



# **Turbocharge Kubernetes with offloaded OVN-Kubernetes and OVS on DPUs**

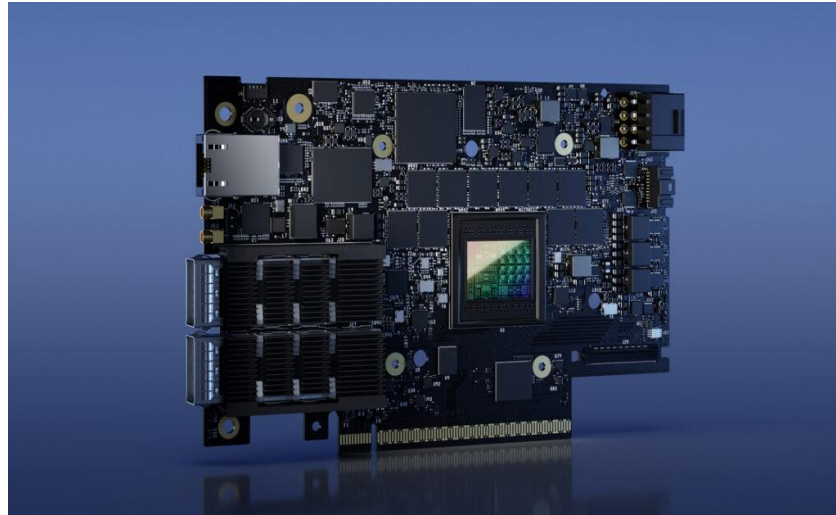
Alin Serdean, NVIDIA; Amit Zala, NVIDIA; Mael Kimmerlin, NVIDIA; Igal  
Tsoiref, RedHat , 20 Nov 2025

# Agenda

- What is a DPU?
- Why did we build DPF?
- SC (Service Chaining)
- Hardware acceleration using OVN-K8S CNI
- Redhat Integration

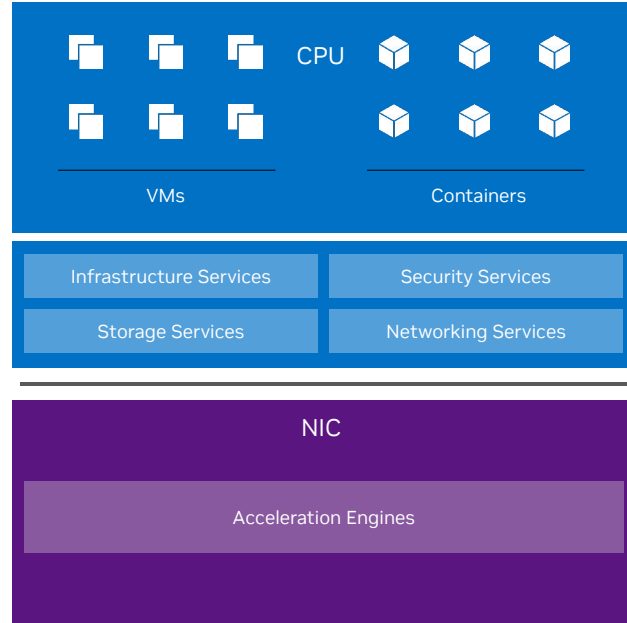
# DPU

- Bluefield DPU accelerates and offload infrastructure tasks like networking, storage, and security in data centers
- It allows CPU x86 cores to focus on handling workloads by freeing it from doing infrastructure tasks
- The BlueField-3 DPU features up to 16 powerful Arm cores and dual 200GbE physical ports, delivering an impressive 400Gb/s line rate for networking traffic.

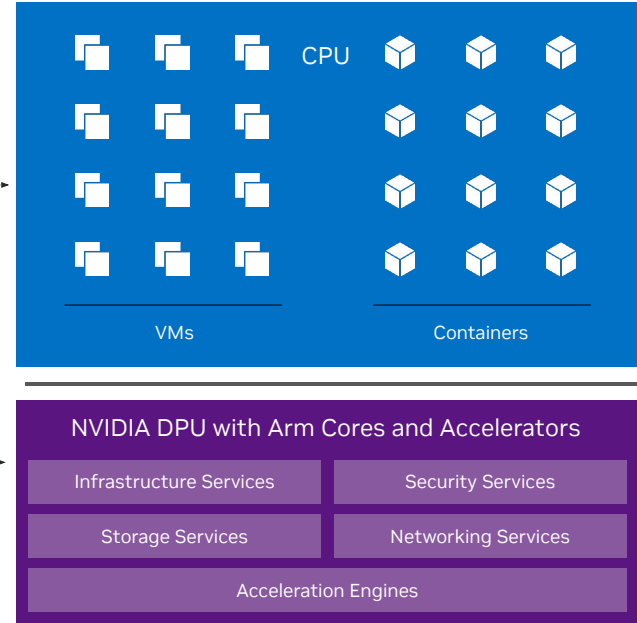


# DPU – Infra Services Execution Engine

TRADITIONAL SERVER



DPU-ACCELERATED SERVER



# Pain points of managing DPU at scale - Life Without DPF

- **Manual Chaos at Scale**

- Provisioning 100+ DPUs? SSH to each, run custom scripts, pray nothing breaks
- One typo in BFB config → re-flash and start over (30+ min per DPU)

- **Service Deployment Nightmare**

- Lifecycle management of services(system services + custom 3<sup>rd</sup> party services) running on DPU
- No standard way to connect services → tribal knowledge, fragile scripts

- **Zero Visibility**

- Which DPUs are alive? Which services crashed? Manual checking or wait for alerts
- Resource usage? Unknown until DPU OOMs

- **Upgrade Terror**

- Incompatibility Failure → Many moving pieces(BFB/services upgrade), no mechanism to verify compatibility
- Zero Rollout Safety → No control over update speed or scale

- **Specific use cases**

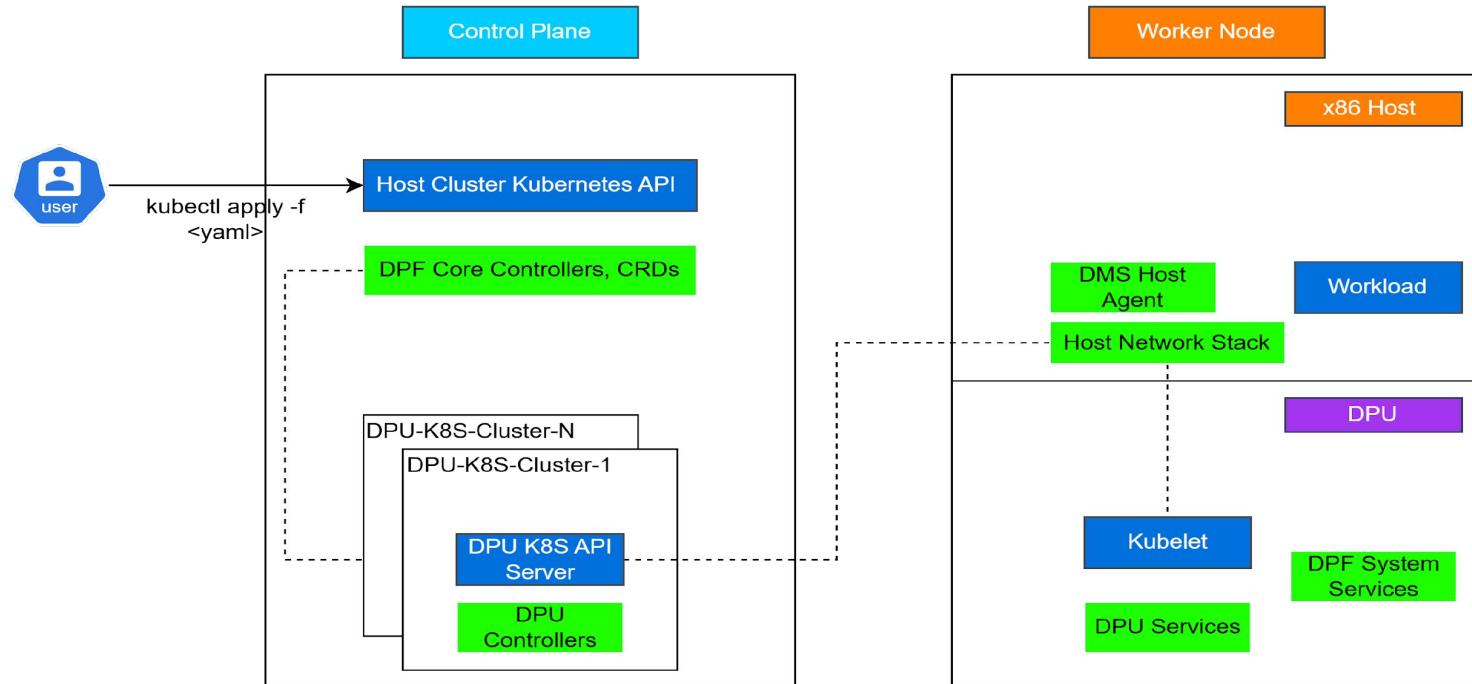
- Multi-tenancy, hardware acceleration for workload etc

# DPF to the Rescue

- **BFB Provisioning:**
  - **One DPUSet CR** → provision entire DPU fleet with different DPU flavor and BFB file
  - **Automated rollout** → DPF handles staging, verification, rollback
- **Service Orchestration:**
  - **DPUService CR** → deploy any containerized service (HBN, OVN, custom 3<sup>rd</sup> party apps)
  - **DPUServiceChain CR** → connect services with YAML: p0 → firewall → HBN
- **Real-Time Observability:**
  - **Centralized dashboard** → DPU health, service status, resource utilization
  - **Automated fleet health checks** → know problems before users do
- **Safe Upgrades:**
  - **Declarative Rollouts** → DPUDeployment manages unified upgrade of both BFB and DPU services.
  - **Version Compatibility Check** → Blocks incompatible updates between DPU Service and BFB versions
  - **Controlled Velocity** → Rolling updates with a maxUnavailable budget for predictable, low-disruption rollouts
- **Built-in use cases:**
  - **Multi-tenancy** → HBN
  - **Hardware acceleration** of workload pods → OVN-K8S

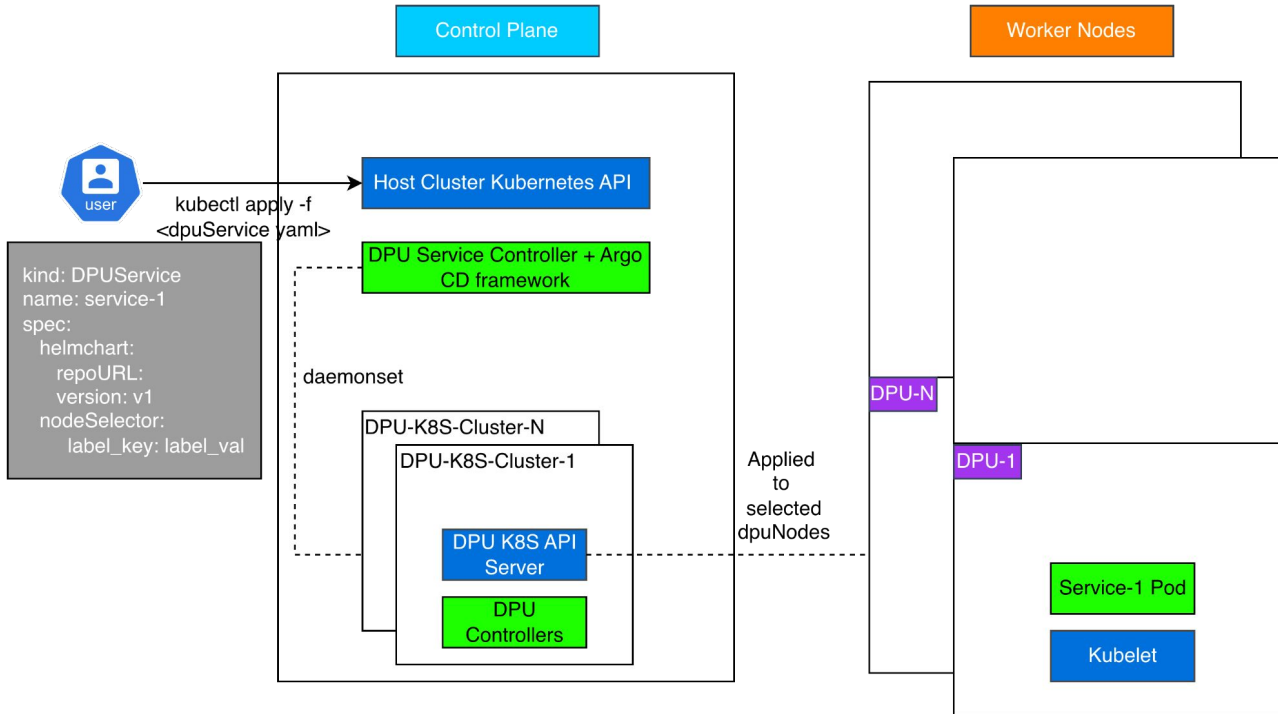
# DPF – DOCA Platform Framework

- Platform to manage the lifecycle of DPU and services running on DPU
- Host cluster to manage the workloads running on the x86 node and DPU cluster to manage the DPUs and DPUServices
- DPU clusters are spawn within the main host cluster, these are Kamaji (but could be anything else as well) based small DPU clusters.



# DPF Services

- Abstraction for pods running on the DPU
- Lifecycle management of services via upstream k8s ArgoCD framework
- Along with DPF system services, user can run 3<sup>rd</sup> party services as well, user needs to create/define Helm chart for the service and use DPUService abstraction.





# SC - Service Chaining

- SC allows you define a pipeline in between DPUServices
- SC is an OpenFlow controller
- DPUServiceNAD – allows you to define where to plug resources and the resource type
- DPUServiceInterface - defines reusable network interface templates that DPUServices reference to declare their connectivity requirements
- DPUServiceChain - orchestrates OpenFlow rules in OVS bridge on DPUs by chaining service interfaces together (service-to-service or service-to-uplink or uplink-to-service)

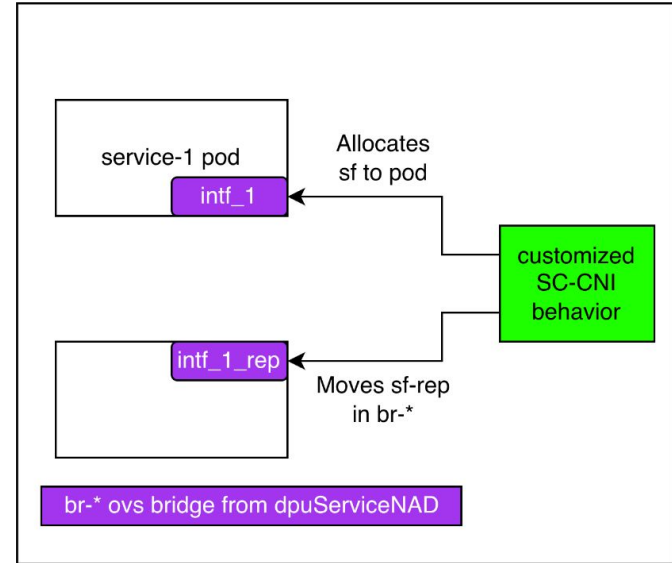
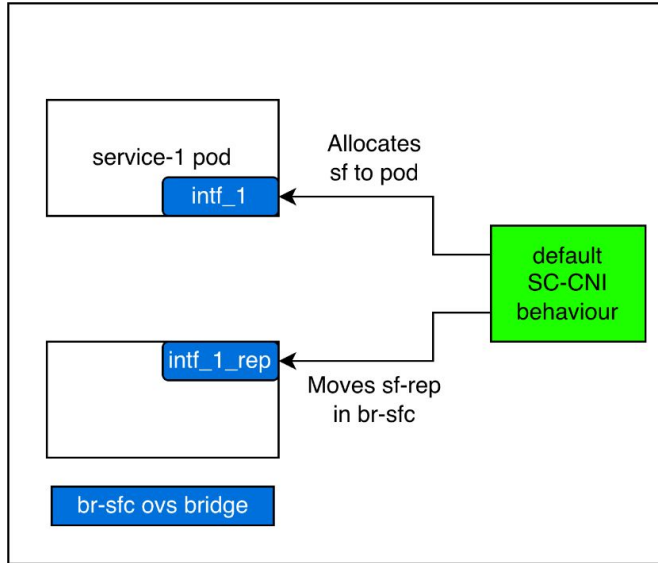
```
kind:DPUServiceInterface
spec:
  labels:
    interface: intf_1
  interfaceType: service
  service:
    serviceid: service-1
  ifname: intf_1
```

```
kind:DPUServiceChain
spec:
  switches:
    - ports:
        - serviceInterface:
            matchLabels:
              uplink: p1
        - serviceInterface:
            matchLabels:
              interface:intf_1
```

```
kind:DPUServiceNAD
spec:
  bridge: br-sfc
  ipam: true
  resourceType: sf
  serviceMTU: 1500
```

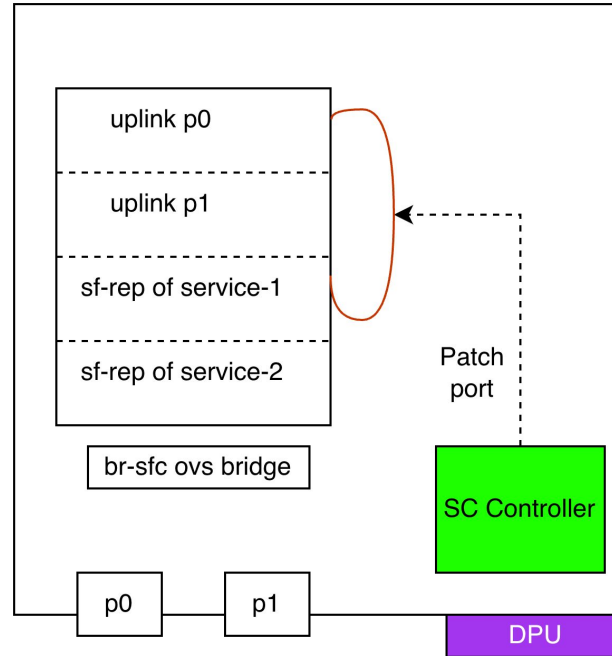
# SC CNI

- When SC CNI allocates sf to the DPUService pod and adds the sf-rep into the OVS bridge (default br-sfc, but can change based on DPUServiceNAD configuration), it also adds metadata specific to the DPFService on the corresponding OVS port.

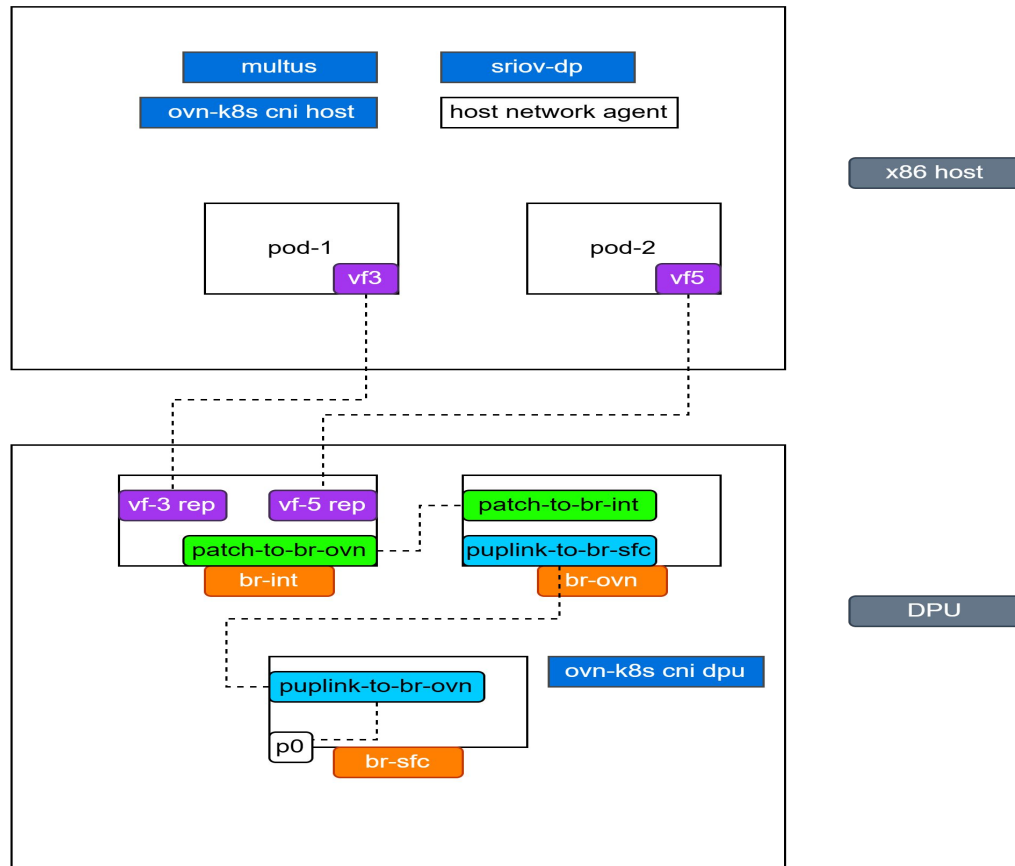


# SC controller

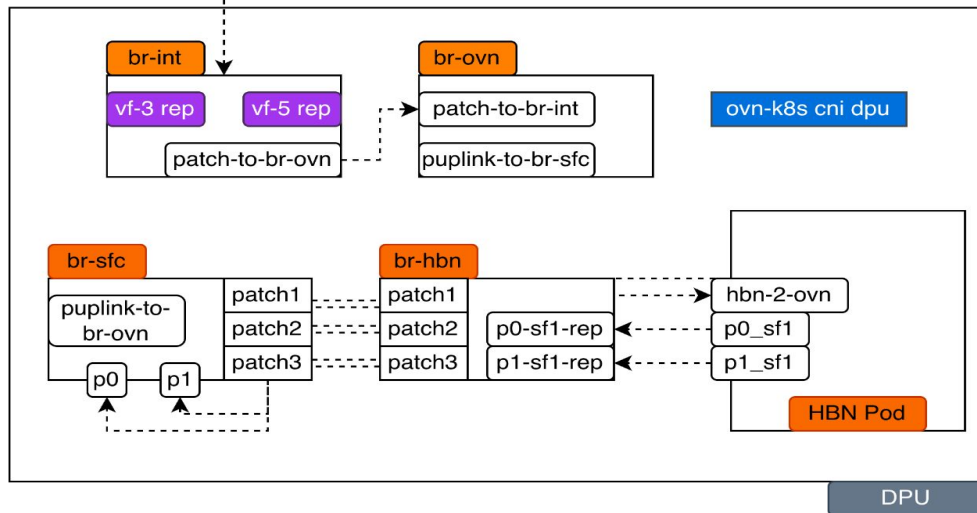
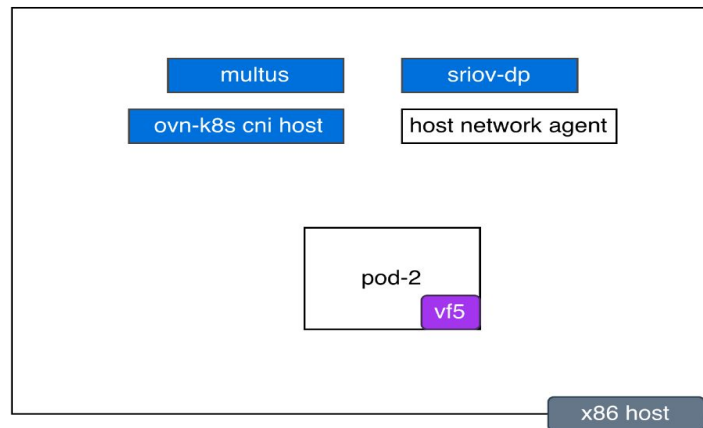
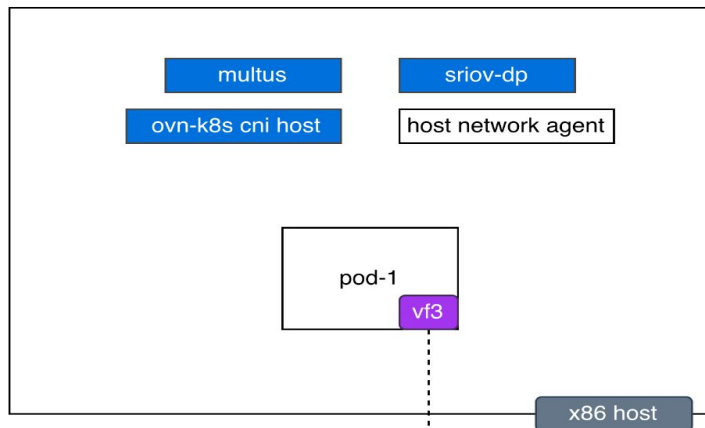
- When SC controller is looking at the chain abstraction ,it can filter through the metadata and can figure out it needs to create a patch paneling between ovs-port1 and ovs-port2.



# Hardware acceleration of workload pods using OVN-K8S CNI



# Hardware acceleration of workload pods using OVN-K8S + HBN



br-int	ovn-controller
br-ovn	ovn-kube-node
br-sfc	sc-cni + sc-controller
br-hbn	sc-cni

# Perf numbers

```

# perf testing
# perf testingcat iperf-throughput_log | head -n 50
Test 1/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
382.3
Test 2/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
333.45
Test 3/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
396.62
Test 4/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
329.21
Test 5/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
371.32
Test 6/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
351.64
Test 7/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
377.1
Test 8/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
379.47
Test 9/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
350.94
Test 10/10
Running iperf3 test with size of 256K for 120 sec on cores 35-55
TOTAL THROUGHPUT
379.47

```

iperf throughput

memory

cpu

power consumption

cat /tmp/perf\_memory\_log

Linux 4.8.0-70-generic (noevec)

09/04/25

\_x86\_64\_

(112 CPU)

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

Linux 4.8.0-70-generic (noevec)

09/04/25

\_x86\_64\_

(112 CP

U)

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

0.000000

Power measurements will start in 0 seconds time

Time User Nice Sys Idle IO Run Cxt/s Iq/s Fork Exec Exit Watts

13:35:27 0.1 0.0 8.9 90.9 0.0 21 118170 232102 177 168 178 415.89

13:35:27 0.1 0.0 9.0 89.9 0.0 1 168481 365336 261 176 208 415.84

13:35:27 0.1 0.0 9.0 89.8 0.0 21 168324 262904 176 168 177 419.13

13:40:27 0.2 0.0 10.1 89.9 0.0 20 166922 299635 266 177 202 420.39

13:40:27 0.1 0.0 10.1 89.8 0.0 15 166596 291482 176 168 173 419.40

13:41:27 0.2 0.0 10.0 89.8 0.0 14 165888 268662 266 176 202 420.18

13:41:27 0.1 0.0 8.8 91.1 0.0 16 161535 266966 186 168 177 409.88

13:42:27 0.2 0.0 9.6 90.3 0.0 17 118973 261876 176 168 174 417.98

13:42:27 0.1 0.0 9.6 90.3 0.0 11 116267 262616 153 143 156 417.69

13:43:27 0.2 0.0 9.6 90.2 0.0 19 119394 262400 263 176 203 417.99

13:43:27 0.1 0.0 8.9 90.9 0.0 19 86661 256153 176 168 177 407.69

13:44:27 0.1 0.0 9.7 90.2 0.0 18 96699 276895 263 176 203 414.58

13:44:27 0.1 0.0 9.7 90.2 0.0 14 86661 276827 176 168 177 414.94

13:45:27 0.1 0.0 9.7 90.1 0.0 17 91165 279015 266 176 207 415.23

13:45:27 0.1 0.0 8.3 91.6 0.0 17 95511 259618 181 168 175 404.80

13:46:27 0.2 0.0 8.8 91.1 0.0 22 167610 282327 262 176 205 412.31

13:46:27 0.2 0.0 8.8 91.1 0.0 11 166249 279241 176 168 176 411.40

13:47:27 0.2 0.0 8.8 91.0 0.0 20 167056 280168 260 176 201 411.54

13:47:27 0.1 0.0 9.1 90.7 0.0 18 88082 261495 182 168 176 407.88

13:48:27 0.1 0.0 10.3 89.5 0.0 18 85992 282297 266 176 206 417.95

13:48:27 0.1 0.0 10.4 89.5 0.0 14 85986 268554 176 168 175 419.37

13:49:27 0.1 0.0 10.4 89.5 0.0 18 85414 279554 261 176 206 410.74

13:49:27 0.1 0.0 9.4 90.5 0.0 23 116269 295710 183 168 177 409.85

13:50:27 0.2 0.0 10.0 89.8 0.0 20 136442 342649 266 177 203 416.53

13:50:27 0.2 0.0 10.0 89.8 0.0 20 137976 342649 266 176 202 416.69

13:51:27 0.2 0.0 10.0 89.8 0.0 15 136714 342336 262 176 202 417.70

13:51:27 0.1 0.0 9.3 90.6 0.0 19 115017 292443 152 143 156 409.64

13:52:27 0.2 0.0 10.1 89.8 0.0 10 112539 294618 182 174 203 416.19

13:52:27 0.1 0.0 10.1 89.8 0.0 16 112893 295833 168 168 176 416.37

13:53:27 0.2 0.0 10.1 89.8 0.0 13 112123 293779 264 176 206 416.70

13:53:27 0.1 0.0 8.8 91.0 0.0 9 98792 271269 182 168 177 407.47

13:54:27 0.1 0.0 9.1 90.8 0.0 18 161032 294794 261 176 203 413.46

Time User Nice Sys Idle IO Run Cxt/s Iq/s Fork Exec Exit Watts

13:54:27 0.1 0.0 9.1 90.8 0.0 24 161138 295708 177 168 183 414.93

13:55:27 0.1 0.0 9.1 90.8 0.0 20 161035 294954 266 176 210 413.78

13:55:27 0.1 0.0 4.9 95.0 0.0 1 57218 163131 177 168 182 372.10

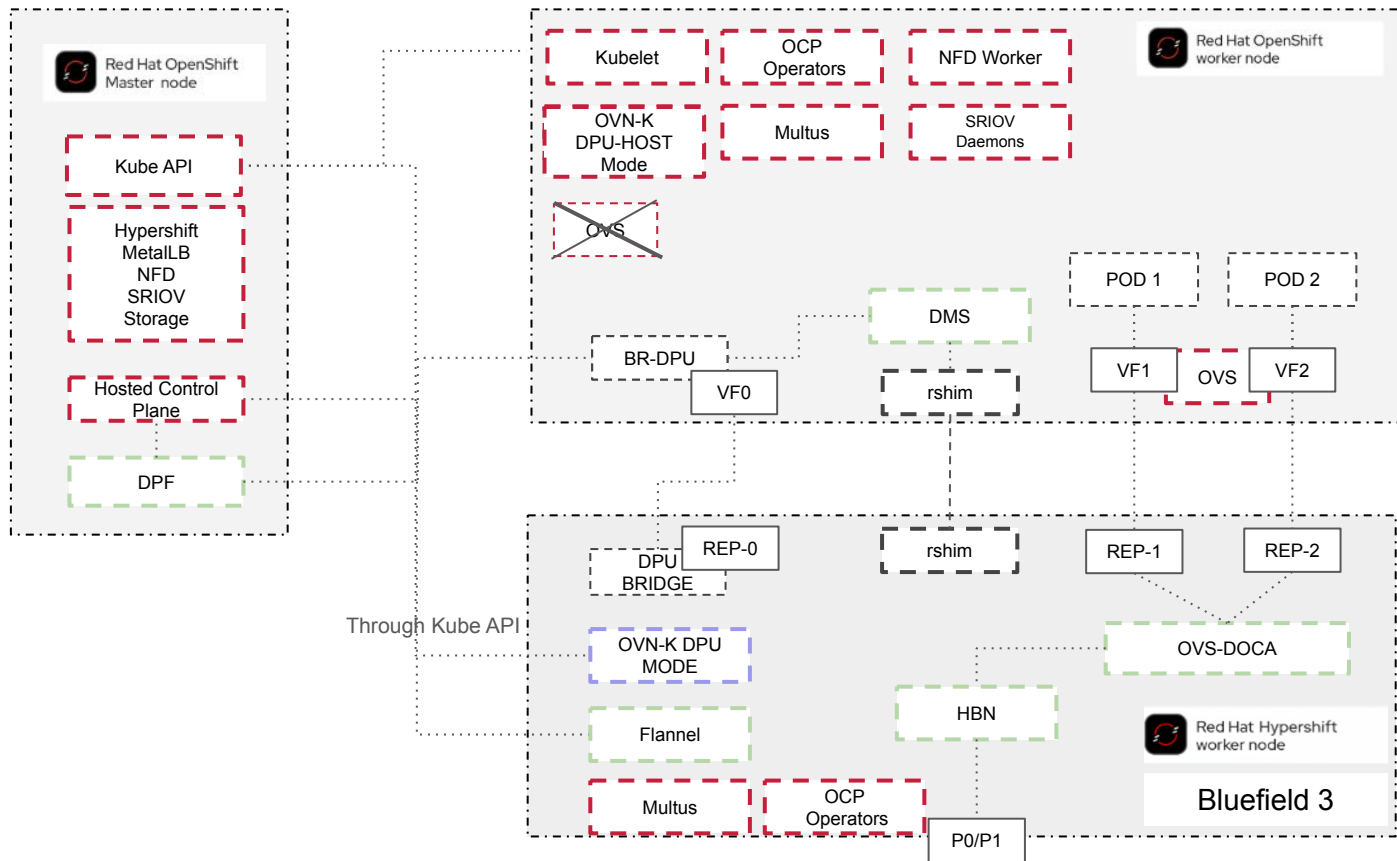
13:56:27 0.0 0.0 0.0 100.0 0.0 1 4181 4691 261 176 206 317.00

# Integration Challenges

- Utilizing the Cluster Network Operator to activate dpu-host mode.
- Migrating from a legacy custom downstream image for OVN-Kubernetes to the current upstream version.
- Adopting HyperShift for managing hosted clusters.
- Standardizing on RHCOS instead of Ubuntu.
- Adapting all necessary operators from upstream versions to their OpenShift counterparts, incorporating changes to support the new model.
- Addressing the stricter OpenShift security model, which requires per-pod adaptation compared to vanilla Kubernetes.



# OpenShift Networking with NVIDIA BlueField-3 and DPF





# Cluster Network Operator DPU-Host Changes

- Successfully restored and validated DPU-Host mode functionality after ~2 years of inactivity.
- Defined and enforced a specific list of compatible OVN-Kubernetes features for DPU-Host mode deployments.

## openshift/**cluster-network-operator**

Create and manage cluster networking configuration



168

Contributors

58

Used by

107

Stars

263

Forks



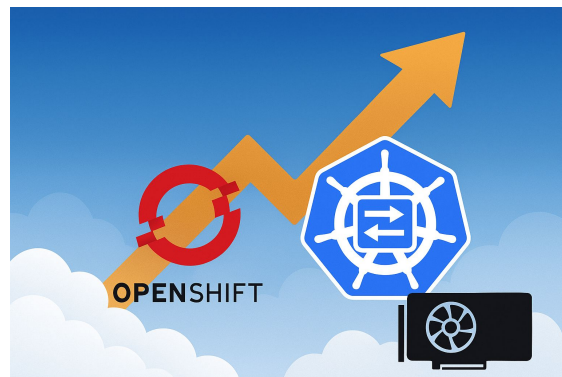
# OVN-Kubernetes: Shifting to Upstream first mentality

- Key challenges were lack of CI, constant shifting codebase. The required DPU changes were in core codebase sections.
- A two-cluster model was introduced.
- OVN-Kubernetes is now aware of both slow and high-speed networks.
- Currently supporting a small set of features
- All workloads on the DPU host are offloaded. There is a need to have hybrid workloads (with both accelerated and non accelerated traffic).



# OVN-Kubernetes Future Work

- OVN-Kubernetes as Primary CNI for DPU Cluster
- Transition away from Flannel dependency
  - Support multiple OVN-Kubernetes instances (2+) on a single host
  - POC of enabling two instances OVN-Kubernetes on a single node with different modes was done already
- Enable RDMA accelerated traffic in Openshift
  - Unlock high-performance networking capabilities for data-intensive workloads
- Enable the currently untested features:
  - Egress IP, QoS, Admin Network Policy, Node Identity ...
- Selective Pod Offloading
  - Explore on-demand pod offloading strategy
  - Move from "offload everything" to targeted offloading based on workload requirements



# HyperShift as Dpu Cluster

- Due to the resource constraints of the DPU, some functionalities from HyperShift had to be disabled:
  - Insights, Console, Openshift Samples, Ingress, NodeTuning
  - Reduced ~10GB of memory
- Instead of using the default primary CNI of HyperShift (ovn-kubernetes), this deployment uses Flannel. The reason behind this decision is that two instances of ovn-kubernetes can't run on the DPU at the same time.
- Moving from Ubuntu to RHCOS
- Security requirements
  - Needed to create SCC for each and every pod that DPF uses on Hosted Cluster.

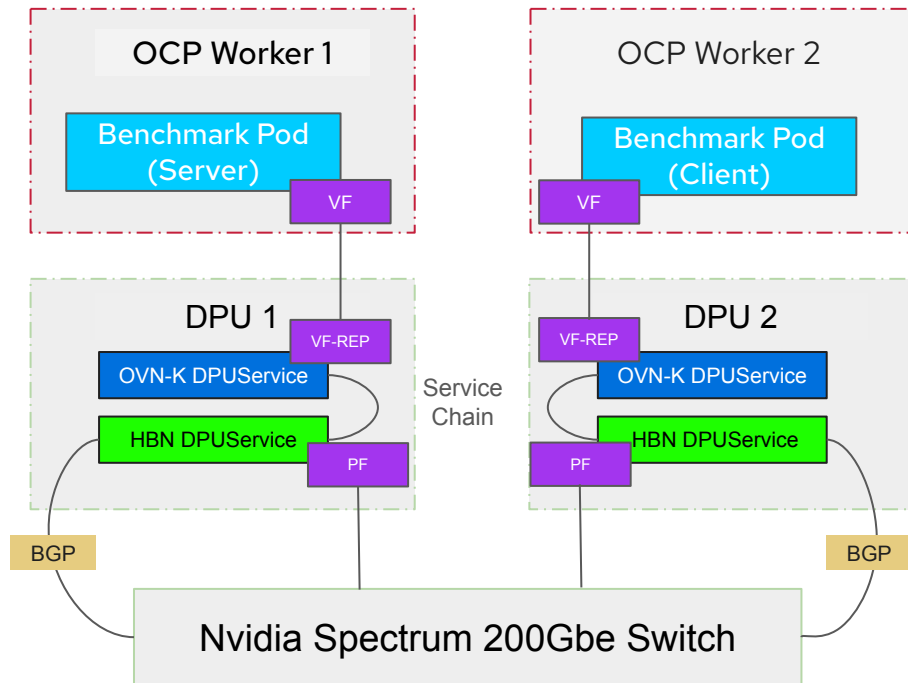
# Performance benchmarks results

## iPerf Test (iperf3 client)

```
$ ./iperf_client.sh 10.128.3.144 33-62 30
```

```
Results for port 5309:  
  Bandwidth: 8.804 Gbit/sec  
Results for port 5311:  
  Bandwidth: 13.361 Gbit/sec  
Results for port 5313:  
  Bandwidth: 8.800 Gbit/sec  
Results for port 5315:  
  Bandwidth: 8.784 Gbit/sec  
Results for port 5317:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5319:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5321:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5323:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5325:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5327:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5329:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5331:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5333:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5335:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5337:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5339:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5341:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5343:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5345:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5347:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5349:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5351:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5353:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5355:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5357:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5359:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5361:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5363:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5365:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5367:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5369:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5371:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5373:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5375:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5377:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5379:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5381:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5383:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5385:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5387:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5389:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5391:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5393:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5395:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5397:  
  Bandwidth: 8.799 Gbit/sec  
Results for port 5399:  
  Bandwidth: 8.799 Gbit/sec  
Total Bandwidth across all streams: 354.924 Gbit/sec  
### Iperf Test (iperf3 Client) Complete ###
```

354.92  
Gbit/Sec



# Deployment and Benchmark Demo

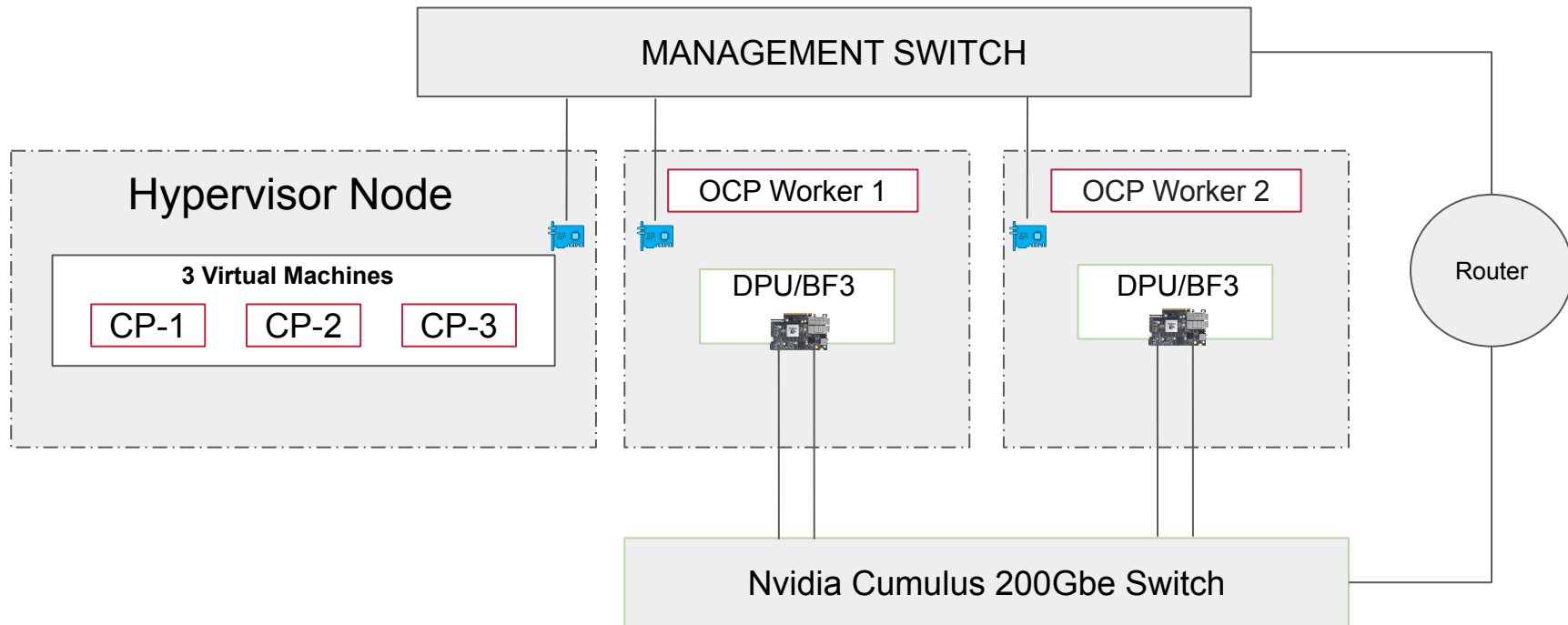
The screenshot shows the Red Hat Hybrid Cloud Console interface. The left sidebar contains navigation links for OpenShift, Overview, Cluster Management, Dashboard, Cluster List, Advisor, Vulnerability Dashboard, Subscriptions Usage, Cost Management, Products, Advanced Cluster Security, OpenShift AI, OpenShift Virtualization, Resources, Learning Resources, Developer Sandbox, Downloads, and Releases. The top navigation bar includes the Red Hat logo, 'Red Hat Hybrid Cloud Console', a 'Services' dropdown, a search icon, and a user profile 'Sagi Zigmon'. The main content area is titled 'Cluster List > doca-cluster' and features an 'Open console' button and an 'Actions' dropdown. A yellow warning banner indicates the 'OpenShift evaluation expiration date' on 20 Apr 2025. Below this, a blue box prompts to 'Optimize your cluster with operators'. The 'Add Hosts' tab is selected, showing an 'Add hosts' button and an 'Information & Troubleshooting' section. A table of discovered hosts is displayed with the following data:

Hostname	Role	Status	Discovered on	CPU Cores	Memory	Total storage
958795f6-4e88-4a9e-d88f-c7d0d694355d	Worker	Discovering	2/19/2025, 1:35:19 PM	-	-	-
nvd-av-24-nvidia-eng-rdu2.dc.redhat.com	Worker	Discovering	2/19/2025, 1:35:19 PM	64	256.00 GiB	736 TB

At the bottom of the 'Add Hosts' tab, there are buttons for 'Install ready hosts' and 'View cluster events'. A 'Feedback' button is located in the bottom right corner of the console interface.



## Lab physical connectivity





# Journey of supporting RHCOS

- We created a workflow built from scratch [coreos/custom-coreos-disk-images](#).
- First Installer:
  - Used a crude OS with rootfs derived from ubi9 (no systemd) and just dd'd the raw disk image onto the NVMe.
  - Made Nvidia's **bootfido** device driver carry ignition instead of their usual bf.cfg file.
    - Ignition is the first-boot provisioning tool in RHCOS that uses a declarative configuration file to set up a machine during its initial startup
  - Early images were with drivers built from source using driver-toolkit.
- Current Installer:
  - The system initiates from in-memory installation artifacts.
  - Uses a custom script that pulls ignition and runs **coreos-installer** to write image to disk.
- Future plans:
  - BFBs generated from generic aarch64 RHCOS artifacts and not our custom build.
  - Reduce custom script dependency by migrating the ignition pulling logic from the external script into RHCOS itself.
  - Moving to layered model where we pull layer with doca-ovs while in the middle of the installation
    - It will allow to have a single BFB image for all the Openshift installation and only build a layer z-stream version

