Deployment Guide
Non-volatile Memory Express over Fabrics
with Universal Remote Direct Memory Access
## Document Revision History

<table>
<thead>
<tr>
<th>Changes</th>
<th>Sections Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Release</td>
<td></td>
</tr>
</tbody>
</table>

Revision A, June 15, 2017
# Table of Contents

## 1 Introduction

- What is NVMe? .................................................. 1
- What is NVME over Fabrics (NVMe-oF)? .................... 1

## 2 Configuring NVMe-oF

- Install Device Drivers on Both Servers .................. 16
  - Driver Installation (General) .......................... 16
  - Driver Installation (Ubuntu) .......................... 19
- Configure the Target Server ............................... 21
- Configure the Initiator Server ............................ 23
- Precondition the Target Server ........................... 24
- Test the NVMe-oF Devices ................................... 25
- Optimize Performance ...................................... 26

## A Optimization Scripts

- IRQ Affinity (multi_rss-affin.sh) ......................... 15
- CPU Frequency (cpufreq.sh) ............................... 16
1 Introduction

What is NVMe?

Non-volatile Memory Express (NVMe) is a new and innovative method of accessing storage media that has been capturing the imagination of data center professionals worldwide. NVMe is an alternative to the small computer system interface (SCSI) standard for connecting and transferring data between a host and a peripheral target storage device or system. SCSI became a standard in 1986, when hard disk drives (HDDs) and tape were the primary storage media. NVMe is designed for use with faster media, such as solid-state drives (SSDs) and post-flash memory-based technologies. NVMe provides a streamlined register interface and command set to reduce the I/O stack's CPU overhead by directly accessing the PCIe bus. The benefits of NVMe-based storage drives include lower latency, additional parallel requests, and higher performance.

What is NVMe over Fabrics (NVMe-oF)?

NVMe over Fabrics (NVMe-oF) enables the use of alternate transports to PCIe to extend the distance over which an NVMe host device and an NVMe storage drive or subsystem can connect. NVMe-oF defines a common architecture that supports a range of storage networking fabrics for NVMe block storage protocol over a storage networking fabric. This includes enabling a front-side interface into storage systems, scaling out to large numbers of NVMe devices and extending the distance within a data center over which NVMe devices and NVMe subsystems can be accessed.

NVMe-oF is designed to extend NVMe onto fabrics such as Ethernet, Fibre Channel, and InfiniBand. Using remote direct memory access (RDMA) with NVMe-oF includes any of the RDMA technologies, including InfiniBand, RDMA over converged Ethernet (RoCE) and Internet wide area RDMA protocol (iWARP). The development of NVMe-oF with RDMA is defined by a technical sub-group of the NVMe organization.
2 Configuring NVMe-oF

This chapter demonstrates how to configure NVMe-oF for a simple network. The example network comprises the following, as shown in Figure 2-1:

- Two servers: an initiator and a target. The target server is equipped with an PCIe SSD drive.
- Operating system: CentOS 7.2 or Ubuntu 17.04 on both servers
- Two adapters: One FastLinQ 45000/41000 Series adapter installed in each server
- An optional switch configured for data center bridging (DCB) and relevant quality of service (QoS) policy

![Figure 2-1. NVMe-oF Network](image-url)
The NVMe-oF configuration process comprises the following steps:

1. Install Device Drivers on Both Servers
2. Configure the Target Server
3. Configure the Initiator Server
4. Precondition the Target Server
5. Test the NVMe-oF Devices
6. Optimize Performance

Install Device Drivers on Both Servers

To install device drivers for most operating systems follow the instructions in “Driver Installation (General)” on page 16. To install device drivers for the Ubuntu operating system, follow the instructions in “Driver Installation (Ubuntu)” on page 19.

Driver Installation (General)

1. Install support applications and libraries.
   
   # yum groupinstall "Infiniband Support"
   # yum install tcl-devel libibverbs-devel libnl-devel
glib2-devel libudev-devel lsscsi perftest

2. Install tools and libraries.
   
   # yum install gcc make git ctags ncurses ncurses-devel openssl
   openssl-devel
   # yum install rdma

3. Download a stable Linux 4.11.3 kernel and extract locally.

   # wget "https://cdn.kernel.org/pub/linux/kernel/v4.x/
   linux-4.11.3.tar.xz"
   # tar -xf linux-4.11.3.tar.xz
   # cd linux-4.11.3

4. Copy the current configuration file into a new kernel directory.
   
   # cp /boot/config-** .config

5. Add NVMe support and OFED to the kernel configuration.
   
   # make menuconfig
6. In the Configuration menu, select **Device Drivers**:

   ![Configuration Menu](image1.png)

   **Figure 2-2. Configuration Menu**

7. In the Device Drivers menu, select `<Select>` to modularize the following drivers by setting the corresponding entry to M, as shown in **Figure 2-3**:

   - NVMe Express block device
   - NVMe Express over Fabrics RDMA host driver
   - NVMe Target support
   - NVMe over Fabrics RDMA target support

   ![Modularizing Device Drivers](image2.png)

   **Figure 2-3. Modularizing Device Drivers**
8. In the Device Drivers menu, select `<Save>`. Type a file name for the configuration file, and then select `<ok>`, as shown in Figure 2-4:

![Figure 2-4. Save Configuration](image)

9. Build and compile a new kernel and modules. Press ENTER as needed to complete the build.

```
# make -j8 && make -j8 modules
```

10. Install the new kernel and modules.

```
# make -j8 modules_install
# make -j8 install
```

11. Reboot the server, and then select the newly installed kernel, as shown in Figure 2-5.

![Figure 2-5. Selecting the New Kernel](image)

12. Install and load the FastLinQ drivers (QED, QEDE, libqedr/QEDR) following all installation instructions in the README.
13. Enable and start the RDMA service.
   
   # systemctl enable rdma
   # systemctl start rdma

   Disregard the RDMA Service Failed error. All OFED modules required by QEDR are already loaded.

14. Obtain the most recent QLogic firmware utility from www.qlogic.com to install and load the FastLinQ libraries and drivers. FastLinQ drivers 8.18.x.x or later support the newer 4.8.x kernels.

   # tar -xjf fastlinq-minor-8.18.x.x.tar.bz2
   # cd fastlinq-minor-8.18.x.x
   # make libqedr_install
   # make; make install
   # rmmod qedr; rmmod qede; rmmod qed
   # modprobe qed; modprobe qede; modprobe qedr

   The QEDR module must be reloaded after each reboot.

**Driver Installation (Ubuntu)**

1. Install support applications and libraries.

   # apt-get install -f build-essential pkg-config vlan automake autoconf dkms git
   # apt-get install -f libibverbs* librdma* ibacm libibcm.*
   libibmad.* libibumad*

2. Install tools and libraries.

   # apt-get install -f libtool ibutils ibverbs-utils rdmacm-utils infiniband-diags perftest srptools librdmacm-dev libibverbs-dev numaclt libnuma-dev libnl-3-200 libnl-route-3-200 libnl-route-3-dev libnl-utils

   Follow Ubuntu package installation instructions and verify that all packages are installed.

3. Edit the /etc/security/limits.conf file to increase the size of memory, which can be locked by non-root user. Add the following lines, and then logout.

   * soft memlock unlimited
   * hard memlock unlimited
   root soft memlock unlimited
   root hard memlock unlimited
4. Log in to the system, and then type the following command:
   
   ```bash
   # ulimit -l
   ```

   Output should show as "unlimited".

5. Reboot the system.

   
   ```bash
   # modprobe -r qedr or modprobe -r qede
   # depmod -a
   ```

7. Install the FastLinQ package to allow the device to be recognized as an InfiniBand device that can be used by OFED.
   
   ```bash
   # cd fastlinq-minor-X.X.X.
   # make clean
   # make install
   ```

8. Install libqedr libraries to work with RDMA user space applications using one of the following methods:
   
   - **Method 1:**
     
     ```bash
     # cd fastlinq-minor-X.X.X.
     # make libqedr_install
     ```
   
   - **Method 2:**
     
     ```bash
     # cd fastlinq-minor-X.X.X./libqedr-X.X.X/
     # ./configure --prefix=/usr --libdir=${exec_prefix}/lib
     --sysconfdir=/etc
     # make install
     ```

9. Load the QLogic RDMA driver.
   
   ```bash
   # modprobe -v qedr
   ```

10. Load the RDMA modules. You must perform this step every time you reboot the system.

    ```bash
    # sudo modprobe rdma_cm
    # sudo modprobe ib_uverbs
    # sudo modprobe rdma_ucm
    # sudo modprobe ib_ucm
    # sudo modprobe ib_umad
    ```
11. **List the RoCE devices.**
   
   ```
   # ibv_devinfo
   ```

12. **Assign IP addresses to the Ethernet interfaces. To assign the static IP address, edit the `/etc/network/interfaces` file and add the following entries:**

   ```
   iface eth0 inet static
   address 192.168.10.5
   netmask 255.255.255.0
   gateway 192.168.10.254
   ```

---

**Configure the Target Server**

1. **Load target modules**
   
   ```
   # modprobe nvmet; modprobe nvmet-rdma
   # lsmod | grep nvme  (Confirm modules loaded)
   ```

2. **Create the target subsystem (NQN) with the name given by `nvme-subsystem-name`.**
   
   ```
   # mkdir /sys/kernel/config/nvmet/subsystems/
   nvme-subsystem-name
   # cd /sys/kernel/config/nvmet/subsystems/nvme-subsystem-name
   ```

3. **Create multiple unique NQNs for additional NVMe devices as needed.**
4. Set the target parameters, as listed in Table 2-1:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td># echo 1 &gt; attr_allow_any_host</td>
<td>Allow any host to connect.</td>
</tr>
<tr>
<td># mkdir namespaces/1</td>
<td>Create a namespace</td>
</tr>
<tr>
<td># echo -n /dev/nvme0n1 &gt; namespaces/1/device_path</td>
<td>Set the NVMe device path. The NVMe device path can differ between systems. Check the device path using the lsblk command. This system has two NVMe devices: nvme0n1 and nvme1n1.</td>
</tr>
<tr>
<td># echo 1 &gt; namespaces/1/enable</td>
<td>Enable the namespace.</td>
</tr>
<tr>
<td># mkdir /sys/kernel/config/nvmet/ports/1</td>
<td>Create NVMe port 1.</td>
</tr>
<tr>
<td># cd /sys/kernel/config/nvmet/ports/1</td>
<td></td>
</tr>
<tr>
<td># echo 1.1.1.1 &gt; addr_traddr ()</td>
<td>Set the IP address (1.1.1.1) for target QLogic 45000/41000 adapter.</td>
</tr>
<tr>
<td># echo rdma &gt; addr_trtype</td>
<td>Set the transport type RDMA.</td>
</tr>
<tr>
<td># echo 1023 &gt; addr_trsvcid</td>
<td>Set the RDMA port to any number.</td>
</tr>
<tr>
<td># echo ipv4 &gt; addr_adrfam</td>
<td>Set the IP address type.</td>
</tr>
</tbody>
</table>

5. Create a sym link to the newly created NQN subsystem.

   # ln -s /sys/kernel/config/nvmet/subsystems/
   nvme-subsystem-name subsystems/nvme-subsystem-name

6. Confirm that the NVMe target is listening on port.

   # dmesg | grep nvmet_rdma

   [ 8769.470043] nvmet_rdma: enabling port 1 (1.1.1.1:1023)
Configure the Initiator Server

1. Load the NVMe modules.
   
   ```
   # modprobe nvme-rdma
   ```

2. Download, compile and install the `nvme-cli` Initiator utility
   
   ```
   # git clone https://github.com/linux-nvme/nvme-cli.git
   # cd nvme-cli
   # make && make install
   ```

   For Ubuntu:
   
   ```
   # apt-get install nvme-cli
   ```

3. Verify the installation version.
   
   ```
   # nvme version
   ```

4. Discover the NVMe-oF target
   
   ```
   # nvme discover -t rdma -a 1.1.1.1 -s 1023
   ```

   Make note of the subsystem NQN (`subnqn`) of the discovered target (Figure 2-6).

   ![Discovery Log](image)

   **Figure 2-6. Subsystem NQN**

5. Connect to the discovered NVMe-oF target (`nvme-qlogic-tgt1`) using NQN.
   
   ```
   # nvme connect -t rdma -n nvme-qlogic-tgt1 -a 1.1.1.1 -s 1023
   ```
6. Confirm the NVMe-oF target connection with the NVMe-oF device, as shown in Figure 2-7.
   
   ```
   # dmesg | grep nvme
   # lsblk
   ```

   ![Figure 2-7. Confirm NVMe-oF Connection](image)

   **Figure 2-7. Confirm NVMe-oF Connection**

### Precondition the Target Server

NVMe target servers that are tested out-of-the-box will show higher than expected performance. Before running a benchmark, the target server needs to be **prefilled** or **preconditioned**.

**To precondition the target server:**

1. Secure-erase the target server with vendor specific tools (similar to formatting). This test example uses an Intel NVMe SSD device, which requires the Intel Data Center Tool that is available at the following link:

   [https://downloadcenter.intel.com/download/23931/Intel-Solid-State-Drive-Data-Center-Tool](https://downloadcenter.intel.com/download/23931/Intel-Solid-State-Drive-Data-Center-Tool)

2. Precondition the target server (nvme0n1) with data, which will guarantee that all available memory is filled. This example uses "DD" disk utility:

   ```
   # dd if=/dev/zero bs=1024k of=/dev/nvme0n1
   ```
Test the NVMe-oF Devices

Comparing the latency of the local NVMe device on the target server with that of the NVMe-oF device on the initiator server, shows the latency that NVMe adds to the system.

1. Update the repo source and install the Flexible Input/Output (FIO) benchmark utility on both the target and initiator servers.

   ```bash
   # yum install epel-release
   # yum install fio
   ```

   ![FIO Utility Installation](image)

   **Figure 2-8. FIO Utility Installation**

2. Run the FIO utility to measure the latency of the initiator NVMe-oF device.

   ```bash
   # fio --filename=/dev/nvme0n1 --direct=1 --time_based
   --rw=randread --refill_buffers --norandommap --randrepeat=0
   --ioengine=libaio --bs=4k --iodepth=1 --numjobs=1
   --runtime=60 --group_reporting --name=temp.out
   ```

   FIO reports two latency types: submission and completion. Submission latency (slat) measures application-to-kernel latency. Completion latency (clat), measures end-to-end kernel latency. The industry-accepted method is to read clat percentiles in the 99.00th range.

3. Run FIO to measure the latency of the local NVMe device on the target server.

   ```bash
   # fio --filename=/dev/nvme0n1 --direct=1 --time_based
   --rw=randread --refill_buffers --norandommap --randrepeat=0
   --ioengine=libaio --bs=4k --iodepth=1 --numjobs=1
   --runtime=60 --group_reporting --name=temp.out
   ```
Optimize Performance

To optimize performance on both initiator and target servers:

1. Configure the system BIOS settings.
   - Power Profiles = 'Max Performance' or equivalent,
   - ALL C-States = disabled
   - Hyperthreading = disabled

2. Configure the Linux kernel parameters by editing the grub file (/etc/default/grub).
   a. Add parameters to end of line GRUB_CMDLINE_LINUX:
      
      ```
      GRUB_CMDLINE_LINUX="nosoftlockup intel_idle.max_cstate=0
      processor.max_cstate=1 mce=ignore_ce idle=poll"
      ```
   b. Save the grub file.
   c. Rebuild the grub file. To rebuild the grub file for a legacy BIOS boot, type the following command:
      
      ```
      # grub2-mkconfig -o /boot/grub2/grub.cfg  (Legacy BIOS boot)
      ```
      
      To rebuild the grub file for an EFI boot, type the following command:
      
      ```
      # grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg  (EFI boot)
      ```
   d. Reboot the server to implement the changes.

3. Set the IRQ affinity for all QLogic 45000/41000 Series adapters.
   - multi_rss-affin.sh is a script file that is listed in “IRQ Affinity (multi_rss-affin.sh)” on page 15.
   
   ```
   # systemctl stop irqbalance
   # ./multi_rss-affin.sh eth1
   ```

4. Set CPU frequency. cpufreq.sh is a script file that is listed in “CPU Frequency (cpufreq.sh)” on page 16.
   
   ```
   # ./cpufreq.sh
   ```
5. Configure the network/memory settings:

sysctl -w net.ipv4.tcp_mem="16777216 16777216 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.core.wmem_max=16777216
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_default=16777216
sysctl -w net.core.rmem_default=16777216
sysctl -w net.core.optmem_max=16777216
sysctl -w net.ipv4.tcp_low_latency=1
sysctl -w net.ipv4.tcp_timestamps=0
sysctl -w net.ipv4.tcp_sack=1
sysctl -w net.ipv4.tcp_window_scaling=0
sysctl -w net.ipv4.tcp_adv_win_scale=1

# echo noop > /sys/block/nvme0n1/queue/scheduler
# echo 0 > /sys/block/nvme0n1/queue/add_random
# echo 2 > /sys/block/nvme0n1/queue/nomerges

NOTE

The following commands apply only to the initiator server.
Optimization Scripts

The following scripts are used in “Optimize Performance” on page 26, Steps 3 and 4, respectively.

**IRQ Affinity (multi_rss-affin.sh)**

```bash
#!/bin/bash
#RSS affinity setup script
#By Vladislav Zolotarov, Broadcom, Oct 2007
#Upgraded to multy interface by Nir Goren. QLogic, Jul 2014
#input: the device name (ethX)
OFFSET=0   0/1   0/1/2   0/1/2/3
FACTOR=1    2      3       4
OFFSET=0
FACTOR=1
LASTCPU=`cat /proc/cpuinfo | grep processor | tail -n1 | cut -d:"" -f2`
MAXCPUID=`echo 2 $LASTCPU ^ p | dc`
OFFSET=`echo 2 $OFFSET ^ p | dc`
FACTOR=`echo 2 $FACTOR ^ p | dc`
CPUID=1

for eth in $*; do

NUM=`grep $eth /proc/interrupts | wc -l`
NUM_FP=$((${NUM}))
INT=`grep -m 1 $eth /proc/interrupts | cut -d: -f1`
echo "$eth: ${NUM} (${NUM_FP} fast path) starting irq ${INT}"

CPUID=$((CPUID*OFFSET))
for ((A=1; A<=${NUM_FP}; A=${A}+1)) ; do
INT=`grep -m $A $eth /proc/interrupts | cut -d:"" -f1`
```
```
CPU Frequency (cpufreq.sh)

The following script sets the CPU frequency.

```bash
SMP=`echo $CPUID 16 o p | dc`
 echo ${INT} smp affinity set to ${SMP}
 echo $($(($SMP))) > /proc/irq/$($(($INT)))/smp_affinity
 CPUID=$(($CPUID*FACTOR))
 if [ ${CPUID} -gt ${MAXCPUID} ]; then
 CPUID=1
 CPUID=$(($CPUID*OFFSET))
 fi
 done
```

CPU Frequency (cpufreq.sh)

The following script sets the CPU frequency.

```bash
#Usage "./nameofscript.sh"
grep -E 'model name|cpu MHz' /proc/cpuinfo
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do 
 [ -f $CPUFREQ ] || continue; echo -n performance > $CPUFREQ; done
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```