Booting your OS across NVMe® over Fabrics

NVMe Boot Specification + Boot over NVMe/TCP Reference Implementation

Curtis Ballard, Distinguished Technologist, HPE
Charles Rose, Senior Principal Engineer, Dell
Agenda

• NVM Express® (NVMe®) Boot Specification Overview
• Standardizing Booting from NVMe and NVMe-oF™ Namespaces
• Ecosystem Cooperation: UEFI and DMTF
• Configuring NVMe-oF Boot (UEFI-Based Example)
• Reference Implementations & Future Enhancements
• Q&A
NVM Express, Inc. Overview

- NVM Express is 110+ members strong and was created to expose the benefits of non-volatile memory in all types of computing environments.
- NVMe technology delivers high bandwidth, low latency storage and overcomes bottlenecks.
- NVM Express technology includes the below specifications:
  - NVM Express® (NVMe®) Base Specification
  - NVMe Express Boot Specification
  - NVM Express Command Set Specifications
  - NVM Express Transport Specifications
  - NVMe Management Interface (NVMe-MI™)
- Markets enhanced by NVM Express technology include:
  - Artificial Intelligence
  - Composable Infrastructure
  - Machine Learning
  - Cloud/Data Center
  - SSD Controllers
  - Storage
  - PC/Mobile/IoT
  - Healthcare
NVMe 2.0 Family of Specifications

*NVMe 2.0 specifications were released on June 3, 2021 - Refer to nvmexpress.org/developers
NVMe Boot Task Group

- Membership: 41 companies
  - AMD
  - Avery Design Systems
  - Beijing MemBlaze Technology
  - Broadcom
  - DapuStor
  - Dell Technologies*
  - FADU
  - Fred Knight
  - Hewlett Packard Enterprise
  - Huawei Technologies
  - IBM
  - InnoGrit
  - Inspur Electronic Information Industry
  - Intel*
  - JetIO Technology
  - Kioxia
  - Lenovo
  - LightBits Labs
  - Marvell
  - Microchip
  - Micron Technology
  - Microsoft
  - NVIDIA*
  - Oracle America
  - Phison Electronics
  - Pliops
  - Qualcomm
  - Samsung
  - ScaleFlux
  - Seagate Technology
  - Shenzhen Unionmemory Information System
  - Silicon Motion
  - Solidigm
  - SUSE
  - Swissbit
  - Teledyne LeCroy
  - ULINK Technology
  - University of New Hampshire
  - VMWare
  - Western Digital
  - Yangtze Memory Technologies

*NVMe Boot Task Group Co-Chair
Why Does NVMe Technology Need a Boot Specification

Currently successful storage networking technologies such as Fibre Channel and iSCSI have standardized solutions that allow attached computer systems to boot from OS images stored on storage nodes.

The lack of a standardized capability in NVMe-oF™ presented a barrier for adoption.

This was a missing requirement for a networked storage technology.
Leveraging Existing Remote Storage Boot Over Ethernet

NVMe/TCP boot enabled standardization to leverage past iSCSI lessons and ecosystem enablement.

iSCSI enabled boot and OS handover through a mechanism called the “iSCSI Boot Firmware Table” (iBFT).

iBFT contains information to be shared between BIOS / pre-boot environments and the OS.
Standardize Booting from NVMe and NVMe-oF™ Namespaces

NVMe/TCP boot enabled standardization to leverage past iSCSI lessons and ecosystem enablement.

iSCSI enabled boot and OS handover through a mechanism called the “iSCSI Boot Firmware Table” (iBFT).

iBFT contains information to be shared between BIOS / pre-boot environments and the OS.

Boot from NVMe/TCP main concepts (boot flow and handover mechanism) are similar to booting from iSCSI.

Delta between booting from iSCSI and booting from NVMe-oF™
Standardize Booting from NVMe and NVMe-oF™ Namespaces

- NVMe/TCP boot enabled standardization to leverage past iSCSI lessons and ecosystem enablement
- iSCSI enabled boot and OS handover through a mechanism called the “iSCSI Boot Firmware Table” (iBFT)
- iBFT contains information to be shared between BIOS / pre-boot environments and the OS
- Boot from NVMe/TCP main concepts (boot flow and handover mechanism) are similar to booting from iSCSI
- NVMe needs a similar configuration mechanism, NBFT (NVMe Boot Firmware Table)

Delta between booting from iSCSI and booting from NVMe-oF

![Diagram showing the delta between iSCSI and NVMe-oF booting process]
Standardize Booting from NVMe and NVMe-oF™ Namespaces

NVMe Boot Specification

- Published on NVMe.org* 11/2022
- Defines constructs & guidelines for booting from NVM Express® interfaces over supported transports
- Version 1.0 defines extensions to the NVMe interface for booting over NVMe/TCP transport
  - Normative content describes
    - General concepts for NVMe/NVMe-oF boot
    - Mechanism for boot device enumeration and configuration handoff from Pre-OS to OS environments (ACPI tables)
  - Informative content Introduces
    - Boot stages and flow in a UEFI pre-OS environment
    - Implementation and adoption guidelines and best-practices
      - NVMe-oF boot configuration in the Pre-boot environment
      - Mechanics for consumption of ACPI tables by the OS
      - OS and fabric transport specifics

*https://nvmexpress.org/specifications/

I/O Command Set Specifications (e.g., NVM, Key Value, Zoned Namespace)
NVMe Base Specification
Transport Specifications (e.g., PCIe®, RDMA, TCP)
Boot Specification
Ecosystem Cooperation to Enable Standardization

- Collaboration with the following ecosystem and industry partners was key
  1. UEFI Forum:
     - ACPI Specification (6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
     - UEFI System Specification (2.10*): Adds device path extension for NVMe-oF™ boot
  2. DMTF: Adds standardization for Redfish NVMe-oF ‘secrets registry’ in the 2021.4 release
  3. NVMe Boot Spec 1.0 – introduces standardization of booting over NVMe and NVMe-oF (starting with Booting over NVMe-TCP)
  4. Public reference implementation: The code for booting over NVMe-oF is based on open-source frameworks.

*See references slide for publication locations
Ecosystem Cooperation to Enable Standardization

- Collaboration with the following ecosystem and industry partners was key:
  
1. UEFI Forum:
   - ACPI Specification (6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
   - UEFI System Specification (2.10*): Adds device path extension for NVMe-oF™ boot

2. DMTF: Adds standardization for Redfish NVMe-oF ‘secrets registry’ in the 2021.4 release

3. NVMe Boot Spec 1.0 – introduces standardization of booting over NVMe and NVMe-oF (starting with Booting over NVMe-TCP)

4. Public reference implementation: The code for booting over NVMe-oF is based on open-source frameworks.

*See references slide for publication locations
UEFI Collaboration

- Added to the ACPI XSDT Signature Table
- NVMe over Fabrics Device Path extension to support for NVMe-oF™ boot from UEFI System Spec

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Byte Offset</th>
<th>Byte Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>00</td>
<td>1</td>
<td>Type 3 – Messaging Device Path</td>
</tr>
<tr>
<td>Sub-Type</td>
<td>01</td>
<td>1</td>
<td>Sub-Type 34 - NVMe-oF Namespace Device Path</td>
</tr>
<tr>
<td>Length</td>
<td>02</td>
<td>2</td>
<td>Length of this Structure in bytes. Length is 20+n bytes where n is the length of the SubsystemQQN</td>
</tr>
<tr>
<td>NIDT</td>
<td>04</td>
<td>1</td>
<td>Namespace Identifier Type (NIDT), for globally unique type values defined in the CNS 03h NIDT field (1h, 2h, or 3h) by the NVM Express® Base Specification®.</td>
</tr>
<tr>
<td>NID</td>
<td>05</td>
<td>16</td>
<td>Namespace Identifier (NID), a globally unique value defined in the Namespace Identification Descriptor list (CNS 03h) by the NVM Express® Base Specification in big endian format.</td>
</tr>
<tr>
<td>SubsystemQQN</td>
<td>21</td>
<td>n</td>
<td>Unique identifier of an NVM subsystem stored as a null-terminated UTF-8 string of n-bytes in compliance with the NVMe Qualified Name in the NVM Express® Base Specification. Subsystem QQN is used for purposes of identification and authentication. Maximum length of 224 bytes.</td>
</tr>
</tbody>
</table>

*https://uefi.org/specs/ACPI/6.5/05_ACPI_Software_Programming_Model.html
Ecosystem Cooperation to Enable Standardization

Collaboration with the following ecosystem and industry partners was key

1. UEFI Forum:
   - ACPI Specification (ECR into 6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
   - UEFI System Specification (ECR into 2.10*): Adds device path extension for NVMe-oF™ boot

2. DMTF: Adds standardization for Redfish NVMe-oF ‘secrets registry’ in the 2021.4 release

3. NVMe Boot Spec 1.0 – introduces standardization of booting over NVMe and NVMe-oF (starting with Booting over NVMe-TCP)

4. Public reference implementation: The code for booting over NVMe-oF is based on open-source frameworks.

*See references slide for publication locations
# DMTF Collaboration

Adds standardization for NVMe-oF ‘secrets registry’ for RF 2021.4

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Attributes</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyString</td>
<td>string</td>
<td>read-only required on create (null)</td>
<td>The string for the key.</td>
</tr>
<tr>
<td>KeyType</td>
<td>string</td>
<td>read-only required on create (null)</td>
<td>The format of the key. For the possible property values, see KeyType in Property details.</td>
</tr>
<tr>
<td>NVMeoF ()</td>
<td>object</td>
<td>(null)</td>
<td>NVMe-oF specific properties.</td>
</tr>
<tr>
<td>HostKeyId</td>
<td>string</td>
<td>read-write (null)</td>
<td>The identifier of the host key paired with this target key.</td>
</tr>
<tr>
<td>NQN</td>
<td>string</td>
<td>read-only required on create (null)</td>
<td>The NVMe Qualified Name (NQN) of the host or target subsystem associated with this key.</td>
</tr>
<tr>
<td>OEMSecurityProtocolType</td>
<td>string</td>
<td>read-only (null)</td>
<td>The OEM security protocol that this key uses.</td>
</tr>
<tr>
<td>SecureHashAllowList []</td>
<td>array (string)</td>
<td>read-only (null)</td>
<td>The secure hash algorithms allowed with the usage of this key. For the possible property values, see SecureHashAllowList in Property details.</td>
</tr>
<tr>
<td>SecurityProtocolType</td>
<td>string (enum)</td>
<td>read-only (null)</td>
<td>The security protocol that this key uses. For the possible property values, see SecurityProtocolType in Property details.</td>
</tr>
</tbody>
</table>

**KeyType:** The format of the key.

<table>
<thead>
<tr>
<th>string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVMeoF</td>
<td>An NVMe-oF key.</td>
</tr>
</tbody>
</table>

**SecureHashAllowList:** The secure hash algorithms allowed with the usage of this key.

<table>
<thead>
<tr>
<th>string</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA256</td>
<td>SHA-256.</td>
</tr>
<tr>
<td>SHA384</td>
<td>SHA-384.</td>
</tr>
<tr>
<td>SHA512</td>
<td>SHA-512.</td>
</tr>
</tbody>
</table>
Collaboration with the following ecosystem and industry partners was key

1. UEFI Forum:
   - ACPI Specification (ECR into 6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
   - UEFI System Specification (ECR into 2.10*): Adds device path extension for NVMe-oF™ boot

2. DMTF: Adds standardization for Redfish NVMe-oF ‘secrets registry’ in the 2021.4 release

3. NVMe Boot Spec 1.0 – introduces standardization of booting over NVMe and NVMe-oF (starting with Booting over NVMe-TCP)

4. Public reference implementation: The code for booting over NVMe-oF is based on open-source frameworks.

*See references slide for publication locations
Information presented to the OS using ACPI XSDT Table at OS boot provides

- local Pre-OS -> OS agnostic configuration communications medium; independent from UEFI, UBOOT, …

- standardized means of passing configuration & connection context from pre-OS Boot environment to an administratively configured OS runtime

### NBFT: Pre-OS to OS Configuration Handoff Mechanism

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>An ACPI structure header with some additional NBFT specific info.</td>
</tr>
<tr>
<td>Control Descriptor</td>
<td>Indicates the location of host, HFI, SSNS, security, and discovery descriptors.</td>
</tr>
<tr>
<td>Host Descriptor</td>
<td>Host information.</td>
</tr>
<tr>
<td>HFI Descriptor</td>
<td>An indexable table of HFI Descriptors, one for each fabric interface on the host.</td>
</tr>
<tr>
<td>Subsystem Namespace Descriptor</td>
<td>An indexable table of SSNS Descriptors.</td>
</tr>
<tr>
<td>Discovery Descriptor</td>
<td>An indexable table of Discovery Ddescriptors.</td>
</tr>
<tr>
<td>HFI Transport Descriptor</td>
<td>Indicated by an HFI Descriptor, corresponds to a specific transport for a single HFI.</td>
</tr>
<tr>
<td>SSNS Extended Info Descriptor</td>
<td>Indicated by an SSNS Descriptor if needed.</td>
</tr>
</tbody>
</table>

NBFT: Pre-OS to OS Configuration Handoff Mechanism
https://nvmexpress.org/specifications/
Public Reference Implementation Based on UEFI

Reference code* for booting over NVMe-oF is based

- on the NVMe Boot Spec 1.0
- on open-source frameworks
  - Developed by a subset of NVM Express member companies including:
    - Dell Technologies
    - NVIDIA
    - Intel
    - SUSE
    - Red Hat
    - Hewlett Packard Enterprise
    - VMware
  - Released* under BSD-3-Clause (or other open-source license as required by components)

*https://github.com/timberland-sig
The seven phases in a UEFI boot sequence*

1. Security (SEC)
2. Pre-EFI Initialization (PEI)
3. Drive Execution Environment (DXE)
4. Boot Device Selection (BDS)
5. Transient System Load (TSL)
6. Runtime (RT)
7. After Life (AL)

*Tianocore: EDK2 Build Specification
Boot Attempt configuration is stored in UEFI variables.

Administrator configures Pre-OS driver:

- target subsystem NQN
- target IP address
- target port #
- target namespace
- host NQN
- security related info
Driver Execution Environment phase: DXE driver supporting NVMe-oF boot is loaded and executed:

- reads configuration from UEFI variables
- sets up network (interfaces, routing, …)
- (optionally) retrieves authentication credentials
- (optionally) performs discovery and authentication
- connects to NVMe subsystems provides namespaces to the UEFI Boot Manager as block devices
- stores the configuration in the NBFT: can later be accessed by the OS as an ACPI table
Configuring NVMe-oF Boot (UEFI-based example): Pre-Operating System Boot

**Boot Device Selection phase:** The Namespace can then be selected as final boot device for OS boot.
Configuring NVMe-oF Boot (UEFI-based example): Pre-Operating System Boot

**Transient System Load phase:**
- OS image loaded from boot device
- UEFI hands over execution to OS specific boot loader
- OS Boot Loader continues the OS boot

At this point, the NBFT has been generated, stored in main memory, and can be accessed by the OS as an ACPI table.
Configuring NVMe-oF Boot (UEFI-based example): OS Transition to Runtime

Runtime phase:
- read the configuration from the NBFT
- set up the network (interfaces, routing, …)
- (optionally) retrieve authentication credentials
- (optionally) perform discovery and authentication
- connect to NVMe subsystems
- provide namespaces to other parts of the OS
Configuring NVMe-oF Boot (UEFI-based example): Typical OS Handover and Initialization

- Normal operating system boot:
  - To persist info to restore NVMe-oF connections, OS may either:
    - continue using the NBFT
    - Use OS specific mechanism

- Operating system installation:
  - A user may either:
    - use the NBFT provided host NQN as its own host NQN
    - set a separate host NQN (if NVMe-oF subsystem supports multiple host NQNs)
Reference Implementation of Booting over NVMe/TCP

Pre-OS time of boot:

- EDK2 NVMe-oF™ UEFI Driver for the NVMe®/TCP transport
  - ACPI NBFT will be produced by this UEFI implementation prior to OS boot

OS Boot and Runtime:

- Linux® reference implementation that:
  - Exposes the NBFT to the user-space
  - Consumes the NBFT contents to connect to configured namespaces
  - Enables common tools (e.g., dracut, nvme-cli) to use the NBFT
Configuring NVMe-oF Boot (UEFI-based example): Pre-Operating System Boot

EDK2 Concept Architecture

EDK2 Reference Architecture as implemented
(Not yet published upstream for review)
Pre-Boot Environment Configuration Tool

- **nvmeofcli for EFI:**
  - Command Line tool to facilitate basic diagnostics and interoperability with pre-OS reference driver
  - `nvmeofcli list` command
  - `nvmeofcli connect` command
Pre-Boot Environment HII
OS Handoff Enablement in Reference Design

- Linux Kernel support for ACPI “NBFT” Table
- User-Space Device Connection and Configuration tools consuming Linux sysfs
- initrd/dracut changes to support NVMe/TCP:
  - Detects NBFT presence
  - connects pertinent networking
  - uses nvme-cli to connect to NVMe Subsystems/Namespaces

Nvme-cli – two new subcommands:
- nvme show-nbft for dumping NBFT content
  - free text / table format
  - JSON format
- nvme connect-nbft
  - connect to subsystems and namespaces listed in or discovered through the NBFT
  - Everything except network setup

Graphic credit Joey Lee, SUSE
nvme-cli – New subcommands: ‘nvme nbft show’ free-text format

```
leapnvmetcp:~ # nvme nbft show --help

Display contents of the ACPI NBFT files.

[ --output-format=<FMT>, -o <FMT> ] --- Output format: normal|json
[ --subsystem, -s ] --- show NBFT subsystems
[ --hfi, -H ] --- show NBFT HFIs
[ --discovery, -d ] --- show NBFT discovery controllers
[ --nbft-path=<STR> ] --- user-defined path for NBFT tables

NBFT Subsystems:

<table>
<thead>
<tr>
<th>Idx</th>
<th>QNQ</th>
<th>Trsp</th>
<th>Address</th>
<th>SvcId</th>
<th>HFI(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>ngn.1988-11.com.dell:powerstore:00:2a64a6b1c5b81f6c4549</td>
<td>tcp</td>
<td>100.71.103.49</td>
<td>4420</td>
<td>1</td>
</tr>
</tbody>
</table>

NBFT HFIs:

<table>
<thead>
<tr>
<th>Idx</th>
<th>Trsp</th>
<th>PCI Addr</th>
<th>MAC Addr</th>
<th>DHCP</th>
<th>IP Addr</th>
<th>Mask</th>
<th>Gateway</th>
<th>DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tcp</td>
<td>0:1:0.0</td>
<td>ec:2a:72:33:06:cc</td>
<td>yes</td>
<td>100.65.144.67</td>
<td>24</td>
<td>100.68.144.254/100.64.0.111</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>tcp</td>
<td>0:1:0.1</td>
<td>ec:2a:72:33:06:cd</td>
<td>yes</td>
<td>100.65.144.137</td>
<td>24</td>
<td>100.68.144.254/100.64.0.111</td>
<td></td>
</tr>
</tbody>
</table>

NBFT Discovery Controllers:

<table>
<thead>
<tr>
<th>Idx</th>
<th>URI</th>
<th>QNQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>nvme+tcp://100.71.103.50:8009/ngn.2014-08.org.nvmeexpress.discovery</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>nvme+tcp://100.71.103.50:8009/ngn.2014-08.org.nvmeexpress.discovery</td>
<td></td>
</tr>
</tbody>
</table>
```
nvme-cli – New subcommands: “nvme nbft show” JSON format

```
[root@localhost nvme-cli]# .build/nvme show-nbft -o json -H -d -o -P /home/nbft_0.65.7jul
{
  "filename": "/home/nbft_0.65.7jul/NBFT",
  "host": {
    "id": "6e454c4e-3400-1036-8038-82e04f313233",
    "host_id_configured": 0,
    "host_qn_configured": 0,
    "primary_admin_host_flag": "not indicated"
  },
  "subsystem": {
    "index": 1,
    "num_hfis": 1,
    "hfi": {
      1
    },
    "transport": "tcp",
    "transport_address": "100.71.103.48",
    "transport_svcid": "4420",
    "subsys_nqn": "nqn.1988-11.com.dell:powerstore:00:2a64abfc5b81f6c4549",
    "controller_id": 0,
    "asqsz": 0,
    "pdu_header_digest_required": 0,
    "data_digest_required": 0
  }
}
```

```
"subsys_nqn": "nqn.1988-11.com.dell:powerstore:00:2a64abfc5b81f6c4549",
"controller_id": 0,
"asqsz": 0,
"pdu_header_digest_required": 0,
"data_digest_required": 0
}
```

```
},
"hfi": {
  "index": 1,
  "transport": "tcp",
  "pcidiv": "0:40:0.0",
  "mac_addr": "a0:02:26:28:7e:7c:0e",
  "vlan": 0,
  "ip_origin": 82,
  "ipaddr": "100.71.245.232",
  "subnet_mask_prefix": 24,
  "gateway_ipaddr": "100.71.245.254",
  "route_metric": 500,
  "primary_dns_ipaddr": "100.64.0.5",
  "secondary_dns_ipaddr": "100.64.0.6",
  "dhcp_server_ipaddr": "100.71.245.254",
  "this_hfi_is_default_route": 1,
  "dhcp_override": 1
}
```

```
},
"discovery": {
  "index": 1,
  "hfi": 1,
  "url": "nvme+tcp://100.71.103.50:8009/",
  "nqn": "nqn.2014-08.org.nvmexpress.discovery"
}
```

Reference Implementations of Booting over NVMe/TCP

Proof-of-Concept for NVMe Boot

- QEMU based PoCs are available for both openSUSE Leap 15.5 and Fedora 37
- These examples are useful because the details of early OS bring-up differ between distributions

Prerequisites

- An Intel based host platform running a current version of openSUSE or Fedora
- A connection to the internet and a root privileged account to administer QEMU

Setup is simple – setup the Host/Hypervisor system then follow the instructions in the PoCs and the scripts will configure and install the software to run the QEMU based POC automatically.

openSUSE and Fedora PoCs are available at: https://github.com/timberland-sig/
Future Enhancements: Open Source and Ecosystem

- Support for Authentication/TLS
- Support for DMTF Redfish Secrets
- Additional OS and installer support
Future Enhancements: NVMe Boot Specification

- Investigate Booting over Additional Transports
- Big Namespace Qty Management in Large Fleets
- Multi-Path Topology Examples
- Support Device Tree
- Setting NVMe-oF Boot Entries in OS

Contributions are welcome!
Join the NVMe Consortium and the NVMe Boot Task Group
https://nvmexpress.org/join-nvme/
Adding new Transport Support to NVMe Boot Specification

Header for new HFI Transport Info Descriptor in NBFT

Bytes 00 – 05: Mandatory to describe the Header structure for a new Transport Info Descriptor type

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Structure ID</td>
</tr>
<tr>
<td>01</td>
<td>Version</td>
</tr>
<tr>
<td>02</td>
<td>HFI Transport Type.</td>
</tr>
<tr>
<td>03</td>
<td>Transport Info Version</td>
</tr>
<tr>
<td>05:04</td>
<td>HFI Descriptor Index</td>
</tr>
</tbody>
</table>

Thereafter Transport-specific descriptor flags as needed following Figure 13 in the NVMe Boot Spec

```
"hfi":{
  "index":1,
  "transport":"tcp",
  "pcidev":"0:40:0.0",
  "mac_addr":"b0:26:28:e8:7c:0e",
  "vlan":0,
  "ip_origin":82,
  "ipaddr":"100.71.245.232",
  "subnet_mask_prefix":24,
  "gateway_ipaddr":"100.71.245.254",
  "route_metric":500,
  "primary_dns_ipaddr":"100.64.0.5",
  "secondary_dns_ipaddr":"100.64.0.6",
  "dhcp_server_ipaddr":"100.71.245.254",
  "this_hfi_is_default_route":1,
  "dhcp_override":1
}
```

Transports may require a new ECR to the UEFI System Spec if they do not already have a Device Path Messaging Type supporting them.
References and Repositories

- **NVM Express®**: [https://nvmexpress.org/specifications/](https://nvmexpress.org/specifications/)
- **ACPI 6.5**: [https://uefi.org/specs/ACPI/6.5/05_ACPI_Software_Programming_Model.html](https://uefi.org/specs/ACPI/6.5/05_ACPI_Software_Programming_Model.html)
- **Open-Source Software Repos**: [https://github.com/timberland-sig](https://github.com/timberland-sig)
  - Note: Most software has been pushed upstream. For edk2 use the version off of the Timberland SIG github. For all other software use the latest upstream version.
Questions?