Turbocharge Kubernetes with offloaded OVN-Kubernetes and OVS on DPUs

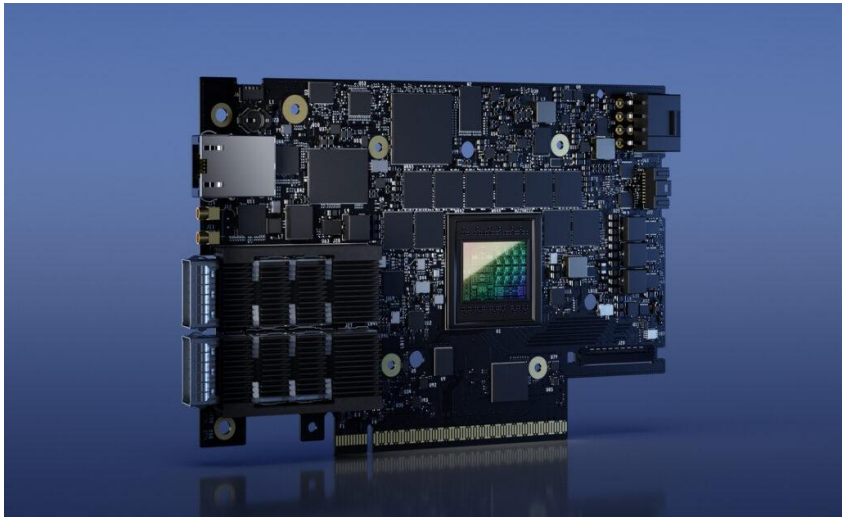
Alin Serdean, NVIDIA; Amit Zala, NVIDIA; Mael Kimmerlin, NVIDIA; Igal Tsoiref, RedHat , 20 Nov 2025

Agenda

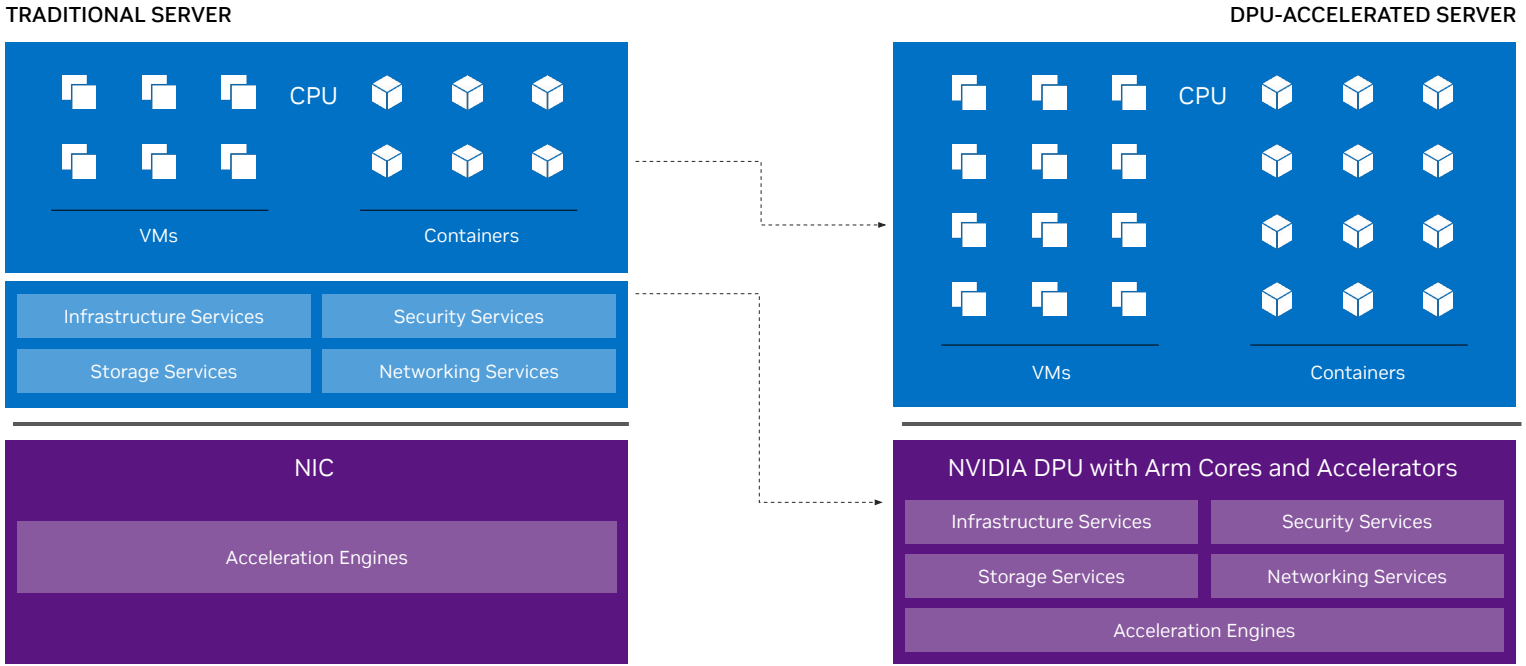
- What is a DPU?
- Why did we build DPF?
- SC (Service Chaining)
- Hardware acceleration using OVN-K8S CNI
- Redhat Integration

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



Pain points of managing DPU at scale - Life Without DPF

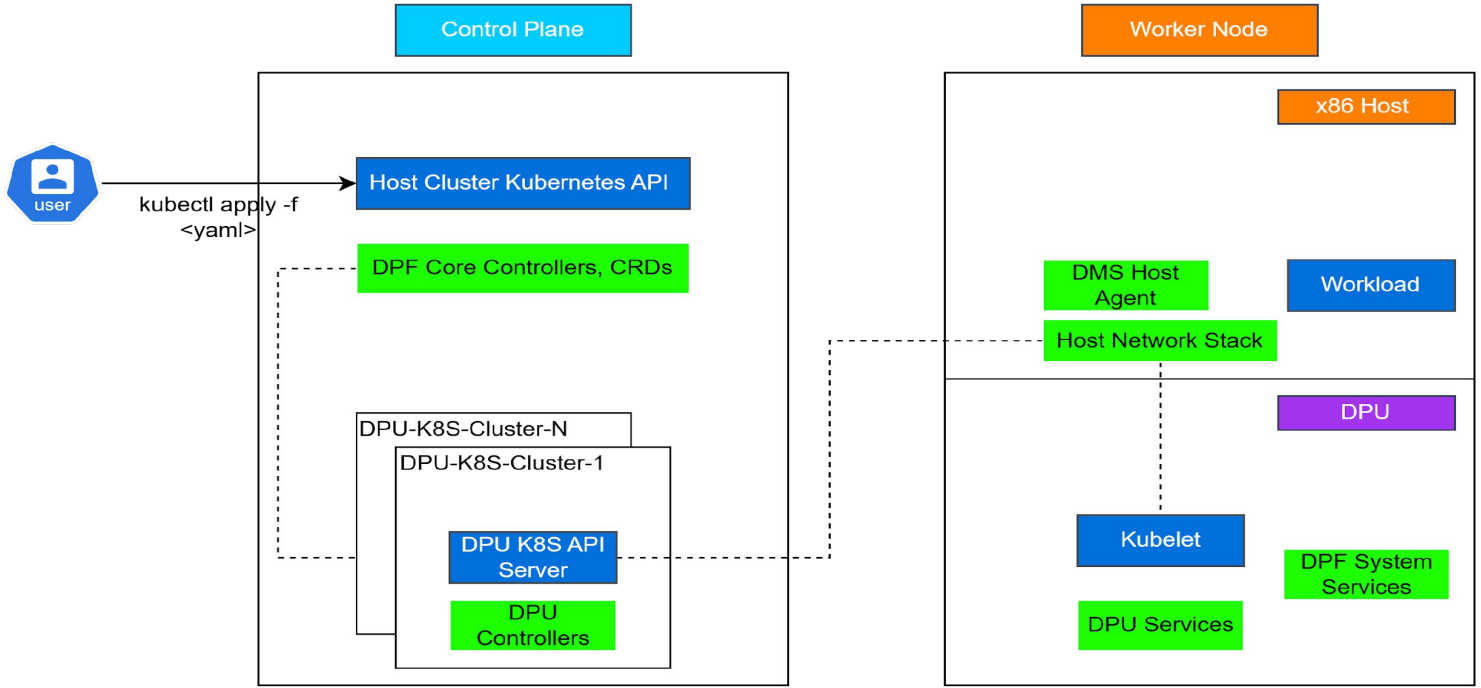
- **Manual Chaos at Scale**
  - Provisioning 100+ DPUs? SSH to each, run custom scripts, pray nothing breaks
  - One typo in BFB config → re-flash and start over (30+ min per DPU)
- **Service Deployment Nightmare**
  - Lifecycle management of services(system services + custom 3rd party services) running 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 3rd 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

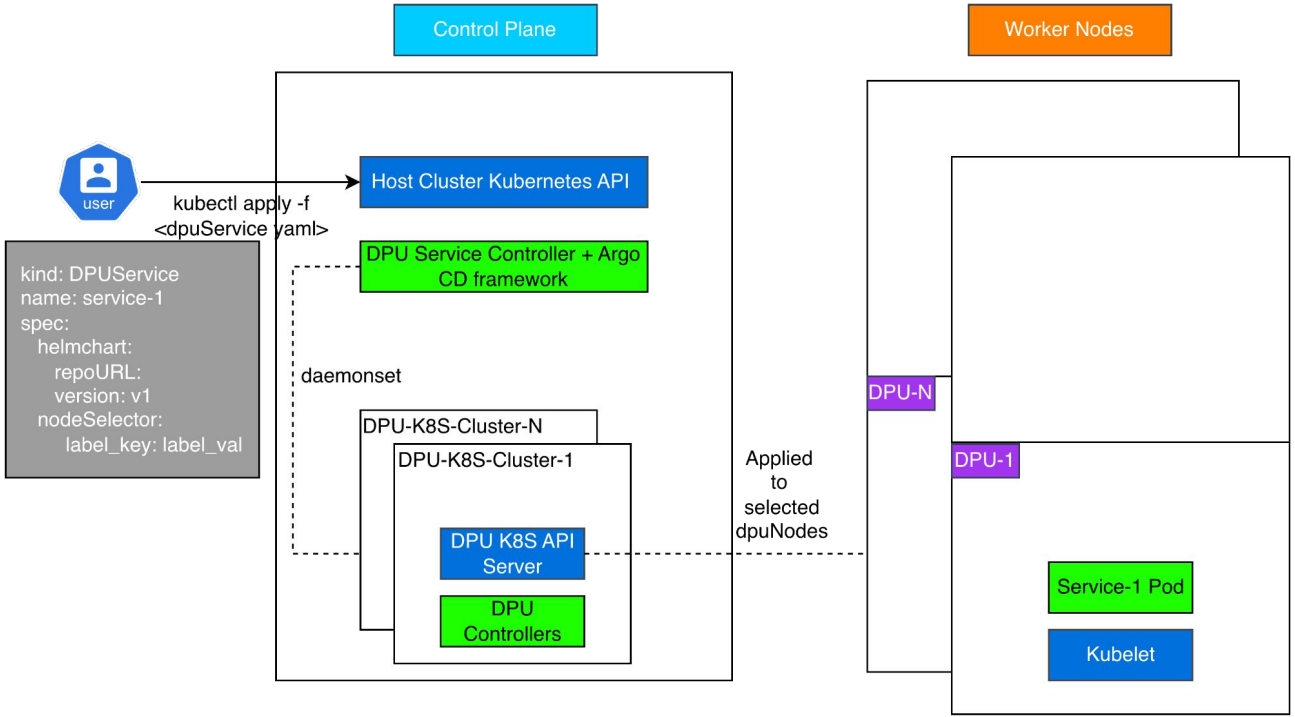
DPF – DOCA Platform Framework

- 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 3rd 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
- DPUServiceInteface - 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)

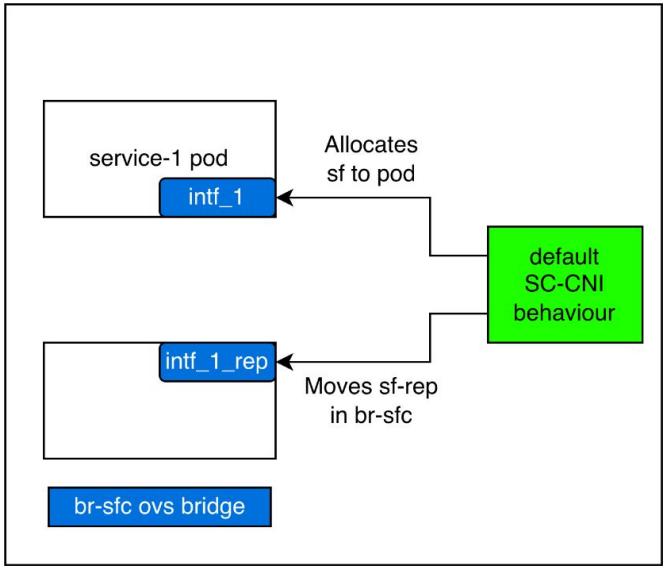
```
kind:DPUServiceInterface
spec:
    labels:
      interface: intf_1
    interfaceType: service
    service:
       serviceid: service-1
       ifname: intf_1
```

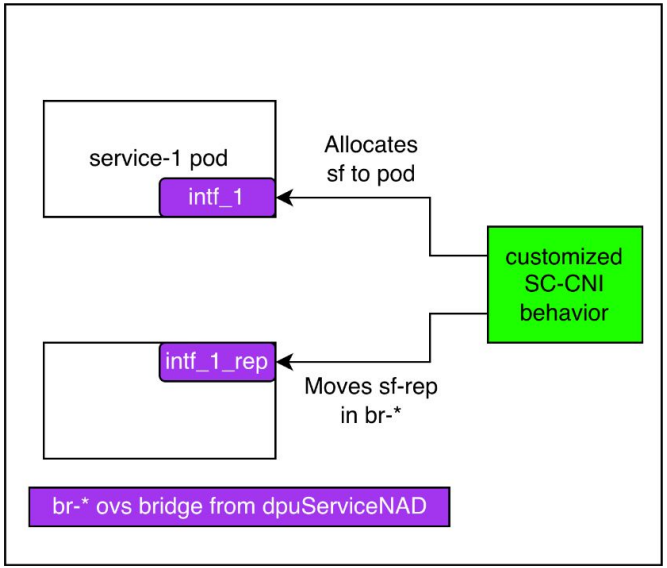
```
kind:DPUServiceChain
spec:
   switches:
   - ports:
     - serviceInterface:
           matchLabels:
               uplink: p1
     - serviceInterface:
           matchLabels:
               interface:intf_1
```

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

SC CNI

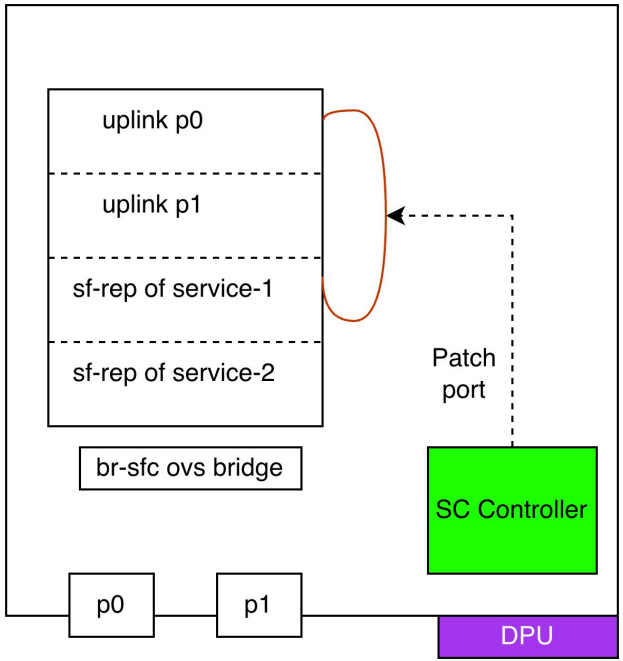
- When SC CNI allocates sf to the DPUService pod and adds the sf-rep into the OVS bridge (default br-sfc, but can change based on DPUServiceNAD configuration), it also adds metadata specific to the DPFService on the coresponding OVS port.



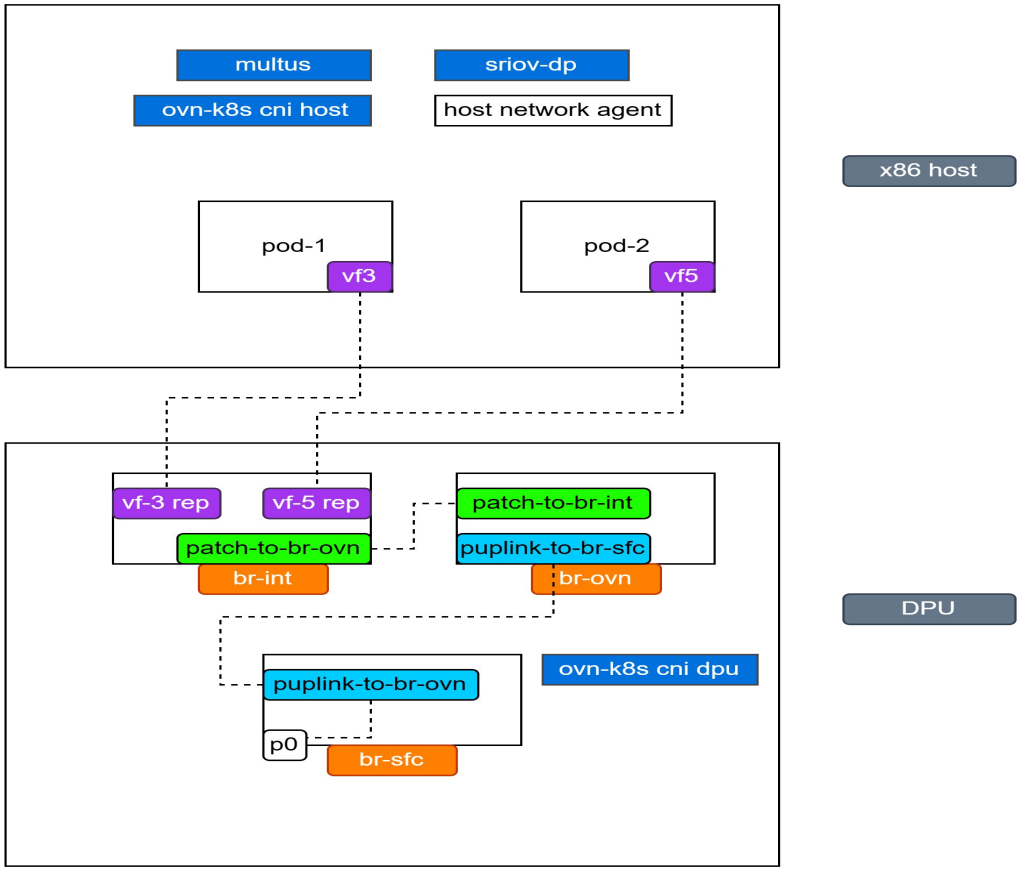


SC controller

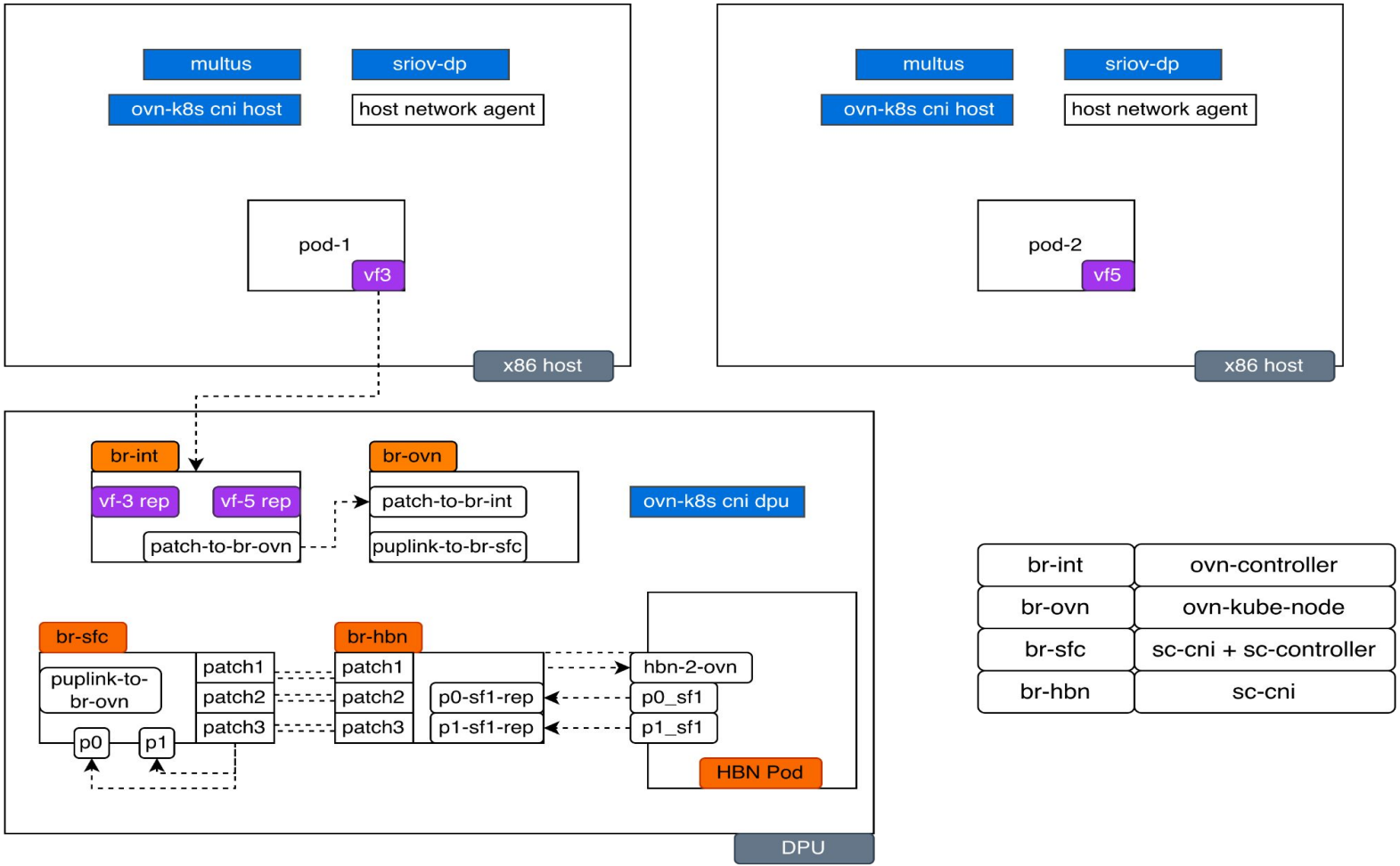
- When SC controller is looking at the chain abstraction ,it can filter through the metadata and can figure out it needs to create a patch paneling between ovs-port1 and ovs-port2.



Hardware acceleration of workload pods using OVN-K8S CNI

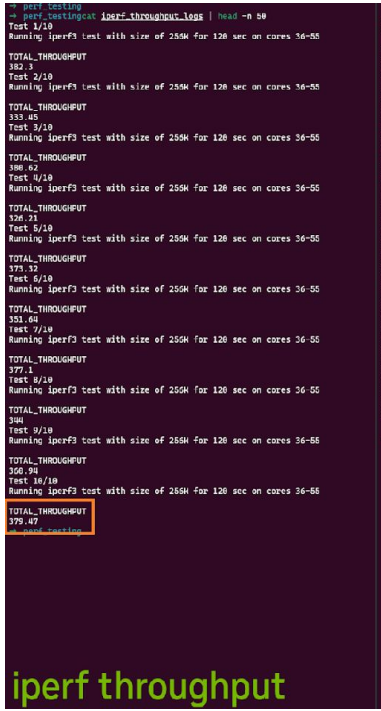


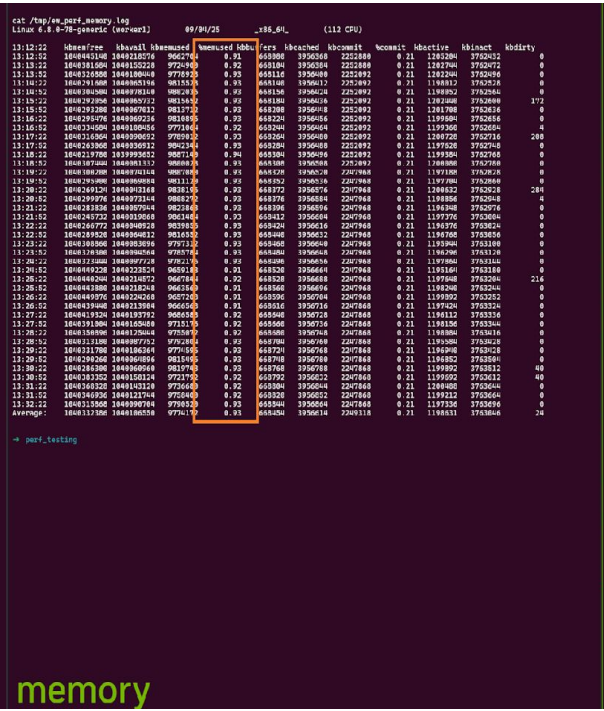
Hardware acceleration of workload pods using OVN-K8S +HBN

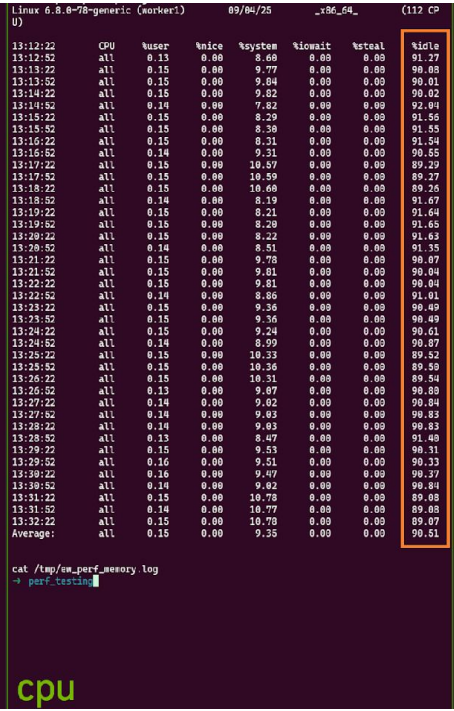


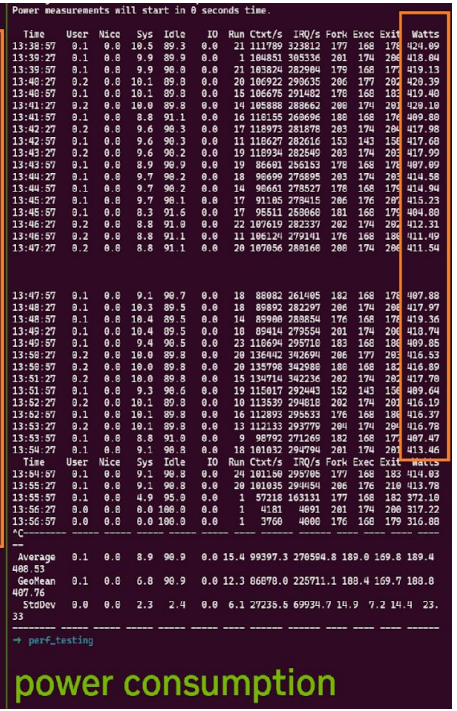
br-int	ovn-controller
br-ovn	ovn-kube-node
br-sfc	sc-cni + sc-controller
br-hbn	sc-cni

Perf numbers





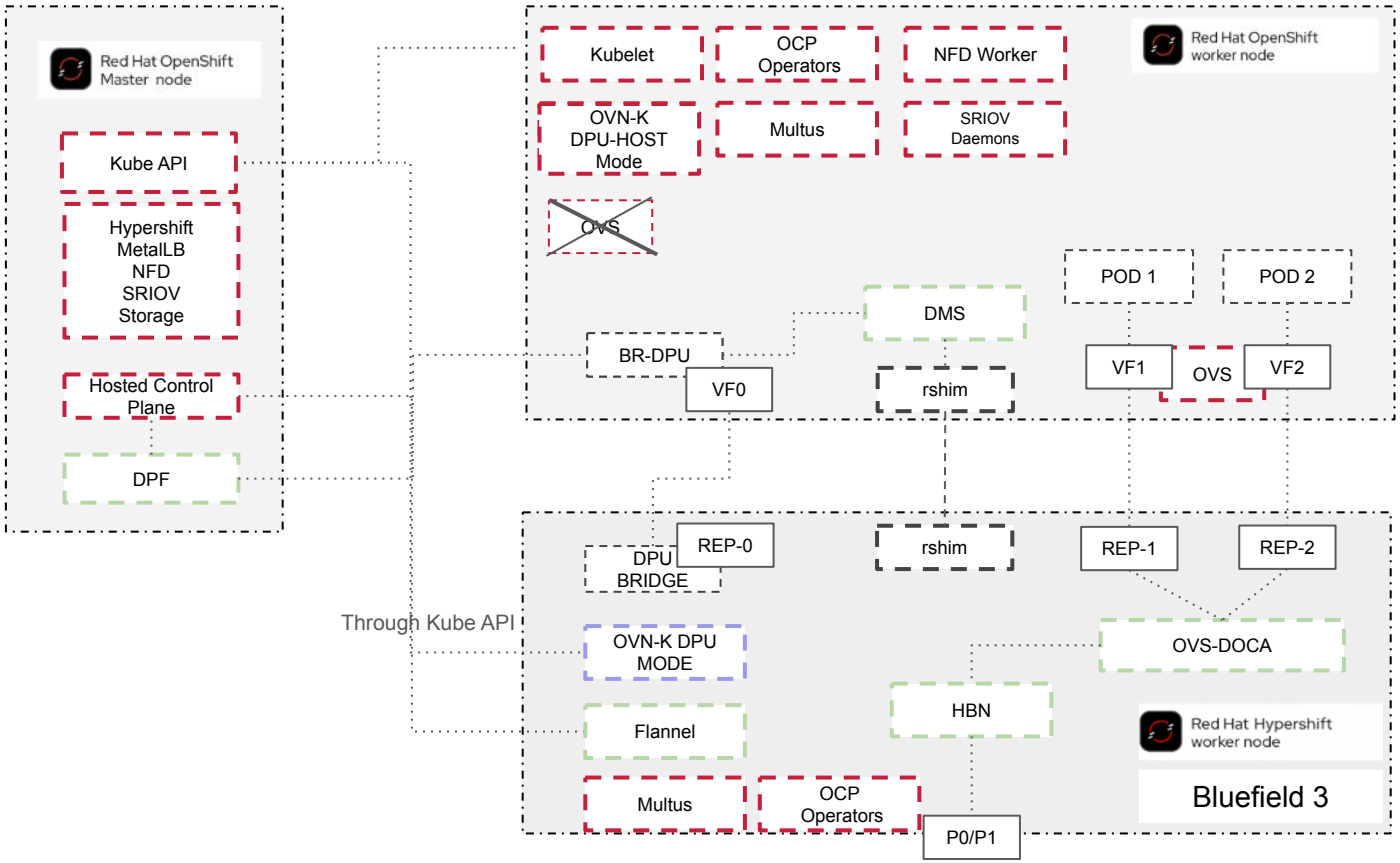




Integration Challenges

- 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.

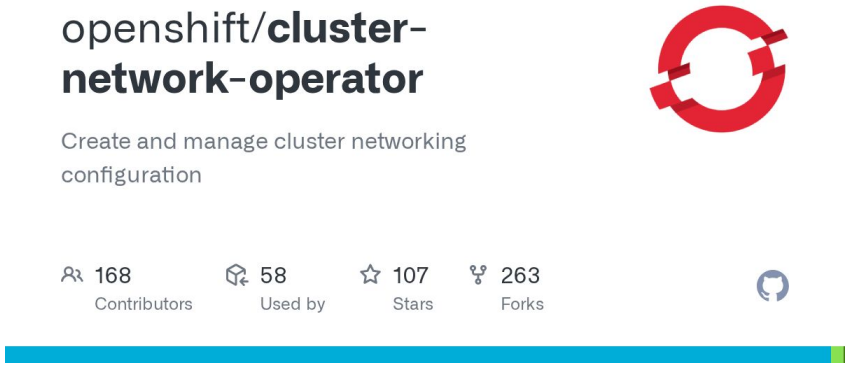
OpenShift Networking with NVIDIA BlueField-3 and DPF



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.

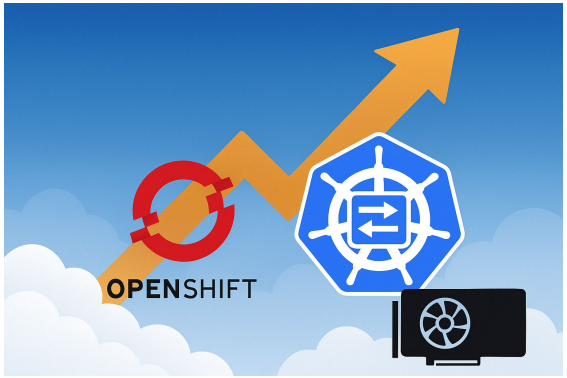


OVN-Kubernetes: Shifting to Upstream first mentality

- 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

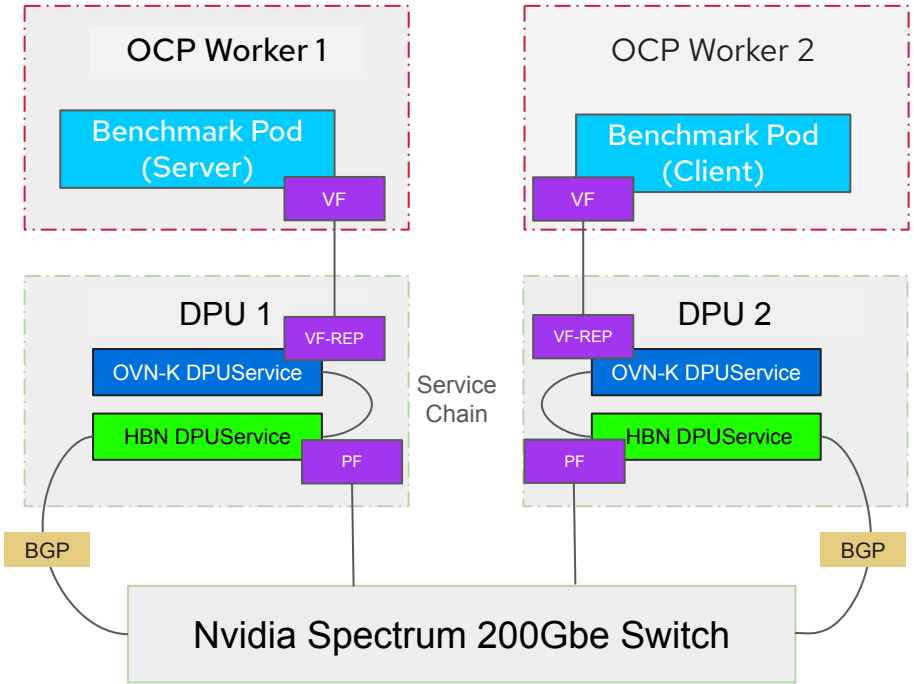
- Due to the resource constraints of the DPU, some functionalities from HyperShift had to be disabled:
  - Insights, Console, Openshift Samples, Ingress, NodeTuning
  - Reduced ~10GB of memory
- Instead of using the default primary CNI of HyperShift (ovn-kubernetes), this deployment uses Flannel. The 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

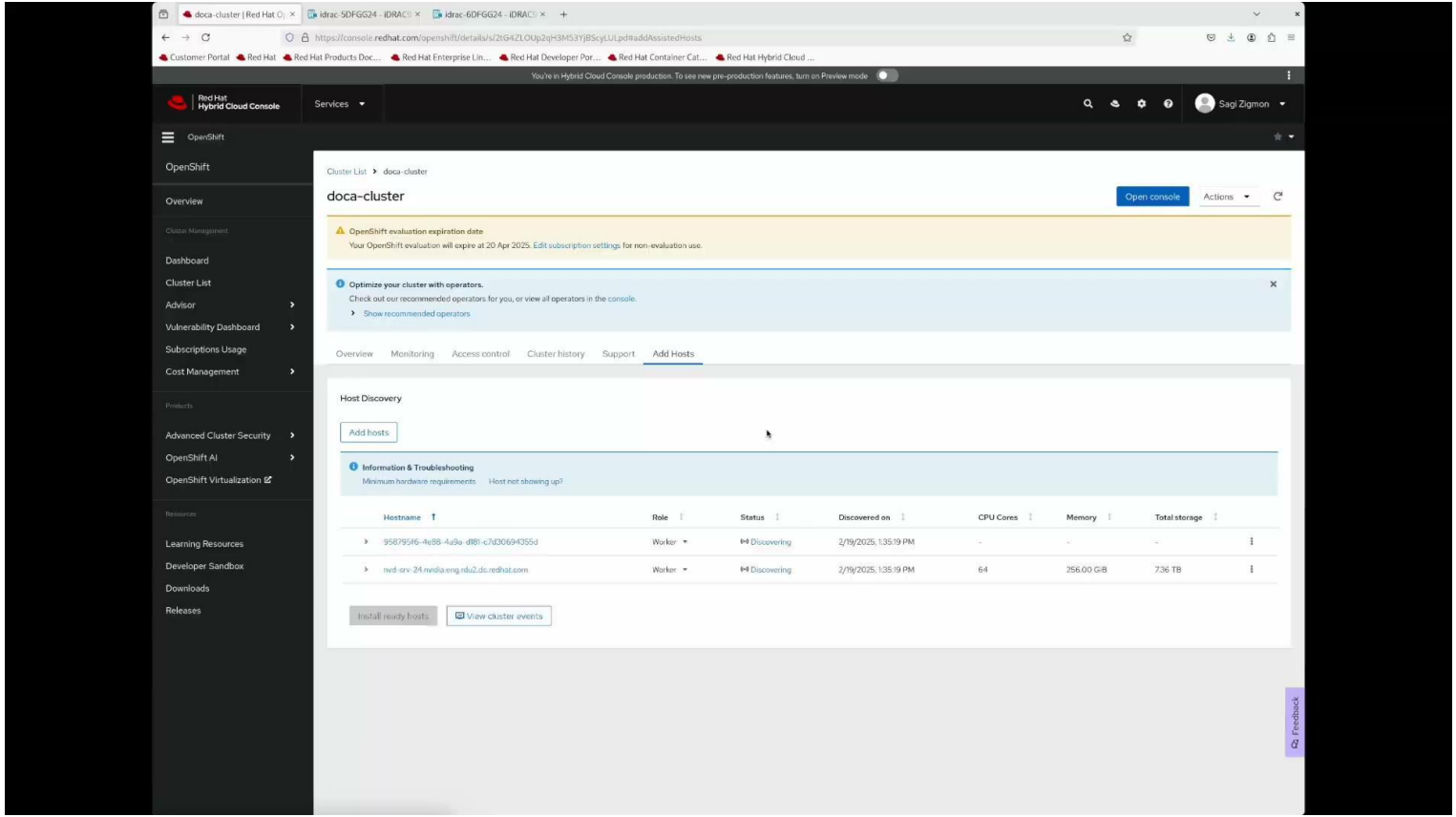
iPerf Test (iperf3 client)

```
$ ./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: 69
Results for
  Bandwidt
Results for
  Bandwidth: 8.
Results for port 53
  Bandwidth: 9.245 Gbit/sec
Results for port 5325:
  Bandwidth: 10.461 Gbit/sec
Total Bandwidth across all streams: 354.924 Gbit/sec
### Iperf Test (iperf3 Client) Complete ###
```

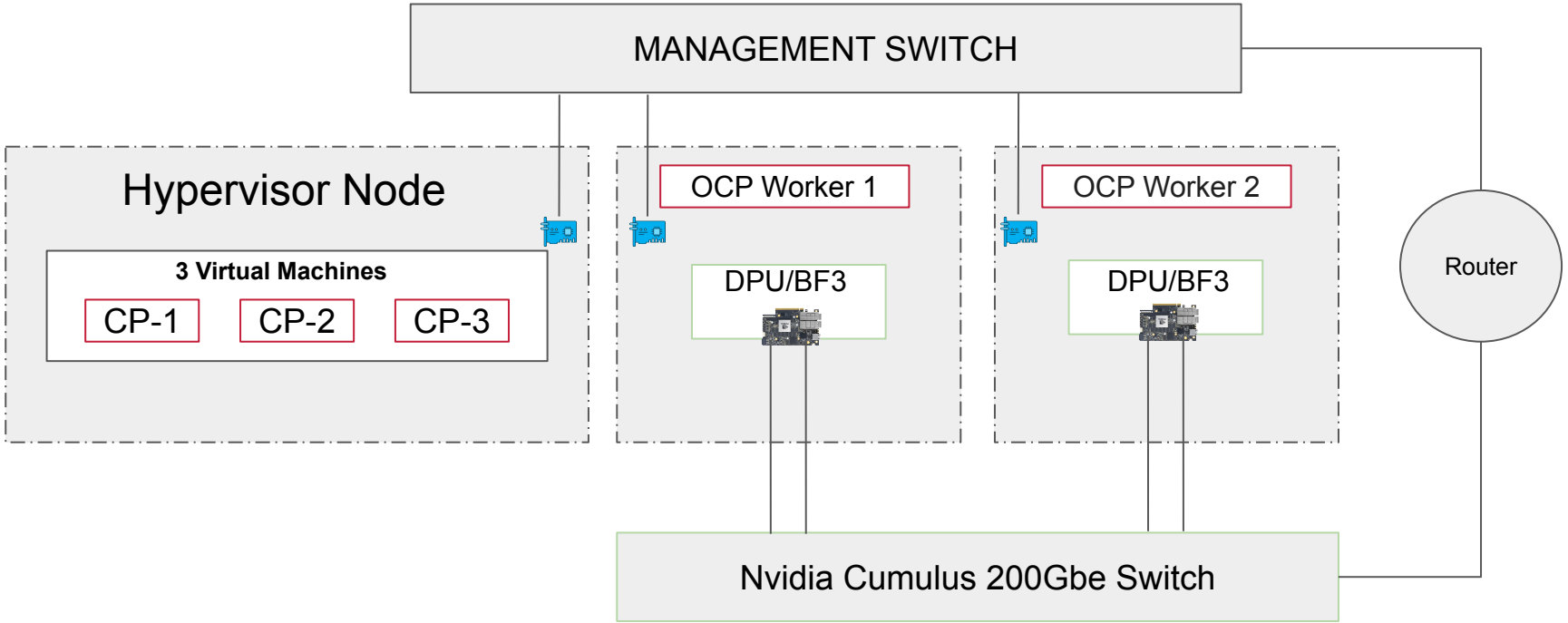
354.92 Gbit/Sec



Deployment and Benchmark Demo



Lab physical connectivity



Red Hat

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 bootfifo 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 Opensfhit installation and only build a layer z-stream version

Red Hat CoreOS