DPDK - Kubernetes Plug-ins For Accelerated Container Networking

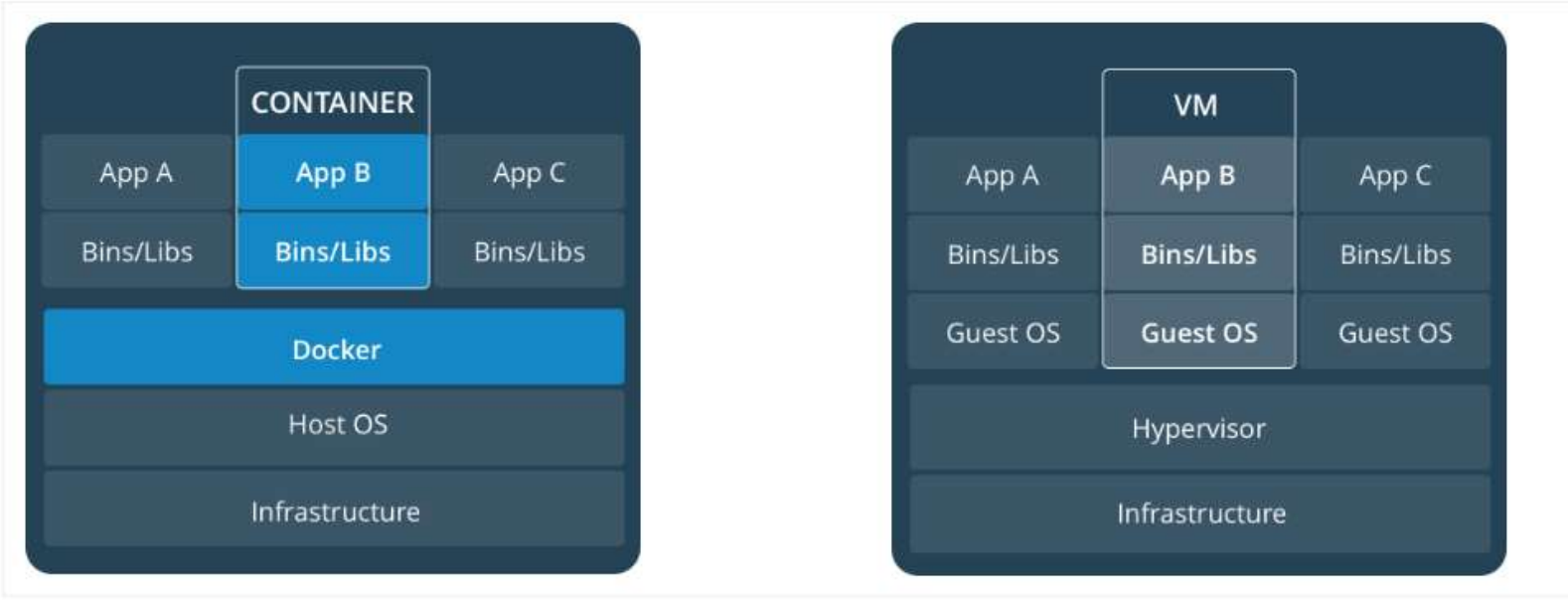
M JAY

PLATFORM APPLICATION ENGINEER

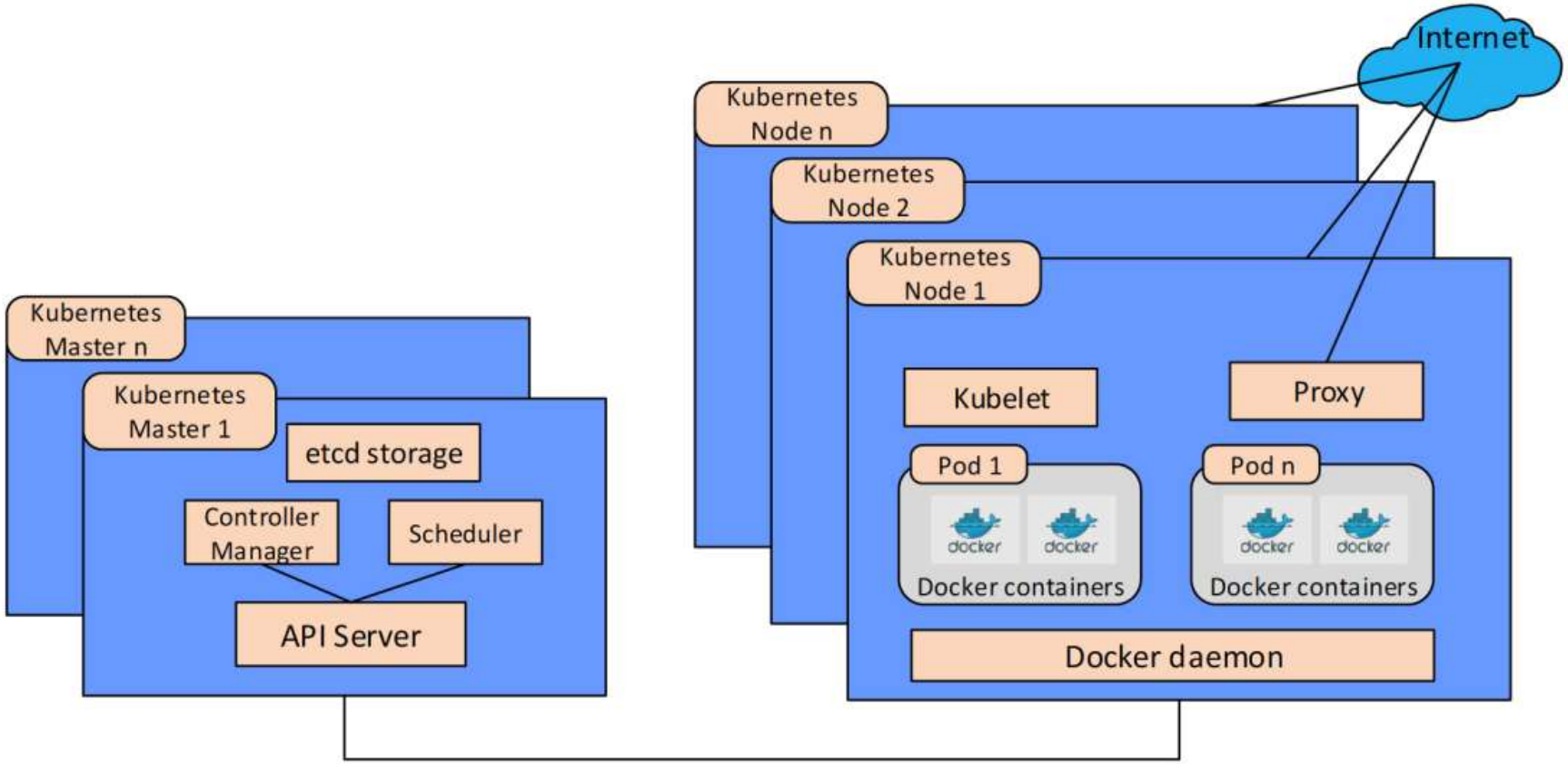
Agenda

- Container – Overview
- Container Vs VM
- What Do You Want?
- Container & DPDK - Commonality
- Why Do You Need Multiple Network Interfaces?
- Not All Nodes are created equal
- Call For Action

Container Versus VM



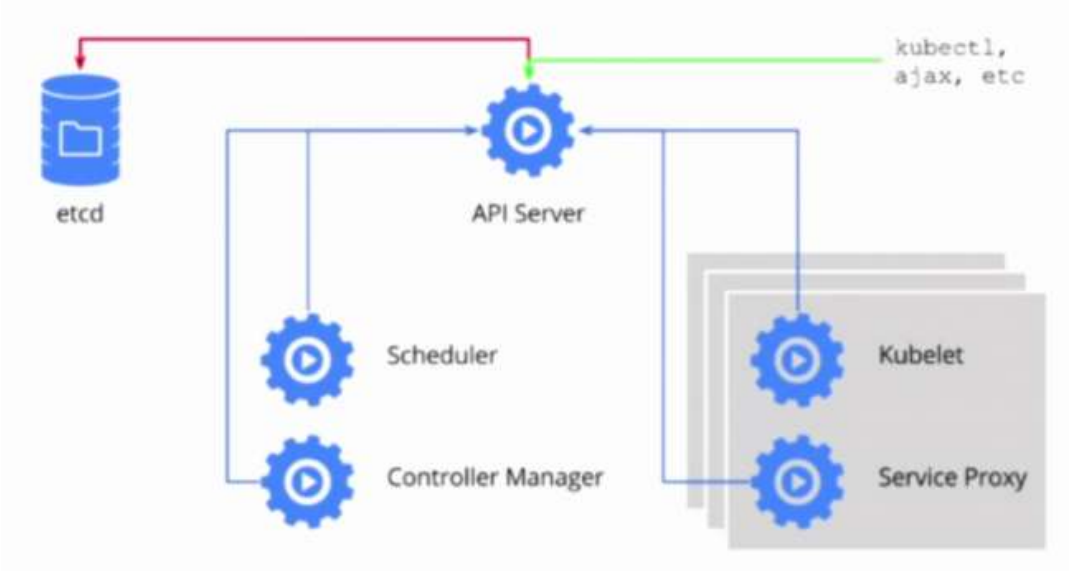
Kubernetes Cluster



https://builders.intel.com/docs/networkbuilders/enabling_new_features_in_kubernetes_for_NFV.pdf

Overview

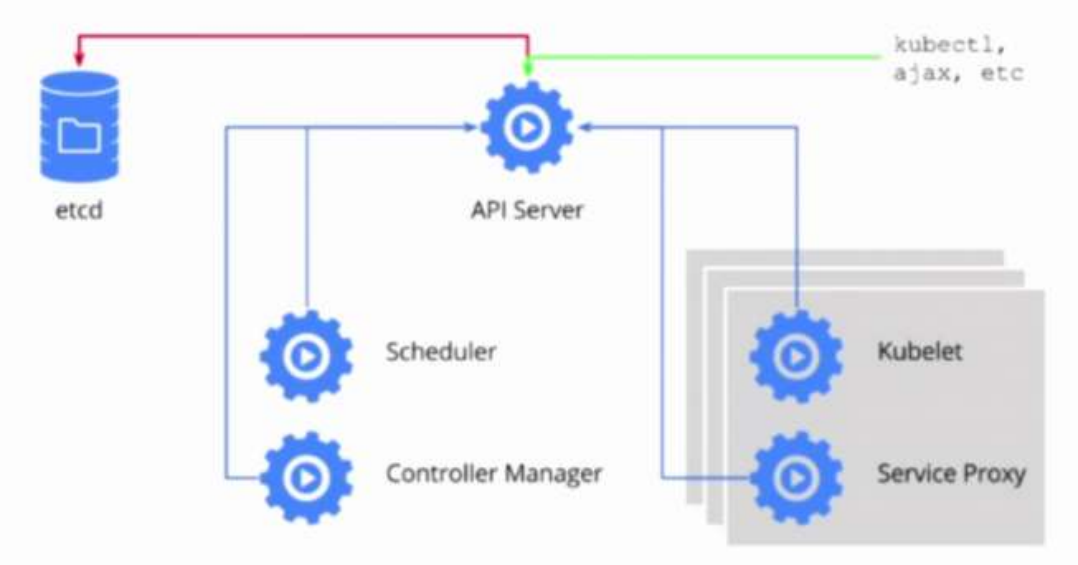
- **Container:**
  - Create an isolation boundary at application level. Portability, Ease of packing
- **Kubernetes:**
  - Automation for deployment, management
  - Application scaling of containers across clusters of servers (hosts)
- **Pod:**
  - Deployment unit. Can have single container or a small number of
  - containers that share resources – tightly coupled
- **Node:**
  - Worker or Minion – the machine where pods are deployed.
- **Kubernetes Master and Minion:**
  - Master controls managing and scheduling of pods to minions



https://thenewstack.io/taking-kubernetes-api-spin/

Overview

- **API Server:**
  - Front end to the clusters through which all other components interact
- **Scheduler:**
  - Decides target node onto which a pod would be sechduled
- **Controller Manager:**
  - Communicates with API server
  - Creates, updates, deletes the pods, service etc.,
- **Container runtime:**
  - Docker and Rocket for instance
- **Kubelet:**
  - Agent that registers a node to the cluster. Syncs up with K8 master.
  - It creates, deletes pods
- **Kube-proxy**
  - Network Proxy and reflects services as defined in Kubernetes API on each node



Macrocosm and Microcosm – Commonality

DPDK
DATA PLANE DEVELOPMENT KIT

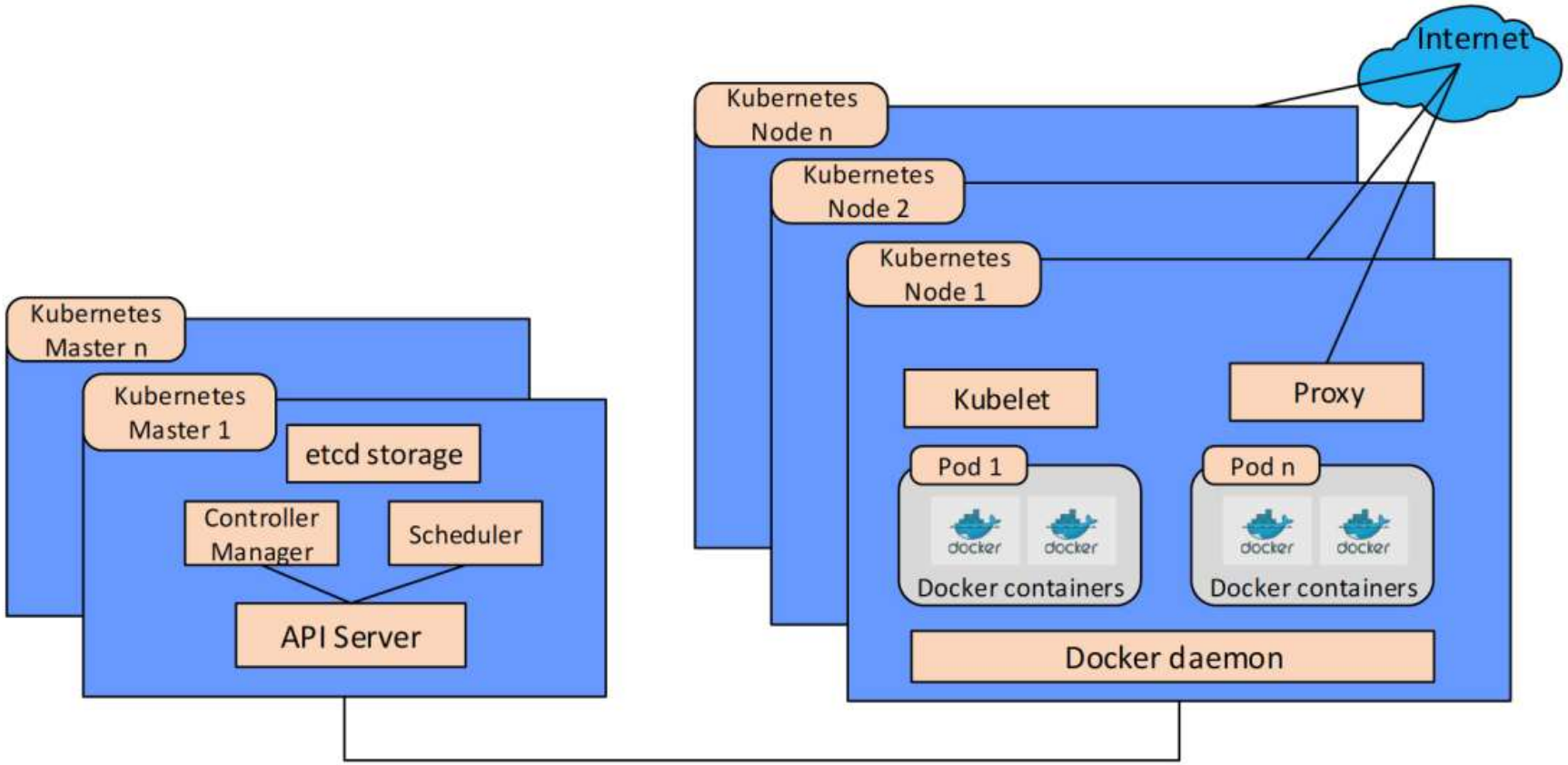
Container Technology	DPDK
Master Container	DPDK Control Plane Master core 0
Pod	Lcore
Node	Physical core
Multiple pods can reside in one Node	Multiple lcores (hyperthreads) reside in one Physical core
Microservice	Pthread / Lthread
Minion	Worker Core

Macrocosm and Microcosm – Commonality

DPDK
DATA PLANE DEVELOPMENT KIT

Container Technology	DPDK
API Server	DPDK Load Balancer Core
Scheduler	1) Pthread to lcore Mapping 2) Core Pinning, NUMA Affinity, Balanced IO 3) Hyperthreads sharing L1, L2 Cache 4) Non Siblings sharing L3 Cache but not L1, L2 Cache
Controller Manager	Environment Abstraction Layer, EAL managing resources
Etcd	Hashing – 5 tuple resolving to worker core
Kubelet	DPDK Dispatcher

Kubernetes Cluster



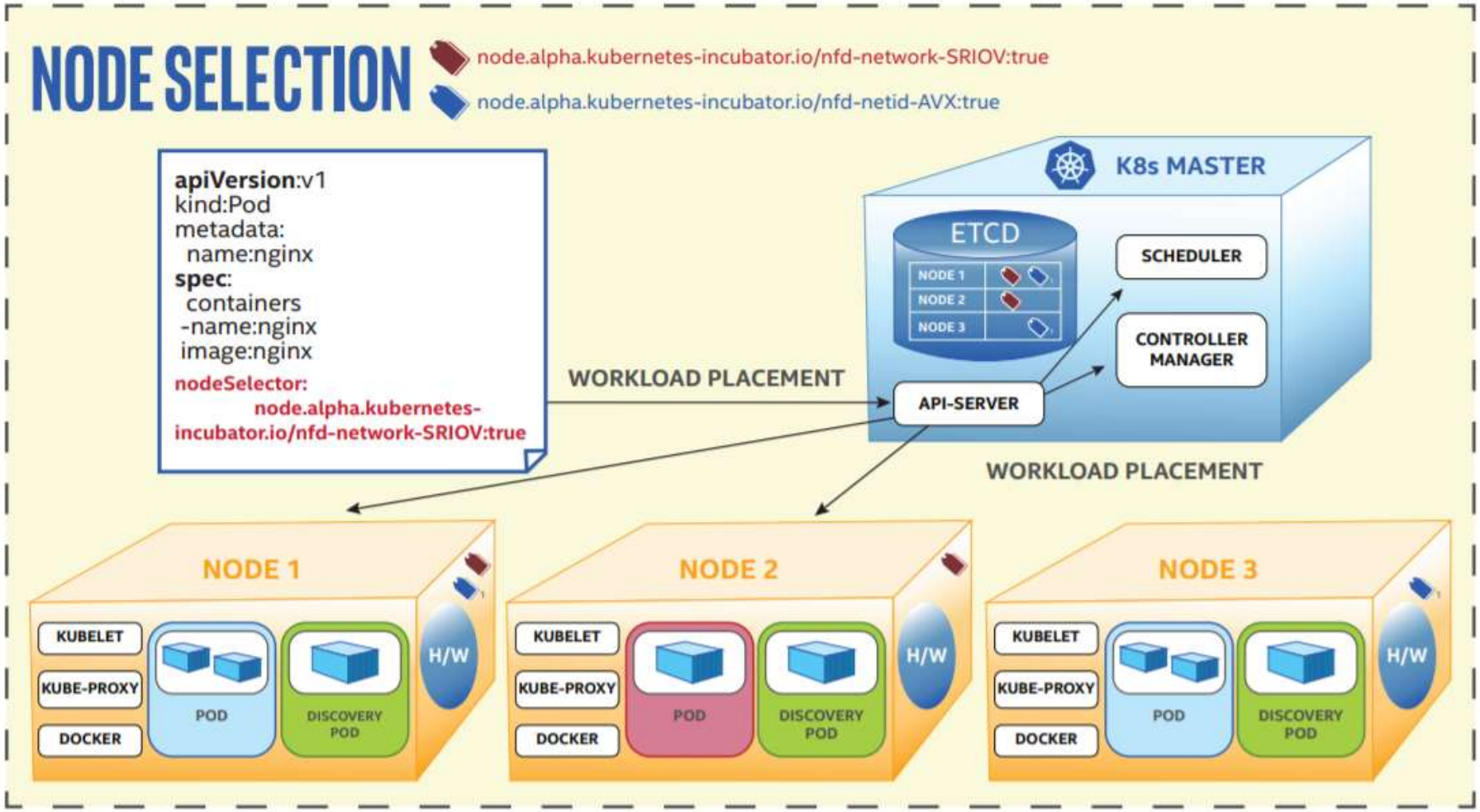
https://builders.intel.com/docs/networkbuilders/enhanced-platform-awareness-in-kubernetes-application-note.pdf

Enhanced Platform Awareness (EPA) represents a methodology and a related suite of changes across multiple layers of the orchestration stack targeting intelligent platform capability, configuration & capacity consumption. For communications service providers (CommSPs) using virtual network functions (VNFs) to provision services, EPA delivers improved application performance, and input/output throughput and determinism.

EPA underpins a three-fold objective of the discovery, scheduling and isolation of server hardware capabilities. This document is a comprehensive description of EPA capabilities and the related platform features on Intel® Xeon® Scalable processor-based systems exposed by EPA. Intel and partners have worked together to make these technologies available in Kubernetes:

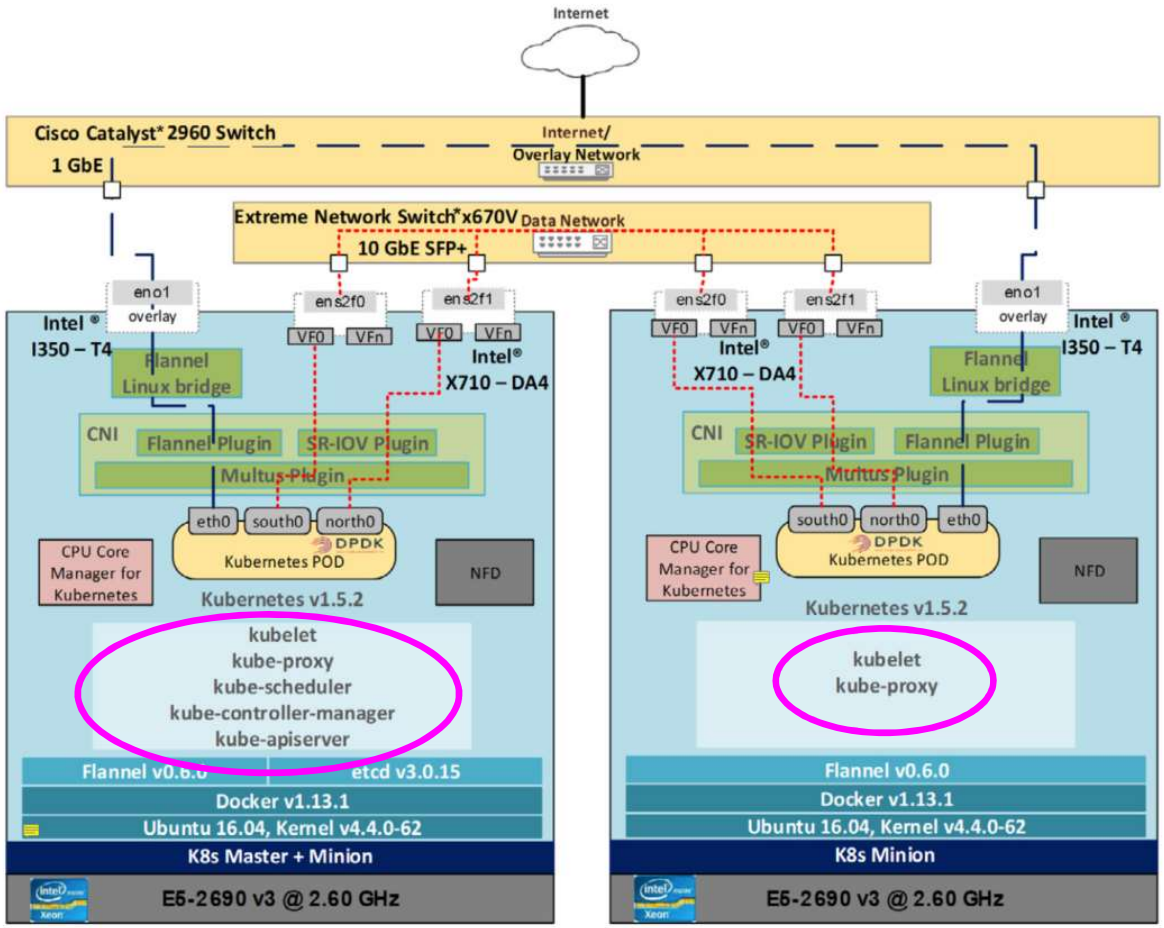
- Node Feature Discovery (NFD) enables Intel Xeon® Processor server hardware capability discovery in Kubernetes
- CPU Manager for Kubernetes (CMK) provides a mechanism for CPU pinning and isolation of containerized workloads
- Huge page support is native in Kubernetes v1.8 and enables the discovery, scheduling and allocation of huge pages as a native first-class resource
- Single Root I/O Virtualization (SR-IOV) for networking

Node Selection



Container Networking

System Architecture



Container Networking

DPDK
DATA PLANE DEVELOPMENT KIT

- Docker0 – Default Networking of Docker
- CNI – Kubernetes Container Networking Interface
  - Basic Overlay networking for containers in Kubernetes cluster
  - Plug-in based container solution for networking
  - common interface between the network plugins and container execution
- **Why CNI?**
- Application containers on Linux are a rapidly evolving area, and within this area networking is not well addressed as it is highly environment-specific.
- Many container runtimes and orchestrators will seek to solve the same problem of making the network layer pluggable.

CNI - Connects with One Network Interface to K8 Pod

Why Do You Need Multiple Network Interfaces?

DPDK
DATA PLANE DEVELOPMENT KIT

- Provide VNFs with redundancy of the network
- Segregate the control plane from the data plane traffic.
- Multiple stacks, different tuning and configuration requirement
- Network Slicing in high performance Networking
- Slicing offers container direct access to high performance NIC hardware

DPDK

CNI Plugin

Overview

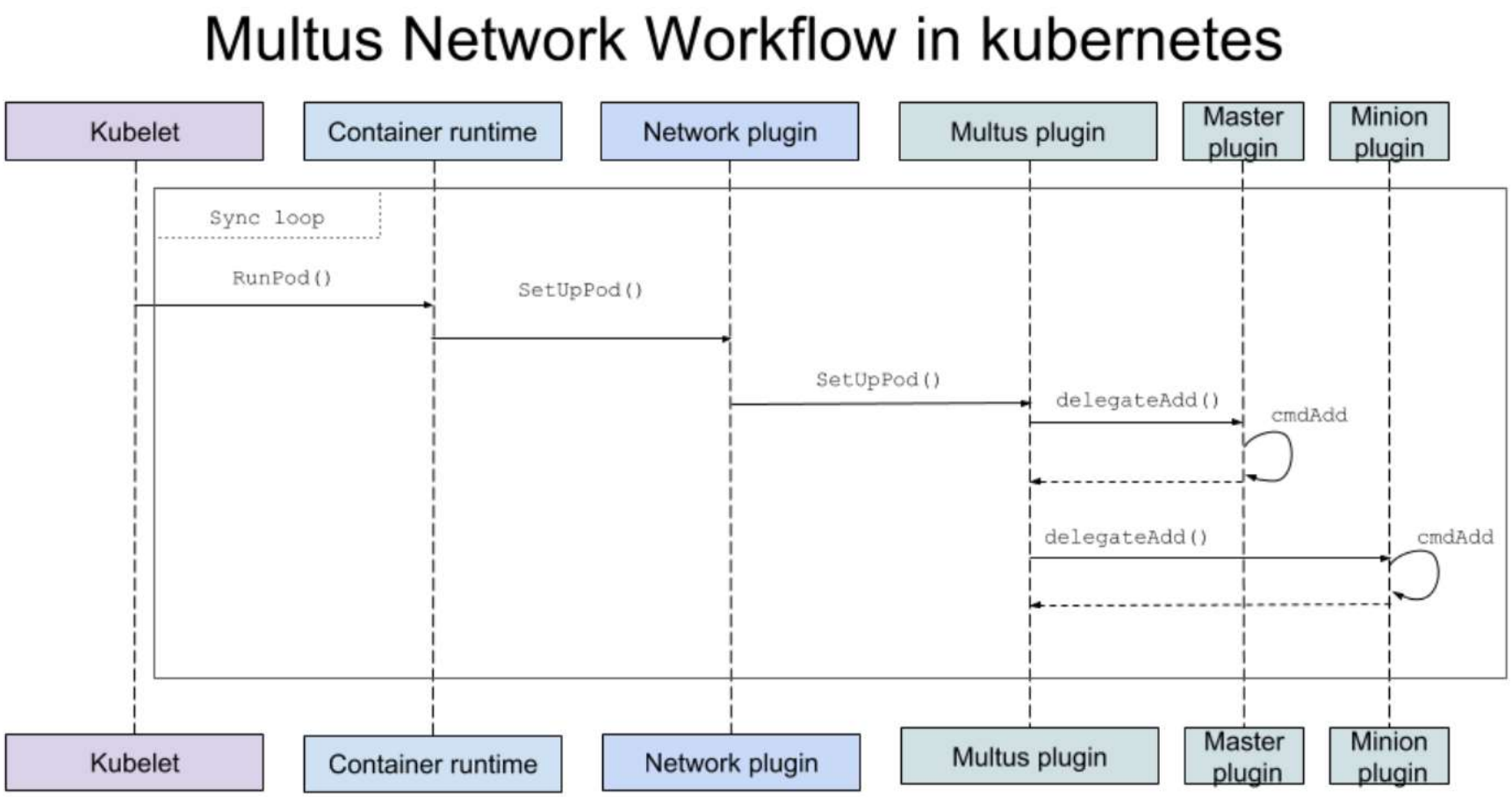
Each CNI plugin must be implemented as an executable that is invoked by the container management system (e.g. rkt or Kubernetes).

A CNI plugin is responsible for inserting a network interface into the container network namespace (e.g. one end of a veth pair) and making any necessary changes on the host (e.g. attaching the other end of the veth into a bridge). It should then assign the IP to the interface and setup the routes consistent with the IP Address Management section by invoking appropriate IPAM plugin.

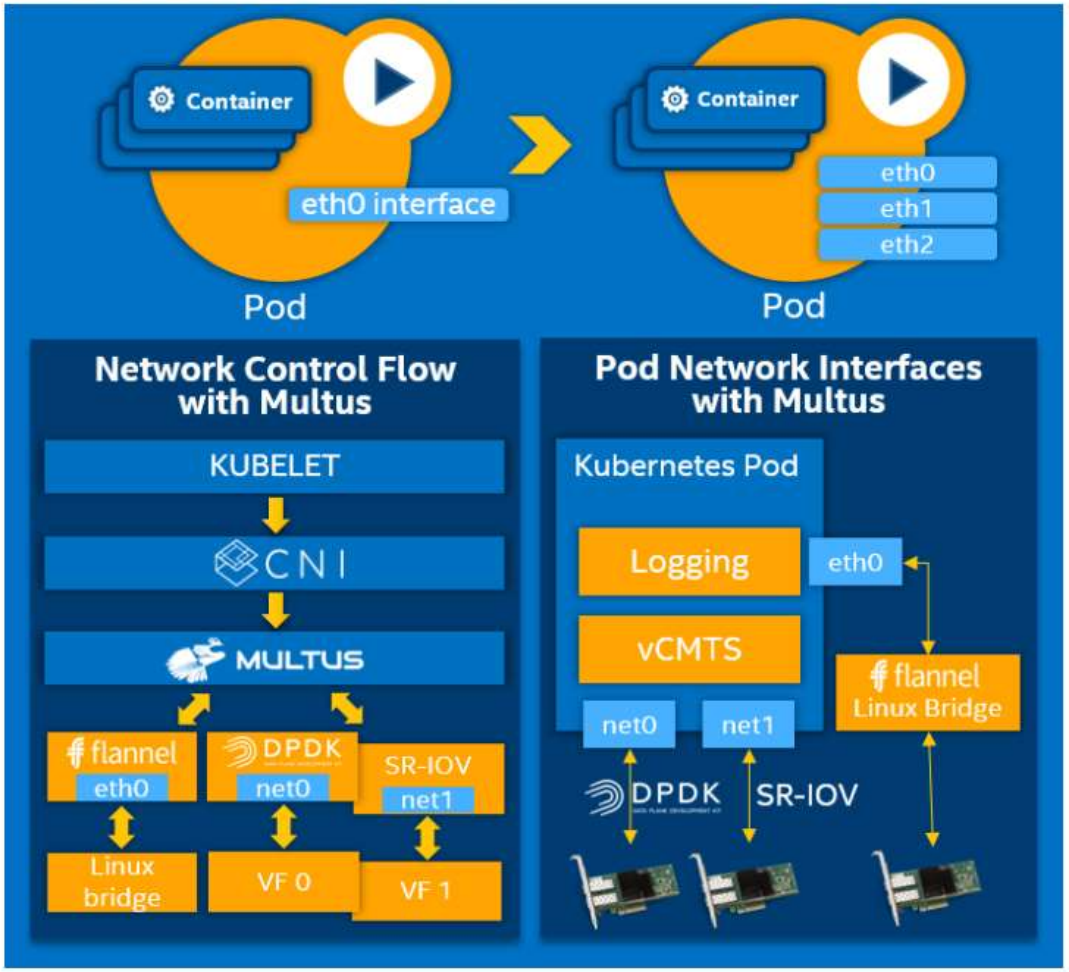
General considerations

- The container runtime must create a new network namespace for the container before invoking any plugins.
- The runtime must then determine which networks this container should belong to, and for each network, which plugins must be executed.

Multus Workflow



Multus Networking with SR-IOV/DPDK CNI



Software Components

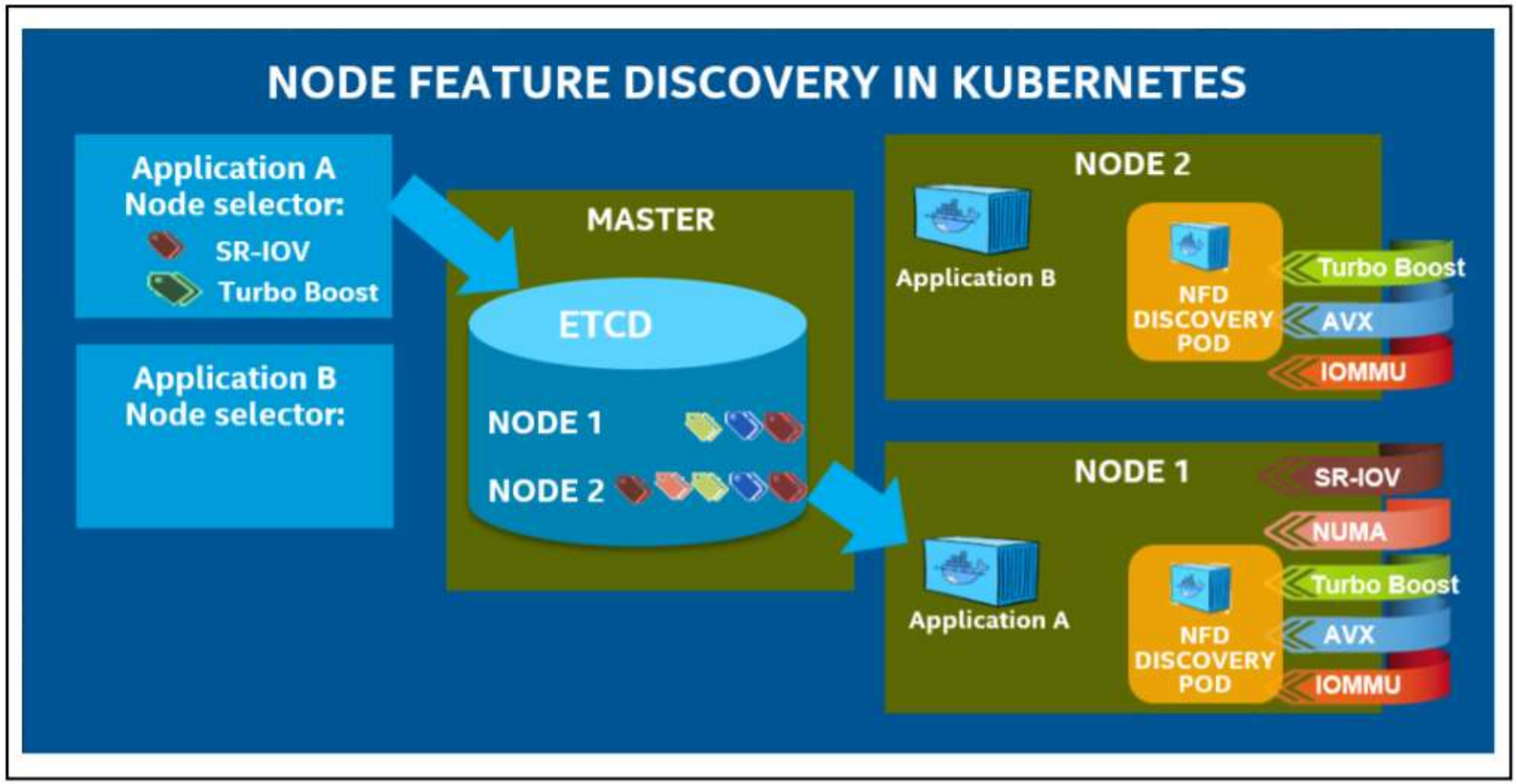
Software Component	Description	References
Host Operating System	Ubuntu* 16.04.2 x86_64 (Server) Kernel: 4.4.0-62-generic	https://www.ubuntu.com/download/server
NIC Kernel Drivers	i40e v2.0.30 i40evf v2.0.30	https://sourceforge.net/projects/e1000/files/i40e%20stable
Data Plane Development Kit	DPDK 17.05	http://fast.dpdk.org/rel/dpdk-17.05.tar.xz
CPU Manager for Kubernetes*	v1.1.0 & v1.2.1	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes
Ansible*	Ansible 2.3.1.0	https://github.com/ansible/ansible/releases
Bare Metal Container Environment Setup scripts	Includes Ansible* scripts to deploy Kubernetes v1.6.6 & 1.8.4	https://github.com/intel/container-experience-kits
Docker*	v1.13.1	https://docs.docker.com/engine/installation/
SR-IOV Network device plugin	V2.0.0 Commit SHA: f01659ef33aee4eb262687669f861bffced740d9	https://github.com/intel/sriov-network-device-plugin
SRIOV-CNI	v0.2-alpha. commit ID: a2b6a7e03d8da456f3848a96c6832e6aefc968a6	https://www.ubuntu.com/download/server

Additional github links

- https://github.com/containernetworking/cni/blob/master/SPEC.md
- https://github.com/Intel-Corp/multus-cni/
- https://github.com/intel/sriov-network-deviceplugin.git
- https://github.com/intel/sriov-network-deviceplugin/blob/master/images/sriovdp-daemonset.yaml
- Sample Deployment Specification Files - https://github.com/intel/sriov-network-deviceplugin/tree/master/deployments
- SR-IOV CNI Plugin https://github.com/intel/sriov-cni.git
- To Build User Space CNI https://github.com/intel/userspace-cni-network-plugin
- User space network object /github.com/intel/userspace-cni-networkplugin/examples

Node Feature Discovery

Node Feature Discovery - NFD



https://builders.intel.com/docs/networkbuilders/node-feature-discovery-application-note.pdf

1st Class.. 2nd Class Distinction

DPDK
DATA PLANE DEVELOPMENT KIT

- How to schedule pods that need high I/O traffic accordingly?
- Not all the nodes are created equal
- Some may have 1st class high performance components
- Others may be regular 2nd class
- How can we benefit in containers?
- How can we know which nodes have 1st class high performant building blocks?

Node Feature Discovery

DPDK
DATA PLANE DEVELOPMENT KIT

- Run-once K8s job
- Detects hardware features that are available at node level
- How does Kubernetes use this information?
- For scheduling containerized VNFs for best match

CPU Core Manager

Cloud Tenants Demand Performance

- Enhanced Platform Awareness & CPU Pinning
- Single node can run many pods
- Some of these pods could be running CPU-intensive workloads
- In such a scenario, these pods might content for more CPU in that node
- Let us say the pod is throttled and depending on availability of CPU,
- The workload could move to different CPU
- Or
- Workload could be sensitive to context switches.

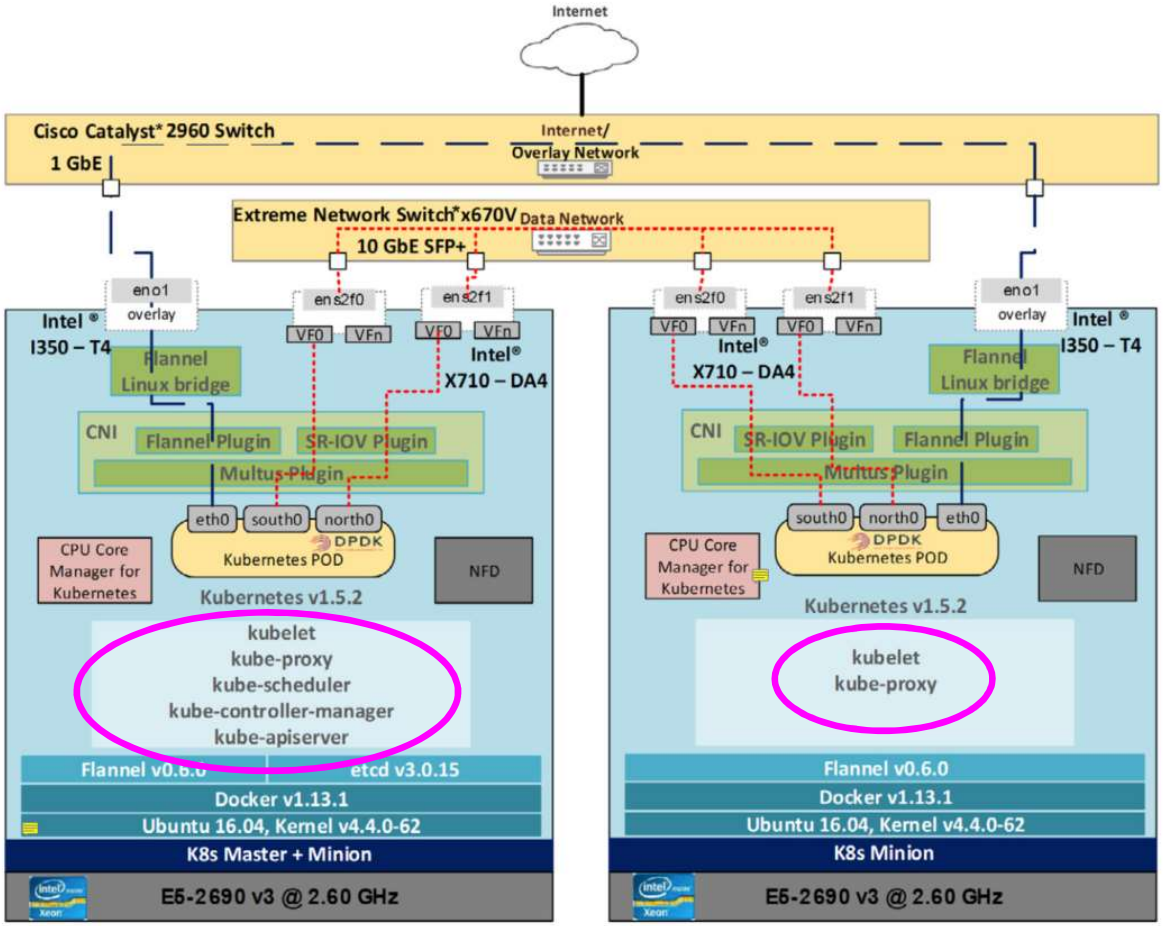
- Enter CPU Core Manager

CPU Manager – Low Latency, High Perf Containers

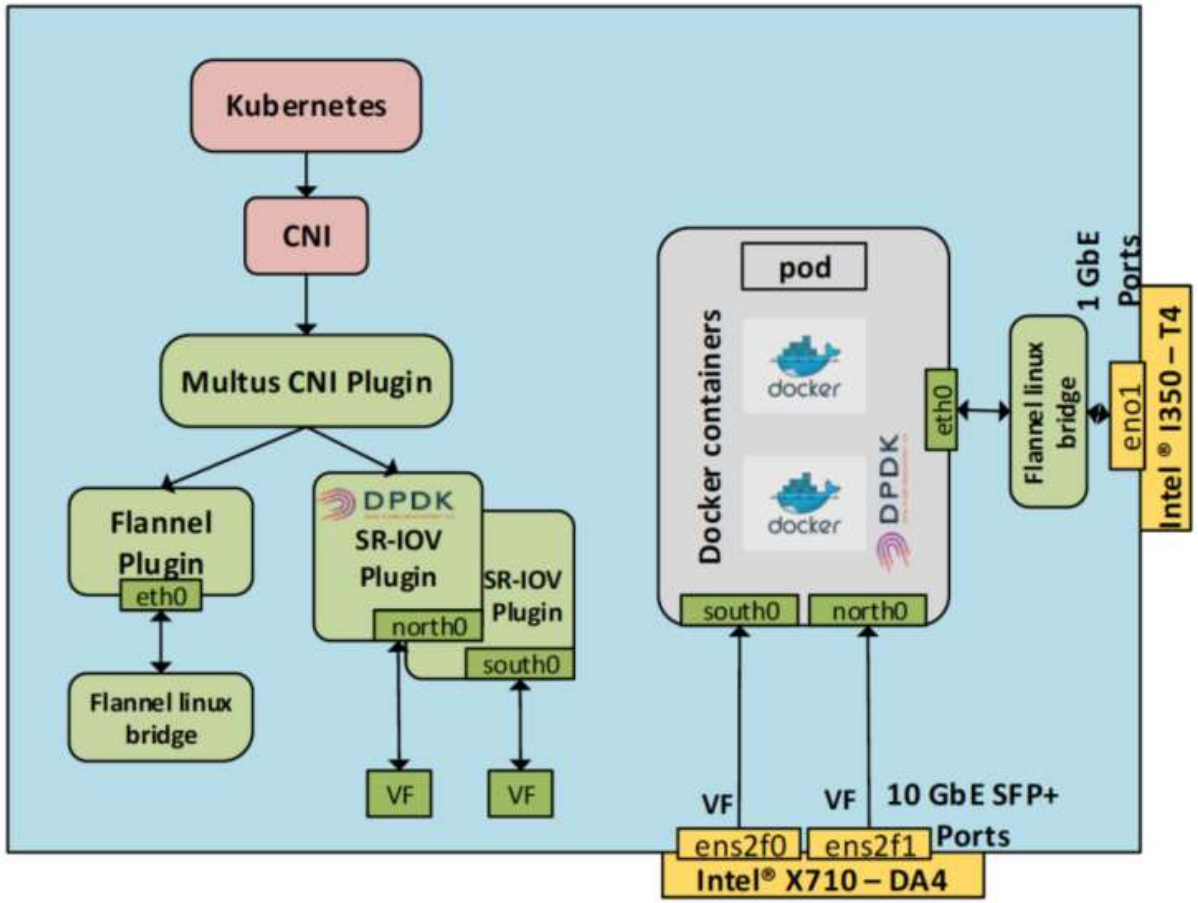
- Workload could be sensitive to context switches.
- How CPU manager can help?
- It allocates exclusive CPUs for demanding workload

- Enable CPU manager with policy
- Configure your pod to be in the Guaranteed QoS class.
- Request whole numbers of cores for containers needing exclusive cores

System Architecture



Multus CNI Plugin



https://builders.intel.com/docs/networkbuilders/enabling_new_features_in_kubernetes_for_NFV.pdf

Multus CNI Plugin

- What enhancement Multus CNI plugin brings?
  - Multus CNI allows K8s pods to be multi-homed
  - High performance Networking
- Which is default route? Who defines it?
  - Masterplugin is for control plane and default route
  - Flannel
- What about High performance Data path?
  - Minionplugin for data plane
- What are some examples of Minionplugin?
  - Flannel,
  - IPAM [Internet Protocol Address Management]
  - SR-IOV CNI Plugin – VF Interface Using Kernel
  - SR-IOV CNI Plugin – VF Interface using DPDK

Multus CNI Plugin

DPDK
DATA PLANE DEVELOPMENT KIT

Network	Description	I/F name shown on host
External / Internet	1) To Access Internet 2) Remote Access to host 3) Access to K8s pods	Eno1
Overlay Network	Overlay network for K8s pods	eno1
Data Network	High performance Data plane – SR-IOV VFs	Ens2f0 / ens2f1

SR-IOV CNI Plugin

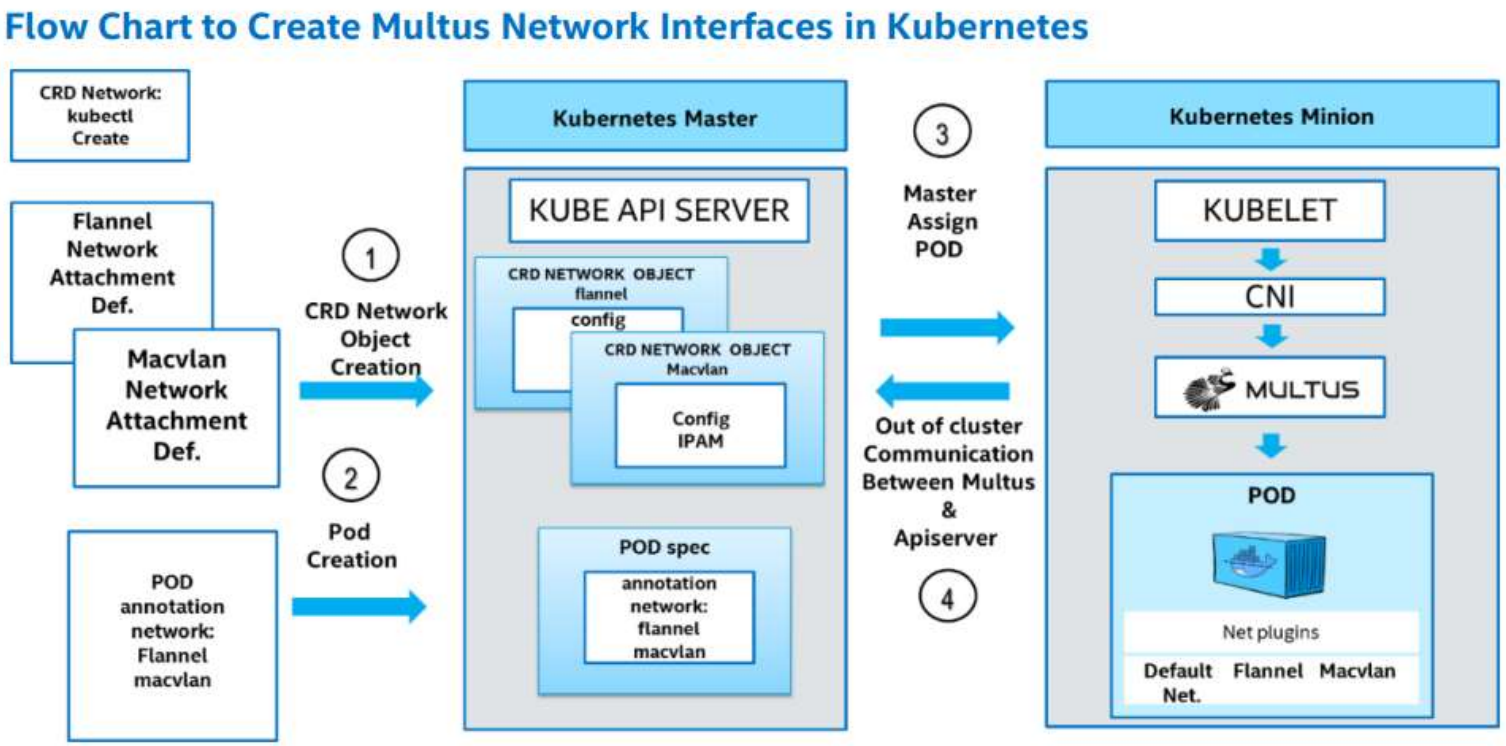
DPDK
DATA PLANE DEVELOPMENT KIT

- What enhancement SR-IOV CNI Plugin brings?
  - SR-IOV CNI plugin enables high performance networking
  - K8s pods to connect to SR-IOV VF.
- What drivers SR-IOV CNI Plugin supports?
  - DPDK Drivers such as VFIO-PCI
  - Kernel VF Drivers
- **Note for general CNI**
- 1) **Configuration files** – For all pods within the node having same network interface
- 2) **Objects** – Each pod has different network interface

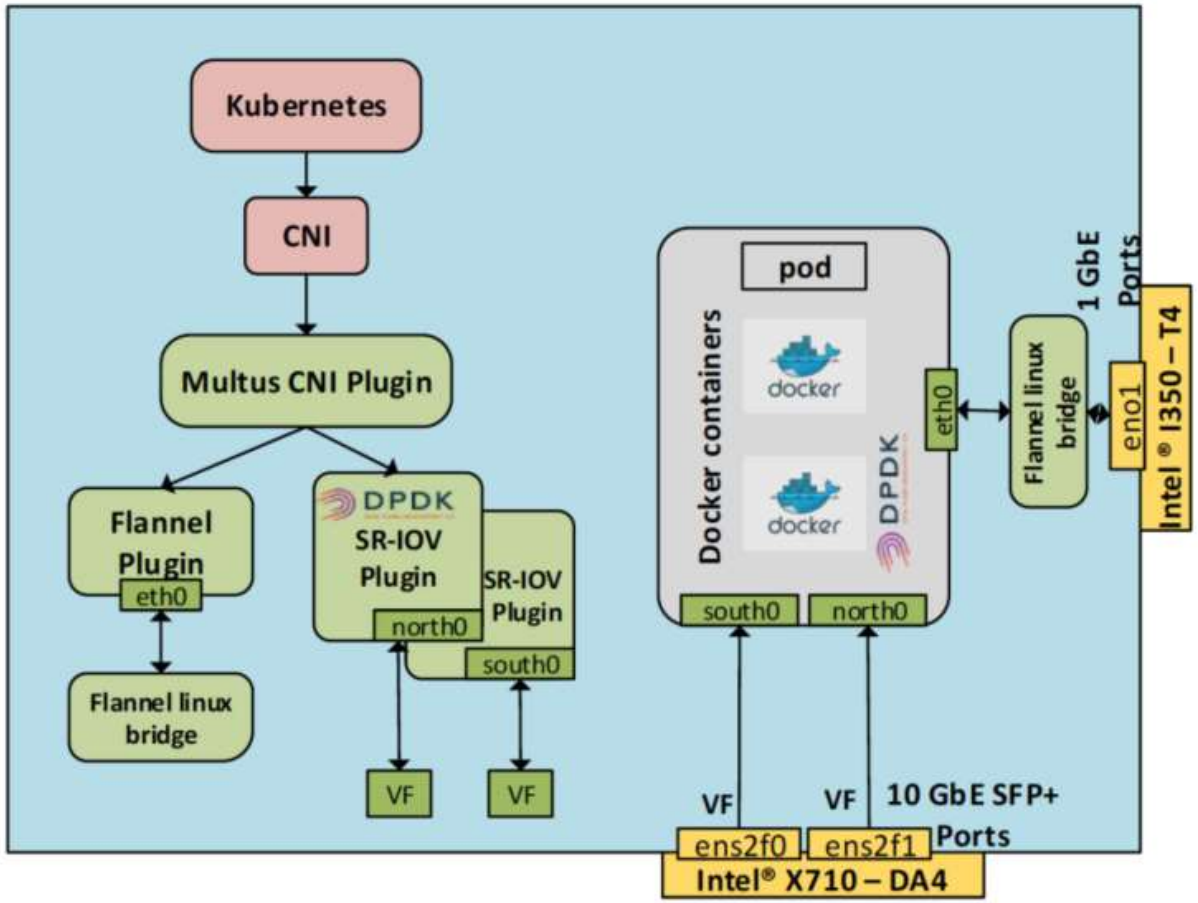
  - Pod A spec with network object annotation "SRIOV" connected to SR-IOV networks with the default network.
  - Pod B spec without any network object annotation, but having "Weave" as default network, connected to Weave network.

Flow Chart

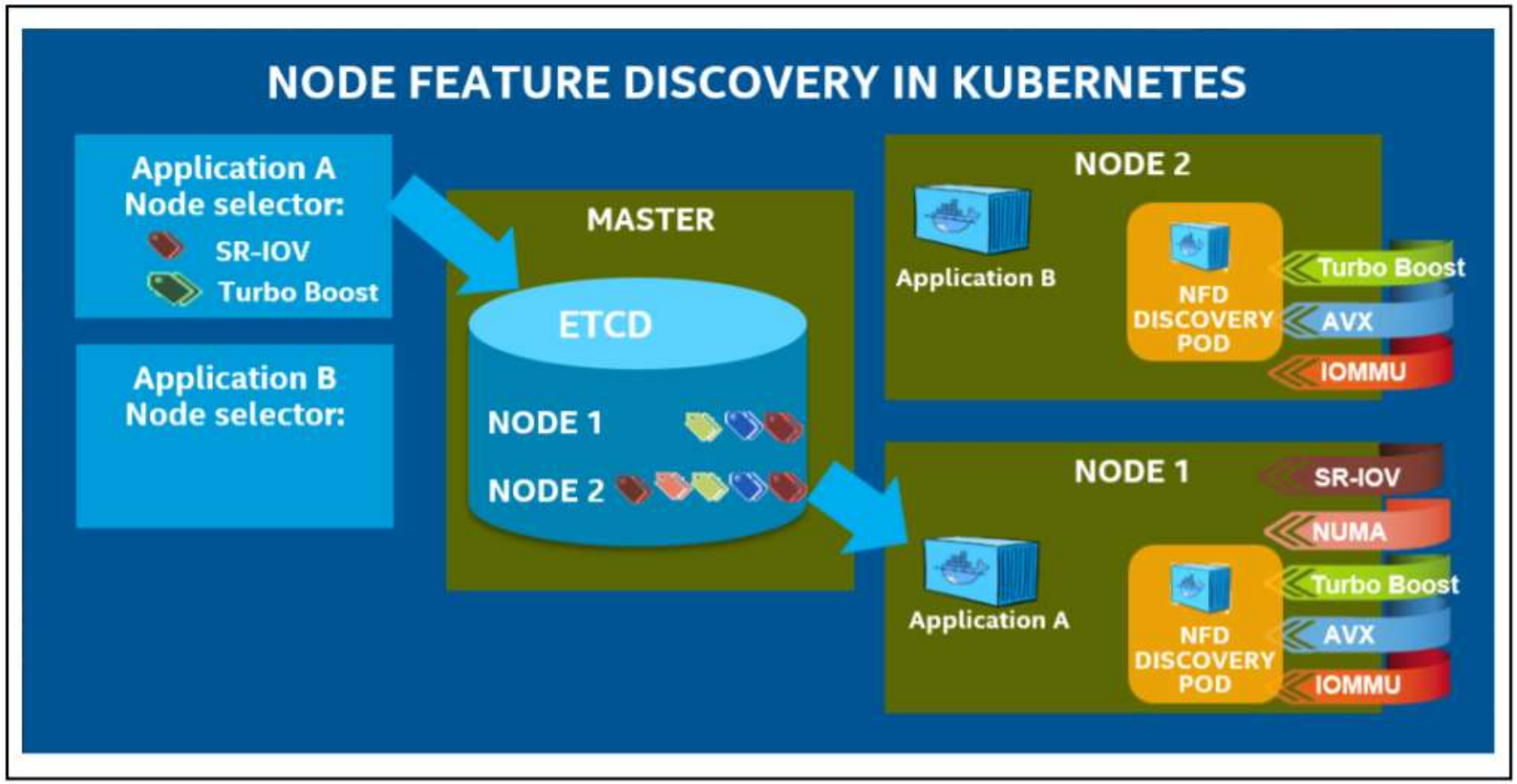
- Custom Resource Definition – Kubectl to create CRD



Multus CNI Plugin



Node Feature Discovery - NFD



https://builders.intel.com/docs/networkbuilders/node-feature-discovery-application-note.pdf

Node Feature Discovery Script - Mechanism

- NFD detects hardware features available on each node in a K8 cluster
- Advertises those features using ***node labels***
- Node Feature Discovery Script launches a job
- That job deploys a single pod on each unlabeled node.
- When each pod runs, it contacts the K8 API server to add labels to the node.

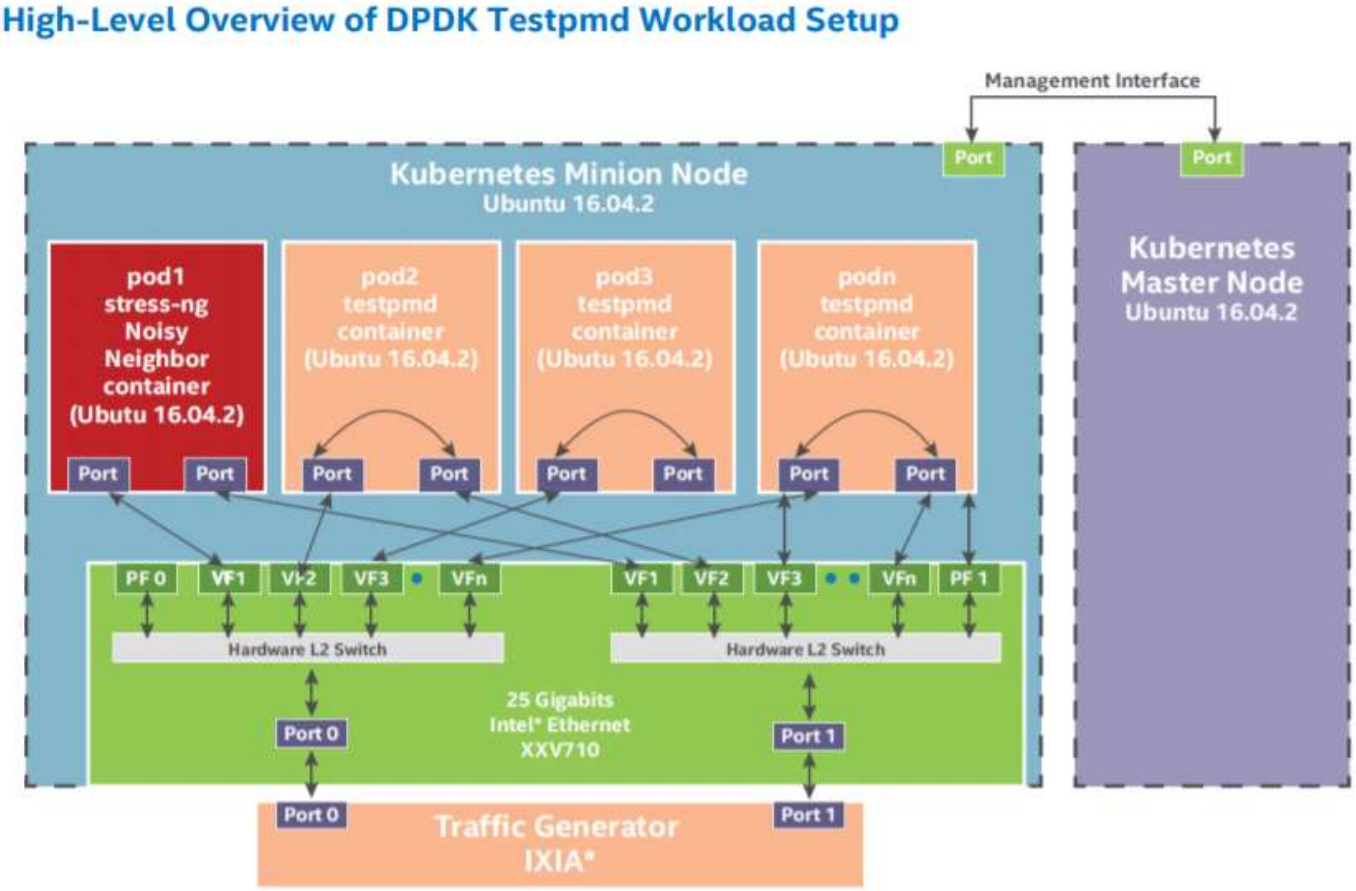
Node Labels Usage - etcd

DPDK
DATA PLANE DEVELOPMENT KIT

- Key/pair values – attached to pods or nodes
- Labels generated by NFD can be checked from the master node with **kubectl** commands

- Specify identifying attributes of objects relevant to end user
- Useful to organize objects into specific subsets
- All the Information is kept within **etcd**

CPU Manager


High-Level Overview of DPDK Testpmd Workload Setup
Management Interface
Kubernetes Minion Node
Ubuntu 16.04.2
Port
Port
pod1
stress-ng
Noisy
Neighbor
container
(Ubutu 16.04.2)
pod2
testpmd
container
(Ubutu 16.04.2)
pod3
testpmd
container
(Ubutu 16.04.2)
podn
testpmd
container
(Ubutu 16.04.2)
Port
Port
Port
Port
Port
Port
Port
Port
PF 0
VF1
VF2
VF3
VFn
VF1
VF2
VF3
VFn
PF 1
Hardware L2 Switch
Hardware L2 Switch
25 Gigabits
Intel® Ethernet
XXV710
Port 0
Port 1
Port 0
Port 1
Traffic Generator
IXIA*
Kubernetes
Master Node
Ubuntu 16.04.2

CPU Core Manager – 3 Pools of Processors

```
etc
|___ kcm
     |___lock                #system lock to prevent conflict
     |___pools
        |___controlplane
        |     |___0,1        #logical core IDS
        |     |    |__tasks #contains linux Process IDs
        |___dataplane
        |     |___2,3
        |     |    |___tasks
        |     |___4,5
        |     |    |___tasks
        |     |___exclusive #Determine exclusivity (exclusive == 1)
        |___infra
              |___6,7
              |    |___tasks
```

- Data plane pool is exclusive
- Control plane pool and Infra pools are shared
- When there is no pool mentioned in pod specification, then ?
  - CPU Core manager will use cores from the infra pools

Infra –> un-isolated

Remember Isolcpu in /proc/cmdline ?

Call For Action

DPDK
DATA PLANE DEVELOPMENT KIT

- Multus CNI Plugin
  - Segregate your data plane from control plane
  - Redundancy in your network
- SR-IOV CNI Plugin
  - Attach directly to high performance Network Slicing
  - High performance Containerized VNF
- Node Feature Discovery
  - Deploy Pods on nodes with the desirable hardware features
- CPU Core Manager For K8s
  - Isolate your minions to do their work

Acknowledgements

https://docs.docker.com/get-started/#containers-vs-virtual-machines#containers-vs-virtual-machines

https://thenewstack.io/taking-kubernetes-api-spin/

https://kubernetes.io/blog/2018/07/24/feature-highlight-cpu-manager/

https://builders.intel.com/docs/networkbuilders/enabling_new_features_in_kubernetes_for_NFV.pdf

Contact

M Jay
Muthurajan.Jayakumar@intel.com