Flash Memory Summit

nvm EXPRESS®
Architected for Performance

Booting your OS across NVMe® over Fabrics

NVMe Boot Specification + Boot over NVMe/TCP Reference Implementation

Rob Davis, VP Storage Technology, NVIDIA

Curtis Ballard, Distinguished Technologist, HPE

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

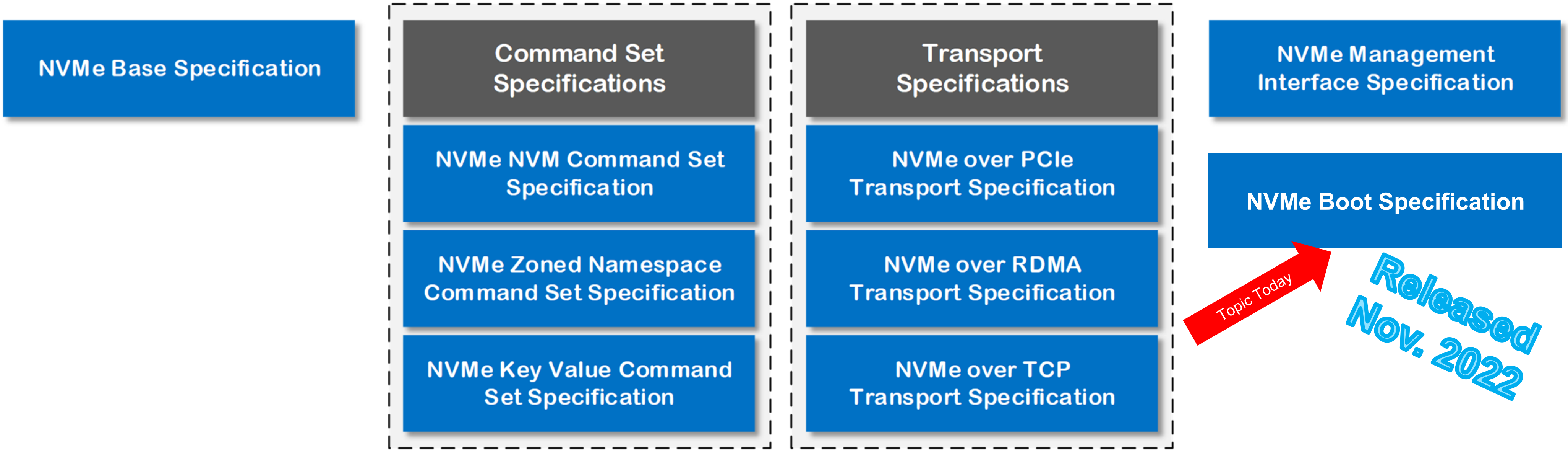
NVM Express, Inc. Overview

- NVM Express is **110+ members** strong and was created to expose the benefits of non-volatile memory in all types of computing environments
- NVM Express® (NVMe®) technology delivers high bandwidth, low latency storage and overcomes bottlenecks
- NVMe technology includes the below specifications:
  - **NVM Express Base Specification**
  - **NVM 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

Promoter Group
2022 - 2023

DELL EMC, Google, Hewlett Packard Enterprise
intel, KIOXIA, MARVELL
Meta, Micron, Microsoft
NetApp, SAMSUNG, SEAGATE
Western Digital

NVMe® 2.0 Family of Specifications



NVMe Base Specification

Command Set Specifications

- NVMe NVM Command Set Specification
- NVMe Zoned Namespace Command Set Specification
- NVMe Key Value Command Set Specification

Transport Specifications

- NVMe over PCIe Transport Specification
- NVMe over RDMA Transport Specification
- NVMe over TCP Transport Specification

NVMe Management Interface Specification

NVMe Boot Specification

Topic Today

Released Nov. 2022

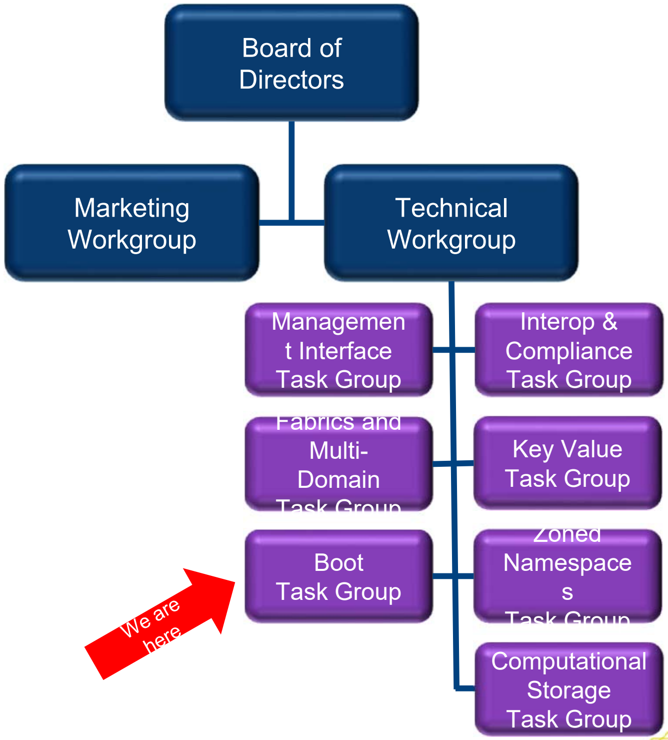
*NVMe 2.0 specifications were released on June 3, 2021 - Refer to nvmexpress.org/developers

NVM Express Boot Task Group

Membership: 34 companies

AMD
Avery Design Systems
Beijing MemBlaze Technology
Broadcom
ByteDance Ltd
Dell Technologies*
FADU
Hewlett Packard Enterprise
Huawei Technologies
Intel*
JetIO Technology
Kioxia
LightBits Labs
Marvell
Micron Technology
Microsoft

NetApp
NVIDIA*
Oracle America
Phison Electronics
Pliops
Samsung
ScaleFlux
Seagate Technology
Silicon Motion
Solidigm
SK Hynix
SUSE
ULINK Technology
University of New Hampshire
VMWare
Western Digital



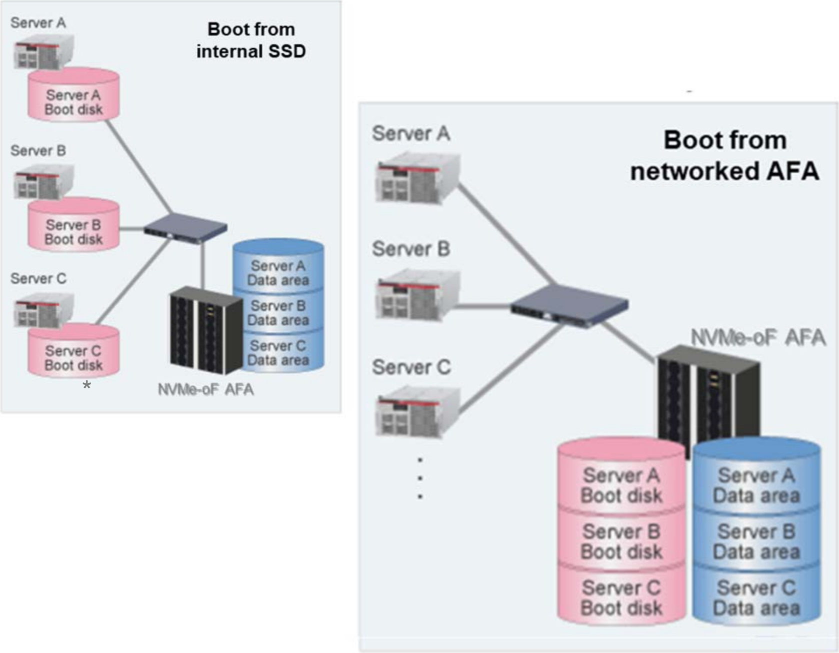
*NVM Express Boot Task Group Co-Chair

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™ technology presented a barrier for adoption.

This was a missing requirement for a networked storage technology.



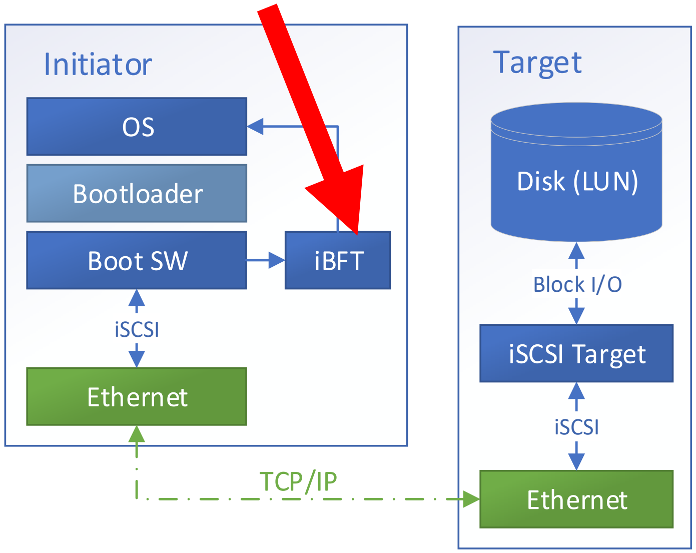
*AFA = All Flash Array Storage System

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



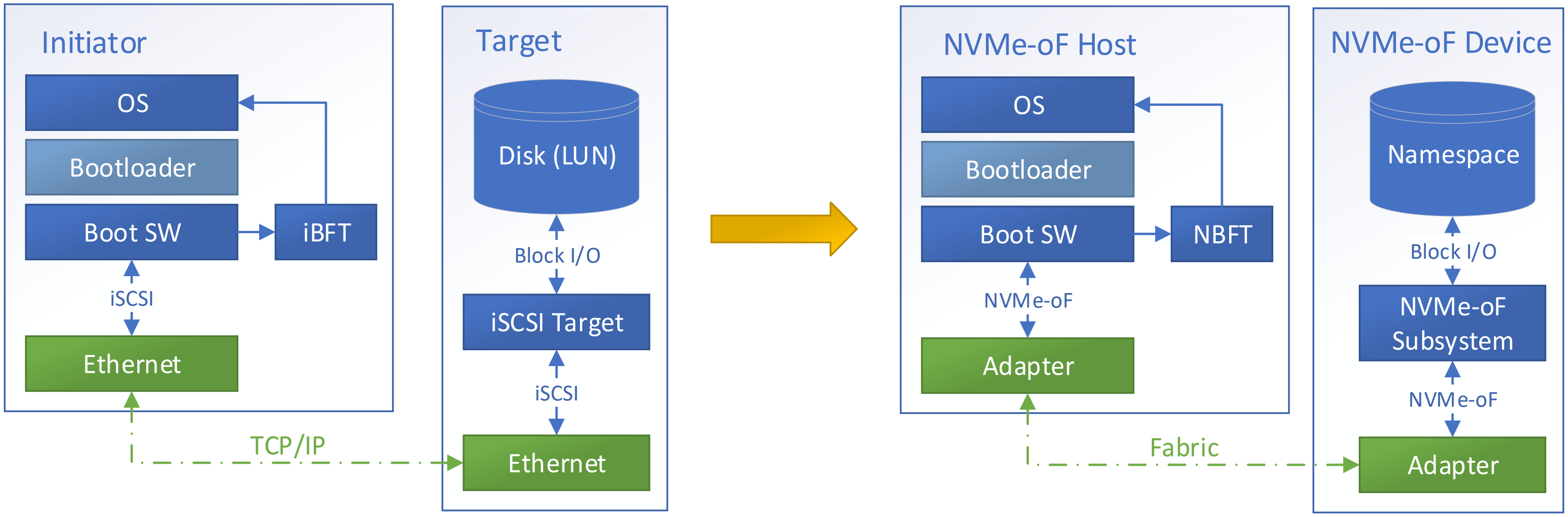
Standardize Booting from NVMe® and NVMe-oF™ Namespaces

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

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



Delta between booting from iSCSI and booting from NVMe-oF transport

Standardize Booting from NVMe® and NVMe-oF™ Namespaces

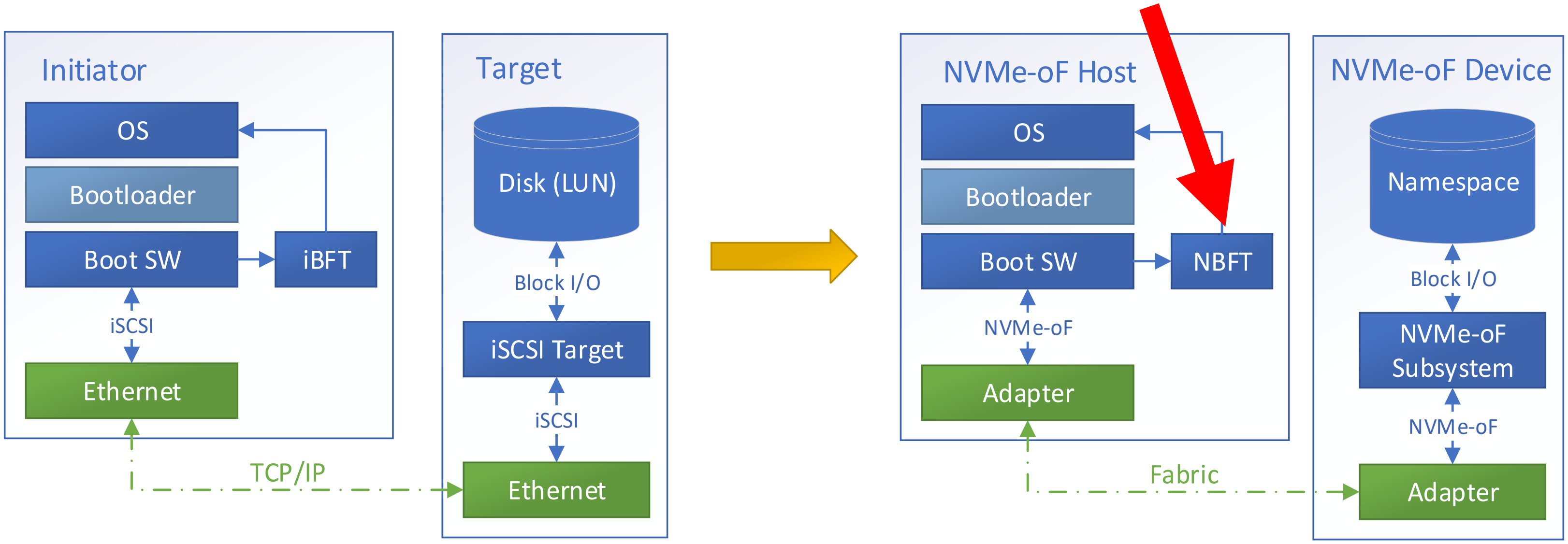
NVMe/TCP boot enabled standardization to leverage past iSCSI lessons and ecosystem enablement

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

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

NVMe needs a similar configuration mechanism, NBFT (NVMe Boot Firmware Table)

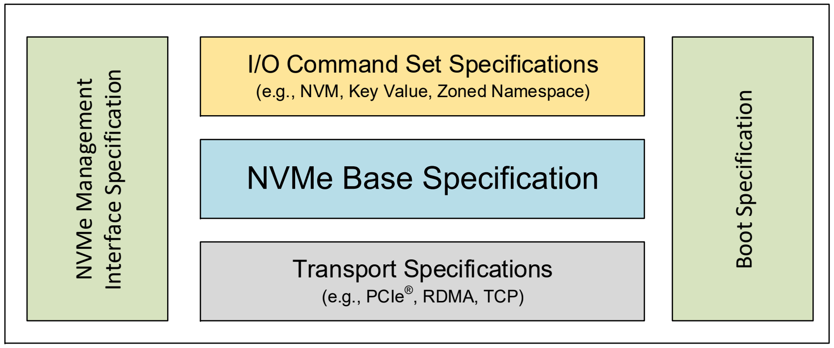


Delta between booting from iSCSI and booting from NVMe-oF transport

Standardize Booting from NVMe® and NVMe-oF™ Namespaces

NVMe Boot Specification

- Published on nvmexpress.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

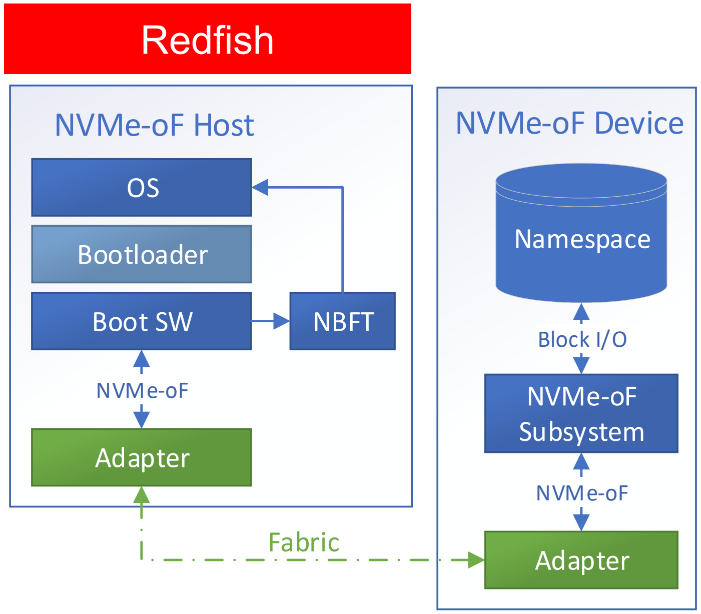

NVMe Management Interface Specification
I/O Command Set Specifications
(e.g., NVM, Key Value, Zoned Namespace)
NVMe Base Specification
Transport Specifications
(e.g., PCIe®, RDMA, TCP)
Boot Specification

*https://nvmexpress.org/specifications/

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 NVMe and NVMe-oF boot (starting with Booting over NVMe/TCP transport)
4. Public reference implementation: The code for NVMe-oF boot is based on open-source frameworks.

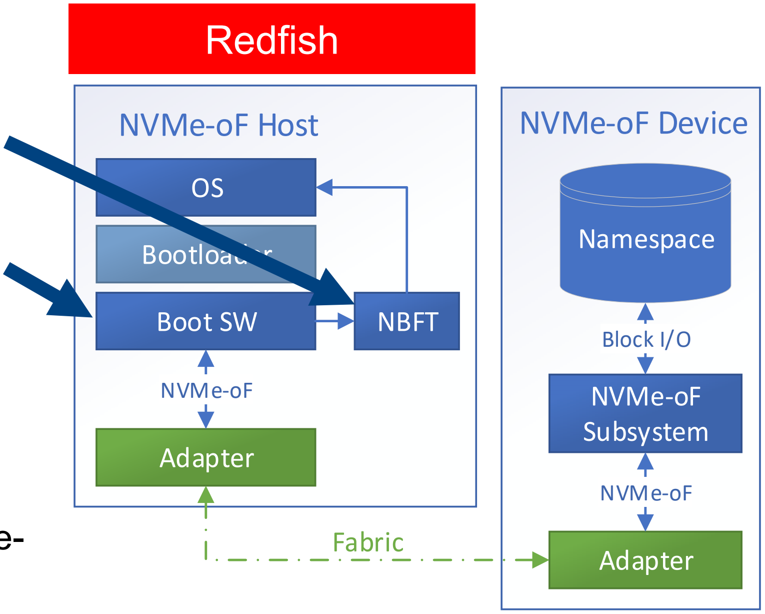


*See references slide for publication locations

Ecosystem Cooperation to Enable Standardization

Collaboration with the following ecosystem and industry partners was key

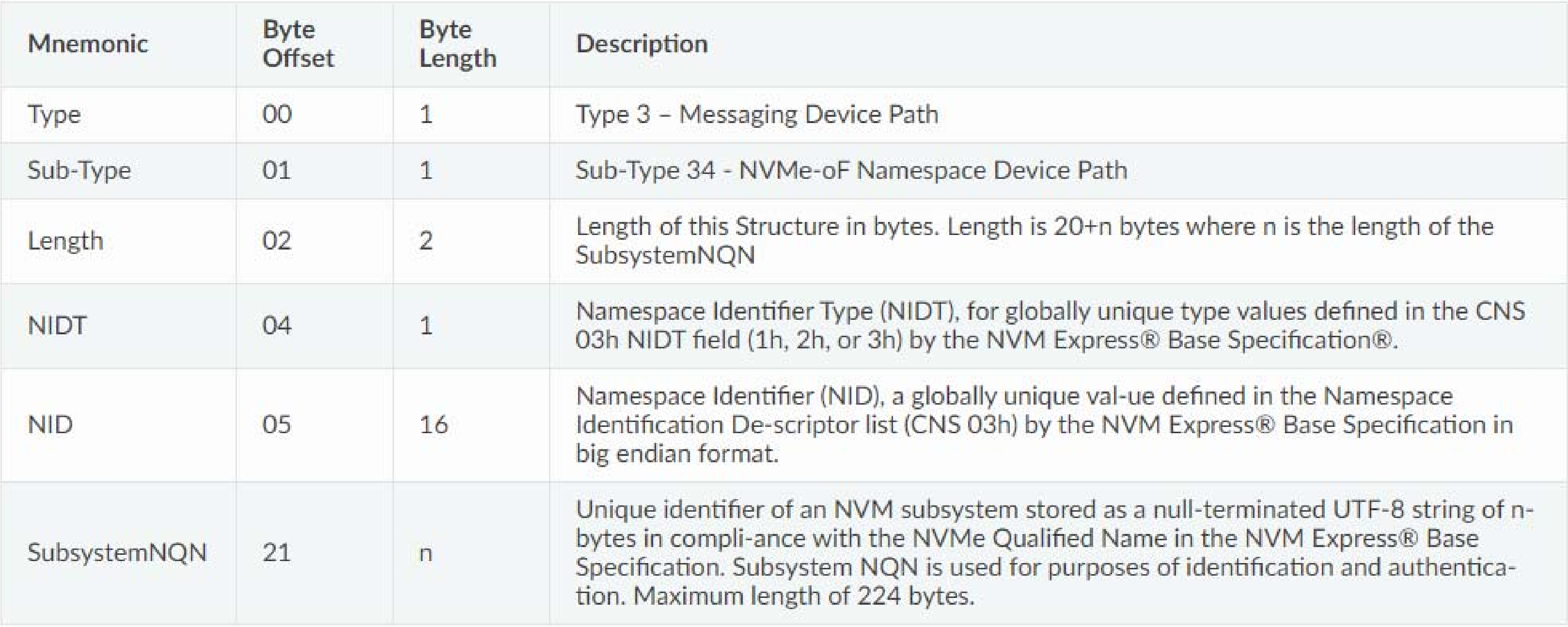
1. UEFI Forum:
   - ACPI Specification (6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
   - UEFI System Specification (2.10*): Adds device path extension for NVMe-oF™ boot
2. DMTF: Adds standardization for Redfish NVMe-oF 'secrets registry' in the 2021.4 release
3. NVMe Boot Spec 1.0 – introduces standardization of NVMe and NVMe-oF boot (starting with Booting over NVMe/TCP transport)
4. Public reference implementation: The code for NVMe-oF boot 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**



Mnemonic	Byte Offset	Byte Length	Description
Type	00	1	Type 3 – Messaging Device Path
Sub-Type	01	1	Sub-Type 34 - NVMe-oF Namespace Device Path
Length	02	2	Length of this Structure in bytes. Length is 20+n bytes where n is the length of the SubsystemNQN
NIDT	04	1	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®.
NID	05	16	Namespace Identifier (NID), a globally unique val-ue defined in the Namespace Identification De-scriptor list (CNS 03h) by the NVM Express® Base Specification in big endian format.
SubsystemNQN	21	n	Unique identifier of an NVM subsystem stored as a null-terminated UTF-8 string of n-bytes in compli-ance with the NVMe Qualified Name in the NVM Express® Base Specification. Subsystem NQN is used for purposes of identification and authentica-tion. Maximum length of 224 bytes.

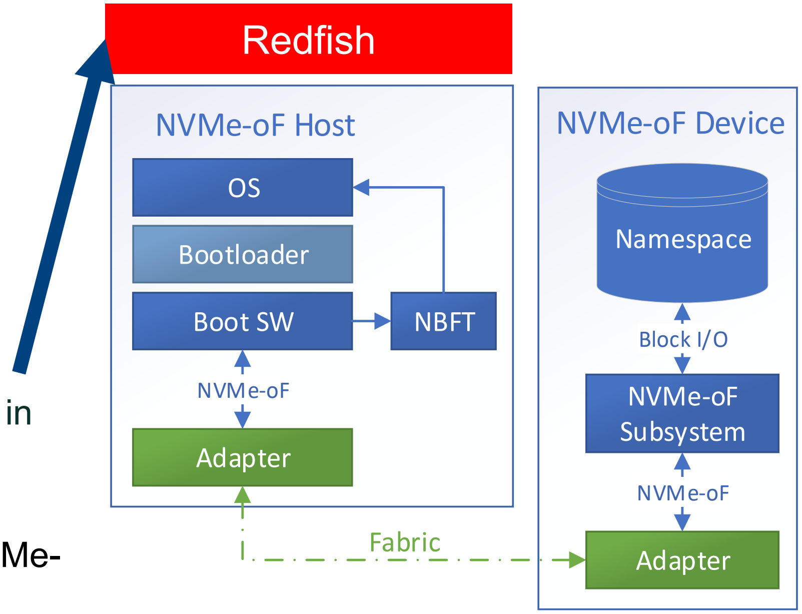
*https://uefi.org/specs/ACPI/6.5/05_ACPI_Software_Programming_Model.html

**https://uefi.org/specs/UEFI/2.10/10_Protocols_Device_Path_Protocol.html#nvme-over-fabric-nvme-of-namespace-device-path

Ecosystem Cooperation to Enable Standardization

Collaboration with the following ecosystem and industry partners was key

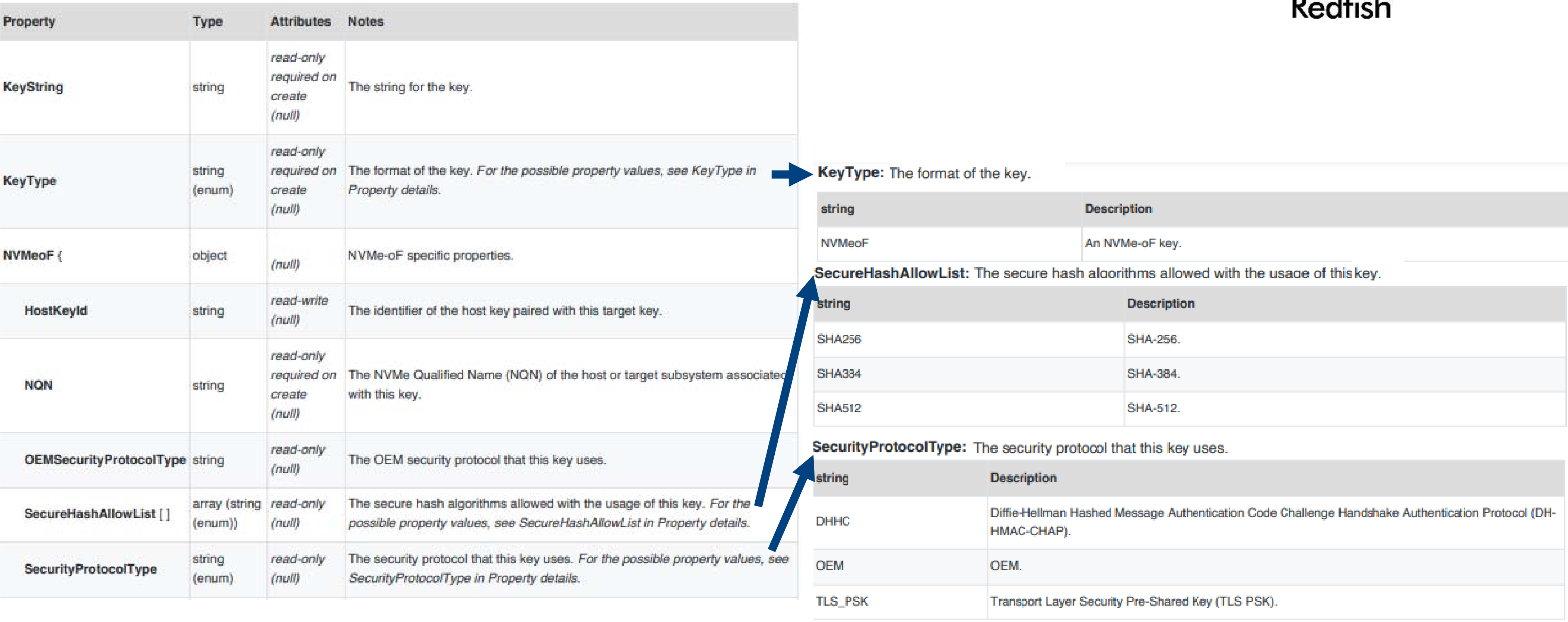
1. UEFI Forum:
   - ACPI Specification (ECR into 6.5*): Adds ACPI NVMe® Boot Firmware Table (NBFT) to ACPI.org
   - UEFI System Specification (ECR into 2.10*): Adds device path extension for NVMe-oF™ boot
2. DMTF: Adds standardization for Redfish NVMe-oF 'secrets registry' in the 2021.4 release
3. NVMe Boot Spec 1.0 – introduces standardization of NVMe and NVMe-oF boot (starting with Booting over NVMe/TCP transport)
4. Public reference implementation: The code for NVMe-oF boot 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



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

KeyType: The format of the key.

string	Description
NVMeoF	An NVMe-oF key.

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

string	Description
SHA256	SHA-256.
SHA384	SHA-384.
SHA512	SHA-512.

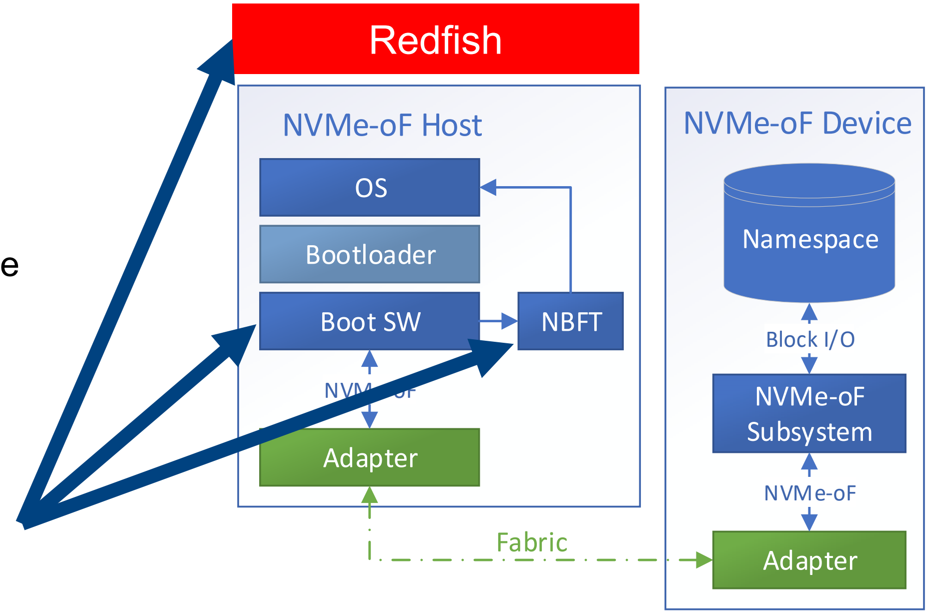
SecurityProtocolType: The security protocol that this key uses.

string	Description
DHHC	Diffie-Hellman Hashed Message Authentication Code Challenge Handshake Authentication Protocol (DH-HMAC-CHAP).
OEM	OEM.
TLS_PSK	Transport Layer Security Pre-Shared Key (TLS PSK).

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 NVMe and NVMe-oF boot (starting with Booting over NVMe/TCP transport)
4. Public reference implementation: The code for NVMe-oF boot is based on open-source frameworks.

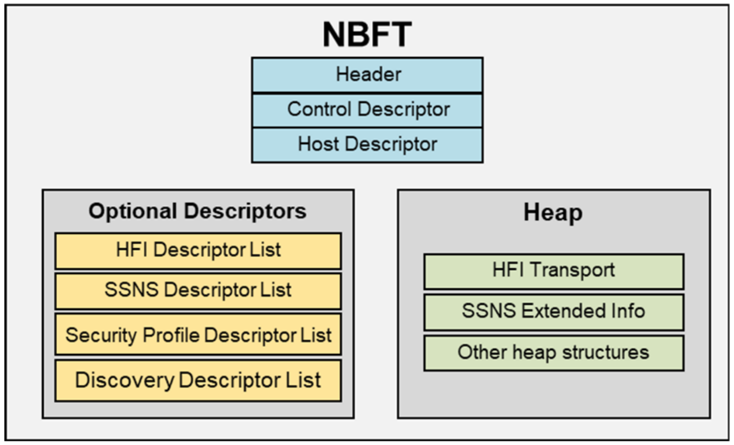


*See references slide for publication locations

NBFT: Pre-OS to OS Configuration Handoff Mechanism

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

Header

Control Descriptor

Host Descriptor

Optional Descriptors

HFI Descriptor List

SSNS Descriptor List

Security Profile Descriptor List

Discovery Descriptor List

Heap

HFI Transport

SSNS Extended Info

Other heap structures

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

https://nvmexpress.org/specifications/

Public Reference Implementation Based on UEFI

Reference code* for NVMe-oF™ boot is based

- on the NVMe® Boot Spec 1.0
- on open-source frameworks

DELL Technologies NVIDIA intel SUSE Red Hat Hewlett Packard Enterprise vmware®

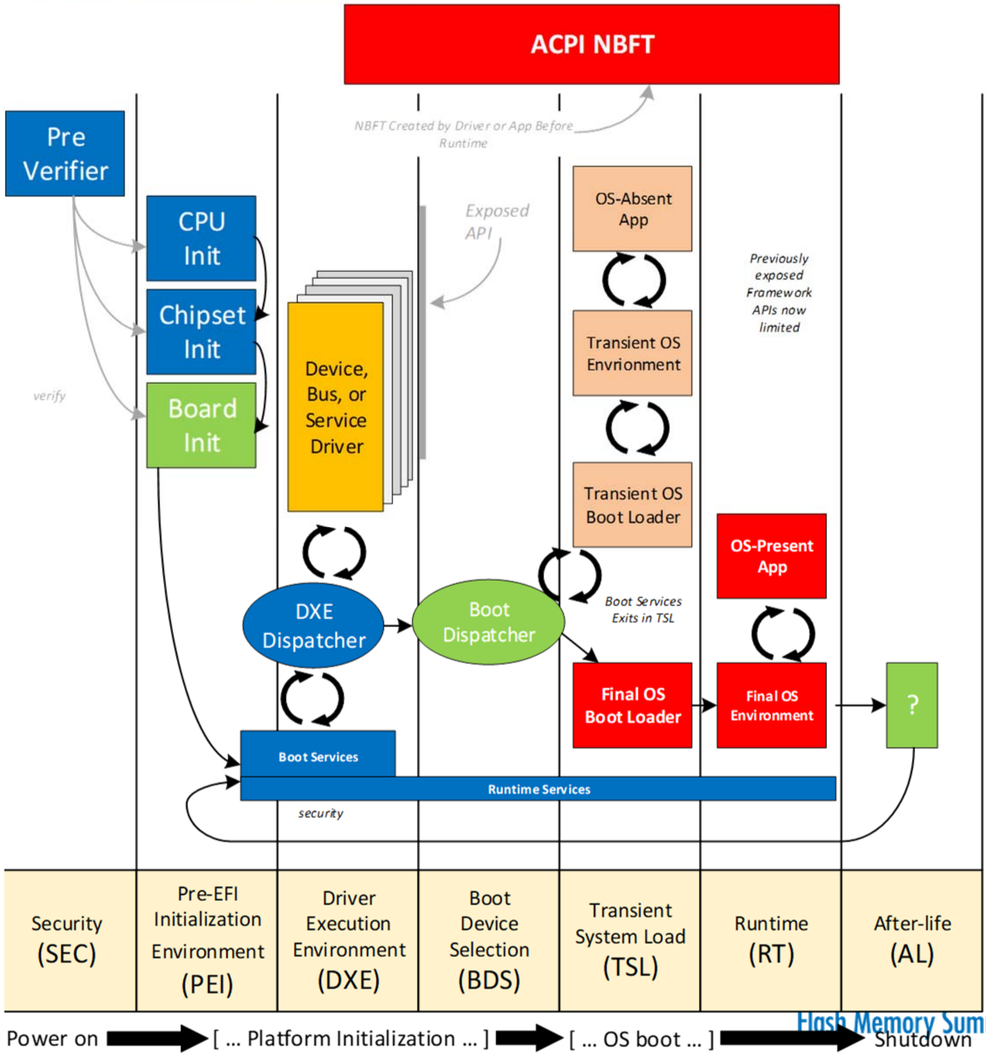
- Developed by a subset of NVM Express member companies including:
- To be released* under BSD-3-Clause (or other open-source license as required by components)

*https://github.com/timberland-sig

UEFI Boot Phases

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)



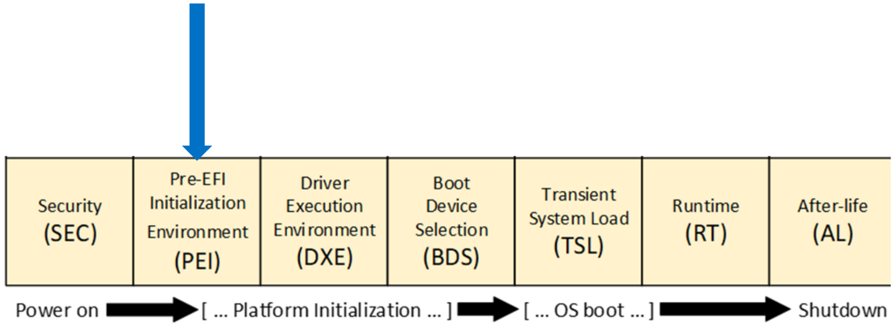
*Tianocore: EDK2 Build Specification

Configuring NVMe-oF™ Boot (UEFI-based example): Pre-Operating System Boot

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

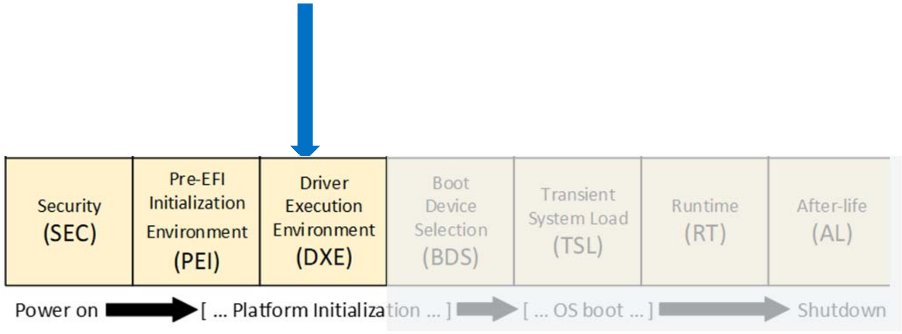

Security (SEC)
Pre-EFI Initialization Environment (PEI)
Driver Execution Environment (DXE)
Boot Device Selection (BDS)
Transient System Load (TSL)
Runtime (RT)
After-life (AL)
Power on
[... Platform Initialization ...]
[... OS boot ...]
Shutdown

Configuring NVMe-oF™ Boot (UEFI-based example): Pre-Operating System Boot

New functionality

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

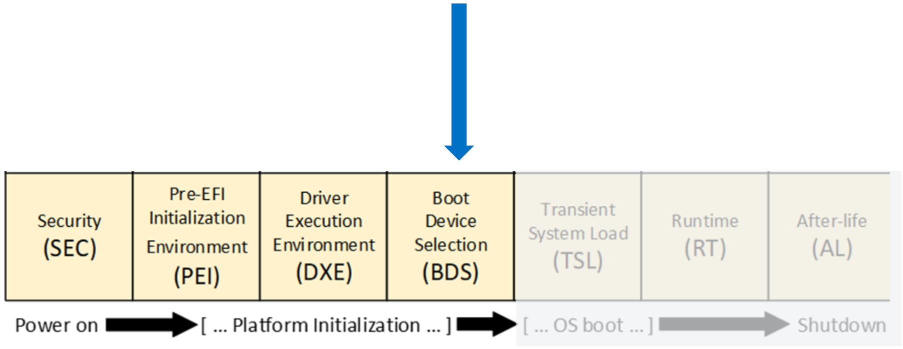


Security (SEC)
Pre-EFI Initialization Environment (PEI)
Driver Execution Environment (DXE)
Boot Device Selection (BDS)
Transient System Load (TSL)
Runtime (RT)
After-life (AL)
Power on
[... Platform Initialization ...]
[... OS boot ...]
Shutdown

Configuring NVMe-oF™ Boot (UEFI-based example): Pre-Operating System Boot

Existing functionality

Boot Device Selection phase: The Namespace can then be selected as final boot device for OS boot



Security (SEC)

Pre-EFI Initialization Environment (PEI)

Driver Execution Environment (DXE)

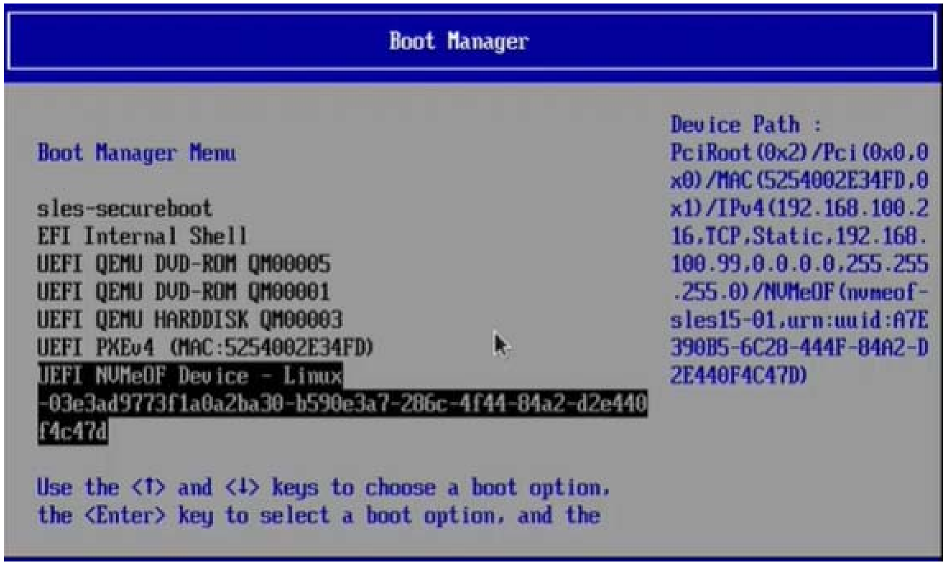
Boot Device Selection (BDS)

Transient System Load (TSL)

Runtime (RT)

After-life (AL)

Power on → [... Platform Initialization ...] → [... OS boot ...] → Shutdown



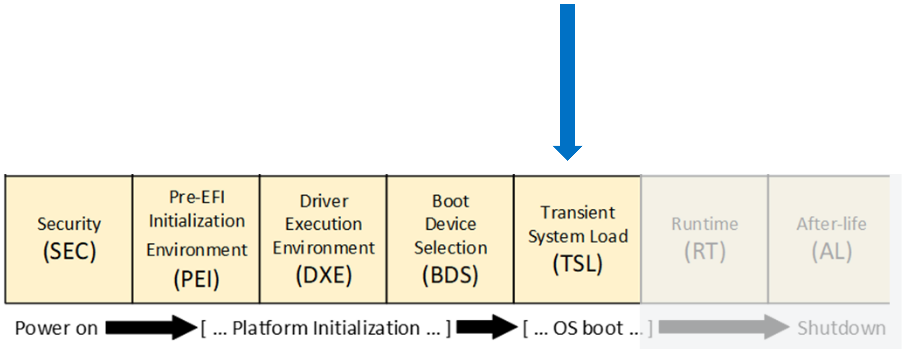
```
                                Boot Manager

                                                  Device Path :
 Boot Manager Menu                                PciRoot(0x2)/Pci(0x0,0
                                                  x0)/MAC(5254002E34FD,0
 sles-secureboot                                  x1)/IPv4(192.168.100.2
 EFI Internal Shell                               16,TCP,Static,192.168.
 UEFI QEMU DVD-ROM QM00005                        100.99,0.0.0.0,255.255
 UEFI QEMU DVD-ROM QM00001                        .255.0)/NVMeOF(nvmeof-
 UEFI QEMU HARDDISK QM00003                       sles15-01,urn:uuid:A7E
 UEFI PXEv4 (MAC:5254002E34FD)                    390B5-6C28-444F-84A2-D
 UEFI NVMeOF Device - Linux                       2E440F4C47D)
 -03e3ad9773f1a0a2ba30-b590e3a7-286c-4f44-84a2-d2e440
 f4c47d

 Use the <↑> and <↓> keys to choose a boot option,
 the <Enter> key to select a boot option, and the
```

Configuring NVMe-oF™ Boot (UEFI-based example): Pre-Operating System Boot

Existing functionality

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

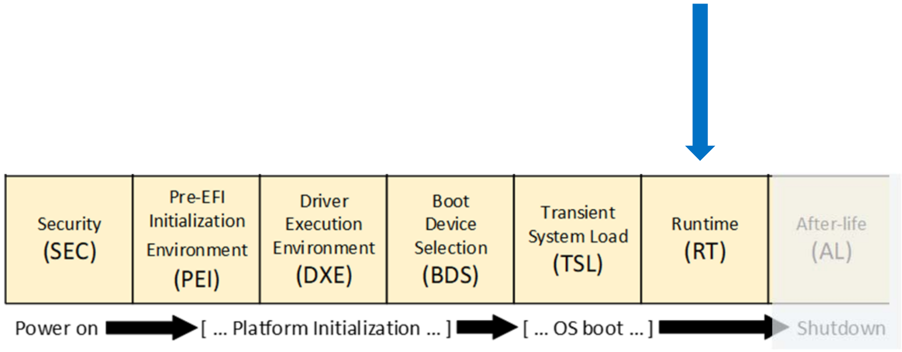

Security (SEC)
Pre-EFI Initialization Environment (PEI)
Driver Execution Environment (DXE)
Boot Device Selection (BDS)
Transient System Load (TSL)
Runtime (RT)
After-life (AL)
Power on
[... Platform Initialization ...]
[... OS boot ...]
Shutdown

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



Security (SEC)

Pre-EFI Initialization Environment (PEI)

Driver Execution Environment (DXE)

Boot Device Selection (BDS)

Transient System Load (TSL)

Runtime (RT)

After-life (AL)

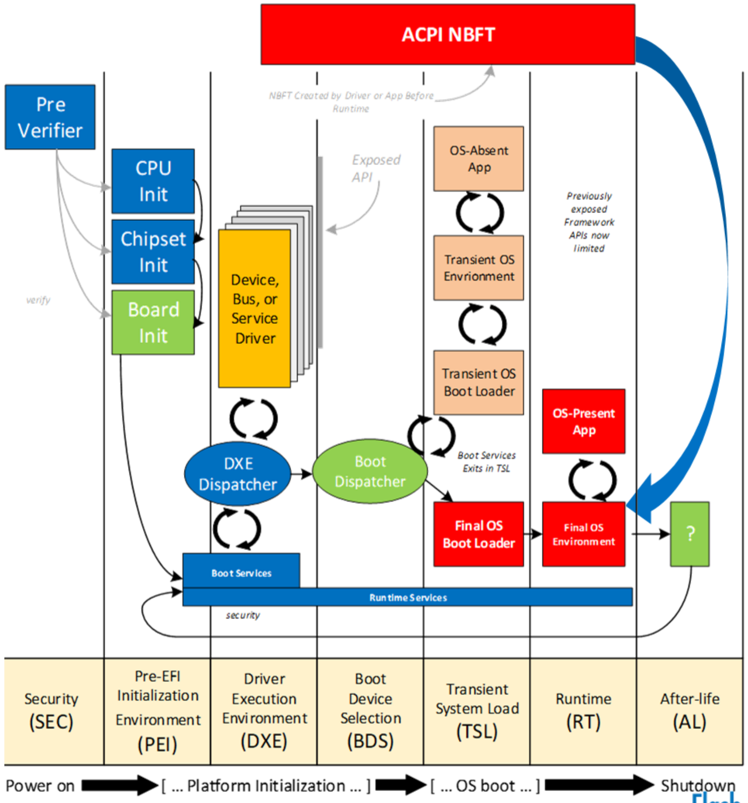
Power on → [… Platform Initialization …] → [… OS boot …] → Shutdown

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:
    a) use the NBFT provided host NQN as its own host NQN

    b) set a separate host NQN (if NVMe-oF subsystem supports multiple host NQNs)



Reference Implementation of Booting over NVMe®/TCP Transport

Pre-OS time of boot:

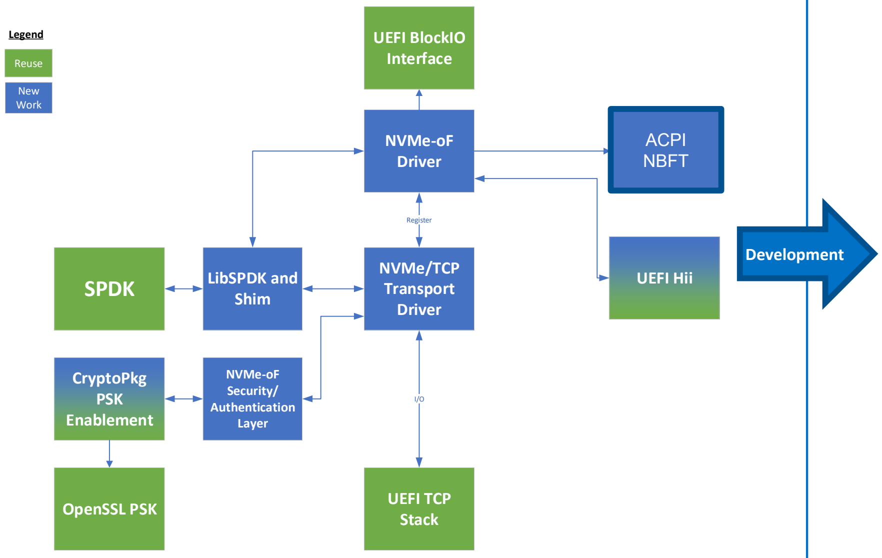
- EDK2 NVMe-oF™ UEFI Driver for the NVMe/TCP transport
  - ACPI NBFT will be produced by this UEFI implementation prior to OS boot

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

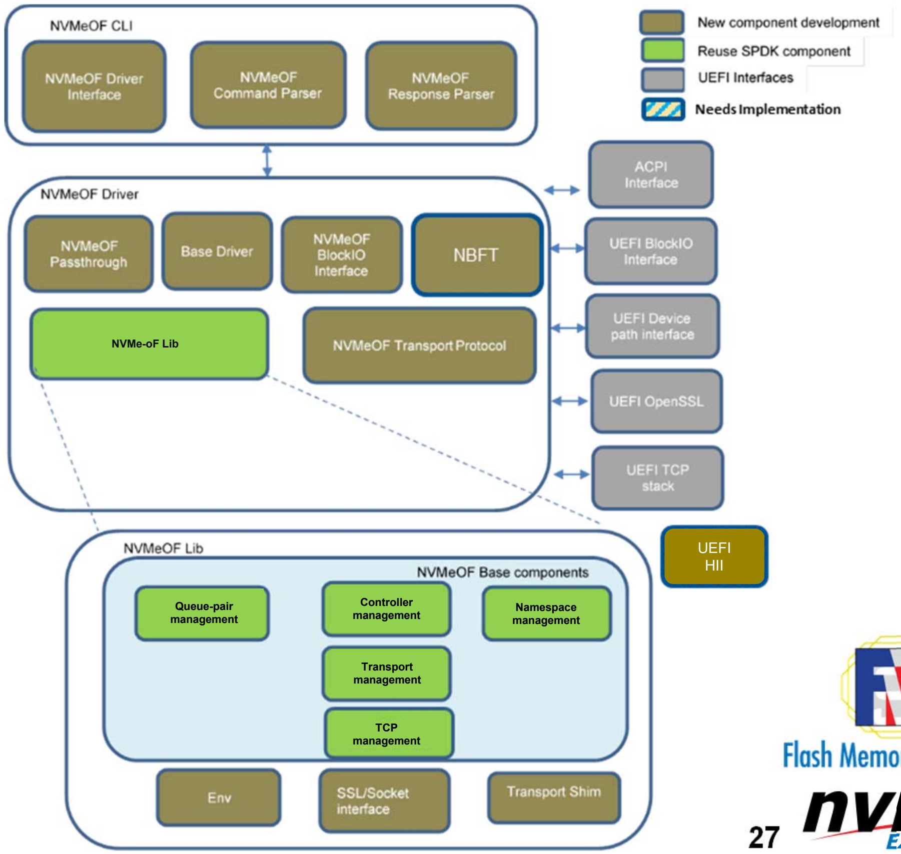
Configuring NVMe-oF™ Boot (UEFI-based example): Pre-Operating System Boot

EDK2 Concept Architecture



EDK2 Reference Architecture as implemented

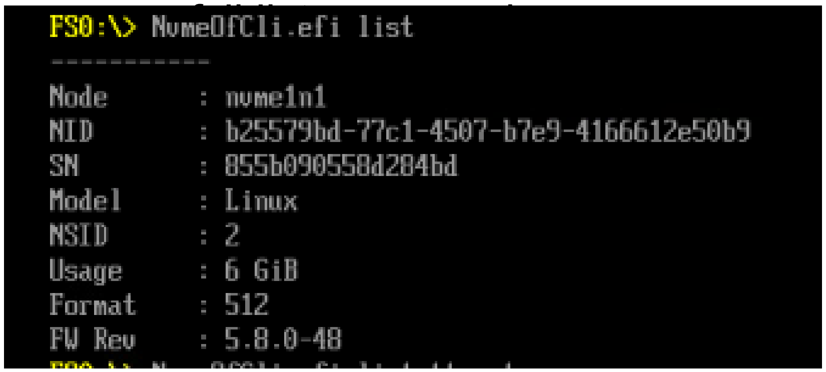
(Not yet published upstream for review)



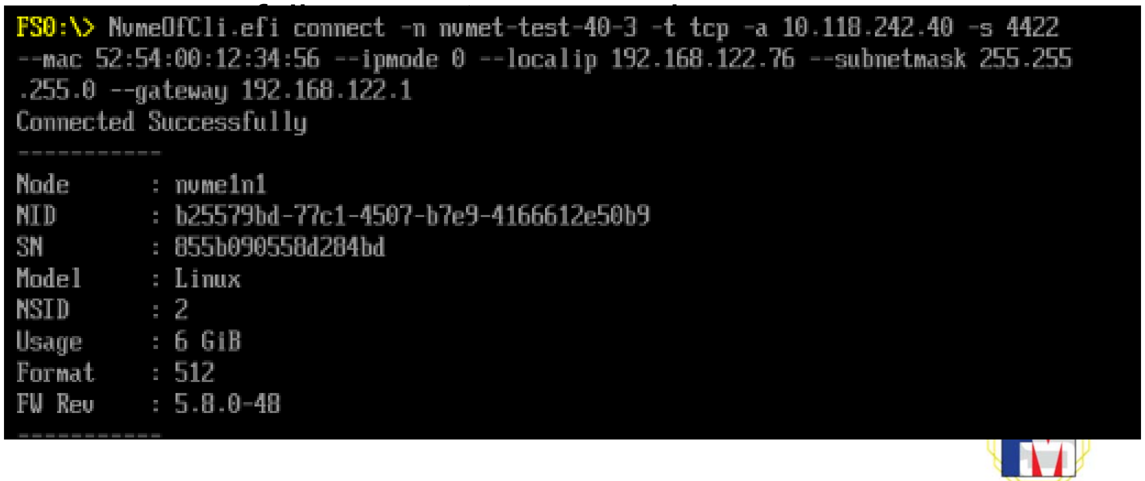
Pre-Boot Environment Configuration Tool

nvmeofcli for EFI:

Command Line tool to facilitate basic diagnostics and interoperability with pre-OS reference driver



```
FS0:\> NvmeOfCli.efi list
-----------
Node        : nvme1n1
NID         : b25579bd-77c1-4507-b7e9-4166612e50b9
SN          : 855b090558d284bd
Model       : Linux
NSID        : 2
Usage       : 6 GiB
Format      : 512
FW Rev      : 5.8.0-48
```



```
FS0:\> NvmeOfCli.efi connect -n nvmet-test-40-3 -t tcp -a 10.118.242.40 -s 4422
--mac 52:54:00:12:34:56 --ipmode 0 --localip 192.168.122.76 --subnetmask 255.255
.255.0 --gateway 192.168.122.1
Connected Successfully
-----------
Node        : nvme1n1
NID         : b25579bd-77c1-4507-b7e9-4166612e50b9
SN          : 855b090558d284bd
Model       : Linux
NSID        : 2
Usage       : 6 GiB
Format      : 512
FW Rev      : 5.8.0-48
-----------
```

OS Handoff Enablement in Reference Design

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 transport:
  - Detects NBFT presence
  - connects pertinent networking
  - uses nvme-cli to connect to NVMe Subsystems/Namespaces

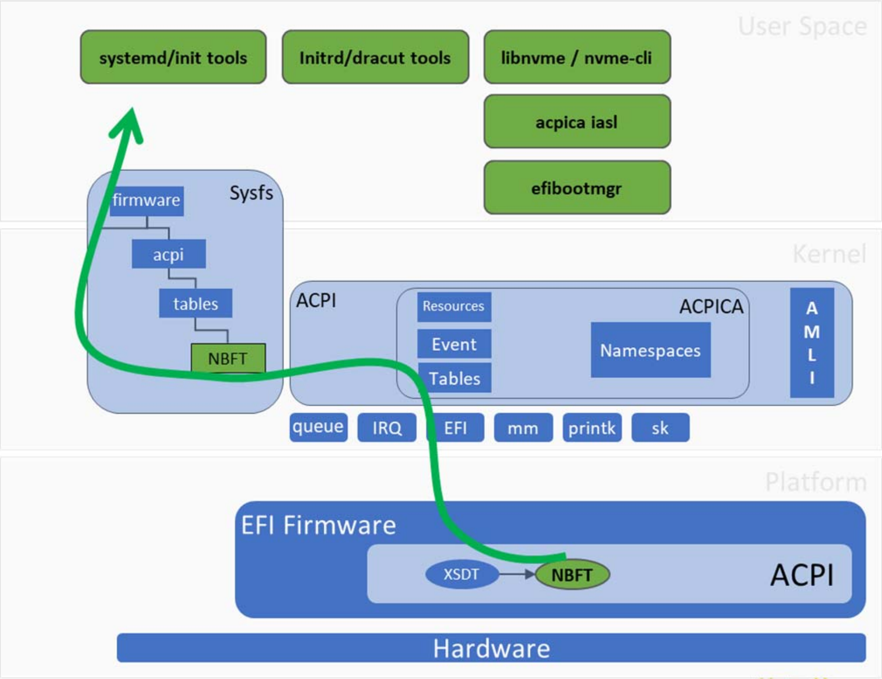
Nvme-cli – two new subcommands:

- 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

Legend**

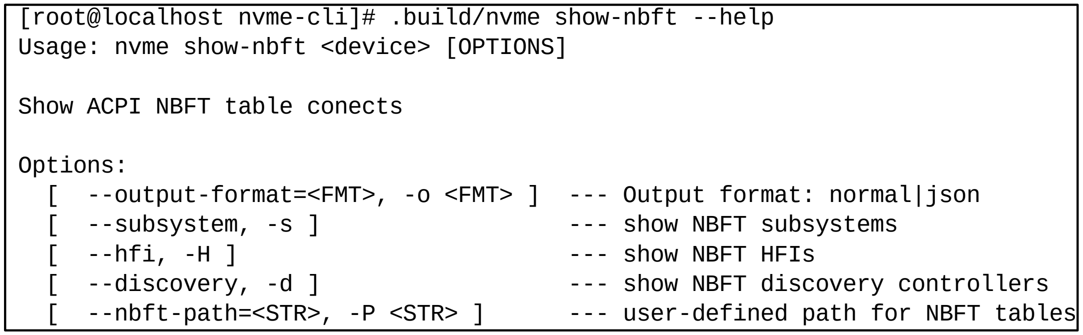
🟩 New Work or Modification

🟦 Existing

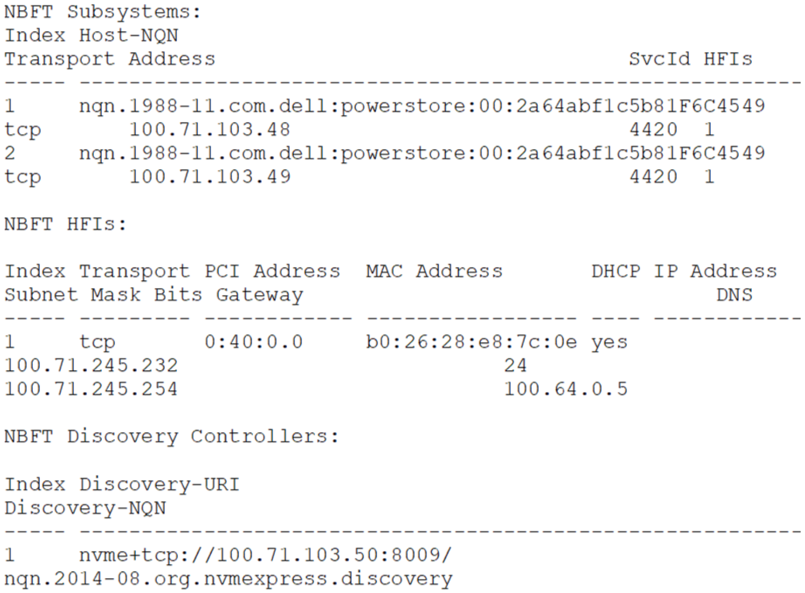


Graphic credit Joey Lee, SUSE

nvme-cli – New subcommands: nvme show-nbft free-text format

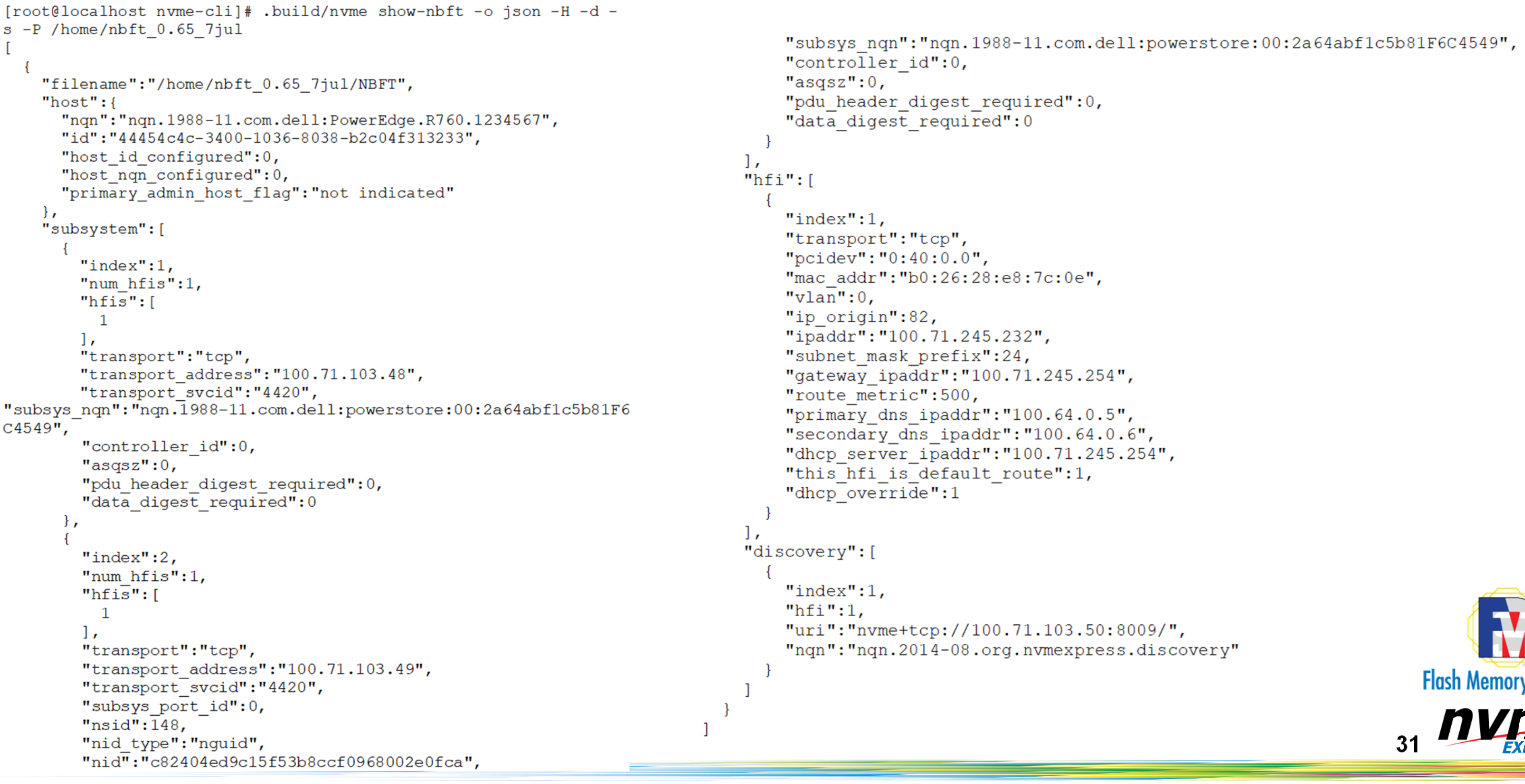


```
[root@localhost nvme-cli]# .build/nvme show-nbft --help
Usage: nvme show-nbft <device> [OPTIONS]

Show ACPI NBFT table conects

Options:
  [  --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>, -P <STR> ]      --- user-defined path for NBFT tables
```



```
NBFT Subsystems:
Index Host-NQN
Transport Address                                 SvcId HFIs
----- ----------------------------------------------------------
1     nqn.1988-11.com.dell:powerstore:00:2a64abf1c5b81F6C4549
tcp       100.71.103.48                           4420  1
2     nqn.1988-11.com.dell:powerstore:00:2a64abf1c5b81F6C4549
tcp       100.71.103.49                           4420  1

NBFT HFIs:

Index Transport PCI Address  MAC Address       DHCP IP Address
Subnet Mask Bits Gateway                                 DNS
----- --------- ------------ ----------------- ---- ------------
1     tcp       0:40:0.0     b0:26:28:e8:7c:0e yes
100.71.245.232                          24
100.71.245.254                          100.64.0.5

NBFT Discovery Controllers:

Index Discovery-URI
Discovery-NQN
----- ----------------------------------------------------------
1     nvme+tcp://100.71.103.50:8009/
nqn.2014-08.org.nvmexpress.discovery
```

nvme-cli – New subcommands: nvme show-nbft JSON format



```
[root@localhost nvme-cli]# .build/nvme show-nbft -o json -H -d -
s -P /home/nbft_0.65_7jul
[
  {
    "filename":"/home/nbft_0.65_7jul/NBFT",
    "host":{
      "nqn":"nqn.1988-11.com.dell:PowerEdge.R760.1234567",
      "id":"44454c4c-3400-1036-8038-b2c04f313233",
      "host_id_configured":0,
      "host_nqn_configured":0,
      "primary_admin_host_flag":"not indicated"
    },
    "subsystem":[
      {
        "index":1,
        "num_hfis":1,
        "hfis":[
          1
        ],
        "transport":"tcp",
        "transport_address":"100.71.103.48",
        "transport_svcid":"4420",
"subsys_nqn":"nqn.1988-11.com.dell:powerstore:00:2a64abf1c5b81F6
C4549",
        "controller_id":0,
        "asqsz":0,
        "pdu_header_digest_required":0,
        "data_digest_required":0
      },
      {
        "index":2,
        "num_hfis":1,
        "hfis":[
          1
        ],
        "transport":"tcp",
        "transport_address":"100.71.103.49",
        "transport_svcid":"4420",
        "subsys_port_id":0,
        "nsid":148,
        "nid_type":"nguid",
        "nid":"c82404ed9c15f53b8ccf0968002e0fca",
```

```
        "subsys_nqn":"nqn.1988-11.com.dell:powerstore:00:2a64abf1c5b81F6C4549",
        "controller_id":0,
        "asqsz":0,
        "pdu_header_digest_required":0,
        "data_digest_required":0
      }
    ],
    "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
      }
    ],
    "discovery":[
      {
        "index":1,
        "hfi":1,
        "uri":"nvme+tcp://100.71.103.50:8009/",
        "nqn":"nqn.2014-08.org.nvmexpress.discovery"
      }
    ]
  }
]
```

Reference Implementations of Booting over NVMe®/TCP Transport

Proof-of-Concept for NVMe Boot

- QEMU based PoCs are available for both openSUSE Leap 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: NVM Express® 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 NVM Express Consortium and the NVM Express Boot Task Group
https://nvmexpress.org/join-nvme/

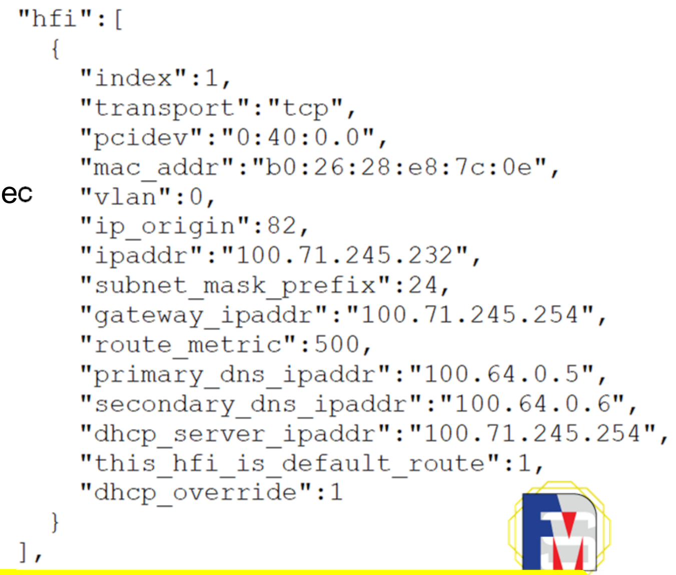
Adding new Transport Support to NVM Express® Boot Specification

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

Bytes	Description
00	Structure ID
01	Version
02	HFI Transport Type.
03	Transport Info Version
05:04	HFI Descriptor Index

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/

UEFI 2.10: https://uefi.org/specs/UEFI/2.10/10_Protocols_Device_Path_Protocol.html

ACPI 6.5: https://uefi.org/specs/ACPI/6.5/05_ACPI_Software_Programming_Model.html

Open-Source Software Repos: 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?

nvm EXPRESS®

Architected for Performance