

SupremeJBOF Implementation Manual

GP Spark Powered by Graid Technology SupremeRAID™ AEC

Stage	MP
Version	1.1.1

File Update History

Date	Updated By	Version	Preparation / Modification Content
2026-09-30	Liam	1.1	• Updated RAID creation and configuration recovery procedures. • Clarified product names, platform requirements and troubleshooting.
2026-10-12		1.1.1	• Added namespace setup procedures for multiple DGX hosts. • Clarified the setup sequence, host tasks and license requirements. • Updated command examples, troubleshooting and FAQs.

Table of Contents

1	Abstract	4
1.1	Product Introduction	4
2	System Requirements for Product Integration	5
2.1	RDMA Network Cards (NICs)	5
2.2	High-Speed Cables	6
3	SupremeJBOF Connection and Mounting Process	6
3.1	Network Configuration	6
3.2	Network Issue Troubleshooting	7
3.3	Modifying NIC Auto-Negotiation	7
3.4	Installing NVMe Software	8
3.5	Loading RDMA Drivers	8
3.6	Configuring Jumbo Frames on the NIC	8
3.7	NVMe SSD Disk Discovery	9
3.8	Connecting NVMe Namespaces (Standalone Access)	9
4	Important Notes	9
4.1	Avoid Adding NVMe-oF Disk Mount Entries to /etc/fstab	9
4.2	Releasing a Virtual Drive Before Modifying It	10
5	System Architecture	12

Table of Contents (continued)

6	SupremeRAID™ AEC Integration	13
6.1	Quick Start with the Automated Setup Script	14
6.2	Installing SupremeRAID™ AEC	17
6.2.1	Troubleshooting: Replacing the Pre-built NVIDIA Driver with a DKMS Build.....	18
6.3	Creating PD / DG / VD from GP Spark NVMe-oF Devices	20
6.3.1	Connecting GP Spark as a Remote NVMe-oF Target	21
6.3.2	Creating Physical Drives, Drive Groups, and Virtual Drives	22
6.4	Upgrading SupremeRAID™ AEC.....	26
6.5	Restoring the SupremeRAID™ AEC Configuration	26
6.6	Sharing GP Spark Across Multiple DGX Spark Hosts.....	31
6.6.1	Prepare the allocation and management connection	33
6.6.2	Provision namespaces with a saved plan.....	35
6.6.3	Assign namespaces and connect each DGX	37
6.6.4	Advanced manual namespace provisioning.....	40
6.6.5	Advanced replacement and recovery	43
6.6.6	Multi-host FAQ.....	46
7	Uninstalling	47
7.1	Full DGX Spark Restore	47
7.2	Remove SupremeRAID™ AEC Only	47
8	Troubleshooting	48
8.1	SupremeRAID™ AEC service does not start.....	48
8.2	Drive group keeps entering RESYNC.....	48
8.3	Create DG/VD fails with graid_create_vd_ns error.....	49
8.4	graidctl cannot connect to the remote target.....	50
8.5	DGX Spark GPU not recognized by the installer (legacy workaround).....	51
8.6	Other SupremeRAID™ AE platforms: GPU fabric prerequisites	51

1 Abstract

This document primarily describes the implementation process and precautions for the Graid Technology and Luisuan Technology GP Spark. For more detailed product information, please refer to the User Manual and User White Paper.

Before you begin — please read. This guide covers integrating SupremeRAID™ AEC with the GP Spark NVMe-oF storage node on a supported NVIDIA DGX Spark or another specifically supported NVIDIA GB10-based system running NVIDIA DGX OS. Confirm that your exact host, operating system, driver, and installer versions are supported before installation.

- **What this is.** SupremeRAID™ AEC provides GPU-accelerated RAID data protection across the GP Spark drives. It is a data-protection solution, not a plug-and-play capacity-expansion box.
- **Skills required.** Installation uses the Linux command line, network configuration, driver preparation, and system services. Administrators should be familiar with Linux. Contact Graid Technology for assistance if needed; an automated setup script is available in Section 6.1.
- **Not plug-and-play.** Bringing the device up requires several configuration commands; this is expected for an enterprise storage integration.
- **Usable capacity.** Redundant RAID levels reserve capacity for protection; RAID 0 provides no redundancy. For example, RAID 5 across four equal-sized drives yields three drives' worth of usable capacity. See Section 6.3.2.

Before installation, please ensure that the above requirements and key considerations are fully understood.

1.1 Product Introduction



SupremeJBOF combines GP Spark NVMe-oF storage with Graid Technology SupremeRAID™ AEC running on a supported NVIDIA DGX Spark or another specifically supported NVIDIA GB10-based host running NVIDIA DGX OS. GP Spark supplies the external flash storage; SupremeRAID™ AEC uses the host GPU for RAID processing and data protection. JBOF stands for Just a Bunch of Flash. GP Spark is designed by Luisuan Technology as an external storage companion for AI development hosts.

Luisuan Technology's GP Spark is a data-center-grade NVMe-oF storage node. Through its hardware offload engine, the RDMA (RoCEv2) and NVMe-oF protocol stack are fully implemented end-to-end within a dedicated chip, achieving end-to-end data acceleration. Housed in a compact 0.81L unibody aluminum-magnesium alloy chassis (150mm × 150mm × 35mm), GP Spark integrates a complete RDMA and NVMe-oF protocol stack into a dedicated ASIC chip via a hardware offload engine. The GP Spark hardware described here supports 100GbE networking using 4 × 25G NRZ. With GPUDirect RDMA, data flows directly into GPU memory — delivering end-to-end latency under 4μs, throughput of up to 60 Gb/s, and up to 1.9 million IOPS with SupremeRAID™ AEC RAID 5 data-protection features, all within a total power envelope of under 65W. The passively-cooled unibody chassis ensures silent operation even under sustained 100% load. -

2 System Requirements for Product Integration

GP Spark: storage connectivity compatibility

The following list describes GP Spark/EBOF storage connectivity only. It does not establish SupremeRAID™ AEC or DGX Spark operating-system support. Verify the GP Spark hardware, firmware, NIC and cabling with Luisuan Technology; the connection used in this guide is 100GbE / 4 × 25G NRZ.

- CentOS 7.5 or later.
- Ubuntu 18.04 or later.
- Domestic operating system Kylin V10 SP1 or later.
- UnionTech UOS, CS2C , and Loongson operating systems.
- Rocky Linux release 9 or later.
- For operating systems not listed here, confirm GP Spark connectivity requirements with Luisuan Technology and SupremeRAID™ AEC compatibility with Graid Support. A minimum kernel version alone does not establish support for the integrated solution.

SupremeRAID™ AEC: driver and host support

The official AE Linux Driver 2.0.1 release notes introduce SupremeRAID™ AI Edition Compact (AE Compact), referred to as SupremeRAID™ AEC in this guide. For deployment, apply the following current platform and OS support requirements together with the release-specific driver and licensing information.

Host platform: SupremeRAID™ AEC currently supports only selected NVIDIA GB10-based systems, including supported NVIDIA DGX Spark configurations. A GB10 GPU alone does not establish platform support.

Driver and license: standard AE driver 2.0.1 with a dedicated AE Compact license; driver 2.0.1 requires a V2 license key.

Installer architecture: use the ARM64 (aarch64) pre-installer for DGX Spark / GB10.

OS scope: SupremeRAID™ AEC currently supports only NVIDIA DGX OS on the supported GB10-based systems. Support does not extend to every GB10 system or every DGX OS release.

Before deployment, confirm the exact GB10 host model, NVIDIA DGX OS release, kernel, NVIDIA driver, AEC package and license combination with Graid Support.

Source: SupremeRAID™ AE Linux Driver 2.0.1 Release Notes (September 10, 2026), New Features, Important Notes, OS and Platform Support, and Dependencies and Utilities.

<https://docs.graidtech.com/sr-ae/release-notes/linux/2.0/2.0.1>

2.1 RDMA Network Cards(NICs)

Network speed and link compatibility. The GP Spark hardware covered by this guide supports a 100GbE connection using four 25 Gb/s NRZ lanes (4 × 25G NRZ). The host NIC or switch port and cable/transceiver combination must support this signaling mode. These four lanes form one 100GbE link, not four independent 25GbE ports.

Luisuan Technology uses Mellanox network cards from NVIDIA, as well as its own self-developed network cards, to connect GP products.

- NVIDIA Mellanox ConnectX Series
- Luisuan Technology LS00201V11 Series

2.2 High-Speed Cables

Use DAC (Direct Attach Copper), AOC (Active Optical Cable), or optical transceivers and fiber compatible with the 100GbE / 4 × 25G NRZ connection. Confirm compatibility across the GP Spark port, host NIC or switch port, and cable/transceiver combination. A nominal speed rating alone does not ensure compatibility with this signaling mode (Section 2.1).

3 SupremeJBOF Connection and Mounting Process

3.1 Network Configuration

Reading the operation blocks. Gray blocks contain commands or configuration text that you can copy. Blue blocks show illustrative output or outcomes, not a record of a run on your system. Output wording, IDs and device paths may differ. Replace all example paths and placeholders before running a command. Run privileged commands as root or with sudo. Treat maintenance, creation and restoration as separate procedures; do not execute every block in sequence.

After completing the network configuration (assigning an IP address to the port, as shown in the figure below.), attempt to ping the storage port (**default IP is 10.10.10.12/24**).

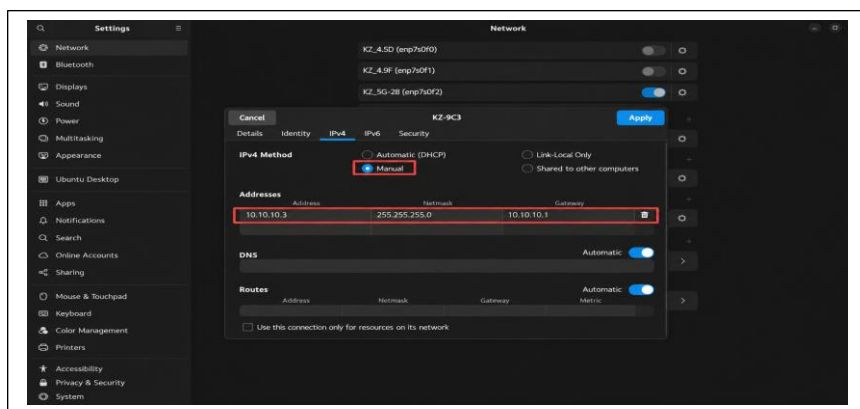
Commands

```
ping -c 4 10.10.10.12
```

Illustrative result

```
64 bytes from 10.10.10.12: icmp_seq=1 ttl=64 time=0.8 ms
# A reply proves IP reachability, not RDMA service availability.
```

The default storage-port address is 10.10.10.12/24. If the configured address is unknown, check the device web UI and network configuration. A ping reply confirms IP reachability; verify NVMe-oF discovery separately.



[Figure 3.1: Desktop network settings — manual IPv4 configuration of the network port]

Changing the GP Spark NVMe-oF IP address. The default storage-port IP is 10.10.10.12/24. To change it, use the built-in web UI:

- Connect to the GP Spark's built-in Wi-Fi access point — a local, isolated network the GP Spark hosts itself, with no internet access (default Wi-Fi password: 12345678).

- Open the web UI in a browser at <http://192.168.10.1> (the GP Spark's own Wi-Fi network gateway) and log in (default username admin / password 123).
- Go to the Storage page, where the IP address field can be edited (default 10.10.10.12).

Note: If the NVMe-oF IP is changed here, use the new address in every subsequent command (ping, nvme discover, and graidctl connect remote_target). The Wi-Fi and web UI defaults above are factory credentials; change them for any non-isolated deployment.

3.2 Network Issue Troubleshooting

For SupremeRAID™ AEC and RAID-related issues, see Section 8 (Troubleshooting).

If the GP Spark port can be pinged normally, the following content may be skipped.

If the port cannot be pinged, verify the following items :

1. The cable is properly seated, with no looseness.
2. The port status is UP.
3. The host storage interface and GP Spark storage port use compatible subnet settings; the factory example is 10.10.10.0/24. Use the actual subnet if the device address has been changed.

3.3 Modifying NIC Auto-Negotiation

Note: If all of the above conditions are met but the NIC port still cannot reach the UP state, follow these steps:

The commands below configure the 100GbE connection using 4 × 25G NRZ described in this guide. Confirm that the NIC or switch port supports this mode before changing auto-negotiation. Setting speed to 100000 does not itself select or guarantee the signaling mode. Replace enP2p1s0f0np0 with your actual NIC interface and check link state.

Disable auto-negotiation and manually set the speed to 100Gbps:

Commands

```
sudo ethtool -s enP2p1s0f0np0 speed 100000 duplex full autoneg off
sudo ethtool enP2p1s0f0np0
```

Illustrative result after link verification

```
Speed: 100000Mb/s
Auto-negotiation: off
Link detected: yes
```

Re-enable auto-negotiation:

Commands

```
sudo ethtool -s enP2p1s0f0np0 autoneg on
sudo ethtool enP2p1s0f0np0
```

Illustrative result

```
Auto-negotiation: on
Link detected: yes
```

3.4 Installing NVMe Software

The host system integrating with the GP Spark product requires NVMe-related management tools to connect to the JBOF device. After configuring the IP address, install nvme-cli first. The host must have internet access to download the package.

Commands for a supported CentOS connectivity host

```
sudo yum install -y nvme-cli
```

Illustrative result

```
# Package manager completes installation of nvme-cli.  
# Review any dependency or repository error before continuing.
```

Commands for DGX OS or Ubuntu

```
sudo apt-get install nvme-cli
```

Illustrative result

```
# Package manager completes installation of nvme-cli.  
# AEC hosts must still meet the DGX OS requirements in Chapter 2.
```

3.5 Loading RDMA Drivers

Use the following commands to load the RDMA-related kernel modules:

Commands

```
modprobe nvme-fabrics  
modprobe rdma_cm  
modprobe nvme  
modprobe -v nvme-core multipath=0  
modprobe -v nvme-rdma register_always=0
```

Illustrative result

```
# Module loading may be silent when modules are already loaded.  
# Stop if modprobe reports an unsupported parameter or load error.
```

3.6 Configuring Jumbo Frames on the NIC

Replace enP2p1s0f0np0 with your storage NIC. Confirm that the target and any switch in the path support the selected MTU.

Commands

```
sudo ip link set dev enP2p1s0f0np0 mtu 5000  
ip link show dev enP2p1s0f0np0
```

Illustrative result

```
enP2p1s0f0np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 5000
```

3.7 NVMe SSD Disk Discovery

Use the following command to discover the NVMe SSD targets on the GP Spark:

Commands

```
TARGET_IP=10.10.10.12
sudo nvme discover -t rdma -a "$TARGET_IP" -s 4420
```

Illustrative result

```
subtype: nvme subsystem
subnqn: nqn.2026-10.com.example:gpspark-ssd0
traddr: 10.10.10.12
trsvcid: 4420
trtype: rdma
```

3.8 Connecting NVMe Namespaces (Standalone Access)

TARGET_IP is the storage-port IP address and SSD_NQN is the exact storage subnqn from discovery. This nvme-cli connection method is for standalone namespace access. For normal AEC setup, use Section 6.3.1 instead. Section 6.5 uses nvme-cli during recovery while RAID services are stopped. Connecting a namespace does not mount a filesystem.

Commands

```
TARGET_IP=10.10.10.12
SSD_NQN='replace-with-exact-storage-subnqn'
sudo nvme connect -t rdma -a "$TARGET_IP" -s 4420 \
-n "$SSD_NQN" --nr-io-queues=8
nvme list-subsys
nvme list
```

Illustrative result

```
NQN=nqn.2026-10.com.example:gpspark-ssd0
+- nvme3 rdma traddr=10.10.10.12,trsvcid=4420 live
# The namespace appears in nvme list; no filesystem is mounted.
```

4 Important Notes

4.1 Avoid Adding NVMe-oF Disk Mount Entries to /etc/fstab

This applies to the **bare NVMe-oF namespaces** — the devices you connect directly with nvme-cli when using the GP Spark **without** SupremeRAID™ AEC (Section 3.8). Because the GP Spark exposes storage over the NVMe-oF Ethernet protocol rather than a local PCIe bus, the OS cannot reach these devices during the early boot sequence when /etc/fstab runs its mounts. A standard fstab entry for such a namespace will fail to mount and can prevent the system from booting.

If this has already caused a boot failure, enter Single User Mode, edit /etc/fstab, remove the NVMe-oF namespace entries, boot normally, then connect and mount the namespaces manually with nvme-cli.

When using SupremeRAID™ AEC, this restriction does not apply to the virtual drives. With AEC, the SupremeRAID™ AEC service connects the GP Spark NVMe-oF target itself and exposes virtual drives such as /dev/gdgXnY. Those VDs **can** be placed in /etc/fstab for persistent mounting, provided the entry waits for the

network and the SupremeRAID™ AEC service (see Section 6.3.2). Do not add the underlying namespaces to fstab — only the AEC virtual drives.

4.2 Releasing a Virtual Drive Before Modifying It

Before an operation requiring a SupremeRAID™ AEC virtual drive (VD) or drive group (DG) to be offline, stop I/O and release every affected VD. Run the commands below as root, substituting the actual device and mount point:

1. Stop applications, jobs and services using the affected RAID volumes, including raw-device I/O. Prevent them from restarting during maintenance.

2. If a systemd automount is configured, stop that automount unit before unmounting so it cannot remount the volume. Use the actual mount-point-derived unit name.

Commands

```
# Only if a systemd automount is configured for this mount point:  
systemctl stop "${systemd-escape --path --suffix=automount /mnt/graid}"
```

Illustrative result

```
# No terminal output is expected on successful stop.  
# The configured automount must not restart during maintenance.
```

3. Check whether the filesystem is still mounted. Stopping automount may already have unmounted it. Run `umount` only if the filesystem remains mounted; if it is busy, identify and stop the holders before retrying.

Commands

```
findmnt -rn -M /mnt/graid
```

Illustrative result

```
/mnt/graid /dev/gdg0n1 xfs rw,relatime  
# If no mount is listed, skip umount and verify holders below.
```

Commands

```
umount /mnt/graid
```

Illustrative result

```
# No terminal output on successful unmount.  
# If busy, stop holders; do not force the unmount.
```

4. Verify that the filesystem is unmounted and no process holds the VD open. Do not proceed while mounts or holders remain. An empty result is expected from the following checks once the VD is released:

Commands

```
findmnt -rn -M /mnt/graid  
lsof /dev/gdg0n1  # anything still holding it open?  
fuser /dev/gdg0n1
```

Illustrative result

```
# No mount or holder output is expected.  
# Empty output alone does not prove remote initiator I/O has stopped.
```

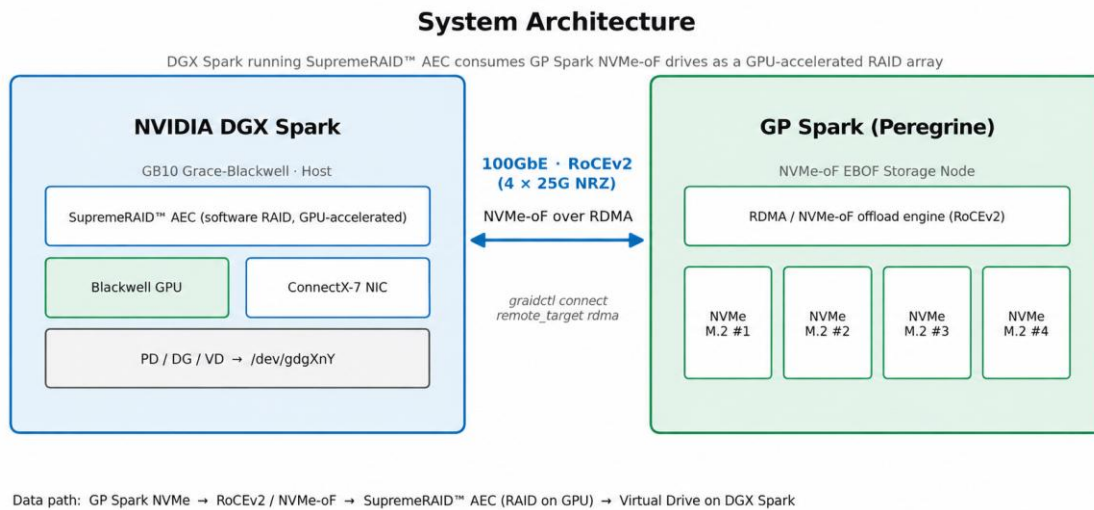
⚠ Warning Skipping this step is a common cause of failures elsewhere in this guide:

1. Create operations that fail (see Section 8.3)
2. Upgrade processes that fail or hang (see Section 6.4)

Both sections refer back to this point.

5 System Architecture

GP Spark is the external NVMe-oF storage node; SupremeRAID™ AEC runs on the GPU of a supported NVIDIA DGX Spark or another specifically supported GB10-based host running NVIDIA DGX OS. The GP Spark connection uses 100GbE over RoCEv2 with 4 × 25G NRZ and compatible ports and cabling (Sections 2.1 and 2.2). SupremeRAID™ AEC attaches remote namespaces through its Remote NVMe-oF Target feature and exposes GPU-accelerated RAID virtual drives.



[Figure 5.1: DGX Spark with SupremeRAID™ AEC connected to GP Spark over 100GbE / RoCEv2 using 4 × 25G NRZ.]

6 SupremeRAID™ AEC Integration

Install SupremeRAID™ AEC on each supported host, then connect GP Spark and create the required RAID volume. Choose one deployment path below. Upgrading (6.4) and restoration (6.5) are separate maintenance procedures.

Deployment	Start here	Continue with
One host, automated installation	6.1 Setup script	6.3 if create_raid=false; otherwise verify the created volume.
One host, manual installation	6.2 Installer procedure	6.3 Connect the target and create PD / DG / VD.
Multiple DGX hosts	6.6 Deployment order	Install with create_raid=false; create namespaces, assign hosts, then build each RAID.

Connection method. SupremeRAID™ AEC manages the GP Spark NVMe-oF connection through its Remote NVMe-oF Target feature. Connect namespaces with graidctl (Section 6.3.1); the service stores the connection and re-establishes it after reboot. No separate nvme-cli auto-connect service is required for normal operation.

Note — GPU sharing with other CUDA workloads. SupremeRAID™ AEC uses the bound GPU only while it is servicing RAID I/O and releases it when idle, so another CUDA workload (for example an inference engine) can run on the same GPU. Sustained, high-frequency RAID I/O increases contention with that workload; size the GPU accordingly if you plan to co-locate a demanding CUDA application. Contact your Graid partner for guidance on specific co-location scenarios.

Pre-flight checklist. Confirm the following before you begin the SupremeRAID™ AEC installation:

- **GP Spark reachable** — the host can ping the configured storage address and discover its NVMe-oF target over RDMA (Sections 3.1 and 3.7).
- **RDMA stack ready** — the RDMA NIC drivers are loaded and the link is UP at the intended speed with jumbo frames set (Sections 3.5, 3.6).
- **UEFI Secure Boot — Disable Secure Boot before installing SupremeRAID™ AEC. Before re-enabling it after installation, ensure that all required kernel modules, including the NVIDIA and SupremeRAID™ AEC modules, are signed and that their signing keys are enrolled and trusted by the system. See Section 6.2 for the post-installation checks.**
- **NVIDIA driver** — record the version and check dkms status. If the installed driver is not DKMS-managed and the installer requires DKMS sources, follow Section 6.2.1.
- **Packages on hand** — the SupremeRAID™ AEC pre-installer and installer for your product line, plus (recommended) the setup script and its gspark_setup.conf, from your Graid partner (Section 6.1).
- **Internet access** — available so the pre-installer can fetch dependencies.
- **RAID plan** — decide the RAID level and how many GP Spark drives to use before creating the drive group (Section 6.3.2).

6.1 Quick Start with the Automated Setup Script

Audience and outcome

This quick start is for administrators with basic Linux command-line skills deploying SupremeJBOF for the first time. The goal is to connect GP Spark, install SupremeRAID™ AEC, and verify a usable RAID volume. Choose installation only or installation plus RAID creation before the first run.

What the script does

gpspark_setup.py checks the NVIDIA driver and replaces a pre-built driver with a DKMS-managed version when needed, runs the pre-installer and installer, saves progress across required reboots, optionally creates and mounts RAID, and verifies the SupremeRAID™ AEC services.

The script does not configure the host network, change UEFI Secure Boot settings, sign kernel modules, or enroll signing keys. Complete those prerequisites separately. RAID creation is disabled by default.

1. Prepare the host and storage

Connect the host and GP Spark using compatible ports and cabling. Use the 100GbE / 4 × 25G NRZ connection configuration (Sections 2.1–3.3). Configure the host storage-network address, load the RDMA stack, and confirm ping and NVMe discovery succeed (Sections 3.1–3.7). For normal AEC setup, leave target connection to graidctl/the setup script rather than connecting namespaces with nvme connect first. For initial multi-host namespace provisioning, complete Section 6.6 before creating RAID objects.

Disable Secure Boot and reboot before installation. Re-enable it only after confirming signatures and trusted signing keys for all required NVIDIA and SupremeRAID™ AEC modules; then verify module loading and services after reboot (Section 6.2).

Obtain the matching SupremeRAID™ AEC pre-installer and installer for the supported host and release through your approved Graid channel. Prepare Internet access and activation information. Stop workloads and back up any data on drives intended for RAID creation.

2. Download and prepare the setup package

Use a persistent local directory on the host, keep the script and configuration together, and keep the directory available across reboots. Run the following commands in that directory:

Commands

```
wget https://download.graidtech.com/misc/tools/gpspark_setup/gpspark_setup.zip
unzip gpspark_setup.zip
```

Illustrative result

```
# Download completes and the archive extracts the setup files.
gpspark_setup.py
gpspark_setup.conf
# Installer packages are supplied separately.
```

Edit gpspark_setup.conf before running the script. Place the pre-installer and installer in the absolute directory configured by [supremeraid] package_dir; the default is /opt/graid/packages. The downloaded setup ZIP contains the script and its configuration, not those installer packages.

3. Review the configuration

[nvidia]: check driver_version, dkms_package, and prebuilt_package against nvidia-smi, the installed kernel, and the matching package names. Do not assume the supplied pinned version matches your host.

[supremeraid]: check package_dir and the installer filename patterns. Confirm product_line with the supplied installer; the accompanying script uses ae, and the AEC product name does not imply an aec CLI value. For automated RAID creation, ensure a license is already applied or provide the appropriate license_key; keep any populated configuration file private.

[raid]: confirm nvme_address (target IP), nvme_svcid (port), nvme_transport, and nvme_address_family. The example defaults are 10.10.10.12, 4420, rdma, and ipv4. Normal installation uses these file values; the interactive target prompts belong to --restore-config only.

[behavior]: auto_resume=true enables automatic continuation after the required reboots. Keep the script, configuration, and installer files accessible at their configured paths.

4. Choose whether to create RAID

Option A — install software only (default): keep [raid] create_raid=false. The script installs software and verifies services but does not create a usable RAID volume. Complete activation and GPU binding as applicable (Section 6.2), then create the PD / DG / VD and filesystem using Section 6.3.

Option B — install and create RAID: set [raid] create_raid=true only after verifying the intended drives and backing up their data. Review nvme_devices, pd_ids, raid_type, vd_size, filesystem, mount_point, and fstab. This path creates RAID objects and formats the virtual drive; it is not a recovery procedure for existing data.

With nvme_devices=auto, the script selects only NVMe paths newly exposed by its target connection. If the target is already connected and no new devices appear, inspect graidctl list nvme_drive and configure the intended device paths explicitly. Never select the host's system disk.

5. Run the setup and follow reboot progress

Run preflight first. It reports failures before installation starts. Missing NVIDIA DKMS is a warning for AEC; the current setup flow may still attempt DKMS replacement. Review the driver configuration before starting installation.

Commands for preflight

```
sudo python3 gpspark_setup.py --dry-run
```

Illustrative result

```
[PASS] Secure Boot is disabled
[PASS] Exactly one pre_installer package
[PASS] Exactly one installer package
...
Preflight passed. No installation or configuration changes
were made by these checks.
```

Commands

```
sudo python3 gpspark_setup.py
```

Illustrative result

```
# The script runs the selected setup stages.
# A required reboot may disconnect the terminal.
# With auto_resume=true, the saved stage continues after reboot.
```

Run from the directory containing the script. Expect reboots during driver preparation and installation. With auto_resume=true, the saved next stage resumes automatically after reboot; with auto_resume=false, log in and run the same command again. Do not start a second setup process while one is running.

Commands

```
sudo python3 gpspark_setup.py --status
```

Illustrative result

```
current stage : DONE
# Also inspect runtime and next action. DONE alone does not
# prove that no setup process is still running.
```

Use `--status` to inspect the current stage, runtime state, and next action. If a stage fails, review `/var/lib/gpspark-setup/setup.log`, resolve the reported issue, and follow the reported next action. Do not use `--reset` as a routine retry command; it clears saved progress.

6. Verify completion

Software installed: confirm `--status` reports stage DONE and use the standalone service check below. Service verification does not create RAID or prove that a filesystem is ready.

Commands

```
sudo python3 gpspark_setup.py --verify
```

Illustrative result

```
[OK] graidcore@0.service is active.
[OK] graid.service is active.
[OK] All SupremeRAID services are active.
```

RAID ready for use: after automated or manual creation, check the target, physical drives, drive groups, virtual drives, and mounted filesystem:

Commands

```
sudo graidctl list remote_target
sudo graidctl list physical_drive
sudo graidctl list drive_group
sudo graidctl list virtual_drive
ls /mnt/graid
findmnt -M /mnt/graid
df -h /mnt/graid
```

Illustrative result

```
# Example acceptance outcome after RAID creation:
Remote target: expected RDMA address and service port
PD / DG / VD: expected members, healthy and online
/mnt/graid: mounted from the intended AEC virtual drive
```

Replace `/mnt/graid` with your configured mount point. First access triggers mounting when automount is enabled. Confirm the expected virtual drive is mounted, the RAID objects are healthy, and the capacity matches your plan. Refer to Section 6.3.2 for state checks and Section 8 for troubleshooting.

If `create_raid=false`, a completed setup alone is not the end of deployment: follow Section 6.3 to build and verify the volume. Changing `create_raid` after a completed installation does not automatically replay the completed CREATE_RAID stage.

Other workflows and version notes

For existing RAID configuration recovery, use `--restore-config` in Section 6.5. For removal or state cleanup, use Section 7. Neither operation is part of the first-install quick start.

The setup script uses the installer's supported-GPU list by default. Leave `[supremeraid] expected_gpu_cards` empty. Set a legacy override only when Graid Support approves it for the supplied package (Section 8.5).

6.2 Installing SupremeRAID™ AEC

Prerequisites

- CPU: 2 GHz or faster with at least 8 cores; RAM: 16 GB minimum.
- Use a specifically supported NVIDIA GB10-based host running a supported NVIDIA DGX OS release and a matching SupremeRAID™ AEC installer (Section 2). Internet access is required for the pre-installer to download dependencies.
- UEFI Secure Boot: disable Secure Boot in the host's UEFI/BIOS settings, save the changes, and reboot before running the pre-installer or installer. After installation, re-enable Secure Boot only after confirming that all required kernel modules, including the NVIDIA and SupremeRAID™ AEC modules, are signed and their signing keys are enrolled and trusted by the system. After rebooting with Secure Boot enabled, verify that the required modules load successfully and that the SupremeRAID™ AEC services and RAID volumes operate normally (Sections 8.1 and 6.3.2). Follow the host vendor's instructions for firmware settings and key enrollment.

Step 1 — Run the Pre-installer. Download the latest **SupremeRAID™ AEC pre-installer** (which includes the NVIDIA driver) from the Graid Technology website or EagleHeart website, make it executable, and run it. The pre-installer configures all required dependencies and the NVIDIA driver automatically.

Commands

```
PRE_INSTALLER='/absolute/path/to/pre-installer.run'
sudo chmod +x "$PRE_INSTALLER"
sudo "$PRE_INSTALLER" --productline=ae --yes
```

Illustrative result

```
# The approved pre-installer prepares its dependencies.
# Follow the package output; --yes may allow a reboot.
```

Use the product-line argument accepted by the supplied package. The command shown here retains the `ae` argument used by the accompanying setup script; the product name SupremeRAID™ AEC does not imply that the CLI accepts `aec`. Confirm the argument for your installer release. The `--yes` flag permits automatic reboot.

Step 2 — Run the Installer. Obtain the SupremeRAID™ AEC package approved for your GB10 host and driver release, make it executable, and run it. Confirm the exact package with Graid Support rather than selecting a package solely from a generic GPU-architecture suffix.

Commands

```
INSTALLER='/absolute/path/to/installer.run'  
sudo chmod +x "$INSTALLER"  
sudo "$INSTALLER" --accept-license
```

Illustrative result

```
# The approved AEC package completes installation.  
# Activation and GPU binding are separate steps.
```

The `--accept-license` flag accepts the end-user license agreement non-interactively. Product activation and GPU binding are separate steps.

DGX Spark GPU recognition. Use a current installer that supports the host and attempt normal installation first. No supported-GPU override is expected as part of the standard procedure.

If the installer rejects the GPU, follow Section 8.5. Do not create `/etc/graid_pre_installer.conf` solely because the host is a DGX Spark.

Step 3 — Complete installation. When using the command above, the EULA is already accepted. If using an interactive installer instead, follow its license-acceptance prompts. Continue to Management Console activation after installation completes.

Step 4 — Open the Management Console. Browse to `https://<DGX-management-IP>:50060` and allow TCP port 50060 as needed. This is the AEC console on the DGX, separate from the GP Spark web UI at `http://192.168.10.1`. If port 50060 is occupied, edit `/etc/graidmgr/service.conf` (`web_port` and `web_addr`), then restart `graid-mgr`.

Step 5 — Enter the host token. Enter the host token into the Management Console. If the token is lost, regenerate it with:

Commands

```
sudo graid-mgr host_token gen
```

Illustrative result

```
<generated-host-token>  
# Use the actual token in the Management Console; keep it private.
```

Step 6 — Select a license mode and bind the GPU. Choose Perpetual (requires a V2 license key for the 2.0 driver) or Subscription (monthly billing). For Perpetual, set a password; for Subscription, log in with your portal email and password. Finally, bind your GPU to start the RAID setup.

6.2.1 Troubleshooting: Replacing the Pre-built NVIDIA Driver with a DKMS Build

When to use this procedure. AEC can use a non-DKMS NVIDIA driver; absence of NVIDIA DKMS alone is not an installation failure. Use this driver-replacement procedure only when your approved installer or Graid Support requires a DKMS build. The current setup flow may still attempt replacement, even though preflight treats missing DKMS as a warning.

If replacement is required, use the matching DKMS open-driver version, rebuild it for the running kernel and refresh `initramfs`. Review the exact package candidate before removing a pre-built module. Run the following as root and stop if any command fails:

Commands

```
KVER="$(uname -r)"
ARCH="$(uname -m)"
NVVER="$(nvidia-smi --query-gpu=driver_version --format=csv,noheader | head -n1)"
BRANCH="${NVVER%%.*}" # major version, e.g. 580
PREBUILT="linux-modules-nvidia-${BRANCH}-open-${KVER}"
echo "Kernel: $KVER"
echo "Arch: $ARCH"
echo "NVIDIA: $NVVER"
```

Illustrative result

```
Kernel: 7.0.0-1019-nvidia
Arch: aarch64
NVIDIA: 580.178.04
# Example values only; derive them from your host.
```

580 is the driver branch in this example. The APT candidate has a full version, including the distribution revision. Check it on this host; the branch number alone is not a version match.

Commands

```
DKMS_PKG="nvidia-dkms-${BRANCH}-open"
apt-cache policy "$DKMS_PKG"
apt-cache madison "$DKMS_PKG"
```

Illustrative result

```
# Review the complete candidate and available versions.
# The driver component must match the running $NVVER.
# If no matching version is available, contact Graid Support.
```

Use the full matching APT version below, including any epoch and distribution suffix. For example, a running driver 580.178.04 requires the corresponding 580.178.04 package, not merely any package in branch 580. Refresh APT indexes if necessary. Stop if the matching version or kernel headers are unavailable.

Commands

```
APT_VERSION='replace-with-full-matching-APT-version'
(
  set -eu
  PKG_DRIVER="${APT_VERSION#*:}"
  PKG_DRIVER="${PKG_DRIVER%%-*}"
  test "$PKG_DRIVER" = "$NVVER"
  apt-get install "linux-headers-${KVER}"
  mkdir -p /etc/dkms
  test ! -e /etc/dkms/no-autoinstall
  touch /etc/dkms/no-autoinstall
  trap 'rm -f /etc/dkms/no-autoinstall' EXIT
  apt-get install "${DKMS_PKG}=${APT_VERSION}"
  test -f "/usr/src/nvidia-${NVVER}/dkms.conf"
)
```

Illustrative result

```
# Continue only if the entire block succeeds.
# A version mismatch, missing source or APT failure stops it.
# If the marker already exists, inspect it before retrying.
```

Do not remove the pre-built package unless the preceding block succeeds. Review APT changes before accepting them. A failure requires investigation before the next command or a reboot.

Commands

```
# Confirm the matching DKMS source before removing the old package.
test -f "/usr/src/nvidia-${NVVER}/dkms.conf" && \
apt-get remove "$PREBUILT"
```

Illustrative result

```
# APT removes the selected pre-built module package.
# Review the proposed package removals before accepting.
```

Commands

```
# Build and install the NVIDIA module through DKMS
dkms install \
-m nvidia \
-v "${NVVER}" \
-k "${KVER}"
```

Illustrative result

```
# DKMS builds and installs NVIDIA for the selected kernel.
# A build failure must be resolved before rebooting.
```

Commands

```
# Refresh module dependencies and initramfs, then reboot
depmod -a "${KVER}"
update-initramfs -u -k "${KVER}"
reboot
```

Illustrative result

```
# Module dependencies and initramfs are refreshed.
# The host reboots; reconnect after the system is available.
```

The commands derive the NVIDIA driver version and branch from `nvidia-smi`. Confirm that matching DKMS packages and kernel headers are available for your installed release before replacing the driver. After reboot, check `dkms status` before re-running the SupremeRAID™ AEC installer.

6.3 Creating PD / DG / VD from GP Spark NVMe-oF Devices

SupremeRAID™ AEC organizes storage into four logical components: Physical Drive (PD), Drive Group (DG), Virtual Drive (VD), and Controller (CX). With SupremeRAID™ AEC 2.0, the GP Spark NVMe-oF namespaces are attached directly through the driver's Remote NVMe-oF Target feature using `graidctl`; the driver then exposes them as physical drives. This replaces the earlier approach of connecting the namespaces manually with `nvme-cli`. Complete the connection in Section 6.3.1 before creating physical drives.

6.3.1 Connecting GP Spark as a Remote NVMe-oF Target

SupremeRAID™ AEC connects to the GP Spark over RDMA (RoCEv2) and remembers the target, so the connection is restored automatically on every reboot by the SupremeRAID™ AEC service. No nvme-cli auto-connect service is needed.

Step 1 — Confirm the target is discoverable. Verify the host can discover the GP Spark NVMe-oF target over RDMA (replace the IP and port with your device port address and service ID):

Commands

```
nvme discover --transport=rdma --traddr=10.10.10.12 --trsvcid=4420
```

Illustrative result

```
subtype: nvme subsystem
subnqn: nqn.2026-10.com.example:gpspark-ssd0
traddr: 10.10.10.12
trsvcid: 4420
trtype: rdma
```

Step 2 — Connect the remote target with graidctl. Once discovery succeeds, connect the GP Spark target through SupremeRAID™ AEC:

Commands

```
graidctl connect remote_target rdma 10.10.10.12 ipv4 4420
```

Illustrative result

```
# Connection succeeds; verify the expected target and namespaces
# with graidctl list remote_target and graidctl list nvme_drive.
```

Step 3 — Automatic reconnection after reboot. After the target is connected in Step 2, SupremeRAID™ AEC stores the connection and the SupremeRAID™ AEC service re-establishes it automatically on every subsequent boot — no additional systemd unit or nvme-cli auto-connect configuration is required. As stated in Section 4.1, do not place the NVMe-oF namespaces in /etc/fstab; let the SupremeRAID™ AEC service bring up the targets, then mount the virtual drive(s).

6.3.2 Creating Physical Drives, Drive Groups, and Virtual Drives

With the GP Spark remote target connected (Section 6.3.1), build the RAID components in the SupremeRAID™ AEC Management Console (or with `graidctl`) in the following order.

Step 1 — Create Physical Drives (PD). Log in to the SupremeRAID™ AEC Management Console and navigate to RAID Management / Physical Drives. Click Create and select the connected GP Spark NVMe namespaces to add as physical drives. When an SSD becomes a PD, the SupremeRAID™ AEC driver unbinds it from the OS: the `/dev/nvmeXn1` node disappears and a corresponding `/dev/gpdX` node is created (used for SMART logs and SSD info). SupremeRAID™ AEC supports a maximum of 4 physical drives. A healthy PD that has not yet joined a DG shows the state `UNCONFIGURED_GOOD`.

CLI alternative for Step 1. List the available NVMe drives and replace each path below with a verified namespace for this host. For the four-drive example, use one namespace from each physical SSD. Record the PD IDs returned by creation. Run either the GUI or CLI creation, not both.

Commands

```
sudo graidctl list nvme_drive
sudo graidctl create physical_drive \
/dev/REPLACE_NS0 /dev/REPLACE_NS1 \
/dev/REPLACE_NS2 /dev/REPLACE_NS3
sudo graidctl list physical_drive
```

Illustrative result

```
# Four PDs are created from the selected namespaces.
# Record their actual PD IDs for the drive-group command.
# Unassigned healthy PDs show UNCONFIGURED_GOOD.
```

Step 2 — Create a Drive Group (DG). Navigate to RAID Management / Drive Group and click Create. Select the physical drives and the desired RAID level. SupremeRAID™ AEC supports RAID 0, 1, 5, 6, and 10, with these minimum drive counts:

- RAID 0: at least 1 drive (no redundancy)
- RAID 1: at least 2 drives
- RAID 5: at least 3 drives
- RAID 6: at least 4 drives
- RAID 10: at least 2 drives

Usable capacity. The following estimates assume equal-sized member drives and exclude metadata and filesystem overhead. Redundant RAID levels reserve capacity for protection; RAID 0 does not:

- RAID 0 — full capacity (no redundancy): sum of all drives.
- RAID 1 — half: one drive's worth per mirrored pair.
- RAID 5 — one drive's worth is used for parity: usable = $(N - 1) \times \text{drive size}$.
- RAID 6 — two drives' worth for parity: usable = $(N - 2) \times \text{drive size}$.
- RAID 10 — half the total capacity (striped mirrors).

For example, four 1 TB drives in RAID 5 provide roughly three drives of usable capacity; the remainder protects against a drive failure. Select a RAID level based on usable capacity and protection requirements.

When the DG is created, the driver initializes the physical drives for the chosen RAID mode. If all drives support the deallocate (DSM) command, Fast Initialization is used and the DG becomes OPTIMAL immediately; otherwise Background Initialization runs, during which the DG remains usable. A maximum of 8 drive groups is supported. Note that RAID 5 and RAID 6 can enable the distributed journaling (double-failure protection) feature, and if you bypass journal-space creation it cannot be recreated later.

GP Spark does not support deallocate (DSM) in the configuration described here, so Fast Initialization is unavailable. Create the drive group using the following command:

Commands

```
# Example only: substitute your verified PD IDs and RAID level.
RAID_MODE=raid5
PD_IDS=0,1,2,3
graidctl create drive_group "$RAID_MODE" "$PD_IDS"
graidctl list drive_group
```

Illustrative result

```
# Record the actual DG ID for virtual-drive creation.
# Check its state; background initialization may still be running.
```

Set RAID_MODE to raid0, raid1, raid5, raid6 or raid10 and PD_IDS to the intended physical-drive IDs. Do not include namespaces belonging to another host.

Step 3 — Create a Virtual Drive (VD). Navigate to RAID Management / Virtual Drives, click Create, and select the drive group in which to create the VD. The VD is the usable RAID volume. When created, a device node /dev/gdgXnY appears on the OS. You can create up to 1023 virtual drives per drive group. For RAID 0 and RAID 10, configurable strip sizes are 4k, 8k, 16k, 32k, 64k, and 128k.

CLI alternative for Step 3. Set DG_ID to the drive group created above. Omitting the size creates a VD with the available capacity. Record its actual device path before formatting; /dev/gdg0n1 below is only an example.

Commands

```
DG_ID='replace-with-created-dg-id'
sudo graidctl create virtual_drive "$DG_ID"
sudo graidctl list virtual_drive
```

Illustrative result

```
# A new VD is created in the selected drive group.
# Record its device path and verify its capacity.
# Do not create another VD if the GUI already created it.
```

Step 4 — Make a filesystem and mount (new VD only). Run these commands as root only for a newly created VD whose contents may be erased. Confirm the actual device path before formatting. For an existing or restored VD, preserve its filesystem and skip mkfs; never run this formatting step as part of configuration restoration. The discard mount option has no effect on the GP Spark devices described here because deallocate/trim is not supported.

Commands

```
mkfs.xfs /dev/gdg0n1
mkdir -p /mnt/graidd
mount /dev/gdg0n1 /mnt/graidd
```

Illustrative result

```
# XFS is created on the NEW virtual drive and mounted.
# Existing or restored VDs must not be formatted.
```

Persistent mounting (optional). To mount the virtual drive automatically at boot, add an `/etc/fstab` entry for the **AEC virtual drive** (not the underlying namespace) with options that wait for the network and the SupremeRAID™ AEC service, so the mount is only attempted once the VD exists:

Commands to print one fstab entry

```
VD_DEV=/dev/gdg0n1 # replace with the actual new VD
MNT=/mnt/graidd
OPTS='_netdev,nofail,x-systemd.automount'
OPTS="$OPTS,x-systemd.requires=graidd.service"
printf '%s %s xfs %s 0 0\n' "$VD_DEV" "$MNT" "$OPTS"
```

Illustrative result

```
# One complete fstab record is printed on one logical line.
# Copy the terminal output, not a visually wrapped PDF line.
# This command does not modify /etc/fstab.
```

Add the printed record once with `sudoedit /etc/fstab`; do not duplicate an existing entry. Verify the file before activation. If verification reports an error, correct it before continuing.

Commands

```
sudoedit /etc/fstab
sudo findmnt --verify --verbose
sudo systemctl daemon-reload
```

Illustrative result

```
# Review the fstab verification report.
# Continue only after resolving reported errors.
```

To test automount now, stop applications using this volume (Section 4.2). If it is already mounted from the previous step, unmount it successfully before starting the automount unit. Do not force a busy unmount.

Commands

```
findmnt -rn -M /mnt/graidd
# Run only if the filesystem is still mounted:
sudo umount /mnt/graidd
```

Illustrative result

```
# If no filesystem is mounted, skip umount.
# If umount fails, stop here and release its holders.
```

Commands

```
MNT=/mnt/graid # use the same mount point as the fstab entry
UNIT="$(systemd-escape --path --suffix=automount "$MNT")"
sudo systemctl start "$UNIT"
ls "$MNT"
findmnt -M "$MNT"
df -h "$MNT"
```

Illustrative result

```
# First access mounts the intended AEC virtual drive.
# Verify the device, filesystem and expected capacity.
# Recheck after reboot before restarting applications.
```

With `x-systemd.automount`, first access triggers the filesystem mount; `nofail` allows boot to continue if it is unavailable. The accessing process may still wait for the network, service, or device. `_netdev` marks the mount as network-dependent. Use a stable `/dev/disk/by-id/` path or UUID if `/dev/gdg0n1` is not stable. Add only the SupremeRAID™ AEC virtual drive to `fstab`, not the underlying NVMe-oF namespaces.

With `x-systemd.automount`, the directory may exist before the RAID filesystem is mounted. `df` may not show that filesystem until first access; `mount` or `findmnt` may show an `autofs` entry. Access such as `ls /mnt/graid` triggers the mount. The setup script exposes mount point, filesystem, and `fstab` settings in `gpspark_setup.conf`.

Note: As described in Section 4.1, do not add the underlying NVMe-oF mount entries to `/etc/fstab`. Because the GP Spark namespaces are reached over the network, they are not available at the time `fstab` runs during early boot. The GP Spark remote target must be connected through `graidctl` (Section 6.3.1) and the SupremeRAID™ AEC service must be running before the RAID components and their virtual drives become available.

Step 5 — Verify the RAID is healthy. Confirm the drive group and virtual drive are online and the physical drives are all present:

Commands

```
graidctl list drive_group
graidctl list virtual_drive
graidctl list physical_drive
```

Illustrative result

```
# Example steady-state checks:
DG: OPTIMAL
PDs: ONLINE
VD: online, with expected size
# Check actual fields and states reported by your release.
```

A healthy drive group shows state `OPTIMAL`; right after creation it may briefly show `RESYNC` or `INITIALIZING` while background initialization runs — this is normal and the volume is usable meanwhile. Each physical drive should show `ONLINE`. If a drive group stays in `RESYNC` or a virtual drive does not come online, see Section 8.2.

You can also confirm the OS sees the virtual drive and its filesystem:

Commands

```
lsblk /dev/gdg0n1  
df -h /mnt/raid # after first access when using automount
```

Illustrative result

```
gdg0n1 ... disk /mnt/raid  
/dev/gdg0n1 ... /mnt/raid  
# Capacity must match your RAID plan and filesystem overhead.
```

6.4 Upgrading SupremeRAID™ AEC

Before you upgrade: release the virtual drives. Stop I/O and applications, stop any configured automount units, unmount the affected filesystems, and verify no holders remain, following Section 4.2. Start the upgrade only after those checks pass.

Upgrade the SupremeRAID™ AEC software. In the normal case you only re-run the installer for the new version:

Commands

```
INSTALLER='/absolute/path/to/approved-new-installer.run'  
sudo "$INSTALLER" --accept-license
```

Illustrative result

```
# The approved upgrade completes.  
# Confirm service state and the existing PD/DG/VD configuration.
```

An upgrade of the SupremeRAID™ AEC software alone does **not** affect existing drive groups or virtual drives — your data and RAID configuration are preserved.

If the NVIDIA driver was upgraded, the SupremeRAID™ AEC kernel module must be rebuilt against the new driver, otherwise the graid driver fails to load. Follow the Graid knowledge-base article:

<https://support.graidtech.com/portal/en/kb/articles/linux-how-to-resolve-graid-driver-failure-after-nvidia-upgrade>

After the module is rebuilt, confirm the service is healthy per Section 8.1.

6.5 Restoring the SupremeRAID™ AEC Configuration

Use this procedure to restore /etc/graid.conf from configuration metadata stored on a physical drive. Run the commands as root. Before stopping RAID services, stop I/O and applications, stop configured automount units, unmount the affected filesystems, and verify no holders remain (Section 4.2).

Restoring configuration may cause data inconsistency or data loss. Verify your backup and review the selected configuration before confirming the operation.

Restore using gpspark_setup.py

Use the latest gpspark_setup.py and gpspark_setup.conf from the setup package (Section 6.1). Run this standalone recovery mode in an interactive terminal. It does not run the installation stages or change saved installation progress; [raid] create_raid does not need to be enabled.

Commands

```
sudo python3 gpspark_setup.py --restore-config
```

Illustrative interactive prompts

```
Target IP/address [10.10.10.12]:  
Target port [4420]:  
Transport type (rdma/tcp/fc) [rdma]:
```

To use a setup configuration file at another location:

Commands

```
sudo python3 gpspark_setup.py --restore-config \  
--config /path/to/gpspark_setup.conf
```

Illustrative result

```
# Defaults are loaded from the specified configuration file.  
# Restoration prompts and confirmations remain interactive.
```

1. Enter the remote target IP/address, port, and transport type when prompted. The displayed defaults come from [raid] nvme_address, nvme_svid, and nvme_transport. Press Enter to accept each default, or enter a replacement. Changes apply only to this restoration and are not written to the setup configuration file. The illustrative prompts above show the supplied defaults.
2. Review the target and confirm that I/O has stopped and your data backup has been verified. The script refuses to continue while GRAID volumes are mounted.
3. The script automatically backs up the existing /etc/raid.conf before stopping either service. It prints the backup path, using the format /etc/raid.conf.backup-<timestamp>-<unique suffix>, with permissions 0600. If backup fails, the script stops before stopping services or restoring configuration. If /etc/raid.conf does not exist, the script reports that no current configuration can be backed up and continues. This configuration backup does not replace a backup of your data.
4. The script stops raid and raid-mgr, runs nvme discover and nvme connect-all with the entered target values, and starts raidctl restore config interactively.
5. Select the intended configuration by index. Review its UUID, epoch, timestamp, physical drives, and drive groups, then personally enter the case-sensitive phrase Confirm Unsafe Operation. The script does not select a configuration or enter this confirmation for you. See the manual procedure below for an example of the prompts and success message.
6. Confirm the successful restore message and review the restored configuration. The script then offers a separate confirmation to restart raid/raid-mgr and query PD/DG/VD state. Accept only when ready. If declined, services remain stopped. Do not repeat the alternative manual restoration procedure.

Illustrative post-restore choice

```
Restart raid/raid-mgr and check services and RAID now?  
# Yes: one restart attempt, followed by service and RAID queries.  
# No: services remain stopped for configuration review.
```

If restoration fails, retain the printed backup path and terminal output. Send /etc/raid.conf (if available), the restore output, and /var/lib/gpspark-setup/setup.log to Graid Support. Interactive raidctl output is not

saved in the setup log. Known restoration issues may require Support to update the configuration before the system can operate correctly.

Manual restoration (alternative to the script)

If you used `--restore-config` above, do not repeat these manual steps. When restoring manually, back up the current `/etc/raid.conf`, if present, before stopping the services.

1. Stop the SupremeRAID™ AEC services:

Commands

```
systemctl stop raid
systemctl stop raid-mgr
```

Illustrative result

```
# Successful stop is normally silent.
# Both RAID services are stopped before recovery connections.
```

2. Discover the remote NVMe-oF devices:

Commands

```
TARGET_IP=10.10.10.12
TARGET_PORT=4420
TARGET_TRANSPORT=rdma
nvme discover -s "$TARGET_PORT" -t "$TARGET_TRANSPORT" \
-a "$TARGET_IP"
```

Illustrative result

```
subtype: nvme subsystem
subnqn: nqn.2026-10.com.example:gpspark-ssd0
traddr: 10.10.10.12
trsvcid: 4420
trtype: rdma
```

Use the actual target address, service port and transport. Keep these same values for the following connection command.

3. Connect the remote devices:

Commands

```
nvme connect-all -s "$TARGET_PORT" -t "$TARGET_TRANSPORT" \
-a "$TARGET_IP"
```

Illustrative result

```
# Storage controllers connect and their namespaces appear.
# This does not mount a filesystem or restore RAID configuration.
```

4. Start configuration restoration:

Commands

```
graidctl restore config
```

Illustrative interactive output

```
Found the following configs:
[0]: Device /dev/nvme2n1 ... UUID ... Epoch ...
Please select one config to restore (0-3):
```

The command scans devices for stored configurations and lists available configurations by index, including their device, UUID, epoch, and timestamp. Devices without a stored configuration are skipped.

5. Select the configuration to restore by entering its index. You may select any listed device containing the intended configuration; review the UUID, epoch, timestamp, physical drive list, and drive group list before proceeding. At the confirmation prompt, type exactly:

Interactive input — type only at the restore confirmation prompt

```
Confirm Unsafe Operation
```

This phrase is case sensitive. In the supplied example, all four devices contain the same configuration UUID and epoch, and index 0 is selected:

Illustrative restore transcript

```
Please select one config to restore (0-3): 0
Timestamp: 2026-09-27 10:05:26 +0800 CST
Epoch: 45966
Config UUID: 215991f4-ec5e-3732-a8a4-821f54acd334
Confirmation: Confirm Unsafe Operation
Restore config graid.conf successfully.
```

6. Confirm that ✓Restore config graid.conf successfully. appears. This confirms restoration of /etc/graid.conf, not service or RAID health. Review the restored configuration, then follow the post-restore checks below.

Post-restore service and RAID checks (both workflows)

Use these manual service-start commands if services remain stopped, after reviewing the restored configuration and resolving any issues. If the script already restarted services, perform the status and RAID-list checks only. Complete any required Graid Support configuration update first; stop and inspect logs if a service fails.

Commands

```
systemctl start graid
systemctl start graid-mgr
systemctl status graid.service graid-mgr.service
graidctl list physical_drive
graidctl list drive_group
graidctl list virtual_drive
```

Illustrative result

```
graid.service: active
graid-mgr.service: active
# Verify expected PD/DG/VD counts and healthy states.
# Restore success alone does not prove RAID health.
```

Confirm that the expected RAID objects are present and healthy. Section 8.1 covers service failures; use only the health-check guidance in Section 6.3.2, not its creation or formatting steps. Once the restored objects are healthy, re-enable any intentionally stopped automount units, or mount the existing filesystem manually. Verify the mounted device and capacity before restarting applications. Do not run mkfs on a restored VD.

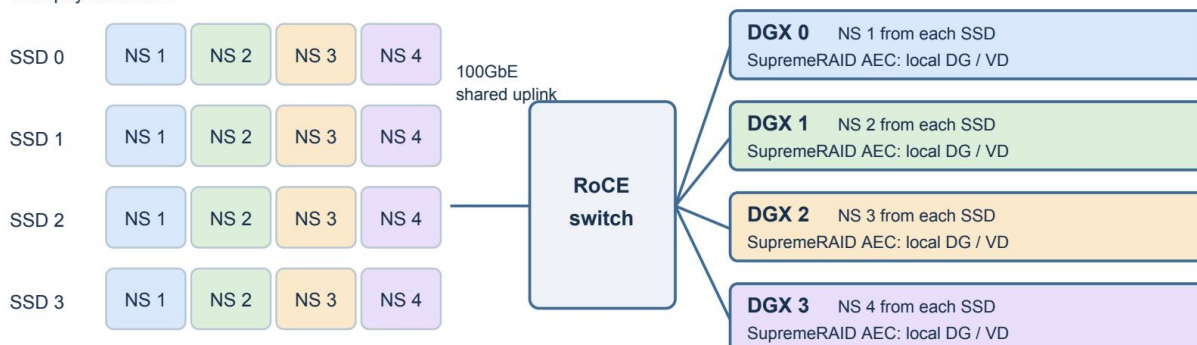
6.6 Sharing GP Spark Across Multiple DGX Spark Hosts

A GP Spark JBOF can divide SSD capacity into separate NVMe namespaces for allocation to multiple DGX Spark hosts. Each host uses its assigned namespaces to build its own SupremeRAID™ AEC drive group and virtual drive. This provides separate storage capacity to each host; it does not provide a shared filesystem or concurrent access to the same VD.

Four DGX hosts are used only to illustrate the allocation; four is not a stated system maximum. Additional hosts require sufficient SSD namespace slots and capacity, supported JBOF host assignment, and validation of the complete configuration. Each DGX running AEC needs its own valid AEC license. Follow the deployment order below before using the manual or script procedure.

GP Spark JBOF

Four physical SSDs



Example allocation only. Actual NSIDs and supported host count must be verified.

Figure 6.6-00 Example of exclusive namespace allocation across four DGX hosts

Host	One namespace from each SSD	AEC storage on that host
DGX 0	NS 1 on SSD 0, 1, 2 and 3	4 PDs → local DG / VD
DGX 1	NS 2 on SSD 0, 1, 2 and 3	4 PDs → local DG / VD
DGX 2	NS 3 on SSD 0, 1, 2 and 3	4 PDs → local DG / VD
DGX 3	NS 4 on SSD 0, 1, 2 and 3	4 PDs → local DG / VD

NSIDs are scoped to each NVMe subsystem. Record the actual IDs returned by provisioning; the labels above are illustrative. A namespace is a slice of an SSD, while a VD is created by AEC from its PDs. Each host may have its own VD 0.

Use one namespace from each of four different physical SSDs for each host's four-PD RAID set. Multiple namespaces on the same SSD share one failure domain. The JBOF 100GbE link (4 × 25G NRZ) is shared by all hosts; it does not provide a dedicated 100Gb/s uplink to each host.

Multi-host deployment order

Use this order for a new deployment. Namespace slicing and host assignment must be complete before any consumer connects this JBOF through `graidctl connect remote_target` or creates PDs, DGs or VDs. Run `nvme discover` and `nvme connect` on the administration DGX to manage the remote namespaces; these are different from the AEC-managed connection.

Step and operator	Action and completion check
1. Every DGX	Install AEC using 6.1 with <code>create_raid=false</code> , or the manual installation in 6.2. Complete the required license activation and GPU binding on each host. Keep this JBOF unconnected through AEC and create no RAID objects.
2. Administration DGX	Complete the management connection in 6.6.1, then save and apply one batch plan for the selected SSDs in 6.6.2. The manual alternative is in 6.6.4; do not run both methods on the same allocation.
3. JBOF administrator	Complete the allocation, JBOF host assignment and connection handoff in 6.6.3. Use its result checklist before any consumer creates PDs. Contact Support with the allocation record if assignment needs assistance.
4. Every consumer DGX	Connect with <code>graidctl connect remote_target</code> , verify the assigned namespaces, then build and check its own RAID using 6.3.2. Do not rerun the installer to create RAID after installation has completed.

Terms used in the following steps

Namespace: a capacity slice on one SSD. PD: a namespace claimed by AEC as a RAID member. DG: the RAID group built from selected PDs. VD: the RAID volume presented to the OS. Subsystem NQN identifies the NVMe subsystem; NSID identifies a namespace within it. Record these together with SSD serial, namespace UUID and capacity.

For the four-SSD example, each host receives one slice from each SSD. With H hosts, plan H slices on each SSD: $4 \times H$ slices across the JBOF, while each host still uses four AEC PDs. This arithmetic does not establish a supported host limit. Check available capacity, namespace/controller limits, host assignment and aggregate load before increasing H .

Only the administration host performs slicing. Consumer hosts do not repeat apply or copy its journal. Existing deployments with AEC connections or RAID objects require a coordinated maintenance plan; do not delete RAID objects just to follow this new-deployment example.

6.6.1 Prepare the allocation and management connection

Designate one DGX as the namespace administration host, for example DGX 0. Complete slicing on every SSD before any consumer uses `graidctl connect remote_target` for this JBOF or creates PDs, DGs or VDs. Use the `nvme-cli` management connection below for slicing. Do not change namespaces concurrently from other hosts.

Coordinate all hosts before changing namespaces

Plan all four physical SSDs and record the intended owner of every slice. Stop I/O on every initiator using an affected SSD, release affected volumes as in Section 4.2, and coordinate any controller detachments. The script checks local use; it cannot establish that every remote host has stopped I/O.

Role	Responsibility
DGX 0 as administration host	Discover and connect the SSDs; run list, plan and apply (or the manual commands); preserve journals and the allocation record.
Every consumer DGX (DGX 0–3 in this example)	After handoff, connect through AEC and use only assigned namespaces. Each host builds and mounts its own RAID virtual drive.
JBOF administrator	Configure the firmware-supported host access and namespace visibility rules, and verify isolation before RAID creation.

Separate provisioning from host assignment

Namespace creation divides SSD capacity. It does not assign ownership to a DGX, establish access isolation, or create an AEC RAID. A namespace attach operation is a controller association, not a permanent host allocation. Complete the handoff in Section 6.6.3 before allowing consumer I/O.

DGX 0 may later use its own assigned slices, but finish namespace provisioning first. Once it has AEC PDs or DGs, the current script blocks live namespace apply. Do not remove an active RAID merely to bypass that check; coordinate subsequent changes with Graid Support.

Gray blocks contain commands or configuration text. Blue blocks show illustrative results. Replace example identities with values from your hardware. Only the administration host performs namespace provisioning.

1. Discover the GP Spark SSD subsystems

Complete the network and RDMA preparation in Chapter 3. Use the actual storage address; the example is 10.10.10.12:4420 over RDMA. Select a storage subsystem record, not the discovery subsystem.

Commands

```
TARGET_IP=10.10.10.12
TARGET_PORT=4420
sudo nvme discover -t rdma -a "$TARGET_IP" -s "$TARGET_PORT"
```

Illustrative result — Discover storage subsystems

```
subtype: nvme subsystem
subnqn: nqn.2026-10.com.example:gpspark-ssd0
traddr: 10.10.10.12 trsvcid: 4420 trtype: rdma
```

2. Connect all four SSDs for namespace administration

Connect each SSD before planning. Replace the four values below with actual discovery subnqn values. Verify each NQN and controller-reported serial maps to the intended SSD; firmware determines this mapping. Reuse suitable existing connections. Never select the local boot SSD.

Commands

```
SSD_NQNS=(
  'replace-with-ssd0-nqn' 'replace-with-ssd1-nqn'
  'replace-with-ssd2-nqn' 'replace-with-ssd3-nqn'
)
for SSD_NQN in "${SSD_NQNS[@]}"; do
  sudo nvme connect -t rdma -a "$TARGET_IP" -s "$TARGET_PORT" \
    -n "$SSD_NQN" --nr-io-queues=8 || break
done
nvme list-subsys
nvme list
```

Illustrative result — Verify the management connection

```
# Abbreviated example: four connected storage subsystems
NQN=...:gpspark-ssd0 nvme3 rdma 10.10.10.12:4420 live
NQN=...:gpspark-ssd1 nvme4 rdma 10.10.10.12:4420 live
NQN=...:gpspark-ssd2 nvme5 rdma 10.10.10.12:4420 live
NQN=...:gpspark-ssd3 nvme6 rdma 10.10.10.12:4420 live
```

Stop on any connection failure. Use --host-traddr when routing requires it and authorize the actual Host NQN. Check all four SSDs are connected before 6.6.2; if one is missing from the selection menu, return here. Device numbers can change. Manual slicing is in 6.6.4.

6.6.2 Provision namespaces with a saved plan

Choose either this script batch workflow or the manual workflow in 6.6.4. Do not run both on the same SSDs. Run on the administration DGX after connecting all SSDs in 6.6.1. Use matching script/config files. Batch mode applies shared sizes; it does not assign hosts or create RAID.

1. Set the target and namespace sizes

Set the target in [raid], keep create_raid=false, and leave namespace_manifest empty. Batch mode selects SSDs by number and ignores nqn/ssd_serial config defaults. The four sizes below apply to each selected SSD: four SSDs create 16 new slices, each 100 GiB.

Configuration text

```
[raid]
create_raid = false
namespace_manifest =
nvmf_address = 10.10.10.12
nvmf_svcid = 4420
nvmf_transport = rdma

[namespace]
nqn =
ssd_serial =
sizes_gib = 100,100,100,100
lba_bytes = 512
```

Configuration text: edit these fields in the existing file. Use --config /absolute/path/gpspark_setup.conf if it is not beside the script. The namespace settings do not affect normal installation.

2. Select four SSDs, save and review the batch plan

Commands

```
sudo python3 gpspark_setup.py --namespace-plan --namespace-batch \
--plan-file batch-plan.json
```

Illustrative result

```
Connected SSDs for batch selection:
[1] GP Spark SSD | GP-EXAMPLE-001 | ...:ssd0 | namespaces: 1
[2] GP Spark SSD | GP-EXAMPLE-002 | ...:ssd1 | namespaces: 1
[3] GP Spark SSD | GP-EXAMPLE-003 | ...:ssd2 | namespaces: 1
[4] GP Spark SSD | GP-EXAMPLE-004 | ...:ssd3 | namespaces: 1
Select SSD numbers (e.g. 1,2,3,4), or q to cancel: 1,2,3,4
SSD 1: GP-EXAMPLE-001 | retain [1] | create GiB 100,100,100,100 | pending
SSD 2: GP-EXAMPLE-002 | retain [1] | create GiB 100,100,100,100 | pending
SSD 3: GP-EXAMPLE-003 | retain [1] | create GiB 100,100,100,100 | pending
SSD 4: GP-EXAMPLE-004 | retain [1] | create GiB 100,100,100,100 | pending
[PLAN ONLY] Batch plan: /home/demo/batch-plan.json; blockers: 0.
No namespace changes.
# Model and NQN values are abbreviated for illustration.
```

Each SSD needs at least 400 GiB of unallocated capacity and four free namespace slots. With one retained namespace, the final count is five per SSD. Any blocked SSD blocks the whole batch before changes. Existing namespaces are retained; batch deletion is not supported. Use 6.6.5 for single-SSD replacement.

Enter 1,2,3,4 after checking the actual model, serial and NQN for all four SSDs; q cancels. Review all retained namespaces and sizes. Use a new plan filename and do not edit the JSON. Plan changes no namespaces. For a single SSD, omit --namespace-batch and select one SSD.

3. Apply the saved plan and record the result

Stop I/O on all affected hosts. Run --namespace-apply to execute the saved plan on real SSDs. After all checks pass, review the SSD list and type the displayed confirmation phrase. Resume also requires confirmation. Plan, --dry-run and --namespace-simulate do not change real namespaces. No additional enable flag is required.

Commands

```
sudo python3 gpspark_setup.py --namespace-apply --namespace-batch \  
--plan-file batch-plan.json
```

Illustrative result

```
Type "APPLY BATCH 4 SSDS" to continue: APPLY BATCH 4 SSDS  
[PASS] SSD 1 complete: GP-EXAMPLE-001  
[PASS] SSD 2 complete: GP-EXAMPLE-002  
[PASS] SSD 3 complete: GP-EXAMPLE-003  
[PASS] SSD 4 complete: GP-EXAMPLE-004  
[PASS] Batch complete. Journals:  
/home/demo/batch-plan.json.journals
```

Apply creates and attaches the namespaces, then verifies the result. It rechecks identity, state, capacity, limits and local usage. It cannot prove that other hosts have stopped I/O. Simulation validates script behavior, not complete GP Spark firmware compatibility; contact Support if an operation fails.

Record each SSD's created NSIDs, UUIDs and capacities from its journal. Keep the batch plan and its .journals directory together on the administration DGX. After all SSDs complete, continue with host assignment in 6.6.3.

Batch operations are sequential and are not rolled back. If SSD 3 fails, SSDs 1 and 2 remain changed. Keep the same plan and all journals. Resume rechecks completed SSDs and continues unfinished work. An uncertain command result blocks automatic resume; contact Support. Never delete journals to bypass a failure.

Commands

```
sudo python3 gpspark_setup.py --namespace-apply --namespace-batch \  
--plan-file batch-plan.json --namespace-resume
```

Illustrative result

```
# Completed SSDs are rechecked without repeating their changes.  
# After confirmation, unfinished SSDs continue.  
[PASS] Batch complete. Journals: /home/demo/batch-plan.json.journals
```

6.6.3 Assign namespaces and connect each DGX

1. Prepare one allocation record for each DGX

Use the record at the end of this section. For each DGX, list one namespace from each SSD with its subsystem NQN, NSID, UUID and capacity. Confirm its AEC license and GPU binding. Do not identify a namespace by its Linux device name alone.

2. Apply the JBOF host assignment and release the management connection

Use the GP Spark firmware procedure to assign each recorded namespace to its intended host and complete the administration-to-consumer connection handoff. Use the actual GRAID Host NQN for AEC access. AEC can use a different identity from `/etc/nvme/hostnqn`; do not substitute that file without checking.

If the firmware procedure is not available, send the allocation record, GP Spark firmware version and administration inventory to Graid Support to obtain the required assignment and handoff steps. Ask which controller attachments to release and how each host retains its assignment after reconnect. Do not delete namespaces or reuse the administration controller ID to transfer ownership.

Commands

```
# Run on the administration host; retain this output.
sudo python3 gpspark_setup.py --namespace-list
nvme list-subsys
```

Illustrative result

```
# Save the SSD identities and current controller connections.
# Match the record to the JBOF host-assignment settings.
# Preserve completed apply journals on this host.
```

3. Connect one consumer DGX and compare its inventory. After assignment and handoff, run the following on that DGX before creating any PDs. Use the actual target address. Inspect the Host NQN under the target in `list-subsys` and compare every namespace with this DGX allocation record.

Commands on each consumer DGX — use the actual target IP and port

```
sudo graidctl connect remote_target rdma 10.10.10.12 ipv4 4420
sudo graidctl list nvme_drive
nvme list-subsys
nvme list
```

Illustrative ownership checklist — verify with real inventory

```
# Operator worksheet, checked against actual inventory:
SSD 0 -> assigned NSID / UUID -> 100 GiB -> DGX 0 only
SSD 1 -> assigned NSID / UUID -> 100 GiB -> DGX 0 only
SSD 2 -> assigned NSID / UUID -> 100 GiB -> DGX 0 only
SSD 3 -> assigned NSID / UUID -> 100 GiB -> DGX 0 only
```

Observed result	Next action
The assigned namespaces are present and access rules match the allocation.	Continue to Step 4 using only the recorded namespaces.
No namespaces, missing slices or a connection error.	Check the target address, route, GRAID Host NQN authorization and attachments. Send the inventory to Support if unresolved.
Unexpected slices, duplicate identities or unclear ownership.	Do not create PDs. Correct host assignment and confirm access isolation with Support. Do not delete another host's namespaces.

4. Build and verify this host RAID, then repeat on the next DGX

Follow the complete PD → DG → VD procedure in 6.3.2 using this host's four recorded namespaces, one from each physical SSD. Verify the filesystem and capacity before moving to the next DGX. Never select all discovered slices. Namespaces on one SSD share the same failure domain.

For this multi-host workflow, installation was completed with `create_raid=false`. Build the RAID using Section 6.3.2; changing `create_raid` does not replay a completed `CREATE_RAID` stage. If an approved deployment instead uses automated RAID creation on an initial setup run, select only that host's verified `nvme_devices` and matching `pd_ids`, and leave `namespace_manifest` empty. Device paths can differ across hosts and change after reconnect.

Commands

```
sudo graidctl list physical_drive
sudo graidctl list drive_group
sudo graidctl list virtual_drive
```

Illustrative result — RAID acceptance checks on each host

```
# Operator checklist after creating the assigned PD / DG / VD:
PDs: four expected namespaces, each on a different SSD
DG: expected RAID level and healthy state
VD: online with capacity matching RAID level and overhead
Filesystem: format only a new VD owned by this host
```

Allocation record for each consumer DGX

Complete one record per DGX from actual device inventory before creating PDs. Retain all records centrally with the administration host's journals. Use full identifiers; short example labels are not sufficient to identify a namespace.

Host field	Recorded value
DGX hostname / management address	<actual host and address>
GRAID Host NQN used for target access	<full actual Host NQN>
JBOF target and firmware	<storage IP, port, transport and firmware version>
Access rule verification	<rule reference, operator, date and isolation-check result>

Assigned namespaces on this DGX

SSD serial / subsystem NQN	NSID	Namespace UUID	Capacity	Local device path
<SSD 0 serial> <full NQN>	<ID>	<full UUID>	<GiB>	<verified path>
<SSD 1 serial> <full NQN>	<ID>	<full UUID>	<GiB>	<verified path>
<SSD 2 serial> <full NQN>	<ID>	<full UUID>	<GiB>	<verified path>
<SSD 3 serial> <full NQN>	<ID>	<full UUID>	<GiB>	<verified path>

Record namespace identities before AEC claims them as PDs; the original NVMe device paths may disappear afterward. Use one namespace from each of four distinct physical SSDs. NSIDs are scoped to a subsystem and cannot identify a slice by themselves.

Acceptance before application I/O

Check each DGX independently: only its assigned namespaces are selected as PDs; its DG and VD are healthy; its own filesystem is mounted with the expected usable capacity. Verify the supported access controls prevent unintended cross-host access. Recheck identity, assignment and service recovery after reconnect and reboot. A successful create or attach message alone does not prove isolation.

A four-member RAID 5 with 100 GiB slices provides approximately 300 GiB before overhead. Test read/write integrity on each host and measure aggregate throughput over the shared 100GbE link.

6.6.4 Advanced manual namespace provisioning

Choose either this vendor-derived manual workflow or the script batch workflow in 6.6.2. Do not run both on the same SSDs. Run these commands on the administration DGX. Create four 100 GiB slices per SSD. The SSD and GP Spark firmware must support namespace management and attachment. Stop after any failed command.

1. Identify the remote controller and namespace format

Replace `/dev/nvmeX` with the remote controller verified in 6.6.1. Record the serial, cntlid, OACS namespace-management capability, available capacity, allocated NSIDs and supported LBA formats. Do not infer the Controller ID from the Linux device number.

Commands

```
CTRL=/dev/nvmeX
sudo nvme id-ctrl "$CTRL" -H
sudo nvme list-ns "$CTRL" --all
OLD_NSID=1 # replace with an actual allocated NSID
sudo nvme id-ns "$CTRL" -n "$OLD_NSID" --force -H
```

Illustrative result — Identify before changing namespaces

```
sn: GP-EXAMPLE-001   cntlid: 7
Namespace Management and Attachment Supported
id-ns: lba: 0: ms:0 lba:9 (512-byte data, no metadata)
```

The example requires a verified format index 0 with 512-byte data and zero metadata. Record it before deletion. If that format or management capability is unavailable, stop; do not run nvme format or reset as a workaround. Neither is a routine prerequisite for this create/attach workflow.

2. Confirm controller attachment and calculate capacity

Set CNTLID to the actual cntlid. Inspect every namespace and all attached controllers. Stop other initiators before making changes; the manual CLI does not perform the script safety checks. Four 100 GiB slices require at least 400 GiB plus any firmware allocation overhead.

Commands

```
CNTLID=7 # example only; use id-ctrl output
LBAF=0 # verified metadata-free 512-byte format
sudo nvme list-ctrl "$CTRL" -n "$OLD_NSID"
LBA_BYTES=512
BLOCKS=$((100 * 1024 * 1024 * 1024 / LBA_BYTES))
printf "Blocks per 100 GiB namespace: %s\n" "$BLOCKS"
```

Illustrative result — Review attachments and capacity

```
num of ctrls present: 1
[0]: 0x7
Blocks per 100 GiB namespace: 209715200
```

3. Detach and delete only an approved old namespace if needed

Skip this step when sufficient unallocated capacity and namespace slots already exist. Deletion destroys access to the selected data. Confirm the SSD serial and backup, then replace only explicitly reviewed NSIDs. If other controllers still have an attachment, coordinate their release first. Never use the delete-all NSID.

Commands

```
sudo nvme detach-ns "$CTRL" -n "$OLD_NSID" -c "$CNTLID"  
# Continue only after successful detach and identity recheck.  
sudo nvme delete-ns "$CTRL" -n "$OLD_NSID"  
sudo nvme list-ns "$CTRL" --all
```

Illustrative result — Remove an explicitly selected old namespace

```
detach-ns: Success, nsid:1  
delete-ns: Success, deleted nsid:1
```

After detach, `/dev/nvmeXnY` may disappear. Continue using the controller path, such as `/dev/nvme3`. If detach returns an invalid controller list, re-read `cntlid` and `list-ctrl`; do not guess a different ID.

4. Create four 100 GiB namespaces

Run the following command once per new slice, four times in total, checking each result before continuing. NSZE and NCAP are logical block counts: $209715200 \times 512 = 107374182400$ bytes = 100 GiB. Record every returned NSID; IDs need not be consecutive or match the illustration.

Commands

```
sudo nvme create-ns "$CTRL" --nsze="$BLOCKS" \  
--ncap="$BLOCKS" --flbas="$LBAF" --dps=0 --nmic=0
```

Illustrative result — Create and record each returned NSID

```
# Four separate successful create operations, example results:  
create-ns: Success, created nsid:1  
create-ns: Success, created nsid:2  
create-ns: Success, created nsid:3  
create-ns: Success, created nsid:4
```

The namespace is not yet usable through the controller until attached. `--nmic=0` follows the vendor single-controller example; it is not a host access policy. Do not enable shared-namespace writes or treat namespace creation as a filesystem operation.

5. Attach each returned NSID to the verified controller

Use each ID returned in Step 4. Repeat the attach command once per ID. A successful attach associates a namespace with this controller; it does not assign it permanently to a DGX host or configure JBOF access rules.

Commands

```
NEW_NSID=1 # replace with one returned NSID; repeat for each  
sudo nvme attach-ns "$CTRL" -n "$NEW_NSID" -c "$CNTLID"  
sudo nvme ns-rescan "$CTRL"  
nvme list
```

Illustrative result — Attach and rescan the new slices

```
attach-ns: Success, nsid:1  
# Repeat with each actual returned NSID.  
# After rescan, verify each namespace in nvme list.
```

6. Verify capacity and save the namespace identities

Verify all four slices with `nvme list` and `id-ns`. Record namespace identity descriptors with `id-ns-descs` where supported. Use actual device paths; Linux names may change after reconnect. The byte count below is expected for each 100 GiB slice. NVMe list may show approximately 107.37 GB because it uses decimal units.

Commands

```
sudo nvme id-ns "$CTRL" -n "$NEW_NSID" -H
sudo nvme id-ns-descs "$CTRL" -n "$NEW_NSID"
NSDEV=/dev/nvmeXnY # replace with the verified namespace path
sudo blockdev --getsize64 "$NSDEV"
sudo nvme read "$NSDEV" -s 0 -c 0 -z 512 --data=/dev/null
```

Illustrative result — Verify one 100 GiB namespace

```
107374182400
read: Success
```

The read command checks one 512-byte logical block only; it is not a data-integrity or performance qualification. Do not format the raw namespace when it will become an AEC PD. Repeat this workflow on the remaining SSDs, then continue at Section 6.6.3. Do not run the alternative script apply on these already provisioned slices.

Reference: Luisuan Technology namespace dynamic slicing support, version A.0, Sections 1–2. The examples here retain individual namespace operations while adding explicit identity, attachment and capacity checks. Command spelling follows `nvme-cli 2.x`; check the installed command help if your version differs.

6.6.5 Advanced replacement and recovery

Replace an existing namespace only when required

Use this procedure only when the SSD has insufficient unallocated space and replacement is intended.

Deletion destroys the selected data. Verify the SSD identity, all initiator activity and backups. Select the same SSD used in 6.6.2; provide exact reviewed old NSIDs explicitly.

Commands

```
REPLACE_NSIDS=1 # example only; replace with reviewed old NSIDs
sudo python3 gpspark_setup.py --namespace-plan \
--delete-nsids "$REPLACE_NSIDS" --plan-file ssd0-replacement.json
```

Illustrative result

```
Delete NSIDs: 1
Create GiB: 100, 100, 100, 100
LBA bytes: 512 | Blockers: 0
# Review the complete plan before apply.
```

Replacement is a single-SSD operation. Apply ssd0-replacement.json with --namespace-apply --plan-file ssd0-replacement.json; omit --namespace-batch. Never use delete-all. Resolve blockers or contact Support before applying.

Check a saved plan without applying it

Commands

```
sudo python3 gpspark_setup.py --namespace-apply --namespace-batch \
--plan-file batch-plan.json --dry-run
```

Illustrative result

```
[PLAN ONLY] No device changes.
# Target state must still match the saved plan.
```

This reads and revalidates the plan without creating a journal or changing devices. If the state changed before any apply, inspect the current inventory and create a new plan with a new filename. After a partial apply, use the existing journal recovery procedure below.

Namespace count limits

Plan uses the SSD subsystem's nonzero MNAN value, or NN when MNAN is zero. It also checks a reported nonzero MAXCNA attachment limit. Unknown or invalid NN/MNAN values block apply; do not edit the plan or guess a limit. Retaining one namespace and requesting six requires seven slots. The script also limits the final inventory to 1023 namespaces. Capacity and firmware constraints still apply. Send blocked plans and id-ctrl output to Graid Support when unclear.

After updating the script, regenerate unused older plans and review them again. For a partially completed job, preserve the original journal and use recovery below instead of starting a new apply.

Single-SSD journal ownership and retention

For single-SSD jobs, run on the administration DGX. `--namespace-journal` specifies the local JSON checkpoint and result file for one SSD provisioning job. It records the target identity, plan, pending operation, completed progress and verified results. It is an operation record, not a backup of SSD data or a host access policy. For batch recovery, use the batch plan and `--namespace-resume` command in 6.6.2.

SSD being provisioned	Example journal on DGX 0
SSD 0	/home/demo/ssd0-plan.json.journal.json
SSD 1	/home/demo/ssd1-plan.json.journal.json
SSD 2	/home/demo/ssd2-plan.json.journal.json
SSD 3	/home/demo/ssd3-plan.json.journal.json

With `--plan-file`, the default journal is the absolute plan path plus `.journal.json`. For example, `/home/demo/ssd0-plan.json` uses `/home/demo/ssd0-plan.json.journal.json`. An explicit `--namespace-journal` overrides this path. Retain the printed path. Each new SSD job needs a new plan and journal; an interrupted job must use its original journal.

Inspect the saved journal on the administration DGX

Commands

```
sudo python3 -m json.tool /home/demo/ssd0-plan.json.journal.json
```

Illustrative result excerpt after a verified four-slice job

```
"simulated": false,
"pending": null,
"created": {"0": 1, "1": 2, "2": 3, "3": 4},
"complete": true
```

Actual IDs and fields must be checked in the complete file. Keep each journal with the allocation record and terminal output. Consumer DGXs do not need to copy it or replay apply: they connect and use their assigned namespaces. Do not use an SSD journal as a per-host `namespace_manifest`.

If interrupted, resume on the original administration host using the same SSD identity and journal (6.6.5). Resume verifies the current state against the saved checkpoint. If a command outcome is uncertain, automatic replay is blocked; reconcile the state before continuing. The journal cannot undo deletions or recover data.

1. Preserve the journal and inspect the current state

Keep the journal, terminal output and namespace inventory. An uncertain command result blocks automatic replay; contact Graid Support to reconcile it. Do not delete the journal or repeat create/delete commands blindly. A journal cannot recover deleted data.

Illustrative result — An uncertain operation requires reconciliation

```
[ERR] Previous command outcome is uncertain.  
Inspect namespaces and journal; automatic replay is blocked  
(especially create).  
# Preserve the journal. Do not repeat the create command.
```

2. Resume only a verified checkpoint

Run on the original administration DGX. Read the journal and allocation record. Set the original SSD identity below and verify the original target in `gpspark_setup.conf`, including any required host/source filters. Do not pass `--plan-file`, new sizes or deletion options during resume. A new login shell does not retain the variables. Uncertain results and state mismatches block resume.

Commands

```
JOURNAL=/home/demo/ssd0-plan.json.journal.json  
sudo python3 -m json.tool "$JOURNAL"  
SSD_NQN='replace-with-original-subsystem-nqn'  
SSD_SERIAL='replace-with-original-ssd-serial'  
sudo python3 gpspark_setup.py --namespace-apply \  
--namespace-resume \  
--namespace-nqn "$SSD_NQN" --ssd-serial "$SSD_SERIAL" \  
--namespace-journal "$JOURNAL"
```

Illustrative result — A reconciled and verified completion

```
[PASS] Namespace apply verified. Journal/result:  
/home/demo/ssd0-plan.json.journal.json  
[NOTE] No GRAID PD/DG, filesystem or mount was created.
```

If manual provisioning fails, retain the last successful command, its returned NSID and the current inventory. Do not switch to script apply as an automatic rollback. Manual operations do not produce a script journal.

3. Get help with an unexpected result

If namespace creation, attachment or consumer access fails, retain the last successful command and its output. Provide Graid Support with the target address and firmware, SSD model and serial, allocation record, namespace inventory and apply journal when available. Keep I/O stopped on affected namespaces until the state is understood.

If no drives appear, check the GRAID Host NQN and namespace assignment. After correction, repeat the checks in 6.6.3 before restarting I/O.

6.6.6 Multi-host FAQ

Must slicing finish before AEC connects the target?

Yes. For a new deployment, finish slicing, record the namespace identities, and complete the supported host assignment before running `graidctl connect remote_target` on any consumer. The administration host uses `nvme connect` earlier for namespace management. The AEC command is `connect remote_target`, not `create remote_target`.

Can the deployment use more than four DGX hosts?

The four-host diagram is an example, not a declared maximum. More hosts may be possible if the SSDs have enough namespace slots and capacity, the JBOF firmware supports the required controllers and host assignments, and the complete configuration is validated. Confirm the supported host count with Graid Support and the JBOF vendor. Do not infer unlimited host support from namespace creation alone.

Does each DGX need an AEC license?

Yes. Every DGX that runs SupremeRAID™ AEC must have its own valid AEC license and complete the required activation and GPU binding. Sharing one GP Spark JBOF does not make one host's AEC license cover the other hosts. Confirm the appropriate AEC license and driver combination for every DGX before deployment.

Do all DGX hosts run namespace apply?

No. One administration DGX runs the batch and keeps its plan and per-SSD journals. Consumer DGXs connect after handoff, verify their assigned slices, and create their own AEC RAID. They do not copy or replay administration journals.

Are NQN and SSD serial required?

Batch plan reads NQN and serial from the selected SSDs; config identity defaults are ignored. Single-SSD interactive plan uses blank identity defaults or supplied filters. Non-interactive single-SSD plan and direct apply/resume require both identities. Saved-plan apply reads them from the file. Kernel targets may report virtual serials.

Is the namespace journal the same as RAID journaling?

No. `--namespace-journal` is the script's local provisioning checkpoint and result file. RAID distributed journaling is a separate AEC data-protection feature. Neither the provisioning journal nor a successful apply assigns host access or replaces a data backup.

What if the target was already connected through AEC?

Do not begin slicing. Inspect existing remote targets, PDs, DGs, VDs and all initiator use, and arrange the supported maintenance and connection-handoff procedure. The script refuses live apply when local AEC PDs or DGs exist. It does not automatically release an existing AEC deployment or prove that remote I/O has stopped.

7 Uninstalling

This section describes a full NVIDIA DGX Spark recovery and removal of SupremeRAID™ AEC only. For another GB10-based host, use that vendor's recovery procedure rather than the NVIDIA DGX Spark recovery image.

7.1 Full DGX Spark Restore

Full system recovery reinstalls the operating system and firmware and completely erases the DGX Spark internal SSD, including applications, configuration and local data. Back up required data first. Creating the recovery USB also erases that USB drive. Disconnect external storage as directed by NVIDIA before recovery.

Use NVIDIA's procedure and recovery image for DGX Spark Founders Edition. For an OEM system, obtain the matching recovery image and instructions from its manufacturer. Follow the official procedure below; this is a full system recovery, not a driver-only rollback.

[NVIDIA DGX Spark User Guide — System Recovery](#)

7.2 Remove SupremeRAID™ AEC Only

To remove SupremeRAID™ AEC while keeping the rest of the system, first stop applications and I/O, stop configured automount units, unmount the affected filesystems and verify no holders remain (Section 4.2). Remove the related fstab entries before removing the packages:

Commands

```
sudo apt remove graid-sr graid-mgr
```

Illustrative result

```
# The selected AEC packages are removed.
# Review package-manager errors before clearing setup state.
```

Then purge what the setup script installed — the resume service, login reminder, saved state, and `/etc/graid_pre_installer.conf`:

Commands

```
sudo python3 gpspark_setup.py --reset
```

Illustrative result

```
# Script-managed resume state and setup helpers are cleared.
# This is not a routine retry or namespace rollback operation.
```

Alternatively, `--uninstall` removes the packages and setup state and also deletes `/etc/graid.conf`. Preserve the configuration first if you need it for recovery. This mode removes the script-managed fstab entry; review any additional custom mount entries separately:

Commands

```
sudo python3 gpspark_setup.py --uninstall
```

Illustrative result

```
# Packages and setup state are removed; /etc/graid.conf is deleted.
# Review the command result and any remaining custom mount entries.
```

Mount cleanup belongs before package removal. Confirm that any custom virtual-drive mount entries have also been removed so the system does not attempt to mount unavailable devices.

Note. Removing SupremeRAID™ AEC this way leaves the DKMS NVIDIA driver in place (it does not restore the original pre-built driver). If you need the pre-built driver back, use the full DGX Spark restore in Section 7.1.

8 Troubleshooting

This section collects common issues and their resolutions. The legacy DGX Spark GPU-recognition workaround is in Section 8.5; fabric prerequisites for other AE platforms are covered separately in Section 8.6.

Related troubleshooting elsewhere in this guide:

- Network port cannot be pinged — see Section 3.2 (Network Issue Troubleshooting).
- NVIDIA driver replacement when required by the approved package — see Section 6.2.1.
- DGX Spark GPU not recognized by the installer — see Section 8.5.

8.1 SupremeRAID™ AEC service does not start

Symptom. The SupremeRAID™ AEC service is not running, or RAID components are unavailable after boot.

Check. Inspect the service status and recent logs:

Commands

```
systemctl status graid.service graid-mgr.service
journalctl -u graid.service -u graid-mgr.service -b
```

Illustrative result

```
graid.service: failed
# Example failure branch: inspect the journal for the actual cause.
# Successful service output instead shows active (running).
```

If either service is inactive or failed, review its logs and resolve the cause. When the configuration is ready and it is appropriate to bring the RAID online, restart graid followed by graid-mgr and check both services again:

Commands

```
systemctl restart graid.service
systemctl restart graid-mgr.service
```

Illustrative result

```
# Restart may be silent on success.
# Re-run the status/journal checks above; do not assume health.
```

8.2 Drive group keeps entering RESYNC

Symptom. A drive group repeatedly enters the RESYNC state instead of reaching OPTIMAL.

Check. This usually indicates an unstable connection to the underlying GP Spark NVMe-oF devices. Confirm the remote target and all namespaces are connected and stable:

Commands

```
graidctl list physical_drive
graidctl list remote_target
nvme list-subsys
dmesg | grep -i nvme
```

Illustrative result

```
# Check expected PDs, target state and connection errors.
# Original NVMe nodes may disappear after AEC claims them as PDs.
# Their absence from nvme list alone is not a disconnection.
```

Verify the cable, link state, RoCEv2 configuration, target address, and namespace stability. After restoring a stable connection, check the existing drive group state and follow the supported recovery procedure. Do not recreate an existing drive group merely to clear RESYNC; contact Graid Support if it does not recover.

Root cause. Repeated RESYNC is usually a symptom of the underlying link dropping and reconnecting, not a RAID fault in itself. A typical sequence is: the network briefly disconnects, so the NVMe-oF namespace drops; the upper-layer mount stays in place (the mount point still exists even though the device underneath is gone); when the link returns and the namespace reconnects, the state is inconsistent and the drive group re-synchronizes. If this happens after every reboot or repeatedly during use, the priority is to stabilize the physical connection — check the cable, the switch/port, the RoCEv2 configuration, and the GP Spark network stability — because that is the root cause.

Mount-layer settings (such as the `x-systemd.automount` option in Section 6.3.2) can make the behavior cleaner when a link drops, but they only mitigate the effect; they do not prevent RESYNC caused by an unstable link. Resolve the connection stability first.

8.3 Create DG/VD fails with `graid_create_vd_ns` error

Symptom. Creating a drive group or virtual drive fails. `dmesg` repeatedly shows:

Illustrative error message

```
Create DG0/VD0 graid_create_vd_ns failed
```

and `graid_server.log` shows:

Illustrative error message

```
[error] [SR] DG0: Can not start VD0: -17
```

Possible cause. Error -17 (EEXIST) indicates an existing object or resource conflict. A mounted virtual drive or another process holding it open is one possible cause; inspect the full driver logs and current RAID objects before selecting a recovery action.

Resolution. Check the connection, then stop applications and I/O, stop configured automount units, and release any mounted or open virtual drives using Section 4.2. Determine whether the failed creation already left a DG or VD behind before retrying. Preserve existing RAID objects and data; contact Graid Support if the state is unclear.

Commands

```
mount | grep gdg      # is a mount still present?
lsof /dev/gdg0n1      # what still has the device open?
fuser -m /dev/gdg0n1  # (alternative) processes using it
```

Illustrative result

```
/dev/gdg0n1 on /mnt/graidd type xfs (...)
# lsof/fuser may show a PID using the VD.
# Stop the listed workload and automount before unmounting.
```

Commands

```
# Stop holders and automount first (Section 4.2).
umount /mnt/graidd
```

Illustrative result

```
# No terminal output on successful unmount.
# A busy error means holders still need to be released.
```

Commands

```
# Inspect existing objects before retrying creation or remounting.
graidctl list drive_group
graidctl list virtual_drive
# Only when the existing VD is healthy and ready to mount:
mount /dev/gdg0n1 /mnt/graidd
```

Illustrative result

```
# Mount succeeds only after the correct existing VD is available.
# Check the actual mount and RAID state before resuming I/O.
```

8.4 graidctl cannot connect to the remote target

Symptom. graidctl connect remote_target fails, or the GP Spark namespaces do not appear as physical drives.

Check. First confirm the host can discover the target over RDMA, independently of SupremeRAID™ AEC:

Commands

```
nvme discover --transport=rdma --traddr=10.10.10.12 --trsvcid=4420
```

Illustrative result

```
subtype: nvme subsystem
subnqn: nqn.2026-10.com.example:gpspark-ssd0
traddr: 10.10.10.12
trsvcid: 4420
trtype: rdma
```

If discovery returns no records, check the target address, port, transport, network, and target configuration. Use the actual storage subnet (factory example 10.10.10.0/24), confirm link state, and verify the RDMA stack. Once discovery succeeds, retry:

Commands

```
graidctl connect remote_target rdma 10.10.10.12 ipv4 4420
```

Illustrative result

```
# Connection succeeds; inspect remote_target and nvme_drive.  
# If no namespaces appear, check target access and Host NQN.
```

8.5 DGX Spark GPU not recognized by the installer (legacy workaround)

Symptom. An older installer or a package with an outdated supported-GPU list rejects the DGX Spark GPU. Current packages are expected to recognize supported DGX Spark systems without this override.

The installer reports that no valid or supported GPU/card was detected; the exact message and product label can vary by release.

Resolution. First verify that the package supports your host and obtain a current installer from Graid Support. Only if Support confirms that the legacy override is needed for that package, run the following as root before retrying the installer. Preserve any existing `/etc/graid_pre_installer.conf` before replacing it:

Commands

```
# Run as root only when Graid Support approves this workaround.  
echo 'EXPECTED_GPU_CARDS="2e12"' > /etc/graid_pre_installer.conf  
INSTALLER='/absolute/path/to/approved-installer.run'  
"$INSTALLER" --accept-license
```

Illustrative result

```
# The override file contains the Support-approved GPU ID.  
# Retry the approved installer and inspect its actual output.
```

`EXPECTED_GPU_CARDS="2e12"` is the historical DGX Spark GB10 example. Verify the actual PCI device ID with `lspci -nn` and confirm it with Graid Support; do not use it to force installation on unsupported hardware. The setup script writes this file only when `[supremeraid] expected_gpu_cards` is non-empty; its supplied default is empty.

8.6 Other SupremeRAID™ AE platforms: GPU fabric prerequisites

This GB10-focused guide does not define support for other multi-GPU SupremeRAID™ AE platforms. For a system using NVLink bridges, NVLink, or NVSwitch, confirm the GPU topology, driver, and SupremeRAID™ AE release with Graid Support. A blanket statement that AE/AEC cannot run in an NVLink environment is not appropriate.

On platforms that require NVIDIA Fabric Manager (particularly NVSwitch-based systems), verify that the matching Fabric Manager software is installed and running and that the fabric is initialized. Direct NVLink-bridge systems and other topologies have platform-specific requirements; the presence of an NVLink label alone does not establish an FM requirement. Installing FM does not by itself establish SupremeRAID™ AE support.

Reference: NVIDIA Fabric Manager User Guide — <https://docs.nvidia.com/datacenter/tesla/fabric-manager-user-guide/>