DPDK in Containers

Hands-on Lab

Clayne Robison, Intel Corporation

Agenda

- Executive Summary
- DPDK and Containers Intro
- Hands-on Lab
- Conclusion

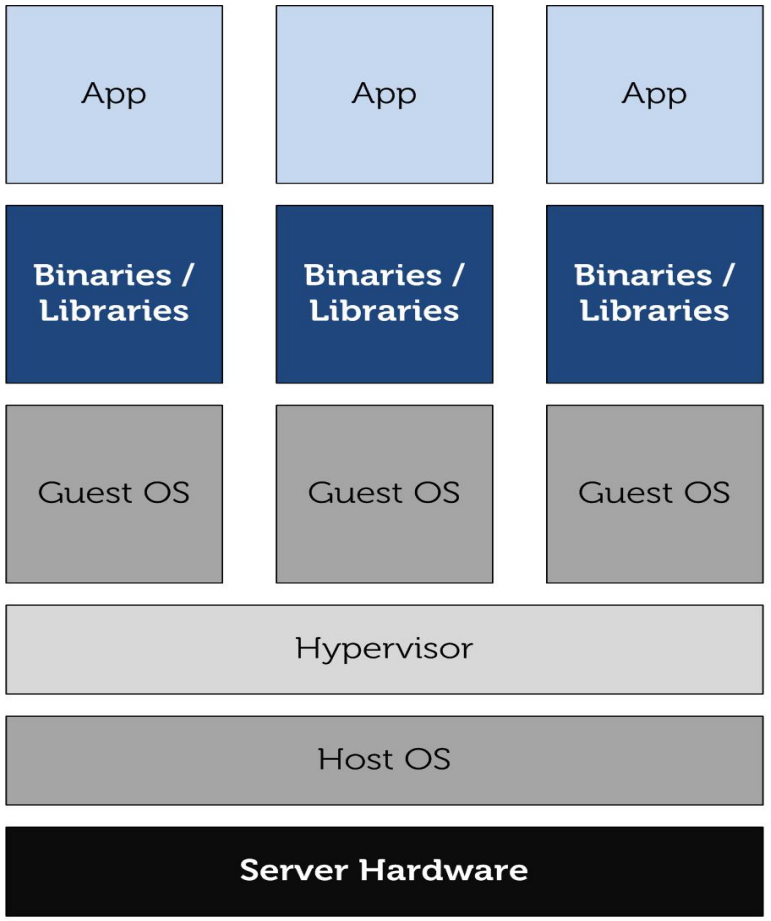
Summary

- Linux* containers use fewer system resources
  - More micro-services per host
  - No VM overhead
- Containers still use kernel network stack
  - Not ideal for SDN/NFV usages
- DPDK (and Open vSwitch*) can be used in Containers
- Elevated privileges required today

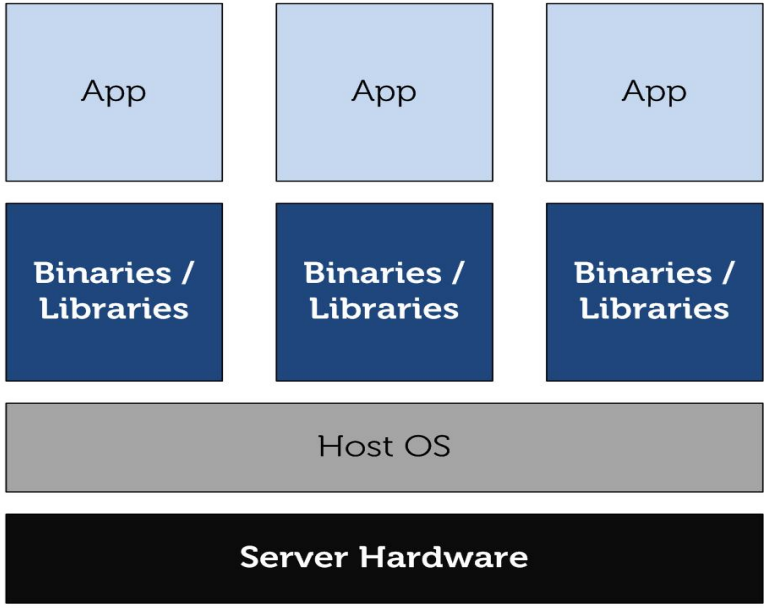
Agenda

- Executive Summary
- DPDK and Containers Intro
- Hands-on Lab
- Conclusion

Containers vs. VMs

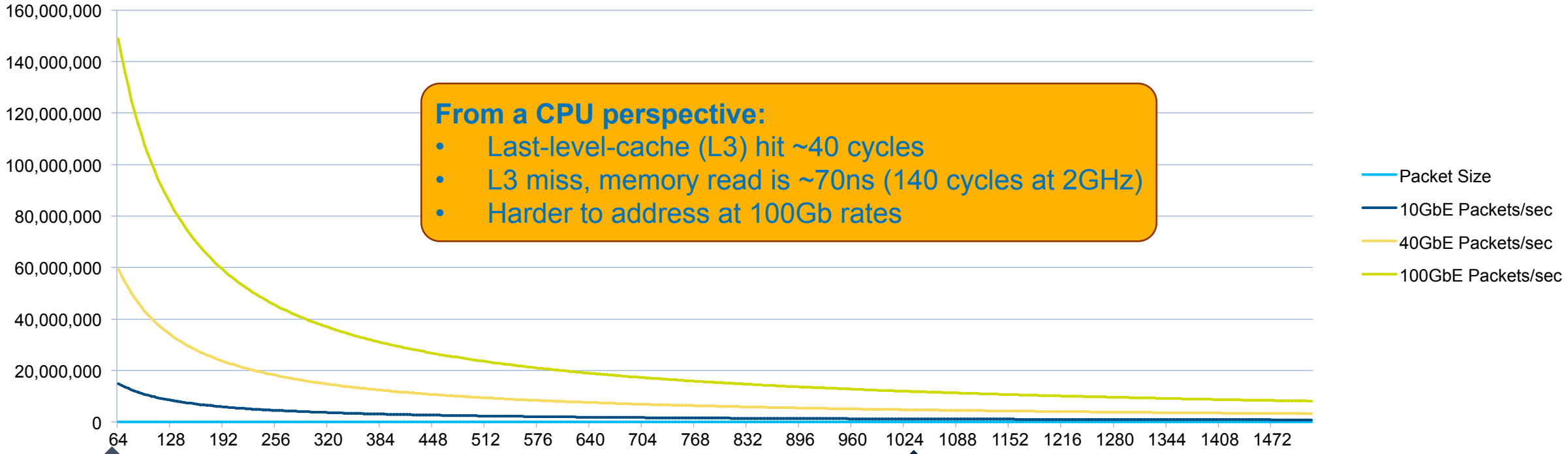


Virtualization



Containers

The DPDK Problem Statement



From a CPU perspective:

- Last-level-cache (L3) hit ~40 cycles
- L3 miss, memory read is ~70ns (140 cycles at 2GHz)
- Harder to address at 100Gb rates

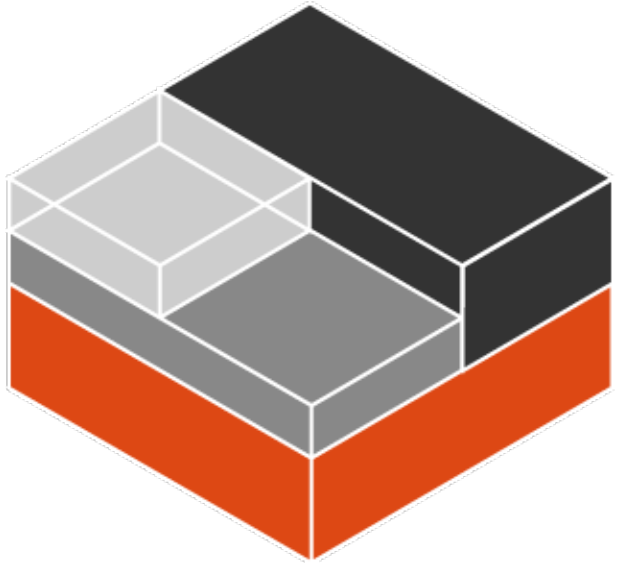
Packet Size	64 Bytes
40G packets/second	59.5 million each way
Packet arrival interval	16.8 ns
2 GHz clock cycles/packet	33 cycles

Typical Network Infrastructure Packet Size

Packet Size	1024 Bytes
40G packets/second	4.8 million each way
Packet arrival interval	208.8 ns
2 GHz clock cycles/packet	417 cycles

Typical Server Packet Size

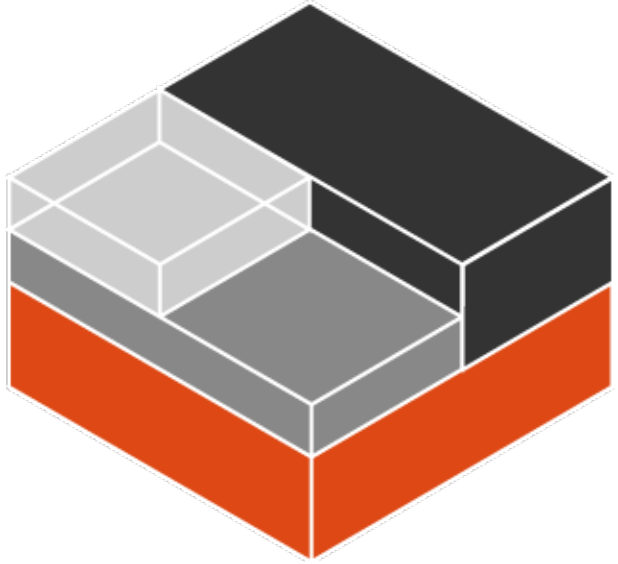
Assumptions



Assumptions

rkt

lxc

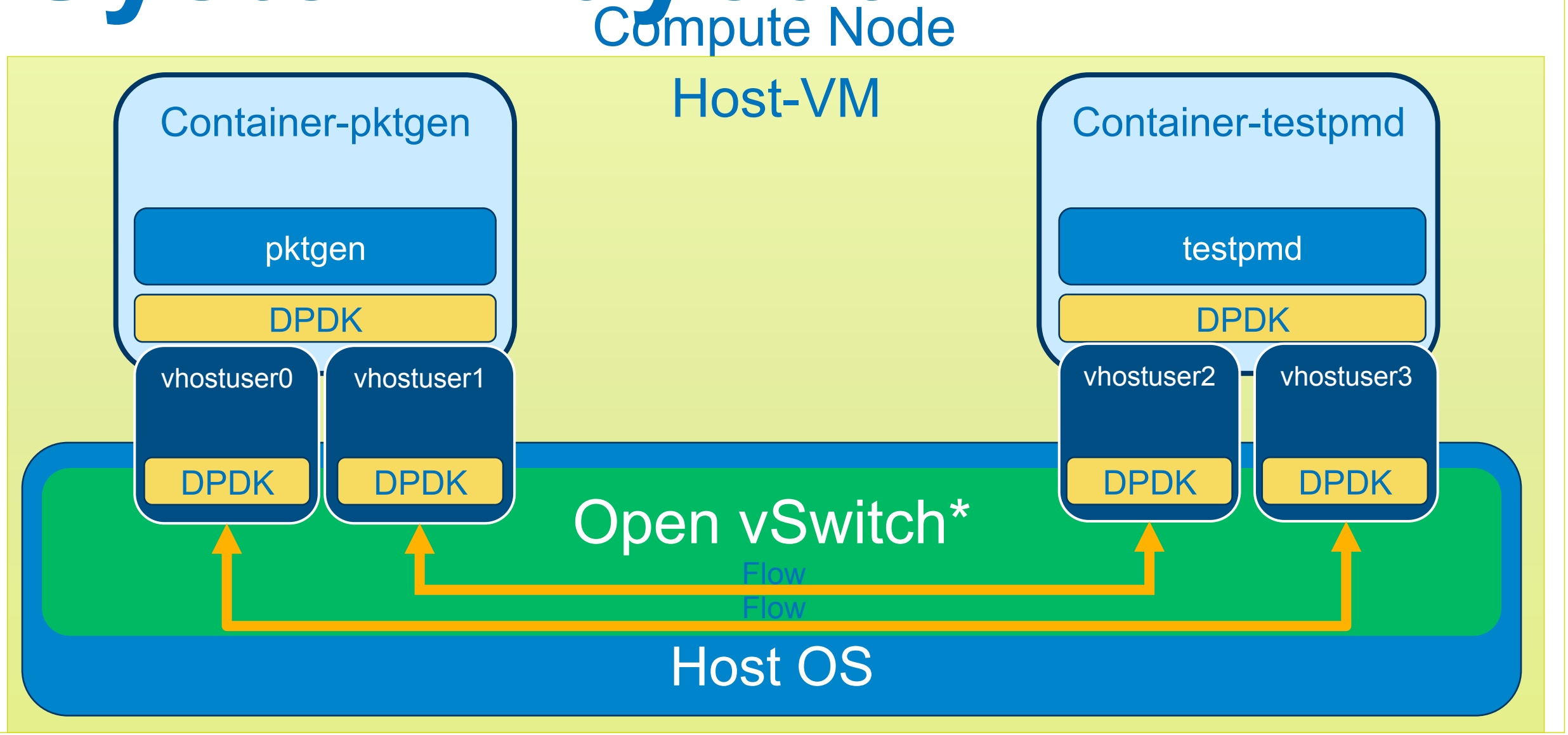


kubernetes

Agenda

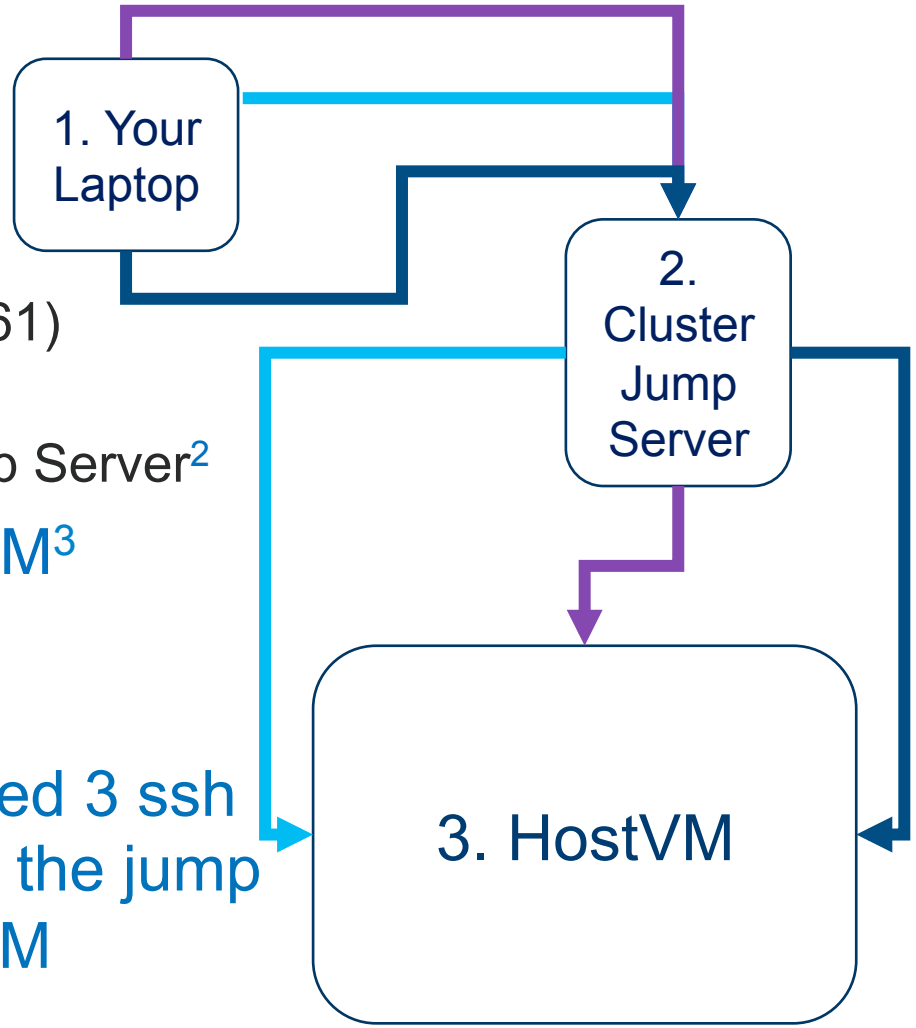
- Executive Summary
- DPDK and Containers Intro
- Hands-on Lab
- Conclusion

System Layout



Enter Lab Environment

- SSH from your laptop[1] in to Cluster Jump Server[2]
  - IP Address: 207.108.8.161
  - SSH v2 preferred
  - Username: student<1-50> ($ ssh student9@207.108.8.161)
  - Password: same as username (e.g. student9)
  - Repeat so that you have multiple connections to the Jump Server[2]
- SSH from Cluster Jump Server[2] in to assigned HostVM[3]
  - $ ssh user@HostVM-____
  - Username: user; Password: password
- Enter lab environment
  - $ cd ~/training/dpdk-container-lab
  - $ sudo su
  - # source setenv.sh



Note: You need 3 ssh sessions into the jump server/HostVM

Lab Slide Key

```
# cd $DPDK_DIR

(build the x86_64-native-linuxapp-gcc flavor of DPDK and
put it in the x86_64-native-linuxapp-gcc dir)

# make  config T=$DPDK_BUILD O=$DPDK_BUILD
# cd $DPDK_BUILD
# make -j
```

Manual Entry Box:
Type this code on the command line, line by line

```
# Short cuts  TRAINING_DIR/
00_build_dpdk.sh
```

Remember
TRAINING_DIR =
/home/user/training/dpdk-container-lab

Bash Script Call-out:
The file in this callout contains the same code as in the Manual Entry Box. Copy/ Paste line by line onto the command line, or simply run the entire script.

Build DPDK 16.11

```
# cd $DPDK_DIR

(build the x86_64-native-linuxapp-gcc flavor of DPDK
and put it in the x86_64-native-linuxapp-gcc dir)

# make  config T=$DPDK_BUILD O=$DPDK_BUILD
# cd $DPDK_BUILD
# make
```

Or

```
# cd $TRAINING_DIR
# ./00_build_dpdk.sh
```

Build Open vSwitch* 2.6.1

```
# cd $OVS_DIR
(run the autoconf magic)
# ./boot.sh
(build OVS with DPDK support)
# CFLAGS='-march=native' ./configure \
    --with-dpdk=$DPDK_DIR/$DPDK_BUILD
# make
```

Or

```
# cd $TRAINING_DIR
# ./01_build_ovs.sh
```

Prepare to Start Open vSwitch*

```
(create openvswitch directories)
# mkdir -p /usr/local/etc/openvswitch
# mkdir -p /usr/local/var/run/openvswitch

(mount the hugepage tlbfs)
# mount -t hugetlbfs -o pagesize=1G none /mnt/huge

(show the fs table)
# mount | grep -i “/mnt/huge”

(insert the user-space IO driver into the kernel)
# modprobe uio
# insmod $DPDK_DIR/$DPDK_BUILD/kmod/igb_uio.ko
```

Or

```
# cd $TRAINING_DIR
# ./02_prep_ovs.sh
```

Start Open vSwitch*

Or
cd $TRAINING_DIR
./03_start_ovs.sh

```
# cd $OVS_DIR
(initialize new OVS database)
# ./ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db \          ./
vswitchd/vswitch.ovsschema

(start database server)
# ./ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
       --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
       --pidfile –detach

(initialize OVS database)
# ./utilities/ovs-vsctl --no-wait init

(configure OVS DPDK using 1GB and the ovswitchd thread on logical core 1)
# ./utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true \
       other_config:dpdk-lcore-mask=0x2 other_config:dpdk-socket-mem="1024”

(start OVS)
# ./vswitchd/ovs-vswitchd unix:/usr/local/var/run/openvswitch/db.sock \
       --pidfile --detach
```

Create the Open vSwitch* Bridge and Ports

```
$ cd $OVS_DIR

(Tell OVS to use Core 2 for the PMD)
# ./utilities/ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x4

(Create bridge br0 and vhost ports that use DPDK)
# ./utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
# ./utilities/ovs-vsctl add-port br0 vhost-user0 \
      -- set Interface vhost-user0 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user1 \
      -- set Interface vhost-user1 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user2 \
      -- set Interface vhost-user2 type=dpdkvhostuser
# ./utilities/ovs-vsctl add-port br0 vhost-user3 \
      -- set Interface vhost-user3 type=dpdkvhostuser

(Show br0 info)
# ./utilities/ovs-vsctl show
```

Or

```
# cd $TRAINING_DIR
# ./04_createports.sh
```

Note port names. You'll see them in a moment

Add Routes/Flows to Open vSwitch*

```
(Clear clear current flows)
#./utilities/ovs-ofctl del-flows br0

(Add bi-directional flow between port 2 and 3 -- vhost-user1 and vhost-user2)
# ./utilities/ovs-ofctl add-flow br0 \
      in_port=2,dl_type=0x800,idle_timeout=0,action=output:3
# ./utilities/ovs-ofctl add-flow br0 \
      in_port=3,dl_type=0x800,idle_timeout=0,action=output:2

(Add bi-directional flow between port 1 and 4 -- vhost-user0 and vhost-user3)
# ./utilities/ovs-ofctl add-flow br0 \
      in_port=1,dl_type=0x800,idle_timeout=0,action=output:4
# ./utilities/ovs-ofctl add-flow br0 \
      in_port=4,dl_type=0x800,idle_timeout=0,action=output:1

(Show the current flow configuration)
# ./utilities/ovs-ofctl show br0
```

Or

```
# cd $TRAINING_DIR
# ./05_addroutes.sh
```

Note the mapping between Open vSwitch and OpenFlow ports.

Create testpmd Docker* Container
(Already Done)

```
$ cat $TRAINING_DIR/docker-build/testpmd/Dockerfile
FROM ubuntu
COPY ./dpdk-container-lab /root/dpdk-container-lab
WORKDIR /root/dpdk-container-lab
COPY ./dpdk /usr/src/dpdk
RUN apt-get update && apt-get install -y build-essential automake python-pip \
      libcap-ng-dev gawk pciutils linux-headers-$(uname -a | awk '{print $3}') \
      vim kmod
RUN pip install -U pip six
ENV DPDK_DIR "/usr/src/dpdk”
ENV DPDK_BUILD "x86_64-native-linuxapp-gcc”
ENV RTE_SDK "/usr/src/dpdk”
ENV RTE_TARGET "x86_64-native-linuxapp-gcc”
ENV TRAINING_DIR /root/dpdk-container-lab
RUN ./build_dpdk.sh
RUN ./build_testpmd.sh
CMD ["/bin/bash"]
```

Create testpmd Docker* Container (Con't)
(Already Done--DO NOT RUN in Lab)

```
$ cat $TRAINING_DIR//build_testpmd_container.sh
#!/bin/bash

DOCKER_BUILD_DIR="$(pwd)/docker-build/testpmd"

DOCKER_TAG="ses2017/testpmd1"

cd $DOCKER_BUILD_DIR

docker build . -t $DOCKER_TAG
```

Create pktgen Docker* Container
(Already Done)

```
$ cat $TRAINING_DIR/docker-build/pktgen/Dockerfile
FROM ses2017/testpmd
COPY ./dpdk-container-lab /root/dpdk-container-lab
WORKDIR /root/dpdk-container-lab
COPY ./dpdk /usr/src/dpdk
COPY ./pktgen /usr/src/pktgen
RUN apt-get update && apt-get install -y build-essential automake python-pip \
      libcap-ng-dev gawk pciutils linux-headers-$(uname -a | awk '{print $3}') \
      vim kmod libpcap-dev
RUN pip install -U pip six
ENV DPDK_DIR "/usr/src/dpdk”
ENV DPDK_BUILD "x86_64-native-linuxapp-gcc”
ENV RTE_SDK "/usr/src/dpdk”
ENV RTE_TARGET "x86_64-native-linuxapp-gcc”
ENV PKTGEN_DIR "/usr/src/pktgen”
ENV TRAINING_DIR /root/dpdk-container-lab
RUN ./build_dpdk.sh
RUN ./build_pktgen.sh
CMD ["/bin/bash"]
```

Create pktgen Docker* Container (Con't)
(Already Done--DO NOT RUN in Lab)

```
$ cat $TRAINING_DIR/build_pktgen_container.sh
#!/bin/bash

DOCKER_BUILD_DIR="$(pwd)/docker-build/pktgen"

DOCKER_TAG="ses2017/pktgen1"

cd $DOCKER_BUILD_DIR

docker build . -t $DOCKER_TAG
```

Run testpmd Docker* Container

```
# export DOCKER_TAG="ses2017/testpmd1”

(Launch docker container in privileged mode with access to host hugepages and
OVS DPDK sockets)
# docker run -ti --privileged \
       -v /mnt/huge:/mnt/huge \
       -v /usr/local/var/run/openvswitch:/var/run/openvswitch \
       $DOCKER_TAG
```

Or

```
# cd $TRAINING_DIR
# 06_start_testpmd_container.sh
```

Run pktgen Docker* Container

```
# export DOCKER_TAG="ses2017/pktgen1”

(Launch docker container in privileged mode with access to host hugepages and
OVS DPDK sockets)
# docker run -ti --privileged \
       -v /mnt/huge:/mnt/huge \
       -v /usr/local/var/run/openvswitch:/var/run/openvswitch \
       $DOCKER_TAG
```

Or

```
# cd $TRAINING_DIR
# 07_start_pktgen_container.sh
```

testpmd Container: Set dpdk parameters

Or
cd $TRAINING_DIR
./run_testpmd.sh

```
/******
* -c 0xE0: DPDK can run on core 5-7: (0b1110 0000)
* --master-lcore 5: master testpmd thread runs on core 5 (0b00100000)
* -n 1: we only have one memory bank in this VM
* --socket-mem 1024: use 1GB per socket
* --file-prefix testpmd: name appended to hugepage files from this process
* --no-pci don't look for any PCI devices
* --vdev=net_virtio_user2,mac=00:00:00:00:00:02,path=/var/run/openvswitch/vhost-user2
* --vdev=net_virtio_user3,mac=00:00:00:00:00:03,path=/var/run/openvswitch/vhost-user3
*      use a virtual device using the net_virtio_user driver, MAC address shown
*      and the path to the unix socket is /var/run/openvswitch/vhost-userX
*****/

# export DPDK_PARAMS="-c 0xE0 --master-lcore 5 -n 1 --socket-mem 1024 --file-prefix\
 testpmd --no-pci \
 --vdev=net_virtio_user2,mac=00:00:00:00:00:02,path=/var/run/openvswitch/vhost-user2 \
 --vdev=net_virtio_user3,mac=00:00:00:00:00:03,path=/var/run/openvswitch/vhost-user3"
```

testpmd Container: Set testpmd Parameters & Run testpmd

Or
cd $TRAINING_DIR
./run_testpmd.sh

```
/*******
* -i -- interactive mode
* --burst=64: we are going to fetch 64 packets at at time
* -txd=2048/--rxd=2048: we want 2048 descriptors in the rx and tx rings
* --forward-mode=io: forward all packets received
* --auto-start: start forwarding packets immediately on launch
*--disable-hw-vlan: disable hardware VLAN
* --coremask=0xC0: lock tespmd to run on cores 6-7 (0b1100 0000)
*****/

# export TESTPMD_PARAMS="--burst=64 -i --disable-hw-vlan --txd=2048 \
      --rxd=2048 --forward-mode=io --auto-start --coremask=0xC0"

(Use the DPDK_DIR, DPDK_PARAMS and TESPMD_PARAMS in the environment)
# $DPDK_DIR/app/test-pmd/testpmd $DPDK_PARAMS -- $TESTPMD_PARAMS
```

pktgen Container: Set dpdk parameters

Or
cd $TRAINING_DIR
./run_pktgen.sh

```
/*****
* -c 0x19: DPDK can run on core 0,3-4: (0b0001 1001)
* --master-lcore 3: make the pktgen dpdk thread run on core 3 (0b1000)
* -n 1: we only have one memory bank in this VM
* --socket-mem 1024: use 1GB per socket
* --file-prefix pktgen: name appended to hugepage files from this process
* --no-pci don't look for any PCI devices
* --vdev=net_virtio_user0,mac=00:00:00:00:00:00,path=/var/run/openvswitch/vhost-user0
* --vdev=net_virtio_user1,mac=00:00:00:00:00:01,path=/var/run/openvswitch/vhost-user1
*      use a virtual device using the net_virtio_user driver, MAC address shown
*      and the path to the unix socket is /var/run/openvswitch/vhost-userX
*****/

# export DPDK_PARAMS="-c 0x19 --master-lcore 3 -n 1 --socket-mem 1024 \
  --file-prefix pktgen --no-pci \
 --vdev=net_virtio_user0,mac=00:00:00:00:00:00,path=/var/run/openvswitch/vhost-user0 \
 --vdev=net_virtio_user1,mac=00:00:00:00:00:01,path=/var/run/openvswitch/vhost-user1"
```

pktgen Container: Set pktgen Parameters & Run pktgen

Or
```
# cd $TRAINING_DIR
# ./run_pktgen.sh
```

```
/*******
* -P: Promiscuous mode
* -T: Color terminal output
* -m "0.0,4.1" (core.port): core 0: port 0 rx/tx; core 4: port 1 rx/tx
*****/
export PKTGEN_PARAMS='-T -P -m ”0.0,4.1"'

(Use the PKTGEN_DIR, DPDK_DIR, DPDK_PARAMS and PKTGEN_PARAMS in the environ)
# cd $PKTGEN_DIR
# ./app/app/$DPDK_BUILD/pktgen $DPDK_PARAMS -- $PKTGEN_PARAMS
```

Useful `testpmd` and `pktgen` Commands

```
(Useful commands to use in testpmd)

testpmd> show port stats all
testpmd> clear port stats all
testpmd> help
```

```
(Useful commands to use in pktgen)

Pktgen> set 0 count 1000000
Pktgen> set 1 count 1000000
Pktgen> start 0
Pktgen> start all
Pktgen> set 0 rate 10
Pktgen> clr
Pktgen> rst
Pktgen> pdump 0
Pktgen> help
```

Viewing CPU Resources on the Host

Application	Parameter	Thread	Core Mask (CPUs 0-7)
Open vSwitch*	dpdk-lcore-mask=0x2	daemon	0b0000 0010
	pmd-cpu-mask=0x4	DPDK PMD	0b0000 0100
pktgen ()	-c 0x19	GUI & Messages	0b0000 1000
	--master-lcore 3	DPDK master lcore	0b0000 1000
	-m “0.0,4.1”	DPDK PMD	0b0001 0001
testpmd	--coremask=0xC0	DPDK master lcore	0b0010 0000
	-c 0xE0	testpmd DPDK PMD	0b1100 0000

Agenda

- Executive Summary
- DPDK and Containers Intro
- Hands-on Lab
- Conclusion

Questions

- What kind of performance are you seeing?
- What should you see with 10GB connection?
- Why is performance so poor?
- Why do ISVs/Telcos/CommSPs care about containers?
- What problems do you see with the DPDK in container setup shown today? How would you solve them?

Conclusion

- Container networks can use DPDK
- Security issues?
- Performance still highly dependent on configuration
- Intel® Clear Containers may provide more ideal solution

References

- http://www.linuxquestions.org/questions/linux-newbie-8/how-to-use-dpdk-inside-linux-containers-4175537584/
- https://builders.intel.com/docs/container-and-kvm-virtualization-for-nfv.PDF
- http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/linux-containers-hypervisor-based-vms-paper.pdf
- http://events.linuxfoundation.org/sites/events/files/slides/Jun_Nakajima_NFV_Container_final.pdf
- http://developerblog.redhat.com/2015/06/02/can-you-run-intels-data-plane-development-kit-dpdk-in-adocker-container-yep/
- http://dpdk.org/ml/archives/dev/2016-January/031219.html
- https://dpdksummit.com/Archive/pdf/2016USA/Day02-Session02-Steve%20Liang-DPDKUSASummit2016.pdf

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit **http://www.intel.com/performance**.