KubeCon | CloudNativeCon

North America 2019

intel

Red Hat

KubeCon | CloudNativeCon

North America 2019

Running High Performance User-space Packet Processing Apps in Kubernetes

Abdul Halim, Intel

Peng Liu, Red Hat

Legal Notices and Disclaimers

- Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration **No product or component can be absolutely secure**. Check with your system manufacturer or retailer or learn more at intel.com.
- Intel, the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.
- *Other names and brands may be claimed as the property of others.
- © Intel Corporation

Topics

- Motivation
- Challenges and overheads
- Utilizing the NICs line-rates
- DPDK
- Sample deployment
- Capabilities needed in K8s
- SR-IOV networking in K8s
- SR-IOV network operator
- Demo

About us

Abdul Halim

Intel

- Cloud Software Engineer at ***Network Control and Logic Group***, Intel
- Enabling high-performance networking solution for NFV with Kubernetes

@ahalim-intel

Peng Liu

Red Hat

- NFV Partner Engineer at ***CTO Office*** of RedHat
- Working on enhance open source softwares for NFV use cases.

@pliurh

Motivation

The 5G network is expected to enable fully mobile and connected society with the vision of providing:

- Greater throughput
- Lower latency
- Ultra-high reliability
- Much higher connectivity density, and
- Higher user mobility

- Demands increasing network speed

5G depends on Kubernetes in the cloud!

Ref: https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf

Challenges

Adapter	Capability
Intel® Ethernet Network Adapter XXV710-DA2	25/10/1GbE
Intel® Ethernet Converged Network Adapter XL710-QDA1	40/10GbE
Intel® Ethernet Controller E810-CAM2/CAM1	100/50/25/10/1GbE

NIC Capability (Line-rate)	Payloads (Bytes)	Packets per seconds	Packets arrival rate(ns)
10 Gbits/s	1500	812.74Kpps	1230.4
	46	14.88Mpps	67.2
40 Gbits/s	1500	3.25Mpps	307.6
	46	59.52Mpps	16.8
100 Gbits/s	1500	8.12Mpps	123.04
	46	148.80Mpps	6.72

- ➢ The higher the packet rate, the lower the time to process it
- ➢ To achieve “*zero-packet-loss*”, time spent in network stack must be <= packets arrival rate

Overheads in Kernel

- ➢ For the smallest frame (84 bytes) there is **67.2 ns** to process a packet given 10Gbit/s link

- ➢ Which is:
  - ○ 201 CPU cycles @ 3GHz

Ref: Brouer, J.D, 2015 - Network stack challenges at increasing speeds

Overheads in Kernel

- A lot of these cycles could be spent on:
  - Context switching
  - System calls
  - Interrupt handling
  - Lock and unlock
  - Data copy
  - Cache misses

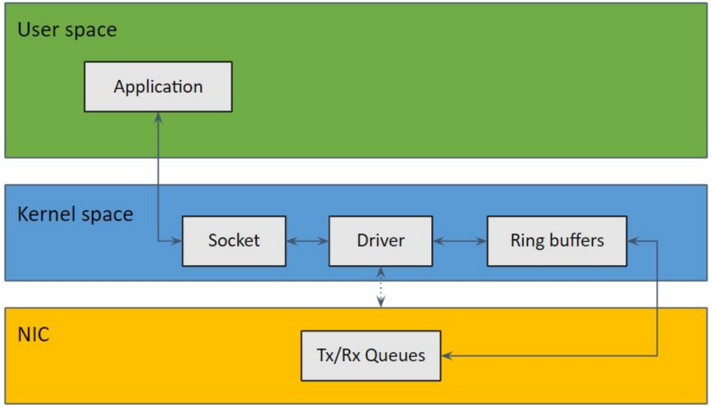


Fig: Packet processing in Linux kernel

Ref: [Brouer, J.D, 2015 - Network stack challenges at increasing speeds]

Utilizing full line-rates

- By adding performance optimization in Kernel stack
- Network stack bypass solution
  - DPDK - Packet processing in user space
  - RDMA
  - Programmable packet processing with XDP

DPDK

- ***DPDK (Data Plane Development Kit)*** *is a framework (under the Linux Foundation) comprised of various userspace libraries and drivers for fast packet processing*

How DPDK does it

➢ DPDK runs in user space

- ○ Dedicated processors -> no context switching
- ○ Dedicated network I/O
- ○ Hugepages -> no swap, TLB
- ○ UIO -> No copy from Kernel
- ○ Polling -> No interrupt overhead
- ○ Lockless synchronization
- ○ Batch packets handling

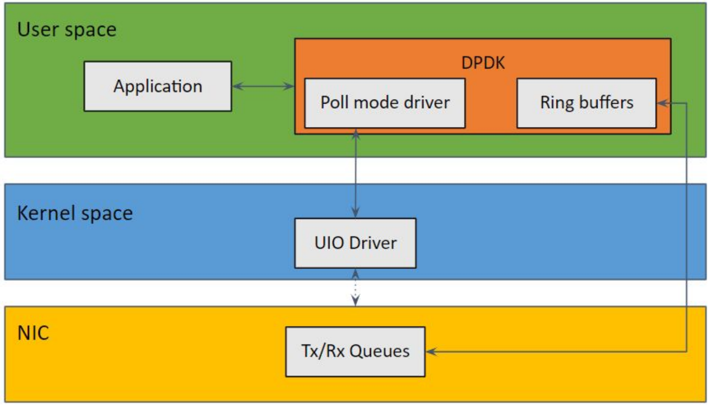


Fig: Packet processing in user space with DPDK

Ref: Brouer, J.D, 2015 - Network stack challenges at increasing speeds

DPDK

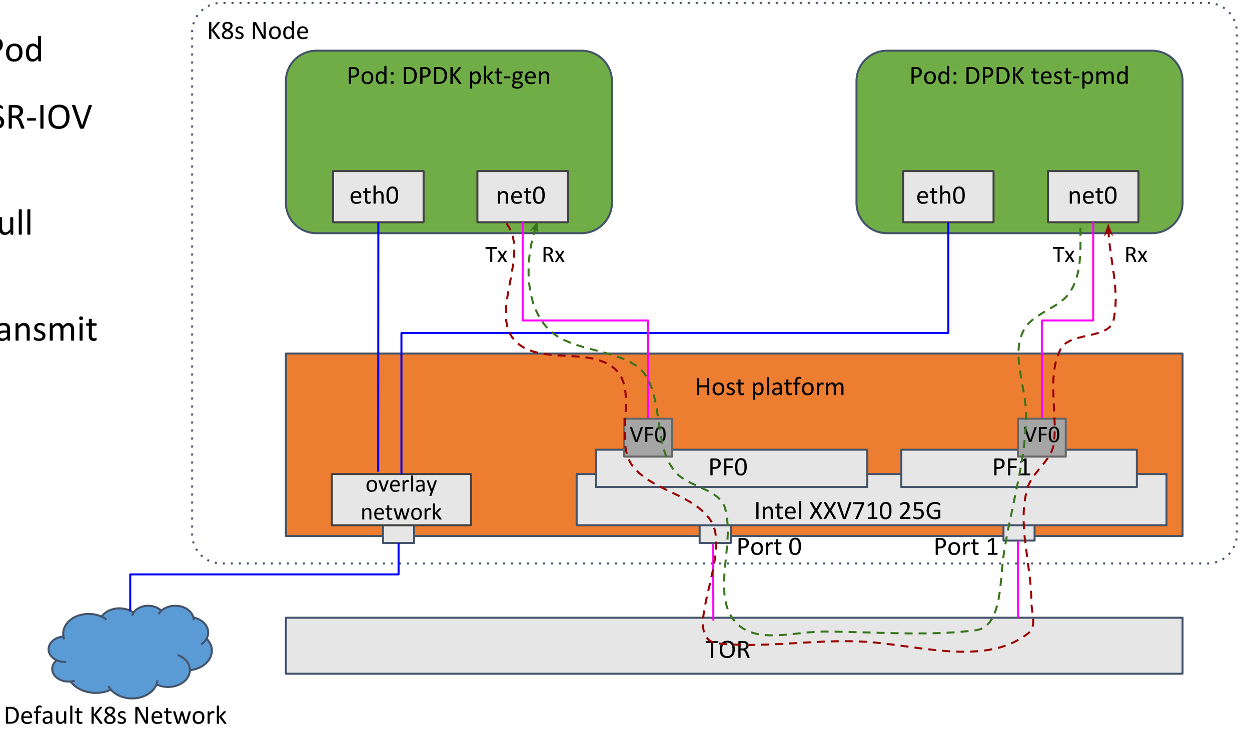
- ➢ But DPDK comes with caveats
- ➢ Guaranteed performance requires optimal resource allocation & tuning
  - ○ Exclusive CPU cores
  - ○ Direct device assignments
  - ○ Huge Page memory
  - ○ NUMA alignments
- ➢ These are the key requirements in Kubernetes for DPDK apps
  - ○ Challenge here is ***resource allocation & management***

This session about

Capabilities needed in Kubernetes to orchestrate resource critical DPDK apps and how to do it

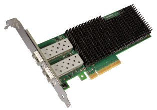
Sample deployment

- 2 DPDK apps running in K8s Pod
- Send/Receive packets using SR-IOV VFs
- Pkt-gen transmit packets at full line-rate
- Test-pmd acts as l2fwd, re-transmit packets to the src addr



Platform overview

Intel® Server Board S2600WFR (Wolf Pass Refresh)



Intel® Ethernet Network Adapter XXV710-DA2 2x25G

Feature	S2600WF0R
Platform	Intel® Server Board S2600WFR (Wolf Pass Refresh)
Processor	2 x Intel® Xeon® Platinum Cascade Lake SP 8280M Processor 28 Cores @ 2.70GHz 38.5MB L3 Cache
Chipset	Intel® C624
Memory	24 DDR4 RDIMM/LRDIMMs, 2 SPC, 12x channels/system
	2666 MT/s @ 2DPC, 72 GB total
PCIe*	Up to 8 PCIe* slots via 3 Risers, One x8 PCIe Gen 3 SAS Mezz Module
NIC (PCIe addons)	Intel® Ethernet Network Adapter XXV710-DA2 2x25G
BIOS	Vendor: Intel Corporation Version: SE5C620.86B.0D.01.0321.011120191026 Release Date: 01/11/2019

Running a DPDK app

Exclusive CPUs

NUMA aligned memory allocation

```
# testpmd -l 3,31 -n 4 -w 0000:18:02.1 -- -i --portmask=0x1 --nb-cores=2 --numa
```

Device assignment

```
EAL: Detected 56 lcore(s)
EAL: Detected 2 NUMA nodes
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode ‘VA’
EAL: No free hugepages reported in hugepages-2048kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: PCI device 0000:18:02.1 on NUMA socket 1
EAL:   probe driver: 8086:154c net_i40e_vf
EAL:   using IOMMU type 1 (Type 1)
Auto-start selected
Set macswap packet forwarding mode
testpmd: create a new mbuf pool <mbuf_pool_socket_0>: n=155456, size=2176, socket=1
Configuring Port 0 (socket 1)
Port 0: BA:AA:7A:88:F2:44
Checking link statuses...
Done
```

Required K8s Capabilities

- Exclusive CPUs
  - Native CPU manager (v1.8)
- Huge Page memory
  - Native Hugepage support (v1.8)
- Direct device assignment
  - Device Plugins (v1.8)
- Resource NUMA alignments
  - Topology manager (v1.16 alpha)

Required other components

- SR-IOV network resource management
  - github.com/intel/sriov-network-device-plugin
- SR-IOV network interface configuration
  - github.com/intel/multus-cni
  - github.com/intel/sriov-cni
- Managing configuration and deployment
  - github.com/openshift/sriov-network-operator
- Pod resource parameters
  - github.com/openshift/app-netutil
- SR-IOV network resource injector
  - github.com/intel/network-resources-injector

SW Configuration

SW	Version
Kubernetes	v1.16
Openshift	v4.3
Host OS: RedHat CoreOS	v4.3
SRIOV CNI Plugin	v2.1
SRIOV Network Device Plugin	v.3.0
SRIOV Network Operator	v4.3
Multus	v3.3
DPDK	v19.08
pktgen-dpdk	v19.08

SW Configuration

➢ Getting exclusive CPUs

- Run containers in Pod with “***Guaranteed***” QoS class
- Request exclusive CPU cores
  - ○ Run Kubelet with: ***`--cpu-manager-policy=static`***
  - ○ Add equal integer values for CPU in both `requests` & `limits`

Guaranteed Pod:

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "2"
      requests:
        memory: "200Mi"
        cpu: "2
```

SW Configuration

➢ Enable resource NUMA alignment

- Available in K8s (v1.16 alpha)
  - Works on Nodes with the static CPU Manager Policy
  - Works on Pods in the ***Guaranteed*** QoS class

- Enabling Topology manager
  - Run Kubelet with: ***--feature-gates=”TopologyManager=true”***
  - And ***--topology-manager-policy=restricted*** OR ***single-numa-node***

SW Configuration

➢ Requesting Huge Page memory

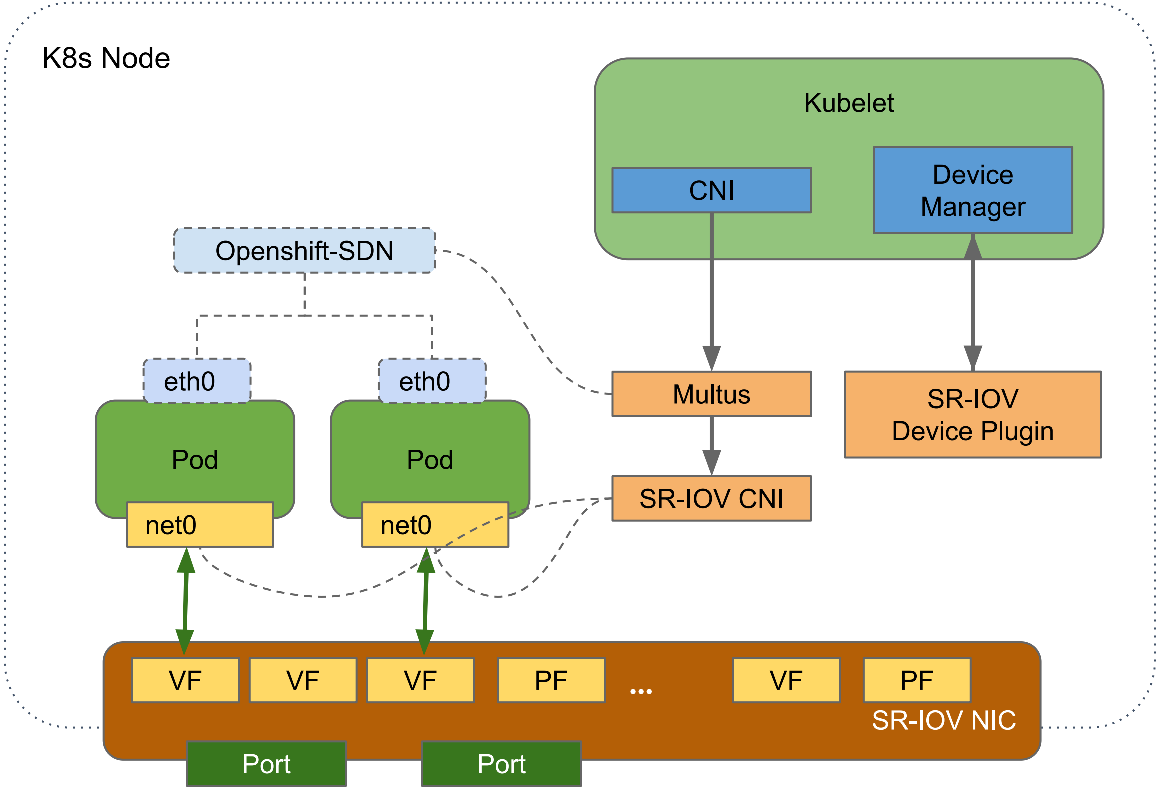
- Nodes must pre-allocate huge pages
- A node may only pre-allocate a single size
- Add huge pages as Volumes in containers

Guaranteed Pod:

```
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        hugepages-2Mi: 2G
      requests:
        hugepages-2Mi: 2G
    volumeMounts:
    - mountPath: /hugepages
volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

SR-IOV networking in K8s

- SR-IOV device plugin
  - Discovery & advertising of SR-IOV network resources
- SR-IOV CNI
  - Configure pod interface
- Multus
  - Adds SR-IOV VF as an additional Pod interface
  - Retrieves Pod device info

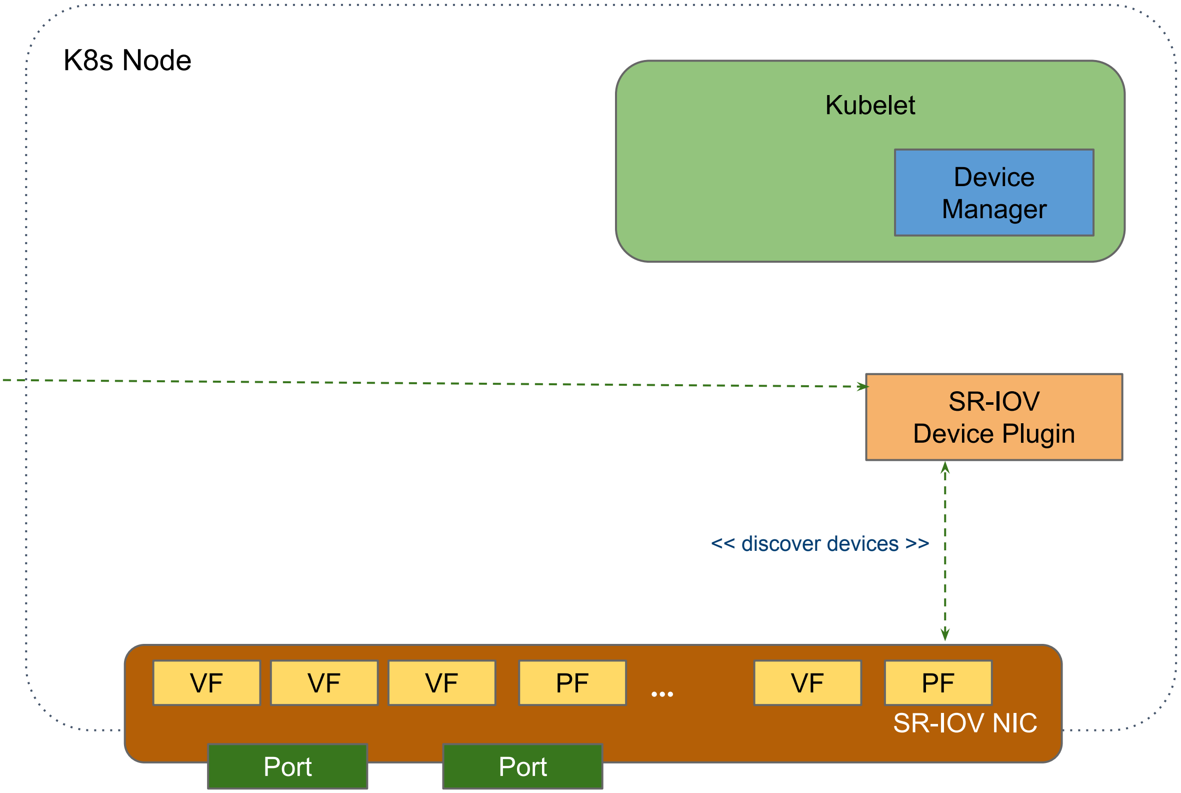


SR-IOV networking in K8s

Kube-apiserver

ConfigMap:

```
{
    "resourceList": [
        {
            "resourceName": "nic1",
                "selectors": {
                    "vendors": ["8086"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["enps803f0"]
            }
        },
        {
            "resourceName": "nic2",
            "selectors": {
                    "vendors": ["8086"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["enps803f1"]
            }
        }
    ]
}
```

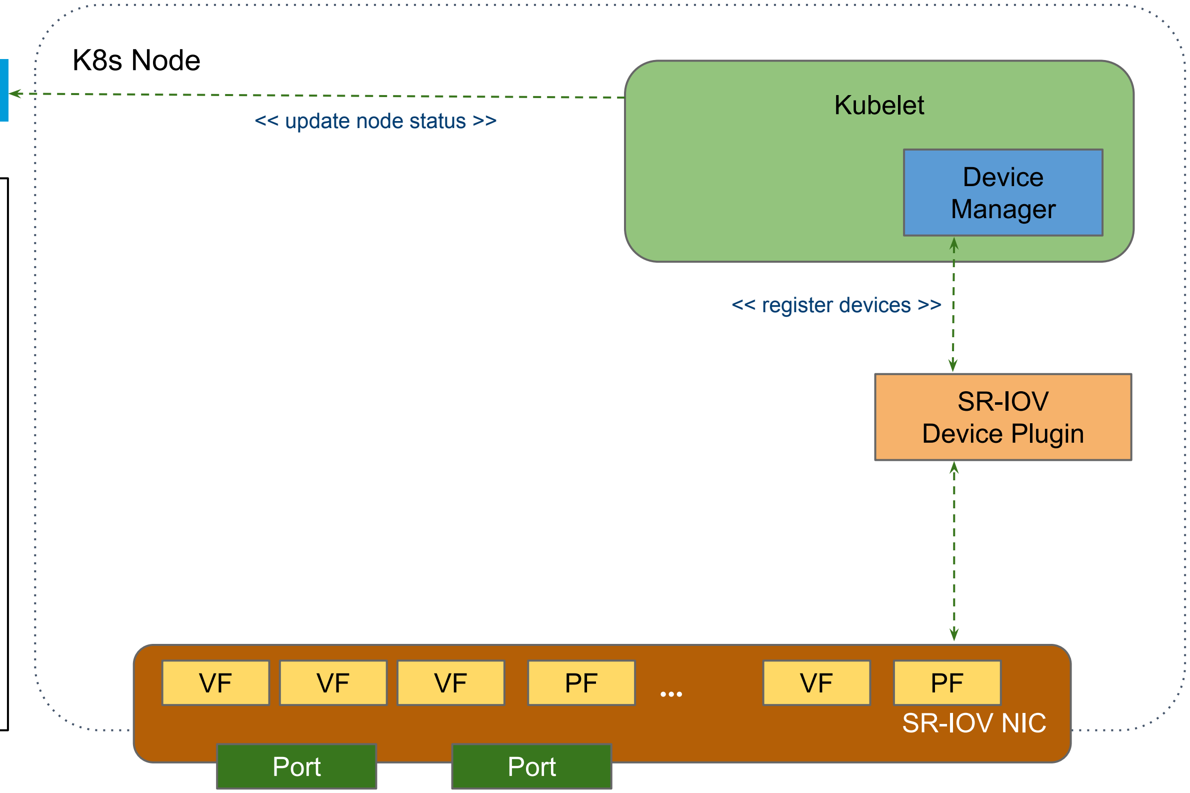


SR-IOV networking in K8s

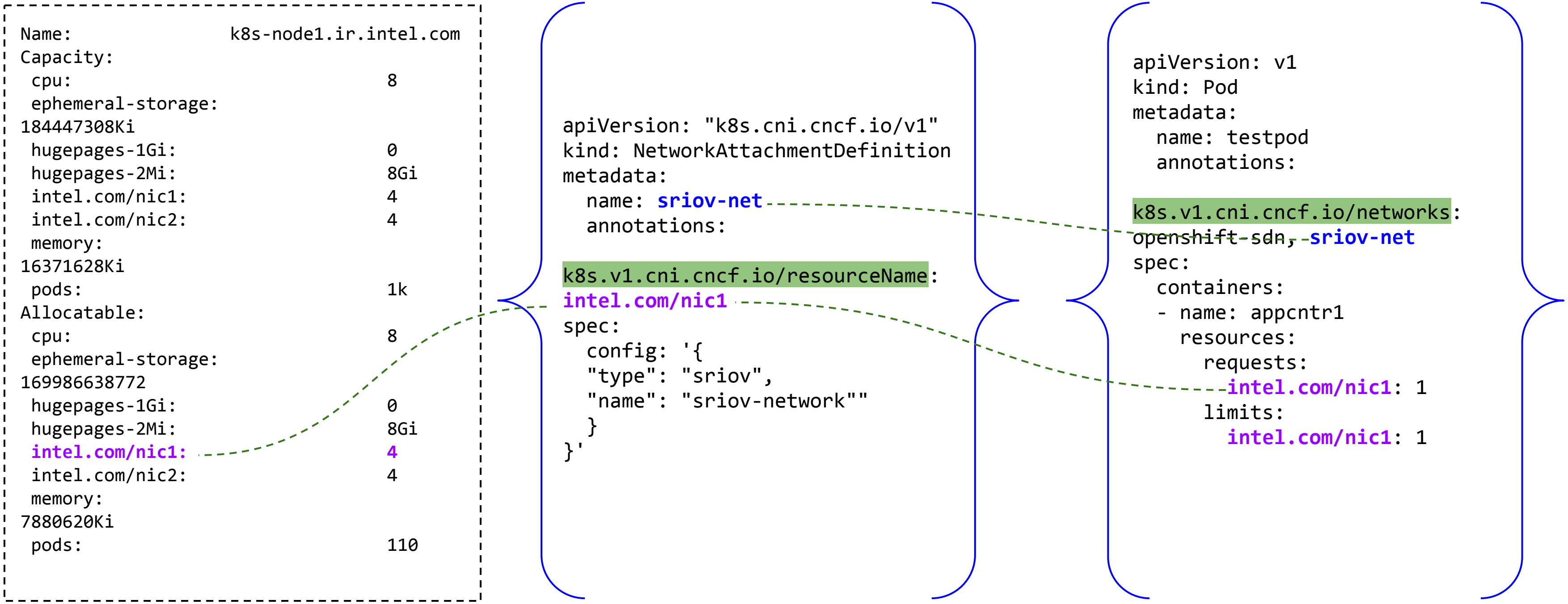
Kube-apiserver

Node status:

```
Name:                 k8s-node1.ir.intel.com
Capacity:
cpu:                                8
ephemeral-storage:                  184447308Ki
hugepages-1Gi:                      0
hugepages-2Mi:                      8Gi
intel.com/nic1:                     4
intel.com/nic2:                     4
memory:                             16371628Ki
pods:                               110
Allocatable:
cpu:                                8
ephemeral-storage:                  169986638772
hugepages-1Gi:                      0
hugepages-2Mi:                      8Gi
intel.com/nic1:                     4
intel.com/nic2:                     4
memory:                             7880620Ki
pods:                               110
```



SR-IOV networking in K8s



Node status

```
Name:                 k8s-node1.ir.intel.com
Capacity:
 cpu:                            8
 ephemeral-storage:
184447308Ki
 hugepages-1Gi:                  0
 hugepages-2Mi:                  8Gi
 intel.com/nic1:                 4
 intel.com/nic2:                 4
 memory:
16371628Ki
 pods:                           1k
Allocatable:
 cpu:                            8
 ephemeral-storage:
169986638772
 hugepages-1Gi:                  0
 hugepages-2Mi:                  8Gi
 intel.com/nic1:                 4
 intel.com/nic2:                 4
 memory:
7880620Ki
 pods:                           110
```

Net-attach CRD

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net
  annotations:

k8s.v1.cni.cncf.io/resourceName:
intel.com/nic1
spec:
  config: '{
  "type": "sriov",
  "name": "sriov-network""
  }
}'
```

Pod Specs

```
apiVersion: v1
kind: Pod
metadata:
  name: testpod
  annotations:

k8s.v1.cni.cncf.io/networks:
openshift-sdn, sriov-net
spec:
  containers:
  - name: appcntr1
    resources:
      requests:
        intel.com/nic1: 1
      limits:
        intel.com/nic1: 1
```

SR-IOV Network Operator

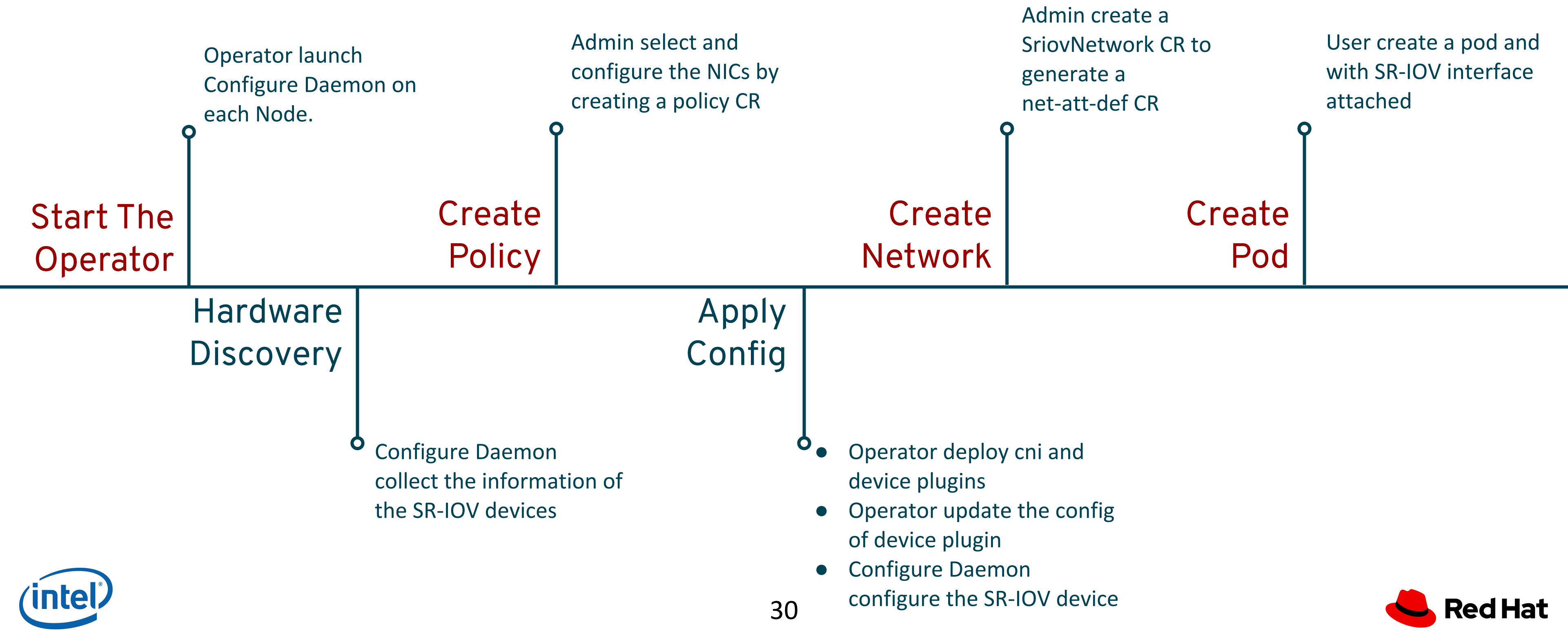
Simplifying SR-IOV networking in K8s with **SR-IOV network operator**

SR-IOV Network Operator

- SR-IOV Network Operator is a tool to hide the complexity of the adopting SR-IOV in K8S. It automates
  - Life-cycle management of SR-IOV software components (SR-IOV CNI plugin, SR-IOV network device plugin, network resource injector …)
  - The configuration management of the SR-IOV software components.
  - The SR-IOV network device configuration
    - Hardware discovery
    - Device configuration
    - Kernel driver management
  - ***NetworkAttachmentDefinition*** CR generation
- Built for Openshift, can also work with vanilla K8S

SR-IOV Network Operator

Operator Workflow



Start The Operator

Operator launch Configure Daemon on each Node.

Hardware Discovery

Configure Daemon collect the information of the SR-IOV devices

Create Policy

Admin select and configure the NICs by creating a policy CR

Apply Config

- Operator deploy cni and device plugins
- Operator update the config of device plugin
- Configure Daemon configure the SR-IOV device

Create Network

Admin create a SriovNetwork CR to generate a net-att-def CR

Create Pod

User create a pod and with SR-IOV interface attached

SR-IOV Network Operator

SriovNetwork

```
apiVersion:
sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-net
  namespace: sriov-network-operator
spec:
  networkNamespace: default
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
    }
  resourceName: nic1
```

Net-attach CRD

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/resourceName:
intel.com/nic1
spec:
  config: '{
  "type": "sriov",
  "name": "sriov-network",
  "ipam": {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
    }
  }'
```

SR-IOV Network Operator

SriovNetworkNodePolicy

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: sriov-network-operator
spec:
  deviceType:           vfio-pci
  numVfs:               4
  mtu:                  1500
  Priority:             90
  resourceName: nic1
  nicSelector:
    rootDevices:
    - 0000:86:00.1
    vendor:             "8086"
    pfName:             "enps803f0"

  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable:
"true"
```

Config options for selected VFs:

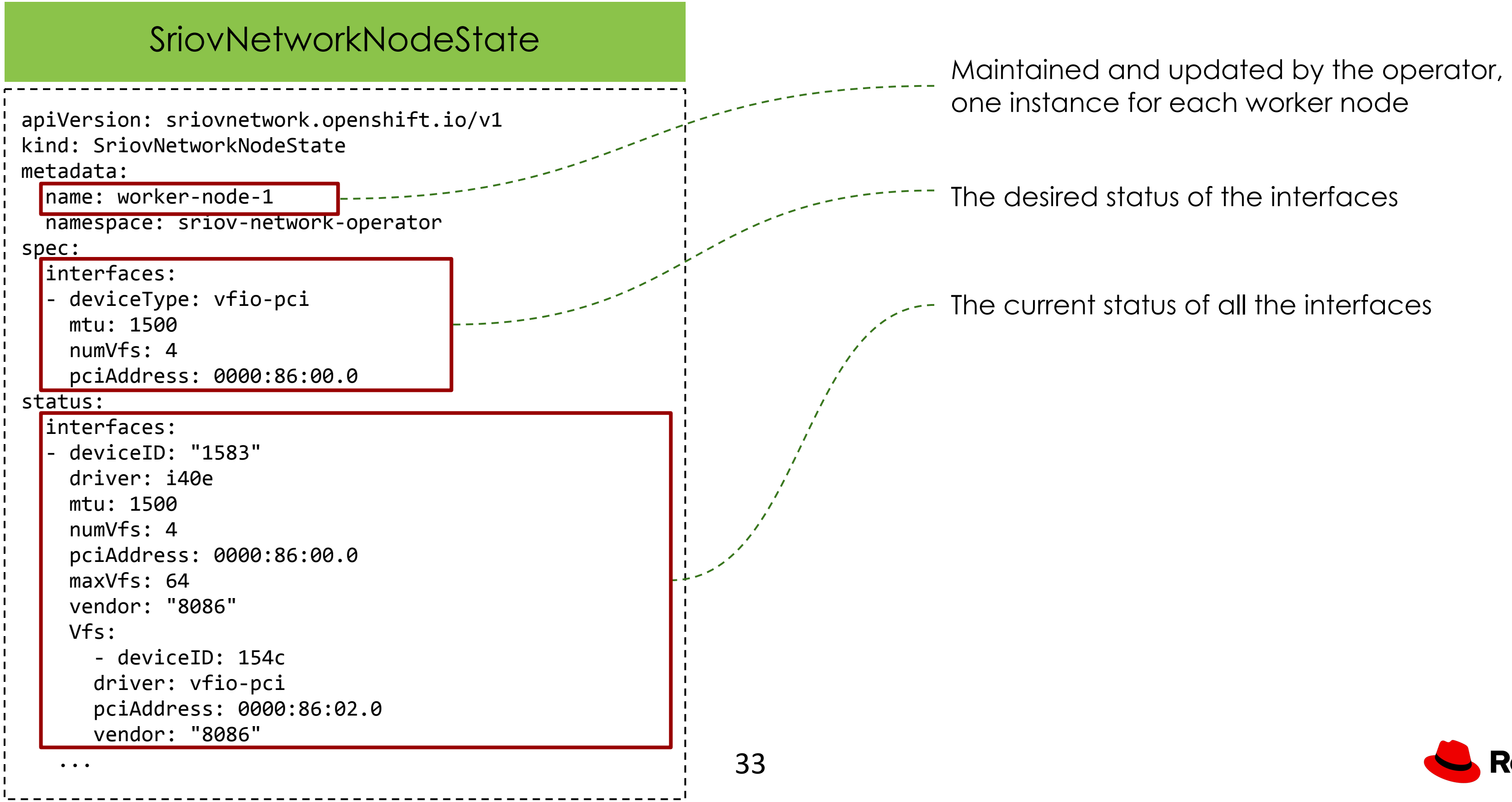
- Number of VFs
- MTU
- Kernel driver, either ‘netdevice’ or ‘vfio-pci’

Select NICs which need to be configured, and generate SRIOV device plugin config

```
{
    "resourceList": [
        {
            "resourceName": "nic1",
                "selectors": {
                    "vendors": ["8086"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["enps803f0"]
            }
        }
}
```

Select Nodes to be configured

SR-IOV Network Operator



SriovNetworkNodeState

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: worker-node-1
  namespace: sriov-network-operator
spec:
  interfaces:
  - deviceType: vfio-pci
    mtu: 1500
    numVfs: 4
    pciAddress: 0000:86:00.0
status:
  interfaces:
  - deviceID: "1583"
    driver: i40e
    mtu: 1500
    numVfs: 4
    pciAddress: 0000:86:00.0
    maxVfs: 64
    vendor: "8086"
    Vfs:
      - deviceID: 154c
      driver: vfio-pci
      pciAddress: 0000:86:02.0
      vendor: "8086"
   ...
```

Maintained and updated by the operator, one instance for each worker node

The desired status of the interfaces

The current status of all the interfaces

Demo

- Video: https://youtu.be/scp2WV5M3TI
- Sample manifests: https://github.com/pliurh/Kubecon2019-DEMO

What needs attention

- Isolated CPU support
  - Kubelet unable to manage isolated CPU cores
  - Multiple OOT solution exist
  - None of these are well integrated with native CPU manager

- Topology aware scheduling
  - K8s default scheduler unaware of resource Topology information
  - May results in Pod scheduled in non-viable nodes

Summary

- ➢ A lot works has been done in the community to support Telco and 5G use-cases in Kubernetes
- ➢ Kubernetes is “***5G Ready***”
- ➢ Some areas still need attention and wider community collaboration

References

- http://core.dpdk.org/perf-reports/
- https://github.com/intel/multus-cni
- https://github.com/intel/network-resources-injector
- https://github.com/intel/sriov-cni/
- https://github.com/intel/sriov-network-device-plugin
- https://github.com/openshift/app-netutil
- https://github.com/openshift/sriov-network-operator
- https://github.com/pktgen/Pktgen-DPDK
- https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/
- https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/
- https://kubernetes.io/docs/tasks/administer-cluster/topology-manager/
- https://kubernetes.io/docs/tasks/manage-hugepages/scheduling-hugepages/
- https://www.dpdk.org/
- https://www.intel.com/content/www/us/en/products/servers/server-chassis-systems/server-board-s2600wf-systems.html

Additional Resources

Please visit Intel® Network Builders site for Bare-metal Containers Experience Kits

https://networkbuilders.intel.com/network-technologies/container-experience-kits