NVM Express over Fabrics for Intel® Ethernet Products with RDMA

Configuration Guide

Ethernet Products Group (EPG)

May 2021

Revision 2.1
608515-002

Revision History

Revision	Date	Comments
2.1	May 7, 2021	Document title change only.
2.0[1]	July 23, 2020	Initial public release.

1. There are no previous publicly-available versions of this document.

Contents

NOTE: ***This page intentionally left blank.***

1.0 Introduction

NVM Express* (NVMe*) drives are high-speed, low-latency, solid-state drives (SSDs), that connect over the server Peripheral Component Interconnect Express* (PCIe*) bus.

The development of these high-performance drives has spurred new innovation in storage over networking protocols, which takes full advantage of the drive capabilities in data center and cloud environments.

NVMe over Fabrics provides networked storage at a latency level close to locally-mounted storage through a re-architected storage protocol that combines the use of low-latency/high-efficiency fabric technologies such as Remote Direct Memory Access (RDMA) or Fibre Channel (FC) with these high-speed NVMe drives.

Intel supports NVMe over Fabrics on two Intel® Ethernet product lines with RDMA technology:

- Intel® Ethernet 800 Series
- Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722.

1.1 Purpose

This document is a reference guide for configuring NVMe over Fabrics on Linux* operating systems using Intel® Ethernet 800 Series or Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722.

1.2 Terminology

Table 1. Terminology

Term	Description
BIOS	Basic Input Output System
DUT	Device Under Test
FC	Fibre Channel
IP	Internet Protocol
LBA	Logical Block Address
LUN	Logical Unit Number
NVM Express	Non-Volatile Memory Express*
NVMe	
PCI Express*	Peripheral Component Interconnect Express*
PCIe	
RDMA	Remote Direct Memory Access
RHEL	Red Hat* Enterprise Linux
SELinux	Security Enhanced Linux
SDD	Solid State Drive

2.0 Prerequisites

2.1 Hardware Prerequisites

Target server platform:

- Intel® Xeon® scalable processors
- Intel® Ethernet 800 Series or Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722
- 40+ GB RAM
- 1+ PCIe Gen3 SSD with NVMe high performance controller interface

Client server platform(s):

- Intel® Xeon® scalable processors
- Intel® Ethernet 800 Series or Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722
- 40+ GB RAM

2.2 Software Prerequisites

- Red Hat Enterprise Linux (RHEL) 7.x or similar (guide was tested with RHEL 7.5)
- Latest stable kernel: https://www.kernel.org (recommend 4.18 or greater)
- **nvmetcli**: ftp://ftp.infradead.org/pub/nvmetcli/
- **configshell_fb** (required to setup **nvmetcli**): https://github.com/openiscsi/configshell-fb/releases
- **nvme-cli**: https://github.com/linux-nvme/nvme-cli/releases
- **fio**: https://github.com/axboe/fio/releases
- Latest driver/NVM upgrades for the network interface under test (either Intel® Ethernet 800 Series, or Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722). Refer to the driver *README* files for driver-specific installation requirements and dependencies.

3.0 Upgrade Kernel (All Servers)

NVMe over Fabrics requires the systems to be on a recent stable kernel (4.18+ is recommended) from kernel.org for NVMe over Fabrics' latest patches/fixes.

3.1 Create Kernel *.config* File

1. Ensure that the **ncurses-devel** and **openssl-devel** packages are installed:

```
yum install ncurses-devel
yum install openssl-devel
```

2. Make the config file based on current settings:

```
cd <path to kernel>
make olddefconfig
```

3. Change the config file either manually or through make **menuconfig** to ensure the following options are set in the *.config* file:

```
grep NVM .config
CONFIG_NVME_CORE=m
CONFIG_BLK_DEV_NVME=m
CONFIG_BLK_DEV_NVME_SCSI=y
CONFIG_NVME_FABRICS=m
CONFIG_NVME_RDMA=m
CONFIG_NVME_TARGET=m
CONFIG_NVME_TARGET_LOOP=m
CONFIG_NVME_TARGET_RDMA=m
# CONFIG_NVM is not set CONFIG_NVMEM=m
```

3.2 Build Kernel

1. Save the config file and build the OS:

```
make -j 8
make modules_install -j 8
make install -j 8
```

2. Reboot into the updated kernel.

3.3 BIOS Tunings

For best performance with NVMe over Fabrics, the following BIOS settings are recommended (the exact names might change according to the platform make/model):

- Disable hyper-threading (logical processors)
- Disable power management:
  - Set the power profile to **Performance**, enable **Turbo**, and disable the **Cstates**.

4.0 Configure and Test RDMA (All Servers)

4.1 Install Intel LAN Driver, RDMA Driver, and Related Dependencies

Download the latest Linux driver package from Intel for the Device Under Test (DUT) and follow the installation procedure outlined in the included RDMA *irdma README* file to install the LAN driver, dependencies, and RDMA driver.

Notes:

- Intel® Ethernet 800 Series supports both RoCEv2 and iWARP RDMA technologies. Refer to the *irdma README* file for instructions on how to select which technology on driver load.
- Intel® Ethernet Connection X722/Intel® Ethernet Network Adapter X722 supports only iWARP RDMA.

4.2 Upgrade NVM to Latest Image on NIC

Download the latest NVM upgrade package from Intel for the DUT and follow the included documentation to perform the upgrade.

4.3 Disable SELinux* and Firewall

When running performance testing, disabling the firewall and Security-Enhanced Linux (SELinux) is recommended for highest performance.

1. Disable the firewall:

```
systemctl stop firewalld
systemctl mask firewalld
```

2. Disable SELinux by editing the following file and changing **enforcing** to **disabled**:

```
vi /etc/selinux/conf
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing - SELinux security policy is enforced.
# permissive - SELinux prints warnings instead of enforcing.
# disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three two values:
# targeted - Targeted processes are protected,
# minimum - Modification of targeted policy. Only selected processes are
protected.
# mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

4.4 Enable Link-Level Flow Control (LLFC)

While not a strict requirement, it is recommended to enable LLFC for RDMA performance testing for best performance.

1. Enable flow control on the adapter using **ethtool -A** command:

   ```
   ethtool -A <ethx> rx on tx on
   ```

2. Confirm the setting with the **ethtool -a** command:

   ```
   ethtool -a <ethx>
   ```

   You should see the following output:

   ```
   Pause parameters for <ethx>:
   Autonegotiate: off
   RX: on
   TX: on
   ```

Note: When connected to a switch, the switch port must also be configured to enable LLFC on both Rx and Tx per port. To enable link-level flow control on the switch, refer to your switch vendor's documentation. Depending on the switch vendor, the technology may be called pause, LLFC, or flow control.

4.5 Check RDMA

1. Ensure that the RDMA interfaces listed on each server are shown when running the following command:

   ```
   ibv_devices
   ```

2. Use **rping** to check for RDMA connectivity between target interface and client interface:

   a. Assign IPs to the RDMA interfaces on Target and Client.

   b. On Target, run the following:

      ```
      rping -sdVa <targetIP>
      ```

   c. On Client run the following:

      ```
      rping -cdVa <targetIP>
      ```

3. Press **Ctrl-C** to exit **rping**.

5.0 Configure NVMe over Fabrics Target (Storage Server)

5.1 Install NVMe over Fabrics Tools

5.1.1 Install nvme-cli

1. Download **nvme-cli** from:

   https://github.com/linux-nvme/nvme-cli/releases

2. Install with the following command:

   ```
   cd /<path_to_nvme-cli>/; python setup.py install
   ```

5.1.2 Install nvmetcli

1. Download **configshell_fb** (required to setup **nvmetcli**) from:

   https://github.com/open- iscsi/configshell-fb/releases

2. Install with the following command:

   ```
   cd /<path_to_configshell-fb>/; python setup.py install
   ```

3. Download **nvmetcli** from:

   ftp://ftp.infradead.org/pub/nvmetcli/

4. Install with the following command:

   ```
   cd /<path_to_nvmetcli>/; python setup.py install
   ```

5.2 Configure NVMe Drives

5.2.1 Install Latest Drivers and Firmware for NVMe Drives

Follow NVMe drive manufacturer instructions.

5.2.2 Format NVMe Drives

Reformatting NVMe drives can be done using **nvme-cli**.

The following example code snippet formats all NVMe partitions on the server using **nvme-cli**:

```
for device in $(ls /dev/nvme*n*p*); do
   nvme format $device
done
```

5.2.3 Partition NVMe Drives

Partition the NVMe drives into as many partitions as needed using **gdisk**. Refer to the **gdisk** documentation for more details.

The following example script partitions all NVMe drives on the server into a specified number of equal partitions. This removes all data on the drives.

add_partitions_gdisk.sh

```
#!/bin/bash partitions=2
for device in $(ls /dev/nvme*n* | grep -v p); do
    inputString="" echo "$device"
    max_sectorsize=$(echo -e "n\n"| gdisk $device |grep sector |grep -oP '(?<=34-).*?(?=,)')
    partNum=$((max_sectorsize/partitions))
    for (( i=1; i<= $partitions; i++))do
        partSize=$((partNum*i))
        echo $partSize
        inputString+="n\n\n\n\n$partSize\n\n"
    done
    inputString+="w\ny\n"
    printf $inputString | gdisk $device
done
```

5.3 Configure NVMe Target System

5.3.1 Load Modules

Load these modules before setting up the subsystems:

```
modprobe nvme nvmet null_blk nvmet_rdma
```

5.3.2 Configure NVMe Subsystems

Note: Refer to the **nvmetcli** documentation for the latest instructions.

There are two options for creating NVMe subsystems using **nvmetcli**:

- Use the **nvmetcli** interactive menu.
- Create the configuration file and use **nvmetcli** restore to load the file.

5.3.3 Create Subsystems Using nvmetcli Interactive Commands

The following example provides commands that interactively configure an NVMe subsystem with a single Logical Unit Number (LUN), where 4420 is the default port number for NVMe over Fabrics, and 10.10.10.20 is the IP Address for the target Intel® Ethernet Connection X722 interface:

```
nvmetcli
/> cd subsystems
/subsystems> create nvme4n1p1
/subsystems> cd nvme4n1p1/namespaces
/subsystems/n...p1/namespaces> create nsid=1
/subsystems/n...p1/namespaces> cd 1
/subsystems/n.../namespaces/1> set device path=/dev/nvme4n1p1
Parameter path is now '/dev/nvme4n1p1'.
/subsystems/n.../namespaces/1> cd .././/
/subsystems/n...p1/namespaces> cd ../
/subsystems/nvme4n1p1> set attr allow_any_host=1
Parameter allow_any_host is now '1'.
/subsystems/nvme4n1p1> cd namespaces/1
/subsystems/n.../namespaces/1> enable
The Namespace has been enabled.
/subsystems/n.../namespaces/1> cd ../../../..
/> cd ports
/ports> create 1
/ports> cd 1
/ports/1> set addr adrfam=ipv4
```

```
Parameter adrfam is now 'ipv4'.
/ports/1> set addr trtype=rdma
Parameter trtype is now 'rdma'.
/ports/1> set addr trsvcid=4420
Parameter trsvcid is now '4420'.
/ports/1> set addr traddr=10.10.10.20
Parameter traddr is now '10.10.10.20.
/ports/1> cd subsystems
/ports/1/subsystems> create nvme4n1p1
```

To save the target configuration to a file:

```
/ports/1/subsystems> saveconfig manual-config.json
/ports/1/subsystems> exit
```

5.3.4 Use nvmetcli to Load Saved Configuration

Create NVMe subsystems using **nvmetcli** restore [config file]:

```
nvmetcli restore savedconfig.json
```

Note: Refer to Section 8.0 for sample script to generate a saved config file automatically.

5.3.5 Clear NVMe Subsystems Using nvmetcli

To clear all NVMe subsystems:

```
nvmetcli clear
```

6.0 Configure NVMe over Fabrics Client(s)

6.1 Install NVMe over Fabrics Tools

6.1.1 Install nvme-cli

1. Download nvme-cli from:

   https://github.com/linux-nvme/nvme-cli/releases

2. Install with the following command:

   ```
   cd /<path_to_nvme-cli>/; python setup.py install
   ```

6.2 Load Modules

Load these modules before setting up the subsystems:

```
modprobe configfs
modprobe nvme
modprobe nvme_rdma
```

6.3 Connect NVMe Drives

1. Discover NVMe drives available for connection:

   ```
   nvme discover -t rdma -a <targetIP> -s 4420
   ```

2. Connect the client to the target and mount an NVMe drive on the client:

   ```
   nvme connect -t rdma -s 4420 -a <targetIP> -n <target_disk_nqn>
   ```

   For example:

   ```
   nvme connect -t rdma -s 4420 -a 10.10.10.20 -n /dev/nvme2n1p1
   ```

6.4 Verify NVMe over Fabrics Connections

To verify drives are mounted, run the following commands:

```
nvme list
lsblk
```

Note: By default, regardless of the name or NQN of the subsystem on the target, the client-mounted subsystems are be named */dev/nvme[#]n1;* where *[#]* is a number starting at 0 (or the lowest available if other NVMe drives are on the system) and incrementing as more NVMe drives are added/mounted.

7.0 Testing NVMe over Fabrics

Because NVMe over Fabrics is a block-based storage protocol, a standard block storage benchmark such as **fio** can be used to test performance.

7.1 Install fio

1. Download **fio**:

   https://github.com/axboe/fio/releases

2. Install with the following command:

```
cd /<path_to_fio>/;./configure; make; make install
```

7.2 Precondition NVMe Drives

If performance testing on NAND-based NVMe drives, preconditioning the drives must be completed. If performance testing on Intel® Optane™ media-based NVMe drives, no preconditioning is required.

For the best performance results, when using NAND-based NVMe drives, precondition the drives first using the **fio** tool.

To precondition the drives, fill up the entire span of the drive with sequential writes twice to ensure each of the Logical Block Addresses (LBAs) is written to.

Note: To benchmark 4 KB random writes, precondition the drives with 4 KB random writes instead of 1 MB sequential writes.

The following example **fio** command preconditions an NVMe drive (Sequential Writes: 1 MB Write, 32 IO depth, 1200 seconds for a 800 GB NVMe drive):

```
fio --filename=/dev/nvme0n1 --numjobs=1 --rw=write --iodepth=32 --bs=1M -runtime=1200
```

7.3 fio Example

Note: For the latest instructions, refer to the **fio** documentation included in the version being used.

Following is an example **fio** test (tests 4 KB read, 16 IO depth, 120 seconds, 4subsystems):

Run on the client:

```
fio fio.conf
```

Where *fio.conf* contains:

```
[global]
ioengine=libaio
rw=randread
iodepth=16
bs=4K
rwmixread=100
direct=1
time_based=1
runtime=120s
norandommap=1
numjobs=1
[filename1]
filename=/dev/nvme0n1
```

```
[filename2]
filename=/dev/nvme1n1
[filename3]
filename=/dev/nvme2n1
[filename4]
filename=/dev/nvme3n1
```

8.0 Example Script to Generate nvmetcli Subsystem Config File

Note: This script is provided as example only. Modify the script as needed to fit the environment.

```
gen_target_config_kernel_doc.sh
#This script creates nvmetcli configuration file that uses all nvme partitions in
/dev/nvme*n*p* for target subsystem creation.
#Note: be sure to change $ip to match your RDMA interface IP
#On NVMe target, after running this script, use 'nvmetcli restore
<filen>' to create nvme subsystems

#file name for config file.
filen="nvme-target-setup-auto"

#RDMA target interface IP
ip="10.10.10.20"

#get all NVMe drive partition paths
nvmePartitions=( $(ls /dev/nvme*n*p* | cut -d'/' -f3) )

#get last partition path in the list
lastnvmePartition=${nvmePartitions[$(( ${#nvmePartitions[*]} - 1
))]}

#create config file
cat > $filen << EOF
{
    "hosts": [],
    "ports": [
        {
        "addr": {
            "adrfam": "ipv4",
        "traddr": "$ip",
        "treq": "not specified",
        "trsvcid": "4420",
        "trtype": "rdma"
        },
    "portid": 1,
    "referrals": [],
    "subsystems": [
EOF

for dev in "${nvmePartitions[@]}"; do
if [[ $dev == "$lastnvmePartition" ]]; then
cat >> $filen << EOF
    "${dev}"
EOF
else
cat >> $filen << EOF
    "${dev}",
EOF
fi
done
```

```
cat >> $filen << EOF
          ]
       }
   ],
   "subsystems: [
EOF

i=1
for dev in "${nvmePartitions[@]}"; do
cat >> $filen << EOF
   {
      "allowed_hosts": [],
      "attr": {
         "allow_any_host": "1"
      },
      "namespaces": [
      {
         "device": {
            "nguid": "00000000-0000-0000-0000-000000000000",
            "path": "/dev/$dev"
         },
      "enable": 1,
      "nsid": 1
      }
   ],
   "nqn": "$dev"
EOF

if [[ $dev == "$lastnvmePartition" ]]; then
cat >> $filen << EOF
   }
EOF
else
cat >> $filen << EOF
   },
EOF
fi
done

cat >> $filen << EOF
   ]
}
EOF
```

LEGAL