High Performance Network for Kubernetes

Using Chelsio Virtual Functions

Executive Summary

Containers are light-weight versions of Virtual Machines (VMs) that hold together all the components and functionalities needed for an application to run. Their small, flexible, decoupled nature allows them to be deployed easily across bare metal systems, public, private, and hybrid, cloud environments. Kubernetes, also known as K8s, is an open source system for provisioning and managing pods (group of containers) across multiple hosts. It integrates with networking, storage, security, telemetry, and other services to provide a comprehensive container infrastructure.

This paper describes how the Chelsio Adapters provide high-performance and low latency Ethernet networking for the containers in multi-host Kubernetes environment. Chelsio adapters support Single Root I/O Virtualization (SR-IOV), which segments a single network device into multiple Virtual Functions (VFs), to make them available for direct IO to the container. Chelsio T6 adapters support up to 64 VFs utilizing an embedded virtual switch.

Test Results

The below graph presents the throughput and %CPU results of Chelsio VFs attached to containers of different hosts. The results are collected using **Iperf2** tool with I/O size varying from 64B to 512 KBytes.

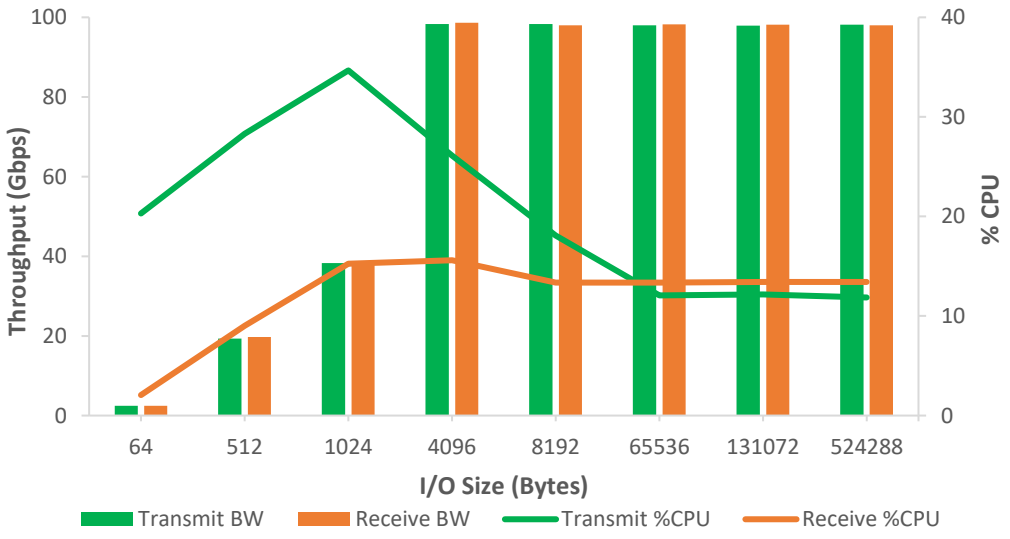

Figure 1 – VF Throughput and %CPU across different nodes vs. I/O size

The Chelsio VFs deliver line-rate 98 Gbps throughput in both transmit and receive directions for the containers. Only a maximum of 35% CPU is used on the host at lower I/O sizes.

Topology

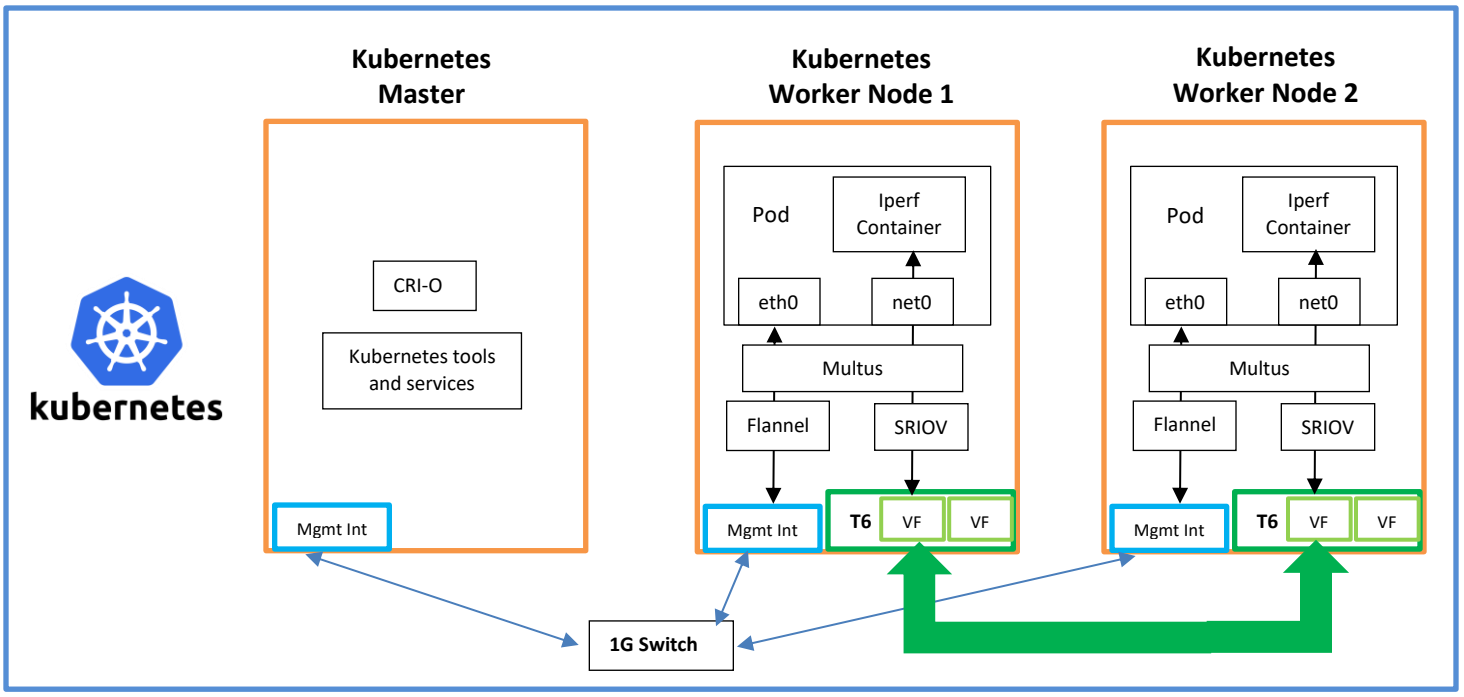

Figure 2 – Test Setup

The test setup consists of a 3 node cluster – 1 Master node and 2 Worker nodes, all connected using on-board interfaces to a 1G Switch for cluster management traffic. Kubernetes v1.30.2 is installed on each node.

- CRI-O is used as the container run time interface.
- Flannel plugin provides a Layer 3 IPv4 network between the nodes.
- Multus plugin is used to configure multiple networks to pods.
- SR-IOV CNI and SR-IOV Network Device Plugins are used to attach the Chelsio VFs to pod.
- Each Worker Node has 1 Pod and each Pod has 1 Iperf container.

Hardware Configuration

- **Master Node:** Configured with 1 Intel Xeon CPU E5-1660 v2 6-core @ 3.70GHz (HT disabled), 64GB of RAM and RHEL 9.2 OS (5.14.0-284.11.1.el9_2.x86_64 kernel).

- **Worker nodes:** Worker Node 1 is configured with 2 Intel Xeon CPUs E5-2687W v4 12-core @ 3.00GHz (HT disabled) and Worker Node 2 is configured with 2 Intel Xeon CPUs E5-2687W v3 10-core @ 3.10GHz (HT disabled) .

  Each worker node has 128GB of RAM, RHEL 9.2 OS (5.14.0-284.11.1.el9_2.x86_64 kernel) and Chelsio T62100-CR adapter. Single port on each worker node is connected directly using a 100G cable. The Chelsio Unified Wire driver v3.19.0.2 is installed. 1 VF is brought up on each worker node and assigned to pods. An MTU of 9000B is used on the ports under test.

Test Configuration

Common Configuration (all nodes)

i. Disable swap configurations.

```
[root@host~]# swapoff -a
```

ii. Install the traffic control utility package.

```
[root@host~]# yum install -y iproute-tc
```

iii. Enable overlay for networking.

```
[root@host~]# modprobe overlay
[root@host~]# modprobe br_netfilter
```

To configure the loading of the drivers permanently,

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
```

iv. Enable IP Forwarding and Network Settings.

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

[root@host~]# sysctl --system
```

v. Set selinux to permissive.

```
[root@host~]# setenforce 0
[root@host~]# sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config
```

vi. Install CRI-O.

```
PROJECT_PATH=prerelease:/main

cat <<EOF | tee /etc/yum.repos.d/cri-o.repo
[cri-o]
name=CRI-O
baseurl=https://pkgs.k8s.io/addons:/cri-o:/$PROJECT_PATH/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/addons:/cri-
o:/$PROJECT_PATH/rpm/repodata/repomd.xml.key
EOF

[root@host~]# dnf install -y cri-o
[root@host~]# systemctl enable --now crio.service
[root@host~]# systemctl status crio.service
```

vii. Install Kubernetes Tools.

```
KUBERNETES_VERSION=v1.30

cat <<EOF | tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/$KUBERNETES_VERSION/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/$KUBERNETES_VERSION/rpm/repodata/re
pomd.xml.key
EOF

[root@host~]# dnf install -y cri-o kubelet kubeadm kubectl
[root@host~]# systemctl enable --now kubelet.service
[root@host~]# systemctl status kubelet.service
```

Maser/Worker Node Configuration

i. Initialize Kubernetes on Master Node.

```
[root@master~]# kubeadm init --pod-network-cidr=10.244.0.0/16
[root@master~]# export KUBECONFIG=/etc/kubernetes/admin.conf
```

ii. Install CNI plugin.

```
[root@master~]# git clone https://github.com/containernetworking/plugins
[root@master~]# cd plugins
[root@master~]# git checkout v1.1.1
[root@master~]#./build_linux.sh
[root@master~]# mkdir -p /opt/cni/bin
[root@master~]# cp bin/* /opt/cni/bin/
```

iii. Create a default cluster network using Flannel plugin.

```
[root@master~]# kubectl apply -f
https://github.com/coreos/flannel/raw/master/Documentation/kube-flannel.yml
[root@master~]# kubectl get pods --all-namespaces
```

iv. Obtain Join Token from Master Node.

```
[root@master~]# kubeadm token create --print-join-command
```

v. Add worker node to the cluster (repeat for the other worker node in the cluster).

```
[root@worker~]# kubeadm join 10.193.184.224:6443 --token
6iij7y.wznrxxfb467329wf --discovery-token-ca-cert-hash
sha256:4452d8f94aba0f422ab1dd13db703cbab15da1533f1ea8c604f48eb4d7dffbbf
```

vi. Verify on the master node.

```
[root@master~]# kubectl get nodes -o wide
```

vii. Create VF(s) on the worker nodes.

```
[root@worker~]# modprobe -v cxgb4
[root@worker~]# ifconfig <pf4_P0> mtu 9000 up
```

```
[root@worker~]# echo 1 > /sys/bus/pci/devices/<Chelsio pf0>/sriov_numvfs
[root@worker~]# modprobe cxgb4vf
[root@worker~]# ifconfig <VF int> mtu 9000
[root@worker~]# lspci -s <VF PCI ID> -n
02:01.0 0200: 1425:680d
```

viii. Install SR-IOV CNI plugin.

```
[root@master~]# git clone https://github.com/k8snetworkplumbingwg/sriov-cni.git
[root@master~]# /root/sriov-cni/
[root@master~]# git checkout v2.8.0
[root@master~]# make
[root@master~]# cp build/sriov /opt/cni/bin/

[root@master~]# kubectl create -f images/sriov-cni-daemonset.yaml --v=5
```

ix. Install SR-IOV Network Device Plugin.

```
[root@master~]# git clone https://github.com/k8snetworkplumbingwg/sriov-
network-device-plugin.git
[root@master~]# cd /root/sriov-network-device-plugin/
[root@master~]# git checkout v3.7.0
```

Create a ConfigMap that defines SR-IOV resource pool configuration (Vendor and Device ID were obtained in step vii.)

```
# cat deployments/configMap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [{
                "resourceName": "chelsio_sriov_netdevice",
                "selectors": {
                    "vendors": ["1425"],
                    "devices": ["680d"],
                    "drivers": ["cxgb4vf"]
                }
            }]
    }

[root@master~]# kubectl create -f deployments/configMap.yaml --v=5
[root@master~]# kubectl create -f deployments/sriovdp-daemonset.yaml --v=5
```

x. Install Multus CNI, for enabling multiple network interfaces to pods/containers.

```
[root@master~]# kubectl apply -f
https://raw.githubusercontent.com/k8snetworkplumbingwg/multus-
cni/master/deployments/multus-daemonset-thick.yml
```

xi. Create the SR-IOV Network CRD with the below changes.

```
[root@master~]# cat deployments/sriov-crd.yaml
```

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: sriov-net1
  annotations:
    k8s.v1.cni.cncf.io/resourceName: intel.com/chelsio_sriov_netdevice
spec:
  config: '{
  "type": "sriov",
  "cniVersion": "0.3.1",
  "name": "sriov-network",
  "ipam": {
    "type": "host-local",
    "subnet": "102.100.2.0/24",
    "routes": [{
      "dst": "0.0.0.0/0"
    }],
    "gateway": "102.100.2.1"
}
}'

[root@master~]# kubectl create -f deployments/sriov-crd.yaml --v=5
```

xii. Rocky Linux 8.6 container (rockylinux:8.6) was downloaded and configured with Iperf v2.2.0.

xiii. Create a pod (testpod1) and initialize 1 container (iperf) which makes use of the defined SR-IOV resource.

```
[root@master~]# cat /root/testpod1_rocky8.6.yaml
apiVersion: v1
kind: Pod
metadata:
  name: testpod1
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-net1
spec:
  nodeName: worker1
  containers:
  - name: iperf
    image: localhost/rockylinux8.6
    imagePullPolicy: IfNotPresent
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        intel.com/chelsio_sriov_netdevice: '1'
      limits:
        intel.com/chelsio_sriov_netdevice: '1'

[root@master~]# kubectl create -f deployments/testpod1-rocky8.6.yaml --v=5
[root@master~]# kubectl describe pod/testpod1 -n default | tail
Events:
Type    Reason          Age   From     Message
----    ------          ----  ----     -------
Normal  AddedInterface  59m   multus   Add eth0 [10.244.1.9/24] from cbr0
Normal  AddedInterface  59m   multus   Add net1 [102.100.2.201/24] from
default/sriov-net1
Normal  Pulled          59m   kubelet  Container image
"localhost/rockylinux8.6" already present on machine
Normal  Created         59m   kubelet  Created container iperf
 Normal  Started         59m   kubelet  Started container iperf
```

Wait till container status shows as "Started". Look for IP address allocated to the VF interface (net1).

xiv. Create another pod (testpod2) and container (iperf) on different worker node in the cluster (using the configuration in step xiii). Wait till container status shows as "Started".

xv. Execute the below command to run iperf tests in the containers.

```
[root@master~]# kubectl exec testpod2 -- iperf -s -w 512K
[root@master~]# kubectl exec testpod1 -- iperf -c 102.100.2.209 -t 60 -w 512K
-l <iosize> -P 8
```

Conclusion

With line-rate 98 Gbps throughput and a maximum 35% host CPU utilization, Chelsio provides leading TCP/IP SR-IOV networking solutions for Kubernetes container environments. With lot of CPU resources left free on the host, more number of containers can be deployed. In addition to SR-IOV, Chelsio adapters support offloading multiple network, storage, and security protocols, and delivering industry-leading performance and efficiency. All the traffic runs over a single network, rather than building and maintaining multiple networks, resulting in significant acquisition and operational cost savings.

Related Links

100G OVS Kernel Datapath Offload for AMD EPYC
Windows d.VMMQ Performance
Windows TCP/IP SR-IOV in Virtual Environments