Single-Root Input/Output Virtualization (SR-IOV) with Linux Containers

By Promotion Published Date 05 - Sep - 2016 | Last Updated 05 - Sep - 2016

Executive Summary

Virtualization is a technology offering an abstraction layer between software and hardware. It can emulate a hardware platform in order to provide the operating system abstractions of various resources. It is also a means to easily manage hardware resources and to run services in the cloud. One of the main disadvantages of the traditional hypervisor-based virtualization is, however, that it

APPS
Google Duo crossed 5 million downloads in one week

APPS
App of the Week: HouseJoy

introduces a large overhead with respect to memory and CPU1 usage. Linux* Containers (abbreviated to LXC) is a lightweight alternative to the traditional approach, providing separation of namespaces and file systems while running all processes on a single kernel. Combined with Single-Root Input/Output Virtualization (abbreviated to SR-IOV) it can provide a solution that allows the containers to appear in the network as separate compute-nodes with exclusive MAC2 addresses, while sharing one link and physical network adapter.

This paper is a result of a joint CERN openlab-Intel research activity with the aim to investigate whether Linux Containers can be used together with SR-IOV in conjunction and complementary to the existing virtualization infrastructure in the CERN Data Centre. This solution could

Input/Output operations while keeping the CPU mostly idle. CERN could benefit from LXC/SR-IOV by running both CPU-intensive and storage jobs in containers on one storage node, with full separation of both applications and control over consumed resources.

We describe a successful setup of Linux Containers on top of an Intel® 10-Gigabit Ethernet adapters with SR-IOV support together with results from synthetic network and storage benchmarks.

Authors: Pawel Szostek (CERN openlab) and Marco Righini (Intel)

1 Central Processing Unit
2 Media Access Control

Abstract

This document investigates the possibility to configure SR-IOV with LXC in the CERN environment with the most recent flavour of the operating system used at CERN (CERN CentOS* 7). We present all the steps necessary for an operational configuration. We also present results from the basic network and storage benchmarks run in this configuration. Our results prove that the technology stack is working. However, minor performance drops when running intensive I/O operations were detected. The future work involves running performance and

1. Introduction

What are Linux Containers?

Linux* Containers, abbreviated to LXC, is a feature implemented in the Linux kernel, allowing processes to be separated inside their own namespaces and file systems, while running on the same kernel. As shown by various studies (Xavier, et al., 2013; Kjallman, et al., 2015), its performance overhead is smaller than for

hypervisor-based, which comes for the cost of lower flexibility and manageability. LXC can be bundled together with cgroups, providing a way to control usage of hardware resources inside the containers, which results in even better separation of the applications running inside the different containers.

LXC differs from Docker in that it provides lightweight namespace separation capabilities while removing traditional VM overhead while Docker is a single application virtualization engine which runs on the top of the containers. It differs significantly from LXC in the way that it is meant to be running a single application, enclosed together with its dependencies in a single image. By default Docker disables storage persistence, making the images stateless over executions.

The Single-Root I/O3 Virtualization (SR-IOV) is a PCI Special Interest Group (PCI-SIG) specification for I/O virtualization. It provides a standard mechanism for devices to advertise their ability to be simultaneously shared among multiple virtual machines. It also allows for the partitioning of a PCI function into many virtual interfaces for the purpose of sharing the resources of a PCI Express* (PCIe*) device in a virtual environment.

Depending on the physical device and the support on the kernel side, a different number of Virtual Functions can be made available inside a system. For this paper we used Intel X520 Cards providing up to 64 Virtual Functions per port. Each of them can be delegated to a container, which will own it during its lifetime. As a result, the device will appear in the network as an independent node with completely isolated traffic.

The Big Picture

The CERN Data Centre lies at the heart of the Worldwide LHC Computing Grid. It is a place where data from the LHC experiments are gathered, stored and processed. It hosts almost 110,000 processor cores and 11,000 servers that run round-the-clock to ensure uninterrupted services. All the nodes

on their use case. In this paper we focus on the storage front-end nodes, whose aim is to collect data coming from the experiments and serve this data on demand to the computing nodes. They are characterized by low CPU usage and intensive use of their persistent storage resources. The aim of the investigation described in this Openlab paper is to look at the possibility of using Linux Containers to provide additional computing resources from the storage servers without the need to sacrifice storage reliability. A

diagram of the target setup is shown in Figure 1.

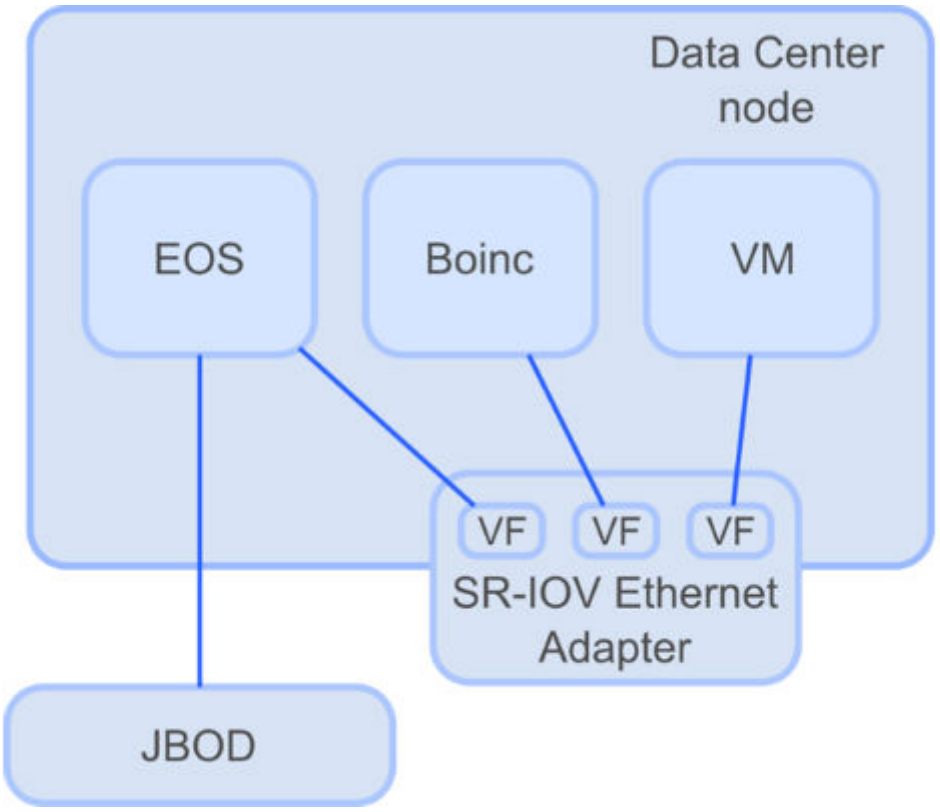

Figure 1: Example of a storage node configuration.

intensive with moderate need for computing resources.

Boinc is a distributed application for data processing – CPU intensive with negligible I/O. In the target configuration EOS and Boinc would run on a single storage node. Such a setup would allow better utilizing CPUs, which are currently underutilized, while keeping the applications separated from each other and providing them with the necessary hardware resources. To this end, SR-IOV enables us to create

many virtual network interfaces to be used inside the containers independently, while cgroups provides the capability for resource separation.

In this paper we describe a setup of LXC with SR-IOV tested with a handful of easy-to-run synthetic benchmarks. We do not investigate the target configuration as depicted in Figure 1, since such a setup is complicated, time-consuming and error-prone. Instead, we focus on running synthetic benchmarks with the aim of stressing I/O-related hardware resources.

3 Input/Output (operations)

2. Hardware and software setup

The node tested is equipped with a JBOD array of 24 Hitachi HUA72302 A800 drives and with an Intel® 82599ES 10

one management 1-Gigabit Ethernet port on the system board.

The machine is booted with pre-production CentOS* 7 with the 3.10.0-123.13.2.el7.x86_64 kernel and CERN-specific add-ons. Standard packages are installed, unless explicitly mentioned.

BIOS configuration

SR-IOV requires the following BIOS settings to work correctly: VT enabled, SR-IOV enabled and VT-D disabled.

3. SR-IOV configuration in the OS

In this section we describe in detail all the steps needed to obtain a working setup of SR-IOV on the test system. In the remainder of the text the host machine will be called p01001534852033, while the containers will be called centos1 and centos2.

First of all, we ensure that the Ethernet adapters are visible in the system. Since these are PCIe devices, we use the lspci command:

[root@p01001534852033 tmp]# lspci | grep 10-Gig 04:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01) 04:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)

Once the drivers are downloaded, we unpack, build and install both ixgbe and ixgbevf kernel modules:

```
tar xvzf ixgbe-*.tar.gz
cd ixgbe-*/src
make install
cd ../..
tar xvzf ixgbevf-*.tar.gz
cd ixgbevf-*/src
make install
```

Afterwards, we configure the system to load the kernel modules automatically at boot-time:

```
echo "ixgbevf\nixgbe" >> /etc/modules-load.d/modules.conf
```

In addition, the ixgbe driver requires virtual memory pages of large size (also called huge memory pages) to be mounted:

```
mkdir -p /mnt/huge chmod 777 /mnt/huge echo "huge /mnt/huge" hugetlbfs defaults 0 0 echo "vm.nr_hugepagesz=2M" >> /etc/sysctl.conf
```

The driver requires the following parameters to be passed at boot to the kernel:

```
intel_iommu=off default_hugepagesz=2M hugepagesz=2M ixgbe.max_vfs=8,8.
```

The last parameter specifies the number

we use 8 virtual interfaces for every physical one.

To this end, we add a custom entry to grub.cfg. Since CentOS7 uses grub2, this has to be done indirectly by adding an entry to /etc/grub.d/40_custom. The entry below is a clone of another entry with the extra parameters added. For further details please consult /boot/grub2/grub.cfg:

```
cat << EOF > /etc/grub.d/40_custom
menuentry 'CentOS Linux (3.10.0-
123.9.3.el7.x86_64) 7 Intel setup' --class
centos --class gnu-linux --class gnu --
class os --unrestricted
$menuentry_id_option 'gnulinux-3.10.0-
123.el7.x86_64-advanced-3d81a16b-
9af7-4ba0-ae53-ef16fb54f864' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_msdos
        insmod ext2
        set root='hd0,msdos1'
         if [ x$feature_platform_search_hint
= xy ]; then
            search --no-floppy --fs-uuid --
set=root --hint-bios=hd0,msdos1 --hint-
efi=hd0,msdos1 --hint-
baremetal=ahci0,msdos1 --
hint='hd0,msdos1' 0b7ebc67-f22d-
```



```
set=root 0b7ebc67-f22d-4760-abd9-
7503761e827f
        fi
                linux16 /vmlinuz-3.10.0-
123.8.1.el7.x86_64
root=/dev/mapper/cc_p01001534852033-
root ro
rd.lvm.lv=cc_p01001534852033/swap
crashkernel=auto
            vconsole.font=latarcyrheb-sun16
rd.lvm.lv=cc_p01001534852033/root
vconsole.keymap=us rhgb quiet
LANG=en_US.UTF-8 intel_iommu=off
```

```
default_hugepagesz=2M
hugepagesz=2M ixgbe.max_vfs=8,8
        initrd16 /initramfs-3.10.0-
123.8.3.el7.x86_64.img
}
EOF
```

Next, we recreate the GRUB2 configuration file to allow it include the latest changes:

```
grub2-mkconfig --
output=/boot/grub2/grub.cfg
```

Finally we reboot the system to check that the SR-IOV-related configuration is correct and reproducible with every boot.

Once the system is rebooted with the new parameters, we test whether virtual functions are visible in the system. If they are not listed by lspci, it might mean that the drivers were not loaded or loaded incorrectly (in this case, consult lsmod and


```
[root@n0100153852033 tmp]# lspci |
grep -i virt
00:11.0 PCI bridge: Intel Corporation
C600/X79 series chipset PCI Express
Virtual Root Port (rev 06)
04:10.0 Ethernet controller: Intel
Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
04:10.1 Ethernet controller: Intel
Corporation 82599 Ethernet Controller
Virtual Function (rev 01)
04:10.2 Ethernet controller: Intel
Corporation 82599 Ethernet Controller
```

Virtual Function (rev 01)
04:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:10.4 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:10.6 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:11.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:11.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller

Virtual Function (rev 01)
04:11.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:11.4 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:11.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
04:11.6 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)

04:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)

We double check that huge memory pages are mounted:

```
[root@p01001534852033 tmp]# mount | grep huge
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
huge on /mnt/huge type hugetlbfs (rw,relatime)
```

Also, all the virtual interfaces should be reported by ifconfig:

```
[root@p01001534852033 ~]# ifconfig | grep enp4s0 | cut -c1-52
```



```
enp4s17:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s0f0:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s0f1:
flags=4099<UP,BROADCAST,MULTICAST>
 mtu 150
enp4s16f2:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s16f4:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s16f6:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
```

```
enp4s17f2:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s17f4:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
enp4s17f6:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
```

If this is not the case, try reloading ixgbe and ixgbevf. However, if you use SSH to connect to the machine, the connection will break. This can be avoided by running.

```
nohup 'rmmod ixgbe && rmmod ixgbevf
&& insmod ixgbe && insmod ixgbevf'.
```

Note that ixgbevf has to be loaded after ixgbe, otherwise the virtual functions will not be reported by the system.

When working on the solution we stumbled upon a bug in the CentOS7 LXC template that results in a very slow

Once the above-mentioned configuration is in place, we create a new CentOS container from scratch:

```
lxc-create -t centos -n centos1
```

Before we can start the container, we need to change its settings so that it does not use the bridged network adapter, but a virtual function instead. A sample configuration is shown below:

```
[root@p01001534852033 tmp]# cat /var/lib/lxc/centos1/config
lxc.autodev = 1
lxc.network.type = phys # was veth
lxc.network.flags = up
lxc.network.link = enp4s16f1  # was virbr0
lxc.rootfs = /var/lib/lxc/centos1/rootfs
lxc.include = /usr/share/lxc/config/centos.common.conf
lxc.arch = x86_64
lxc.utsname = centos1.cern.ch
lxc.autodev = 1
```

Each time the ixgbevf kernel module is loaded, the virtual functions are assigned new MAC addresses. To prevent any consequences (like misrouting the packets through the CERN network), we use Python script below to reset the addresses. We are aware that this is a suboptimal solution, but should work as a temporary workaround.


```
import subprocess
intf_mac_list0 = [("enp4s16","d6:d6:d6:54:f9:f8"),
("enp4s16f2", "ee:2b:1d:e6:f7:aa"),
("enp4s16f4", "1e:99:1e:f0:bb:64"),
("enp4s16f6", "6e:82:0c:ff:f8:47"),
("enp4s17", "da:61:59:96:ab:1d"),
("enp4s17f2", "fe:66:3f:87:74:12"),
("enp4s17f4", "72:7b:1a:0c:bf:fc"),
("enp4s17f6", "46:02:c9:b1:3a:a7")]
intf_mac_list1 = [("enp4s16f1", "7a:e7:81:9c:43:38"),
("enp4s16f3", "ca:fa:89:58:95:92"),
```

```
("enp4s16f5", "ba:7d:2e:d3:1b:96"),
("enp4s16f7", "3e:72:0d:07:2b:e9"),
("enp4s17f1", "52:7b:24:90:11:01"),
("enp4s17f3", "fe:14:4f:45:6d:49"),
("enp4s17f5", "86:1e:f4:cc:fa:bb"),
("enp4s17f7", "0a:93:df:2e:62:2b")]
for idx, tup in enumerate(intf_mac_list0):
   intf, mac = tup
    cmd = "ip link set %s vf %d mac %s" %
("enp4s0f0", idx, mac)
   print(cmd)
        subprocess.Popen(cmd,  shell=True,
stdin=subprocess.PIPE,
stdout=subprocess.PIPE)
for idx, tup in enumerate(intf_mac_list1):
   intf, mac = tup
    cmd = "ip link set %s vf %d mac %s" %
("enp4s0f1", idx, mac)
   print(cmd)
        subprocess.Popen(cmd,  shell=True,
stdin=subprocess.PIPE,
```


connect to its terminal:

```
[root@p01001534852033  ~]#  lxc-start
–d -n centos1
[root@p01001534852033 ~]# lxc-attach
-n centos1
[root@centos1 ~]#
```

https://lists.linuxcontainers.org/pipermail/lxc-users/2014-July/007443.html
https://github.com/tukiyo/lxc/commit/dbf45a526bf5a2f0503737b0b68d244e9389a2

4. Tests

Network Bandwidth Tests

We conducted very simple tests to prove that the network bandwidth obtained inside and outside of the container is at the same level. For this purpose, we set up two machines with 10GbE cards connected to the same 10GbE Ethernet switch. We used the iperf3, which benchmarks client-server data transmission. Our client is the machine hosting the containers, while the server is another machine (cd1001534-0004132575) connected directly to the same switch. First, we ran the benchmark both in and outside container and measured the bandwidth. Then we repeated the tests in reverse mode, i.e. with traffic going from the server to the client. The results show that the link is symmetrical and using LXC does not influence network traffic in this


```
[root@centos3 ~]# iperf3 -c cd1001534-
0004132575 -O 3 -t 20
Connecting to host cd1001534-
0004132575, port 5201
...
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
   Retr
[  4]  0.00-20.00  sec  21.9 GBytes  9.41
Gbits/sec  0             sender
[  4]  0.00-20.00  sec  22.0 GBytes  9.43
Gbits/sec                receiver
[root@p0100015340741 88 ~]# iperf3 -c
```

```
cd1001534-0004132575 -O 3 -t 20
...
- - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
   Retr
[  4]  0.00-20.00  sec  21.9 GBytes  9.41
Gbits/sec  0             sender
[  4]  0.00-20.00  sec  21.9 GBytes  9.40
Gbits/sec                receiver
```

Disk tests

In our tests we wanted to understand whether there are any limitations in the software stack while accessing the drives. We ran two disk benchmarks – iozone and ssdbench. The former is designed to test disk throughput and was run on external HGST HUA72302 spindles located in a JBOD array connected via a SAS HBA. The latter benchmarks focuses on both throughput and IOPS for SSD and

Spinning drive

For this benchmark we connected the test system over a LSI 9207-8e SAS host bus adapter to an external JBOD array with HGST HUA72302 2TB drives. The tests with iozone aim at mimicking accesses done by CASTOR (CERN Advanced STORage manager). On every device tested we created an XFS file system, which was mounted using the following options:

logbufs=8,logbsize=256k,noatime,swalloc,inode64.

In the first tests we measured maximum throughput of read and write operations for every drive separately. We ran the test four times and then calculated mean values of sequential read, random read, sequential write, random write and mixed workload. The command used is shown below:

```
for i in `seq 1 4`; do /opt/iozone/bin/iozone -R -l 1 -u 1 -r 1m -s 4g -F /srv/castor/{drive}/iozone.dat | tee -a iozone-1m-4g-first-disk-loop-"$i".txt; done
```

The second test performs simultaneous reads and writes to/from 9 spindles. We used the following command:

```
for i in `seq 1 4`; do /opt/iozone/bin/iozone -R -l 1 -u 1 -r 1m -s 4g -F /srv/castor/{1,2,3,4,5,6,7,8,9}/iozone.dat | tee -a iozone-1m-4g-24fs-loop-"$i".txt; done
```


The difference between results obtained on the OS and inside a container was lower than 1%. The values obtained on the host OS are shown in the Figure 2.

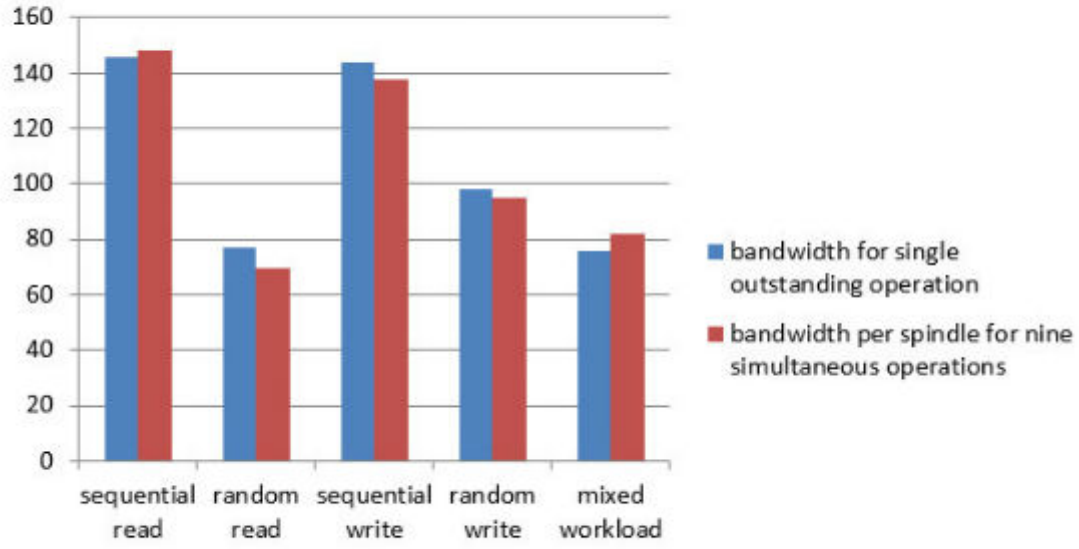

Figure 2: Spinning drive I/O performance

Solid-state drives

In order to test the setup with high bandwidth and low latency drive, we fitted

Peak gen 3 bus. To access the drive we used the fio command, which provides a convenient interface for the various operations on the drives. With this test we intended to check whether there are any limitations in the software stack on the I/O operations inside the containers that might affect IOPS, throughput or latencies.

In our experiments we conducted following tests:

- Single-threaded 512B sequential write with varying queue depth
- Single-threaded sequential write with varying block size
- Sustained 4kB random read with varying number of threads
- Sustained multi-threaded random read with varying block size
- Sustained 4kB random mixed access with varying number of threads
- Sustained 4kB random write with varying number of threads
- Sustained random write with varying number of threads

In most cases the results from inside and outside of container vary at most by 1%. However, when running the tests with larger blocks (32kB and 64kB) IOPS, bandwidth and latency are significantly worse inside the containers. These results can be found in Table 1 and

	77094		.00%		244		1070.81	107	
1024	497015	499390	-0.48%	508	511	-0.59%	1027.77	1022.66	-0.50%
2048	469362	472190	-0.60%	961	967	-0.62%	1088.18	1081.56	-0.61%
4096	389633	392483	-0.73%	1595	1607	-0.75%	1311.51	1301.63	-0.75%
8192	206444	205842	0.29%	1691	1686	0.30%	2477.98	2484.89	0.28%
16384	103754	103441	0.30%	1699	1694	0.29%	4932.13	4946.21	0.29%
32768	44451	23158	**47.90%**	1456	758	**47.94%**	11514.24	22100.29	**91.94%**
65536	14289	11889	**16.80%**	936	779	**16.77%**	35821.39	43051.09	**20.18%**
131072	2837	2865	-0.99%	371	375	-1.08%	180377.07	178622.08	-0.97%

Table 1: Sustained Multi-Threaded random read tests by block size, 512 threads, IO queue depth=1

Block (B)	IOPS (host)	IOPS (LXC)	IOPS Penalty	BW (MB/s, host)	BW (MB/s, LXC)	BW penalty	Read latency (host)	Read latency (LXC)	Read latency penalty
4096	46040	46966	-2.01%	188	192	2.13%	19.83	19.54	-1.46%
8192	39340	38146	3.04%	322	312	-3.11%	23.48	24.28	3.41%
16384	27561	27243	1.15%	451	446	-1.11%	33.64	33.88	0.71%
32768	20531	16495	19.66%	672	540	-19.64%	46.59	57.78	24.02%
65536	13563	11395	15.98%	888	746	-15.99%	71.62	84.16	17.51%

Table 2: Sequential single-threaded write tests by block size, IO queue depth=1

drop when using larger block sizes. For instance, in the sustained multi-threaded read benchmark run we obtained 47.90% less IOPS, 47.94% less bandwidth and 91.94% higher latency for blocks of size 32kB. Similarly, in the sequential single-threaded benchmark run we obtained 19.66% lower IOPS, 19.64% lower bandwidth and 24.02% higher write latency. These irregularieties have to be taken into account when designing a computing ecosystem based on the tested technologies (SR-IOV, LXC, NVMe).

5. Conclusions

In our investigations we managed to create a working setup of Linux Containers on top of SR-IOV. We ran networking and disk benchmarks that show that the setup is working, but minor irregularities might be expected when accessing solid-state drives. We consider this technology stack to be potentially useful in the CERN infrastructure, as described in paragraph 1.4, and suitable for further studies at CERN.

6. Bibliography

CERN Advanced STORage manager (CASTOR) http://castor.web.cern.ch/.

Kjallman J and Komu M Hypervisors vs. Lightweight Virtualization: A Performance Comparison [Conference] // IEEE International Conference on Cloud

Xavier Miguel [et al.] Performance Evaluation of Container-based Virtualization for High Performance Computing Environments [Conference] // Parallel, Distributed and Network-Based Processing . - Belfast : IEEE, 2013. - pp. 233-240.

Appendix A: Controlling resource usage inside containers

Limiting network usage

The Linux Traffic Controller (tc) can be easily used to limit traffic on the network interfaces. An example of limiting bandwidth to 1kbps on eth0 is shown below:

tc qdisc add dev eth0 handle 1: root htb default 11

tc class add dev eth0 parent 1: classid 1:1 htb rate 1kbps

tc class add dev eth0 parent 1:1 classid 1:11 htb rate 1kbps
To remove a limit on eth0, one need to run the following command:
tc qdisc del dev eth0 root

Limiting usage of other resources

Linux containers allow limiting resource usage per container. The limits can be set

/var/lib/lxc/<container_name>/config or dynamically with the lxc-cgroups command. Manual page for the lxc-cgroups command provides further explanation of the available features.

Enabling usage of block and character devices inside containers

By default, block and character devices available in the system are not visible in a container. In order to use them, one has to recreate device files in the container

namespace. So far, the best way we have found to do so is to combine autodev hook script together with corresponding cgroup settings:

[root@p01001534852033 centos2]# cat config

--
lxc.autodev = 1
lxc.hook.autodev = ./sd.sh
lxc.cgroup.devices.allow = b 8:* rwm
lxc.cgroup.devices.allow = b 259:* rwm
[root@p01001534852033 centos2]# cat sd.sh
#!/bin/bash
cd ${LXC_ROOTFS_MOUNT}/dev
mknod sdc b 8 32
mknod sdd b 8 48
mknod sde b 8 64
mknod sdf b 8 80
mknod sdg b 8 96
mknod sdh b 8 112
mknod nvme0 c 10 58
mknod nvme0n1 b 259 0

For more such intel Modern Code and tools from Intel, please visit the Intel® Modern Code

Source:https://software.intel.com/en-us/articles/single-root-inputoutput-virtualization-sr-iov-with-linux-containers

LOAD PAGE 1

Digit caters to the largest community of tech buyers, users and enthusiasts in India. The all new Digit.in continues the legacy of Thinkdigit.com as one of the largest portals in India committed to technology users and buyers. Digit is also one of the most trusted names when it comes to technology reviews and buying advice and is home to the Digit Test Lab, India's most proficient center for testing and reviewing technology products.

We are about leadership – the 9.9 kind! Building a leading media company out of India. And, grooming new leaders for this promising industry.