sriov-net0
              opaque:
                driver: cni.dra.networking.x-k8s.io
                parameters: # CNIParameters with the GVK, interface name and CNI Config (in YAML format).
                  apiVersion: cni.networking.x-k8s.io/v1alpha1
                  kind: CNI
                  config:
                    cniVersion: 1.0.0
                    name: sriov-net0
                    plugins:
                      - type: sriov
                        ipam:
                          type: static
                          addresses:
                            - address: 192.168.139.119/24
```

Filter allocatedBandwidth > 50000 (50 Gbps)

IP address: 192.168.139.119

Demo - Deploy Pod A



KubeCon



CloudNativeCon

Europe 2026

```
$ # Pod definition
$ yq 'select(.kind == "Pod")' pod-a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: demo-sriovnetdev-pod-a
spec:
  containers:
  - name: ubuntu
    image: localhost/demo-ubuntu
    imagePullPolicy: IfNotPresent
    command:
    - sleep
    - infinity
    resources:
      requests:
        memory: 1024Mi
        cpu: 2
      limits:
        memory: 1024Mi
        cpu: 2
      claims:
      - name: sriov
    resourceClaims:
    - name: sriov
      resourceClaimTemplateName: sriov-net0
$ kubectl apply -f pod-a.yaml
resourceclaimtemplate.resource.k8s.io/sriov-net0 created
pod/demo-sriovnetdev-pod-a created
$ kubectl get pod demo-sriovnetdev-pod-a
```

NAME	READY	STATUS	RESTARTS	AGE
demo-sriovnetdev-pod-a	1/1	Running	0	15s

Demo - Check ResourceClaim status for Pod A



KubeCon



CloudNativeCon

Europe 2026

```
$ # Check ResourceClaim status for pod-a
$ kubectl get resourceclaim demo-sriovnetdev-pod-a-sriov-k6q9c -o jsonpath='{.status.devices}' |jq .
[
  {
    "conditions": null,
    "data": {
      "cniVersion": "1.0.0",
      "interfaces": [
        {
          "mac": "ba:a4:6a:d5:e7:71",
          "mtu": 1500,
          "name": "ens1f0v1",
          "sandbox": "/var/run/netns/cc9c0c92-7e2f-4b6d-8b1a-bba8200ef419"
        }
      ],
      "ips": [
        {
          "address": "192.168.130.119/24",
          "interface": 0
        }
      ]
    },
    "device": "ens1f0v1",
    "driver": "cni.dra.networking.x-k8s.io",
    "networkData": {
      "hardwareAddress": "ba:a4:6a:d5:e7:71",
      "interfaceName": "ens1f0v1",
      "ips": [
        "192.168.130.119/24"
      ]
    },
    "pool": "eagle05"
  }
]
```

- eagle05 v1 (= 60 Gbps) was allocated
- Selected device with “> 50 Gbps” bandwidth
- IP: 192.168.130.119

Demo - Check NICs inside Pod A



KubeCon



CloudNativeCon

Europe 2026

```
$ # Check the network interface inside Pod A
$ kubectl exec -it demo-sriovnetdev-pod-a -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    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: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: eth0@if29: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP group default qlen 1000
    link/ether 0e:3a:4c:40:a6:5a brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 182.16.37.95/32 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::c3a:4cff:fe40:a65a/64 scope link
        valid_lft forever preferred_lft forever
8: ens1f0v1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether ba:a4:6a:d5:e7:71 brd ff:ff:ff:ff:ff:ff permaddr 26:ee:93:e9:ab:9f
    altname enp39s0f0v1
    inet 192.168.130.119/24 brd 192.168.130.255 scope global ens1f0v1
        valid_lft forever preferred_lft forever
    inet6 fe80::b8a4:6aff:fed5:e771/64 scope link
        valid_lft forever preferred_lft forever
```

Demo - Deploy Pod B, Check ResourceClaim status



KubeCon



CloudNativeCon

Europe 2026

```
$ # Deploy Pod B : request a VF with bandwidth > 50Gbps
$ kubectl apply -f pod-b.yaml
resourceclaimtemplate.resource.k8s.io/sriov-net1 created
pod/demo-sriovnetdev-pod-b created
$ kubectl get pod demo-sriovnetdev-pod-b
NAME                READY   STATUS    RESTARTS   AGE
demo-sriovnetdev-pod-b 1/1     Running   0           17s
$ # Check ResourceClaim status for Pod B
$ kubectl get resourceclaim demo-sriovnetdev-pod-b-sriov-98wqg -o jsonpath='{.status.devices}' | jq .
[
  {
    "conditions": null,
    "data": {
      "cniVersion": "1.0.0",
      "interfaces": [
        {
          "mac": "ba:24:e4:d1:09:e4",
          "mtu": 1500,
          "name": "enp168s0f0v0",
          "sandbox": "/var/run/netns/dcb8f916-1bf0-45b9-9106-a43f9734ad4b"
        }
      ],
      "ips": [
        {
          "address": "192.168.130.120/24",
          "interface": 0
        }
      ]
    },
    "device": "enp168s0f0v0",
    "driver": "cni.dra.networking.x-k8s.io",
    "networkData": {
      "hardwareAddress": "ba:24:e4:d1:09:e4",
      "interfaceName": "enp168s0f0v0",
      "ips": [
        "192.168.130.120/24"
      ]
    },
    "pool": "eagle04"
  }
]
```

ResourceClaimTemplate for Pod B

- Filter allocatedBandwidth > 50000 (50 Gbps)
- IP: 192.168.130.120
- eagle04 v0 (= 80 Gbps) was allocated
- Selected device with “> 50 Gbps” bandwidth
- IP: 192.168.130.120

Demo - Check NICs inside Pod B



KubeCon



CloudNativeCon

Europe 2026

```
$ # Check the network interface inside Pod B
$ kubectl exec -it demo-sriovnetdev-pod-b -- ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    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: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc noqueue state UP group default qlen 1000
    link/ether ee:19:01:89:c9:2b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 182.16.56.149/32 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::ec19:1ff:fe89:c92b/64 scope link
        valid_lft forever preferred_lft forever
9: enp168s0f0v0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether ba:24:e4:d1:09:e4 brd ff:ff:ff:ff:ff:ff
    inet 192.168.130.120/24 brd 192.168.130.255 scope global enp168s0f0v0
        valid_lft forever preferred_lft forever
    inet6 fe80::b824:e4ff:fed1:9e4/64 scope link
        valid_lft forever preferred_lft forever
```

Disadvantages of the implementation with Partitionable Devices API



KubeCon



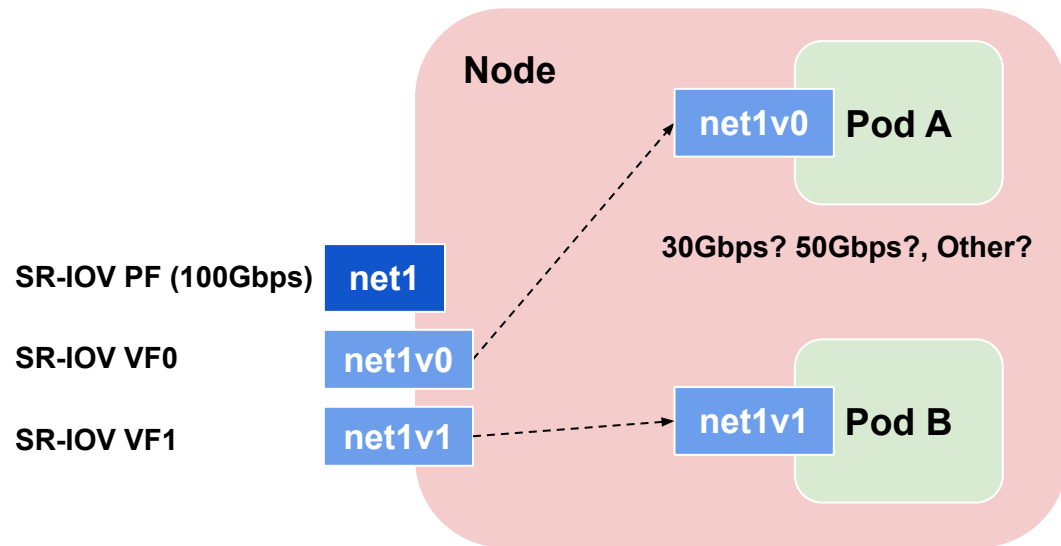
CloudNativeCon

Europe 2026

Less flexible: bandwidth is pre-configured, not requested at deploy time

We can configure bandwidth for each VF with Partitionable Devices API but we have to do it in advance before deploying Pods.

Users would like to request bandwidth when they deploy Pods!



It would be nice if the bandwidth of an SR-IOV VF could be dynamically specified via the Pod spec during assignment — but how?



KEP-5941 - Shared Consumable Capacities Across Related Devices



KubeCon



CloudNativeCon

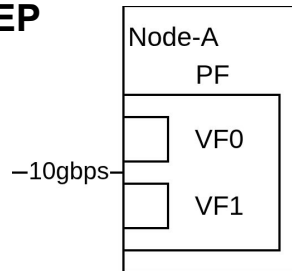
Europe 2026

KEP-5075 Consumable Capacity - Extension of the KEP

- Two new improvements based on KEP-5075

1. Shared Capacity

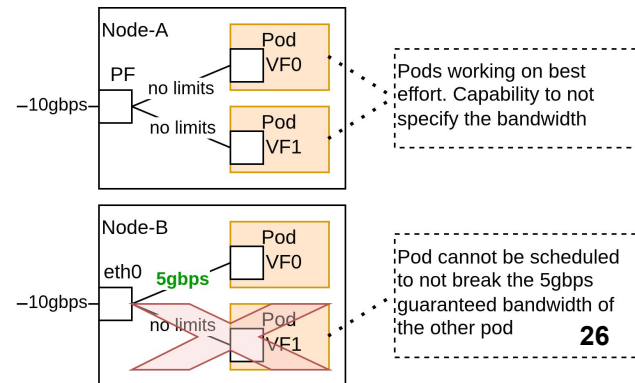
- Expose Multiple device with shared capacity.
- Use-Cases:
 - Multiple VFs sharing the same bandwidth.
 - IP management.
 - CPU: Sharing Cache / Memory Bandwidth



```
kind: ResourceSlice
spec:
  sharedCapacity:
    - name: enp0s0f0-capacities # NIC
      capacity:
        bandwidth:
          value: 10Gi
  devices:
    - name: enp0s0f0v0 # VF0
      capacity:
        enp0s0-capacities/bandwidth: {}
    - name: enp0s0f0v1 # VF1
      capacity:
        enp0s0-capacities/bandwidth: {}
```

2. Optional Capacity with Guarantees

- Allow capacity to be optional.
- Use-Cases:
 - Some workload requiring guaranteed bandwidth.
 - Some workload working on best effort.



What can we use today for assigning NICs?

Kubernetes Network Drivers

- Incubating DRA Drivers

DRANET

- A promising option for handling RDMA NICs with DRA.

CNI DRA Driver

- A basic NIC-oriented DRA driver built on standard CNI, which can be used as a foundation for prototyping new features.
- Proof-Of-Concept, Experimentation project, Prototyping new features, Demos...

DRA Driver SR-IOV

- A DRA driver for SR-IOV VFs, designed with backward compatibility in mind for sriov-network-device-plugin and Multus.

Timeline for DRA features

Planned and Potential KEPs for Networking Support

- [KEP-5075](#): Consumable Capacity: v1.36 Beta
- [KEP-4815](#): Partitionable Device: v1.36 Beta
- [KEP-4817](#): Resource Claim Status: v1.36 Stable
- [KEP-5304](#): Device Attributes in Downward API: v1.36 Alpha
- [KEP-5941](#): Shared Consumable Capacities Across Related Devices: v1.37 Alpha

Get Involved in the Community



linktr.ee/kubecon.eu.2026.sriov

Slack Channels / Meeting Notes / Zoom meeting / Google Group

- SIG-network
- Multi Network
- WG Device Management



KubeCon



CloudNativeCon

Europe 2026

