Intel® NVMe™ SSDs and Intel® RSTe for Linux*

Quick Start Guide

January 2016

Order Number: 333745-001US

1 Overview

This document is intended to be used by OEMs/ODMs, technical analysts, and technology enthusiasts. It is a reference to evaluate Intel® NVMe™ solid-state drives (SSDs) utilizing Intel's® Rapid Storage Technology enterprise (Intel® RSTe) for NVMe RAID technology on the Linux* operating system, using the Linux mdadm application to configure, manage and monitor software RAID devices.

NVMe SSDs bring high-performance and low-latency PCIe® SSDs into the mainstream with a streamlined protocol that is efficient, scalable, and easily deployed. The Intel RSTe for NVMe application allows NVMe devices to be pooled into different RAID arrays, allowing for even greater performance and optional data redundancy for fault tolerant implementations.

This guide will help enable you to:

1. Locate Intel NVMe SSDs in the Linux Operating System.
2. Create NVMe Intel RSTe RAID containers and arrays using mdadm, populate with a file system and mount the new array for data access and performance testing.
3. Manage your NVMe RAID Arrays using the mdadm application.

This guide assumes you are experienced with the Linux Operating System and RAID. This guide does not address additional uses, options, or other configurations. For specific information outside of the scope of this document, contact your Intel Technical Marketing representative.

1.1 Features and Technology

At the time of publication, all Intel NVMe SSDs include the following:

- **Enhanced PLI (Power Loss Imminent)** – protection from unplanned power loss obtained by a propriety combination of hardware, firmware algorithms, and the robust validation at a multitude of corner cases.
- **Intel 20nm HET (High Endurance Technology) NAND** - for up to 5 DWPD (Drive Writes Per Day) over 5 years. The 4TB can write up to 21.9PB over the life of the drive.
- **End-to-end data protection and ECC** - for protection on all internal and external memories in the data path at every layer.
- **NVMe SMART monitoring and sideband health monitoring** – monitor the health of the SSD for attributes like temperature, endurance, power cycles, power on hours, and NVMe critical warnings. Sideband or out-of-band health monitoring enables OS agnostic monitoring through the standard NVM Express Management Interface Specification.

1.2 Prerequisites and System Considerations

At the time of publication, Intel RSTe for NVMe supports and qualifies the following operating systems:

- Red Hat Enterprise Linux* (RHEL) 6.7
- Red Hat Enterprise Linux (RHEL) 7.2
- SUSE Linux Enterprise Server* (SLES) 11 Service Pack (SP)4
- SUSE Linux Enterprise Server (SLES) 12 Service Pack (SP)1

All steps and examples in this document were conducted on an Intel® S2600WT Server with Red Hat Enterprise Linux (RHEL) 6.7.

For system hardware, ensure you have enough PCIe slots or U.2 connectors (formerly SFF-8639) available for the NVMe drives intended to be used to evaluate Intel NVMe SSDs with Intel RSTe for NVMe RAID on Linux.

To attain the highest possible performance, verify that all NVMe SSDs are connected to the same system processor.

2 Creating and Managing RAID Arrays with mdadm

WARNING: ***Creating a RAID array of any level will permanently delete any existing data on the selected drives. Back up all important data before proceeding with these steps.***

2.1 Intel Matrix RAID

Intel Matrix RAID Technology allows you to create two RAID volumes on a single RAID array.

As an example, a system with an Intel® C600-A chipset, Intel® Matrix Storage Manager (IMSM) allows you to create both a RAID 0 volume as well as a RAID 5 volume across four drives. An important requirement for Matrix RAID is that the volumes within the container span the same set of member drives. When creating an array with the IMSM metadata, use the mdadm application's '-e imsm' option.

Another feature of the IMSM metadata is the ability to roam an array between Linux and Microsoft® Windows* hosts.

Intel RSTe supports the following RAID levels:

- RAID 0
- RAID 1
- RAID 5
- RAID 10
- Intel® Matrix RAID

All instruction beyond this point assumes familiarity with the Linux Bash command line. For complete instruction on the mdadm options, use the help page (# mdadm --help) and man page (# man mdadm).

The following steps show an example of how to create a four-drive RAID5 array:

1. Create a container of Intel IMSM Metadata.

   For example:

   ```
   # mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm
   # Continue creating array? Y
   # mdadm: container /dev/md/imsm0 prepared
   ```

   This command creates a RAID container with Intel Matrix Storage Manager metadata format.

   The device node for the container will be /dev/md/imsm0.

   In this example NVMe drives 0,1,2,3 will be used for this RAID array, and the total number of drives is 4.

   The '/dev/nvme[0-3]n1' wildcard expression can be used to specify the range of drives. Individual drives can also be used to specify the container and array. For example:

   ```
   /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1 or, /dev/nvme[0123]n1
   ```

2. The RAID 5 array is created using full capacity of the drives. For example:

```
# mdadm -C /dev/md0 /dev/md/imsm0 -n 4 -l 5
# Continue creating array? y
# mdadm: array /dev/md0 started
```

This command creates a RAID 5 array '/dev/md0 within the /dev/md/imsm0' container.

2.2 Additional RAID Container and Array Creation Examples and Options

Create a 2 drive container and a RAID 0 array

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-1]n1 -n 2 -e imsm
# mdadm -C /dev/md0 /dev/md/imsm0 -n 2 -l 0
```

Create a 2 drive container and a RAID 1 array

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-1]n1 -n 2 -e imsm
# mdadm -C /dev/md0 /dev/md/imsm0 -n 2 –l 1
```

Create a 3 drive container and a RAID 5 array

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-2]n1 -n 3 -e imsm
# mdadm -C /dev/md0 /dev/md/imsm0 -n 3 -l 5
```

Create a RAID 10 container and a RAID 10 array (RAID 10 requires 4 drives)

```
# mdadm -C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm
# mdadm -C /dev/md0 /dev/md/imsm0 -n 4 -l 10
```

Note: To create multiple RAID volumes in the same container, they MUST span equal number of drives. For example, in order to have a RAID 0 volume and a RAID 5 volume in the same container, four drives must be used for both volumes.

The following command parameters can also be used in conjunction to give finer control for the creation of the RAID array:

-n	Number of active RAID drives to be used in the array.
-x	Specifies the number of spare devices in the initial array.
-c	Specifies the chunk (strip) size in multiples of 4Kib. The default is 128KiB.
-l	Specifies the RAID level. The supported levels are 0, 1, 5, and 10.
-z	Specifies the size (in Kilobytes) of space dedicated on each drive to the RAID volume. This must be a multiple of the chunk size. For example:

```
 # mdadm –C /dev/md0 /dev/md/imsm0 -n 3 -l 5 -z $((100*1024*1024))
```

The command above creates a RAID array inside the '/dev/md/imsm0' container with 100GiB of drive space used on each member drive.

A suffix of 'M' or 'G' can be given to indicate Megabytes or Gigabytes respectively. This also applies to the -C parameter. So the above command is equivalent to this:

```
 # mdadm –C /dev/md0 /dev/md/imsm0 -n 3 -l 5 -z 100G
```

2.3 Creating a File System

After the RAID array has been created, now a file system can be written to allow the array to be mounted. There are many different files systems to choose from. For this example, the Linux ext4 file system will be used:

```
# mkfs.ext4 /dev/md0
```

Once the file system is created the array can now be mounted to the location of your choice:

```
# mount /dev/md0 /mnt/<mount point>
```

2.4 Monitoring and Additional Features

The mdadm application offers many options to monitor arrays and related hardware, as well as additional options to configure and manipulate arrays. This section covers some of that functionality, but for a complete list of these options, consult the mdadm man page.

2.4.1 Monitoring

To find NVMe drives in the system:

```
# ls -l /dev/nvme*
```

To get a snapshot of the system including the Intel RSTe version, RAID capabilities and several other options:

```
# mdadm --detail-platform
```

To get detail of RAID volume including the status of a volume synchronization (build/rebuild):

```
# mdadm -D /dev/md0
```

Or

```
# cat /proc/mdstat
```

To watch the progress of an array during resync (build/rebuild):

```
# while true; do mdadm -D /dev/md0 | grep –i resync; sleep 5; clear; done
```

2.4.2 Additional mdadm Configuration Options

To add a spare drive to a container:

```
# mdadm -a /dev/md/imsm0 /dev/nvme0n<X>
```

To print out verbose details of a RAID member drive:

```
# mdadm -E /dev/nvme0n<X>
```

To stop a container:

```
# mdadm -S /dev/md0
```

To stop and delete a RAID array:

```
# mdadm --stop /dev/md0
```

To scan for and stop ALL active Intel RSTe for NVMe RAID arrays:

```
# mdadm -S /dev/md*
```

To fail a drive from an array:

```
# mdadm --fail /dev/md0 /dev/nvme0n<X>
```

To erase a drive's metadata:

```
# mdadm --zero-superblock /dev/nvme0n<X>
```

2.5 Performance

There are many different ways to test performance of NVMe RAID arrays for the Linux Operating system. In this section, we will demonstrate how to run a performance test using the Linux fio application. Fio is a synthetic I/O measuring tool used for benchmarking and stressing the storage hardware in a system and is available for download from many sources. When testing NVMe SSDs with synthetic benchmarking tools, it is important to pre-condition the RAID array by filling it with large sequential writes to reach steady state performance. This can easily be done with the native Linux dd application:

```
# dd of=/dev/md0 if=/dev/urandom oflag=direct
```

To create a 4 NVMe SSD RAID 5 and test its performance, use the Fio script below:

```
# mdadm –C /dev/md/imsm0 /dev/nvme[0-3]n1 -n 4 -e imsm
# mdadm –C /dev/md0 /dev/md/imsm0 -n 4 -l 5
```

(Alternative without imsm container: # mdadm –C /dev/md0 /dev/nvme[0-3]n1 -n 4 -l 5)

The following commands allow you to write a file system on the array, mount it then run fio for performance testing. The fio command below runs for 10 minutes, with 60 seconds of ramp time, using 16 threads and a 50GB file. For this example, the system memory is only 32GB, so we make our file 50GB to ensure memory saturation.

If this does not match your configuration, adjust the file size as necessary so that it is approximately twice the size of the system memory.

```
# mkfs.ext4 /dev/md0
# mkdir /mnt/raid
# mount /dev/md0 /mnt/raid
# fio --name=jobname1 --nrfiles=1 --rwmixwrite=0 --rw=randrw --runtime=600 --ramp_time=60 --
numjobs=16 --iodepth=1 --size=50G --filesize=50G --direct=1  --thread --ioengine=libaio --
time_based --eta=never --norandommap --group_reporting --overwrite=0 --randrepeat=0 --
end_fsync=0 --filename=/mnt/raid/test --blocksize=4k
```

For a complete explanation of all the options below and its output, consult the fio man page.

Appendix

Revision History

Revision Number	Description	Revision Date
001	Initial release	January 2016

Terms and Acronyms

Term	Definition
AHCI	Advanced Host Controller Interface
API	Application Programming Interface
ATA	Advanced Technology Attachment
BIOS	Basic Input/Output System
Fio	Flexible I/O Tester
GB	Gigabyte
HDD	Hard Disk Drive
KB	Kilobytes
I/O	Input/Output
IMSM	Intel Matrix Storage Manager
IOPS	Input/Output Operations Per Second
MB	Megabytes
Member	A block storage device within a RAID array.
MDADM	Multiple Disk Admin
NCQ	Native Command Queuing
OS	Operating System
POST	Power-On Self-Test
NVMe	Non-Volatile Memory express
RAID	Redundant Array of Independent Disks
RSTe	Rapid Storage Technology enterprise
SATA	Serial Advanced Technology Attachment
SCSI	Small Computer System Interface
SAS	Serial attached SCSI
SSD	Solid State Drive
TB	Terabyte