NVM Express over Fabrics with SPDK for Intel® Ethernet Products with RDMA

Configuration Guide

Ethernet Products Group (EPG)

May 2021

Revision 2.3
613986-002

Revision History

Revision	Date	Comments
2.3	May 7, 2021	Document title change only.
2.2[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 (NVMe-oF) 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.

The Storage Performance Development Kit (SPDK) provides a set of tools and libraries for writing high-performance, scalable, user-mode storage applications. SPDK is an open-source project focused on optimizing storage software for the latest generation CPUs, NVMe SSDs, and NICs to improve the performance and efficiency of storage applications. It achieves high-performance by moving all of the necessary storage drivers into userspace and operating in a polled mode instead of interrupts, which avoids kernel context switches and eliminates interrupt handling overhead.

The SPDK community started with a userspace, polled-mode, asynchronous, lock-less NVMe driver and then extended the performance and efficiencies of SPDK to storage networking and virtualization. SPDK provides a high-performant NVMe-oF target and host components that are spec compliant. Therefore, users can use the Linux Kernel NVMe-oF host to connect to an SPDK NVMe-oF target and vice versa.

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 the SPDK NVMe over Fabrics target application on the 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
DPDK	Data Plane Development Kit
DUT	Device Under Test
FC	Fibre Channel
IP	Internet Protocol
LBA	Logical Block Address
LFC	Link-Level Flow Control
LUN	Logical Unit Number
NVM Express	Non-Volatile Memory Express*
NVMe	

Table 1. Terminology [continued]

Term	Description
NVMe-oF	NVMe over Fabrics
PCI Express*	Peripheral Component Interconnect Express*
PCIe	
PFC	Priority Flow Control
RDMA	Remote Direct Memory Access
RHEL	Red Hat* Enterprise Linux
SELinux*	Security Enhanced Linux
SSD	Solid State Drive
SPDK	Storage Performance Development Kit

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

Host 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/open-iscsi/configshell-fb/releases
- **nvme-cli**:

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

  https://github.com/axboe/fio/releases
- SPDK (this document is tested with v19.10.1):

  https://github.com/spdk/spdk
- DPDK: Included as a submodule in SPDK package or can be downloaded from:

  https://github.com/DPDK/dpdk
- 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
```

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 power management:
  - Set the power profile to **Performance**, enable **Turbo**, and disable the **C-states**.

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** (requires reboot):

```
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 Flow Control

It is recommended to enable Link-Level Flow Control (LFC) or Priority Flow Control (PFC) for RDMA performance testing for best performance. LFC is provided here as a reference for simplicity.

1. Enable flow control on the adapter using the **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 LFC 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 host interface:

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

   b. On Target, run the following:

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

   c. On Host, run the following:

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

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

5.0 Configure NVMe Over Fabrics SPDK Target

5.1 Configure NVME Drives

5.1.1 Install Latest Drivers and Firmware for NVMe Drives

Follow NVMe drive manufacturer instructions.

5.1.2 NUMA node and Distribution of NVMe Drives over PCIe Lanes

For performance testing, it is recommended that the NVMe drives are on the same NUMA node as that of the NIC.

To check the NUMA node of a device:

```
cat /sys/bus/pci/devices/<pcie_bus_address>/numa_node
```

In all storage performance tests, it is important to ensure that the storage drives are not introducing a performance bottleneck. One way to test this (other than by checking the specifications of the storage drives) is to first run the benchmarks locally on the storage target without involving the network.

Another common bottleneck is on the PCIe bus. It is recommended to ensure that the NVMe drives are distributed uniformly across the NVMe controller's PCIe lanes, and that each drive has enough PCIe bandwidth to drive full performance.

lspci tree view can be used to check the distribution of the NVMe drives in the system.

In this example, the six Intel® Optane™ drives are distributed as two on the first controller, two on the second controller, and two on the third controller.

```
lspci -tvv | grep -B4 -i nvme
-+-[0000:d7]-+-00.0-[d8-dd]----00.0-[d9-dd]--+-04.0-[da]--
 |           |                                +-05.0-[db]--
 |           |                              +-06.0-[dc]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
 |           |                              \-07.0-[dd]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
--

+-[0000:ae]-+-00.0-[af-b4]----00.0-[b0-b4]--+-04.0-[b1]--
 |           |                                +-05.0-[b2]--
 |           |                              +-06.0-[b3]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
 |           |                              \-07.0-[b4]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
--

+-[0000:5d]-+-02.0-[5e-63]----00.0-[5f-63]--+-04.0-[60]--
 |           |                                +-05.0-[61]--
 |           |                              +-06.0-[62]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
 |           |                              \-07.0-[63]----00.0  Intel Corporation NVMe Datacenter SSD [Optane]
```

5.1.3 Format NVMe Drives

Reformatting NVMe drives can be done using a tool preset in the SPDK package **nvme-manage**.

Within the SPDK package, execute the following command:

```
./examples/nvme/nvme_manage/nvme_manage
```

To format the drives, use Option 6.

To erase the drives, use the “No secure erase operation requested” option.

To format the NVMe drive into 4 KB block size, use Option 1.

To format the NVMe drive into 512-byte block size, use Option 0.

Note: The SPDK setup script must be run to use the tools included in the SPDK package, such as the **nvme-manage** tool.

5.1.4 Precondition NVMe Drives on the SPDK Target

For performance testing on NAND-based NVMe drives, preconditioning the drives is recommended for accurate and predictable results. If performance testing on 3D XPoint-based NVMe drives, such as Intel® Optane™, no preconditioning is required.

fio can be used to precondition the drives.

To benchmark random reads, fill up the entire span of the drive with sequential writes twice to ensure that each Logical Block Address (LBA) has been written to. For a 2 TB drive about 50 minutes of writing is sufficient. Following is a sample command for a drive */dev/nvme0n1*:

```
fio --filename=/dev/nvme0n1 --direct=1 --rw=write --bs=128K --numjobs=4 --iodepth=32
--size=100% --loops=2 --runtime=1200 --ramp_time=60 --time_based --ioengine=libaio -
output-format=normal
```

5.2 Install SPDK NVMe over Fabrics

5.2.1 Install SPDK

Refer to the *README* file included with SPDK for the latest installation instructions here:

https://spdk.io/doc/getting_started.html

Get the latest SPDK release from:

https://github.com/spdk/spdk

The following is provided as a reference for SPDK release 19.10.1.

1. Download the tarball of the latest release:

   ```
   wget https://github.com/spdk/spdk/archive/<v19.10.1.tar.gz>
   ```

2. Extract the downloaded files to a folder:

   ```
   tar -xvfz <spdk-filename>

   cd <spdk-filename>
   git submodule update --init
   ```

3. Install the prerequisites automatically using the following command:

   ```
   ./scripts/pkgdep.sh
   ```

   Note: Ensure that the **pkgdep** script does not install conflicting libraries.

4. To build SPDK with a different version of DPDK than what is included in the submodule, execute the following command:

   ```
   ./configure --with-dpdk=/path-to-dpdk/x86_64-native-linuxapp.gcc DESTDIR=.
   ```

   Note: DPDK submodule included in the SPDK release was used in testing for this document.

5. To build SPDK on Linux with RDMA support, execute the following command:

```
./configure --with-rdma

make
```

Note: **uuid**, **uuid-devel**, **libuuid**, **libuuid-devel**, **CUnit**, and **CUnit-devel** packages are required to install SPDK.

6. To allocate huge pages and to unbind the native Linux kernel drivers for NVMe and I/OAT devices, use the following command:

```
./scripts/setup.sh
```

Note: This script unbinds the NVMe SSDs from the Linux Kernel driver and binds them to the UIO/VFIO driver.

7. To check the status of the NVMe drives, execute the following command:

```
./scripts/setup.sh status
```

8. To allocate huge pages (in MB) other than the default of 2048 MB, execute the following command:

```
HUGEMEM=4096 ./scripts/setup.sh
```

9. If it is desired to revert back to non-SPDK drivers (after SPDK testing is finished), the following command can be used:

```
./scripts/setup.sh reset
```

5.3 Configure SPDK NVMe over Fabrics Target System

SPDK NVMe over Fabrics targets can be configured either by dynamically using SPDK's JSON-RPC (Remote Procedure Call) API or (deprecated) creating and loading a configuration file. Either method creates the target framework including namespaces, subsystems, and block devices (bdevs) for the NVMe drives on the target.

An NVMe namespace is a storage volume that can be formatted into logical blocks. Namespaces are built on top of the SPDK block device (bdev) abstraction layer that acts similar to the operating systems' block storage layer in a traditional kernel storage stack (https://spdk.io/doc/bdev.html).

An NVMe subsystem can include one or more NVMe namespaces, NVMe controllers, NVM subsystem ports, an NVM storage medium, and an interface between the controller and the NVM storage medium. An NVMe over Fabrics target can contain one or more NVMe subsystems.

RPC methods to set up the SPDK target are explained in Section 5.3.1.

Setting up the SPDK target via the SPDK configuration file method is described in Section 5.3.2, and a sample configuration file is provided in Section 7.0.

5.3.1 Option 1: Configure NVMe SPDK Target Subsystems Using RPC Methods

SPDK implements JSON-RPC methods to dynamically configure the SPDK components.

Note: Refer to the SPDK documentation for the latest instructions.

1. Start the `nvmf_tgt` application with elevated privileges and (optionally) run on specific CPU cores (`-m`). Note that for best performance it is recommended to run the SPDK target on CPU cores local to the NUMA node of the NVMe drives being used. The number of cores used by the target is a tunable value that can be adjusted based on experimentation.

   ```
   ./app/nvmf_tgt/nvmf_tgt -m[2,4,6,8…]
   ```

2. Set the target with a polling interval for incoming connections (`-r`) (microseconds):

   ```
   ./scripts/rpc.py nvmf_set_config -r 10000
   ```

3. Once the target is started, the `nvmf_create_transport` RPC can be used to initialize a given transport. We are using the RDMA transport option (`-t RDMA`) with a queue depth of 32 (`-q`) and 1023 pooled data buffers available to the transport (`-n`).

   ```
   ./scripts/rpc.py nvmf_create_transport -t RDMA -q 32 -n 1023
   ```

4. Create a subsystem with an user-selected NQN address (for example, `nqn.2019-06.io.spdk:cnode1`):

   ```
   ./scripts/rpc.py nvmf_create_subsystem nqn.2019-06.io.spdk:cnode1
   ```

5. [Optional] Set bdev NVMe options with how often the admin queue is polled for asynchronous events (`-p`), the number of attempts per I/O when an I/O fails (`-n`) and what action to take on time out (`-a`) while tracking timeouts (`-t`):

   ```
   ./scripts/rpc.py bdev_nvme_set_options -n 4 -t 0 -a none -p 100000
   ```

6. Start the subsystem initialization, if starting SPDK was deferred using `-w` option:

   ```
   ./scripts/rpc.py framework_start_init
   ```

7. List the NVMe drives that are available on the target:

   ```
   ./scripts/setup.sh status
   ```

8. Construct a bdev controller for each local NVMe device (with controller name `-b`) to be used on the target. The following command connects the local NVMe drives connected via PCIe (`-t`) with a bus address (`-a`). Note that the bdev name created uses the NVMe controller name as prefix, adds "n<#>", where <#> starts at 1 then increments by 1 for each PCIe drive attached. for example, Nvme0n1, Nvme0n2, and son on.

   ```
   ./scripts/rpc.py bdev_nvme_attach_controller -b Nvme0 -t pcie -a 0000:60:00.0
   ```

9. Create a subsystem namespace for each NVMe bdev using the subsystem NQN (`nqn.2019-06.io.spdk:cnode1`) and bdev_name that was created in the previous step (`Nvme0n1`):

   ```
   ./scripts/rpc.py nvmf_subsystem_add_ns nqn.2019-06.io.spdk:cnode1 Nvme0n1
   ```

10. [Optional] Enable (`-e`) any host to access the target subsystem NQN:

    ```
    ./scripts/rpc.py nvmf_subsystem_allow_any_host -e nqn.2019-06.io.spdk:cnode1
    ```

11. Add a listener address (`-a`) to a port (`-s`) with a specific transport (`-t`) to a subsystem NQN:

```
./scripts/rpc.py nvmf_subsystem_add_listener -t RDMA -a 13.100.15.1 -s 4420
nqn.2019-06.io.spdk:cnode1
```

At this point the target is set up with a single subsystem and a single namespace with a single drive attached. Steps can be repeated to add more subsystems and/or more namespaces to the same subsystem.

Detailed help for each command can be displayed by adding `-h` flag as a command parameter.

To get the currently-supported set of RPC commands directly from SPDK application, run:

```
./scripts/rpc.py rpc_get_methods
```

5.3.2 Option 2: Configure NVMe Target Subsystems Using a Configuration File

Note: This method is deprecated in current SPDK versions and is only offered here for reference. Refer to the SPDK documentation for the latest instructions.

To start the SPDK target using a configuration file, execute the following command:

```
/spdk-XX.XX/app/nvmf_tgt/nvmf_tgt -c <path_to_config_file>
```

Note: SPDK provides a sample configuration file that can be modified according to the user's environment. The sample/default file can be found within the SPDK package *etc/spdk/nvmf.conf.ini*, or refer to Section 7.0 for a full sample configuration file for the setup being used in this document.

The configuration file is divided into sections to set various options for the NVMe subsystems on the target.

1. The `[Nvmf]` section is used to set the NVMe over Fabrics Target with a polling interval for incoming connections (microseconds).

```
[Nvmf]
  AcceptorPollRate 10000
```

2. The `[Transport]` section can be used to set transport (RDMA) used to make the target devices available with queue depth and number of queues that can be used per RDMA call.

```
[Transport]
  Type RDMA
  MaxQueueDepth 32
  MaxQueuesPerSession 44
```

3. The `[NVMe]` section can be used to set the NVMe devices/bdevs on the target that can be found using `./scripts/setup.sh status`. This provides a list of local NVMe devices and their PCIe bus addresses (`traddr`) that can be used to add the NVMe devices here.

```
[Nvme]
  TransportID "trtype:PCIe traddr:0000:60:00.0" Nvme0
  TransportID "trtype:PCIe traddr:0000:61:00.0" Nvme1
  RetryCount 4
  TimeoutUsec 0
  ActionOnTimeout None
  AdminPollRate 100000
```

4. The `[Subsystem]` section can be used to map the namespace (`Nvme0n1`), controller (`SPDK_Controller1`) and port IP (`13.100.1.15:4420`) address to be used with that NVMe subsystem that can be identified with the NQN (`nqn.2019-06.io.spdk:cnode1`) set here.

```
[Subsystem1]
  NQN nqn.2019-06.io.spdk:cnode1
  Host nqn.2019-06.io.spdk:init
  Listen RDMA 13.100.1.15:4420
  AllowAnyHost Yes
  SN SPDK00000000000001
  MN SPDK_Controller1
  Namespace Nvme0n1 1
```

5.4 Clear the NVMe Target Subsystems Using SPDK

To clear all NVMe subsystems after finishing testing:

1. Remove all bdevs from target controller.

   For each bdev, run:

```
python3 /opt/spdk/scripts/rpc.py bdev_nvme_detach_controller ${bdev}
```

2. Stop *nvmeof_target* process.
3. Revert drives back to kernel drivers.

```
./scripts/setup.sh reset
```

6.0 Configure and Test NVMe over Fabrics Host(s) to Connect to SPDK Target

The SPDK NVMe-oF target system is spec compliant, which allows for the use of either an SPDK host or Linux Kernel host to connect to an NVMe-oF subsystem exported by the SPDK NVMe-oF target.

This section outlines both methods:

- Section 6.1, "Option 1: SPDK Host"
- Section 6.2, "Option 2: Linux Kernel Host"

6.1 Option 1: SPDK Host

6.1.1 Install fio for SPDK Host System

1. Download **fio** from https://github.com/axboe/fio/releases
2. Install **fio** on the host:

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

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

6.1.2 Install SPDK on the Host

Install SPDK on the host using the steps outlined in Section 5.2.

When installing SPDK on the host as described in Section 5.2, use the following `./configure` parameters to configure it with **fio** plugin:

```
./configure --with-fio=/opt/<path-to-fio>/ --with-rdma --enable-lto
```

6.1.3 Discover NVMe Drives Available on Target

Discover the NVMe drives available for connection over a transport (`-t`), with a target IP Address (`-a`) on a port (`-s`):

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

6.1.4 Connect NVMe-oF Drives

There are two different ways to connect NVMe over Fabrics drives to the SPDK host for testing.

Option 1: JSON-RPC methods

Use the following `bdev_nvme_attach_controller` command to attach the target controller using controller name (`-b`), transport fabric RDMA (`-t`), IP Address family IPv4 (`-f`), port number 4220 (`-s`), and target NQN (`nqn.2019-06.io.spdk:cnode1`):

```
rpc.py bdev_nvme_attach_controller -b Nvme0 -t RDMA -a <targetIP> -f IPv4 -s 4420
-n nqn.2019-06.io.spdk:cnode1
```

Option 2: fio plugin for benchmarking

If using fio benchmark for testing, the **fio** plugin handles the NVMe-oF drive connection dynamically during the benchmark run. See:

https://github.com/spdk/spdk/tree/master/examples/nvme/fio_plugin)

1. Create bdev.conf configuration file on the host side that lists the NVMe subsystems/bdevs that the **fio** host will connect to on the target during storage testing.

   Note: Multiple namespaces can be added to an NVMe subsystem, but only the bdev subsystems are required to be in the *bdev.conf* file.

   Example *bdev.conf*:

   ```
   [Nvme]
     TransportId "trtype:RDMA adrfam:IPv4 traddr:13.100.1.15 trsvcid:4420
     subnqn:nqn.2019-09.io.spdk:cnode1" Nvme0
     TransportId "trtype:RDMA adrfam:IPv4 traddr:13.100.1.15 trsvcid:4420
     subnqn:nqn.2019-09.io.spdk:cnode2" Nvme1
   ```

2. In the *fio.conf* file, include these lines:

   ```
   spdk_conf=<path to bdev.conf>
   ioengine=<path to fio-plugin>
   ```

3. On the host, run:

   ```
   fio <path to fio.conf>
   ```

   Example *fio.config* file for a test (4 KB read, 16 IO depth, 8 threads, 120 seconds, 2 subsystems */dev/nvme0n1 and /dev/nvme1n1*):

   ```
   [global]

   thread=1
   spdk_conf=<path to bdev.conf>
   ioengine=<path to fio-plugin>
   iodepth=16
   bs=4K
   numjobs=1
   ramp_time=10s
   runtime=120s
   rw=randread
   rwmixread=0
   iodepth_batch=8
   iodepth_batch_complete_max=16
   direct=1
   time_based=1
   thread=1
   [job1]
   filename=/dev/nvme0n1
   cpus_allowed=4,5,6,7,8,9
   [job2]
   filename=/dev/nvme1n1
   cpus_allowed=10,11,12,13,14
   ```

6.1.5 Disconnect the NVMe drives

1. Once testing is completed, if the drives are connected using rpc.py methods, disconnect the drives on the host from the target using the bdev names associated with them.

   ```
   ./scripts/rpc.py bdev_nvme_detach_controller <bdev>
   ```

2. Unbind the NVMe drives from the SPDK driver and bind them to the Linux kernel driver.

   ```
   ./scripts/setup.sh reset
   ```

6.2 Option 2: Linux Kernel Host

6.2.1 Install fio

1. Download fio from https://github.com/axboe/fio/releases
2. Install **fio** on the host.

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

6.2.2 Install nvme-cli

1. Download nvme-cli from https://github.com/linux-nvme/nvme-cli/releases
2. Install using the following command:

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

6.2.3 Load Modules

Load these modules before setting up the subsystems:

```
modprobe configfs
modprobe nvme
```

If using RDMA as the transport protocol, load:

```
modprobe nvme_rdma
```

6.2.4 Connect NVMe Drives

1. Discover the NVMe drives available for connection over a transport (`-t`), with a target IP Address (`-a`) on a port (`-s`):

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

2. Connect the host to the target and mount an NVMe drive on the host with the device NQNs (`-n`) found using `nvme discover`:

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

   For example:

   ```
   nvme connect -t rdma -s 4420 -a 13.100.1.15 -n /dev/nvme2n1p1
   ```

6.2.5 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 host-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 by 1.

6.2.6 Testing NVMe over Fabrics with Kernel Host

fio can be used to test performance on the kernel host as well. **fio** is used to measure the performance of a storage device over a given period of time.

Run **fio** test on the host:

```
fio <path to fio.conf>
```

Following is an example *fio.config* file for a test (4 KB read, 16 IO depth, 8 threads, 120 seconds, 2 subsystems */dev/nvme0n1 and /dev/nvme1n1*):

```
[global]

thread=1
group_reporting=1
rw=randread
iodepth=16
bs=4K
rwmixread=100
numjobs=8
direct=1
time_based=1
runtime=120s
norandommap=1
numjobs=8
[filename1]
filename=/dev/nvme0n1
[filename2]
filename=/dev/nvme1n1
```

6.2.7 Disconnect NVMe Drives

After finishing the required testing, disconnect the host from the target with the device NQNs (`-n`).

```
nvme disconnect -n "nqn.2016-06.io.spdk:cnode1"
```

7.0 Example SPDK Configuration File

Notes:

- This script is provided as an example only. Modify the script as needed to fit the environment.
- This sample configuration file is for two NVMe drives and using a NIC with one IP Address on one port.
- Sample configuration files are also available in the SPDK package under */etc/spdk*.
- It is required to comment out any bracketed section of the configuration file that is not being used for the setup.

nvmf.conf:

```
# NVMf Target Configuration File
# Please write all parameters using ASCII.
# The parameter must be quoted if it includes whitespace.

# Configuration syntax:
# Leading whitespace is ignored.
# Lines starting with '#' are comments.
# Lines ending with '\' are concatenated with the next line.
# Bracketed ([]) names define sections

[Nvmf]
  AcceptorPollRate 10000

[Transport]
  Type RDMA
  MaxQueueDepth 32
  MaxQueuesPerSession 44

[Nvme]
  TransportID "trtype:PCIe traddr:0000:60:00.0" Nvme0
  TransportID "trtype:PCIe traddr:0000:61:00.0" Nvme1
  RetryCount 4
  TimeoutUsec 0
  ActionOnTimeout None
  AdminPollRate 100000

[Subsystem1]
  NQN nqn.2019-06.io.spdk:cnode1
  Host nqn.2019-06.io.spdk:init
  Listen RDMA 13.100.1.15:4420
  AllowAnyHost Yes
  SN SPDK00000000000001
  MN SPDK_Controller1
  Namespace Nvme0n1 1

[Subsystem2]
  NQN nqn.2019-06.io.spdk:cnode2
  Host nqn.2019-06.io.spdk:init
  Listen RDMA 13.100.1.15:4420
  AllowAnyHost Yes
  SN SPDK00000000000002
  MN SPDK_Controller1
  Namespace Nvme1n1 1
```

LEGAL