Validating DPDK performance on OpenShift

Updated Yesterday at 12:21 AM - English

TABLE OF CONTENTS

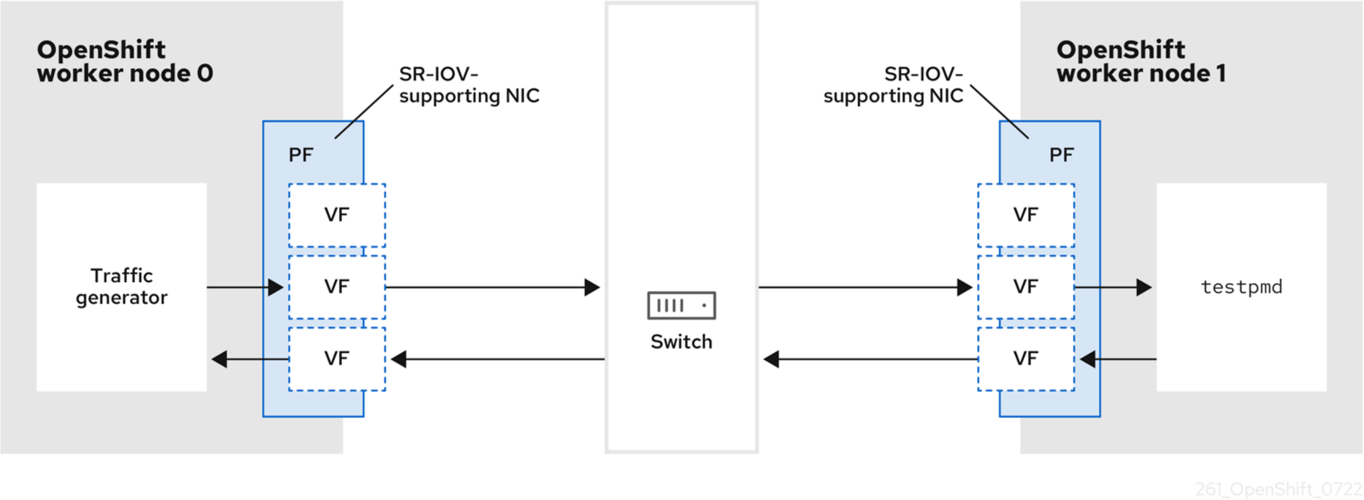
Validating DPDK performance on OpenShift

This article describes the build and the deployment of a traffic generating application inside a container. The traffic generator validates Data Plane Development Kit (DPDK) line rate performance on OpenShift Container Platform. It is used with the following network elements:
- Hardware networking

- Cloud elements
- Physical and virtual functions
- Test application

Traffic testing environment

The following diagram shows the components of a traffic-testing environment:



- **Traffic generator**: An application that can generate high-volume packet traffic.
- **SR-IOV-supporting NIC**: A network interface card compatible with Single Root I/O Virtualization. The card runs a number of virtual functions on a physical interface.
- **Physical Function (PF)**: A PCI Express (PCIe) function of a network adapter that supports the single root I/O virtualization (SR-IOV) interface.
- **Virtual Function (VF)**: A lightweight PCIe function on a network adapter that supports Single Root I/O virtualization (SR-IOV). The VF is associated with the PCIe Physical Function (PF) on the network adapter, and represents a virtualized instance of the network adapter.
- **Switch**: Network switch. Nodes can also be connected back-to-back.
- **`testpmd`** : An example application included with DPDK. The `testpmd` application can be used to test the DPDK in a packet forwarding mode. `testpmd` is also an example of how to build a fully-fledged application using the DPDK SDK.
- **worker 0** and **worker 1**: OpenShift Container Platform nodes.

Running the validation

For this stage of the development of the test environment, you must do the following:
1. Build the TRex container image
2. Deploy an OpenShift Container Platform cluster
3. Create TRex configuration from template
4. Define a TRex pod

Building the TRex container image

To build the TRex container image, run the following build script:

```
FROM quay.io/centos/centos:stream8

ARG TREX_VERSION=2.87
ENV TREX_VERSION ${TREX_VERSION}

# install requirements
RUN dnf install -y --nodocs git wget procps python3 vim python3-pip pciutils
gettext https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm &&
dnf clean all
RUN dnf install -y --nodocs hostname iproute net-tools ethtool nmap iputils perf
numactl sysstat htop rdma-core-devel libibverbs libibverbs-devel net-tools && dnf
clean all

# install trex server
WORKDIR /opt/
RUN wget --no-check-certificate https://trex-
tgn.cisco.com/trex/release/v${TREX_VERSION}.tar.gz && \
   tar -xzf v${TREX_VERSION}.tar.gz && \
   mv v${TREX_VERSION} trex && \
   rm v${TREX_VERSION}.tar.gz

WORKDIR /opt/trex
```

Deploying an OpenShift Container Platform cluster

You must deploy and configure both the Node Tuning Operator and the SR-IOV Network Operator. See the OpenShift Container Platform 4.10 Documentation for more information. See also Additional Resources at the end of this article.

Example performance profile

The following code block illustrates a typical performance profile:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
 name: performance
spec:
 globallyDisableIrqLoadBalancing: true
 cpu:
   isolated: 21-51,73-103
   reserved: 0-20,52-72
 hugepages:
   defaultHugepagesSize: 1G
   pages:
   - count: 32
   size: 1G
 numa:
   topologyPolicy: "single-numa-node"
 nodeSelector:
   node-role.kubernetes.io/worker-cnf: ""
```

Description

- `isolated` : Defines the isolated CPUs for guaranteed workloads.
- `defaultHugepagesSize` : Defines the default `hugepages` size: typically set to 1G.
- `topologyPolicy` : Defines the Topology policy. The policy should always allocate from a single NUMA. If that is not possible, block the pod deployment.

Example SR-IOV network policy

The following code block illustrates a typical SR-IOV network policy:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: dpdk-nic-1
 namespace: openshift-sriov-network-operator
spec:
 deviceType: vfio-pci
 needVhostNet: true
 nicSelector:
   pfNames: ["ens3f0"]
 nodeSelector:
   node-role.kubernetes.io/worker-cnf: ""
 numVfs: 5
 priority: 99
 resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: dpdk-nic-2
 namespace: openshift-sriov-network-operator
spec:
 deviceType: vfio-pci
 nicSelector:
   pfNames: ["ens3f1"]
 nodeSelector:
   node-role.kubernetes.io/worker-cnf: ""
 numVfs: 5
 priority: 99
 resourceName: dpdk_nic_2
```

Note: for Mellanix, use `deviceType: netdevice` and `isRdma: True`

Example SR-IOV network

The following code block illustrates a typical SR-IOV network:

```
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: dpdk-network-1-vlan
 namespace: openshift-sriov-network-operator
spec:
 ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.1.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'
 networkNamespace: seba
 spoofChk: "on"
 trust: "on"
 vlan: 2004
 resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: dpdk-network-2
 namespace: openshift-sriov-network-operator
spec:
 ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.2.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-2"}'
 networkNamespace: seba
 spoofChk: "on"
 trust: "on"
 resourceName: dpdk_nic_2
```

Note: Here, we are using `vlan` tag for the VFs as an example. This is not mandatory

Defining a TRex pod

The following `yaml` file defines a TRex pod:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: dpdk
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: trex-info-for-config
  namespace: dpdk
data:
  PORT_BANDWIDTH_GB: "25"
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: trex-config-template
data:
  trex_cfg.yaml : |
    - port_limit: 2
      version: 2
      interfaces:
        - "${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1}"
        - "${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2}"
      port_bandwidth_gb: ${PORT_BANDWIDTH_GB}
      port_info:
        - ip: 10.10.10.2
          default_gw: 10.10.10.1
        - ip: 10.10.20.2
          default_gw: 10.10.20.1
      platform:
        master_thread_id: $MASTER
        latency_thread_id: $LATENCY
        dual_if:
          - socket: ${SOCKET}
            threads: [${CPU}]
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: trex-tests
  namespace: dpdk
data:
  testpmd.py : |
    from trex_stl_lib.api import *

    from testpmd_addr import *

    # Wild local MACs
    mac_localport0='50:00:00:00:00:01'
    mac_localport1='50:00:00:00:00:02'

    class STLS1(object):

        def __init__ (self):
            self.fsize  =64; # the size of the packet
            self.number = 0

        def create_stream (self, direction = 0):

            size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
            dport = 1026 + self.number
            self.number = self.number + 1
            if direction == 0:
                base_pkt =
Ether(dst=mac_telco0,src=mac_localport0)/IP(src="16.0.0.1",dst=ip_telco0)/UDP(dport
=15,sport=1026)
            else:
                base_pkt =
Ether(dst=mac_telco1,src=mac_localport1)/IP(src="16.1.0.1",dst=ip_telco1)/UDP(dport
=16,sport=1026)
            #pad = max(0, size - len(base_pkt)) * 'x'
            pad = (60 - len(base_pkt)) * 'x'

            return STLStream(
                packet =
                STLPktBuilder(
                    pkt = base_pkt / pad
                ),
                mode = STLTXCont())

        def create_stats_stream (self, rate_pps = 1000, pgid = 7, direction = 0):

            size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
            if direction == 0:
                base_pkt =
Ether(dst=mac_telco0,src=mac_localport0)/IP(src="17.0.0.1",dst=ip_telco0)/UDP(dport
=dport,sport=1026)
            else:
                base_pkt =
Ether(dst=mac_telco1,src=mac_localport1)/IP(src="17.1.0.1",dst=ip_telco1)/UDP(dport
=dport,sport=1026)
            pad = max(0, size - len(base_pkt)) * 'x'

            return STLStream(
                packet =
                STLPktBuilder(
                    pkt = base_pkt / pad
                ),
                mode = STLTXCont(pps = rate_pps),
                flow_stats = STLFlowLatencyStats(pg_id = pgid))
            #flow_stats = STLFlowStats(pg_id = pgid))

        def get_streams (self, direction = 0, **kwargs):
            # create multiple streams, one stream per core...
            s = []
            for i in range(14):
                s.append(self.create_stream(direction = direction))
            #if direction == 0:
            #    s.append(self.create_stats_stream(rate_pps=1000, pgid=10,
direction = direction))
            #else:
            #    s.append(self.create_stats_stream(rate_pps=1000, pgid=11,
direction = direction))

            return s

    # dynamic load - used for trex console or simulator
    def register():
        return STLS1()

    testpmd_addr.py: |
    # wild second XL710 mac
    mac_telco0 = '60:00:00:00:00:01'
    # we don’t care of the IP in this phase
    ip_telco0  = '10.0.0.1'
    # wild first XL710 mac
    mac_telco1 = '60:00:00:00:00:02'
    ip_telco1 = '10.1.1.1'
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
       "name": "dpdk-network-1-vlan",
       "mac": "50:00:00:00:00:01",
       "namespace": "dpdk"
      },
      {
       "name": "dpdk-network-2",
       "mac": "50:00:00:00:00:02",
       "namespace": "dpdk"
      }
    ]'
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
    irq-load-balancing.crio.io: "disable"
  labels:
    app: trex
  name: trex
  namespace: dpdk
spec:
  runtimeClassName: performance-performance
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - dpdk
          topologyKey: kubernetes.io/hostname
  containers:
    - command:
        - /bin/bash
        - -c
        - sleep INF
      image: <TREX-IMAGE>
      imagePullPolicy: Always
      name: trex
      envFrom:
        - configMapRef:
            name: trex-info-for-config
      resources:
        limits:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 1Gi
        requests:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 1Gi
      securityContext:

        capabilities:
          add:
            - IPC_LOCK
            - SYS_RESOURCE
            - NET_RAW
            - NET_ADMIN
        runAsUser: 0
      volumeMounts:
        - name: trex-config-template
          mountPath: /opt/templates/
        - name: trex-tests
          mountPath: /opt/tests/
        - mountPath: /mnt/huge
          name: hugepages
        - name: modules
          mountPath: /lib/modules
  terminationGracePeriodSeconds: 5
  volumes:
    - name: modules
      hostPath:
        path: /lib/modules
    - configMap:
        name: trex-info-for-config
      name: trex-info-for-config
    - name: trex-config-template
      configMap:
        name: trex-config-template
    - name: trex-tests
      configMap:
        name: trex-tests
    - emptyDir:
        medium: HugePages
      name: hugepages
```

Description

`-` `PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1` : The PCI deviceID for the first network.
`-` `PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2` : The PCI deviceID for the second network.
`-` `Trex-config-template` : The template configuration. Create this before starting the `trex` process.
`-` `TREX-IMAGE` : the TRex image built before the deployment

Create TRex configuration from template

Access the TRex pod with `oc -n dpdk rsh trex` the copy the template file `cp /opt/templates/trex_cfg.yaml /etc/trex_cfg.yaml`
and edit the file using the following parameters

```
port_limit: 2
 version: 2
 interfaces: ["{PCIDEVICE_1}","{PCIDEVICE_2}"]
 port_bandwidth_gb: {PORT_BANDWIDTH_GB}
 port_info:
   - ip: 10.10.10.2
     default_gw: 10.10.10.1
   - ip: 10.10.20.2
     default_gw: 10.10.20.1
 platform:
   master_thread_id: {MASTER}
   latency_thread_id: {LATENCY}
   dual_if:
     - socket: {SOCKET}
       threads: [{CPUS}]
```

replace the parameters with the following configuration

PCIDEVICE_1 : check the PCI address that was allocated for the trex pod running `env | grep PCIDEVICE_OPENSHIFT_IO`

PCIDEVICE_2 : check the PCI address that was allocated for the trex pod running `env | grep PCIDEVICE_OPENSHIFT_IO`

PORT_BANDWIDTH_GB: the port speed in GB (example: 25)

MASTER: select the master/UI CPU. check the available CPUs for the pod using `cat /sys/fs/cgroup/cpuset/cpuset.cpus` . Check also for sibling CPUs example for cpu0 (cat /sys/devices/system/cpu/cpu0/topology/core_cpus_list)

LATENCY: select the latency measurement CPU (better to use the sibling from the MASTER CPU selected

SOCKET: select the Numa where all the allocated CPUs are (you can check using the lspcu | grep

CPUS: add all the CPUs from `cat /sys/fs/cgroup/cpuset/cpuset.cpus` excluding the ones used by MASTER and LATENCY

Output Example:

```
- port_limit: 2
  version: 2
  interfaces:
    - "0000:d8:02.1"
    - "0000:d8:0a.3"
  port_bandwidth_gb: 25
  port_info:
    - ip: 10.10.10.2
      default_gw: 10.10.10.1
    - ip: 10.10.20.2
      default_gw: 10.10.20.1
  platform:
    master_thread_id: 21
    latency_thread_id: 23
    dual_if:
      - socket: 1
        threads: [25,27,29,31,33,35,73,75,77,79,81,83,85,87]
```

Then you can lunch TRex with the following command:

```
./t-rex-64 --no-ofed-check --no-hw-flow-stat -i -c 14
```

Test configuration

The test is injected into the container using a `configmap` . Using this method, you do not need to recreate the pod if you change the code. Apply the `configmap` again and `k8s` will mount the new files.
The following is an example `testpmd.py` :

```
from trex_stl_lib.api import *

   from testpmd_addr import *

   # Wild local MACs
   mac_localport0='50:00:00:00:00:01'
   mac_localport1='50:00:00:00:00:02'

   class STLS1(object):

  def __init__ (self):
  self.fsize  =64; # the size of the packet
  self.number = 0

  def create_stream (self, direction = 0):

  size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
  dport = 1026 + self.number
  self.number = self.number + 1
  if direction == 0:
base_pkt =
Ether(dst=mac_telco0,src=mac_localport0)/IP(src="16.0.0.1",dst=ip_telco0)/UDP(dport
=15,sport=1026)
  else:
base_pkt =
Ether(dst=mac_telco1,src=mac_localport1)/IP(src="16.1.0.1",dst=ip_telco1)/UDP(dport
=16,sport=1026)
  #pad = max(0, size - len(base_pkt)) * 'x'
  pad = (60 - len(base_pkt)) * 'x'

  return STLStream(
  packet =
  STLPktBuilder(
  pkt = base_pkt / pad
  ),
   mode = STLTXCont())

  def create_stats_stream (self, rate_pps = 1000, pgid = 7, direction = 0):

  size = self.fsize - 4; # HW will add 4 bytes ethernet FCS
  if direction == 0:
base_pkt =
Ether(dst=mac_telco0,src=mac_localport0)/IP(src="17.0.0.1",dst=ip_telco0)/UDP(dport
=dport,sport=1026)
  else:
base_pkt =
Ether(dst=mac_telco1,src=mac_localport1)/IP(src="17.1.0.1",dst=ip_telco1)/UDP(dport
=dport,sport=1026)
  pad = max(0, size - len(base_pkt)) * 'x'

  return STLStream(
  packet =
  STLPktBuilder(
  pkt = base_pkt / pad
  ),
   mode = STLTXCont(pps = rate_pps),
   flow_stats = STLFlowLatencyStats(pg_id = pgid))

  def get_streams (self, direction = 0, **kwargs):
  # create multiple streams, one stream per core...
  s = []
  for i in range(14):
   s.append(self.create_stream(direction = direction))
  return s

   # dynamic load - used for trex console or simulator
   def register():
  return STLS1()
```

The following is an example `testpmd_addr.py` :

```
# wild second XL710 mac
   mac_telco0 = '60:00:00:00:00:01'
   # we don't care of the IP in this phase
   ip_telco0  = '10.0.0.1'
   # wild first XL710 mac
   mac_telco1 = '60:00:00:00:00:02'
   ip_telco1 = '10.1.1.1'
```

Commands for TRex

From a new terminal access the TRex pod `oc -n dpdk rsh trex` , and start the connection to TRex. You can then load the traffic generator stream.

```
./trex-console
> tui
> # start -f /opt/tests/testpmd.py -m <number-of-packets> -p <ports to use>
> start -f /opt/tests/testpmd.py -m 24mpps -p 0
> stop -a
```

This code example runs the traffic generator, sending 24 million packets per second using port 0 and receiving from port 1:

Example output

```
Global Statistitcs

connection   : localhost, Port 4501   total_tx_L2  : 12.76 Gbps
version  : STL @ v2.87total_tx_L1  : 16.74 Gbps
cpu_util.: 29.65% @ 14 cores (14 per dual port)   total_rx : 12.09 Gbps
rx_cpu_util. : 0.0% / 0 pps   total_pps: 24.91 Mpps
async_util.  : 0.02% / 10.48 Kbps drop_rate: 0 bps
total_cps.   : 0 cps  queue_full   : 0 pkts

Port Statistics

   port| 0 | 1 |   total
-----------+-------------------+------------------+-----------------
owner  |  root |  root |
link   |UP |UP |
state  |  IDLE |  TRANSMITTING |
speed  |   25 Gb/s |   25 Gb/s |
CPU util.  |  0.0% |29.65% |
-- |   |   |
Tx bps L2  | 0 bps |12.76 Gbps |12.76 Gbps
Tx bps L1  | 0 bps |16.74 Gbps |16.74 Gbps
Tx pps | 0 pps |24.91 Mpps |24.91 Mpps
Line Util. |   0 % |   66.97 % |
---|   |   |
Rx bps |12.09 Gbps | 0 bps |12.09 Gbps
Rx pps |20.99 Mpps | 0 pps |20.99 Mpps
----   |   |   |
opackets   |2654147973 |1223919476 |3878067449
ipackets   |1030488553 |2012583907 |3043072460
obytes |  180482062450 |   78330844946 |  258812907396
ibytes |   74195178964 |  136855711844 |  211050890808
tx-pkts|2.65 Gpkts |1.22 Gpkts |3.88 Gpkts
rx-pkts|1.03 Gpkts |2.01 Gpkts |3.04 Gpkts
tx-bytes   | 180.48 GB |  78.33 GB | 258.81 GB
rx-bytes   |   74.2 GB | 136.86 GB | 211.05 GB
-----  |   |   |
oerrors| 0 | 0 | 0
ierrors| 0 | 0 | 0
```

On the other side of the traffic testing environment is a machine that can process the packets and send them back to the TRex machine on port 1. Possible usages are `testpmd`, `vpp` and `ovs-dpdk`.

Additional resources

- About the Performance Profile Creator
- Adjusting the NIC queues with the performance profile

- Provisioning a worker with real-time capabilities
- Installing the SR-IOV Network Operator
- SR-IOV network node configuration object
- Dynamic IP address assignment configuration with Whereabouts

SBR **Networking** **Product(s)** **Red Hat OpenShift Container Platform**

Category **Developer** **Article Type** **General**

Was this helpful?

YES NO

People who viewed this article also viewed

How to troubleshoot OVS DPDK performance issue

Solution - May 18, 2024

How to troubleshoot OVS DPDK performance issue

Solution - Jun 14, 2024

OpenShift on OpenStack with DPDK

Solution - Jun 14, 2024

Comments

RED HAT

NEWBIE

Add comment

Formatting Help

☑ **Send notifications to content followers**

☐ **Mark comment as private**

Submit

RED HAT

ACTIVE CONTRIB...

148 Points

Private Comment May 22, 2023 1:27 PM

Anil Dhingra

was not able to start Trex - so started with --no-scapy-server

```
sh-4.4# ./t-rex-64 --no-ofed-check --no-hw-flow-stat -i -c 3 --no-scapy-server
```

Try troubleshoot why scapy not starting - assuming without this option may not be able to run .py profiles

Reply Privately

RED HAT

ACTIVE CONTRIB...

148 Points

Private Comment May 25, 2023 7:30 AM

Anil Dhingra

Worth to highlight why below parameter is required , users are facing packet drop and multiple performance related cases from partners/vendors cpu-quota.crio.io: "disable" runtimeClassName: performance-performance

https://issues.redhat.com/browse/RFE-3758

↩ Reply Privately

Quick Links

Help

Site Info

Related Sites

About

Red Hat Subscription Value

About Red Hat

Red Hat Jobs