



# Best Practices for DGX

## Best Practices

# Table of Contents

Chapter 1. Overview.....	1
Chapter 2. Storage.....	2
2.1. Internal Storage (NFS Cache).....	2
2.1.1. NFS Cache for Deep Learning.....	2
2.1.2. RAID-0.....	3
2.1.3. DGX Internal Storage.....	3
2.1.4. Monitoring the RAID Array.....	4
2.2. External Storage.....	6
2.2.1. NFS Storage.....	7
2.2.2. Distributed Filesystems.....	9
2.2.3. Scaling Out Recommendations.....	9
Chapter 3. Authenticating Users.....	11
3.1. Local.....	11
3.2. NIS Vs NIS+.....	11
3.3. LDAP.....	12
3.4. Active Directory.....	12
Chapter 4. Time Synchronization.....	13
4.1. Ubuntu 16.04.....	13
Chapter 5. Monitoring.....	14
5.2. Using ctop For Monitoring.....	16
5.3. Monitoring A Specific DGX Using nvidia-smi.....	17
Chapter 6. Managing Resources.....	18
6.1. Example: SLURM.....	19
6.1.1. Simple GPU Scheduling With Exclusive Node Access.....	19
6.1.2. Scheduling Resources At the Per-GPU Level.....	20
6.2. Example: Univa Grid Engine.....	21
6.3. Example: IBM Spectrum LSF.....	21
6.4. Example: Altair PBS Pro.....	21
Chapter 7. Provisioning and Cluster Management.....	22
7.1. Example: Bright Computing Cluster.....	22
Chapter 8. Networking.....	23
8.1. DGX-1 Networking.....	23
8.1.1. DGX-1 InfiniBand Networking.....	24
8.1.2. DGX-1 Ethernet Networking.....	25

8.1.3. DGX-1 Bonded NICs.....	26
8.2. DGX-2 Networking.....	27
Chapter 9. SSH Tunneling.....	28
Chapter 10. Head Node.....	30
Chapter 11. DGX-2 KVM Networking.....	31
11.1. Introduction.....	31
11.1.1. Network Configuration Options.....	31
11.1.2. Acronyms.....	32
11.2. Virtual Networking.....	32
11.2.1. Default Configuration.....	32
11.2.2. Using Static IP.....	34
11.2.3. Binding the Virtual Network to a Specific Physical NIC.....	35
11.3. Bridged Networking.....	36
11.3.1. Introduction.....	36
11.3.2. Using DHCP.....	36
11.3.3. Using Static IP.....	37
11.4. Bridged Networking with Bonding.....	38
11.4.1. Introduction.....	38
11.4.2. Using DHCP.....	38
11.4.3. Using Static IP.....	39
11.5. MacVTap.....	40
11.5.1. Introduction.....	40
11.5.2. Macvtap Modes.....	41
11.5.3. How to Change the Macvtap and Physical NIC Configuration.....	44
11.5.4. How to Configure the Guest VM Using privateIP.....	45
11.6. SR-IOV.....	45
11.6.1. Introduction.....	45
11.6.2. Device Configuration.....	47
11.6.3. Generic Configuration.....	47
11.6.4. Using DHCP.....	48
11.6.5. Using Static IP.....	49
11.7. Getting the Guest VM IP Address.....	49
11.8. Improving Network Performance.....	50
11.8.1. Jumbo Frames.....	50
11.8.2. Multi-Queue Support.....	51
11.8.3. QOS.....	53
11.9. References.....	53

<b>Chapter 12. DGX-2 KVM Performance Tuning.....</b>	<b>55</b>
12.1. Background.....	55
12.2. CPU Tuning.....	57
12.2.1. vCPU Pinning.....	57
12.2.2. How to Disable vCPU Pinning.....	59
12.2.3. Core Affinity Optimization.....	59
12.3. Memory tuning.....	62
12.3.1. Huge Pages Support.....	62
12.3.1.1. How to set up Huge Pages at Runtime.....	63
12.3.1.2. How to set up Huge Pages only for boot.....	64
12.3.1.3. How to disable Huge Pages in the Host.....	65
12.4. NUMA Tuning.....	65
12.4.1. Automatic NUMA Balancing.....	66
12.4.2. Enabling NUMA Tuning.....	66
12.4.2.1. Setting Up NUMA Tuning.....	66
12.4.2.2. Effects of Enabling NUMA Tuning.....	67
12.5. Emulatorpin.....	67
12.6. I/O tuning.....	68
12.6.1. Using Multiple-queues with Logical Volumes.....	68
12.6.1.1. I/O Threads.....	69
12.6.1.2. How to Set up I/O Tuning.....	69
12.6.2. NVMe Drives as PCI-Passthrough Devices.....	70
12.6.2.1. How to Set Up PCI-Passthrough for NVME Drives.....	70
12.6.2.2. How to Revert PCI-Passthrough of NVMe Drives.....	71
12.6.3. Physical Drive Passthrough.....	72
12.6.3.1. How to Set Up Drive Passthrough.....	73
12.6.3.2. How to Revert Drive Passthrough.....	73
12.6.4. Drive Partition Passthrough.....	74
12.6.4.1. How to Set Up Drive Partition Passthrough.....	76
12.6.4.2. How to Revert Drive Partition Passthrough.....	78

---

# Chapter 1. Overview

NVIDIA has created the DGX family as appliances to make administration and operation as simple as possible. However, like any computational resource it still requires administration. This section discusses some of the best practices around configuring and administering a single DGX or several DGX appliances.

There is also some discussion about how to plan for external storage, networking, and other configuration aspects for the DGX, focusing on DGX-2, DGX-1, and DGX Station (the DGX "family").

For detailed information about implementation, see:

- ▶ [DGX-2 User Guide](#)
- ▶ [DGX-1 User Guide](#)
- ▶ [DGX Station User Guide](#)

---

## Chapter 2. Storage

For deep learning to be effective and to take full advantage of the DGX family, the various aspects of the system have to be balanced. This includes storage and I/O. This is particularly important for feeding data to the GPUs to keep them busy and dramatically reduce run times for models. This section presents some best practices for storage within and outside of the DGX-2, DGX-1, or DGX Station. It also talks about storage considerations as the number of DGX units are scaled out, primarily the DGX-1 and DGX-2.

### 2.1. Internal Storage (NFS Cache)

The first storage consideration is storage within the DGX itself. The focus of the internal storage, outside of the OS drive, is performance.

#### 2.1.1. NFS Cache for Deep Learning

Deep Learning I/O patterns typically consist of multiple iterations of reading the training data. The first epoch of training reads the data that is used to start training the model. Subsequent passes through the data can avoid rereading the data from NFS if adequate local caching is provided on the node. If you can estimate the maximum size of your data, you can architect your system to provide enough cache so that the data only needs to be read once during any training job. A set of very fast SSD disks can provide an inexpensive and scalable way of providing adequate caching for your applications. The DGX family NFS read cache was created for precisely this purpose, offering roughly 5, 7, and 30+ TB of fast local cache on DGX Station, DGX-1, and DGX-2, respectively.

For training the best possible model, the input data is randomized. This adds some additional statistical noise to the training and also keeps the model from being “overfit” on the training data (in other words, trained very well on the training data but doesn’t do well on the validation data). Randomizing the order of the data for training puts pressure on the data access. The I/O pattern becomes random oriented rather than streaming oriented. The DGX family NFS cache is SSD-based with a very high level of random IOPs performance.

The benefit of adequate caching is that your external filesystem does not have to provide maximum performance during a cold start (the first epoch), since this first pass through the data is only a small part of the overall training time. For example, typical training sessions can iterate over the data 100 times. If we assume a 5x slower read access time during the first

cold start iteration vs the remaining iterations with cached access, then the total run time of training increases by the following amount.

- ▶ 5x slower shared storage 1st iteration + 99 local cached storage iterations
  - ▶ > 4% increase in runtime over 100 iterations

Even if your external file system cannot sustain peak training IO performance, it has only a small impact on overall training time. This should be considered when creating your storage system to allow you to develop the most cost-effective storage systems for your workloads.

For either the DGX Station or the DGX-1 **you cannot put additional drives into the system without voiding your warranty.** For the DGX-2, you can add additional 8 U.2 NVMe drives to those already in the system.

## 2.1.2. RAID-0

The internal SSD drives are configured as RAID-0 array, formatted with [ext4](#), and mounted as a file system. This is then used as an NFS read cache to cache data reads. Recall that its number one focus is performance.

RAID-0 stripes the contents of each file across all disks in the RAID group, but doesn't perform any mirroring or parity checks. This reduces the availability of the RAID group but it also improves its performance and capacity. The capacity of a RAID-0 group is the sum of the capacities of the drives in the set.

The performance of a RAID-0 group, which results in improved throughput of read and write operations to any file, is the number of drives multiplied by their performance. As an example, if the drives are capable of a sequential read throughput of 550 MB/s and you have three drives in the RAID group, then the theoretical sequential throughput is  $3 \times 550\text{MB/s} = 1650\text{ MB/s}$ .

## 2.1.3. DGX Internal Storage

The **DGX-2** has 8 or 16 3.84 TB NVMe drives that are managed by the OS using `mdadm` (software RAID). On systems with 8 NVMe drives, you can add an additional 8.

The **DGX-1** has a total five 1.92TB SSDs. These are plugged into the LSI controller (hardware RAID). In the DGX-1, there are a total of five or six 1.92TB SSDs. These are plugged into the LSI controller. Two RAID arrays are configured:

- ▶ Either a single-drive RAID-0 or a dual-drive RAID-1 array for the OS, and a
- ▶ Four-drive RAID-0 array to be used as read cache for NFS file systems. The Storage Command Line Tool (StorCLI) is used by the LSI card.



**Note:** You cannot put additional cache drives into the DGX-1 without voiding your warranty.

The **DGX Station** has three 1.92 TB SSDs in a RAID-0 group. The Linux software RAID tool, `mdadm`, is used to manage and monitor the RAID-0 group.



**Note:** You cannot put additional cache drives into the DGX Station without voiding your warranty.

## 2.1.4. Monitoring the RAID Array

This section explains how to use `mdadm` to monitor the RAID array in DGX-2 and DGX Station systems.

The RAID-0 group is created and managed by Linux software, [mdadm](#). `mdadm` is also referred to as “software RAID” because all of the common RAID functions are carried out by the host CPUs and the host OS instead of a dedicated RAID controller processor. Linux software RAID configurations can include anything presented to the Linux kernel as a block device. Examples include whole hard drives (for example, `/dev/sda`), and their partitions (for example, `/dev/sda1`).

Of particular importance is that since version 3.7 of the Linux kernel mainline, `mdadm` supports [TRIM](#) operations for the underlying solid-state drives (SSDs), for linear, RAID 0, RAID 1, RAID 5 and RAID 10 layouts. TRIM is very important because it helps with garbage collection on SSDs. This reduces [write amplification](#) and reduces the wear on the drive.

There are some very simple commands using `mdadm` that you can use for monitoring the status of the RAID array. The first thing you should do is find the mount point for the RAID group. You can do this by simply running the command `mount -a`. Look through the output for `mdadm`-created devices with naming format `/dev/md*`.

```
# mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
...
/dev/md0 on /raid type ext4 (rw,relatime,discard,stripe=384,data=ordered)
/dev/sdb1 on /boot/efi type vfat
(rw,relatime,mask=0077,dmask=0077,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
...
```

This `/dev/md0` is a RAID-0 array that acts as a read cache for NFS file systems.

One of the first commands that can be run is to check the status of the RAID group. The command is simple, `cat /proc/mdstat`.

```
# cat /proc/mdstat
Personalities : [raid0] [linear] [multipath] [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid0 sde[2] sdd[1] sdc[0]
      5625729024 blocks super 1.2 512k chunks

unused devices: <none>
```

This is a command to read the `mdstat` file in the `/proc` filesystem. The output looks compact but there is a great deal of information in that output. The first line of output is a list of the possible ways to use `mdadm` in this version of Linux.

The next lines of output will present some details on each `md` device. In this case, the DGX Station only has one RAID group, `/dev/md0`. The output for `/dev/md0` means that it is an active RAID-0 group. It has three devices:

- ▶ `sdc`
- ▶ `sdd`
- ▶ `sde`



It also lists the number of blocks in the device, the version of the super block (1.2), and the chunk size (512k). This is the size that is written to each device when `mdadm` breaks up a file. This information would be repeated for each `md` device (if there are more than one).

Another option you can use with `mdadm` is to examine/query the individual block devices in the RAID group and examine/query the RAID groups themselves. A simple example from a DGX Station is below. The command queries the RAID group.

```
# mdadm --query /dev/md0
/dev/md0: 5365.11GiB raid0 3 devices, 0 spares. Use mdadm --detail for more detail.
```

Notice that there are 3 devices with a total capacity of 5,365.11 GiB (this is different than GB). If this were a RAID level that supported redundancy rather than focusing on maximizing performance, you could allocate drives as 'spares' in case an active one failed. Because the DGX use RAID-0 across all available cache drives, there are no spares.

Next is an example of querying a block device that is part of a RAID group.

```
# mdadm --query /dev/sdc
/dev/sdc: is not an md array
/dev/sdc: device 0 in 3 device active raid0 /dev/md0. Use mdadm --examine for more detail.
```

The query informs you that the drive is not a RAID group but is part of a RAID group (`/dev/md0`). It also advises to examine the RAID group using the "examine" (`-E`) option.

Querying the block devices and the RAID group itself, you can put together how the block devices are part of the RAID group. Also notice that the commands are run by the root user (or something with root privileges).

To get even more detail about the `md` RAID group, you can use the `--examine` option. It prints the `md` superblock (if present) from a block device that could be a group component.

```
# mdadm --examine /dev/sdc
/dev/sdc:
    Magic : a92b4efc
    Version : 1.2
    Feature Map : 0x0
    Array UUID : 1feabd66:ec5037af:9a40a569:d7023bc5
    Name : demouser-DGX-Station:0 (local to host demouser-DGX-Station)
    Creation Time : Wed Mar 14 16:01:24 2018
    Raid Level : raid0
    Raid Devices : 3

    Avail Dev Size : 3750486704 (1788.37 GiB 1920.25 GB)
    Data Offset : 262144 sectors
    Super Offset : 8 sectors
    Unused Space : before=262056 sectors, after=0 sectors
    State : clean
    Device UUID : 482e0074:35289a95:7d15e226:fe5cbf30

    Update Time : Wed Mar 14 16:01:24 2018
    Bad Block Log : 512 entries available at offset 72 sectors
    Checksum : ee25db67 - correct
    Events : 0

    Chunk Size : 512K

    Device Role : Active device 0
    Array State : AAA ('A' == active, '.' == missing, 'R' == replacing)
```

It provides information about the RAID array (group) including things such as:

- Creation time
- UUID of the array (RAID group)

- ▶ RAID level (this is RAID-0)
- ▶ Number of RAID devices
- ▶ Size of the device both in GiB and GB (they are different)
- ▶ The state of the device (clean)
- ▶ Number of active devices in RAID array (3)
- ▶ The role of the device (if is device 0 in the raid array)
- ▶ The checksum and if it is correct
- ▶ Lists the number of events on the array

Another way to get just about the same information but some extra detail, is to use the `--detail` option with the raid array as below.

```
# mdadm --detail /dev/md0
/dev/md0:
    Version : 1.2
    Creation Time : Wed Mar 14 16:01:24 2018
    Raid Level : raid0
    Array Size : 5625729024 (5365.11 GiB 5760.75 GB)
    Raid Devices : 3
    Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Wed Mar 14 16:01:24 2018
    State : clean
    Active Devices : 3
    Working Devices : 3
    Failed Devices : 0
    Spare Devices : 0

    Chunk Size : 512K

    Name : demouser-DGX-Station:0 (local to host demouser-DGX-Station)
    UUID : 1feabd66:ec5037af:9a40a569:d7023bc5
    Events : 0

    Number   Major   Minor   RaidDevice State
    0         8       32      0        active sync  /dev/sdc
    1         8       48      1        active sync  /dev/sdd
    2         8       64      2        active sync  /dev/sde
```

## 2.2. External Storage

As an organization scales out their GPU enabled data center, there are many shared storage technologies which pair well with GPU applications. Since the performance of a GPU enabled server is so much greater than a traditional CPU server, special care needs to be taken to ensure the performance of your storage system is not a bottleneck to your workflow.

Different data types require different considerations for efficient access from filesystems. For example:

- ▶ Running parallel HPC applications may require the storage technology to support multiple processes accessing the same files simultaneously.

- ▶ To support accelerated analytics, storage technologies often need to support many threads with quick access to small pieces of data.
- ▶ For vision based deep learning, accessing images or video used in classification, object detection or segmentation may require high streaming bandwidth, fast random access, or fast memory mapped (`mmap()`) performance.
- ▶ For other deep learning techniques, such as recurrent networks, working with text or speech can require any combination of fast bandwidth with random and small files.

HPC workloads typically drive high simultaneous multi-system write performance and benefit greatly from traditional scalable parallel file system solutions. You can size HPC storage and network performance to meet the increased dense compute needs of GPU servers. It is not uncommon to see per-node performance increases from between 10-40x for a 4 GPU system vs a CPU system for many [HPC applications](#).

Data Analytics workloads, similar to HPC, drive high simultaneous access, but are more read focused than HPC. Again, it is important to size Data Analytics storage to match the dense compute performance of GPU servers. As you adopt accelerated analytics technologies such as GPU-enabled in-memory databases, make sure that you can populate the database from your data warehousing solution quickly to minimize startup time when you change database schemas. This may require a network with 10 Gbe for greater performance. To support clients at this rate, you may have to revisit your data warehouse architecture to identify and eliminate bottlenecks.

Deep learning is a fast evolving computational paradigm and it is important to know what your requirements are in the near and long term to properly architect a storage system. The [ImageNet database](#) is often used as a reference when benchmarking deep learning frameworks and networks. The resolution of the images in ImageNet are 256x256. However, it is more common to find images at 1080p or 4k. Images in 1080p resolution are **30 times** larger than those in ImageNet. Images in 4k resolution are 4 times larger than that (**120X the size of ImageNet images**). Uncompressed images are **5-10 times larger** than compressed images. If your data cannot be compressed for some reason, for example if you are using a custom image formats, the bandwidth requirements increase dramatically.

For AI-Driven Storage, it is suggested that you make use of deep learning framework features that build databases and archives versus accessing small files directly; reading and writing many small files will reduce performance on the network and local file systems. Storing files in formats such as HDF5, LMDB or TFRecord can reduce metadata access to the filesystem helping performance. However, these formats can lead to their own challenges with additional memory overhead or requiring support for fast `mmap()` performance. All this means that you should plan to be able to read data at 150-200 MB/s per GPU for files at 1080p resolution. Consider more if you are working with 4k or uncompressed files.

## 2.2.1. NFS Storage

NFS can provide a good starting point for AI workloads on small GPU server configurations with properly sized storage and network bandwidth. NFS based solutions can scale well for larger deployments, but be aware of possible single node and aggregate bandwidth requirements and make sure that matches your vendor of choice. As you scale your data center to need more than 10 GB/s or your data center grows to hundreds or thousands of nodes, other technologies may be more efficient and scale better.

Generally, it is a good idea to start with NFS using one or more of the Gigabit Ethernet connections on the DGX family. After this is configured, it is recommended that you run your applications and check if IO performance is a bottleneck. Typically, NFS over 10Gb/s Ethernet provides up to 1.25 GB/s of IO throughput for large block sizes. If, in your testing, you see NFS performance that is significantly lower than this, check the network between the NFS server and a DGX server to make sure there are no bottlenecks (for example, a 1 GigE network connection somewhere, a misconfigured NFS server, or a smaller MTU somewhere in the network).

There are a number of online articles, [such as this one](#), that list some suggestions for tuning NFS performance on both the client and the server. For example:

- ▶ Increasing Read, Write buffer sizes
- ▶ TCP optimizations including larger buffer sizes
- ▶ Increasing the MTU size to 9000
- ▶ Sync vs. Async
- ▶ NFS Server options
- ▶ Increasing the number of NFS server daemons
- ▶ Increasing the amount of NFS server memory

Linux is very flexible and by default most distributions are conservative about their choice of IO buffer sizes since the amount of memory on the client system is unknown. A quick example is increasing the size of the read buffers on the DGX (the NFS client). This can be achieved with the following system parameters:

- ▶ `net.core.rmem_max=67108864`
- ▶ `net.core.rmem_default=67108864`
- ▶ `net.core.optmem_max=67108864`

The values after the variable are example values (they are in bytes). You can change these values on the NFS client and the NFS server, and then run experiments to determine if the IO performance improves.

The previous examples are for the kernel read buffer values. You can also do the same thing for the write buffers where you use `wmem` instead `rmem`.

You can also tune the TCP parameters in the NFS client to make them larger. For example, you could change the `net.ipv4.tcp_rmem="4096 87380 33554432"` system parameter.

This changes the TCP buffer size, for IPv4, to 4,096 bytes as a minimum, 87,380 bytes as the default, and 33,554,432 bytes as the maximum.

If you can control the NFS server, one suggestion is to increase the number of NFS daemons on the server.

One way to determine whether more NFS threads helps performance is to check the data in `/proc/net/rpc/nfs` entry for the load on the NFS daemons. The output line that starts with `th` lists the number of threads, and the last 10 numbers are a histogram of the number of seconds the first 10% of threads were busy, the second 10%, and so on.

Ideally, you want the last two numbers to be zero or close to zero, indicating that the threads are busy and you are not "wasting" any threads. If the last two numbers are fairly high, you should add NFS daemons, because the NFS server has become the bottleneck. If the last two, three, or four numbers are zero, then some threads are probably not being used.

One other option, while a little more complex, can prove to be useful if the IO pattern becomes more **write** intensive. If you are not getting the IO performance you need, change the mount behavior on the NFS clients from "sync" to "async".



**CAUTION:** By default, NFS file systems are mounted as "sync" which means the NFS client is told the data is on the NFS server after it has actually been written to the storage indicating the data is safe. Some systems will respond that the data is safe if it has made it to the write buffer on the NFS server and not the actual storage.

Switching from "sync" to "async" means that the NFS server responds to the NFS client that the data has been received when the data is in the NFS buffers on the server (in other words, in memory). The data hasn't actually been written to the storage yet, it's still in memory. Typically, writing to the storage is much slower than writing to memory, so write performance with "async" is much faster than with "sync". However, if, for some reason, the NFS server goes down before the data in memory is written to the storage, then the data is lost.

If you try using "async" on the NFS client (in other words, the DGX system), ensure that the data on the NFS server is replicated somewhere else so that if the server goes down, there is always a copy of the original data. The reason is if the NFS clients are using "async" and the NFS server goes down, data that is in memory on the NFS server will be lost and cannot be recovered.

NFS "async" mode is very useful for write IO, both streaming (sequential) and random IO. It is also very useful for "scratch" file systems where data is stored temporarily (in other words, not permanent storage or storage that is not replicated or backed up).

If you find that the IO performance is not what you expected and your applications are spending a great deal of time waiting for data, then you can also connect NFS to the DGX system over InfiniBand using IPoIB (IP over IB). This is part of the DGX family software stack and can be easily configured. The main point is that the NFS server should be InfiniBand attached as well as the NFS clients. This can greatly improve IO performance.

## 2.2.2. Distributed Filesystems

Distributed filesystems such as [EXAScaler](#), [GRIDScaler](#), [Ceph](#), [Lustre](#), [MapR-FS](#), [General Parallel File System](#), [Weka.io](#), and [Gluster](#) can provide features like improved aggregate IO performance, scalability, and/or reliability (fault tolerance). These filesystems are supported by their respective providers unless otherwise noted.

## 2.2.3. Scaling Out Recommendations

Based on the general IO patterns of deep learning frameworks (see [External Storage](#)), below are suggestions for storage needs based on the use case. These are suggestions only and are to be viewed as general guidelines.

Table 1. Scaling out suggestions and guidelines

Use Case	Adequate Read Cache?	Network Type Recommended	Network File System Options
Data Analytics	NA	10 GbE	Object-Storage, NFS, or other system with good multithreaded read and small file performance
HPC	NA	10/40/100 GbE, InfiniBand	NFS or HPC targeted filesystem with support for large numbers of clients and fast single-node performance
DL, 256x256 images	yes	10 GbE	NFS or storage with good small file support
DL, 1080p images	yes	10/40 GbE, InfiniBand	High-end NFS, HPC filesystem or storage with fast streaming performance
DL, 4k images	yes	40 GbE, InfiniBand	HPC file system, high-end NFS or storage with fast streaming performance capable of 3+ GB/s per node
DL, uncompressed Images	yes	InfiniBand, 40/100 GbE	HPC filesystem, high-end NFS or storage with fast streaming performance capable of 3+ GB/s per node
DL, Datasets that are not cached	no	InfiniBand, 10/40/100 GbE	Same as above, aggregate storage performance must scale to meet the all applications simultaneously

As always, it is best to understand your own applications' requirements to architect the optimal storage system.

Lastly, this discussion has focused only on performance needs. Reliability, resiliency and manageability are as important as the performance characteristics. When choosing between different solutions that meet your performance needs, make sure that you have considered all aspects of running a storage system and the needs of your organization to select the solution that will provide the maximum overall value.

---

## Chapter 3. Authenticating Users

To make the DGX useful, users need to be added to the system in some fashion so they can be authenticated to use the system. Generally, this is referred to as *user authentication*. There are several different ways this can be accomplished, however, each method has its own pros and cons.

### 3.1. Local

The first way is to create users directly on the DGX system using the `useradd` command. Let's assume you want to add a user `dgxuser`. You would first add the user via the following command.

```
$ useradd -m -s /bin/bash dgxuser
```

Where `-s` refers to the default shell for the user and `-m` creates the user's home directory. After creating the user you need to add them to the `docker` group on the DGX.

```
$ sudo usermod -aG docker dgxuser
```

This adds the user `dgxuser` to the group `docker`. Any user that runs Docker containers has to be a member of this group.

Using authentication on the DGX is simple but not without its issues. First, there have been occasions when an OS upgrade on the DGX requires the reformatting of all the drives in the appliance. If this happens, you first must make sure all user data is copied somewhere off the DGX-1 before the upgrade. Second, you will have to recreate the users and add them to the `docker` group and copy their home data back to the DGX. This adds work and time to upgrading the system.



**Important:** While the 2x 960GB NVME SSDs on the DGX-2, meant for the OS partition, are in RAID-1 configuration, there is no RAID-1 on the OS drive for the DGX-1 and DGX Station. **Hence, if the OS drive fails on the DGX-1 or the DGX Station, you will lose all the users and everything in the `/home` directories. Therefore, it is highly recommended that you backup the pertinent files on the DGX system as well as `/home` for the users.**

### 3.2. NIS Vs NIS+

Another authentication option is to use [NIS or NIS+](#). In this case, the DGX would be a client in the NIS/NIS+ configuration. As with using local authentication as previously discussed, there

is the possibility that the OS drive in the DGX could be overwritten during an upgrade (not all upgrades reformat the drives, but it's possible). This means that the administrator may have to reinstall the NIS configuration on the DGX.

Also, remember that the DGX-1 and DGX Station have a single OS drive. If this drive fails, the administrator will have to re-configure the NIS/NIS+ configuration, therefore, backups are encouraged; even for DGX-2 systems, which do have 2x OS drives in a RAID-1 configuration.



**Note:** It is possible that if, in the unlikely event that technical support for the DGX is needed, the NVIDIA engineers may require the administrator to disconnect from the NIS/NIS+ server.

## 3.3. LDAP

A third option for authentication is [LDAP](#) (Lightweight Directory Access Protocol). It has become very popular in the clustering world, particularly for Linux. You can configure LDAP on the DGX for user information and authentication from an LDAP server. However, as with NIS, there are possible repercussions.



### CAUTION:

- ▶ The first is that the OS drive is a single drive on the DGX-1 and DGX Station. If the drive fails, you will have to rebuild the LDAP configuration (backups are highly recommended).
- ▶ The second is that, as previously mentioned, if, in the unlikely event of needing tech support, you may be asked to disconnect the DGX system from the LDAP server so that the system can be triaged.

## 3.4. Active Directory

One other option for user authentication is connecting the DGX system to an Active Directory (AD) server. This may require the system administrator to install some extra tools into the DGX. This means that this approach should also include the two cautions that were repeated before where the single OS drive may be reformatted for an upgrade or that it may fail (again, backups are highly recommended). It also means that in the unlikely case of needing to involve NVIDIA technical support, you may be asked to take the system off the AD network and remove any added software (this is unlikely but possible).



---

# Chapter 4. Time Synchronization

Time synchronization is very important for clusters of systems including storage. It is especially true for MPI (Message Passing Interface) applications such as those in the HPC world. Without time synchronization, you can get wrong answers or your application can fail. Therefore, it is a good idea to sync the DGX-2, DGX-1, or DGX Station time.

## 4.1. Ubuntu 16.04

If you are using Ubuntu 16.04 as the base for your DGX OS image, realize that it uses `systemd` instead of `init`, so the process of configuring NTP (network time protocol) is a little different than using Ubuntu 14.04. If you are unsure on how to accomplish this, below are some basic instructions.

For more information, you can:

- ▶ Run the following commands:

```
$ man timedatectl command
$ man systemd-timesyncd.service
```

- ▶ Read the [timesyncd.conf](#) article.

Here is an outline of the steps you should follow:

1. Edit the `/etc/systemd/timesyncd.conf` file and set NTP and other options. For more information, see the [timesyncd.conf](#) article.

2. Run as root the following command:

```
$ timedatectl set-ntp true
```

3. Check that `timesyncd` is enabled and run the following command:

```
$ systemctl status systemd-timesyncd.service
```

4. Ensure `timesyncd` is enabled. If `timesyncd` is not enabled, run the following command.

```
$ systemctl enable systemd-timesyncd.service && systemctl start systemd-timesyncd.service
```

You can also check via `timedatectl` that you configured the correct timezone and other basic options.

---

# Chapter 5. Monitoring

Being able to monitor your systems is the first step in being able to manage them. NVIDIA provides some very useful command line tools that can be used specifically for monitoring the GPUs.

## 5.1. DCGM

[NVIDIA Data Center GPU Manager™](#) (DCGM) simplifies GPU administration in the data center. It improves resource reliability and uptime, automates administrative tasks, and helps drive overall infrastructure efficiency. It can perform the following tasks with very low overhead on the appliance.

- ▶ Active health monitoring
- ▶ Diagnostics
- ▶ System validation
- ▶ Policies
- ▶ Power and clock management
- ▶ Group configuration and accounting

The DCGM Toolkit comes with a User Guide that explains how to use the command-line tool called `dcmi`, as well as an API Guide (there is no GUI with DCGM). In addition to the command-line tool, DCGM also comes with headers and libraries for writing your own tools in Python or C.

Rather than treat each GPU as a separate resource, DCGM allows you to group them and then apply policies or tuning options to the group. This also includes being able to run diagnostics on the group.

There are several best practices for using DCGM with the DGX appliances. The first is that the command line tool can run diagnostics on the GPUs. You could create a simple `cron` job on the DGX to check the GPUs and store the results either into a simple flat file or into a simple database.

There are three levels of diagnostics that can be run starting with level 1.

- ▶ Level 1 runs in just a few seconds.

- Level 3 takes about 4 minutes to run. An example of the output from running a level 3 diagnostic is below.

Figure 1. Levels of diagnostics

```
dcgmi diag -r 3
```

D diagnostic	Result
----- Deployment -----	
Blacklist	Pass
NVML Library	Pass
CUDA Main Library	Pass
CUDA Toolkit Libraries	Pass
Permissions and OS Blocks	Pass
Persistence Mode	Pass
Environment Variables	Pass
Page Retirement	Pass
Graphics Processes	Pass
----- Hardware -----	
GPU Memory	Pass - All
Diagnostic	Pass - All
----- Integration -----	
PCIe	Pass - All
----- Performance -----	
SM Performance	Pass - All
Targeted Performance	Pass - All
Targeted Power	Warn - All

It is fairly easy to parse this output looking for Error in the output. You can easily send an email or raise some other alert if an Error is discovered.

A second best practice for utilizing DCGM is if you have a resource manager (in other words, a job scheduler) installed. Before the user's job is run, the resource manager can usually perform what is termed a prologue. That is, any system calls before the user's job is executed. This is a good place to run a quick diagnostic and also use DCGM to start gathering statistics on the job. Below is an example of statistics gathering for a particular job:

Figure 2. Statistics gathering

```
dcgm stats --job demojob -v -g 2
Successfully retrieved statistics for job: demojob.
```

GPU ID: 0	
-----	
----- Execution Stats -----	
Start Time	Wed Mar 9 15:07:34 2016
End Time	Wed Mar 9 15:08:00 2016
Total Execution Time (sec)	25.48
No. of Processes	1
Compute PID	23112
----- Performance Stats -----	
Energy Consumed (Joules)	1437
Max GPU Memory Used (bytes)	120324096
SM Clock (MHz)	Avg: 998, Max: 1177, Min: 405
Memory Clock (MHz)	Avg: 2068, Max: 2505, Min: 324
SM Utilization (%)	Avg: 76, Max: 100, Min: 0
Memory Utilization (%)	Avg: 0, Max: 1, Min: 0
PCIe Rx Bandwidth (megabytes)	Avg: 0, Max: 0, Min: 0
PCIe Tx Bandwidth (megabytes)	Avg: 0, Max: 0, Min: 0
----- Event Stats -----	
Single Bit ECC Errors	0
Double Bit ECC Errors	0
PCIe Replay Warnings	0
Critical XID Errors	0
----- Slowdown Stats -----	
Due to - Power (%)	0
- Thermal (%)	Not Supported
- Reliability (%)	Not Supported
- Board Limit (%)	Not Supported
- Low Utilization (%)	Not Supported
- Sync Boost (%)	0
-----	

When the user's job is complete, the resource manager can run something called an epilogue. This is a place where the system can run some system calls for doing such things as cleaning up the environment or summarizing the results of the run including the GPU stats as from the above command. Consult the user's guide to learn more about stats with DCGM.

If you create a set of prologue and epilogue scripts that run diagnostics you might want to consider storing the results in a flat file or a simple database. This allows you to keep a history of the diagnostics of the GPUs so you can pinpoint any issues (if there are any).

A third way to effectively use DCGM is to combine it with a [parallel shell tool](#) such as [pdsh](#). With a parallel shell you can run the same command across all of the nodes in a cluster or a specific subset of nodes. You can use it to run `dcgm` to run diagnostics across several DGX appliances or a combination of DGX appliances and non-GPU enabled systems. You can easily capture this output and store it in a flat file or a database. Then you can parse the output and create warnings or emails based on the output.

Having all of this diagnostic output is also an excellent source of information for creating reports regarding topics such as utilization.

For more information about DCGM, see [NVIDIA Data Center GPU Manager Simplifies Cluster Administration](#).

## 5.2. Using `ctop` For Monitoring

Containers can make monitoring a little more challenging than the classic system monitoring. One of the classic tools used by system administrators is [top](#). By default, top displays the load on the system as well as the ordered list of processes on the system.

There is a top-like tool for Docker containers and runC, named [ctop](#). It lists real-time metrics for more than one container and is easy to install and update the resource usage for the running containers.



**ATTENTION:** ctop runs on a single DGX-1 system only. Most likely you will have to log into the specific node and run ctop. A best practice is to use [tmux](#) and create a pane for ctop for each DGX appliance if the number of systems is fairly small (approximately less than 10).

## 5.3. Monitoring A Specific DGX Using `nvidia-smi`

As previously discussed, DCGM is a great tool for monitoring GPUs across multiple nodes. Sometimes, a system administrator may want to monitor a specific DGX system in real-time. An easy way to do this is to login into the DGX and run `nvidia-smi` in conjunction with the [watch](#) command.

For example, you could run the command `watch -n 1 nvidia-smi` that runs the `nvidia-smi` command every second (`-n 1` means to run the command with 1 second intervals). You could also add the `-d` option to watch so that it highlights changes or differences since the last time it was run. This allows you to easily see what has changed.

Just like ctop, you can use `nvidia-smi` and `watch` in a pane in a `tmux` terminal to keep an eye on a relatively small number of DGX servers.

---

## Chapter 6. Managing Resources

One of the common questions from DGX customers is how can they effectively share the DGX system between users without any inadvertent problems or data exchange. The generic phrase for this is resource management; the tools are called resource managers. They can also be called schedulers or job schedulers. These terms are often used interchangeably.

Everything on the DGX system can be viewed as a resource. This includes memory, CPUs, GPUs, and even storage. Users submit a request to the resource manager with their requirements and the resource manager assigns the resources to the user if they are available and not being used. Otherwise, the resource manager puts the request in a queue to wait for the resources to become available. When the resources are available, the resource manager assigns the resources to the user request. This request is known as a "job".

A resource manager provides functionality to act on jobs such as starting, canceling, or monitoring them. It manages a queue of jobs for a single cluster of resources, each job using a subset of computing resources. It also monitors resource configuration and health, launching jobs to a single FIFO queue.

A job scheduler ties together multiple resource managers into one integrated domain, managing jobs across all machines in the domain. It implements policy mechanisms to achieve efficient utilization of resources, manages software licenses, and collects and reports resource usage statistics.

Some tools that started as resource managers have graduated to include job scheduler features, making the terms largely synonymous.

Resource managers and job schedulers have been around for a long time and are extensively used in the HPC world. The end of this section will include examples of how to run solutions such as [SLURM](#), [Univa Grid Engine](#), [IBM Spectrum LSF](#), and [Altair PBS Pro](#). If you haven't used these tools before, you should perform some simple experiments first to understand how they work. For example, take a single server and install the software, then try running some simple jobs using the cores on the server. Expand as desired and add more nodes to the cluster.

The following subsections discuss how to install and use a job scheduler on a DGX system. For DGX systems, NVIDIA supports deploying the Slurm or Kubernetes resource managers through the use of DeepOps. DeepOps is a modular collection of ansible scripts which automate the deployment of Kubernetes, Slurm, or a hybrid combination of the two, along with monitoring services and other ancillary functionality to help manage systems.



**ATTENTION:** DGX systems do not come pre-installed with job schedulers, although NPN (NVIDIA Partner Network) partners may elect to install a job scheduler as part of the larger

deployment service. NVIDIA Support may request disabling or removing the job scheduler software for debugging purposes. They may also ask for a factory image to be installed. Without these changes, NVIDIA Support will not be able to continue with debugging process.

## 6.1. Example: SLURM

Slurm is a batch scheduler often used in HPC environments, but is simple to install and flexible in configuration, so has seen wide adoption in a variety of areas. The following are suggested methods for installing SLURM.

- Using [Bright Cluster Manager](#)

Refer also to the [Bright Cluster Manager 9.1 Administrator Manual](#), section 7.5 "Configuring and Running Individual Workload Managers".

- Using [DeepOps](#)

DeepOps is a modular collection of ansible scripts which automate the deployment of Kubernetes, Slurm, or a hybrid combination of the two across. To install Slurm with DeepOps, follow the steps in the [Slurm Deployment Guide](#).

Refer also to the technical blog [Deploying Rich Cluster API on DGX for Multi-User Sharing](#)

- Using SLURM Native GPU Support

SLURM has native GPU support in > v19.05, which can be installed by following the [Slurm Administrator Guide](#).

After Slurm is installed and configured on a DGX-2, DGX-1, or DGX Station, the next step is to plan how to use the DGX system. The first, and by far the easiest, configuration is to assume that a user gets exclusive access to the entire node. In this case the user gets the entire DGX system, i.e. access to all GPUs and CPU cores. No other users can use the resources while the first user is using them.

The second way, is to make the GPUs a consumable resource. The user will then ask for the number of GPUs they need ranging from 1 to 8 for the DGX-1 and 1 to 16 for the DGX-2.

At a high level, there are two basic options for configuring SLURM with GPUs and DGX systems. The first is to use what is called exclusive mode access and the second allows each GPU to be scheduled independently of the others.

### 6.1.1. Simple GPU Scheduling With Exclusive Node Access

If there is no interest in allowing simultaneous multiple jobs per compute node, then Slurm might not need to be aware of the GPUs in the system and the configuration can be greatly simplified.

One way of scheduling GPUs without making use of GRES (Generic Resource Scheduling) is to create partitions or queues for logical groups of GPUs. For example, grouping nodes with V100 GPUs into a V100 partition would result in something like the following:

```
$ sinfo -s
PARTITION AVAIL  TIMELIMIT  NODES (A/I/O/T)  NODELIST
```

```
v100      up    infinite      4/9/3/16  node[212-213,215-218,220-229]
```

The corresponding partition configuration via the SLURM configuration file, `slurm.conf`, would be something like the following:

```
NodeName=node[212-213,215-218,220-229]
PartitionName=v100 Default=NO DefaultTime=01:00:00 State=UP
Nodes=node[212-213,215-218,220-229]
```

If a user requests a node from the `v100` partition, then they would have access to all of the resources in that node, and other users would not. This is what is called *exclusive access*.

This approach can be advantageous if there is concern that sharing resources might result in performance issues on the node or if there are concerns about overloading the node resources. For example, in the case of a DGX-1, if multiple users might overwhelm the 8TB NFS read cache, exclusive mode should be considered. Or if the concern is that users may use all of the physical memory causing page swapping with a corresponding reduction in performance, then exclusive mode might be useful.

## 6.1.2. Scheduling Resources At the Per-GPU Level

A second option for using SLURM, is to treat the GPUs like a consumable resource and allow users to request them in integer units (i.e. 1, 2, 3, etc.). SLURM can be made aware of GPUs as a consumable resource to allow jobs to request any number of GPUs. This feature requires job accounting to be enabled first; for more info, see [Accounting and Resource Limits](#). A very quick overview is below.

The SLURM configuration file, `slurm.conf`, needs parameters set to enable `cgroups` for resource management and GPU resource scheduling. An example is the following:

```
# General
ProctrackType=proctrack/cgroup
TaskPlugin=task/cgroup

# Scheduling
SelectType=select/cons_res
SelectTypeParameters=CR_Core_Memory

# Logging and Accounting
AccountingStorageTRES=gres/gpu
DebugFlags=CPU_Bind,gres          # show detailed information in Slurm logs about GPU
binding and affinity
JobAcctGatherType=jobacct_gather/cgroup
```

The partition information in `slurm.conf` defines the available GPUs for each resource. Here is an example:

```
# Partitions
GresTypes=gpu
NodeName=slurm-node-0[0-1] Gres=gpu:2 CPUs=10 Sockets=1 CoresPerSocket=10 ThreadsPerCore=1
RealMemory=30000 State=UNKNOWN
PartitionName=compute Nodes=ALL Default=YES MaxTime=48:00:00 DefaultTime=04:00:00 MaxNodes=2
State=UP DefMemPerCPU=3000
```

The way that resource management is enforced is through [cgroups](#). The `cgroups` configuration require a separate configuration file, `cgroup.conf`, such as the following:

```
CgroupAutomount=yes
CgroupReleaseAgentDir="/etc/slurm/cgroup"

ConstrainCores=yes
ConstrainDevices=yes
ConstrainRAMSpace=yes
```



```
#TaskAffinity=yes
```

To schedule GPU resources requires a configuration file to define the available GPUs and their CPU affinity. An example configuration file, `gres.conf`, is below:

```
Name=gpu File=/dev/nvidia0 CPUs=0-4
Name=gpu File=/dev/nvidia1 CPUs=5-9
```

To run a job utilizing GPU resources requires using the `--gres` flag with the `srun` command. For example, to run a job requiring a single GPU the following `srun` command can be used.

```
$ srun --gres=gpu:1 nvidia-smi
```

You also may want to restrict memory usage on shared nodes so that a user doesn't cause swapping with other user or system processes. A convenient way to do this is with memory `cgroups`.

Using memory `cgroups` can be used to restrict jobs to allocated memory resources requires setting kernel parameters. On Ubuntu systems this is configurable via the file `/etc/default/grub`.

```
GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
```

## 6.2. Example: Univa Grid Engine

See the document [Using NVIDIA® DGX™ Systems with Univa Grid Engine](#)

## 6.3. Example: IBM Spectrum LSF

See the knowledge base article [Using IBM Spectrum LSF with NVIDIA DGX Systems](#)

## 6.4. Example: Altair PBS Pro

See the following site for a link to the technical whitepaper: [Altair PBS Professional Support on NVIDIA DGX Systems](#)

---

# Chapter 7. Provisioning and Cluster Management

Cluster management tools go beyond resource managers and job schedulers, managing the state of each node in an entire cluster. They typically include mechanisms to provision the nodes in the cluster (install the operating system image, firmware, and drivers), deploy a job scheduler, monitor and manage hardware, configure user access, and make modifications to the software stack.

Provisioning and cluster management of DGX Systems may be bootstrapped with [DeepOps](#). DeepOps is open source and highly modular. It has defaults which can be configured to meet organizational needs and incorporates best practices for deploying GPU-accelerated Kubernetes and Slurm.

Alternatively, Bright Cluster Manager deploys complete DGX PODs over bare metal and manages them effectively. It provides management for the entire DGX POD, including the hardware, operating system, and users. It even manages the Data Analytics software, NGC, Bright Data Science, Kubernetes, Docker and Singularity Containers. With Bright Cluster Manager, a system administrator can quickly stand up DGX PODs and keep them running reliably throughout their life cycle—all with the ease and elegance of a fully-featured, enterprise-grade cluster manager.

## 7.1. Example: Bright Computing Cluster

See the knowledge base article [How do I add NVIDIA DGX nodes to a Bright cluster using the official Ubuntu DGX software stack?](#)

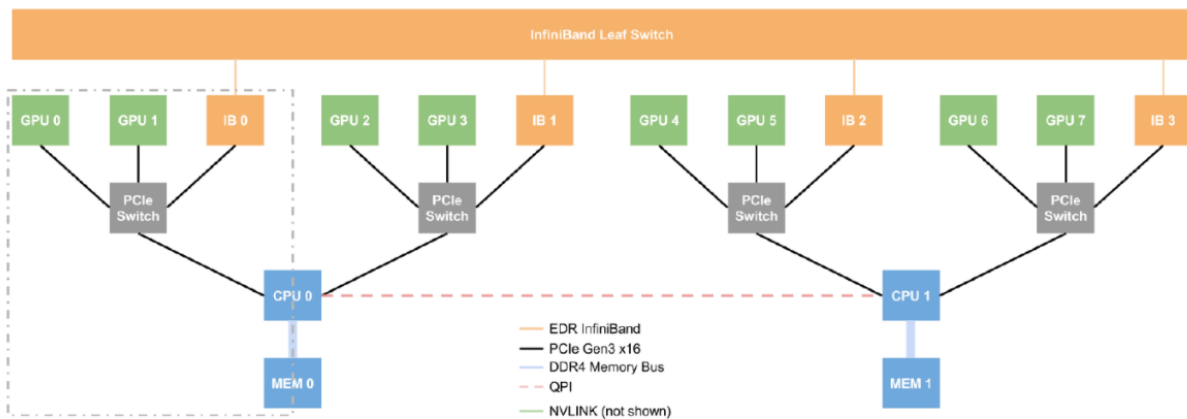
# Chapter 8. Networking

Networking DGX appliances is an important topic because of the need to provide data to the GPUs for processing. GPUs are remarkably faster than CPUs for many tasks, particularly deep learning. Therefore, the network principles used for connecting CPU servers may not be sufficient for DGX appliances. This is particularly important as the number of DGX appliances grows over time.

## 8.1. DGX-1 Networking

To understand best practices for networking the DGX-1 and for planning for future growth, it is best to start with a brief review of the DGX-1 appliance itself. Recall that the DGX-1 comes with four EDR InfiniBand cards (100 Gb/s each) and two 10Gb/s Ethernet cards (copper). These networking interfaces can be used for connecting the DGX-1 to the network for both communications and storage.

Figure 3. Networking interfaces



Notice that every two GPUs are connected to a single PCIe switch that is on the system board. The switch also connects to an InfiniBand (IB) network card. To reduce latency and improve throughput, and network traffic from these two GPUs should go to the associated IB card. This is why there are four IB cards in the DGX-1 appliance.

### 8.1.1. DGX-1 InfiniBand Networking

If you want to use the InfiniBand (IB) network to connect DGX appliances, theoretically, you only have to use one of the IB cards. However, this will push data traffic over the QPI link between the CPUs, which is a very slow link for GPU traffic (i.e. it becomes a bottleneck). A better solution would be to use two IB cards, one connected to each CPU. This could be IB0 and IB2, or IB1 and IB3, or IB0 and IB3, or IB1 and IB2. This would greatly reduce the traffic that has to traverse the QPI link. The best performance is always going to be using all four of the IB links to an IB switch.

The best approach for using IB links to connect all four IB cards to an IB fabric. This will result in the best performance (full bisectional bandwidth and lowest latency) if you are using multiple DGX appliances for training.

Typically, the smallest IB switch comes with 36-ports. This means a single IB switch could accommodate nine (9) DGX-1 appliances using all four IB cards. This allows 400 Gb/s of bandwidth from the DGX-1 to the switch.

If your applications do not need the bandwidth between DGX-1 appliances, you can use two IB connections per DGX-1 as mentioned previously. This allows you to connect up to 18 DGX-1 appliances to a single 36-port IB switch.



**Note:** It is not recommended to use only a single IB card, but if for some reason that is the configuration, then you can connect up to 36 DGX-1 appliances to a single switch.

For larger numbers of DGX-1 appliances, you will likely have to use two levels of switching. The classic HPC configuration is to use 36-port IB switches for the first level (sometimes called leaf switches) and connect them to a single large core switch, which is sometimes called a director class switch. The largest director class InfiniBand switch has 648 ports. You can use more than one core switch but the configuration will get rather complex. If this is something you are considering, please contact your NVIDIA sales team for a discussion.

For two tiers of switching, if all four IB cards per DGX-1 appliance are used to connect to a 36-port switch, and there is no over-subscription, the largest number of DGX-1 appliances per switch is 4. This is 4 ports from each DGX-1 into the switch for a total of 16. Then, there are 16 uplinks from the leaf switch to the core switch (the director class switch). A total of 40x 36-port leaf switches can be connected to the 648-port core switch (648/16). This results in 160 DGX-1 appliances being connected with full bi-sectional bandwidth.

You can also use what is termed over-subscription in designing the IB network. Over-subscription means that the bandwidth from an uplink is less than the bandwidth coming into the unit (in other words, poorer bandwidth performance). If we use 2:1 over-subscription from the DGX-1 appliances to the first level of switches (36-port leaf switches), then each DGX-1 appliance is only using two IB cards to connect to the switches. This results in less bandwidth than if we used all four cards and also higher latency.

If we keep the network bandwidth from the leaf switches to the core directory switch as 1:1 (in other words, no over-subscription, full bi-sectional bandwidth), then we can put nine (9) DGX-1 appliances into a single leaf switch (a total of 18 ports into the leaf switch from the DGX appliances and 18 uplink ports to the core switch). The result is that a total of 36 leaf switches

can be connected to the core switch. This allows a grand total of 324 DGX-1 appliances to be connected together.

You can tailor the IB network even further by using over-subscription from the leaf switches to the core switch. This can be done using four IB connections to a leaf switch from each DGX appliance and then doing 2:1 over-subscription to the core switch or even using two IB connections to the leaf switches and then 2:1 over-subscription to the core switch. These designs are left up to the user to determine but if this is something you want to consider, please contact your NVIDIA sales team for a discussion.

Another important aspect of InfiniBand networking is the Subnet Manager (SM). The SM simply manages the IB network. There is one SM that manages the IB fabric at any one time but you can have [other SM's running](#) and ready to take over if the first SM crashes. Choosing how many SM's to run and where to run them can have a major impact on the design of the cluster.

The first decision to make is where you want to run the SM's. They can be run on the IB switches if you desire. This is called hardware SM since it runs on the switch hardware. The advantage of this is that you do not need any other servers which could also run the SM. Running the SM on a node is called a software SM. A disadvantage to running a hardware SM is that if the IB traffic is large, the SM could have a difficult time. For lots of IB traffic and for larger networks, it is a best practice to use a software SM on a dedicated server.

The second decision to make is how many SM's you want to run. At a minimum, you will have to run one SM. The least expensive solution is to run a single hardware SM. This will work fine for small clusters of DGX-1 appliances (perhaps 2-4). As the number of units grow, you will want to consider running two SM's at the same time to get HA (High Availability) capability. The reason you want HA is that more users are on the cluster and having it go down has a larger impact than just a small number of appliances.

As the number of appliances grow, consider running the SM's on dedicated servers (software SM). You will also want to run at least two SM's for the cluster. Ideally, this means two dedicated servers for the SM's, but there may be a better solution that solves some other problems; a head node.

## 8.1.2. DGX-1 Ethernet Networking

Each DGX-1 system comes with two 10Gb/s NICs. These can be used to connect the systems to the local network for a variety of functions such as logins and storage traffic. As a starting point, it is recommended to push NFS traffic over these NICs to the DGX-1. You should monitor the impact of IO on the performance of your models in this configuration.

If you need to go to more than one level of Ethernet switching to connect all of the DGX-1 units and the storage, be careful of how you configure the network. More than likely, you will have to enable the spanning tree protocol to prevent loops in the network. The spanning tree protocol can impact network performance, therefore, you could see a decrease in application performance.

The InfiniBand NICs that come with the DGX-1 can also be used as Ethernet NICs running TCP. The ports on the cards are QSFP28 so you can plug them into a compatible Ethernet network or a compatible InfiniBand network. You will have to add some software to the appliance and change the networking but you can use the NICs as 100GigE Ethernet cards.

For more information, see [Switch InfiniBand and Ethernet in DGX-1](#).

### 8.1.3. DGX-1 Bonded NICs

The DGX-1 provides two 10GbE ports. Out of the factory these two ports are not bonded but they can be bonded if desired. In particular, VLAN Tagged, Bonded NICs across the two 10 GbE cards can be accomplished.

Before bonding the NICs together, ensure you are familiar with the following:

- ▶ Ensure your network team is involved because you will need to choose a bonding mode for the NICs.
- ▶ Ensure you have a working network connection to pull down the VLAN packages. To do so, first setup a basic, single NIC network (no VLAN/bonding) connection and download the appropriate packages. Then, reconfigure the switch for LACP/VLANs.



**Tip:** Since the networking goes up and down throughout this process, it's easier to work from a remote console.

The process below walks through the steps of an example for bonding the two NICs together.

1. Edit the `/etc/network/interfaces` file to setup an interface on a standard network so that we can access required packages.

```
auto em1
iface em1 inet static
    address 10.253.0.50
    netmask 255.255.255.0
    network 10.253.0.0
    gateway 10.253.0.1
    dns-nameservers 8.8.8.8
```

2. Bring up the updated interface.

```
sudo ifdown em1 && sudo ifup em1
```

3. Pull down the required bonding and VLAN packages.

```
sudo apt-get install vlan
sudo apt-get install ifenslave
```

4. Shut down the networking.

```
sudo stop networking
```

5. Add the following lines to `/etc/modules` to load appropriate drivers.

```
sudo echo "8021q" >> /etc/modules
sudo echo "bonding" >> /etc/modules
```

6. Load the drivers.

```
sudo modprobe 8021q
sudo modprobe bonding
```

7. Reconfigure your `/etc/network/interfaces` file. There are some configuration parameters that will be customer network dependent and you will want to work with one of your network engineers.

The following example creates a bonded network over em1/em2 with IP 172.16.1.11 and VLAN ID 430. You specify the VLAN ID in the NIC name (bond0.###). Also notice that this example uses a bond-mode of 4. Which mode you use is up to you and your situation.

```
auto lo
iface lo inet loopback
```

```
# The following 3 sections create the bond (bond0) and associated network ports (em1,
em2)
auto bond0
iface bond0 inet manual
bond-mode 4
bond-miimon 100
bond-slaves em1 em2

auto em1
iface em1 inet manual
bond-master bond0
bond-primary em1

auto em2
iface em2 inet manual
bond-master bond0

# This section creates a VLAN on top of the bond. The naming format is device.vlan_id
auto bond0.430
iface bond0.430 inet static
address 172.16.1.11
netmask 255.255.255.0
gateway 172.16.1.254
dns-nameservers 172.16.1.254
dns-search company.net
vlan-raw-device bond0
```

8. Restart the networking.

```
sudo start networking
```

9. Bring up the bonded interfaces.

```
ifup bond0
```

10. Engage your network engineers to re-configure LACP and VLANs on switch.
11. Test the configuration.

## 8.2. DGX-2 Networking

Because there are more network devices in the DGX-2 relative to the DGX-1 and DGX Station, and they can be used in different ways, to learn more about DGX-2 networking, see the [DGX-2 User Guide](#).

## Chapter 9. SSH Tunneling

Some environments are not configured or limit access (firewall or otherwise) to computer nodes within an intranet. They are also very useful for running Jupyter notebooks inside containers when working remotely. When running a container with a service or application exposed on a port, such as , remote access must be enabled on the remote system to that port on the DGX system. The following steps use PuTTY to create SSH tunnel from a remote system into the DGX system. If you are using an SSH utility, one can set up tunneling via the `-L` option.



**Note:** A PuTTY SSH tunnel session must be up, logged in, and running for tunnel to function. SSH tunnels are commonly used for the following applications (with listed port numbers).

Table 2. Commonly used applications for SSH tunnels

Application	Port	Notes
	5000	If multiple users, each selects own port
VNC Viewer	5901, 6901	5901 for VNC app, 6901 for web app

To create an SSH Tunnel session with PuTTY, perform the following steps:

1. Run the PuTTY application.
2. In the **Host Name** field, enter the host name you want to connect to.
3. In the **Saved Sessions** section, enter a name to save the session under and click **Save**.
4. Click **Category > Connection**, click **+** next to **SSH** to expand the section.
5. Click **Tunnels** for Tunnel configuration.
6. Add the port for forwarding.
  - a). In the **Source Port** section, enter 5000, which is the port you need to forward for .
7. In the **Destination** section, enter `localhost:5000` for the local port that you will connect to.
8. Click **Add** to save the added Tunnel.
9. In the **Category** section, click **Session**.
10. In the **Saved Sessions** section, click the name you previously created, then click **Save** to save the added Tunnels.



To use `PuTTY` with tunnels, perform the following steps:

1. Run the `PuTTY` application.
2. In the **Saved Sessions** section, select the Save Session that you created.
3. Click **Load**.
4. Click **Open** to start session and login. The SSH tunnel is created and you can connect to a remote system via tunnel. As an example, for , you can start a web browser and connect to `http://localhost:5000`.

---

# Chapter 10. Head Node

A head node is a very useful server within a cluster. Typically, it runs the cluster management software, the resource manager, and any monitoring tools that are used. For smaller clusters, it is also used as a login node for users to create and submit jobs.

For clusters of any size that include the DGX-2, DGX-1, or even a group of DGX Stations, a head node can be very helpful. It allows the DGX systems to focus solely on computing rather than any interactive logins or post-processing that users may be doing. As the number of nodes in a cluster increases, it is recommended to use a head node.

It is recommended to size the head node for things such as:

- ▶ Interactive user logins
- ▶ Resource management (running a job scheduler)
- ▶ Graphical pre-processing and post-processing
  - ▶ Consider a GPU in the head node for visualization
- ▶ Cluster monitoring
- ▶ Cluster management

Since the head node becomes an important part of the operation of the cluster, consider using RAID-1 for the OS drive in the head node as well as redundant power supplies. This can help improve the uptime of the head node.

For smaller clusters, you can also use the head node as an NFS server by adding storage and more memory to the head node and NFS export the storage to the cluster clients. For larger clusters, it is recommended to have dedicated storage, either NFS or a parallel file system.

For InfiniBand networks, the head node can also be used for running the software SM. If you want some HA for the SM, run the primary SM on the head node and use an SM on the IB switch as a secondary SM (hardware SM).

As the cluster grows, it is recommended to consider splitting the login and data processing functions from the head node to one or more dedicated login nodes. This is also true as the number of users grows. You can run the primary SM on the head node and other SM's on the login nodes. You could even use the hardware SM's on the switches as backups.

---

# Chapter 11. DGX-2 KVM Networking

## 11.1. Introduction

The NVIDIA DGX-2 system supports GPU multi-tenancy through the NVIDIA Kernel-based Virtual Machine solution (based on the Linux Kernel Virtual Machine (<https://www.linux-kvm.org>)). This allows different users to run concurrent deep learning jobs using multiple virtual machines (*guest GPU VMs*) within a single DGX-2 System.

This chapter describes the standard and most commonly used network configurations for KVM-based guest GPU VMs running on the NVIDIA® DGX-2™ server. All the network configurations described in this document are based on Netplan - the preferred network configuration method for Ubuntu 18.04-based systems such as the DGX-2 server.

### 11.1.1. Network Configuration Options

The two common network configurations are "Virtual Network" and "Shared Physical Device". The former is identical across all Linux distributions and available out-of-the-box. The latter needs distribution-specific manual configuration.

The type of network configuration suitable for any deployment depends on the following factors:

- ▶ Whether the guest VM needs to be accessible by users outside of the DGX-2 KVM host
- ▶ Type of network services hosted by the guest VM
- ▶ Number of available public IPv4 and IPv6 addresses
- ▶ What kind of security is required for the guest VM
- ▶ The throughput and latency requirements of the guest VM

The rest of this document describes the following network configurations in detail.

- ▶ Virtual Network
- ▶ Bridged Network
- ▶ SR-IOV

## 11.1.2. Acronyms

- ▶ KVM - Linux Kernel based Virtual Machine
- ▶ NAT - Network Address Translation
- ▶ DHCP - Dynamic Host Configuration Protocol
- ▶ SR-IOV - Single Root IO Virtualization
- ▶ QOS - Quality of Service
- ▶ MTU - Maximum Transmission Unit

## 11.2. Virtual Networking

Libvirt virtual networking uses the concept of a virtual network switch, also known as Usermode Networking. A virtual network switch is a software construct that operates on a physical server host to which guest VMs connect. By default, it operates in NAT mode. The network traffic for a guest VM is directed through this switch, and consequently all guest VMs will use the Host IP address of the connected Physical NIC interface when communicating with the external world.

### 11.2.1. Default Configuration

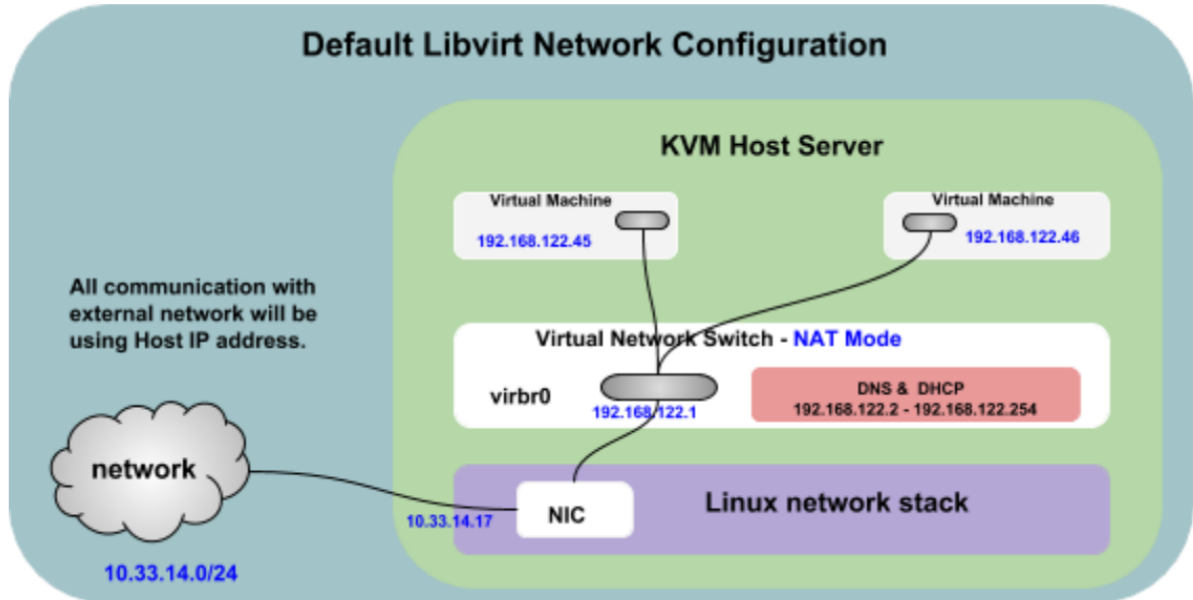
The Linux host physical server represents a virtual network switch as a network interface. When the libvirtd daemon (libvirtd) is first installed and started, the default network interface representing the virtual network switch is virbr0.

By default, an instance of the dnsmasq server is automatically configured and started by libvirt for each virtual network switch that needs it. It is responsible for running a DHCP server (to decide which IP address to lease to each VM) and a DNS server (to respond to queries from VMs).

In the default virtual network switch configuration, the guest OS will get an IP address in the 192.168.122.0/24 address space and the host OS will be reachable at 192.168.122.1. You should be able to SSH into the host OS (at 192.168.122.1) from inside the guest OS and use SCP to copy files back and forth.

In the default configuration, the guest OS will have access to network services but will not itself be visible to other machines on the network. For example, the guest VM will be able to browse the web, but will not be able to host an accessible web server.

You can create additional virtual networks using the steps described in the latter part of this section except you must use a different range of DHCP IP addresses. For example, 192.168.123.0/24.



The following are limitations of the Virtual Network Configuration when used in NAT mode.

- ▶ Guest VMs do not communicate with an external network through a unique address.
- ▶ Guest VMs communicate with an external network through the Host IP address of the connected Physical NIC interface.

When used in NAT mode, you may encounter certain restrictions (such as connection timeouts) due to the number of active connections per Host/IP address, especially if all guest VMs are communicating with the same server at the same time. It also depends on the features and restrictions enforced on the server side.

```
net/http: request canceled while waiting for connection(Client.Timeout exceeded
while awaiting headers)
```

If the default configuration is suitable for your purposes, no other configuration is required.

A couple of advance virtual network configurations can be used for better network performance. Refer to [Improving Network Performance](#) for more details.

### Verifying the Host Configuration

Every standard libvirt installation provides NAT-based connectivity to virtual machines out of the box. This is referred to as the 'default virtual network'. Verify that it is available with the `virsh net-list --all` command.

```
$ virsh net-list --all
Name                State      Autostart
-----
default             active     yes
```

If the default network is missing, the following example XML configuration file can be reloaded and activated.

```
$ virsh net-dumpxml default
<network>
  <name>default</name>
  <uuid>92d49672-3020-40a1-90f5-73fe07216122</uuid>
  <forward mode='nat'>
    <nat>
```

```

    <port start='1024' end='65535' />
  </nat>
</forward>
<bridge name='virbr0' stp='on' delay='0' />
<mac address='52:54:00:40:cc:23' />
<ip address='192.168.122.1' netmask='255.255.255.0'>
  <dhcp>
    <range start='192.168.122.2' end='192.168.122.254' />
  </dhcp>
</ip>
</network>

```

In the above XML contents, “default” is the name of the virtual network, and “virbr0” is the name of the virtual network switch.

```
$ virsh net-define /etc/libvirt/qemu/networks/default.xml
```

```
The default network is defined from /etc/libvirt/qemu/networks/default.xml
```

Mark the default network to automatically start:

```
$ virsh net-autostart default
```

```
Network default marked as autostarted
```

Start the default network:

```
$ virsh net-start defaultNetwork default started
```

Once the libvirt default virtual network is running, you will see a virtual network switch device. This device does not have any physical interfaces added, since it uses NAT and IP forwarding to connect to the outside world. This virtual network switch will just use whatever Physical NIC interface that is being used by Host. Do not add new interfaces.

```
$ brctl show
```

bridge name	bridge id	STP enabled	interfaces
virbr0	8000.0000000000000	yes	

Once the host configuration is complete, a guest can be connected to the virtual network based on its name or bridge. To connect a guest VM to using virtual bridge name “virbr0”, the following XML can be used in the virsh configuration for the guest VM:

```

<interface type='bridge'>
  <source bridge='virbr0' />
  <model type='virtio' />
</interface>

```

## 11.2.2. Using Static IP

You can reserve and allocate static IP addresses for the specific guest VMs from the default DHCP range (192.168.122.2 - 192.168.122.254) of the virtual network switch. Also, you should exclude those reserved/assigned static IP addresses from the DHCP ranges.

### Configurations Made from the Host

To use static IP addressing, check the Mac address of the guest VM.

```
$ virsh edit lgpu-vm-1g0
```

```

<domain type='kvm' id='3'>
  <name>lgpu-vm-1g0</name>
  <uuid>c40f6b9d-ea15-45b0-ac42-83801eef73d4</uuid>
  .....
  <interface type='bridge'>
    <mac address='52:54:00:e1:28:3e' />
    <source bridge='virbr0' />
    <model type='virtio' />
    <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0' />
  </interface>

```

```

...
</domain>
$ virsh net-edit default
<network>
  <name>default</name>
  <uuid>92d49672-3020-40a1-90f5-73fe07216122</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535' />
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0' />
  <mac address='52:54:00:40:cc:23' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.100' end='192.168.122.254' />
      <host mac='52:54:00:e1:28:3e' ip='192.168.122.45' />
    </dhcp>
  </ip>
</network>
$ virsh net-destroy default
$ virsh net-start default

```

Start/restart the guest VM after updating the “default” virtual network with the guest Mac address.

```

$ virsh net-dhcp-leases default

```

Expiry Time	MAC address	Protocol	IP address	Hostname	Client
ID or DUID					
2018-08-29 13:18:58	52:54:00:e1:28:3e	ipv4	192.168.122.45/24	lgpu-vm-1g0	

### 11.2.3. Binding the Virtual Network to a Specific Physical NIC

KVM will use *virtual network switch* as the default networking configuration for all guest VMs and it will operate in NAT mode. The network traffic for a guest is directed through this switch, and consequently all guests will use one of the Host Physical NIC interface while communicating with the external world. By default, it is not bound to any specific physical NIC interface but you can restrict the virtual network switch to use a specific physical NIC interface; for example, you can limit the virtual network to use `enp6s0` only.

```

$ virsh net-edit default
<network>
  <name>default</name>
  <uuid>92d49672-3020-40a1-90f5-73fe07216122</uuid>
  <forward dev='enp6s0' mode='nat' />
  <bridge name='virbr0' stp='on' delay='0' />
  <mac address='52:54:00:40:cc:23' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
    </dhcp>
  </ip>
</network>
$ virsh net-destroy default
$ virsh net-start default

```

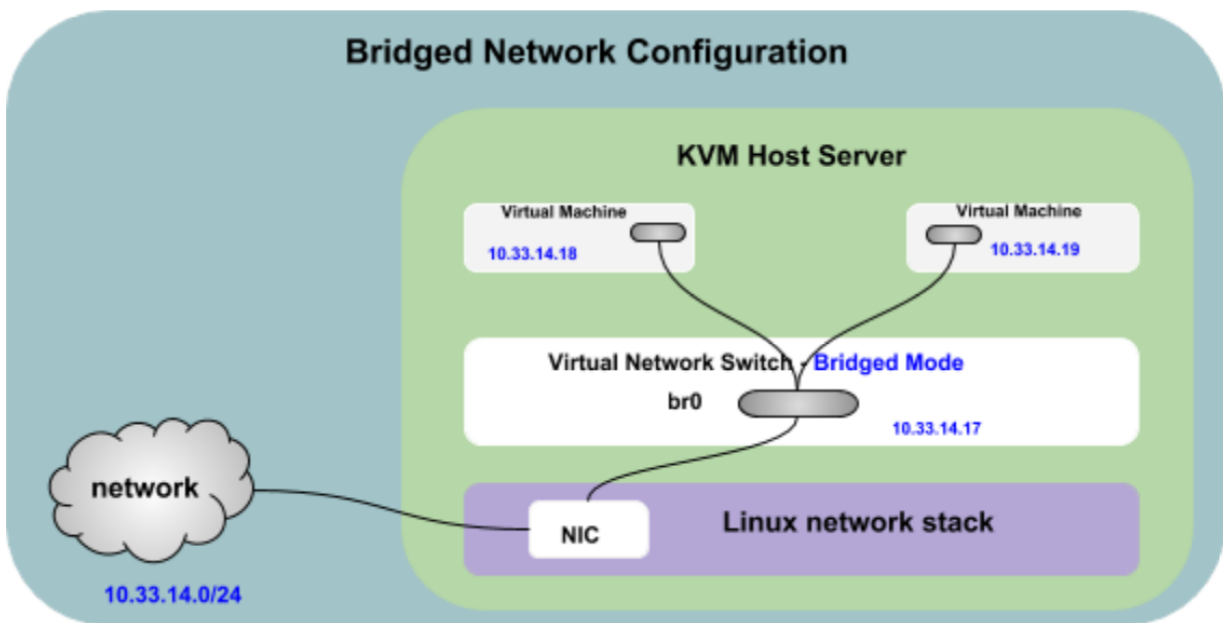
Start/restart the guest after updating “default” virtual network configuration.

## 11.3. Bridged Networking

### 11.3.1. Introduction

A bridged network shares a real Ethernet device with KVM guest VMs. When using *Bridged mode*, all the guest virtual machines appear within the same subnet as the host physical machine. All other physical machines on the same physical network are aware of, and can access, the virtual machines. Bridging operates on Layer 2 of the OSI networking model.

Each guest VM can bind directly to any available IPv4 or IPv6 addresses on the LAN, just like a physical server. Bridging offers the best performance with the least complication out of all the libvirt network types. A bridge is only possible when there are enough IP addresses to allocate one per guest VM. This is not a problem for IPv6, as hosting providers usually provide many free IPv6 addresses. However, extra IPv4 addresses are rarely free.



### 11.3.2. Using DHCP

#### Configuration from the Host

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp134s0f0:
      dhcp4: yes
  bridges:
    br0:
      dhcp4: yes
```



```
interfaces: [ enp134s0f0 ]
```



**Note:** Use the Host NIC interface (Ex: enp134s0f0) that is connected on your system.

```
$ sudo netplan apply
```

### Guest VM Configuration

Once the host configuration is complete, a guest can be connected to the bridged network based on its name. To connect a guest to the 'br0' network, the following XML can be used for the guest:

```
$ virsh edit <VM name or ID>
<interface type='bridge'>
  <source bridge=br0/>
  <model type='virtio'/></interface>
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

## 11.3.3. Using Static IP

### Host Configuration

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernet:
    enp134s0f0:
      dhcp4: no
  bridges:
    br0:
      dhcp4: no
      addresses: [ 10.33.14.17/24 ]
      gateway4: 10.33.14.1
      nameservers:
        search: [ nvidia.com ]
        addresses: [ 172.16.200.26, 172.17.188.26 ]
      interfaces: [ enp134s0f0 ]
```



**Note:** Use the Host NIC interface (Ex: enp134s0f0) that you have connected to your network. Consult your network administrator for the actual IP addresses of your guest VM.

```
$ sudo netplan apply
```

### Guest VM Configuration

Once the host configuration is complete, a guest VM can be connected to the bridged network based on its name. To connect a guest VM to the 'br0' network, the following XML can be used for the guest VM:

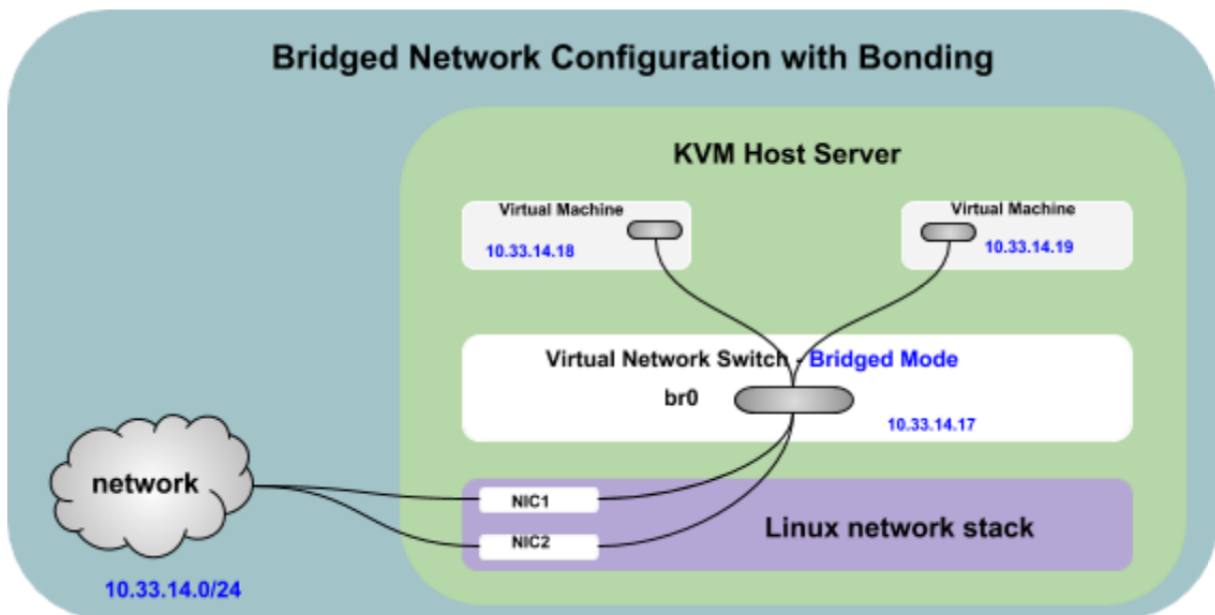
```
$ virsh edit <VM name or ID>
<interface type='bridge'>
  <source bridge=br0/>
  <model type='virtio'/>
</interface>
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

## 11.4. Bridged Networking with Bonding

### 11.4.1. Introduction

Network bonding refers to the combining of multiple physical network interfaces on one host for redundancy and/or increased throughput. Redundancy is the key factor: we want to protect our virtualized environment from loss of service due to failure of a single physical link. This network bonding is the same as Linux network bonding. The bond is added to a bridge and then guest virtual machines are added onto the bridge, similar to bridged mode as discussed in [Bridged Networking](#). However, the bonding driver has several modes of operation, and only a few of these modes work with a bridge where virtual guest machines are in use.



There are three key modes of network bonding:

- ▶ **Active-Passive:** there is one NIC active while another NIC is asleep. If the active NIC goes down, another NIC becomes active.
- ▶ **Link Aggregation:** aggregated NICs act as one NIC which results in a higher throughput.
- ▶ **Load Balanced:** the network traffic is equally balanced over the NICs of the machine.

The following section explains the bonding configuration based on IEEE 802.3 link aggregation. This mode is also known as a Dynamic Link Aggregation mode that creates aggregation groups having the same speed. It requires a switch that supports IEEE 802.3ad Dynamic Link aggregation.

### 11.4.2. Using DHCP

#### Configuration from the Host

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    bond-ports:
      dhcp4: no
      match:
        name: enp134*
  bonds:
    bond0:
      dhcp4: no
      interfaces: [ bond-ports ]
      parameters:
        mode: 802.3ad
  bridges:
    br0:
      dhcp4: yes
      interfaces: [ bond0 ]
```



**Note:** Use the Host NIC interface (Ex: enp134\*) based on what is connected on your system.

```
$ sudo netplan apply
```

## Guest VM Configuration

Once the host configuration is complete, a guest can be connected to the bridged network based on its name. To connect a guest to the 'br0' network, the following XML can be used in the guest:

```
$ virsh edit <VM name or ID>
<interface type='bridge'>
  <source bridge=br0/>
  <model type='virtio'/>
</interface>
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

## 11.4.3. Using Static IP

### Host Configuration

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    bond-ports:
      dhcp4: no
      match:
        name: enp134*
  bonds:
    bond0:
      dhcp4: no
      interfaces: [ bond-ports ]
      parameters:
        mode: 802.3ad
```

```
bridges:
  br0:
    addresses: [ 10.33.14.17/24 ]
    gateway4: 10.33.14.1
    nameservers:
      search: [ nvidia.com ]
      addresses: [ 172.16.200.26, 172.17.188.26 ]
    interfaces: [ bond0 ]
```



**Note:** Use the Host NIC interface (Ex: enp134\*) based on what is connected on your system.

```
$ sudo netplan apply
```

## Guest VM Configuration

Once the host configuration is complete, a guest can be connected to the bridged network based on its name. To connect a guest to the 'br0' network, the following XML can be used in the guest:

```
$ virsh edit <VM name or ID>
<interface type='bridge'>
  <source bridge=br0/>
  <model type='virtio'/>
</interface>
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

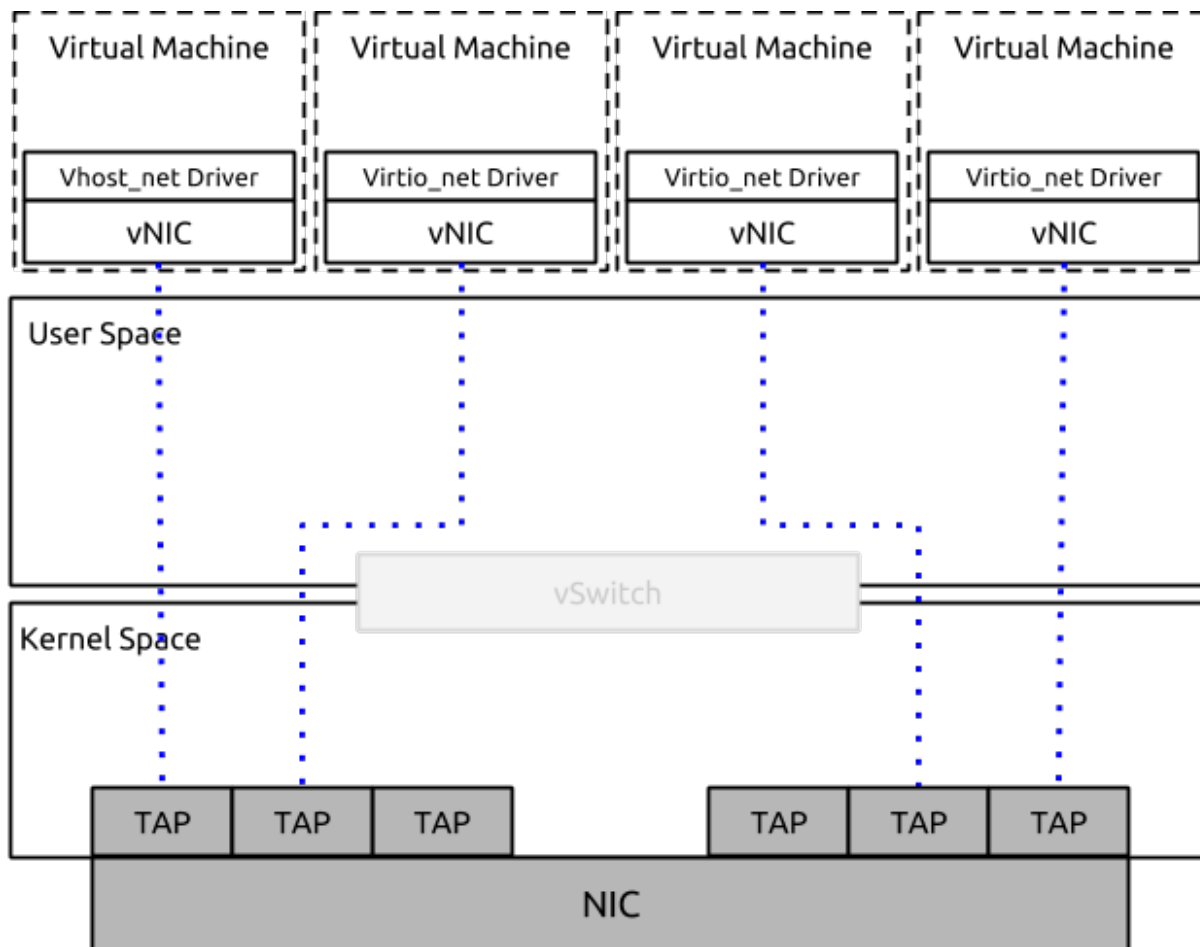
# 11.5. MacVTap

## 11.5.1. Introduction

As an alternative to the default NAT connection, you can use the *macvtap* driver to attach the guest's NIC directly to a specified physical interface of the host machine. Macvtap is a Linux device driver, based upon the combination of Macvlan and a network Terminal Access Point (TAP)(descriptions below), that allows for the creation of virtual (tap-like) interfaces. Each virtual network interface is assigned its own MAC and IP address, then attached to the physical interface (also known as the lower interface),

- ▶ Macvlan - Linux kernel driver that makes it possible to create virtual network interfaces that can be attached to the physical network adapter (aka the lower interface).
- ▶ TAP - A software-only interface that allows user space programs to read and write via TAP device files (/dev/tapN).

A key difference between using a bridge and using macvtap is that macvtap connects directly to the network interface in the KVM host. This direct connection effectively shortens the code path by bypassing much of the code and components in the KVM host associated with connecting to and using a software bridge. This shorter code path usually improves throughput and reduces latencies to external systems.

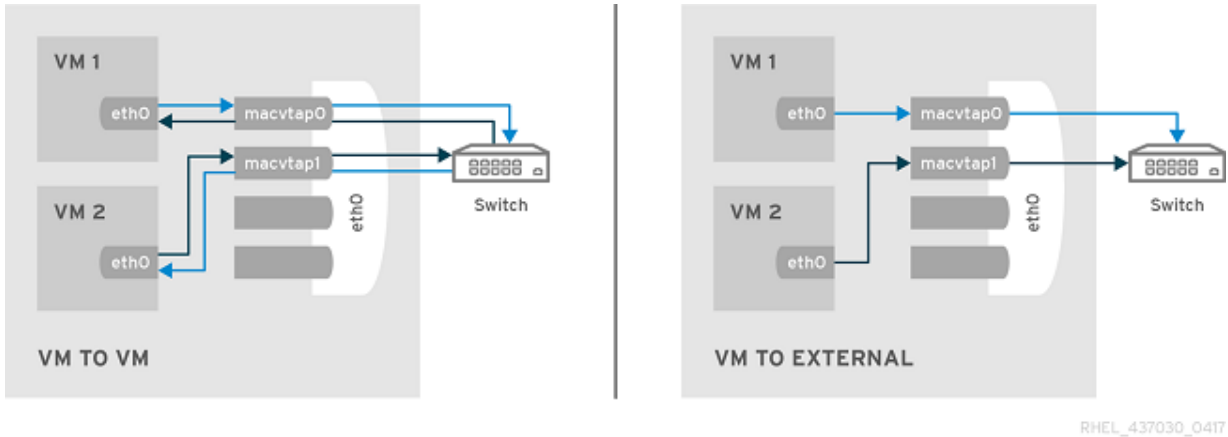


## 11.5.2. Macvtap Modes

There are four modes of operation that control how the endpoints communicate with each other - VEPA, Bridge, Private, and Passthrough.

### VEPA

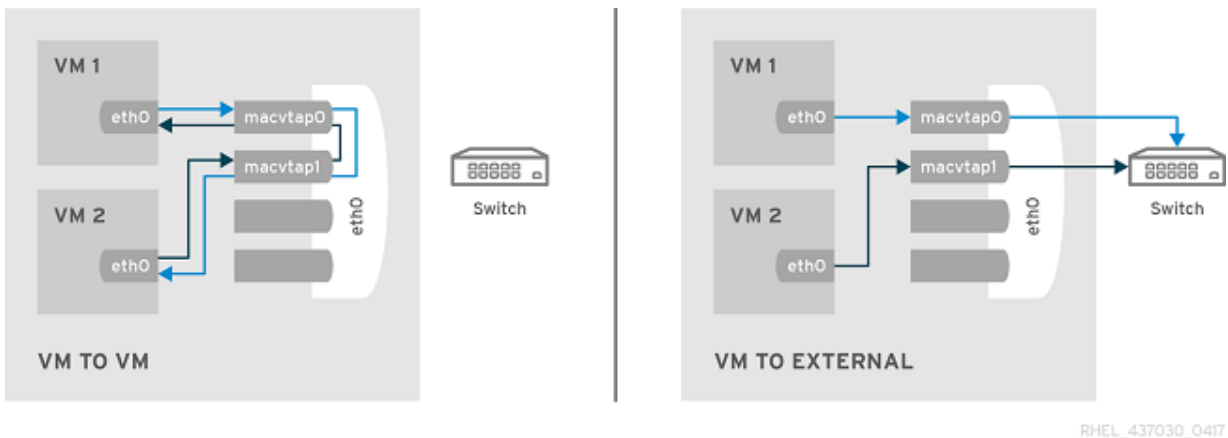
Virtual Ethernet Port Aggregator (**VEPA**) is typically the default mode. Data flows from one endpoint down through the source device in the KVM host out to the external switch. If the switch supports hairpin mode, the data is sent back to the source device in the KVM host and from there sent to the destination endpoint.



RHEL\_437030\_Q417

## Bridge

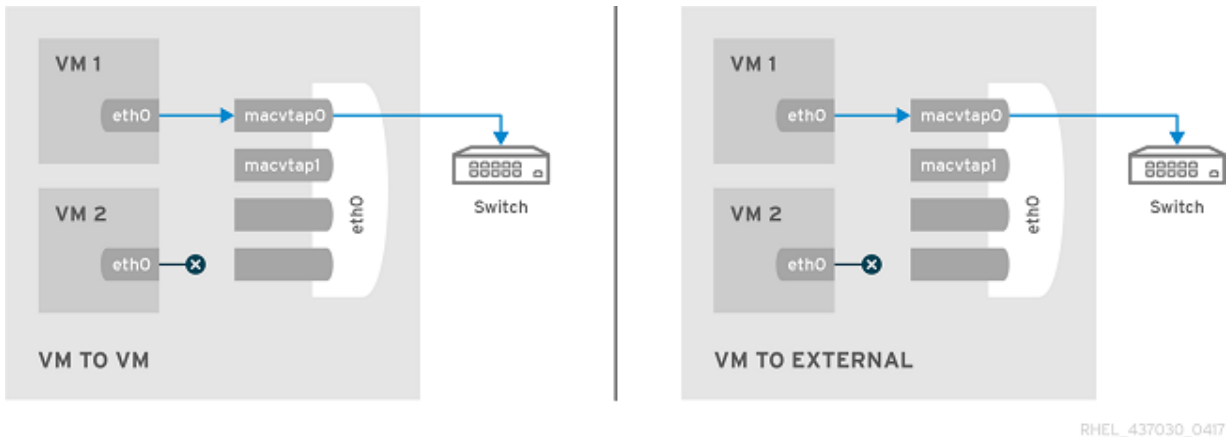
In bridged mode, all endpoints are connected directly to each other. Two endpoints that are both in bridge mode can exchange frames directly, without the round trip through the external bridge. This is the most useful mode for setups with classic switches, and when inter-guest communication is performance critical.



RHEL\_437030\_Q417

## Private

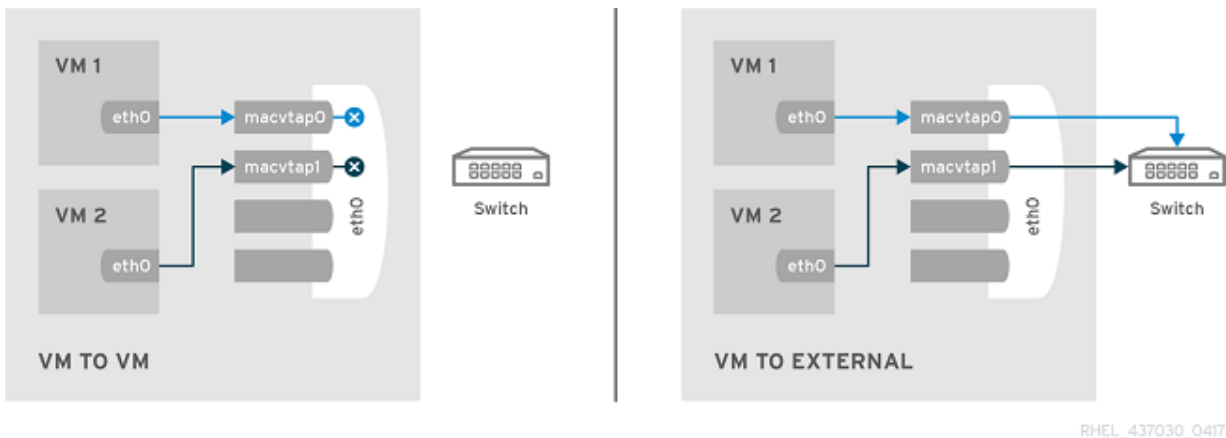
All packets are sent to the external switch and are delivered to a target guest on the same host machine only if they are sent through an external router or gateway. The packets are then sent back to the host. Private mode can be used to prevent individual guests on the single host from communicating with each other.



RHEL\_437030\_Q417

## Passthrough

This feature attaches a physical interface device or a [SR-IOV](#) Virtual Function (VF) directly to a guest without losing the migration capability. All packets are sent directly to the designated network device. Note that a single network device can only be passed through to a single guest, as a network device cannot be shared between guests in passthrough mode.



RHEL\_437030\_Q417

Without a switch that supports hairpin mode, KVM guests configured to use VEPA mode will work the same as bridge mode, and will not be able to communicate directly with the KVM host using the KVM host interface. This limitation can be overcome if the KVM host has multiple interfaces using different ethernet segments (subnets).

MacVTap Modes	VM<->VM	VM<->External	VM<->HOST<->VM	Comment
Vepa	YES/NO	YES	YES/NO	YES only if External Switch supports hairpin mode

MacVTap Modes	VM<->VM	VM<->External	VM<->HOST<->VM	Comment
Bridge	YES	YES	<u>NO</u>	Recommended configuration.  <a href="#">In this configuration the host cannot connect with VMs.</a>
Private	NO	YES	NO	External switch without hairpin mode
Pass-through	YES	YES	YES?	SR-IOV and Non-SRIOV NICs

### 11.5.3. How to Change the Macvtap and Physical NIC Configuration

By default, DGX systems in KVM guests are configured to use a macvtap network in “Bridge” mode. Use the following commands to change the physical NIC and macvtap mode.

1. Edit macvtap-net.

```
$ virsh net-edit macvtap-net<network>
<name>macvtap-net</name>
<uuid>8b403750-2ad5-49df-8a7b-26b10053429d</uuid>
<forward dev='<device-interface>' mode='<macvtap-mode>'
<interface dev='<device-interface>' />
</forward>
</network>
```

Where

*<device-interface>* is the name of the network port, such as enp1s0f0.

*<macvtap-mode>* is the mode you want to set.

- ▶ bridge = Bridge mode
- ▶ private = Private mode
- ▶ vepa = VEPA mode
- ▶ passthrough = passthrough mode

2. Restart macvtap-net with the following commands.

```
$ virsh net-destroy macvtap-net
$ virsh net-start macvtap-net
```



## 11.5.4. How to Configure the Guest VM Using privateIP

If macvtap is configured in "Bridge" mode but you need Host-to-VM network connectivity, you can configure a privateIP network for the VM as follows.

- Configuring privateIP when creating new guest VMs

Specify `--privateIP` when creating the VM so that the second virtual network interface will be added based on private-net network for Host-to-VM connectivity.

- Configuring privateIP for existing guest VMs

1. Edit the VM (`virsh edit <vm-name>`) and add the following line to the `<devices>` section:

```
<interface type='network'>
  <source network='private-net' />
  <model type='virtio' />
</interface>
```

2. Shutdown and restart the VM.

```
$ virsh shutdown <vm-name>
$ virsh start <vm-name>
```

## 11.6. SR-IOV

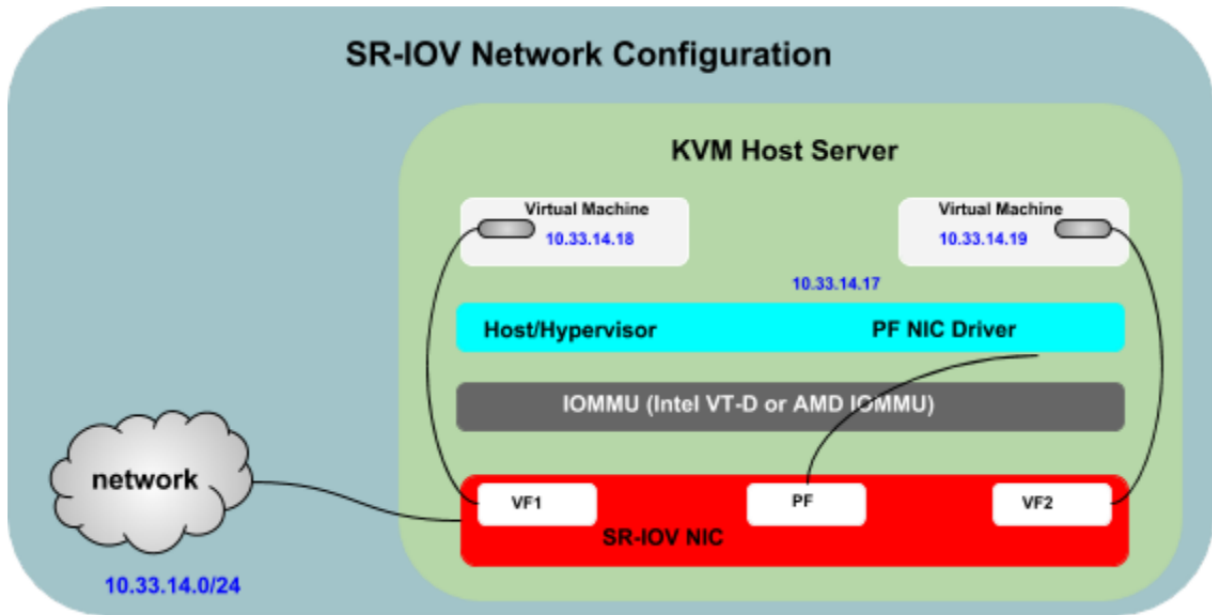
### 11.6.1. Introduction

The SR-IOV technology is a hardware-based virtualization solution that improves both performance and scalability. The SR-IOV standard enables efficient sharing of PCIe (Peripheral Component Interconnect) Express devices among virtual machines and is implemented in the hardware to achieve I/O performance which is comparable to native performance. The SR-IOV specification defines a new standard wherein the new devices that are created will enable the virtual machine to be directly connected to the I/O device.

The SR-IOV specification is defined and maintained by PCI-SIG at <http://www.pcisig.com>.

A single I/O resource can be shared by many virtual machines. The shared devices will provide dedicated resources and also utilize shared common resources. In this way, each virtual machine will have access to unique resources. Therefore, a PCIe device, such as an Ethernet Port, that is SR-IOV enabled with appropriate hardware and OS support can appear as multiple, separate physical devices, each with its own configuration space.

The following figure illustrates the SR-IOV technology for PCIe hardware.



Two new function types in SR-IOV are:

### Physical Function (PF)

A PCI Function that supports the SR-IOV capabilities as defined in SR-IOV specification. A PF contains the SR-IOV capability structure and is used to manage the SR-IOV functionality. PFs are fully-featured PCIe functions that can be discovered, managed, and manipulated like any other PCIe device. PFs have full configuration resources and can be used to configure or control the PCIe device.

### Virtual Function (VF)

A Virtual Function is a function that is associated with a Physical Function. A VF is a lightweight PCIe function that shares one or more physical resources with the Physical Function and with other VFs that are associated with the same PF. VFs are only allowed to have configuration resources for its own behavior.

An SR-IOV device can have hundreds of Virtual Functions (VFs) associating with a Physical Function (PF). The creation of VFs can be dynamically controlled by the PF through registers designed to turn on the SR-IOV capability. By default, the SR-IOV capability is turned off, and the PF behaves as traditional PCIe device.

The following are the advantages and disadvantages of SR-IOV.

#### ► Advantages

- Performance – Direct access to hardware from virtual machines environment and benefits include:
  - Lower CPU utilization
  - Lower network latency
  - Higher network throughput
- Cost Reduction – Capital and operational expenditure savings include:

- ▶ Power savings
- ▶ Reduced adapter count
- ▶ Less cabling
- ▶ Fewer switch ports
- ▶ **Disadvantages**
  - ▶ Guest VM Migration - Harder to migrate guest from one physical server to another

There are several proposals being used or implemented in the industry and each has its own merit/demerits.

## 11.6.2. Device Configuration

SR-IOV and VFs are not enabled by default in all devices. For example, the dual port 100GbE Mellanox card in the DGX-2 doesn't have VFs enabled by default. Follow the instructions in section 5 of the [Mellanox SR-IOV NIC Configuration](#) guide to enable the SR-IOV and the desired number of functions in firmware.

## 11.6.3. Generic Configuration

Use the following steps to enable SR-IOV in KVM host, as it will define a pool of virtual function (VF) devices associated with a physical NIC and automatically assign VF device to each guest from the pool to VF BDFs.

### Configuration from the Host

1. Define a network for a pool of VFs.
2. Read the supported number of VFs.

```
$ cat /sys/class/net/enp134s0f0/device/sriov_totalvfs63
```

3. Enable the required number of VFs (Ex: 16).

```
$ sudo echo 16 > /sys/class/net/enp134s0f0/device/sriov_numvfs
```

4. Create a new SR-IOV network.

Generate an XML file with text similar to the following example.

```
$ sudo vi /etc/libvirt/qemu/networks/iovnet0.xml
```

```
<network>
  <name>iovnet0</name>
  <forward mode='hostdev' managed='yes'>
    <pf dev='enp134s0f0' />
  </forward>
</network>
```



**Note:** Note: Change the value of pf dev to the ethdev (Ex: enp134s0f0) corresponding to you SR-IOV device's physical function.

5. Execute the following commands

```
$ virsh net-define /etc/libvirt/qemu/networks/iovnet0.xml
```

```
$ virsh net-autostart iovnet0
```

```
$ virsh net-start iovnet0
```

Guest VM Configuration

After the defining and starting SR-IOV (iovnet0) network, modify the guest XML definition to specify the network.

```
$ virsh edit <VM name or ID>
<interface type='network'>    <source network='iovnet0' /> </interface>
```

When the guest VM starts, a VF is automatically assigned to the guest VM. If the guest VM is already running, you need to restart it.

### Guest VM Configuration

After the defining and starting SR-IOV (iovnet0) network, modify the guest XML definition to specify the network.

```
$ virsh edit <VM name or ID>
<interface type='network'>
  <source network='iovnet0' />
</interface>
```

When the guest VM starts, a VF is automatically assigned to the guest VM. If the guest VM is already running, you need to restart it.

## 11.6.4. Using DHCP

### Configuration from the Host

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp134s0f0:
      dhcp4: yes
```



**Note:** Use the Host NIC interface (Ex: enp134s0f0) based on what is connected on your system.

```
$ sudo netplan apply
```

### Guest VM Configuration

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp8s0:
      dhcp4: yes
```



**Note:** Use the guest VM NIC interface (Ex: enp8s0) by checking “ifconfig -a” output.

```
$ sudo netplan apply
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

## 11.6.5. Using Static IP

### Configuration from the Host

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp134s0f0:
      dhcp4: no
      addresses: [ 10.33.14.17/24 ]
      gateway4: 10.33.14.1
      nameservers:
        search: [ nvidia.com ]
        addresses: [ 172.16.200.26, 172.17.188.26 ]
```



**Note:** Use Host NIC interface (Ex: enp134s0f0) based on what is being connected on your system.

```
$ sudo netplan apply
```

### Guest VM Configuration

```
$ sudo vi /etc/netplan/01-netcfg.yaml
# This file describes the network interfaces available on your system
# For more information, see netplan(5).
network:
  version: 2
  renderer: networkd
  ethernets:
    enp8s0:
      dhcp4: no
      addresses: [ 10.33.14.18/24 ]
      gateway4: 10.33.14.1
      nameservers:
        search: [ nvidia.com ]
        addresses: [ 172.16.200.26, 172.17.188.26 ]
```



**Note:** Use guest VM NIC interface (Ex: enp8s0) by checking “ifconfig -a” output.

```
$ sudo netplan apply
```

Refer to [Getting the Guest VM IP Address](#) for instructions on how to determine the guest VM IP address.

## 11.7. Getting the Guest VM IP Address

If you are using Bridged and SR-IOV network configurations, use the following steps to determine the guest VM IP address from the Host.

Install and configure [QEMU Guest Agent](#) to retrieve the guest VM IP address. The QEMU guest agent runs inside the guest VM and allows the host machine to issue commands to the guest

VM operating system using libvirt. The guest VM operating system then responds to those commands asynchronously.



**Note:** Note: It is only safe to rely on the guest agent when run by trusted guests. An untrusted guest may maliciously ignore or abuse the guest agent protocol, and although built-in safeguards exist to prevent a denial of service attack on the host, the host requires guest co-operation for operations to run as expected.

## Configuration from the Host

Add the following lines to guest VM XML file under <devices> using

```
$ virsh edit <VM name or ID>
    <channel type='unix'>
        <target type='virtio' name='org.qemu.guest_agent.0' />
    </channel>
```

## Guest VM Configuration

```
$ sudo apt-get install qemu-guest-agent
$ virsh shutdown <VM name or ID>
$ virsh start <VM name or ID>
```

After these steps, run the following command in the Host to check a specific guest VM IP address.

```
$ virsh domifaddr <VM name or ID> --source agent
```

Name	MAC address	Protocol	Address
lo	00:00:00:00:00:00	ipv4	127.0.0.1/8
-	-	ipv6	::1/128
enp1s0	52:54:00:b2:d9:a7	ipv4	10.33.14.18/24
-	-	ipv6	fe80::5054:ff:feb2:d9a7/64
docker0	02:42:3e:48:87:61	ipv4	172.17.0.1/16

# 11.8. Improving Network Performance

This section describes some ways to improve network performance.

## 11.8.1. Jumbo Frames

A jumbo frame is an Ethernet frame with a payload greater than the standard maximum transmission unit (MTU) of 1,500 bytes. Jumbo frames are used on local area networks that support at least [1 Gbps](#) and can be as large as 9,000 bytes. Enabling jumbo frames can improve network performance by making data transmissions more efficient. The CPUs on Switches and Routers can only process one frame at a time. By putting a larger payload into each frame, the CPUs have fewer frames to process. Jumbo frames should be enabled only if each link in the network path, including servers and endpoints, is configured to enable jumbo frames at the same MTU. Otherwise, performance may decrease as incompatible devices drop frames or fragment them; the latter which can task the CPU with higher processing requirements.

In the case of a libvirt-managed network (one with forward mode of NAT, Route), this will be the MTU assigned to the bridge device (virbr0) when libvirt creates it, and thereafter also

assigned to all tap devices created to connect guest interfaces. If MTU is unspecified, the default setting for the type of device being used is assumed and it is usually set to 1500 bytes.

We can enable jumbo frame configuration for the default virtual network switch using the following commands. All guest virtual network interfaces will inherit jumbo frame or MTU of 9000 Bytes configuration.

```
$ virsh net-edit default
<network>
  <name>default</name>
  <uuid>a47b420d-608e-499a-96e4-e75fc45e60c4</uuid>
  <forward mode='nat'>/>
  <bridge name='virbr0' stp='on' delay='0'>/>
  <mtu size='9000'>/>
  <mac address='52:54:00:f2:e3:2a'>/>
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254'>/>
    </dhcp>
  </ip>
</network>
$ virsh net-destroy default
$ virsh net-start default
```

## 11.8.2. Multi-Queue Support

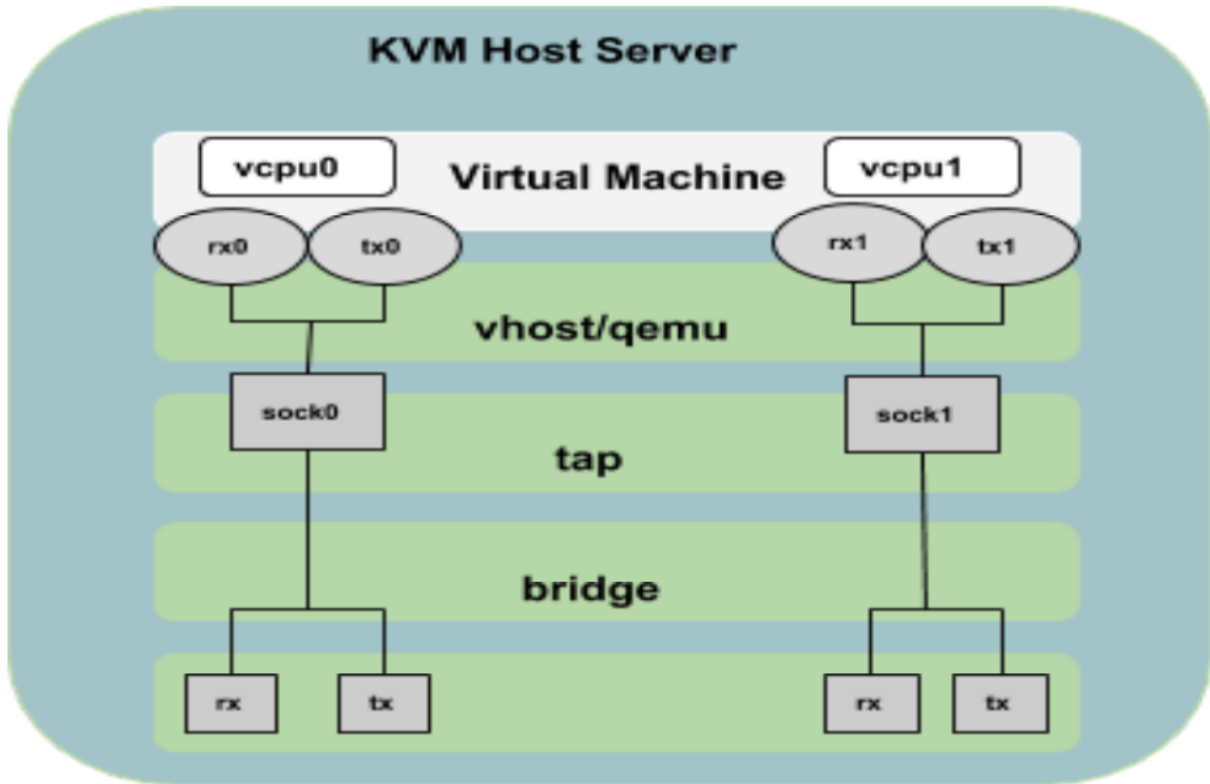
This section describes multi-queue and, for KVM packages prior to `dgx-kvm-image-4-0-3`, provides instructions for enabling multi-queue. Starting with `dgx-kvm-image-4-0-3`, multi-queue is enabled by default.

The KVM guest VM will use virtio-net driver when it is using network interface based on Virtual Network Switch either in NAT or Bridged mode. By default, this virtio-net driver will use one pair of TX and RX queues and this can limit the guest network performance, even though it may be configured to use multiple vCPUs and their network interface is bound to 10/100G Host Physical NIC. Multi-queue support in virtio-net driver will

- ▶ Enable packet send/receive processing to scale with the number of available virtual CPUs in a guest
- ▶ Allow each guest virtual CPU to have its own separate TX and RX queue and interrupts that can be used without influencing other virtual CPUs.
- ▶ Provide better application scalability and improved network performance in many cases.

Multi-queue virtio-net provides the greatest performance benefit when:

- ▶ Traffic packets are relatively large.
- ▶ The guest is active on many connections at the same time, with traffic running between guests, guest to host, or guest to an external system.
- ▶ The number of queues is equal to the number of vCPUs. This is because multi-queue support optimizes RX interrupt affinity and TX queue selection in order to make a specific queue private to a specific vCPU.



**Note:** Multi-queue virtio-net works well for incoming traffic, but can occasionally hurt performance for outgoing traffic. Enabling multi-queue virtio-net increases the total throughput, and in parallel increases CPU consumption.

To use multi-queue virtio-net, enable support in the guest by adding the following to the guest XML configuration (where the value of N is from 1 to 256, as the kernel supports up to 256 queues for a multi-queue tap device). For the best results, match the number of queues with number of vCPU cores configured on the VM.



**Note:** This is not needed with KVM image dgx-kvm-image-4-0-3 or later.

```
$ virsh edit <VM name or ID>
<interface type='bridge'>
  <source bridge='virbr0'/>
  <model type='virtio'/>
  <driver name='vhost' queues='N'/>
</interface>
```

When running a virtual machine with N virtio-net queues in the guest VM, you can check the number of enabled queues using

```
$ ethtool -L <interface>
$ /sys/class/net/<interface>/queues
```

You can change the number of enabled queues (where the value of M is from 1 to N):



```
$ ethtool -L <interface> combined M
```



**Note:** When using multi-queue, it is recommended to change the `max_files` variable in the `/etc/libvirt/qemu.conf` file to 2048. The default limit of 1024 can be insufficient for multi-queue and cause guests to be unable to start when multi-queue is configured.

This is enabled by default on DGX-2 guest VMs for the current release of KVM SW.

### 11.8.3. QOS

By default, Virtual Network Switch will treat the network traffic from all guests equally and process them order in which it receive the packets. Virtual machine network quality of service is a feature that allows limiting both the inbound and outbound traffic of individual virtual network interface controllers or guests.

Virtual machine network quality of service settings allow you to configure bandwidth limits for both inbound and outbound traffic on three distinct levels.

- ▶ **Average:** The average speed of inbound or outbound traffic. Specifies the desired average bit rate for the interface being shaped (in kilobytes/second).
- ▶ **Peak:** The speed of inbound or outbound traffic during peak times. Optional attribute which specifies the maximum rate at which the bridge can send data (in kilobytes/second). Note the limitation of implementation: this attribute in the outbound element is ignored (as Linux ingress filters don't know it yet).
- ▶ **Burst:** The speed of inbound or outbound traffic during bursts. This is an optional attribute which specifies the number of kilobytes that can be transmitted in a single burst at peak speed.

The libvirt domain specification includes this functionality already. You can specify separate settings for incoming and outgoing traffic. When you open the XML file of your virtual machine, find the block with interface type tag. Try to add the following.

```
$ virsh edit <VM name or ID>
```

```
<bandwidth>
  <inbound average='NNN' peak='NNN' burst='NNN' />
  <outbound average='NNN' peak='NNN' burst='NNN' />
</bandwidth>
```

Where **NNN** is desired speed in KBS and it can be different for inbound/outbound and also, average/peak/burst can have different values.

This is not enabled by default on DGX-2 guest VMs for the current release of KVM SW.

## 11.9. References

- ▶ [Ubuntu KVM Networking](#)
- ▶ [Ubuntu Network Bonding](#)
- ▶ [Libvirt Networking](#)
- ▶ [Libvirt Networking Handbook](#)
- ▶ [Multi-Queue Virtio Net](#)

- ▶ [Virtual Network QOS](#)
- ▶ [Redhat Virtualization Tuning and Optimization Guide](#)
- ▶ [PCI SIG SR IOV Primer](#)
- ▶ [Mellanox SR-IOV NIC Configuration](#)

---

# Chapter 12. DGX-2 KVM Performance Tuning

NVIDIA DGX-2 virtualization supports guest GPU VMs as described in the KVM chapter of the NVIDIA DGX-2 System User Guide. The guest VMs are statically resourced with the default number of resources such as vCPUs, GPUs, and memory. The default values take into consideration PCIe topology, CPU affinity, and NVLink topology to provide optimal performance.

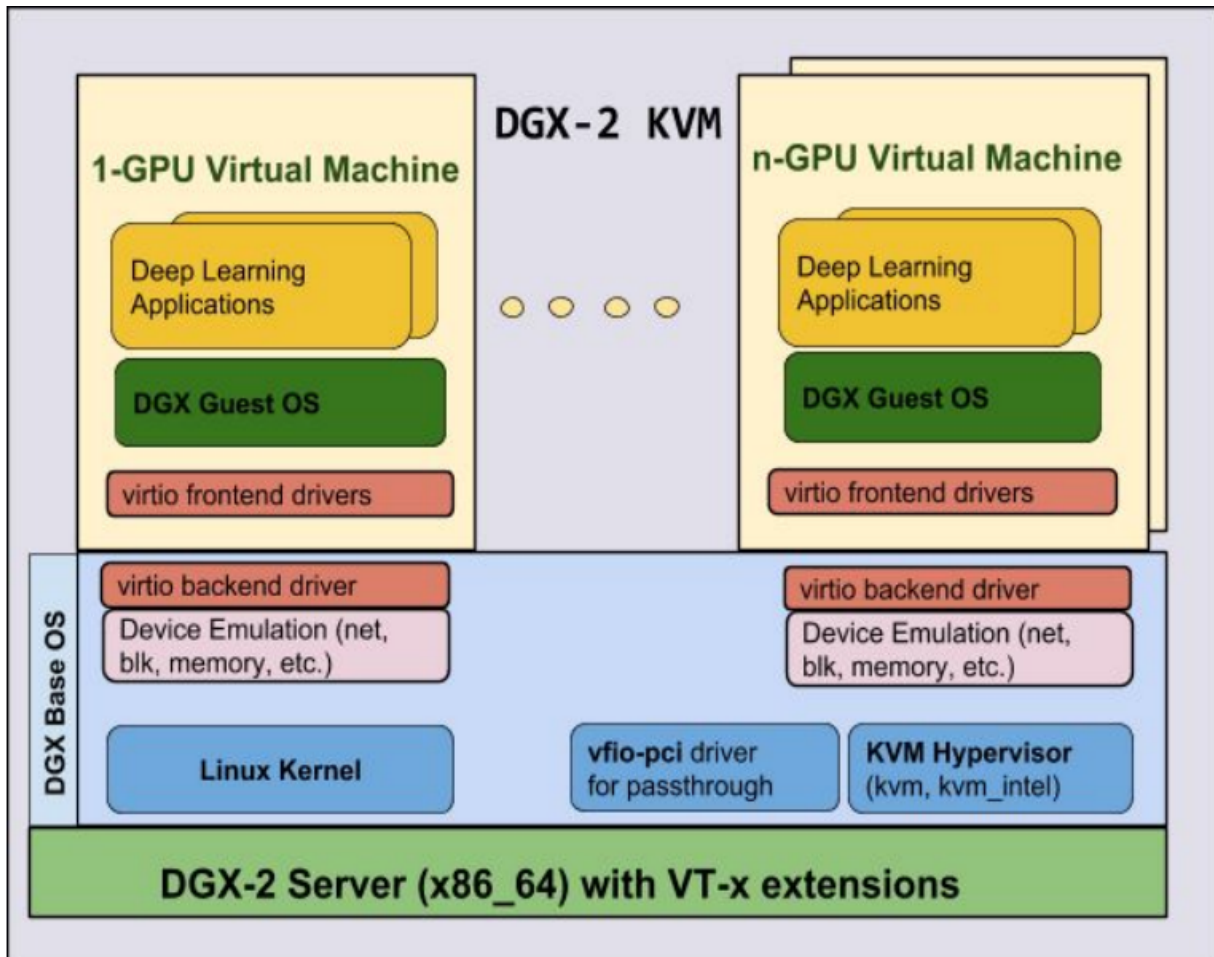
These default settings can be overridden to provide additional performance optimizations. This chapter discusses these performance optimizations.

## 12.1. Background

Guest GPU VMs run as simple user-space Linux processes in the KVM Host while vCPUs are POSIX threads running in the Host. The Linux kernel running on the KVM Host uses a built-in scheduler to handle these processes and threads. The default settings provide generic functionality, but you may need to apply some level of performance tuning to achieve better performance. Since performance settings aren't generic enough to accommodate every application's needs, you should treat performance tuning as an iterative process - change a setting, run tests, then evaluate results, repeating the process until you achieve optimal performance for the particular application or set of applications. Use bare-metal values as a baseline of what can be achieved, then compare guest VM results to that of the bare-metal as you work towards improving the results.

### Performance Tuning Using the Paravirtualized Drivers

The diagram below shows the I/O flow between the hypervisor and guests.



The DGX-2 KVM uses the following paravirtualized drivers (virtio) to help improve the DGX-2 KVM performance.

- ▶ **virtio-net**: The virtio-net driver supports virtual network devices.
- ▶ **virtio-blk**: The virtio-blk driver supports virtual block devices (OS drive, Data drive).
- ▶ **virtio-balloon**: The virtio memory balloon driver manages guest memory.
- ▶ **virtio-console**: The virtio-console drivers manage data flow between the guest and KVM host

You can use change the default settings of these drivers to improve performance.

This chapter describes the following areas of performance tuning.

- ▶ CPU tuning
- ▶ Memory tuning using huge pages
- ▶ NUMA tuning
- ▶ I/O tuning

## 12.2. CPU Tuning

Although KVM supports overcommitting virtualized CPUs, the DGX-2 implementation limits vCPUs to a 1:1 match of the number of hyperthreads available. As previously mentioned, vCPUs are actually POSIX threads in the KVM host; they are subject to the scheduler running on the DGX-2 system's policy which assigns equal priority to each vCPU.

To achieve optimal performance, the software pins vCPUs to hyperthreads as described in the following section. Since vCPUs run as user-space tasks on the host operating system, pinning increases cache efficiency.



**Note:** While you can override the default number of vCPUs provided and over-commit the vCPUs, this is not recommended as the effects this would have on performance are not defined.

### 12.2.1. vCPU Pinning

The NVIDIA DGX-2 system is a NUMA-aware system; by pinning vCPUs to hyperthreaded physical CPU cores, applications can increase the CPU cache hit ratio and reduce the number of costly context switches. Another advantage of vCPU pinning is applications can avoid slow memory access to remote NUMA Nodes since all vCPUs are pinned to a single NUMA node. With vCPU pinning, large performance improvements can be obtained with no known negative side effects.

By default, DGX-2 guest GPU VMs support vCPU pinning and no extra steps are needed. The default vCPU pinning is based on DGX-2's NUMA topology.

#### How to verify if vCPU pinning is enabled

The outputs below show vCPU pinning: enabled in the first VM and disabled in the second. When vCPU pinning is enabled, the `'virsh vcpuinfo'` CPU Affinity output shows a 'y' for the pinned vCPU and '-' for all other hyperthreaded physical CPU cores. When vCPU pinning is disabled, the `'virsh vcpuinfo'` CPU Affinity output shows a 'y' for all hyperthreaded physical CPU cores.

Obtain the ID numbers for the VMs on the system.

```
lab@expl-dvt-64:~$ virsh list
 Id      Name                                     State
-----
 14      dgx2vm-labThu1726-8g0-7               running
 15      dgx2vm-labThu1733                      running
```

In this example there are two VM IDs - 14 and 15. The following output shows that VM #14 has vCPU pinning enabled as indicated by each CPU Affinity line containing a single 'y'.

```
lab@expl-dvt-64:~$ virsh vcpuinfo 14
VCPU:      0
CPU:        0
```

```

State:      running
CPU time:   54.9s
CPU Affinity: y-----
VCPU:      1
CPU:       1
State:      running
CPU time:   6.8s
CPU Affinity: -y-----
VCPU:      2
CPU:       2
State:      running
CPU time:   8.3s
CPU Affinity: --y-----
VCPU:      3
CPU:       3
State:      running
CPU time:   3.5s
CPU Affinity: ---y-----
VCPU:      4
CPU:       4
State:      running
CPU time:   7.6s
CPU Affinity: ----y-----
...
VCPU:      22
CPU:       22
State:      running
CPU time:   2.6s
CPU Affinity: -----y-----
VCPU:      23
CPU:       48
State:      running
CPU time:   2.9s
CPU Affinity: -----
y-----
VCPU:      24
CPU:       49
State:      running
CPU time:   2.5s
CPU Affinity: -----
y-----
...
VCPU:      44
CPU:       69
State:      running
CPU time:   4.6s
CPU Affinity: -----
y-----
VCPU:      45
CPU:       70
State:      running
CPU time:   5.6s
CPU Affinity: -----
y-----

```

The following example shows vCPU pinning is disabled on VM #15 as indicated by each CPU Affinity all filled with y's.

```

lab@xpl-dvt-64:~$ virsh vcpuinfo 15
VCPU:      0
CPU:       5
State:      running

```

```

CPU time:      11.0s
CPU Affinity:
 yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
VCPU:          1
CPU:           11
State:         running
CPU time:      9.2s
CPU Affinity:
 yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
lab@expl-dvt-64:~$

```

Verify that there is only a single 'y' on each "CPU Affinity:" line. For a VM without vCPU pinning, there are multiple 'y's on the affinity line.

## 12.2.2. How to Disable vCPU Pinning

vCPU pinning is always enabled by default on DGX-2 KVMs. The following describes how to disable vCPU pinning.

1. Launch a GPU VM.
2. Shut the VM down using `virsh shutdown`.
3. Edit the VM XML file using `virsh edit <vm-name>`.
4. Remove vCPU pin entries.

The following example shows the vCPU entries in the XML file for a 2-GPU VM. These are the lines that need to be removed to disable vCPU pinning.

```

lab@dgx2~$ virsh edit 2gpu-vm-2g0-1

<vcpupin vcpu='0' cpuset='0' />
<vcpupin vcpu='1' cpuset='1' />
<vcpupin vcpu='2' cpuset='2' />
<vcpupin vcpu='3' cpuset='3' />
<vcpupin vcpu='4' cpuset='4' />
<vcpupin vcpu='5' cpuset='5' />
<vcpupin vcpu='6' cpuset='48' />
<vcpupin vcpu='7' cpuset='49' />
<vcpupin vcpu='8' cpuset='50' />
<vcpupin vcpu='9' cpuset='51' />
<vcpupin vcpu='10' cpuset='52' />

```

5. Restart the VM using `virsh start <vm-name>`

## 12.2.3. Core Affinity Optimization

The NVIDIA DGX-2 system has hyperthreading enabled, which means the Linux kernel displays two threads (or pCPUs) for each physical core. Guest VMS will need to pin their vCPUs to the matching physical CPU id values. These physical CPUs, or threads, will not be used to schedule jobs from other guests, provided the other guests pin their vCPUs to different pCPUs.

### Benefits of Core Affinity Optimization

The benefits of core affinity optimization are twofold. First, cores are not split across guests, so cache utilization of the cores improve. This is because the L1/L2 caches of a core are not shared across guests. Second, the guest VM's conception of cores and threads perfectly

matches real cores and threads - allowing the OS task scheduler to schedule jobs on the underlying processors more efficiently. Core affinity, in a nutshell, is about pinning guest vCPUs to host pCPUs in a way that matches the guest VM's notion of cores and threads onto actual pCPU cores and threads.

## Core Affinity Optimization on the NVIDIA DGX-2

The NVIDIA DGX-2 uses dual Intel CPUs (2 sockets), each with 24 cores/48 threads, for a total of 48 cores/96 threads with hyperthreading enabled. The kernel enumerates the pCPUs across both sockets before enumerating the thread siblings.

- ▶ **On the Intel system's first socket:** pCPUs are numbered 0 - 23.
- ▶ **On the Intel system's second socket:** pCPUs are numbered 24 - 47.
- ▶ **On the Intel system's first socket:** Thread siblings are numbered 48 - 71.

These form thread pairs with the pCPUs first enumerated on the first socket - (0, 48), (1, 49), (2, 50).....(23, 71), each pair sharing a common core.

- ▶ **On the Intel system's second socket:** Thread siblings are numbered 72 - 95.

These form thread pairs with the pCPUs first enumerated on the second socket - (24, 72), (25, 73), (26, 74).....(47, 95), each pair sharing a common core.

The relationship between which threads share a core can be read from the `sysfs` file: `/sys/devices/system/cpu/cpuX/topology/thread_siblings_list`, where `X` is the pCPU number enumerated by the host kernel.

For performance reasons, it is best not to split threads in a common core across guest VMs. Since pCPUs sharing a core also share L1/L2 caches, processes running on different guests will compete for the same cache resources on such a split-core scenario, potentially affecting performance. In addition, the guest VMs' view of hyperthreading needs to map to the actual physical threads sharing a core.

For optimal performance, allocate an even number of vCPUs to each KVM guest.

When the guest kernel schedules jobs, it will assume each successive pair of vCPUs belong to the same physical core - as you can see, this is indeed the case when the domain XML file specifies the CPU pinning in the above way.

When creating guests on a DGX system, depending on the actual number of cores per socket (e.g. DGX-2 has 24 cores per socket, for a total of 96 threads), the actual mapping will need to be adjusted.

## Enabling Core Affinity Optimization

1. Create the guest VM using `nvidia-vm`.
2. Shut down the VM.
3. Edit the VM's XML file located at `/etc/libvirt/qemu/`.



Make a backup of the XML file before making any modification so that the original setting can be easily restored if needed.

Edit the file according to the GPU size of the VM as well as the number of sockets used as described in the following sections.

4. Start the guest using the `virsh start` command.

## 1-GPU VMs

It is best not to attempt to use core affinity optimization for guest VMs using one GPU as it uses an odd number of vCPUs.

Currently, `nvidia-vm` allocates an odd number of CPUs to each single-GPU guest VM. For maximum benefits of hyperthread scheduling, the guest should have an even number of vCPUs. While it is possible to enable core affinity optimization for such VMs by editing the domain XML file to add or remove vCPUs from the guest, doing so may conflict with other `nvidia-vm` operations such as the pCPUs allocated to other guests.

## 2, 4, or 8 GPU VMs

For 2, 4, or 8-GPU VMs, core affinity can be enabled by editing the `<cputune>` element in the domain XML file -

- Specify that only one socket is used.

```
<cpu>
...
<topology sockets='1' cores='23' threads='2' />
```

- Pin each vCPU to a pCPU according to the numbering outlined in the previous section.

## 16-GPU VMs

The change is similar for a 16-GPU guest VM. Core affinity can be enabled by editing the `<cputune>` element in the domain XML file -

- Specify that two sockets are used.

```
<cpu>
...
<topology sockets='2' cores='23' threads='2' />
```

- Pin each vCPU to a pCPU according to the numbering outlined in the previous section.

When editing the domain XML file, be sure to take into account both sockets.

## Example of Enabling Core Affinity Optimization on a 2-GPU VM

Ten vCPUs are allocated to a 2-GPU VM.

After shutting down the VM, edit the VM's XML file located at `/etc/libvirt/qemu/`.

- Specify the vCPU pinning.

Example of pinning to the pCPUs on the first socket.

```
<cputune>
<vcpupin vcpu='0' cpuset='0' />
<vcpupin vcpu='1' cpuset='48' />
```

```
<vcpupin vcpu='2' cpuset='1' />
<vcpupin vcpu='3' cpuset='49' />
<vcpupin vcpu='4' cpuset='2' />
<vcpupin vcpu='5' cpuset='50' />
<vcpupin vcpu='6' cpuset='3' />
<vcpupin vcpu='7' cpuset='51' />
<vcpupin vcpu='8' cpuset='4' />
<vcpupin vcpu='9' cpuset='52' />
```

Example of pinning to the pCPUs on the second socket.

```
<cputune>
<vcpupin vcpu='0' cpuset='24' />
<vcpupin vcpu='1' cpuset='72' />
<vcpupin vcpu='2' cpuset='25' />
<vcpupin vcpu='3' cpuset='73' />
<vcpupin vcpu='4' cpuset='26' />
<vcpupin vcpu='5' cpuset='74' />
<vcpupin vcpu='6' cpuset='27' />
<vcpupin vcpu='7' cpuset='75' />
<vcpupin vcpu='8' cpuset='28' />
<vcpupin vcpu='9' cpuset='76' />
```

- Specify that only one socket is used.

```
<topology sockets='1' cores='23' threads='2' />
```

## Disabling Core Affinity

To disable core affinity for a running guest,

1. Stop the guest VM.
2. Restore the original XML file, or remove the above modifications from the updated XML file.
3. Restart the guest VM.

## 12.3. Memory tuning

By default, DGX-2 guest VMs receive host memory allocation based on the number of GPUs assigned to the guest. This static allocation can always be overridden by editing the guest VM template.

### 12.3.1. Huge Pages Support

The NVIDIA DGX-2 KVM host and guest OS supports huge pages which help improve memory management performance.

The Linux kernel manages memory in page-granularity with the default size of 4 KB. The Linux kernel also supports larger page sizes of 2 MB to 1 GB. These are called huge pages. Huge pages significantly improve performance by increasing CPU cache hits against the Translation LookAside Buffer (TLB).

All pages are managed by a Memory Management Unit (MMU) built into the CPU. You could use 2MB pages with systems that have many GBs of memory, and 1GB pages with systems that have TBs of memory (such as the NVIDIA DGX-2).



**Note:** When requesting 1GB huge pages, the Linux kernel may limit how many can be allocated.

Huge pages can be configured and used using one of the following methods.

► [Transparent Huge Pages \(THP\)](#)

THP supports multiple configurations. In the default Linux configuration, the Linux kernel attempts to allocate THP huge pages automatically. No special configuration is required.

► [Persistent Huge Pages \(Huge TLB\)](#)

Using Huge TLB requires special configuration. See below for details.

The rest of this section describes how to enable huge pages using Persistent Huge Pages (HugeTLB) during both run-time and boot-time.

### 12.3.1.1. How to set up Huge Pages at Runtime

Enabling huge pages results in performance improvements when the OS boots. For normal running of guest VMs, enable huge pages only when the workload and sizes are known, plannable, and rarely changed.

The example below shows how to set up a 16-GPU VM to use 2MB huge pages.

1. Stop the 16-GPU VM.

```
$ virsh list
```

Id	Name	State
4	dgx2vm-labMon1906-16g0-15	running

```
$ virsh shutdown dgx2vm-labMon1906-16g0-15
```

2. Set up huge pages on this VM using 2MB huge pages.

- a). View how much RAM the VM is using.

```
$ virsh edit dgx2vm-labMon1906-16g0-15
```

```
<memory unit='KiB'>1516912640</memory>
<currentMemory unit='KiB'>1516912640</currentMemory>
```

- b). Convert this memory value to 2 MB units by dividing by 2048.

This results in 740680 2MB huge pages.

- c). Set up the VM to use 740680 2MB huge pages.

```
$ echo 740680 | sudo tee /etc/sysctl.conf
```

```
$ sudo sysctl vm.nr_hugepages=740680
```

- d). Verify the changes are in place:

```
$ cat /proc/meminfo | grep -i huge
```

```
AnonHugePages:      0 kB
ShmemHugePages:     0 kB
HugePages_Total:    740680
HugePages_Free:     740680
HugePages_Rsvd:      0
HugePages_Surp:      0
Hugepagesize:       2048 kB
```

- Restart libvirtd for the above changes to take effect.

```
$ sudo systemctl restart libvirtd
```

- Modify the guest VM.

```
$ virsh edit dgx2vm-labMon1906-16g0-15
```

Add the following lines to the beginning of the XML file.

```
<domain type='kvm'>
  <name>dgx2vm-labMon1906-16g0-15</name>
  <uuid>0c9296c6-2d8f-4712-8883-dac654e6bc69</uuid>
  <memory unit='KiB'>1516912640</memory>
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  <currentMemory unit='KiB'>1516912640</currentMemory>
```

- Restart the GPU VM

```
$ time virsh start dgx2vm-labMon1906-16g0-15
```

```
Domain dgx2vm-labMon1906-16g0-15 started
real5m32.559s
user0m0.016s
sys0m0.010s
```

### 12.3.1.2. How to set up Huge Pages only for boot

To allocate different sizes of huge pages at boot time, modify GRUB in the DGX Host OS image, specifying the number of huge pages.

This example allocates 1 GB huge pages for a 16-GPU VM.

- Shut down the guest VM.

```
$ virsh shutdown dgx2vm-labMon1906-16g0-1
```

- Calculate the maximum number of 1 GB huge pages required for a 16-GPU VM.

- Determine the amount of memory allocated or used by the 16-GPU VM.

```
$ virsh edit dgx2vm-labMon1906-16g0-15
```

**Example output:**

```
<domain type='kvm'>
  <name>dgx2vm-labMon1906-16g0-15</name>
  <uuid>0c9296c6-2d8f-4712-8883-dac654e6bc69</uuid>
  <memory unit='KiB'>1516912640</memory>
```

In this example, 1516912640 KB (1,517 GB) of memory is allocated to the VM.

- Calculate the required number of 1 GB huge pages using the formula:

Number of 1 GB huge pages = (memory allocated (KB))/(1024\*1024)

Using the example, 1516912640 / (1024\*1024) = 1446, so 1446 huge pages are needed, 1 GB each.

- Set the number of huge pages to allocate at boot time.

Edit /etc/default/grub and change the following line to specify the number of huge pages to be allocated at boot for the 16-GPU VM

```
GRUB_CMDLINE_LINUX=""
```

Example:

```
GRUB_CMDLINE_LINUX="default_hugepagesz=1G hugepages=1446"
```

- After modifying GRUB, run the following command for the changes to take effect.

```
$ sudo update-grub
```

5. Reboot the KVM host.

```
$ sudo reboot
```

6. Modify the guest VM.

```
$ virsh edit dgx2vm-labMon1906-16g0-15
```

- a). Add the following lines to the XML file

```
<memoryBacking>
  <hugepages/>
</memoryBacking>
```

- b). Add the following lines to the beginning of the XML file.

```
<domain type='kvm'>
  <name>dgx2vm-labMon1906-16g0-15</name>
  <uuid>0c9296c6-2d8f-4712-8883-dac654e6bc69</uuid>
  <memory unit='KiB'>1516912640</memory>
  <memoryBacking>
    <hugepages/>
  </memoryBacking>
  <currentMemory unit='KiB'>1516912640</currentMemory>
```

7. Restart the guest VM.

```
$ time virsh start dgx2vm-labMon1906-16g0-15
Domain dgx2vm-labMon1906-16g0-15 started
real5m32.559s
user0m0.016s
sys0m0.010s
```

### 12.3.1.3. How to disable Huge Pages in the Host

1. Stop any running Guests.
2. Disable Huge Pages support
3. Restart libvirtd.
4. Before you restart the VM, ensure you remove the Hugepage entry from the XML file.

```
$ echo 0 | sudo tee /etc/sysctl.conf
$ sudo sysctl vm.nr_hugepages=0
```

```
$ sudo systemctl restart libvirtd
```

```
$ sudo virsh edit dgx2vm-labMon1906-16g0-15
```

```
<memoryBacking>
  <hugepages/>
</memoryBacking>
```

5. Save the file and restart your VM.



**Note:** The effect on boot time will not be significant as most of the time is spent allocating RAM for the guest. Hence, no numbers are published here.

## 12.4. NUMA Tuning

Non-Uniform Memory Access (NUMA) allows system memory to be divided into zones (nodes). NUMA nodes are allocated to particular CPUs or sockets. In contrast to the traditional monolithic memory approach where each CPU/core can access all the memory regardless of its locality, usually resulting in larger latencies, NUMA-bound processes can access memory

that is local to the CPU they are being executed on. In most cases, this is much faster than the memory connected to the remote CPUs on the system.

DGX-2 divides its memory to be equally accessible by its Skylake processors (nodes) using the NUMA architecture. This means that a particular set of Skylake processor has identical access latency to the local subset of system RAM. For virtualized environments, a few tweaks are needed to get the maximum performance out of the NUMA architectures.

## 12.4.1. Automatic NUMA Balancing

If the threads scheduled by an application are accessing memory on the same NUMA node, the performance of the application will be generally better. Automatic NUMA balancing moves tasks (which can be threads or processes) closer to the memory they are accessing. It also moves application data to memory closer to the tasks that reference it. This is all done automatically by the kernel when automatic NUMA balancing is active.

Automatic NUMA balancing uses a number of algorithms and data structures which are only active and allocated if automatic NUMA balancing is active on the system.

Automatic NUMA balancing is enabled by default on DGX-2 systems and it improves the performance of applications. There are no side effects of enabling NUMA balancing.

1. To check and enable automatic NUMA balancing, enter the following.

```
# cat /proc/sys/kernel/numa_balancing
```

This should return 1.

2. If 1 is not returned, then enter the following.

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

## 12.4.2. Enabling NUMA Tuning

DGX-2 node shows that it supports a total of 2 nodes (by running virsh capabilities). For a 16-GPU VM that supports up to 1.5TB memory, split the memory evenly into two cells such that each cell gets memory locality.

### 12.4.2.1. Setting Up NUMA Tuning

To set up NUMA tuning,

1. Stop the 16-GPU VM.

```
$ virsh list
  Id    Name
  ----  ---
  2     dgx2vm-labFri2209-16g0-15
  State
  ----  ---
  2     running
```

```
$ virsh shutdown 2
Domain 2 is being shutdown
```

2. Edit the XML file by adding lines as indicated.

```
$ virsh edit dgx2vm-labFri2209-16g0-15
<cpu>
<numa>
  <cell id="0" cpus="0-45" memory="758456320" unit="KiB"/>
  id="1" cpus="46-91" memory="758456320" unit="KiB"/>
</numa>
</cpu>
```

This defines two vNUMA nodes, each with 739 GiB of memory, and that cores 0-45 have low latency access to one of the 739GB sets, while 46-91 have low latency access to the other set. Applications that can be optimized for NUMA will be able to take this into account so that they try to limit the number of remote memory accesses they make.

3. Once the vNUMA cells are defined, use the `numatune` element to assign the physical NUMA node from which these cells will allocate memory.

```
<numatune>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

4. Restart the VM.

```
$ virsh start dgx2vm-labFri2209-16g0-15
Domain dgx2vm-labFri2209-16g0-15started
```

### 12.4.2.2. Effects of Enabling NUMA Tuning

There are no side effects of enabling NUMA tuning. Enabling NUMA tuning has shown performance improvements with 16-GPU VMs but largely varies upon the workload and application.

Adding NUMA elements is also recommended for smaller VMs to ensure memory is allocated from their associated physical NUMA node. The following examples show a 1-GPU VM on physical node 0:

```
<cpu>
  ...
  <numa>      <cell id='0' memory='10485760' unit='KiB' cpus='0-4' />    </numa></
cpu>
  ...
<numatune>
  <memnode cellid="0" mode="strict" nodeset="0"/>
</numatune>
```

and a 1-GPU VM on physical node 1:

```
<cpu>
  ...
  <numa>      <cell id='0' memory='10485760' unit='KiB' cpus='0-4' />    </numa></
cpu>
  ...
<numatune>
  <memnode cellid="0" mode="strict" nodeset="1"/>
</numatune>
```

## 12.5. Emulatorpin

The guest VM runs as process in the KVM Host. The process itself can run on any of the cores on the DGX-2. This Linux guest VM process (emulator) can also be pinned to run on some physical CPUs. If not pinned, the emulator is by default utilizing all the physical CPUs regardless of the NUMA affinity of the VM.

By using the optional `emulatorpin` element, you can achieve pinning the “emulator” to physical CPUs. The current recommendation is to pin the emulator to the free physical CPUs on the same CPU socket utilized by the VM. Pinning the emulator to the same CPU socket as the VM removes NUMA hops and QPI messages. Here is an example of how to do this for a 1-GPU VM:

```
<cputune>
...
<emulatorpin cpuset='50,53,56,59,62,65,68,71' />
</cputune>
```

## 12.6. I/O tuning

### 12.6.1. Using Multiple-queues with Logical Volumes

By default, each VM gets a data drive that is created using file-based storage, and a QCOW2-based logical volume is created.

```
$ nvidia-vm create --domain testme --gpu-count 8 --gpu-index 8
testme-8g8-15: create start mac: 52:54:00:d8:ec:20 ip: 192.168.122.26

$ virsh dumpxml testme-8g8-15

<snip> ..
  <disk type='file' device='disk'>
    <driver name='qemu' type='vmdk' />
    <source file='/raid/dgx-kvm/vol-testme-8g8-15' />
    <backingStore />
    <target dev='vdb' bus='virtio' />
    <alias name='virtio-disk1' />
    <address type='pci' domain='0x0000' bus='0x03' slot='0x00' function='0x0' />
  </disk>
</snip> ..
Login VM machine
nvidia@testme-8g8-15:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda   252:0    0  50G  0 disk
├─vda1 252:1    0  50G  0 part /
vdb   252:16   0 13.9T  0 disk
├─vdb1 252:17   0 13.9T  0 part /raid
nvidia@testme-8g8-15:~$
```

Data drive performance is not optimal, but you can set up the following features (not enabled by default) to improve data drive performance.

- ▶ Use the 'raw' drive type instead of QCOW2
  - ▶ QEMU Copy on Write version 2.0 (QCOW2) decouples physical storage layer from virtual layer by adding a mapping between logical and physical blocks. Each logical block is mapped to its physical offset.
  - ▶ RAW format uses no formatting and directly maps I/O written to the same offset in the backing file, thus providing the best performance.
- ▶ Disable caching
  - ▶ The host page cache is bypassed and I/O occurs directly between the hypervisor user space buffers and the backing store. It is equivalent to direct access to the Host's drives.
  - ▶ CONS: Disabling cache may affect data integrity.
- ▶ Enable multiple queues
  - ▶ Multiple queues provide improved storage performance and scalability in the virtio-blk driver. It enables each virtual CPU to have a separate queue and interrupt to use



without affecting other vCPUs. Ideally, the number of queues should be fewer than the total number of vCPUs belonging to a group, and the closest power of 2 number. For example, for a 92-vCPU VM, the ideal setting is 64.

Enabling these three has shown huge data drive performance gains.

### 12.6.1.1. I/O Threads

I/O threads are dedicated event loop threads. The threads allow disk devices to perform block I/O requests in order to improve scalability, especially on an SMP host/guest. This is a QEMU-only option and can be specified via an XML file schema in the following two ways.

#### ► **iothreads**

This optional element defines the number of I/O threads to be assigned to the domain for use by supported target storage devices.

```
<domain>
... <iothreads>4</iothreads>
...
</domain>
```

#### ► **iothreadids**

The optional `iothreadids` element provides the capability to specifically define the I/O thread ID's for the domain. These are sequentially numbered starting from 1 through the number of `iothreads` defined for the domain. The `id` attribute is used to define the I/O thread ID and is a positive integer greater than 0.

```
<domain>
... <iothreadids>
  <iothread id="2"/>
  <iothread id="4"/>
  <iothread id="6"/>
  <iothread id="8"/>
</iothreadids>
...
</domain>
```

### 12.6.1.2. How to Set up I/O Tuning

In the domain XML file, make following changes:

Add "`<iothreads>46</iothreads>`" line.

- 8-GPU VM is typically launched with 46 vCPUs
- Change this number to match number of vCPUs for your xGPU VM

Change `<driver>` tag for the `/raid` to "`<driver name='qemu' type='raw' cache='none' io='native' queues='32'/>`"

- The number of queues may not exceed the total number of vCPUs available

Example

1. Shut down the VM and then edit the XML file.

```
$ virsh list
```

Id	Name	State
-----		

```

1      testme-8g8-15      running
$ virsh shutdown testme-8g8-15
Domain testme-8g8-15 is being shutdown
$ virsh edit testme-8g8-15
<domain type='kvm'>
  <name>testme-8g8-15</name>
  <uuid>056f1635-d510-4d06-9e05-879e46479c08</uuid>
  <iothreads>46</iothreads>
  <snip> ..
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native' queues='32' />
      <source file='/raid/dgx-kvm/vol-testme-8g8-15' />
      <target dev='vdb' bus='virtio' />
      <address type='pci' domain='0x0000' bus='0x03' slot='0x00' function='0x00' />
    </disk>
  <snip> ..

```

2. Save the XML file and restart your VM.

```
$ virsh start testme-8g8-15
```

Running standard filesystem performance test tools (such as fio) on a data drive shows a 3x to 4x performance boost.

## 12.6.2. NVMe Drives as PCI-Passthrough Devices

By default, the DGX-2 guest GPU VMs support two drives when launched.

- OS Drive: /dev/vda (50 GB fixed in size)
- Data Drive: /dev/vdb (size varies depending on the number of GPUs in the VM, from 1.9 TB to 27 TB)

The OS drive and the data drive are a logical volume on the Host's NVMe drive, and as such, may not deliver the best performance. To improve performance, you can use PCI-passthrough to expose all the physical NVMe drives inside the VM.

This section describes how to pass through the NVMe SSDs to a 16-GPU guest VM using PCI-passthrough.

### 12.6.2.1. How to Set Up PCI-Passthrough for NVME Drives

Perform the following on the KVM host.

1. Stop the running RAID-0 on the KVM Host.

```

$ sudo cat /proc/mdstat
Personalities : [raid1] [raid0] [linear] [multipath] [raid6] [raid5] [raid4]
               [raid10]
md1  : active raid0 nvme2n1[2] nvme9n1[0] nvme4n1[3] nvme8n1[5] nvme3n1[4]
       nvme5n1[7] nvme7n1[1] nvme6n1[6]
       30004846592 blocks super 1.2 512k chunks
md0  : active raid1 nvme0n1p2[0] nvme1n1p2[1]
       937034752 blocks super 1.2 [2/2] [UU]
       bitmap: 1/7 pages [4KB], 65536KB chunk
unused devices: <none>
$ sudo umount /raid
$ sudo mdadm --stop /dev/md1
mdadm: stopped /dev/md1

```

2. Pass NVMe devices to the guest.

For each PCI Bus:Device:Function of NVMe devices, create an XML file by running this script:

```
#!/bin/bash
lspci | awk '{dev=0} /Micron/ {dev="nvme"} {
  if (dev!=0) { gsub(".*", "", $1); printf("%s %s\n", dev, $1); }}' | \
while read DEV PCID; do
echo "<hostdev mode='subsystem' type='pci' managed='yes'> <source> <address
  domain='0x0000' bus='0x${PCID}' slot='0x0' function='0x0' /> </source> </
hostdev>" > hw-${DEV}-${PCID}.xml;
done
```

This creates the following files

```
$ ls hw*
hw-nvme-2e.xml hw-nvme-2f.xml hw-nvme-51.xml hw-nvme-52.xml hw-nvme-b1.xml
hw-nvme-b2.xml hw-nvme-da.xml hw-nvme-db.xml
```

The following is an example of one of the files.

```
$ cat hw-nvme-2f.xml
<hostdev mode='subsystem' type='pci' managed='yes'> <source> <address
  domain='0x0000' bus='0x2f' slot='0x0' function='0x0' /> </source> </hostdev>
```

### 3. Pass one of these devices to the Guest VM as NVMe Passthrough

a). Create a GPU Guest VM without a data drive.

```
$ nvidia-vm create --domain nvme-passthrough --gpu-index 0 --gpu-count 16
nvme-passthrough-16g0-15: create start mac: 52:54:00:46:f3:34 ip:
192.168.122.91
$ virsh list --all
Id Name State
-----
1 nvme-passthrough-16g0-15 running
```

b). Pass an NVMe drive to the VM.

```
$ virsh attach-device nvme-passthrough-16g0-15 hw-nvme-2e.xml --live
Device attached successfully
```

c). Verify the NVMe device inside the VM.

```
$ virsh console nvme-passthrough-16g0-15
nvidia@nvme-passthrough-16g0-15:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda          252:0    0   50G  0 disk
└─vda1       252:1    0   50G  0 part /
nvme0n1      259:0    0  3.5T  0 disk
nvidia@nvme-passthrough-16g0-15:~$
```

There are no side effects of doing NVMe passthrough. Enabling NVMe passthrough has shown vast performance improvements with GPU VMs with various workloads and applications.

## 12.6.2.2. How to Revert PCI-Passthrough of NVMe Drives

These steps describe how to undo previous changes and are performed from the KVM Host.

#### 1. Destroy the VM

```
$ sudo nvidia-vm delete --domain nvme-passthrough-16g0-15 --force
```

#### 2. Recreate RAID-0 on the KVM Host

```
$ sudo mdadm --create --verbose /dev/md1 --level=0 --raid-devices=8 /dev/nvme2n1 /dev/
nvme3n1 /dev/nvme4n1 /dev/nvme5n1 /dev/nvme6n1 /dev/nvme7n1 /dev/nvme8n1 /dev/nvme9n1
mdadm: Defaulting to version 1.2 metadata
```

```
mdadm: array /dev/md1 started.
$ sudo mkfs.ext4 /dev/md1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 7501211648 4k blocks and 468826112 inodes
Filesystem UUID: 0e1d6cb6-020e-47d3-80a1-d2c93b259ff7
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848, 512000000, 550731776, 644972544, 1934917632,
2560000000, 3855122432, 5804752896
Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

### 3. Mount RAID-0 inside the KVM Host.

```
$ sudo mount /dev/md1 /raid
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
ARRAY /dev/md/0 metadata=1.2 name=dgx-18-04:0
  UUID=1740dd3f:6c26bdc1:c6ed2395:690d0707
ARRAY /dev/md1 metadata=1.2 name=xpl-dvt-34:1
  UUID=dfa7e422:430a396b:89fc4b74:9a5d8c3c
```



**Note:** Make sure these entries show up in `/etc/mdadm/mdadm.conf` and that they replace any previously existing entries.

After replacing the entries in `/etc/dmadm/mdadm.conf`; ensure that only the two lines from above show up. For example,

```
$ grep ARRAY /etc/mdadm/mdadm.conf
ARRAY /dev/md/0 metadata=1.2 name=dgx-18-04:0
  UUID=1740dd3f:6c26bdc1:c6ed2395:690d0707
ARRAY /dev/md1 metadata=1.2 name=xpl-dvt-34:1
  UUID=dfa7e422:430a396b:89fc4b74:9a5d8c3c
```

## 12.6.3. Physical Drive Passthrough

This section explains how to pass through a drive to a GPU Guest VM.

### Preliminary Steps

Be sure to perform the following before setting up passthrough for the physical drive.

1. Ensure mdadm raid isn't running on NVMe drives.

```
$ sudo ls /dev/md*
/dev/md0

/dev/md:
0
```

2. If you also see "md1", stop it.

```
$ sudo umount /raid
$ sudo mdadm --stop /dev/md1
mdadm: stopped /dev/md1
```

3. Create a large partition using parted.

```
$ sudo parted /dev/nvme4n1
GNU Parted 3.2
Using /dev/nvme4n1
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

```
(parted) p
Error: /dev/nvme4n1: unrecognised disk label
Model: NVMe Device (nvme)
Disk /dev/nvme4n1: 3841GB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
(parted) mklabel gpt
(parted) unit GB
(parted) mkpart 1 0 3841
(parted) quit
```

### 12.6.3.1. How to Set Up Drive Passthrough

1. Put a filesystem on a drive's partition (here nvme4n1 is used)

```
$ sudo mkfs.ext4 /dev/nvme4n1p1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 937684224 4k blocks and 234422272 inodes
Filesystem UUID: d3853f33-5241-478f-8a06-5010db70543d
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848, 512000000, 550731776, 644972544
Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

2. Launch a GPU VM, shut it down and pass no Data Drive

```
$ nvidia-vm create --domain disk-passthrough --gpu-count 1 --gpu-index 8 --volGB 0
WARNING: Host Data volume not setup, no VM data volume will be created
disk-passthrough-1g8: create start mac: 52:54:00:50:c3:95 ip: 192.168.122.198
```

3. Add these lines to XML

```
$ virsh shutdown disk-passthrough-1g8
$ virsh edit disk-passthrough-1g8
<disk type='block' device='disk'>
<driver name='qemu' type='raw'>
<source dev='/dev/nvme4n1'>
<target dev='vdb' bus='virtio'>
</disk>
```

4. Save and restart Guest VM

```
$ virsh start disk-passthrough-1g8
```

5. Verify that drive shows up inside the Guest VM:

```
$ virsh console disk-passthrough-1g8
nvidia@disk-passthrough-1g8:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda 252:0 0 50G 0 disk
└─vda1 252:1 0 50G 0 part /
vdb 252:16 0 3.5T 0 disk
└─vdb1 252:17 0 3.5T 0 part /raid
```

### 12.6.3.2. How to Revert Drive Passthrough

These steps explain how to undo the previous changes to set up drive passthrough. Perform these steps on the KVM Host.

1. Destroy the VM.

```
$ sudo nvidia-vm delete --domain disk-passthrough-1g8 --force
```

2. Recreate RAID-0 on the KVM host.

```
$ sudo mdadm --create --verbose /dev/md1 --level=0 --raid-devices=8 /dev/nvme2n1 /dev/nvme3n1 /dev/nvme4n1 /dev/nvme5n1 /dev/nvme6n1 /dev/nvme7n1 /dev/nvme8n1 /dev/nvme9n1
```

```
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

```
$ sudo mkfs.ext4 /dev/md1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 7501211648 4k blocks and 468826112 inodes
Filesystem UUID: 0e1d6cb6-020e-47d3-80a1-d2c93b259ff7
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848, 512000000, 550731776, 644972544, 1934917632, 2560000000,
3855122432, 5804752896
Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

3. Mount RAID-0 inside the KVM Host

```
$ sudo mount /dev/md1 /raid
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf
ARRAY /dev/md0 metadata=1.2 name=dgx-18-04:0
UUID=1740dd3f:6c26bdc1:c6ed2395:690d0707
ARRAY /dev/md1 metadata=1.2 name=xpl-dvt-34:1
UUID=dfa7e422:430a396b:89fc4b74:9a5d8c3c
```



**Note:** Make sure these entries show up in `/etc/mdadm/mdadm.conf` and replace existing ones. After replacing the entries in `/etc/dmadm/mdadm.conf`; ensure only the two lines from above appear.

```
$ grep ARRAY /etc/mdadm/mdadm.conf
ARRAY /dev/md0 metadata=1.2 name=dgx-18-04:0
UUID=1740dd3f:6c26bdc1:c6ed2395:690d0707
ARRAY /dev/md1 metadata=1.2 name=xpl-dvt-34:1
UUID=dfa7e422:430a396b:89fc4b74:9a5d8c3c
```

## 12.6.4. Drive Partition Passthrough

If there are not enough drives to support the number of VMs that need to be created, you can create multiple partitions on a disk and then pass through each partition to the VMs. This section explains how to pass a drive partition to a guest VM.

1. Stop RAID on `/dev/md1`, see the previous sections for an example.
2. Create two drive partitions (here `nvme5n1` is used) using `fdisk`.

```
$ fdisk /dev/nvme5n1
Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
The size of this disk is 3.5 TiB (3840755982336 bytes). DOS partition table
format cannot be used on drives for volumes larger than 2199023255040 bytes for
512-byte sectors. Use GUID partition table format (GPT).
Command (m for help): d
Selected partition 1
Partition 1 has been deleted.
Command (m for help): n
```

```

Partition type
  p primary (0 primary, 0 extended, 4 free)
  e extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-4294967295, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-4294967294, default 4294967294):
2147485695
Created a new partition 1 of type 'Linux' and of size 1 TiB.
Command (m for help): n
Partition type
  p primary (1 primary, 0 extended, 3 free)
  e extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2):
First sector (2147485696-4294967295, default 2147485696):
Last sector, +sectors or +size{K,M,G,T,P} (2147485696-4294967294, default
4294967294):
Created a new partition 2 of type 'Linux' and of size 1024 GiB.
Command (m for help): wq
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

```

### 3. Verify that the two partitions exist.

```

$ sudo fdisk -l /dev/nvme5n1
Disk /dev/nvme5n1: 3.5 TiB, 3840755982336 bytes, 7501476528 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb1dc316c

Device            Boot      Start          End      Sectors   Size Id Type
/dev/nvme5n1p1                2048 2147485695 2147483648    1T 83 Linux
/dev/nvme5n1p2      2147485696 4294967294 2147481599 1024G 83 Linux
$ lsblk
NAME            MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sr0              11:0    1  1024M  0 rom
nvme0n1         259:0    0 894.3G  0 disk
└─nvme0n1p1     259:1    0   512M  0 part /boot/efi
└─nvme0n1p2     259:2    0 893.8G  0 part
    └─md0        9:0    0 893.6G  0 raid1 /
nvme1n1         259:3    0 894.3G  0 disk
└─nvme1n1p1     259:4    0   512M  0 part
└─nvme1n1p2     259:5    0 893.8G  0 part
    └─md0        9:0    0 893.6G  0 raid1 /
nvme3n1         259:6    0   3.5T  0 disk
nvme4n1         259:7    0   3.5T  0 disk
nvme5n1         259:9    0   3.5T  0 disk
└─nvme5n1p1     259:8    0    1T  0 part
└─nvme5n1p2     259:17   0 1024G  0 part
nvme6n1         259:10   0   3.5T  0 disk
nvme2n1         259:11   0   3.5T  0 disk
nvme9n1         259:12   0   3.5T  0 disk
nvme8n1         259:13   0   3.5T  0 disk
nvme7n1         259:14   0   3.5T  0 disk

```

The example shows two partitions, with the intent of passing the 2nd partition to a guest VM.

### 4. Put a filesystem on a drive's partition (here nvme5n1p1 is used).

```

$ mkfs.ext4 /dev/nvme5n1p1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 268435456 4k blocks and 67108864 inodes
Filesystem UUID: 7fa91b82-51db-4953-87d5-4364958951e5

```

```

Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848
Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done

```

- Figure out the path to disk partition by UUID.

```

$ ls -l /dev/disk/by-partuuid/ | grep 5n1p1
lrwxrwxrwx 1 root root 15 Sep 26 08:35 b1dc316c-01 -> ../../nvme5n1p1

```

- Launch a GPU VM, shut it down and pass no data drive.

```

$ nvidia-vm create --domain disk-passthrough --gpu-count 1 --gpu-index 9 --volGB 0
WARNING: Host Data volume not setup, no VM data volume will be created
disk-passthrough-1g9: create start mac: 52:54:00:96:6c:38 ip: 192.168.122.140

```

- Capture how many devices are visible in the VM first.

```

nvidia@disk-passthrough-1g9:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda         252:0    0   50G  0 disk
└─vda1      252:1    0   50G  0 part /
nvidia@disk-passthrough-1g9:~$
$ virsh shutdown disk-passthrough-1g9

```

- Add these lines to XML next to existing <disk> entry

```

$ virsh edit disk-passthrough-1g9
    <disk type='block' device='disk'>
    <driver name='qemu' type='raw'/>
    <source dev='/dev/disk/by-partuuid/b1dc316c-01'/>
    <target dev='vdb' bus='virtio'/>
  </disk>

```



**Note:** Make sure the UUID matches that from the previous steps

- Save and restart the guest VM.

```

$ virsh start disk-passthrough-1g9

```

- Verify that drive shows up inside the guest VM.

```

$ virsh console disk-passthrough-1g9
nvidia@disk-passthrough-1g8:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda         252:0    0   50G  0 disk
└─vda1      252:1    0   50G  0 part /
vdb         252:16   0 1024G  0 disk
└─vdb1      252:17   0 1024G  0 part /raid

```

### 12.6.4.1. How to Set Up Drive Partition Passthrough

How to pass a Drive partition to a GPU Guest VM.

- Stop RAID on /dev/md1.  
See previous sections for an example.
- Create two drive partition (here nvme5n1 is used) using fdisk.

```

$ fdisk /dev/nvme5n1
Welcome to fdisk (util-linux 2.31.1).

```



```

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
The size of this disk is 3.5 TiB (3840755982336 bytes). DOS partition table
format cannot be used on drives for volumes larger than 2199023255040 bytes for
512-byte sectors. Use GUID partition table format (GPT).
Command (m for help): d
Selected partition 1
Partition 1 has been deleted.
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-4294967295, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-4294967294, default 4294967294):
2147485695
Created a new partition 1 of type 'Linux' and of size 1 TiB.
Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2):
First sector (2147485696-4294967295, default 2147485696):
Last sector, +sectors or +size{K,M,G,T,P} (2147485696-4294967294, default
4294967294):
Created a new partition 2 of type 'Linux' and of size 1024 GiB.
Command (m for help): wq
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

```

### 3. Verify that the two partitions exist.

```

$ sudo fdisk -l /dev/nvme5n1
Disk /dev/nvme5n1: 3.5 TiB, 3840755982336 bytes, 7501476528 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb1dc316c

Device            Boot      Start          End      Sectors  Size Id Type
/dev/nvme5n1p1                2048 2147485695 2147483648    1T 83 Linux
/dev/nvme5n1p2      2147485696 4294967294 2147481599 1024G 83 Linux
$ lsblk
NAME            MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sr0              11:0    1  1024M  0 rom
nvme0n1          259:0    0 894.3G  0 disk
├─nvme0n1p1      259:1    0   512M  0 part /boot/efi
├─nvme0n1p2      259:2    0 893.8G  0 part
└─┬md0           9:0     0 893.6G  0 raid1 /
nvme1n1          259:3    0 894.3G  0 disk
├─nvme1n1p1      259:4    0   512M  0 part
├─nvme1n1p2      259:5    0 893.8G  0 part
└─┬md0           9:0     0 893.6G  0 raid1 /
nvme3n1          259:6    0   3.5T  0 disk
nvme4n1          259:7    0   3.5T  0 disk
nvme5n1          259:9    0   3.5T  0 disk
├─nvme5n1p1      259:8    0    1T  0 part
├─nvme5n1p2      259:17   0 1024G  0 part
nvme6n1          259:10   0   3.5T  0 disk
nvme2n1          259:11   0   3.5T  0 disk
nvme9n1          259:12   0   3.5T  0 disk
nvme8n1          259:13   0   3.5T  0 disk
nvme7n1          259:14   0   3.5T  0 disk

```

This examples shows two partitions, with the intent of passing the 2nd partition to a guest VM.

4. Put a filesystem on a drive's partition (here nvme5n1p1 is used).

```
$ mkfs.ext4 /dev/nvme5n1p1
mke2fs 1.44.1 (24-Mar-2018)
Discarding device blocks: done
Creating filesystem with 268435456 4k blocks and 67108864 inodes
Filesystem UUID: 7fa91b82-51db-4953-87d5-4364958951e5
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
102400000, 214990848
Allocating group tables: done
Writing inode tables: done
Creating journal (262144 blocks): done
Writing superblocks and filesystem accounting information: done
```

5. Figure out the path to disk partition by UUID.

```
$ ls -l /dev/disk/by-partuuid/ | grep 5n1p1
lrwxrwxrwx 1 root root 15 Sep 26 08:35 b1dc316c-01 -> ../../nvme5n1p1
```

6. Launch a GPU VM, shut it down, and pass no data drive.

```
$ nvidia-vm create --domain disk-passthrough --gpu-count 1 --gpu-index 9 --volGB 0
WARNING: Host Data volume not setup, no VM data volume will be created
disk-passthrough-1g9: create start mac: 52:54:00:96:6c:38 ip: 192.168.122.140
```

7. Capture how many devices are visible in the VM.

```
nvidia@disk-passthrough-1g9:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda 252:0 0 50G 0 disk
└─vda1 252:1 0 50G 0 part /
nvidia@disk-passthrough-1g9:~$
$ virsh shutdown disk-passthrough-1g9
```

8. Add these lines to XML next to the existing <disk> entry.

```
$ virsh edit disk-passthrough-1g9
<disk type='block' device='disk'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/disk/by-partuuid/b1dc316c-01'/>
  <target dev='vdb' bus='virtio'/>
</disk>
```



**Note:** Make sure the UUID matches that from the previous steps.

9. Save and restart the guest VM.

```
$ virsh start disk-passthrough-1g9
```

10. Verify that drive shows up inside the guest VM.

```
$ virsh console disk-passthrough-1g9
nvidia@disk-passthrough-1g8:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda 252:0 0 50G 0 disk
└─vda1 252:1 0 50G 0 part /
vdb 252:16 0 1024G 0 disk
└─vdb1 252:17 0 1024G 0 part /raid
```

## 12.6.4.2. How to Revert Drive Partition Passthrough

To revert drive partition passthrough, follow the instructions in [How to Revert Drive Passthrough](#) as the same instructions apply.

## Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## Trademarks

NVIDIA, the NVIDIA logo, DGX, DGX-1, DGX-2, DGX A100, DGX Station, and DGX Station A100 are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

© 2022 NVIDIA Corporation. All rights reserved.