KubeCon | CloudNativeCon

North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Kubernetes Networking Infrastructure Offload

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Nabil Bitar
Bloomberg

Dan Daly
Intel

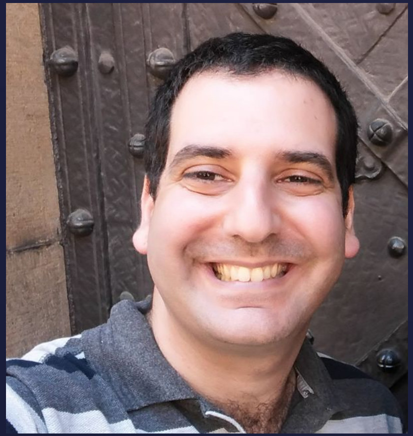
Nupur Jain
Intel

KubeCon | CloudNativeCon
North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

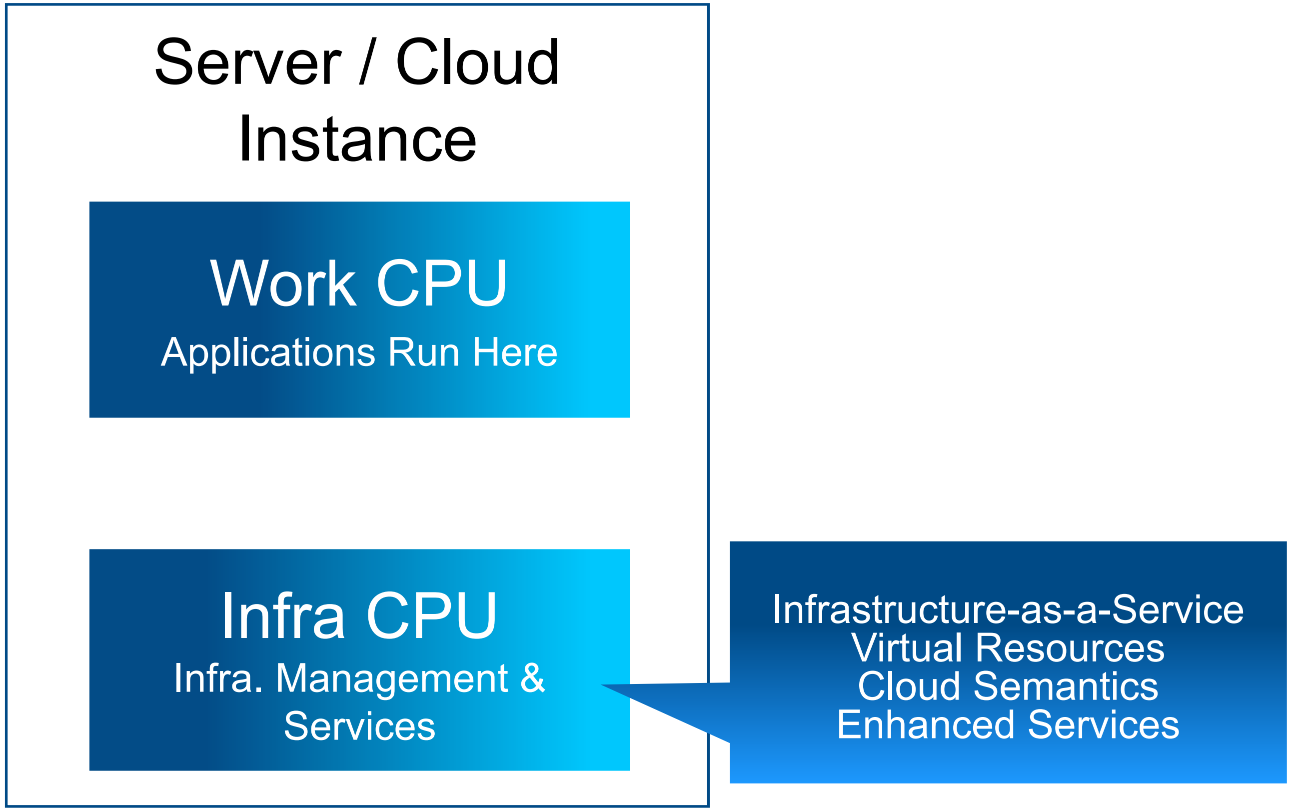
October 24-28, 2021



Moshe Levi
NVIDIA

Valas Valancius
Google

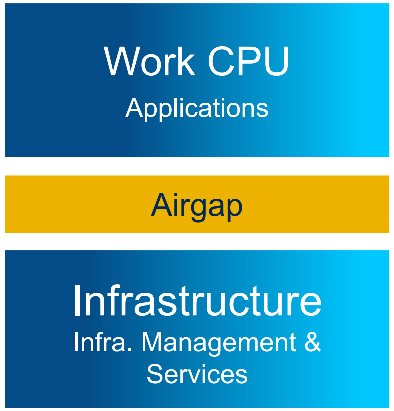
What is Infrastructure?



Why Separate?

Security

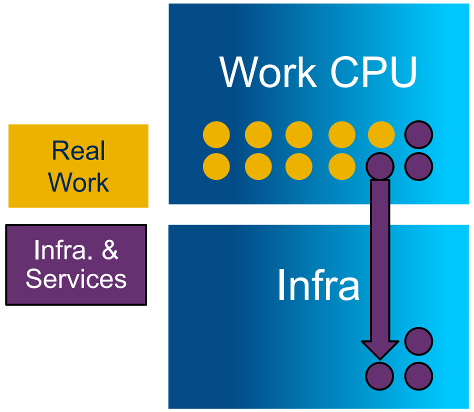
Increased Infrastructure Security



Application & Tenant Isolation from Infrastructure

Infrastructure Task Migration

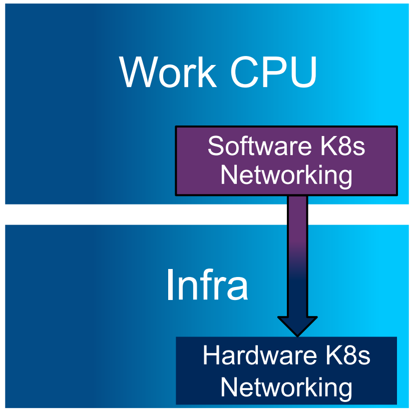
Lower Server Overhead



Move Infra Services Out of the Server / Cloud Instance

Accelerate for Performance

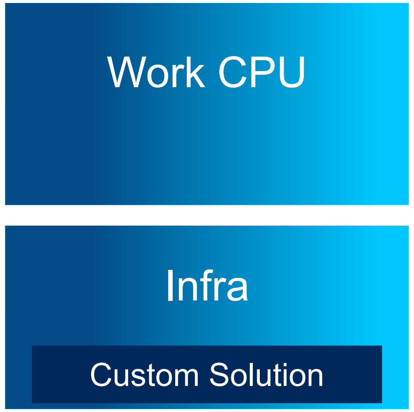
Hardware Accelerated Infrastructure



Software and Hardware Optimized for Infrastructure Tasks

Feature Velocity

Customizations at the Speed of Software



Evolve Infrastructure Independently

IPU/DPU Offload Goals and Requirements

Nabil Bitar, Bloomberg

What is Offload - Networking Speak

- Mainly targets data packet processing offload from software on the host CPU to network interface card (NIC) hardware - traditionally known as smart NICs/performance NICs

- Infrastructure/Data Processing Units (IPU/DPUs) evolved from Smart NICs
  - Include compute (CPU and memory) in addition to ASIC/FPGA (for fast packet processing),
  - Improve scale, performance and support additional capabilities (e.g., control functions, storage networking, Load balancing, virtualization).
  - Examples:
    - Intel IPU
    - Nvidia DPU
    - AMD Pensando DPU

Objectives

- Improved performance (latency and throughput) per host
- Improved security
- Improved overall efficiency in terms of power and compute density for application processing
    - Free up host CPU for application processing (what CPUs are good at)
    - Leverage IPU/DPU for networking

Goals/Requirements

- All data plane functions offloaded to IPU/DPU, isolating and alleviating networking state information and processing from host CPUs. Examples:
  - FIB and Forwarding
  - Policies and enforcement
  - Connection tracking
  - Tunneling
  - Traffic policing and shaping
  - Load balancing - L3/L4 and L7 as applicable
  - TLS Encryption/decryption
  - NVMe/TCP
  - Statistics and flow logging for allowed and denied flows
- Industry standard hardware abstraction layer that enables easy integration between networking solutions from different sources (projects/vendors) and hardware from different vendors

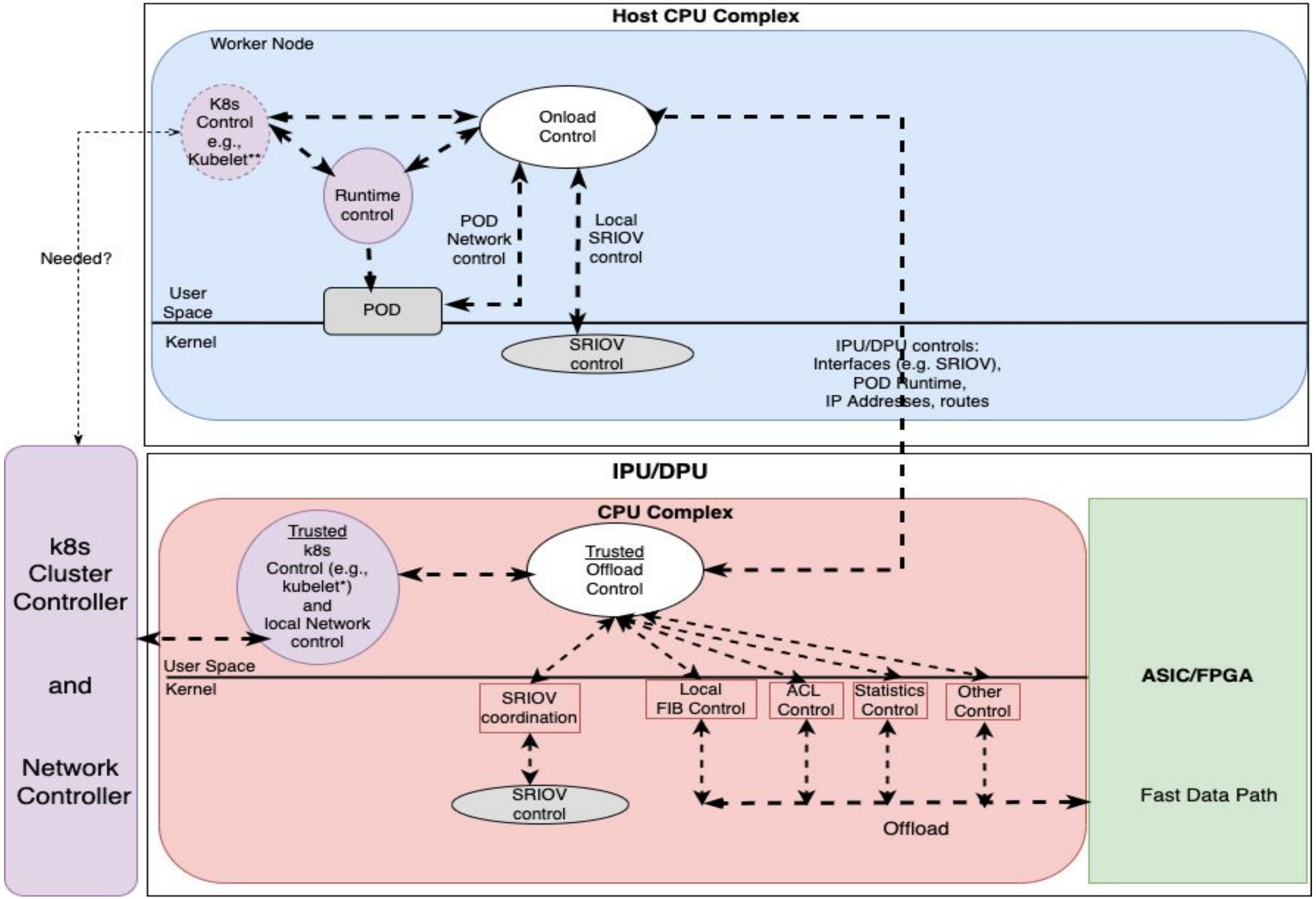
Goals/Requirements

- Compute and networking control to be optimally designed for IPU/DPU to provide for security/trust and performance
  - Control path between compute/network controllers and compute nodes
  - Offload of control plane functions to IPU/DPU and coordination with control functions on host CPU

- Support data plane and control plane for various compute endpoints
  - Bare metal
  - Virtual machines
  - Containers on bare metal hosts and on VMs as worker nodes

Potential High Level Target Architecture

Functional distribution between host processor and IPU/DPU



Existing Public Cloud Models

Valas Valancius, Google

Public Cloud VPC & K8s Networking

- Public clouds **offload VPC networking** to IPU/DPU.
- The offloaded VPC networking features usually consists of:
  - Routing, policy-routing, internal/external loadbalancing, security policies, observability, etc.
- K8s networking encompases a similar feature set:
  - Pod reachability, ClusterIP/NodePort/ExternalIP, Network Policies, observability, etc.

Can we offload K8s networking to IPU/DPU as well?

K8s Offload in Public Cloud: Current

- GKE already offloads some limited networking functions to IPU/DPU
- Pod-to-Pod routing:
  - Removes tunneling overheads.
  - Each node gets /24 range for pod addresses.
  - VPC ensures reachability at scale (15K nodes * 255 pods).
- Intra node visibility:
  - GKE can route intra-node traffic via hypervisor.
  - Enables GCP Flow Logging, GCP Firewall, GCP Packet Mirroring features for intra-node traffic.

K8s Offload in Pubic Cloud: Opportunity

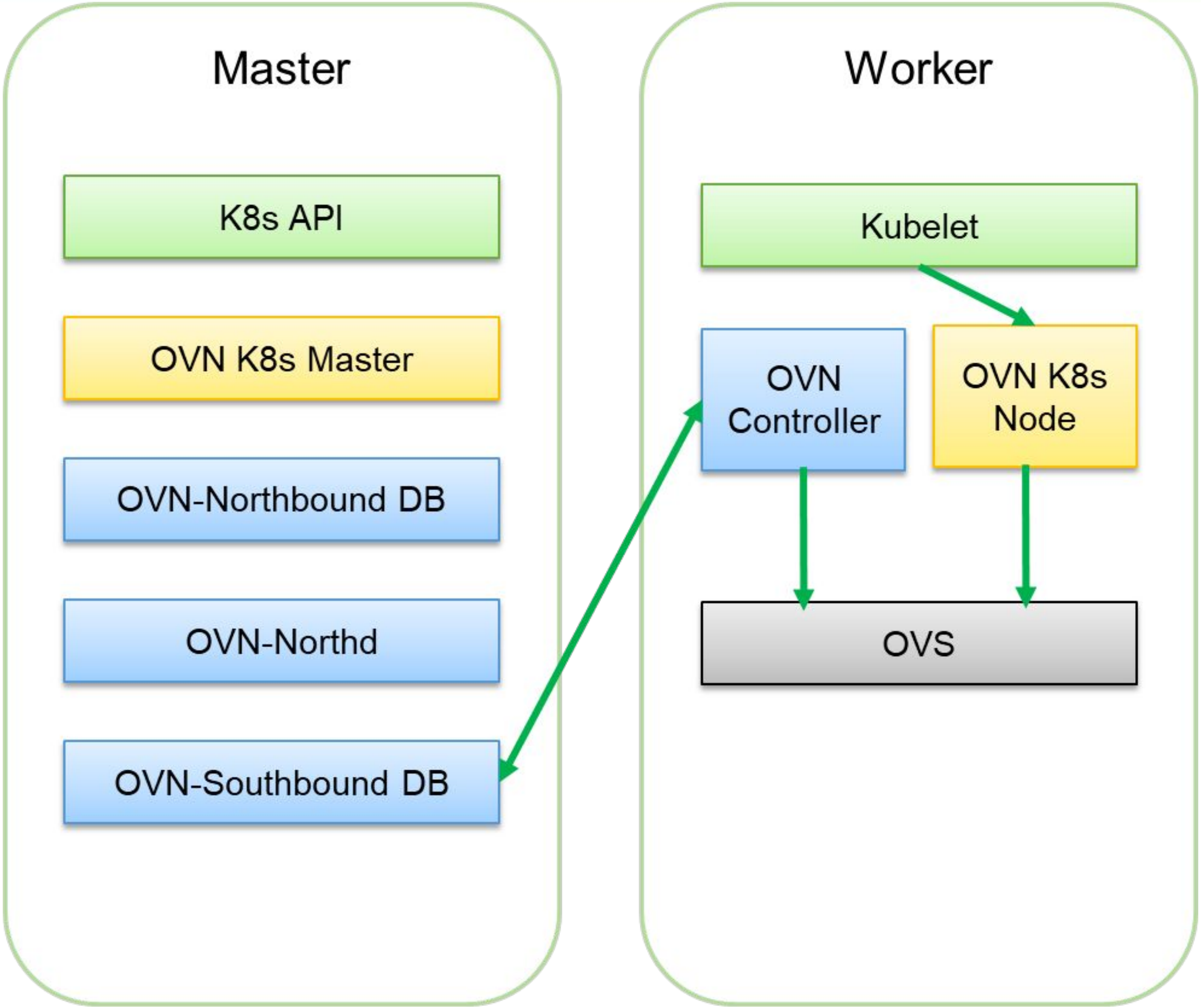
- GKE still relies heavily on **onloaded** implementation for most of K8s networking.
    - Reasons: time-to-market, familiarity, resource accounting (i.e. billed to the user).
- Strong tail-winds to offload more:
    - Maturing IPU/DPU offload infrastructure.
    - Transparent (to user) evolution of K8s networking stack - increased feature velocity.
    - Single datapath implementation for Linux, Windows, BSD.
    - Opportunity to offer K8s features to kernel bypass (DPDK) users and to sandboxes (GVisor).
    - Significant efficiency gains (next in the talk).

K8s Networking Infrastructure Offload

Moshe Levi, NVIDIA

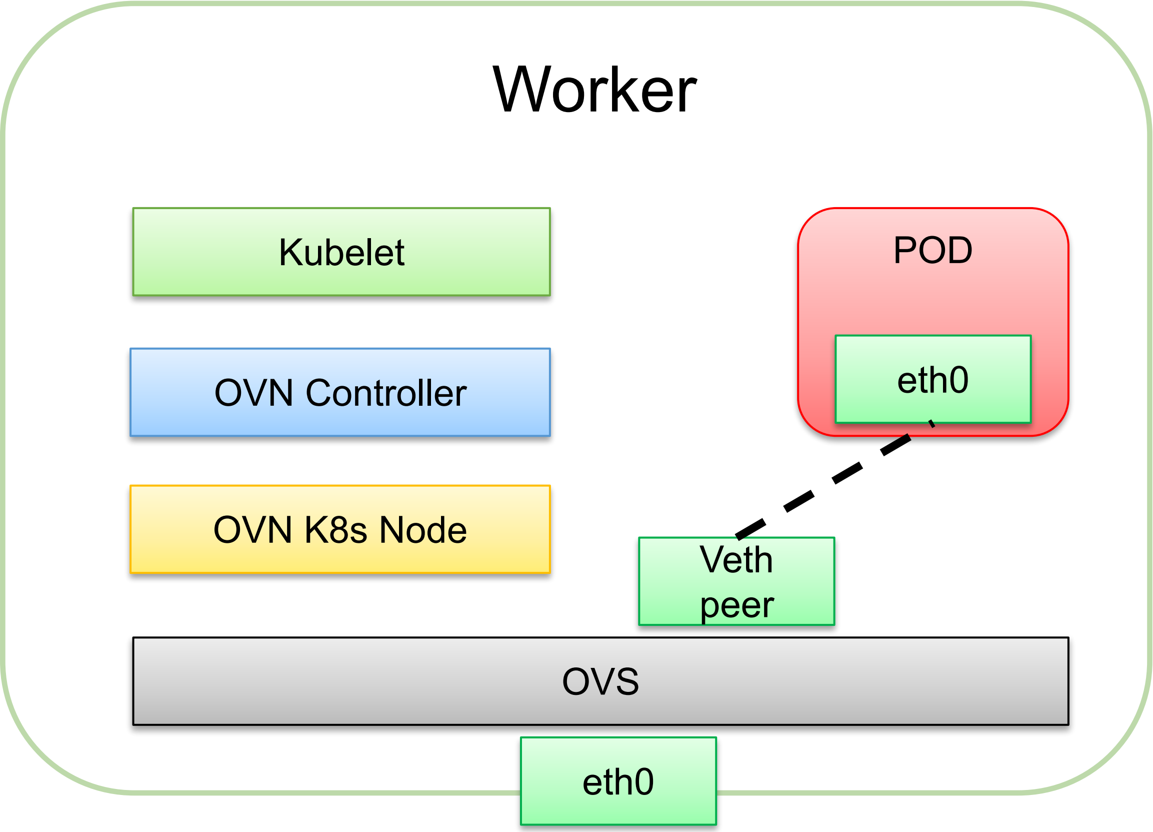
OVN Kubernetes CNI – Overview

- Based on OVS & OVN Community
- Uses OVN logical components e.g., logical switches, logical routers…
- Doesn't use Kube-Proxy



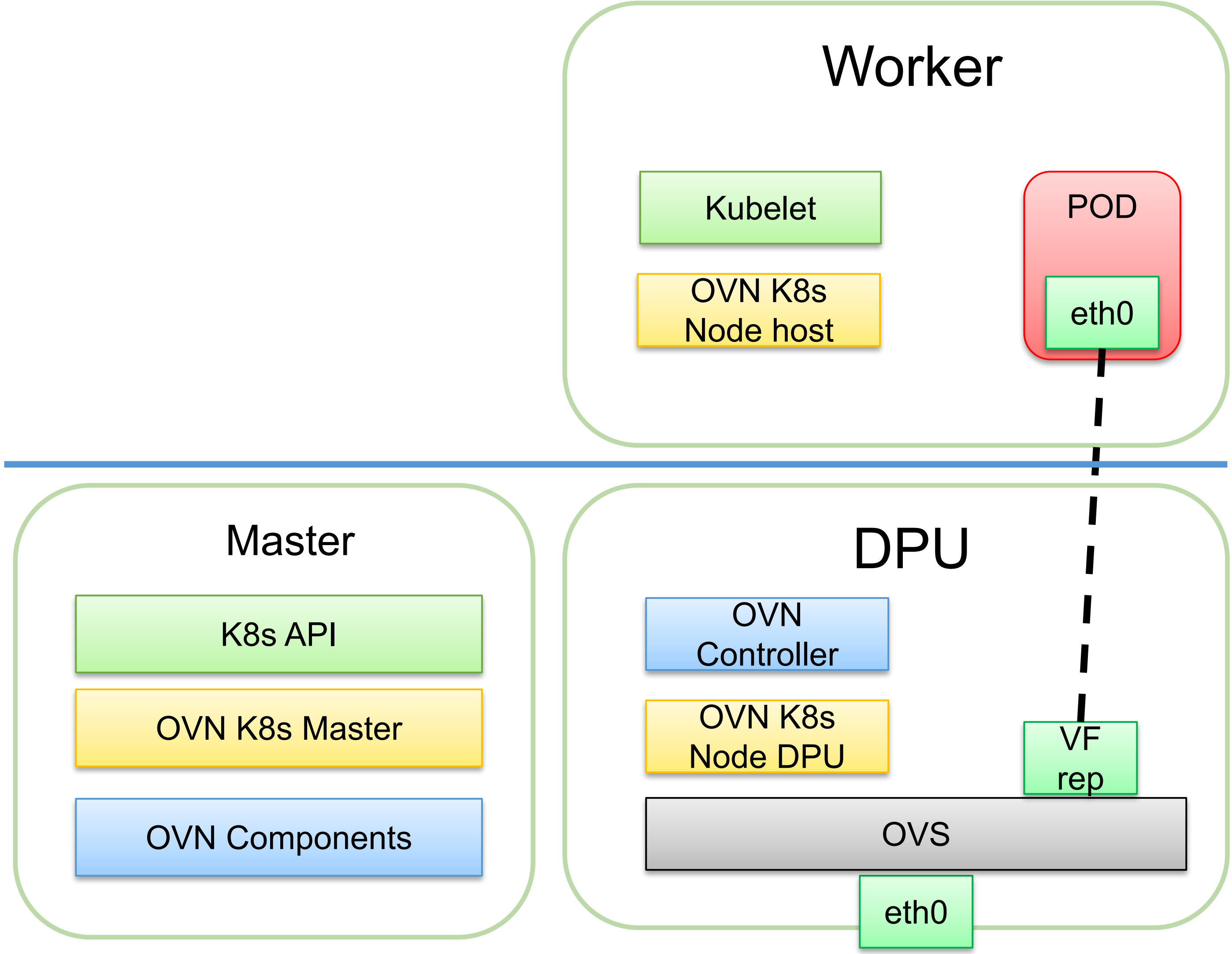
OVN K8s Worker with NIC

- Veth pair networking
- OVS/OVN run on worker node
- High CPU utilization
- Limited to software performance



OVN K8s Worker with DPU

- SR-IOV switchdev networking
- OVS/OVN run on DPU
- Low CPU Utilization on worker
- Full Isolation

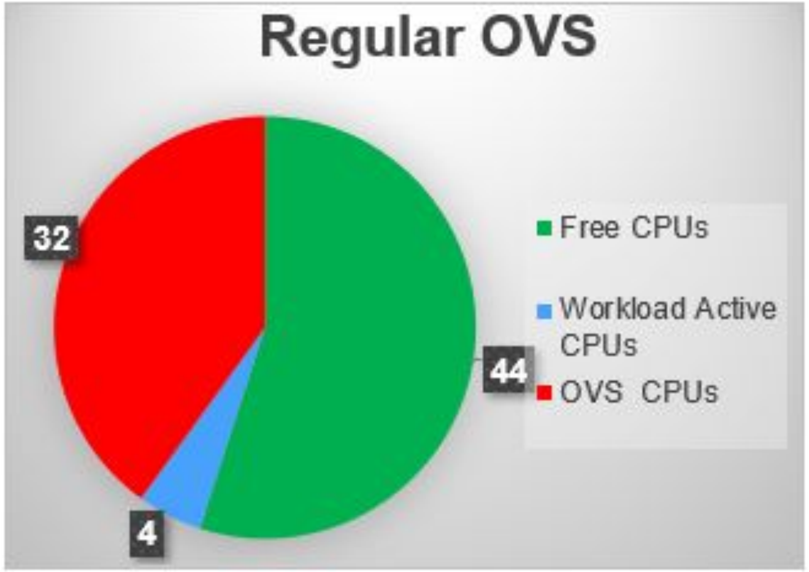


DPU Hardware Acceleration

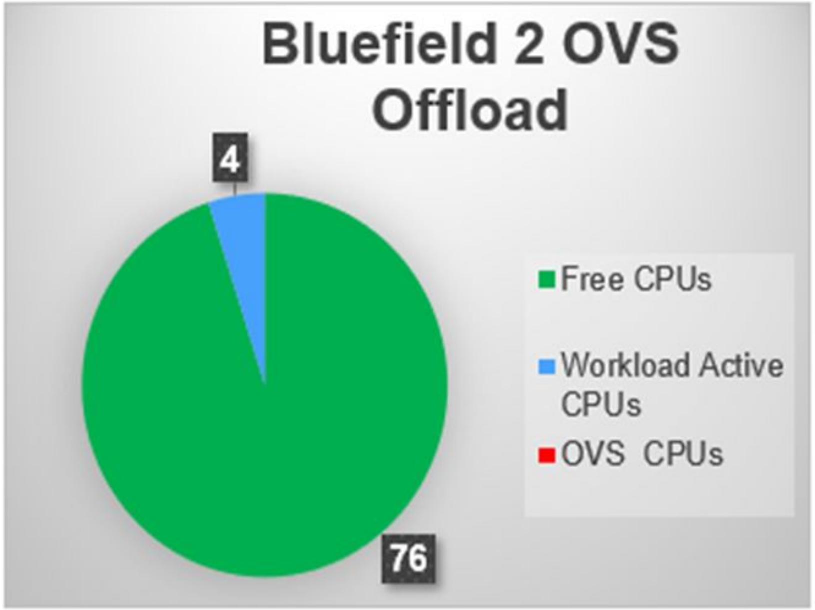
- OVS acceleration leverage SR-IOV switchdev
- OVS programs DPU E-switch datapath on first packet arrival
- Standard kernel API, TC flower, to program the E-switch
- Fallback to DPU kernel datapath on unsupported flows
- Reduce ARM Cores utilization on DPU
- Low latency and line rate performance

DPU Performance Results

- IXIA Packet Generator
- PowerEdge R740 -2 x Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz Total: 80 Logical CPUs
- OVS with ConnectX-6 Lx 2x25GbE: veth
- OVS acceleration with BlueField-2 2x25GbE: SR-IOV
- Workload: testpmd pinned to 4 Logical CPU
- **Datapath: Geneve + Connection Tracking 500K Connection**

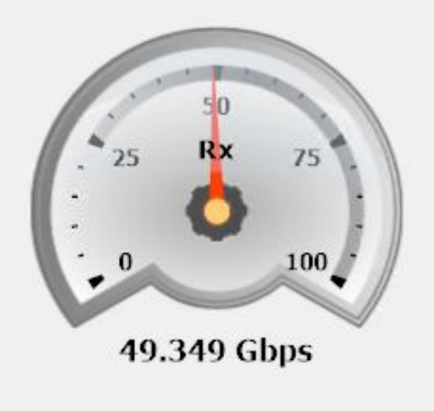


Reduce Logical CPU Utilization



DPU Performance Results

Higher Throughput



Latency

Min	16,335 ns
Avg	132,825 ns

Lower Latency

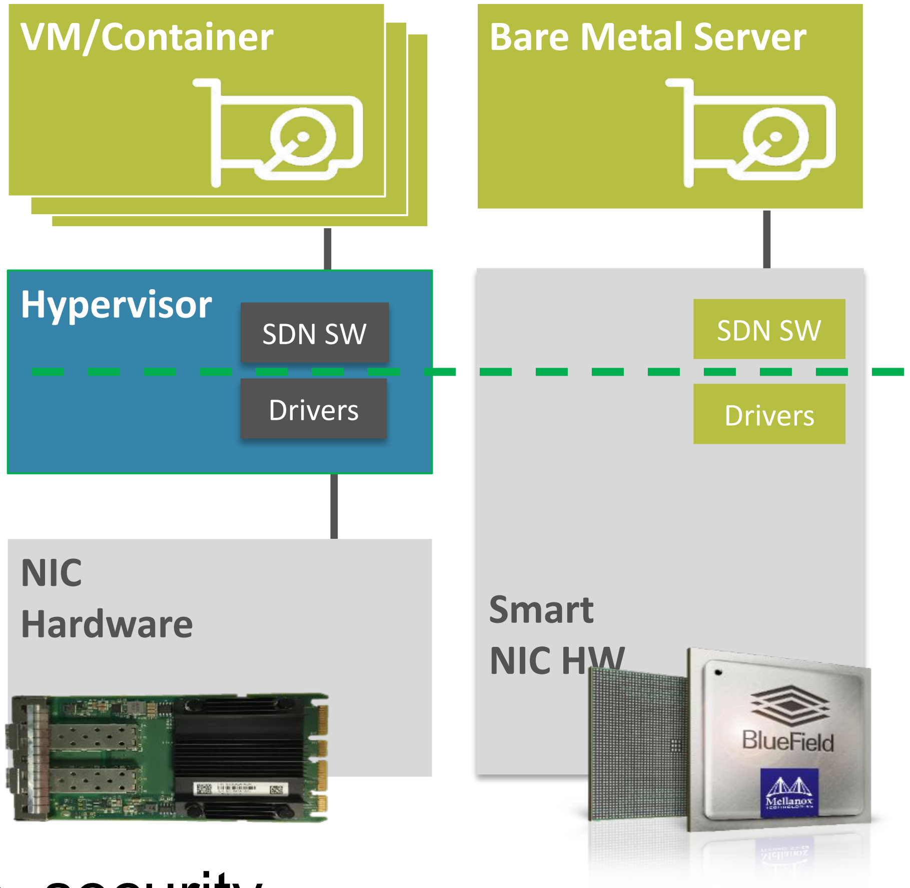
x3 Better Min Latency
x4 Better Avg Latency

Latency

Min	5,480 ns
Avg	31,505 ns

DPU Benefits

- Uncompromised performance
- Reduce CPU utilization to minimum
- Full network isolation
- Single network solution for Pod, VM and BM
- DPU can run additional services e.g., storage, security…



KubeCon | CloudNativeCon
North America 2022

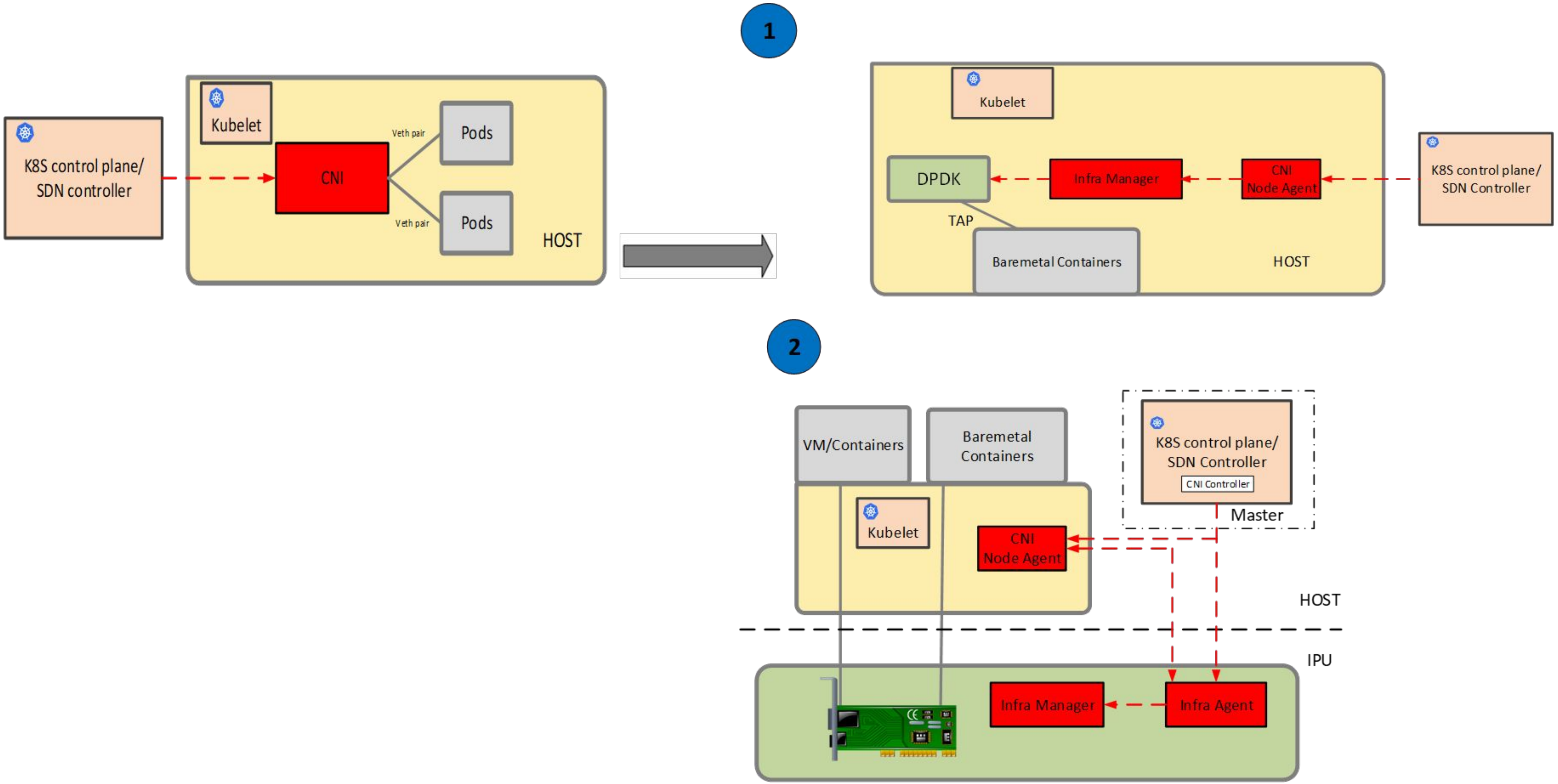
BUILDING FOR THE ROAD AHEAD

DETROIT 2022

K8s Networking Infrastructure Offload

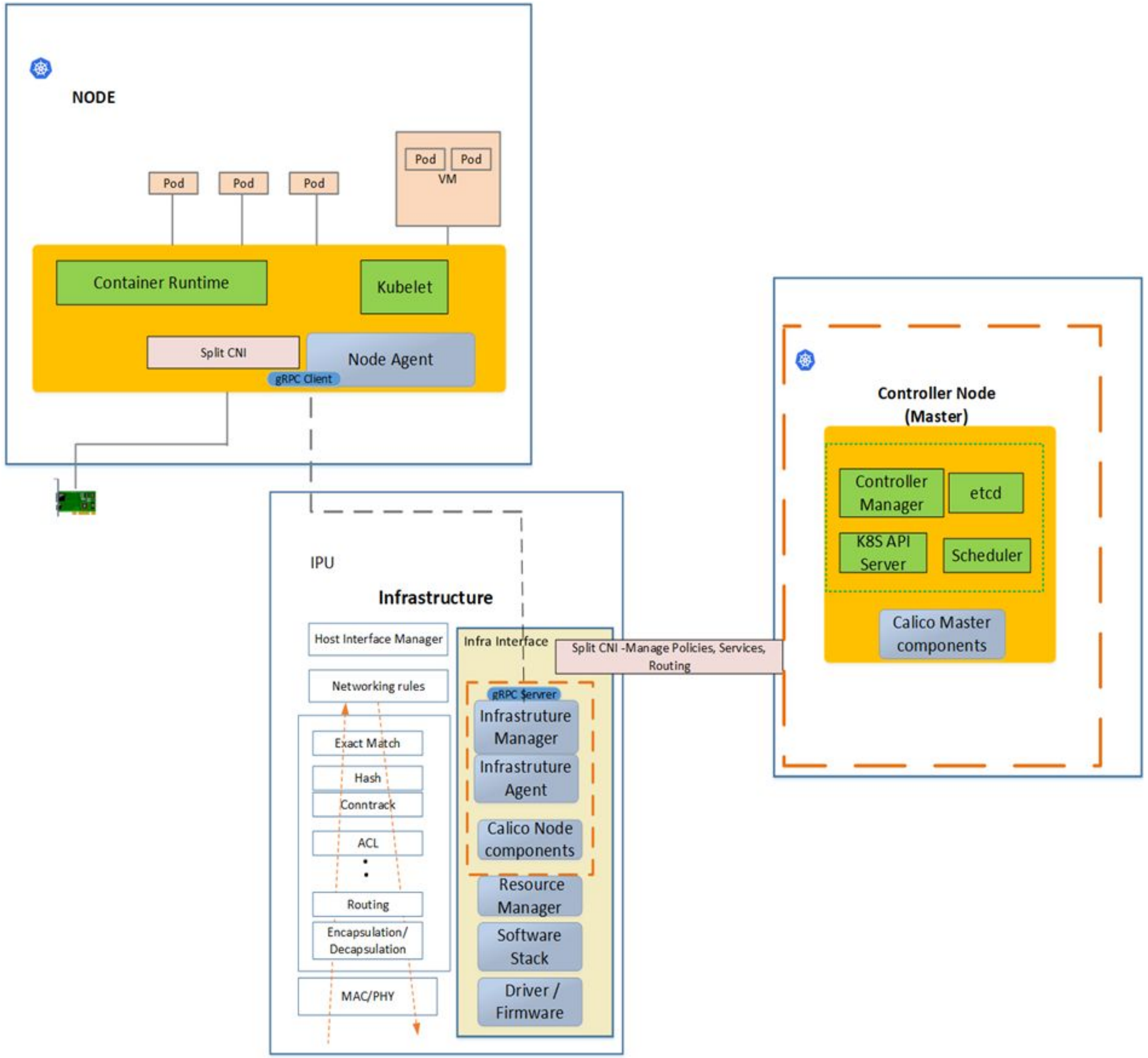
Nupur Jain, Intel IPDK Team

Offload Approach



Infrastructure Offload

- No Modifications to existing CNI
- Introduction of external gRPC Dataplane
- **Split Implementation**
  - Infrastructure Agent for API handling/watch
  - Node agent for interface addition
  - Infrastructure Manager for secure datapath provisioning
  - gRPC with mTLS enabled for message protection



Implementation Example – P4 DPDK

[root@localhost p4-k8s]# kubectl get pods -A

```
NAMESPACE     NAME                                       READY   STATUS
RESTARTS   AGE
default       test-pod                         1/1     Running           0
35s
default       test-pod1                         1/1     Running           0
25s
kube-system   calico-kube-controllers-58dbc876ff-zhrqv   0/1
Running   0          43s
kube-system   calico-node-gwg99                          1/1     Running   0
43s
kube-system   coredns-565d847f94-jw5mp                   0/1     Running
0          18m
kube-system   coredns-565d847f94-s4gdn                   0/1     Running
0          18m
kube-system   etcd-mev03                                 1/1     Running   0
19m
kube-system   infraagent-ds-w7zq6                        1/1     Running   0
61s
kube-system   inframanager-ds-v5xcn                      1/1     Running   0
61s
kube-system   kube-apiserver-mev03                       1/1     Running   0
19m
kube-system   kube-controller-manager-mev03              1/1     Running
0          19m
kube-system   kube-proxy-8tflq                           1/1     Running   0
18m
kube-system   kube-scheduler-mev03                       1/1     Running   0
19m
```

P4-OVS & DPDK

#ps -ef | grep ovs

```
root       56228       1  0 19:24 ?        00:00:00 ovsdb-server
--remote=punix:/usr/local/var/run/openvswitch/db.sock
--remote=db:Open_vSwitch,Open_vSwitch,manager_options
--pidfile --detach
root       56231       1 99 19:24 ?        00:48:17 ovs-vswitchd --pidfile
--detach --no-chdir unix:/usr/local/var/run/openvswitch/db.sock
--mlockall --log-file=/tmp/ovs-vswitchd.log

Thread 0x7f787b1f1640 (LWP 56237) "eal-intr-thread"
0x00007f788266080e in epoll_wait () from /lib64/libc.so.6
  4    Thread 0x7f787a9f0640 (LWP 56238) "rte_mp_handle"
0x00007f78828c5dfd in recvmsg () from /lib64/libpthread.so.0
  5    Thread 0x7f787a1ef640 (LWP 56239) "lcore-worker-1"
0x00007f7881c6fb26 in rte_pktmbuf_free () from
/lib64/librte_port.so.22
```

P4 Example – Service (CT, NAT)

```
action pinned_flows_hit (FlowId_t flow_id, PortId_t p,
                         ModDataPtr_t ptr) {
    // This action should only be executed for Tx packets.
    meta.dst_port = p;
    send_to_port(p);
    meta.mod_action = WRITE_DEST_IP;
    meta.mod_blob_ptr = ptr;
}

// Note: This action does nothing at all if
// do_clb_pinned_flows_add_on_miss is false.
action pinned_flows_miss() {
    if (do_clb_pinned_flows_add_on_miss) {
        //my_flow_id = allocate_flow_id();//DPDK doesn't yet support allocate_flow_id()
        my_flow_id = (FlowId_t)0;
        add_succeeded =
            add_entry(action_name = "pinned_flows_hit", // action name
                      action_params = (clb_pinned_flows_hit_params_t)
                                      {flow_id = my_flow_id,
                                       p = meta.dst_port,
                                       ptr = meta.mod_blob_ptr});
    }
}

table pinned_flows {
    key = {
        // other key fields also possible, e.g. VRF
        SelectByDirection(istd.direction, hdr.ipv4.srcAddr,
                                          hdr.ipv4.dstAddr):
            exact @name("ipv4_addr_0");
        SelectByDirection(istd.direction, hdr.ipv4.dstAddr,
                                          hdr.ipv4.srcAddr):
            exact @name("ipv4_addr_1");
```

Where to Learn More

https://github.com/ipdk-io/k8s-infra-offload

Summary

Infrastructure Offload requires:

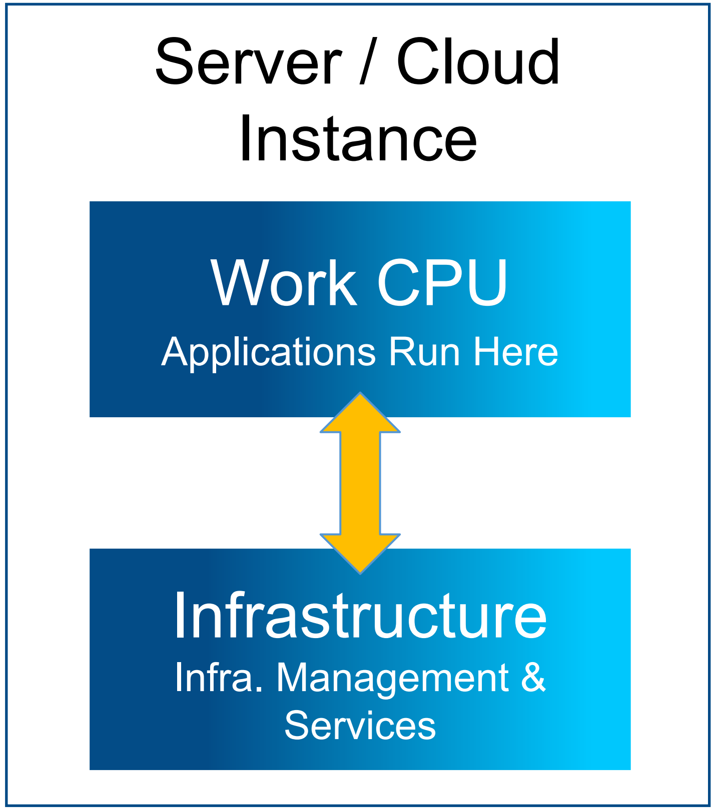
- Infra-to-pod Connections
- Infrastructure Programming

Common Methodology Across:

- Public & Private Cloud, On-prem
- Software & Hardware
- Vendors & Implementations

Separation Provides:

- Security (Airgap)
- More Available Cores
- Hardware Acceleration
- Feature Velocity



Session QR Codes will be sent via email before the event

Please scan the QR Code above to leave feedback on this session

Session Name: Kubernetes Networking Infrastructure Offload

Date: 8/17/22

Version #1

Format: Panel Total duration: 35 min.

Speaker/s **Dan Daly & Nupur Jain, Intel; Nabil Bitar, Bloomberg; Moshe Levi, Nvidia; Vytautas (Valas) Valancius, Google**

Section	**Intro**	**Requirements**	**Existing Public Cloud Models**	**Existing Private Cloud Models**	**Expanding Functionality**	**Call To Action & Questions**
Time	5 min (all)	5 min. (Nabil)	5 min. (Valas)	5 min. (Moshe)	10 min (Nupur)	5 min. (all)
Key Topics	Who We Are: • Dan (Panel Moderator) • Nabil (User) • Valas (Public Cloud Provider) • Moshe (DPU Vendor) • Nupur (IPDK Developer) Problem Statement: Current Kubernetes I/O is low performance, injects latency & jitter and burns up CPU cycles that could otherwise be doing more profitable things. It is also relatively difficult to set up and secure. [Release Velocity, be able to release new features seamless to the user, frictionless to deploy new features]	Want to attempt to solve this problem from the perspective of the user (Nabil represents us here) • The containerized workloads should not be impacted or need to change • The deployment of containers shouldn't have to change • The mechanisms to set up the network or storage should not change Want to address the performance & cycle usage issues without creating new ones in its place Enables accelerated datapath (bypasses) because the Policy is no longer in the kernel Host <-> Infrastructure Connection?] Why do we need this? - Performance - Security - Business Efficiency	Google Cloud already has mechanisms today to offload the routing between containers. • Description of how this is set up • Release Velocity • All of the fancy features are in the guest, contrast this with how VPC works • Compare the VPC VM performance to a container performance	NVIDIA has mechanisms to offload routing between containers in a private cloud environment using OVN. • Description of how this is set up • OVN is controlling the rules via SDN	Target Feature Set: Policies (Kubernetes Policy API), [Connection Aware] Load Balancing (Endpoint Detection), Routing (Pod-to-Pod) How it is done today (Kernel, eBPF) Optional Alternative- do it in the Infrastructure Theoretical Benefits of this Frees up cores Improves Bandwidth/Latency/Jitter Enables [kernel bypass technology] like AF_XDP, CNDP Enables [Networkless] in the Future Code – RPC to move the information into the Infrastructure, direct-infra-to-pod connection Demo – Set up the DPDK dataplane, deploy both the infrastructure pods & some test pods. Ping across Summary – Move the dataplane to the infrastructure. Enables IaaS vendors to optimize that logic in software or hardware Coming Soon – Purpose built IPUs that have this logic already optimized and running in their hardware dataplane	Users want this, and they want it to work the same across public, private, hybrid, etc. deployments Want to make it easy (a toggle) to decide if this functionality is done in the server w/ for example eBPF or if it is done in the infrastructure We want this to be supported in the same way independent of implementation, vendor, etc. Will work in open source to drive this consensus across the industry in a similar fashion that we arrived at eBPF solutions. Kubernetes control talking to the infrastructure (VPC, SmartNIC, DPU, IPU, Switch, etc.)

Abstract

Networking is central to Kubernetes, as it enables secure and deterministic scale out. As the number of services, pods, and interconnections increases, the kernel overhead will use more compute cycles, thereby lowering throughput and increasing latencies. Infrastructure Offload moves the Kubernetes cluster network policy, routing, and load balancing rules off of the compute platform and into the infrastructure. The cloud provider can then optimize these operations in software or in programmable hardware, such as an IPU or DPU, without requiring any changes to the end user's applications. In this panel, we discuss various approaches that share a common methodology based on existing Kubernetes APIs to improve performance, free up compute cycles, and preserve compatibility with existing cloud native applications.

Title

Content

Secure your cluster to cluster traffic, the agnostic way

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

October 24-28, 2021

Dave Kerr
Software Engineer
Workday

Pauline Lallinec
Software Engineer
Workday

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

Title

KubeCon | CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022

October 24-28, 2021

Speaker Name

Job Title, *Company*

Title

KubeCon | CloudNativeCon

North America 2022

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

October 24-28, 2021

PHOTO

Speaker Name
Job Title
Company

PHOTO

Speaker Name
Job Title
Company

PHOTO

Speaker Name
Job Title
Company

PHOTO

Speaker Name
Job Title
Company

PHOTO

Speaker Name
Job Title
Company

Session QR Codes will be sent via email before the event

Please scan the QR Code above to leave feedback on this session

Infrastructure Offload - Advantages

- Enhanced Security - Infrastructure components on a separate compute complex
- Enhanced performance with optimizations in dataplane
- Improved packet path latencies
- Additional free cores on the host for applications
- Software that can support multiple Cloud deployment models
- Feature velocity and ease of new feature introduction

Infrastructure Offload - Advantages

- Acceleration of network functionalities like encryption, compression, ACLs, packet encapsulation, decapsulation
- Dedicated memory for pipeline blocks
- Flow level visibility for monitoring and observability
- Access interface to other infrastructure pieces like storage
- Common plugin approach for Cloud Deployments with VM and Containers
- Support for Accelerated I/O devices