KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon
North America 2019

RDMA Enabled Kubernetes for High Performance Computing

Jacob Anders, CSIRO
Feng Pan, Red Hat

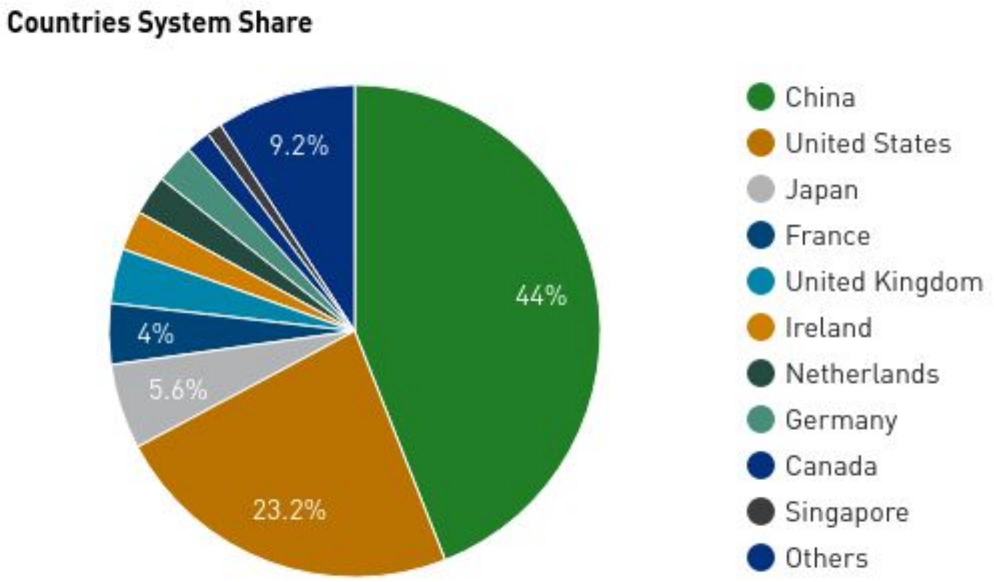
What is HPC and why is it important?

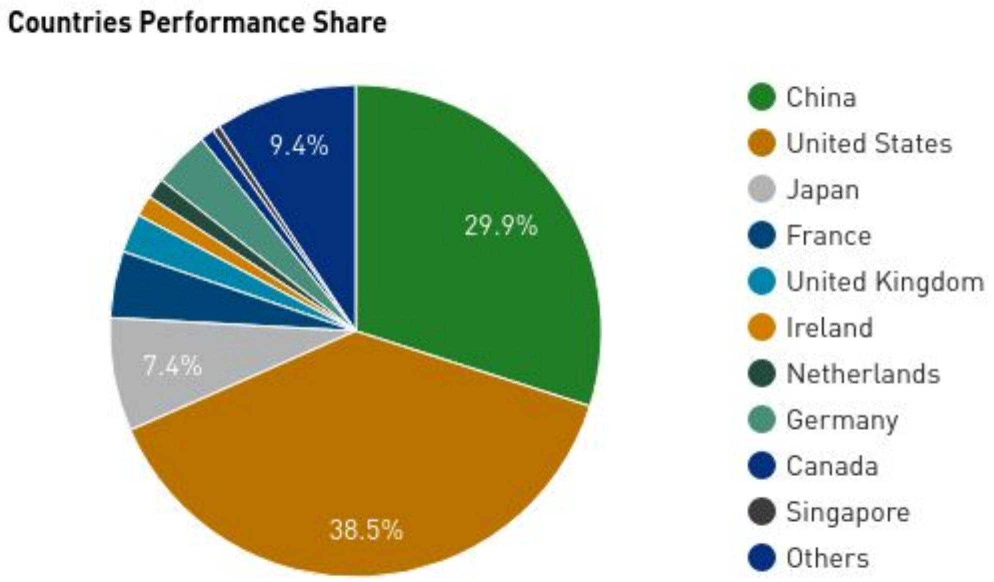
- **High-performance computing** (HPC) refers to systems that, through a combination of processing capability and storage capacity, **can rapidly solve difficult computational problems**

- Access to state-of-the-art HPC facilities **key to success in key areas of research**, such as medicine, engineering, geoscience, chemistry, defence technologies, weather modeling and more

What is HPC and why is it important?

- **Highly competitive field**, with strong government support and investment in US, China, Europe, Australia, Singapore and other countries **competing for** regional and/or **global leadership**





(data source - http://top500.org)

How can Kubernetes help HPC?

- HPC community developed a wide range of optimised hardware and software (job schedulers, communication libraries, parallel file systems)

- However, among HPC users there is a **growing requirement for tools** and capabilities that **don’t fully fit in traditional boundaries** of HPC. This includes:

  - **interactive** and **persistent** workloads
  - network **isolation**
  - custom application stacks *(“can I run my own docker container”)*

- These capabilities are often **requested by researchers** working in **emerging and strategically important** areas of science such as **biomedical research, cyber security and machine learning**

How can Kubernetes help HPC?

Advantages that Kubernetes can bring to the HPC community:

- **portability** and **reproducibility**, ease of automation
- **proven scalability**, wide adoption, well-tested code base
- **agility** - from code in git to production in minutes
- **large user base** and active community

Advantages that HPC community can bring to Kubernetes:

- HPC has **scale and performance** requirements **rarely seen elsewhere**
- Challenges solved for HPC often **benefit other areas** (e.g. NFV)
- Perfecting **general-purpose tools** and frameworks **benefits wider community**.

Networking in Cloud Computing

Key considerations are **security and adaptability**:

- Strong focus on on **multi-tenancy** and security
- **Dynamic** - Software-defined-networking (**SDN**) a de-facto standard
- **Generic** to ensure portability of workloads
- Latency and bandwidth can vary
- “noisy neighbour” scenario is not uncommon

Performance is important, but in most cases **is secondary** to the above

Networking in HPC

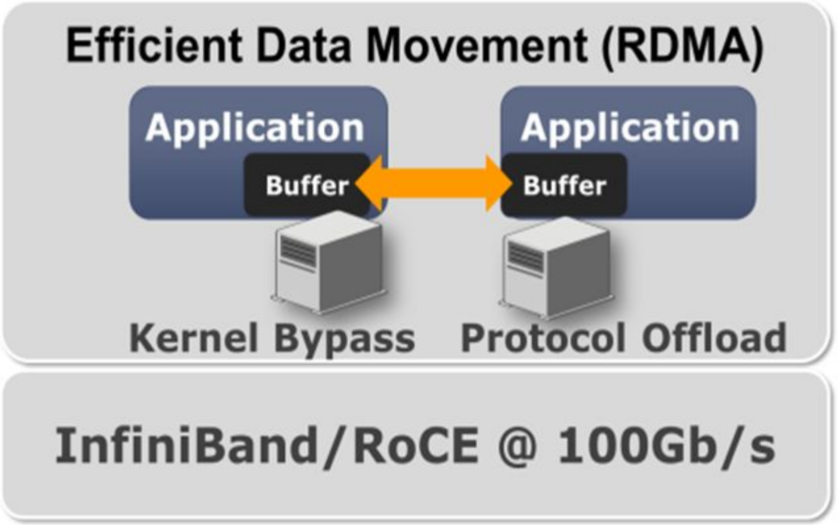
High and consistent **performance is king**...

- **Low latency**
- **High bandwidth**
- Minimising jitter (performance fluctuation)
- **advanced topologies** maximising bisection bandwidth
- often utilising **specialised hardware** and offloads
- typically quite **static**
- **limited multi-tenancy** - data access typically managed with filesystem permissions

… other requirements are considered secondary

Networking in HPC - RDMA

- Remote Direct Memory Access (RDMA)
- Advance transport protocol (same layer as TCP and UDP)
- Main features
  - Remote memory read/write semantics in addition to send/receive
  - Kernel bypass / direct user space access
  - Transport fully offloaded to the NIC HW
  - Secure, channel based IO
- Application advantage
  - Low latency
  - High bandwidth
  - Low CPU consumption
- RoCE: RDMA over Converged Ethernet
  - Available for all Ethernet speeds 10 – 200G
- Verbs: RDMA SW API (Similar to sockets)

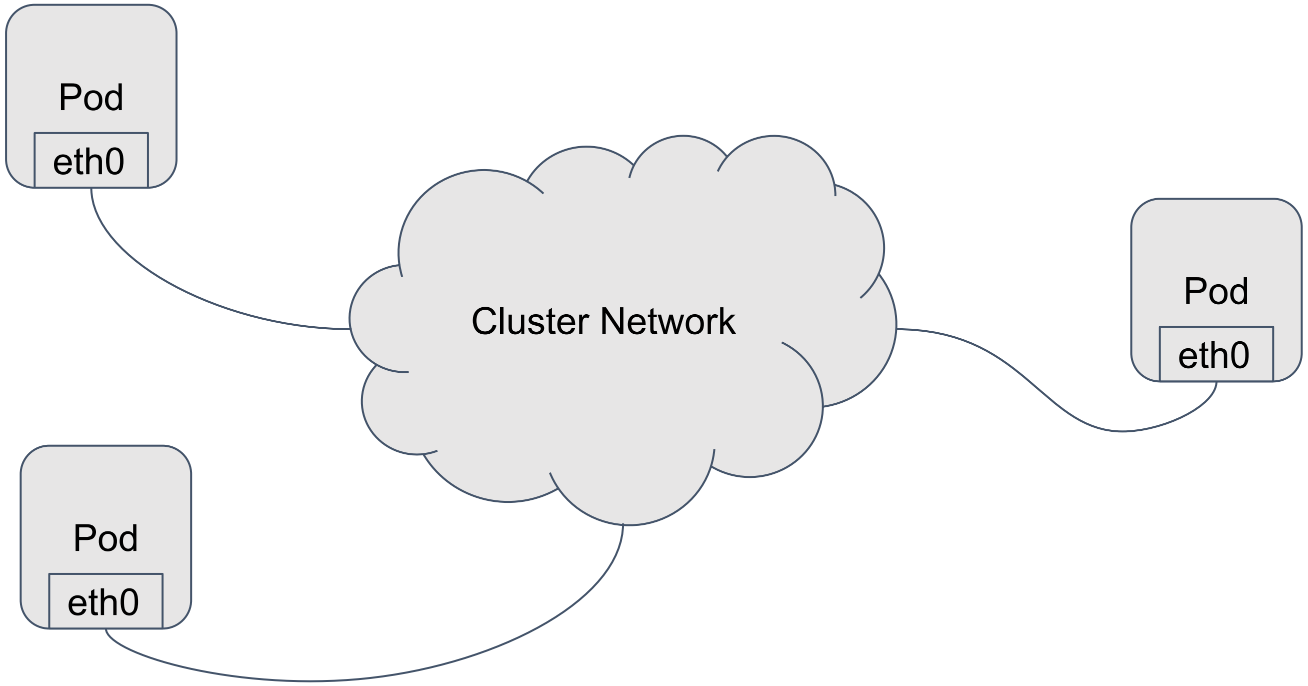


Implementation

- Multus
- SR-IOV Device Plugin with RDMA support
- SR-IOV CNI
- Kubernetes CPU Manager
- Kubernetes Topology Manager

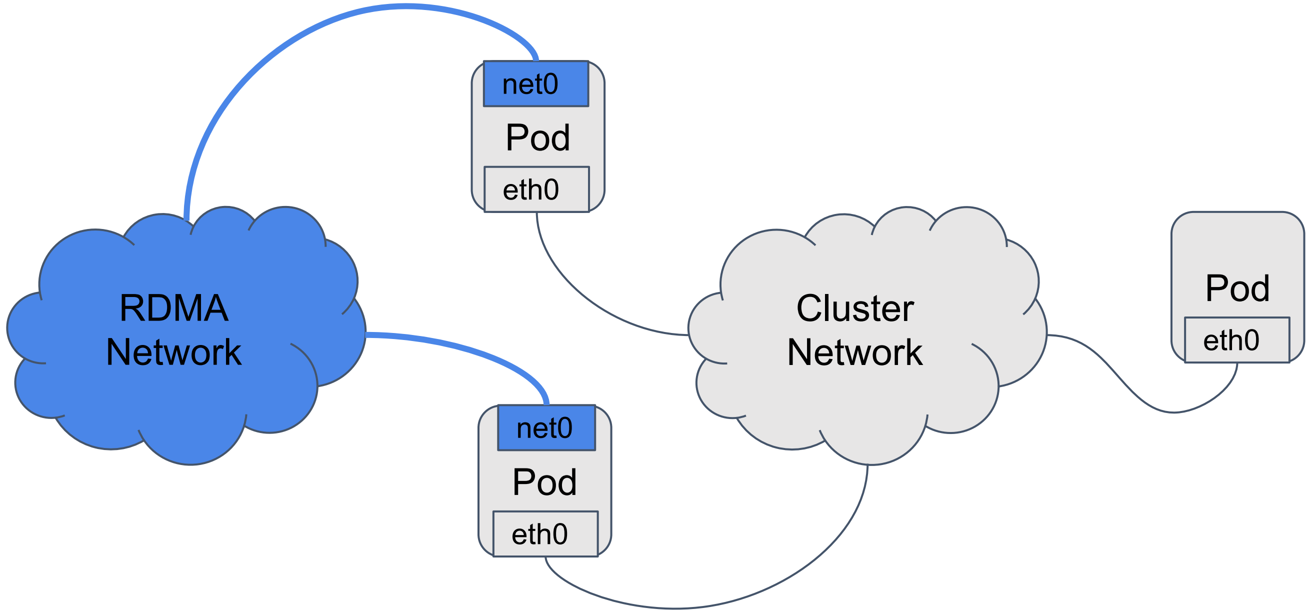
Kubernetes Networking

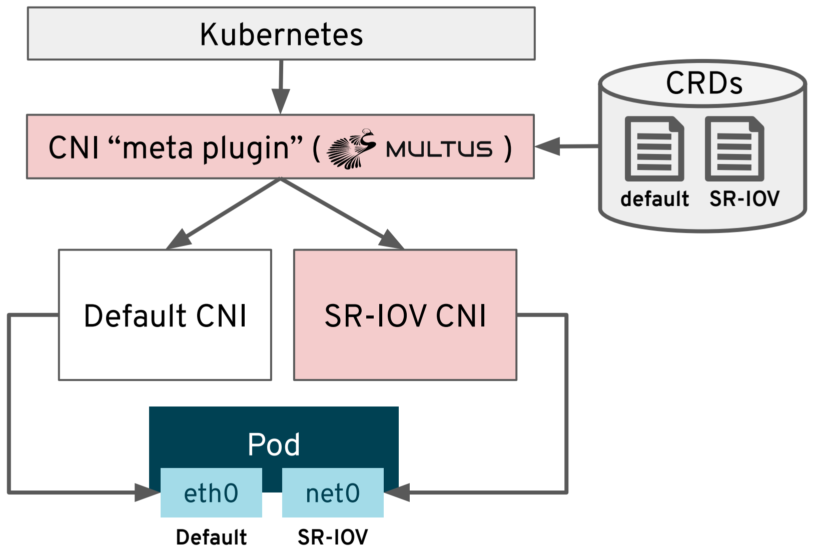
- Every pod is assigned its own IP address
- A pod can communicate with all other pods without NAT
- A node can communicate with all pods on all nodes without NAT



Multus

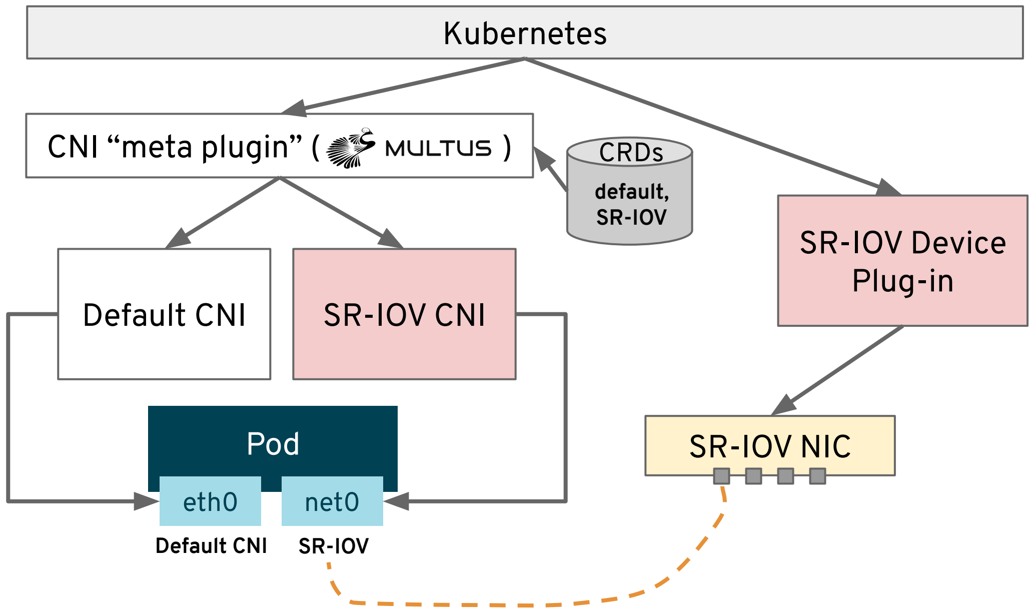
- Meta CNI Plugin enabling pods to connect to multiple networks
- Network Plumbing Working Group standard CRD spec





Enabling SR-IOV and RDMA

- SR-IOV Device Plugin
  - Manages and advertises SR-IOV nics to Kubernetes
  - Configure RDMA
  - Communicates NUMA information with Kubelet
- SR-IOV CNI
  - Plugs SR-IOV VF into a pod
  - Supports both Kernel and DPDK modes
  - Manages IPAM



CPU Manager and Topology Manager

- CPU Manager
  - Static Policy
  - Allocate dedicated CPUs to container
  - Node level config
- Topology Manager
  - Alpha feature in 1.16
  - Coordinate CPU and Device resource on node level
  - NUMA Alignment

RDMA Workload Pod Spec

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: rdma-pod-1
 annotations:
   k8s.v1.cni.cncf.io/networks: rdma-network
spec:
(...)
   resources:
     requests:
       mellanox.com/mlnx_sriov_rdma: '1'
       hugepages-1Gi: 4Gi
       cpu: '6'
       memory: 100Mi
     limits:
       mellanox.com/mlnx_sriov_rdma: '1'
       hugepages-1Gi: 4Gi
       cpu: '6'
```

Lab setup

Performance benchmarks were run on two servers with:

- Kubernetes 1.16.2,
- CentOS 7.7,
- Kernel 3.10.0-1062.4.1.el7.x86_64,
- Intel Xeon Gold 6136 CPU @ 3.00GHz,
- Mellanox ConnectX5 VPI cards,
- Mellanox SX6036 VPI switch (40Gbit/s Ethernet and 56Gbit/s InfiniBand),

Kubernetes setup

- kubeadm base install with flannel networking, plus
  - multus-cni
  - sriov-cni
  - sriov-network-device-plugin

- post-install configuration:
  - enabling hugepages
  - enabling SRIOV
  - creating kubernetes resources
    - configmap
    - daemonset
    - network

ConfigMap specifies the details of SRIOV capable NIC

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
 name: sriovdp-config
 namespace: kube-system
data:
 config.json: |
   {
       "resourceList": [{
               "resourceName": "mlnx_sriov_rdma",
               "isRdma": true,
               "selectors": {
                   "vendors": ["15b3"],
                   "devices": ["1018"],
                   "pfNames": ["p4p1"]
               }
(...)
```

K8s-RDMA performance benchmark

```
[root@kube02 ~]# kubectl create -f yaml/pod1.yaml
pod/pod1.yaml  created
[root@kube02 ~]# kubectl create -f yaml/pod2.yaml
pod/pod2.yaml  created
[root@kube02 ~]# kubectl create -f yaml/pod3.yaml
pod/pod3.yaml  created

[root@kube02 ~]# kubectl get pods -o wide
NAME                            READY   STATUS     RESTARTS   AGE    IP            NODE
testpod1                        1/1     Running    0          5d4h    10.244.1.16   kube01
testpod2                        1/1     Running    0          5d4h    10.244.0.25   kube02
testpod3                        1/1     Running    0          5d4h   10.244.1.17   kube01
```

K8s-RDMA performance benchmark

```
[root@kube01 ~]# docker exec -it 2f23bd37c78d  ib_send_bw -d mlx5_4 -i 1 -x 0
************************************
* Waiting for client to connect... *
************************************
[root@kube02 ~]# docker exec -it c1bab072f584 ib_send_bw -F -d mlx5_6 -i 1  10.244.1.16 -x 0
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : mlx5_6
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
(...)
---------------------------------------------------------------------------------------
 local address: LID 0000 QPN 0x06ad PSN 0x93a32c
 GID: 254:128:00:00:00:00:00:00:76:108:46:255:254:55:185:117
 remote address: LID 0000 QPN 0x02af PSN 0xf66cdf
 GID: 254:128:00:00:00:00:00:00:40:128:192:255:254:20:95:137
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      1000             4362.56            4362.49              0.069800
---------------------------------------------------------------------------------------
[root@kube02 ~]#
```

K8s-RDMA vs bare-metal

```
(Kubernetes)
[root@kube02 ~]# docker exec -it c1bab072f584 ib_send_bw -F -d mlx5_6 -i 1 10.244.1.16 -x 0
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : mlx5_6
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
(...)
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      1000             4362.56               4362.49             0.069800
---------------------------------------------------------------------------------------
(bare-metal)
[root@kube02 ~]# ib_send_bw -F 172.16.3.1 -x 0
---------------------------------------------------------------------------------------
                    Send BW Test
 Dual-port       : OFF          Device         : mlx5_0
 Number of qps   : 1        Transport type : IB
 Connection type : RC       Using SRQ      : OFF
---------------------------------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]   MsgRate[Mpps]
 65536      1000             4363.24               4363.17             0.069811
```

Conclusions

- SRIOV-based RDMA support in K8s works well
- Performance is very good and closely matches bare-metal
- It enables running applications requiring ultra-fast networking in K8s containers

Future Work

- InfiniBand support
- Integration with high-performance storage
- Support for GPUDirect

References

- Multus
- SR-IOV CNI
- SR-IOV Device Plugin
- Kubernetes CPU Manager
- Kubernetes Topology Manager

Thank you

Questions?