DGX Spark User Guide

NVIDIA Corporation

Jun 24, 2026

Release Notes

This guide is also available for download as a PDF.

Chapter 1. Overview

The DGX Spark is NVIDIA's compact AI computer designed for developers, data scientists, and AI researchers who need powerful computing capabilities for AI development and deployment.

Chapter 2. Getting Started

2.1. DGX Spark Release Notes

This section provides release notes for the DGX Spark, including information about new features, known issues, and software version updates.

2.1.1. Current Software Versions

The following table shows the current version information for the DGX Spark software stack.

Component	Version
NVIDIA DGX OS	7.5.0
NVIDIA GPU Driver	580.159.03
NVIDIA CUDA Toolkit	13.0.2
Canonical Kernel	6.17
UEFI	1.108.20
Embedded Controller (EC)	3.3.2
USB Power Delivery (USB PD)	0.5.22
Trusted Platform Module (TPM)	7.516.1
System on Chip (SoC)	2.152.15

Note

These release versions apply only to the DGX Spark Founders Edition. GB10-based partner systems may not receive updates at the same time.

2.1.1.1 June 2026 Release

2.1.1.1.1 What's New

- **Out-of-Box-Experience (OOBE) Enhancements** - Updates to the OOBE that enable faster initial device setup time and easier navigation to discover and install local AI agents, including:

- **Over-the-Air (OTA) Updates** - OTA updates are not installed by default during initial setup, allowing users to start using their system sooner. Users can download and install OTA updates after initial system setup.
- **NemoClaw Playbook** - After initial system boot, the DGX Spark playbook site opens, where the NemoClaw playbook is prominently displayed for streamlined setup of a sandboxed local agent.

- **New Application and Library Features** - NVIDIA Sync and NVIDIA Collective Communications Library (NCCL) include the following updates:
  - **NVIDIA Sync Cluster Assistant** - The Cluster Assistant is now available on the **Settings** page in the NVIDIA Sync application. The Cluster Assistant helps users connect up to three devices without a network switch, and up to four devices using a switch.
  - **NCCL** - NCCL (Version 2.30u1) has updated support for connecting three DGX Spark systems in a ring topology.

2.1.1.2 April 2026 Release

2.1.1.2.1 What's New

- **Enterprise Features** - Support for tools and capabilities designed to facilitate system management in enterprise environments, including:
  - **Enterprise Management Guide** - Enables IT administrators to plan, configure, and manage DGX Spark deployments at scale with a dedicated reference covering workflows, provisioning, update procedures, and system lifecycle management across fleets. For more information, refer to *Enterprise Manageability*.
  - **Skip the Automated OOBE for IT Administrator Provisioning** - Allows IT administrators to bypass the OOBE entirely during provisioning, reducing manual per-device setup steps and minimizing touch while deploying across multiple DGX Spark units.
  - **USB and Local Repository Support for Installations and Updates** - Allows IT administrators to deliver OS installations and software updates through USB drives or internal local package repositories, removing the dependency on cloud connectivity for software distribution and patch management, and allowing administrators to control when new releases are installed.
  - **Support for Air Gapped Deployment and Updates** - Enables IT administrators to deploy and operate DGX Spark systems on isolated networks with no external internet access, suitable for strict security, compliance, and data sovereignty requirements.
  - **Customized Enterprise ISOs Through Cloud-init** - Allows IT administrators to embed site-specific configurations (users, network settings, software, and policies) directly into custom DGX OS images using Cloud-init, ensuring every unit arrives pre-configured to enterprise standards on first boot.
- **DGX Dashboard Enhancements** - Release highlights are now provided for software releases, enabling users to determine the urgency of applying updates.

2.1.1.2.2 Fixed Issues

- **Bluetooth Keyboard Pairing** - Addressed an issue where the Bluetooth keyboard pairing modal did not appear in some instances when a keyboard was plugged in.
- **Unable to Detect Network During System Setup** - Addressed Wi-Fi and Ethernet connectivity issues reported during system setup.

2.1.1.3 March 2026 Release

2.1.1.3.1 What's New

- **Partner Factory Updates** - Firmware update to facilitate GB10 partner factory updates.

2.1.1.4 February 2026 Release

2.1.1.4.1 Fixed Issues

- **Improved Performance with Multiple DGX Spark Systems** - Addressed performance regression noted by some users with multiple connected DGX Spark systems after updating to DGX OS 7.4.0.

2.1.1.5 January 2026 Release

2.1.1.5.1 What's New

- **Power Management** - Added hot plug support for the ConnectX-7 network adapter, saving up to 18W of power when the adapter is not in use.
- **Bluetooth Audio Support** - Support for Bluetooth audio devices, including headphones and headsets, is now available.
- **Additional Security Controls** - Wi-Fi and Bluetooth can now be disabled in the Unified Extensible Firmware Interface (UEFI) for environments requiring additional security.

2.1.1.5.2 Fixed Issues

- **Improved Monitor Support** - Better compatibility with TVs and monitors, including multi-monitor configurations and the use of non-native resolutions and refresh rates.
- **Improved Peripheral Setup** - Smoother and more complete OOBE navigation.

2.1.1.6 November 2025 Release

2.1.1.6.1 What's New

- **New DGX OS kernel** - The operating system now incorporates the Ubuntu 6.14 Hardware Enablement (HWE) kernel stack. The new kernel brings ongoing performance gains, enhanced stability, broader hardware compatibility, and the latest security updates for a more secure operating environment.
- **JupyterLab updated to latest CUDA and PyTorch** - Updated JupyterLab to CUDA 13.0.2 and the latest PyTorch version, enabling users to work with updated frameworks immediately without extra downloads.

2.1.1.6.2 Fixed Issues

- **Improved memory reporting in DGX Dashboard** - Addressed the issue with memory reporting differences with unified memory architecture. The readout is now consistent with CUDA guidance for unified memory systems. For more information, refer to CUDA unified memory guidance.
- **Image Generation in JupyterLab** - Resolved the inability to generate images in Stable Diffusion XL Playbook example. Users can now run end-to-end example workflow inside JupyterLab.
- **Improved Peripheral Interoperability** - Better compatibility with USB-C devices, monitors, Bluetooth peripherals, and Wi-Fi access points.

- **Improved recovery image reliability** - The recovery image now installs correctly on macOS and also when multiple external USB-C drives are connected.
- **Keyboard accessibility improvements** - Smoother and more complete OOBE navigation, including the ability to complete entire system setup process using only keyboard.

2.1.2. Known Issues

For an updated list of known issues, refer to *Known Issues*.

2.2. Known Issues

This topic provides a summary of known issues with DGX Spark systems.

2.2.1. Use the supplied power adapter for optimal performance

For optimal performance, use the supplied power adapter with the DGX Spark system. Using a different adapter may reduce performance, prevent boot, or cause unexpected shutdowns.

2.2.2. `nvidia-smi` reports “Memory-Usage: Not Supported”

On iGPU platforms, `nvidia-smi` will display “Memory-Usage: Not Supported” even though per-process GPU memory is listed. This is expected because iGPUs do not have dedicated framebuffer memory.

2.2.3. CUDA Support on DGX Spark

The CUDA version on your DGX Spark device has been verified to work with your system hardware at the time of software update release. The latest features and performance improvements are available through NVIDIA NGC containers (for example, PyTorch, vLLM, and TensorRT-LLM).

2.2.4. Guidance for reporting memory resources with unified memory architecture

NVIDIA is actively working with third-party ecosystem partners to bring their software to DGX Spark. For example, we have provided direction on implementing memory management on systems based on a unified memory architecture (UMA) to help ensure accurate reporting of available resources.

DGX Spark systems use a unified memory architecture (UMA), where the GPU shares system memory (DRAM) with the CPU and other compute engines. This design reduces latency and allows larger amounts of memory to be used for GPU workloads. On UMA systems, the CPU can dynamically manage DRAM contents, including freeing up memory by swapping pages between DRAM and the system’s SWAP area. However, the `cudaMemGetInfo` API does not account for memory that could potentially

be reclaimed from SWAP. As a result, the memory size reported by `cudaMemGetInfo` may be smaller than the actual allocatable memory, since the CPU may be able to release additional DRAM pages by moving them to SWAP.

To more accurately estimate the amount of allocatable device memory on DGX Spark platforms, CUDA application developers should consider the possibility of DRAM reclamation via SWAP and not rely solely on the values returned by *cudaMemGetInfo*. The following provides an example implementation using C standard libraries:

```c
#include <stdio.h>

int getAvailableMemory(long *availableMemoryKb, long *freeSwapKb) {
    FILE *meminfoFile = NULL;
    char lineBuffer[256];
    long hugeTlbTotalPages = -1;
    long hugeTlbFreePages = -1;
    long hugeTlbPageSize = -1;

    if (availableMemoryKb == NULL || freeSwapKb == NULL) {
        return 1;
    }

    meminfoFile = fopen("/proc/meminfo", "r");
    if (meminfoFile == NULL) {
        return 1;
    }

    *availableMemoryKb = -1;
    *freeSwapKb = -1;

    while (fgets(lineBuffer, sizeof(lineBuffer), meminfoFile)) {
        long value;

        if (sscanf(lineBuffer, "MemAvailable: %ld kB", &value) == 1) {
            *availableMemoryKb = value;
        } else if (sscanf(lineBuffer, "SwapFree: %ld kB", &value) == 1) {
            *freeSwapKb = value;
        } else if (sscanf(lineBuffer, "HugePages_Total: %ld", &value) == 1) {
            hugeTlbTotalPages = value;
        } else if (sscanf(lineBuffer, "HugePages_Free: %ld", &value) == 1) {
            hugeTlbFreePages = value;
        } else if (sscanf(lineBuffer, "Hugepagesize: %ld kB", &value) == 1) {
            hugeTlbPageSize = value;
        }

        if (*availableMemoryKb != -1 &&
            *freeSwapKb != -1 &&
            hugeTlbTotalPages != -1 &&
            hugeTlbFreePages != -1 &&
            hugeTlbPageSize != -1) {
            break;
        }
    }
```

(continues on next page)

(continued from previous page)

```
    fclose(meminfoFile);

    if (hugeTlbTotalPages != 0 && hugeTlbTotalPages != -1) {
        *availableMemoryKb = hugeTlbFreePages * hugeTlbPageSize;

        // Hugetlbfs pages are not swappable.
        *freeSwapKb = 0;
    }

    return 0;
}
```

As a workaround for debugging purposes, you can flush the buffer cache manually with the following command:

```
sudo sh -c 'sync; echo 3 > /proc/sys/vm/drop_caches'
```

After flushing the cache, restart your application.

2.2.5. Attached HDMI display goes into deep sleep mode after an extended period of inactivity and does not wake

In some cases, after a long period of inactivity, the attached HDMI display might go into a deep sleep mode and not wake up when you press a key or move the mouse. If this happens, press a physical button on the monitor to wake it up.

2.3. Hardware Overview

Powered by the NVIDIA Grace Blackwell architecture, DGX Spark enables developers, researchers, and data scientists to prototype, deploy, and fine-tune large AI models on their desktop. This section provides information about the hardware components and specifications.

2.3.1. System Overview

The DGX Spark features:

- NVIDIA Grace Blackwell architecture with integrated GPU and CPU
- 20-core Arm processor with high-performance cores
- 128 GB unified system memory
- Compact desktop form factor
- Advanced connectivity including Wi-Fi 7, 10 GbE, and ConnectX-7
- Support for AI models up to 200 billion parameters (or 405B for dual-Spark configuration)

2.3.1.1 Component Descriptions

The DGX Spark includes the following components:

Table 1: Component Specifications

Component	Specification
GPU	NVIDIA Blackwell Architecture with 5th Generation Tensor Cores, 4th Generation RT Cores
CPU	20-core Arm processor (10 Cortex-X925 + 10 Cortex-A725)
Memory	128 GB LPDDR5x unified system memory, 256-bit interface, 4266 MHz, 273 GB/s bandwidth
Storage	1 TB or 4 TB NVMe M.2 with self-encryption
Network	1x RJ-45 (10 GbE), ConnectX-7 Smart NIC, Wi-Fi 7, Bluetooth 5.4
Connectivity	4x USB Type-C, 1x HDMI 2.1a, HDMI multichannel audio
Video Processing	1x NVENC, 1x NVDEC

2.3.2. Physical Specifications

2.3.2.1 Form Factor

- **Chassis Type**: Small form factor (SFF)
- **Dimensions**: 150 mm (L) x 150 mm (W) x 50.5 mm (H)
- **Weight**: 1.2 kg (2.6 lbs)

2.3.2.2 Environmental Requirements

Table 2: Environmental Specifications

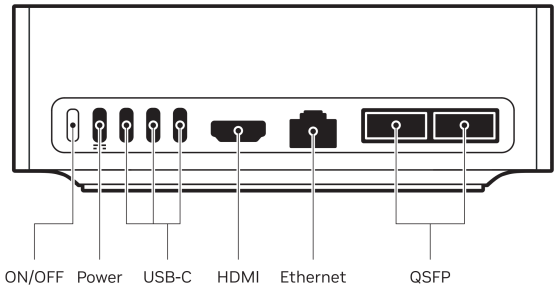
Specification	Value
Ideal Operating Temperature	5°C to 30°C (41°F to 86°F)
Operating Humidity	10% to 90% (non-condensing)

2.3.3. Connectivity and I/O

2.3.3.1 Rear Panel

- Power button
- 4x USB Type-C (one for power delivery)
- 1x HDMI 2.1a display connector
- 1x RJ-45 Ethernet connector (10 GbE)

- 2x QSFP Network connectors (ConnectX-7)



2.3.4. Performance Specifications

2.3.4.1 Compute Performance

- **AI Compute**: Up to 1,000 TOPS (trillion operations per second) inference and up to 1 PFLOP (petaFLOP) at FP4 precision with sparsity
- **CUDA Cores**: 6,144
- **Copy Engines**: 2 (enables simultaneous data transfers to and from GPU memory, improving throughput for AI workloads)
- **CPU Performance**: 20 cores (10 Cortex-X925 + 10 Cortex-A725)
- **Memory Bandwidth**: 273 GB/s
- **Memory Channels**: 16 channels (256 bit) LPDDR5X 8533

2.3.4.2 AI/ML Capabilities

- **Model Support**: AI models up to 200 billion parameters
- **Tensor Performance**: 5th Generation Tensor Cores with FP4 support
- **Framework Support**: PyTorch, TRT-LLM, and other AI frameworks
- **Use Cases**: Inference, deployment, and fine-tuning of large language models

2.3.5. Power and Thermal Management

2.3.5.1 Power Requirements

- **Power Supply**: 240W external power supply (included)
  - GB10 SOC Thermal Design Power (TDP) is 140W
  - 100W is available for other system components (ConnectX-7, Wi-Fi, SSD, USB-C, etc.)
- **Usage Requirement**: Use of the provided 240W power supply is required for optimal performance. Using a different or lower-rated power supply may result in reduced system performance, failure to boot, or unexpected shutdowns.
- **Input Voltage**: Standard AC power input

2.3.5.2 Thermal Management

- **Cooling Solution**: Integrated thermal management system
- **Form Factor**: Compact design optimized for desktop placement
- **Ideal Operating Temperature**: 5°C to 30°C (41°F to 86°F)

2.4. Initial Setup - First Boot

This guide walks you through setting up your DGX Spark for the first time. You'll choose how to access your system during the initial setup, and run the first-time setup utility to configure everything. The access method you choose is only for completing the initial setup - after setup is complete, you can access your DGX Spark any way you like: locally with a monitor and keyboard, over the local network from another computer, or a mix of both.

2.4.1. What You'll Do

This setup process includes:

- Choosing how to access the system during initial setup (with a display, or as a network appliance)
- Preparing your system and connections
- Running the first-time setup utility to configure your system

2.4.2. Choose how to access your system during initial setup

To complete the initial setup, you'll need to access your DGX Spark. You can do this in one of two ways:

With a Display (Local Setup)

- Connect keyboard and mouse via USB or Bluetooth
- Connect a display to work directly on the system
- Follow the setup wizard on screen

Over the Network (as a Network Appliance)

- Access the system over your local network from another computer
- Use another computer to complete setup via web browser
- No Spark display or keyboard required for the setup process

> **Note**
>
> This choice is only about how you'll complete the initial setup process. After setup is finished, you can access your DGX Spark however you prefer. You're not locked into your original choice.

2.4.3. Get Ready

Important

The DGX Spark device starts up immediately when power is applied. Please attach all peripherals (display, keyboard, mouse, network, etc.) before connecting the power supply.

Before starting, ensure you have:

- A fast, reliable internet connection (Wi-Fi or Ethernet) to download required updates during the initial setup process. Connections using captive portals (such as hotel or airport Wi-Fi) or those prone to disconnections (like phone hotspots) are not recommended. If you do not have access to a stable connection, consider downloading the system recovery media and using *System Recovery* to install the latest software for your DGX Spark.
- For Local Setup: A display, keyboard, and mouse connected (or available using Bluetooth).
- For Network Setup: A computer on the same network to access the setup interface.
- Power connected to the system (the system will start automatically when power is applied).

Note

Display Troubleshooting: Some displays may have trouble with Spark out of the box. If you are connecting over USB-C/DisplayPort and there is no display, try using HDMI instead.

Note

If you plan to use a wired network connection, plug in the network cable before starting the installation. This helps avoid connection issues later in the process.

2.4.4. Run the First-Time Setup

The first-time setup utility will guide you through:

- Powering on and initializing the system
- Selecting your preferred setup mode
- Downloading and installing critical updates
- Completing your initial configuration

Warning

Critical: Do not shut down or reboot the system during the update process. The installation cannot be interrupted once the download begins, and powering down during updates can cause system damage.

2.4.4.1 Getting Started

The way you start the installation depends on your chosen access method:

With a Display (Local Setup)

1. Power on the system
2. The first-time setup utility will start automatically on the connected display
3. Use your wired keyboard and mouse (already connected) to navigate
4. If a keyboard or mouse is not detected, you will be prompted to put your Bluetooth devices in pairing mode:

   USB devices can be plugged in at any time and should start working, even if detected improperly. Bluetooth devices can be put into pairing mode and will generally still pair while on the "Get Started" screen (exception - keyboards that require a passcode to type in won't work on this screen). Once you click on "Get Started," Bluetooth pairing stops, so you will have to power cycle to try again.
5. Follow the on-screen prompts to complete the setup process

Over the Network (as a Network Appliance)

1. Power on the system. This creates a Wi-Fi hotspot that you will use to connect to the system and continue the setup process. The SSID and password for the Wi-Fi hotspot are printed on a sticker attached to the Quick Start Guide included with your DGX Spark's packaging
2. From another computer, connect to the Spark's Wi-Fi hotspot using the SSID and password provided on the Quick Start Guide. A captive portal page will open in the default web browser on your computer. If it does not open automatically, use your browser to navigate to the Spark's system setup page listed on the Quick Start Guide.
3. Follow the on-screen prompts to continue the setup process.

   When the DGX Spark joins your home network, its Wi-Fi hotspot will turn off and your computer will reconnect to the device through your Wi-Fi network to resume the setup process. If you are not able to connect to the DGX Spark after it joins your home network, you must connect a display/keyboard/mouse to continue.

> **Note**
>
> It may take as long as 10 minutes for the DGX Spark to install all of it's updates and join your home network.

2.4.4.2 What to Expect During Setup

The first-time setup utility will guide you through several configuration steps. Simply follow the on-screen prompts to complete each step.

Setup Process Steps:

1. **Language and Time Zone Selection**

   Choose your preferred language and time zone settings for the system. Note that the input fields will filter as you type.
2. **Keyboard Layout Selection** (Local Setup only)

   Select your keyboard layout (e.g. US keyboard vs. Russian keyboard). This screen only appears when using a display during setup.

3. **Terms and Conditions**

   Review and accept the terms and conditions to continue with the installation.

4. **User Account Creation**

   Create your username and password for system access.

5. **Information Sharing Settings** (Optional)

   Configure analytics and crash reporting preferences. You can skip this step if desired.

6. **Wi-Fi Network Selection**

   Select your Wi-Fi network. This step is automatically skipped if an Ethernet cable that is providing internet access is connected.

7. **Wi-Fi Password**

   Enter the password for your selected Wi-Fi network.

8. **Joining Wi-Fi Network**

   The system connects to your Wi-Fi network and tears down the access point. Your computer will automatically reconnect to your default network.

   Note

   Network Connection Issues:

   - If your computer automatically reconnects to the same network as the Spark, the installation should continue seamlessly
   - If not, you'll need to connect your computer to the same network as the Spark while the setup app is waiting for the network setup process to complete
   - If the setup fails, you must connect a display/keyboard/mouse to continue
   - The modal instructs you to try to reconnect to the Spark's hotspot and try again. This will work if the Spark actually failed to join the network (e.g. wrong password) versus your laptop can't communicate with the Spark
   - If you DO NOT see the hotspot available when this error modal appears, that means the Spark did join the network but your laptop can't communicate with it. This could be because:
     - Device isolation
     - You failed to join the same network as your Spark
     - mDNS does not work on your network due to its configuration (e.g. a complex corporate network)

9. **Software Download and Installation**

   After connecting to the network, the system will automatically download and install the full software image. This process can take some time, and the system may reboot more than once before it's complete. If you are setting up over the network from another computer, you will not be able to access the device during this time. The process may continue for up to 10 minutes after the interface shows that the device is rebooting to finish the setup

> **Warning**
>
> **Do not shut down or reboot the system during this process.** The installation cannot be interrupted once the download begins.

10. **Installation Complete**

    The device will reboot automatically when installation is complete, and you can then use it normally.

2.4.5. Next Steps

Congratulations! Your DGX Spark is now ready to use. Here are the recommended ways to access and start working with your system:

Access Your System

- **Locally**: Use your DGX Spark like any desktop computer with a monitor, keyboard, and mouse
- **Over the Network**: Connect to your DGX Spark from another computer on the same network using the NVIDIA Sync tool (see *NVIDIA Sync*), SSH, or remote desktop
- **Mix Both**: Use whatever access method fits your current workflow - you can switch freely between local and network access
- **Dashboard Access**: Use the built-in DGX Dashboard for system monitoring, updates, and JupyterLab access. See *DGX Dashboard* for more information

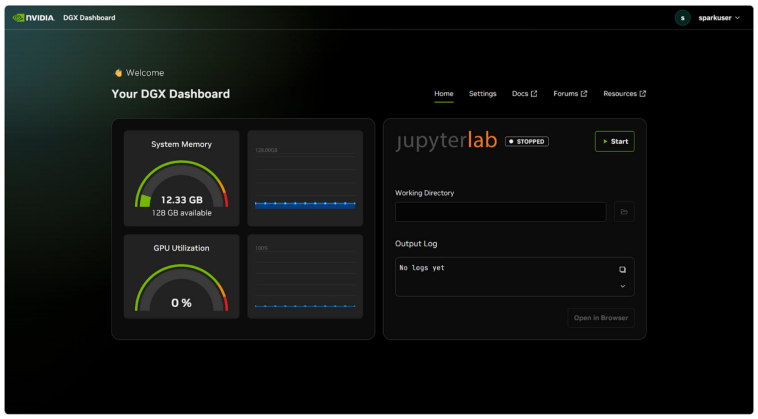


Fig. 1: The DGX Dashboard provides an intuitive interface for system monitoring and development

Additional Resources

- Visit the NVIDIA Spark Developer Portal at https://build.nvidia.com/spark for the latest guides, tutorials, and updates
- Refer to *DGX Spark Release Notes* for the latest software updates and features
- See *Known Issues* for troubleshooting common problems

Your DGX Spark is now ready to power your AI development and deployment workflows!

2.5. System Configuration and Operation

Configuring and operating your DGX Spark effectively is key to delivering consistent results across AI/ML workflows. This section provides an overview of the platform, recommended UEFI settings, clustering procedures, and foundational security guidance to help you deploy, manage, and scale with confidence.

2.5.1. System Overview

Powered by the NVIDIA Grace Blackwell architecture, DGX Spark enables developers, researchers, and data scientists to prototype, deploy, and fine-tune large AI models on their desktop.

2.5.1.1 Flexible Access and Usage

The DGX Spark is designed for maximum flexibility in how you access and use it. You can seamlessly switch between different access methods based on your needs:

- **Local Access**: Connect a keyboard, mouse, and monitor to work directly on the system
- **Network Access**: Access your system from another computer on the same network using SSH, *NVIDIA Sync*, or remote desktop tools
- **Hybrid Usage**: Mix and match access methods - work locally one day and over the network the next, or even simultaneously

All access methods are fully supported and equally capable. Your DGX Spark adapts to your workflow, whether you're working at your desk with a monitor or accessing it remotely as a network appliance on the same network.

2.5.1.2 Key Capabilities

Your DGX Spark enables you to:

- **Run Inference**: Deploy models for real-time AI applications
- **Develop AI Models**: Train and fine-tune models with up to 200 billion parameters
- **Process Data**: Handle large datasets with high-performance computing
- **Experiment Freely**: Test new ideas without cloud computing costs
- **Scale Workloads**: Connect multiple systems for larger projects

2.5.1.3 System Architecture

The DGX Spark is built on NVIDIA's Grace Blackwell architecture, providing:

- **Unified Memory**: 128 GB of high-bandwidth memory for large models
- **High-Performance Computing**: 20-core ARM64-based processor with integrated GPU
- **Advanced Connectivity**: Wi-Fi 7, 10 GbE, ConnectX-7 NIC, and multiple I/O options
- **Compact Form Factor**: 150mm x 150mm x 50.5mm desktop design

For detailed hardware specifications, see *Hardware Overview*.

2.5.1.4 Software

Your system comes pre-configured with:

- **NVIDIA DGX OS**: Optimized operating system for AI workloads
- **Development Tools**: CUDA, cuDNN, and NVIDIA's development ecosystem
- **Container Support**: Docker and NVIDIA Container Runtime for easy deployment
- **NGC Integration**: Access to NVIDIA's container registry

For detailed software information, see *Software*.

2.5.1.5 Getting Started

To begin using your DGX Spark:

1. **Initial Setup**: Follow the *Initial Setup - First Boot* to configure your system
2. **Explore Examples**: Try sample workloads to understand capabilities
3. **Configure Development Environment**: Set up your preferred tools and frameworks
4. **Start Building**: Begin your AI development projects

> **Note**
>
> For the most up-to-date tutorials and examples, visit https://build.nvidia.com/spark. This site is regularly updated with new content and serves as the primary resource for practical guides and use cases.

2.5.2. UEFI Settings

This topic provides guidance on accessing and configuring the UEFI settings for the DGX Spark system. While there are no Spark-specific features that require UEFI configuration, you may need to access the UEFI for general system configuration or troubleshooting purposes.

2.5.2.1 Access UEFI

> **Important**
>
> To access the UEFI setup menu, you must be using a keyboard that is physically connected to the DGX Spark device.
>
> If the keyboard is a Mac keyboard, it might not have a key that UEFI recognizes as Del; use Esc only.

To access the UEFI setup menu:

1. Power on or restart the system.
2. Immediately press `Esc` or `Del` and hold it until the UEFI setup menu appears.

> **Note**
>
> The timing for pressing `Esc` or `Del` is critical. Press the key as soon as the system starts booting, before the OS begins loading.

2.5.2.2 UEFI Documentation

For detailed information about UEFI settings and configuration options, refer to the DGX Spark UEFI Manual.

2.5.2.3 Enable or Disable WiFi and Bluetooth

You can enable or disable both Wireless LAN and Bluetooth together in UEFI, which completely disables all wireless access for security enhancement. To disable only one (WiFi or Bluetooth) individually, use the OS settings instead. You might disable wireless to meet lab or security policy, or enable it for initial setup when Ethernet is not available.

1. Go to **Advanced** → **Advanced Menu** → **IO Port Access**.
2. Set **Wireless LAN & Bluetooth** to the desired state (enabled or disabled).

2.5.2.4 Boot from a USB Device

To boot from a USB device, use one of the following methods. You might do this to run system recovery, reinstall the OS, or boot from live or diagnostic media.

Set USB as the first boot device (persistent):

1. Go to **Boot** → **Boot Option Priorities**.
2. Use `Enter` on each entry to set the boot order. Place the USB device at the top.

Boot from USB once (one-time):

1. Go to **Save & Exit**.
2. Under **Boot Override**, select the USB device and press `Enter`. The system boots from that device for the current boot only.

The USB device must appear in the boot option list. If it does not, ensure the device is connected before you enter UEFI and that it is bootable.

2.5.2.5 Enable or Disable Secure Boot

To enable or disable Secure Boot, perform the following steps. You might disable it to use PXE boot or run custom boot loaders, or enable it to meet security or compliance requirements.

1. Go to **Security** → **Secure Boot**.
2. Set **Secure Boot** to **Enabled** or **Disabled** as needed. The default is **Enabled**.
3. Save changes and exit from the **Save & Exit** menu. A platform reset may be required for the change to take effect.

Secure Boot is active when it is enabled, the Platform Key (PK) is enrolled, and the system is in User mode.

2.5.2.6 Configure PXE Boot

To configure UEFI for PXE boot, perform the following steps. You might do this to deploy or reimage the DGX Spark over the network, or to boot the recovery image from a PXE server.

> **Note**
>
> Before enabling PXE boot, you must either disable Secure Boot or enroll the PXE bootloader (`grubnetaa64.efi.signed`) in UEFI. For details, see *PXE Boot Setup*.

1. Go to **Advanced** → **Network Stack Configuration**.
2. Set **Network Stack** to **Enabled**.
3. Set **Ipv4 PXE Support** or **Ipv6 PXE Support** (or both) to **Enabled** as needed.
4. Optionally set **PXE boot wait time** (seconds to press Esc to abort PXE) and **Media detect count**.
5. In **Advanced**, open the network configuration screen for the NIC you want to use for PXE (listed by MAC address). Enable and configure IPv4 or IPv6 for that device as required.
6. Save changes and exit from the **Save & Exit** menu.

For PXE server setup, bootloader configuration, and DHCP setup, see *PXE Boot Setup*.

> **Important**
>
> Always save your UEFI configuration changes from the **Save & Exit** menu so the system restarts with your new settings applied.

2.5.2.7 Troubleshooting

If you experience issues after making UEFI changes:

- Revert to UEFI defaults and restart
- Apply changes incrementally to identify problematic settings
- Update to the latest UEFI version if available
- Consult the DGX Spark UEFI Manual for specific setting descriptions
- For support options, see *Get the Right Support for Your DGX Spark*.

2.5.3. Spark Stacking

2.5.3.1 Connecting DGX Spark Systems into a Cluster

2.5.3.1.1 Overview

This guide explains how to connect multiple DGX Spark systems into a compute cluster using simplified networking configuration and a QSFP/CX7 cable for high-performance interconnect.

The goal is to enable distributed workloads across Grace Blackwell GPUs using MPI (for inter-process CPU communication) and NCCL v2.28.3 (for GPU-accelerated collective operations).

2.5.3.1.2 Connect the QSFP/CX7 Cable

Before configuring networking, connect the two DGX Spark systems with the approved QSFP/CX7 cable. Each unit has two ConnectX-7 ports on the rear panel; see *Connectivity and I/O* for port locations.

1. On each DGX Spark, locate the two QSFP/CX7 ports on the rear panel.
2. Plug one end of the cable into a ConnectX-7 port on the first unit, and plug the other end into the same ConnectX-7 port on the second unit (for example, both left ports or both right ports when viewed from the rear of the unit).
3. Orient the connector so the pull-tab (ring tab) faces the top of the DGX Spark (toward the top surface of the unit), as shown in the following figure.

4. Align the connector with the port and insert it until it seats fully. When aligned correctly, the connector slides in without force.
5. To remove the cable, pull the ring tab straight out. Removal should be smooth when the connector is fully seated.

Warning

Do not force the QSFP/CX7 connector. If it does not slide in smoothly, stop, verify tab orientation and port alignment, and try again. Forcing an upside-down or misaligned connector can damage the port.

2.5.3.1.3 Connect Multiple DGX Spark Systems

Refer to the following playbooks for information on various connection options:

- Connect Two Sparks
- Connect Three Sparks
- Multi Sparks Through a Switch

2.5.3.1.4 Next Steps

Once tested, this configuration can be scaled to support:

- Job orchestration with Slurm or Kubernetes
- Containerized execution with Singularity or Docker

2.6. Software

The DGX Spark comes with a comprehensive software stack optimized for AI development, machine learning, and data science workflows. This section provides detailed information about the included software components and their configuration.

2.6.1. DGX Dashboard

2.6.1.1 Overview

The DGX Spark comes with a built-in dashboard that provides an overview of the system's current operational metrics, the ability to apply updates, change some system settings, and access local Jupyter Notebooks.

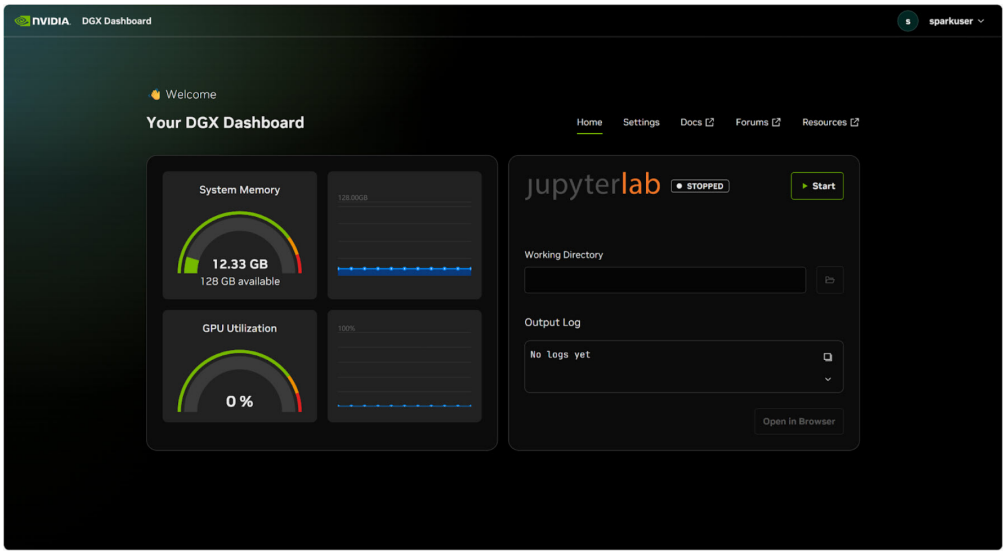


Fig. 2: The DGX Dashboard provides real-time system monitoring and integrated JupyterLab access

Note

To run updates and change the device name, you must have `sudo` access. The account created during initial setup will have access.

2.6.1.2 Integrated JupyterLab

The dashboard includes an integrated JupyterLab instance that provides a convenient development environment:

- When started, JupyterLab creates a virtual environment in the specified working directory and automatically installs a set of recommended packages
- If you enter a new working directory and start JupyterLab, a new environment will be created
- Each user account on the device is assigned a port located in `/opt/nvidia/dgx-dashboard-service/jupyterlab_ports.yaml`
- To access JupyterLab remotely, you must tunnel it just like the dashboard itself. The port to tunnel is in the ports file. Using *NVIDIA Sync*, this tunnel is managed for you automatically and just works

2.6.1.3 Accessing the Dashboard

The dashboard can be accessed locally by clicking on the "Show Apps" button in the bottom left corner of the Ubuntu desktop. Then, in the app grid, select the "DGX Dashboard" shortcut to open the dashboard in your default web browser.

Remotely, the dashboard can be accessed using *NVIDIA Sync* or through a manually created SSH tunnel. If using NVIDIA Sync, after connecting, simply click on the "DGX Dashboard" button and the dashboard will open in your default web browser at http://localhost:11000.

To manually access over SSH, first open a tunnel, e.g., `ssh -L 11000:localhost:11000 <username>@<IP or spark-abcd.local>`. Then, open the dashboard in your web browser at http://localhost:11000.

2.6.2. NVIDIA Sync

2.6.2.1 Overview

NVIDIA Sync is a system tray utility for Windows, macOS, and Ubuntu. It simplifies working with your DGX Spark and other remote Linux systems from another computer on your network.

Install NVIDIA Sync on your laptop or workstation, add your DGX Spark, and connect to launch applications on the device. Sync handles SSH connections, port forwarding, and tunnels so you can use local tools against the remote system.

2.6.2.1.1 Key Capabilities

- One-click launch of supported IDEs and tools (for example, VS Code, Cursor, and NVIDIA AI Workbench)
- Integrated access to the *DGX Dashboard* on your DGX Spark without manual tunnel configuration
- Optional Tailscale integration for secure access when you are not on the same network as your device

2.6.2.2 Using NVIDIA Sync with Your DGX Spark

After initial setup, your DGX Spark advertises its hostname on the local network using multicast DNS (mDNS). You can add the device in NVIDIA Sync by selecting it from device discovery or by entering its hostname or IP address and your login credentials.

Typical workflow:

1. Install NVIDIA Sync on your local computer.
2. Add your DGX Spark in NVIDIA Sync.
3. Connect to the device.
4. Launch an application or the DGX Dashboard from the Sync menu.

2.6.2.3 For More Information

For installation, device management, Tailscale, custom applications, troubleshooting, and more, see the NVIDIA Sync User Guide.

2.6.3. DGX OS

2.6.3.1 Overview

NVIDIA DGX OS is a customized Linux distribution that provides a stable, tested, and supported operating system foundation for running AI, machine learning, and analytics applications on DGX systems. It includes platform-specific optimizations, drivers, and diagnostic tools tailored for NVIDIA hardware.

DGX OS serves as the underlying operating system for your DGX Spark, providing:

- A robust Linux foundation optimized for AI workloads
- Pre-configured drivers and system settings for NVIDIA hardware
- Security updates and system maintenance capabilities
- Compatibility with the broader NVIDIA software ecosystem

Important

Reinstalling or recovering your DGX Spark: DGX Spark uses a different recovery process than enterprise DGX systems. Do not use the Enterprise Download Center or DGX OS ISO. For additional instructions, refer to *System Recovery*. The recovery image is available from developer.nvidia.com and does not require an enterprise account.

Note

For more information about DGX OS, see the official documentation at: https://docs.nvidia.com/dgx/dgx-os-7-user-guide/introduction.html

2.6.3.2 Security and Compliance

DGX OS is based on Ubuntu. For Ubuntu security capabilities and supported compliance standards (such as FIPS, CIS, and DISA-STIG), see the Ubuntu security standards.

2.6.3.3 Release Cadence

DGX OS follows a regular release schedule with updates typically provided twice per year, around February and August, for the first two years after initial release. Additional updates and security patches are provided between major releases and throughout the support lifecycle.

2.6.4. NVIDIA Nsight

2.6.4.1 Overview

NVIDIA Nsight is a family of GPU profiling, debugging, and analysis tools for CUDA and graphics workloads. Individual tools focus on different tasks in the optimization workflow:

- **Nsight Systems** — Timeline profiling of CPU, GPU, and system activity for a full application run.
- **Nsight Graphics** — Frame capture and pipeline inspection for graphics APIs.
- **Nsight DL Designer** — Visualization and analysis for deep learning models.
- **Nsight Compute** — CUDA kernel profiling, memory metrics, and guided performance reports.

The DGX Spark stack follows the broader NVIDIA software release model. Current installers, release notes, supported platforms, and user documentation for each Nsight tool are maintained on the NVIDIA Developer website. Use the pages in the following table to retrieve the latest build for your development system.

2.6.4.2 Latest Versions

Tool	NVIDIA Developer Page
Nsight Systems	Get started with Nsight Systems
Nsight Graphics	Get started with Nsight Graphics
Nsight DL Designer	Get started with Nsight DL Designer
Nsight Compute	Get started with Nsight Compute

2.6.5. NVIDIA Container Runtime for Docker

2.6.5.1 Overview

The NVIDIA Container Runtime enables Docker containers to access GPU resources on DGX Spark systems. It provides hooks based on the Open Container Initiative (OCI) specification, which is an open standard for container runtimes. These OCI hooks are triggered when the `--gpus` flag is used, ensuring GPU devices are made available inside the container. The runtime acts as a bridge between Docker and the NVIDIA drivers, allowing containers to utilize GPU acceleration for AI/ML workloads, CUDA applications, and other GPU-accelerated software.

Key benefits: - Seamless GPU access within containers - Automatic driver and library management - Support for multi-GPU configurations - Compatibility with popular container orchestration platforms

The runtime works in conjunction with the NVIDIA Container Toolkit, which provides the necessary components to expose GPU devices and CUDA libraries to containerized applications.

2.6.5.2 Installation

The NVIDIA Container Toolkit is preinstalled and configured on DGX Spark systems. This includes:

- NVIDIA Container Runtime
- Docker integration
- GPU device access configuration
- CUDA library management

The runtime is ready to use out of the box for running GPU-accelerated containers.

2.6.5.2.1 Optional: Add User to Docker Group

By default, Docker requires `sudo` privileges to run commands. Adding your user to the docker group allows you to run Docker commands without `sudo`, which provides:

- **Convenience**: No need to type *sudo* before every Docker command
- **Better workflow**: Seamless integration with development tools and scripts
- **Reduced friction**: Faster iteration when working with containers

To add your user to the docker group:

```
sudo usermod -aG docker $USER
newgrp docker
```

> **Note**
>
> This step is optional. You can continue using Docker with `sudo` if you prefer not to modify group memberships.

2.6.5.3 Usage

2.6.5.3.1 Basic GPU Access

Run a container with GPU access using the `--gpus` flag:

```
docker run -it --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.04 nvidia-
↪smi
```

This command: - Runs an interactive container (`-it`) - Enables access to all GPUs (`--gpus=all`) - Uses the NVIDIA CUDA development image - Executes `nvidia-smi` to display GPU information

2.6.5.3.2 Set GPU Capabilities

Control which GPU capabilities are available to the container:

```
docker run -it --gpus '"capabilities=compute,utility"' nvcr.io/nvidia/cuda:13.
↪0.1-devel-ubuntu24.04 nvidia-smi
```

2.6.5.4 Validation

2.6.5.4.1 Test GPU Access

Run the test command to verify GPU access:

```
docker run -it --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.04 nvidia-
↪smi
```

Expected output should show:

- GPU device information
- Driver version
- CUDA version
- Memory usage and temperature

2.6.5.5 Troubleshooting

2.6.5.5.1 Runtime Not Found

If you encounter “runtime not found” errors:

1. Verify NVIDIA Container Toolkit is installed:

   ```
   nvidia-ctk --version
   ```

2. Check Docker daemon configuration:

   ```
   cat /etc/docker/daemon.json
   ```

3. Restart Docker service:

   ```
   sudo systemctl restart docker
   ```

2.6.5.5.2 Driver/Container CUDA Mismatch

If you see CUDA version mismatches:

1. Check host CUDA driver version:

   ```
   nvidia-smi
   ```

2. Use a container image with compatible CUDA version:

   ```
   docker run -it --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.04
   ↪nvidia-smi
   ```

2.6.5.5.3 Permission Issues

If you encounter permission errors:

1. Ensure your user is in the docker group (if not using sudo):

```
groups $USER
```

2. Check device permissions:

```
ls -la /dev/nvidia*
```

3. Verify Docker daemon has access to GPU devices:

```
sudo docker run -it --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.
↪04 nvidia-smi
```

2.6.5.5.4 Container Startup Issues

If containers fail to start:

1. Check Docker logs:

```
docker logs <container_id>
```

2. Verify GPU devices are available on host:

```
ls /dev/nvidia*
```

3. Test with a minimal container:

```
docker run --rm --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.04
↪echo "GPU test successful"
```

2.6.6. NGC

2.6.6.1 Overview

NVIDIA GPU Cloud (NGC) is a comprehensive registry of GPU-optimized containers, pre-trained models, and AI/ML software that enables rapid development and deployment of AI applications. For DGX Spark users, NGC provides access to the latest frameworks, tools, and optimized environments specifically designed for the Grace Blackwell architecture.

Key benefits for DGX Spark users:

- **Optimized Containers**: Pre-configured environments with the latest AI/ML frameworks, CUDA, and libraries optimized for Grace Blackwell GPUs
- **Pre-trained Models**: Access to state-of-the-art models and model collections for various AI tasks
- **Rapid Development**: Skip complex environment setup and focus on your AI/ML projects
- **Cutting-edge Software**: Access to the latest NVIDIA software stack and experimental features

NGC is particularly valuable for DGX Spark users because it provides the most current and optimized software stack for this new platform, ensuring you have access to the latest performance optimizations and features.

2.6.6.2 Getting Started

2.6.6.2.1 Create an NGC Account

1. Visit the NGC website
2. Click **Sign Up** and create a free account
3. Verify your email address
4. Complete your profile information

2.6.6.2.2 Generate an API Key

1. Log in to your NGC account
2. Navigate to **Setup** -> **API Key**
3. Click **Generate API Key**
4. Copy and securely store your API key

> **Note**
>
> Your API key is required for pulling containers and accessing NGC resources. Keep it secure and never share it publicly.

2.6.6.2.3 Install NGC CLI (Optional)

The NGC CLI provides convenient command-line access to NGC resources. Your DGX Spark system requires the ARM64 version of the NGC CLI which can be found on the **ARM64 Linux** tab at https://org.ngc.nvidia.com/setup/installers/cli.

For more information on installing and using the NGC CLI, see NGC CLI Documentation.

2.6.6.2.4 Authenticate with Docker

Configure Docker to access NGC registries:

```
# Login to NGC with Docker
docker login nvcr.io
# Username: $oauthtoken
# Password: <your-api-key>
```

2.6.6.3 Basic Usage

2.6.6.3.1 Pull and Run a Container

Start with a popular AI/ML framework container:

```
# Pull a PyTorch container optimized for Grace Blackwell
docker pull nvcr.io/nvidia/pytorch:24.08-py3

# Run the container with GPU access
docker run -it --gpus=all nvcr.io/nvidia/pytorch:24.08-py3
```

2.6.6.3.2 Explore Available Resources

Browse NGC resources through the web interface:

- **Containers**: AI/ML frameworks, development environments, and specialized tools
- **Models**: Pre-trained models for computer vision, natural language processing, and more
- **Helm Charts**: Kubernetes deployment configurations
- **Jupyter Notebooks**: Interactive tutorials and examples

2.6.6.4 Common Workflows

2.6.6.4.1 Development Environment

Use NGC containers as your development environment:

```
# Run a development container with persistent storage
docker run -it --gpus=all \
  -v /path/to/your/project:/workspace \
  nvcr.io/nvidia/pytorch:24.08-py3
```

2.6.6.4.2 Model Inference and Training

Access pre-trained models and training scripts:

```
# Pull a model from NGC
ngc registry model download-version "nvidia/nemo/bertbaseuncased:1.0.0rc1"

# Or use models directly in containers
docker run -it --gpus=all \
  nvcr.io/nvidia/pytorch:24.08-py3
```

2.6.6.5 Best Practices

2.6.6.5.1 Container Management

- **Pin Versions**: Use specific container tags for reproducible environments
- **Regular Updates**: Periodically update to newer container versions for latest optimizations
- **Resource Limits**: Set appropriate memory and CPU limits for your workloads

2.6.6.5.2 Data Persistence

- **Volume Mounts**: Mount your data directories into containers for persistence
- **Model Storage**: Store trained models and checkpoints outside containers
- **Configuration**: Keep configuration files in version control

2.6.6.5.3 Security

- **API Key Security**: Store your NGC API key securely and rotate it regularly
- **Container Scanning**: Scan containers for vulnerabilities before use
- **Network Security**: Use appropriate network configurations for your environment

2.6.6.6 Troubleshooting

2.6.6.6.1 Common Issues

Authentication Failures

```
# Verify your API key is correct
docker login nvcr.io
# Check if your account has access to the requested resource
```

Container Pull Issues

```
# Check network connectivity to NGC registry
curl -I https://ngc.nvidia.com

# View container catalog on NGC website
# Visit https://catalog.ngc.nvidia.com/containers

# Or use NGC CLI to list available containers
ngc registry image list nvidia/*

# Try pulling with verbose output to see specific error
docker pull nvcr.io/nvidia/pytorch:24.08-py3
```

GPU Access Problems

```
# Verify NVIDIA Container Runtime is installed
docker run --rm --gpus=all nvcr.io/nvidia/cuda:13.0.1-devel-ubuntu24.04
↪nvidia-smi
```

2.6.6.6.2 Getting Help

- **NGC Documentation**: Visit the NGC documentation
- **Community Forums**: Join the NVIDIA Developer Forums
- **Additional Support**: For support options, see *Get the Right Support for Your DGX Spark*

2.6.7. NVIDIA AI Enterprise—DGX Spark Quick Start Guide

NVIDIA AI Enterprise—DGX Spark is an enterprise-grade software platform for AI development, deployment, and optimization on DGX Spark and NVIDIA GB10 Grace Blackwell Superchip-based systems. It provides libraries, frameworks, microservices, drivers, and enterprise support to accelerate inference, training, customization, observability, and guardrailing.

2.6.7.1 Set Up Your DGX Spark System

Before accessing NVIDIA AI Enterprise—DGX Spark, you must complete the initial setup of your DGX Spark system. Follow the steps in *First Boot and Initial Configuration*. After your system is powered on and connected to your network, continue with the steps in this guide.

2.6.7.2 Optional: NVIDIA AI Enterprise—DGX Spark Evaluation

If you would like to try NVIDIA AI Enterprise—DGX Spark before purchasing, NVIDIA offers a 90-day trial of NVIDIA AI Enterprise. You can register here.

2.6.7.3 Order Confirmation Message

After your order is processed, you will receive an order confirmation message from NVIDIA containing the information required to access NVIDIA AI Enterprise software and enterprise support.

Your NVIDIA Entitlement Certificate is attached to the message and provides instructions for using the certificate.

If you manage software access or licensing in your organization, follow the instructions in the NVIDIA Entitlement Certificate. Otherwise, forward the order confirmation message and NVIDIA Entitlement Certificate to your IT administrator.

2.6.7.4 Create Your NVIDIA AI Enterprise Account

To access NVIDIA AI Enterprise—DGX Spark and technical support, you must have an NVIDIA Cloud Account (NVIDIA AI Enterprise Account), which provides access to:

- **NVIDIA NGC** - Access to NVIDIA AI Enterprise—DGX Spark software component.
- **NVIDIA Enterprise Support Portal** - Access to support services.

Both can be accessed from the NVIDIA Application Hub.

Before you begin, have your order confirmation message available.

1. In the NVIDIA Entitlement Certificate instructions, follow the **Register** link.
2. Fill out the form on the NVIDIA Enterprise Account Registration page and click **REGISTER**. You will receive an email with instructions to log in to the NVIDIA Application Hub.
3. Open the email and click **Log In**.
4. On the Login page, enter your email address and click **Sign In**.
5. On the Create Your Account page, provide and confirm a password, then click **Create Account**. You will receive an email to verify your address.
6. Open the verification email and click **Verify Email Address**.

You can now log in to the NVIDIA Application Hub to access NVIDIA NGC and the NVIDIA Enterprise Support Portal.

2.6.7.5 Access NVIDIA AI Enterprise—DGX Spark Software Asset

1. Go to the NVIDIA NGC Catalog and log in if prompted.
2. On the **NVIDIA NGC Search** page, set the **DGX Spark** filter or look for the **DGX Spark Collection**.
3. Click the software asset to learn more or download it.

2.7. Common Use Cases

The DGX Spark is designed to support a wide range of AI, machine learning, and data science workflows. Visit https://build.nvidia.com/spark for practical guides to help you get started. That site is

regularly updated with new content and information and will be the single source of truth for your Spark device.

2.8. PXE Boot Setup

The DGX Spark UEFI BIOS supports PXE boot. Several manual customization steps are required to get PXE to boot the DGX OS image or the DGX Spark recovery image.

Caution

This document is meant to be used as a reference. Explicit instructions are not given to configure the DHCP, HTTP, and TFTP servers. The end user's IT team is expected to configure these servers to fit their company's environment and security guidelines.

2.8.1. Requirements

- TFTP server
  - Software that provides TFTP service.
- HTTP server
  - An HTTP server is used to transfer large files, such as the iso image and `initrd`. alternatively, TFTP can be used for this purpose. HTTP is used in the example below.
- DHCP server
  - Software that provides dynamic host configuration protocol (DHCP) service.

Note

The TFTP server, HTTP server, and DHCP server can all be configured on the same system, or they can each be on different systems.

- linux bootloader
- ip address: `<ftp ip>`
- fully qualified host: `<ftp host>`

This topic provides some guidance concerning how to set up a PXE boot environment for DGX systems. For complete details, refer to online documentation for setting up a PXE boot server. In this example, `xinetd` is used to provide TFTP service; `dnsmasq` is used to provide DHCP service; and `syslinux` is used as the bootloader.

2.8.2. Overview of the PXE Server

The PXE server requires configuration in the following areas:

- bootloader (grub)

- TFTP contents (the kernel and `initrd`)

  In this example, TFTP is configured to serve files from `/local/tftp/`. You will need to configure your TFTP server to serve files from `/local/tftp` or the directory you desire to use.

- HTTP contents (the iso image)

  In this example, HTTP is configured to serve files from `/local/http/`. You will need to configure your HTTP server to serve files from `/local/http` or the directory you desire to use.

- DHCP

2.8.2.1 PXE Server Configuration for DGX OS Image

In this example, the directory structure on the HTTP and TFTP server looks like this:

```
/local/
   http/
      base_os_7.0.0/
         base_os_7.0.0.iso

   tftp/
      baseos/
         vmlinuz
         initrd
      grub/
         grub.cfg
      grubnetaa64.efi.signed
```

Note

The `vmlinuz` and `initrd` files are specified relative to the TFTP root, `/local/tftp/`; and the location of the ISO, `base_os_7.0.0.iso`, is relative to the HTTP root, `/local/http/`.

Here, the DHCP and PXE servers are configured to use the above directory structure. The person responsible for deploying the PXE environment should change the directory names and structure to fit their infrastructure.

You can set up the directory structure on your HTTP and TFTP server similarly.

The contents of the /local/tftp/grub/grub.cfg file should look something like this:

```
set default=0
set timeout=5

menuentry 'Install BaseOS 7.0.0' {
   linux /baseos/vmlinuz fsck.mode=skip autoinstall ip=dhcp url=http://
↪<Server IP>/base_os_7.0.0/base_os_7.0.0.iso nvme-core.multipath=n nouveau.
↪modeset=0
   initrd /baseos/initrd
}
```

Note

The kernel boot parameters should match the contents of the corresponding ISO's boot menu found in `/mnt/boot/grub/grub.cfg`.

When the system being installed boots via PXE, boot files located on `/local/tftp` are retrieved from the TFTP server. (In this example, the TFTP server is provided by the `xinetd` service whose configuration file, `/etc/xinetd.d/tftp`, specifies that the boot files are located on `/local/tftp`.) When a system is PXE booted, the `grubnetaa64.efi.signedi` file that is designated in the DHCP server's `dhcpd.conf` file is retrieved by TFTP transfer (see *Configure Your DHCP Server*). By default, after the `grubnetaa64.efi.signed` is booted, the PXE boot `grub.cfg` file, `grub/grub.cfg` in this example, provides menu options for booting further. The PXE boot `grub.cfg` config file specifies the locations of the kernel and `initrd` files relative to the tftp directory.

- Configure the HTTP Directory:

  Configure the HTTP file directory and ISO image by placing a copy of the Base OS 7.0.0 ISO in directory `/local/http/base_os_7.0.0/`. In this example, the full path is `/local/http/base_os_7.0.0/base_os_7.0.0.iso`.

- Configure the TFTP Directory By Using the Following Steps:

  Mount the Base OS 7.0.0 ISO. Assume your mount point is `/mnt`:

  ```
  sudo mount -o loop /local/http/base_os_7.0.0/base_os_7.0.0.iso /mnt
  ```

  Copy the kernel and `initrd` from the ISO to the tftp directory:

  ```
  cp /mnt/casper/vmlinuz /local/tftp/baseos/vmlinuz
  cp /mnt/casper/initrd /local/tftp/baseos/initrd
  ```

  Unmount the Base OS 7.0.0 ISO:

  ```
  umount /mnt
  ```

- Download GRUB binaries and Copy it for PXE Booting into Place:

  ```
  cd /tmp
  wget -q http://ports.ubuntu.com/ubuntu-ports/dists/jammy/main/uefi/grub2-
   ↪arm64/current/grubnetaa64.efi.signed
  cp grubnetaa64.efi.signed /local/tftp/
  ```

2.8.2.2 PXE Server Configuration for Recovery Media

In this example, the directory structure on the HTTP and TFTP server looks like this:

```
/local/
   http/
      fastos/
         usb.customer.tar.gz
   tftp/
      fastos/
         vmlinuz
         initrd
      grub/
         grub.cfg
      grubnetaa64.efi.signed
```

Note

The `vmlinuz` and `initrd` files are specified relative to the TFTP root, `/local/tftp/`; and the location of the ISO, `usb.customer.tar.gz`, is relative to the HTTP root, `/local/http/`.

Here, the DHCP and PXE servers are configured to use the above directory structure. The person responsible for deploying the PXE environment should change the directory names and structure to fit their infrastructure.

You can set up the directory structure on your HTTP and TFTP server similarly.

The contents of the /local/tftp/grub/grub.cfg file should look something like this:

```
set default=0
set timeout=5

menuentry "Install DGX Spark FastOS" {
    linux /fastos/vmlinuz nouveau.modeset=0 console=tty0 console=ttyS0,921600
 ↪sbsa_gwdt.action=1 noui pxeinstall=true fastos_usbimg_url=http://<Server IP>
 ↪/fastos/usb.customer.tar.gz ip=dhcp static_ip=<static ip>:<gateway ip>
    initrd /fastos/initrd
}
```

When the system being installed boots via PXE, boot files located on `/local/tftp` are retrieved from the TFTP server. (In this example, the TFTP server is provided by the `xinetd` service whose configuration file, `/etc/xinetd.d/tftp`, specifies that the boot files are located on `/local/tftp`.) When a system is PXE booted, the `grubnetaa64.efi.signedi` file that is designated in the DHCP server's `dhcpd.conf` file is retrieved by TFTP transfer (see *Configure Your DHCP Server*). By default, after the `grubnetaa64.efi.signed` is booted, the PXE boot `grub.cfg` file, `grub/grub.cfg` in this example, provides menu options for booting further. The PXE boot `grub.cfg` config file specifies the locations of the kernel and `initrd` files relative to the tftp directory.

- Configure the HTTP Directory:

  Configure the HTTP file directory and ISO image by placing a copy of the DGX OS exactly as `/local/http/usb.customer.tar.gz`. Download the recovery media archive file (tar.gz format) from https://developer.nvidia.com/downloads/dgx-spark/dgx-spark-recovery-image-1.135.29.tar.gz and change its name to `usb.customer.tar.gz`.

- Configure the TFTP Directory By Using the Following Steps:

  Untar the `usb.customer.tar.gz` file to the `/tmp` directory:

  ```
  tar xpfv /tmp/usb.customer.tar.gz
  ```

  Copy the kernel and `initrd` from the tarball to the tftp directory:

  ```
  cp /tmp/usbimg.customer/usb/vmlinuz /local/tftp/fastos/
  cp /tmp/usbimg.customer/usb/initrd /local/tftp/fastos/
  ```

- Download GRUB binaries and Copy it for PXE Booting into Place:

  ```
  cd /tmp
  wget -q http://ports.ubuntu.com/ubuntu-ports/dists/jammy/main/uefi/grub2-
   ↪arm64/current/grubnetaa64.efi.signed
  cp grubnetaa64.efi.signed /local/tftp/
  ```

2.8.3. TFTP and HTTP Server Verification

After you have set up all elements of your PXE server, prior to doing a PXE install, verify that the TFTP and HTTP servers are working properly.

TFTP Server Verification

To verify that the TFTP server has been set up correctly, from a different system on the same subnet, use `tftp` to get one of the files that will be obtained via `tftp` during the PXE boot. In this example, the TFTP server has been set up to serve files from `/local/tftp`. The grub configuration file, `grub.cfg` is located on `/local/tftp/grub/grub.cfg`; therefore, from the TFTP command prompt, request grub.cfg via `get grub/grub.cfg`.

```
cd /tmp
tftp <TFTP_Server_IP>
get grub/grub.cfg
quit
```

HTTP Server Verification

To verify that the HTTP server has been set up correctly, use the `wget` command to get one of the files that will be obtained via HTTP during the PXE boot. In this example, the HTTP server has been set up to serve files from `/local/http`. The tarball, `usb.customer.tar.gz`, is located on `/local/http/fastos/usb.customer.tar.gz`; therefore, test the HTTP request to retrieve `usb.customer.tar.gz` by running the following commands:

```
cd /tmp
wget http://<HTTP Server IP>/fastos/usb.customer.tar.gz
```

2.8.4. Useful Parameters for Configuring Your System's Network Interfaces

`static_ip=<static ip>:<gateway ip>`: tells the initramfs to configure the system's interfaces using a static IP address.

2.8.5. Parameters Unique to the Spark OS Installer

- `usb.skipfw`: Skips the firmware update step during the installation.
- `usb.shell`: Starts a shell instead of installing the OS.
- `noui`: Disables the user interface during the installation.
- `autoinstall`: Enables the autoinstall feature during the installation.

2.8.6. Configure Your DHCP Server

The DHCP server is responsible for providing the IP address of the TFTP server and the name of the bootloader file in addition to the usual role of providing dynamic IP addresses. The address of the TFTP server is specified in the DHCP configuration file as `next-server`, and the bootloader file is specified as `filename`. The architecture option can be used to detect the architecture of the client system and to serve the correct version of the grub bootloader (x86, IA-32, ARM, and so on).

An example of the PXE portion of `dhcpd.conf` is:

```
class "pxeclients" {
  match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
    filename "grubnetaa64.efi.signed";
    next-server <TFTP_Server_IP>;
    option root-path "/local/tftp";
}
```

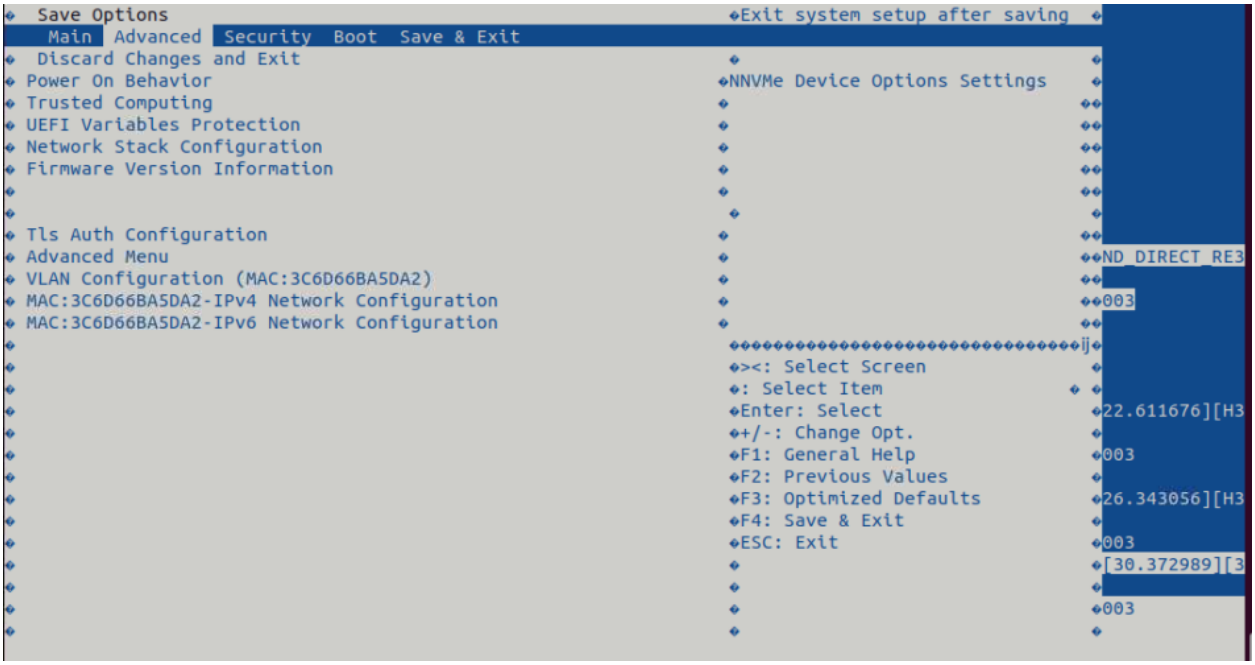
For example, this is a dhcpd.conf file:

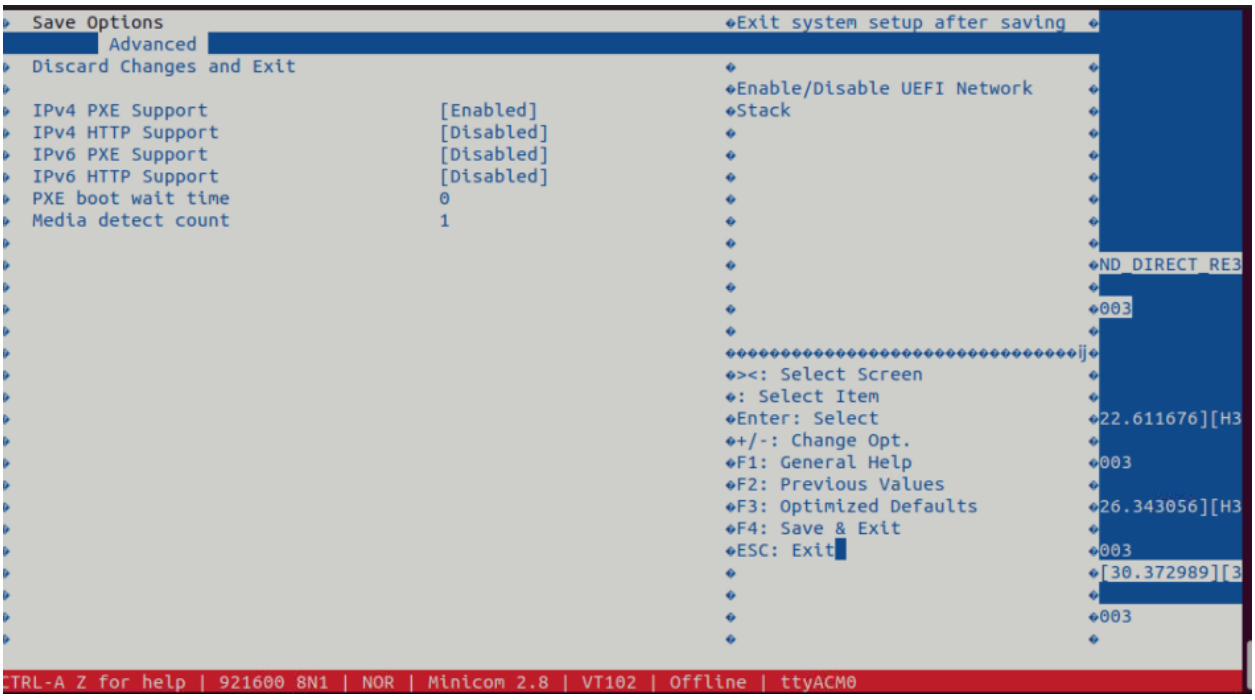
```
authoritative;
default-lease-time 120;
max-lease-time 120;
ddns-update-style none;


subnet 192.168.99.0 netmask 255.255.255.0 {
pool {
   range 192.168.99.2;

# Required for PXE Boot
class "pxeclients" {
   match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
      filename "grubnetaa64.efi.signed";
      # TFTP Server IP address
      next-server 192.168.99.1;
      option root-path "/local/tftp/";
}

}
}
```

2.8.7. Enable PXE Boot on the DGX Spark

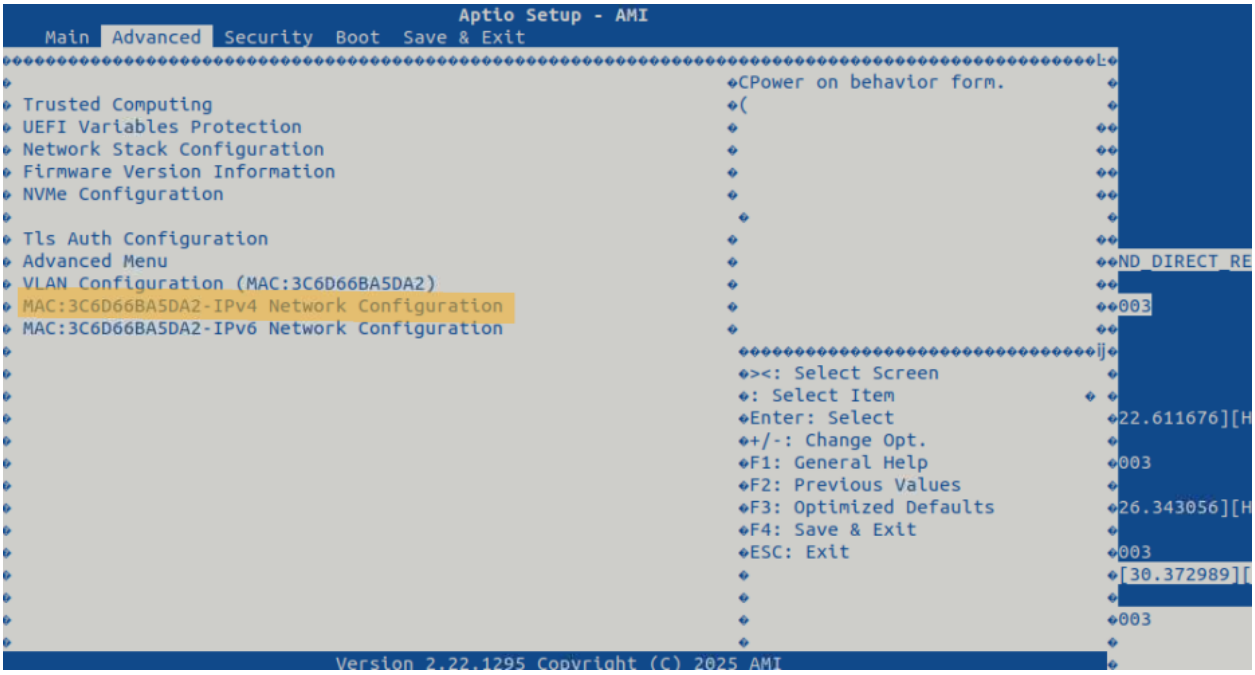
Before enabling PXE Boot, you need to disable the secure boot feature in the UEFI menu or enroll the grubnetaa64.efi.signed in the UEFI menu.

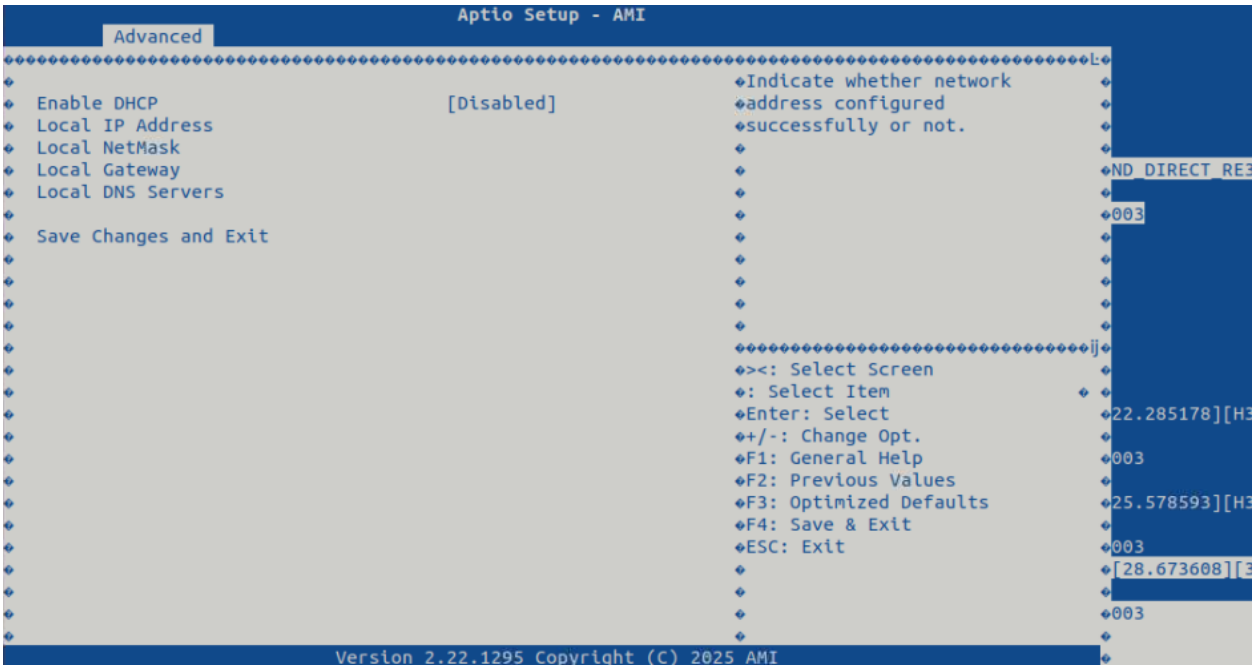
Enable PXE Boot on the DGX Spark by enabling it in `Network Stack Configuration` menu.





Select the NIC you want to use for PXE Boot and enable PXE Boot.





Save and Exit.

After you finished, you can use the UEFI boot menu to do PXE boot and modify the boot order. See https://docs.nvidia.com/dgx/dgx-spark-uefi/boot-tab.html.

2.9. Enterprise Manageability

Enterprise IT teams that operate DGX Spark systems at scale can use the following sections for fleet lifecycle integration and for custom installation when removable media or on-premises mirrors replace direct use of public package sources.

Note

Enterprise lifecycle integration scripts. Procedures in *Enterprise Lifecycle Integration* reference a downloadable collection of example Python tools and shell scripts that are not preinstalled on DGX Spark systems. Use them as reference material and adapt them within your own enterprise management platforms. Download Enterprise Lifecycle Integration Scripts package (ZIP) and extract the archive on your integration host, then follow `README.md` in the extracted tree. Optional `install.sh` scripts are included if you choose to deploy examples on endpoints.

2.9.1. Enterprise Lifecycle Integration

2.9.1.1 How to Use This Information

This section provides information on the anticipated uses and limitations of this information.

2.9.1.1.1 Intended Audience

This information is written for the following audience:

- Enterprise IT administrators and endpoint operations teams
- Platform engineering or SRE teams operating fleets at scale
- Security operations teams that require evidence-driven workflows
- Support and escalation teams that need standardized diagnostics artifacts

2.9.1.1.2 Document Scope

The following are within the scope of this document.

- Lifecycle-driven operational playbooks. For example, the following workflow:

Procure Provision Monitor Maintain Respond Retire

- Agentless remote execution model (SSH) with a strict stdout JSON contract
- Evidence minimization and artifact handling patterns
- Wrapper patterns for enterprise platforms (for example, job systems, orchestration, and configuration management)

The following are not within the scope of this document.

- Prescriptive identity architecture (for example, IAM design, key management, and PKI)
- Organization-specific network segmentation and firewall policy
- Business or legal policy decisions (such as retention durations and legal hold workflows)
- Detailed application-layer configuration beyond manageability requirements
- Detailed ISO repack, USB `OEMDATA` layout, and Cloud-init seed procedures (see *Custom Installation with Cloud-Init*)

2.9.1.1.3 Document Conventions

This document uses the following conventions:

- Collectors are read-only tools safe to run frequently and at high concurrency.
- Controllers change state and must be gated by changing windows, rings or waves, and validation.
- Tools return exactly one bounded JSON document on stdout per invocation.
- Large outputs and deep evidence are stored as artifacts; stdout returns pointers only.

When you see inline code, it represents literal strings, such as file names, flags, fields, or paths.

2.9.1.1.4 Operational Guardrails

Note: Read this before running controllers.

The following are guardrails to consider.

- Run controllers only in approved change windows with staged rollouts (for example pilot waves broad).
- Keep stdout JSON bounded to avoid platform truncation.
- Treat artifact creation as on-demand; store and retain artifacts per policy.
- Separate transport or auth failures (platform layer) from tool failures and endpoint health failures.

2.9.1.2 Support Boundaries and Operational Responsibilities

This section clarifies what is provided and supported as part of the DGX Spark manageability baseline versus what remains the responsibility of the enterprise IT environment. It is intended to reduce ambiguity during deployment, escalation, and long-term operations.

2.9.1.2.1 Supported Baseline Posture

2.9.1.2.1.1 Supported Baseline (high-level)

DGX Spark enterprise manageability is designed and validated against the supported baseline OS image (DGX OS).

Baseline Topic	Guidance
OS baseline	DGX OS is the supported baseline for the guide's operational contract.
Driver stack	Driver behavior, versions, and compatible combinations are validated relative to the baseline.
Firmware alignment	Firmware inventory and update posture are interpreted relative to the baseline and platform capabilities.
Tooling	Production tools installed into DGX_spark_management/bin/ are intended for fleet automation.
Reference scripts	Landscape scripts are reference implementations and may require adaptation for production governance.

2.9.1.2.1.2 Shipped Software Baseline and SBOM Source

For DGX Spark Founders Edition, customers who require a software inventory or SBOM can generate one from the DGX Spark System Recovery Image. The recovery image represents the recovery media to reset to factory settings for the latest software image.

For DGX Spark systems from other manufacturers, use the recovery image or software delivery process provided by the respective OEM. OEM variant images might differ from the DGX Spark Founders Edition recovery baseline.

For the recovery media download and recovery procedure, refer to *System Recovery*. For current component version information, refer to *DGX Spark Release Notes*.

2.9.1.2.2 Management Platform Support Posture

Canonical Landscape is the primary recommended management platform for DGX Spark, and its license is included as part of the Ubuntu Pro entitlement. Landscape provides user management, policy deployment, and OS, firmware, and driver updates from a local repository. For enrollment steps, see *Ubuntu Pro and Landscape client enrollment*.

In addition, ecosystem tools such as Ansible, Puppet, Tanium, and others may offer ARM64-capable agents that can be used for broader fleet management, including approval workflows and rollback strategies. These alternatives are supported solely by their respective vendors. Any integration examples referenced in this document do not imply NVIDIA support or validation.

2.9.1.2.3 Reimaging Posture

Reimaging may be required in certain customer workflows, but it has consequences for predictability and supportability.

2.9.1.2.3.1 Reimaging Posture

Reimaging away from the supported baseline OS can change the behavior across drivers and firmware.

Scenario	Operational Expectation
Baseline DGX OS	Predictable behavior: Spark Manageability Guide playbooks apply directly.
Baseline and approved updates	Predictable within validated update channels; confirm before and after evidence.
Non-baseline OS image	Higher variance; validate collectors and controllers; adjust wrappers and evidence paths as needed.
Mixed fleet baselines	Increase drift and support complexity; maintain baseline identity records per ring or group.

2.9.1.2.4 Division of Responsibility (NVIDIA vs Enterprise IT)

NVIDIA Provides (Typical)	Enterprise IT Owns (Typical)
DGX OS baseline image and guidance	Identity strategy, authentication, RBAC, and secrets management
Reference tools for inventory, diagnostics and update control	Orchestration platform selection, job packaging, scheduling, and rings and waves
Reference scripts demonstrating platform patterns (Landscape)	Change management, approvals, and maintenance windows
Evidence paths and recommended minimization model	Artifact storage, retention policies, legal hold, and ticket linkage
Escalation guidance and support interfaces	Network segmentation, firewall rules, and access pathways
Controller Class	Minimum Governance Expectation
Update control (spark_updatectl.py)	Change window with ring rollout and precheck and postcheck evidence. Rollback awareness is retained.
Diagnostics bundles (spark_diagctl.py bundle modes)	On-demand only. Artifacts are handled through evidence store. Ticket linkage is recommended.

2.9.1.3 DGX Spark Overview, Enterprise Challenges, Lifecycle Backbone, and the JSON/Agentless SSH

DGX Spark is being introduced into environments that already have mature endpoint operations. This section establishes a lifecycle-following framework and a simple operational contract so enterprise tooling can manage DGX Spark consistently at fleet scale.

Key Takeaways

- Treat DGX Spark as a enterprise endpoint with an appliance mindset.
- Standardize on SSH for remote execution and JSON on stdout as the integration contract.
- Use artifacts only when deep evidence is required; keep routine results small and bounded.

2.9.1.3.1 Capabilities Summary

This matrix summarizes the current DGX Spark manageability capabilities exposed by this repository. It is intended to help an Enterprise IT administrator quickly understand the following:

- What exists today and how it is delivered (installed tool vs reference script)
- What evidence is produced (stdout JSON vs artifacts).

How to Read This Table

- **Production (installed)**: Fleet-ready tooling installed into DGX_spark_management/bin/.
- **Reference (in-place)**: Example scripts designed for Canonical Landscape constraints; not installed by default.

- **Evidence model**: stdout JSON is the primary contract; artifacts are generated only when deep evidence is required.

Table 3: DGX Spark manageability capabilities (summary)

Capability area	Integration overview	Delivery	Primary commands	Evidence produced
Identity and asset acceptance	Capture stable identifiers and an "as-received" acceptance snapshot.	Production	`device_identity.py`, `os_build_identity.py`	stdout JSON; optional on-device JSON record
Hardware configuration inventory	Enumerate key hardware configuration (CPU/GPU/SSD/NIC/memor	Production	`hardware_config.py`	stdout JSON; optional on-device JSON record
Firmware inventory	Report firmware versions (UEFI/BIOS, NIC, SSD, GPU as available).	Production	`firmware_reporte.py`	stdout JSON; optional on-device JSON record
Driver inventory	Report key driver versions (GPU/NIC/storage/USB as implemented).	Production	`driver_inventory.py`	stdout JSON; optional on-device JSON record
Software inventory	Enumerate installed software inventories for drift and compliance contexts.	Production	`software_invento.py`	stdout JSON; optional on-device JSON record
UEFI-backed tags (optional)	Read/write UEFI-backed metadata tags for fleet workflows (if enabled).	Production	`NVAIAread`, `NVAIAwrite`	stdout JSON (read/write result); optional on-device record
Health posture and diagnostics (L1)	Return bounded health posture signals for monitoring and triage.	Production	`spark_diagctl.py`	stdout JSON summary
Deep evidence bundles (L2)	Generate targeted or full diagnostics bundles when escalation requires evidence.	Production	`spark_diagctl.py`	stdout JSON + artifact pointer(s)
Reset / reboot context	Explain reboot/reset reasons for stability correlation and incident triage.	Production	`reset_reason_rep.py`	stdout JSON; optional on-device JSON record
Controlled SW/FW updates	Expose update posture and controlled update operations within maintenance windows.	Production	`spark_updatectl.py`	stdout JSON; optional update evidence artifacts depending on mode
Landscape reference execution	Run platform-friendly checks and store per-run evidence under a run directory.	Reference (`landscap` scripts)	`landscape_*` scripts	Short stdout + on-device evidence under `<runid.log>`

2.9.1.3.2 DGX Spark Overview

DGX Spark is deployed as an enterprise-managed, locally accelerated AI companion device, often alongside standard PC fleets, to provide on-prem or edge AI compute where latency, privacy, data gravity, or disconnected operations make cloud-only workflows impractical.

In enterprise terms, DGX Spark behaves less like a general-purpose end-user PC and more like a managed endpoint appliance:

- It runs NVIDIA's base OS (DGX OS).
- It is typically administered remotely, at scale, with minimal local interaction.
- It should integrate into existing IT operations: inventory, monitoring, patching, incident response, and retirement.

Documentation Principle: Treat DGX Spark as a first-class enterprise endpoint but manage it with an appliance mindset: baseline control, automation, and evidence-driven operations.

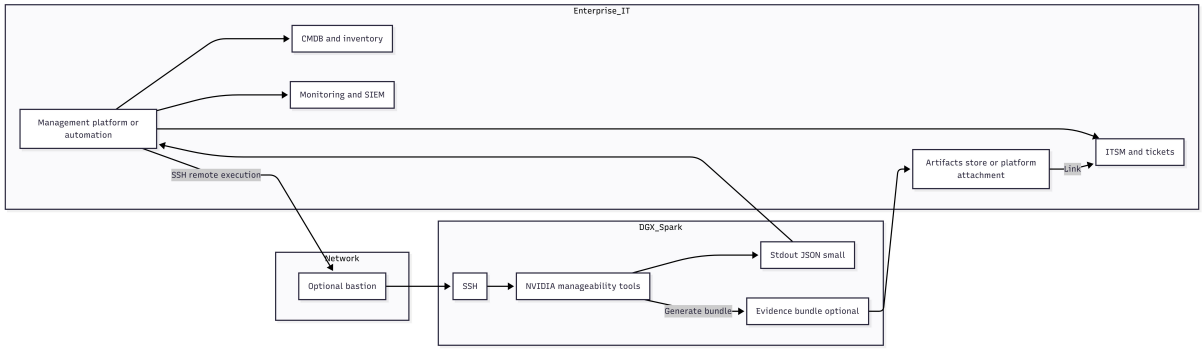


Figure 1: DGX Spark context (PC fleet + AI workloads + enterprise tooling).

2.9.1.3.3 Enterprise Management Challenges

Enterprise IT teams managing PCs globally typically rely on:

- Rich device identity and hardware inventory
- Standardized patching and reboot orchestration
- Predictable driver and firmware servicing channels
- Mature remote diagnostics and support bundles
- CMDB integration, drift detection, and reporting. Linux-based devices often break those assumptions:
- Management approaches differ across distros and package managers
- Endpoint agents vary by platform and may not be desirable in regulated environments
- Ad-hoc scripting without consistent outputs leads to fragile automations
- Troubleshooting evidence collection can be inconsistent, large, and noisy. DGX Spark's manageability approach addresses these challenges by providing:
  - Bounded, machine-ingestible outputs (JSON)
  - An agentless control-plane model (SSH)
  - Optional artifact bundles when deep evidence is required
  - Lifecycle-aligned playbooks that map to standard IT operations

PC-Centric Expectation	DGX Spark Operational Equivalent
Identity and inventory are always queryable	Standard SSH collectors emit bounded JSON snapshots for ingestion.
Patching and reboots are orchestrated	Maintenance-window playbooks: precheck update and reboot postcheck, all returning JSON.
Driver and firmware servicing is predictable	Standardize on a supported baseline (DGX OS) and validate drift through inventory JSON.
Support bundles are standardized	Generate evidence artifacts only when needed and reference them from the JSON result.
Drift detection and reporting are continuous	Record baseline OS build, firmware, drivers, and critical packages. Compare them over time.

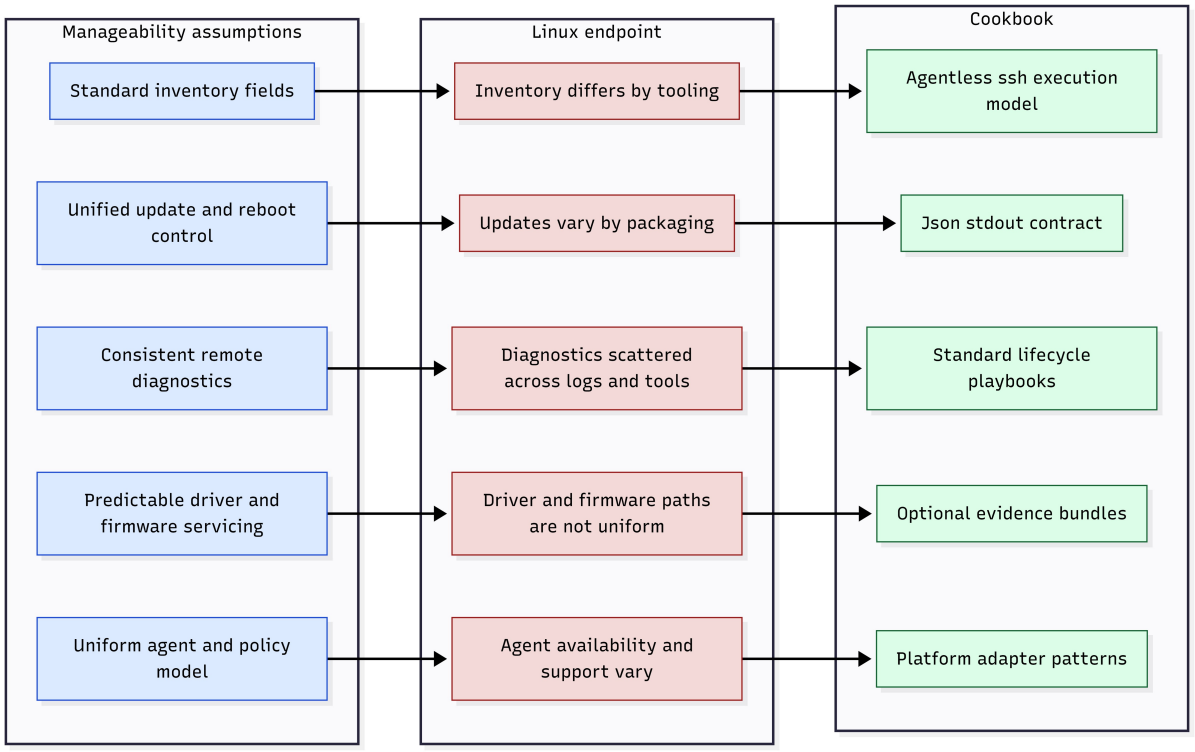


Figure 2: Manageability assumptions vs Linux endpoint realities.

2.9.1.3.4 Lifecycle Backbone (How Enterprise IT Actually Runs Fleets)

The information in this section is organized around a lifecycle backbone that enterprise IT already understands. It deliberately starts before the device is powered on, because procurement decisions (SKUs, support, warranty, spares, standards alignment, and so on) drive long-term operational cost.

Procurement and Receiving

- Define standard SKUs and configurations, support entitlements, and regional logistics.
- Establish asset identity expectations (serial and UUID conventions, labeling, CMDB records).
- Plan sparing, RMA flows, and replacement pools aligned to global sites.

Initial Provisioning

- Establish device identity, hardware, firmware, software inventory, and baseline state.
- Apply initial configuration required for remote management (SSH reachability, time sync, and so on).
- Record enrollment metadata and ownership tags (as defined by your IT processes).
- Apply automated first-boot configuration with Cloud-init and repacked BaseOS media where required. See *Cloud-init for DGX Spark* and *Custom Installation with Cloud-Init*.

Ongoing Monitoring

- Run health checks and alerting signals.
- Detect drift from known good baselines (such as OS build, firmware set, driver set, critical packages).
- Analyze reset reasons and stability indicators to catch systemic issues early.

Maintenance Windows

- Coordinate controlled updates and reboots within change windows.
- Validate update outcomes and preserve rollback safety.
- Enforce staged rollouts (rings and waves) to reduce fleet risk.

Incident Response

- Collect targeted evidence or full diagnostics bundles when needed.
- Package artifacts for escalation (such as, engineering, NVIDIA support, or OEM workflows).
- Execute remediation consistently and record outcomes for postmortems.

End-of-Life

- Execute factory reset aligned to retirement policy.
- Produce retirement evidence (such as method, timestamps, success and failure, and artifact references).
- Handle redeploy, transfer, and disposal with chain-of-custody documentation.

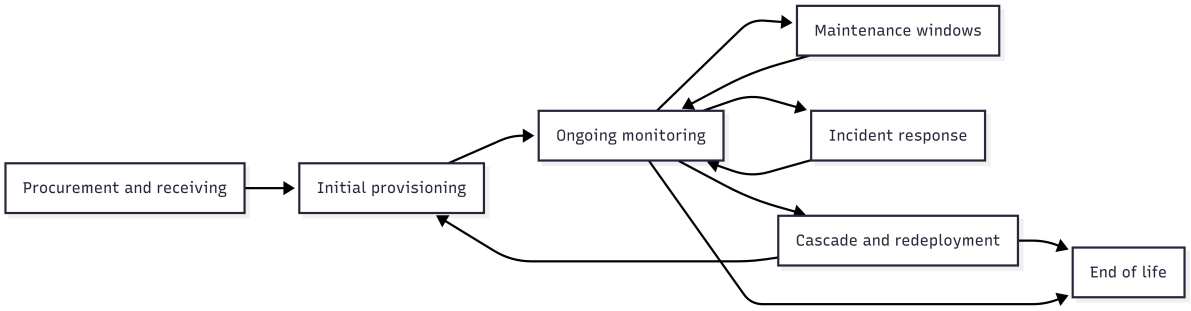


Figure 3: Lifecycle backbone (Procure Provision Monitor Maintain Respond Retire).

2.9.1.3.5 JSON-First, Agentless SSH Execution Model

The information in this section standardizes on one universal remote control plane:

- **Remote execution:** SSH (works from Windows and from enterprise management platforms)
- **Output contract:** JSON on stdout (small, bounded; or machine-ingestible)
- **Deep evidence:** Artifacts (tarballs and log bundles) referenced by JSON and pulled only when needed. This model intentionally avoids requiring a resident management agent on the DGX Spark endpoint.

Existing platforms orchestrate SSH execution and ingest JSON results.

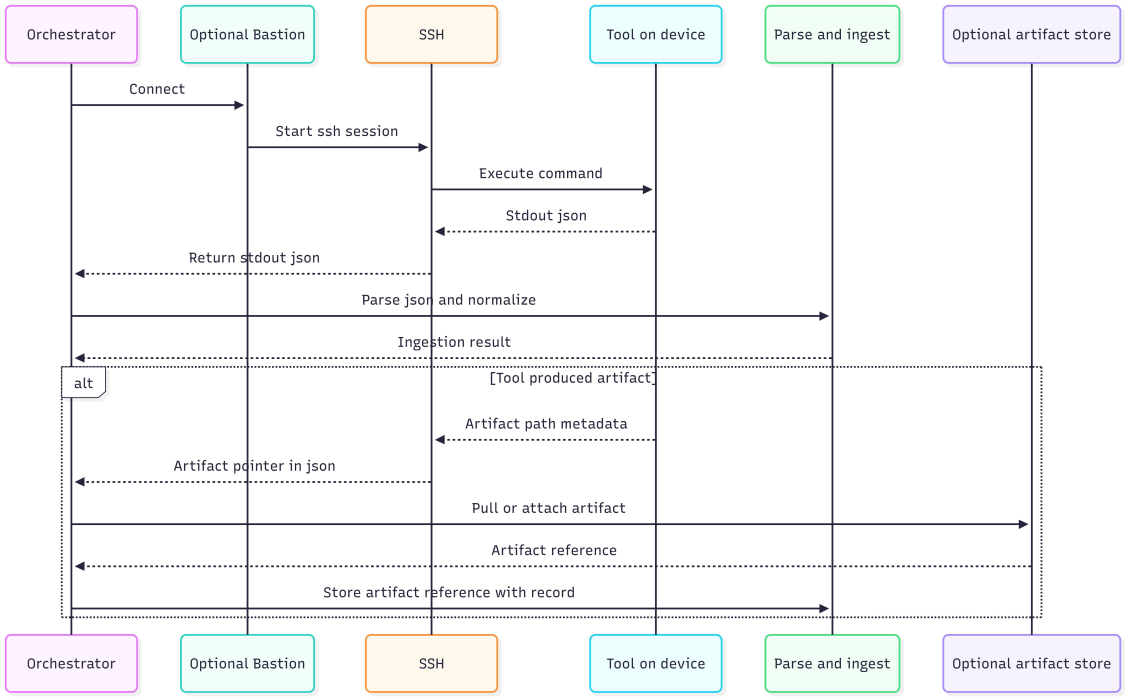


Figure 4: Orchestrator SSH Tool stdout JSON parse or ingest (plus optional artifact pull).

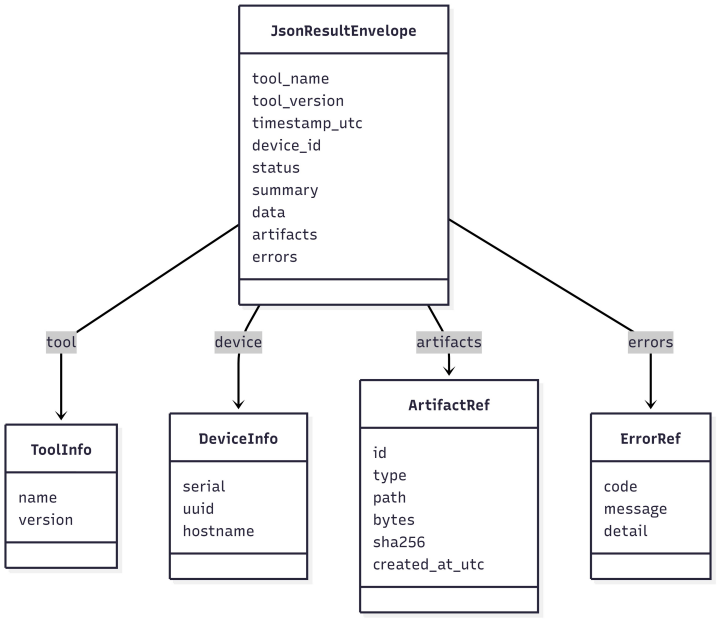


Figure 5: JSON result envelope (fields) and artifact pointer model.

2.9.1.3.6 Simple Operations

IT teams can run the entire control plane from endpoints and servers using built-in OpenSSH capabilities:

- Direct SSH command execution
- Scripting and scheduling
- JSON capture and forwarding into CMDB, SIEM, and monitoring pipelines

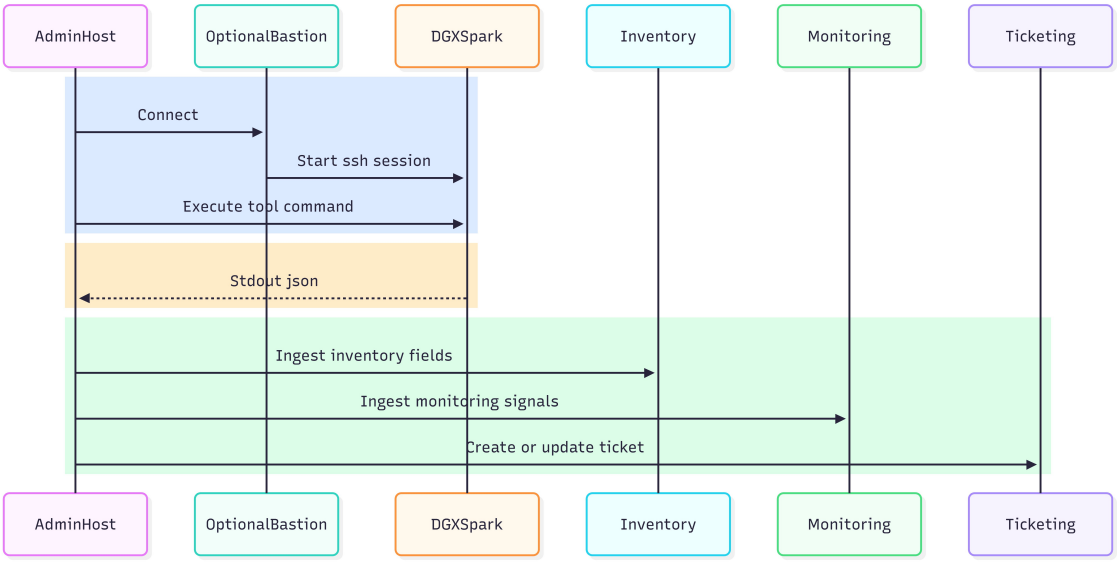


Figure 6: Admin workstation or server SSH DGX Spark JSON ingestion targets.

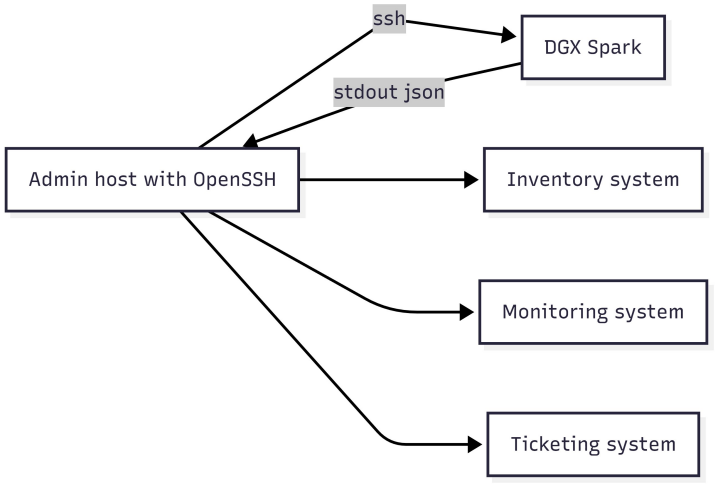


Figure 7: Admin host SSH invocation example flow

2.9.1.3.7 Artifact Strategy

Use two tiers of operational evidence:

- **stdout JSON:** Bounded, predictable, and easy to store and index
- **artifacts:** Collected selectively (incident response and escalation), stored with retention controls; and CMDB stores pointers only

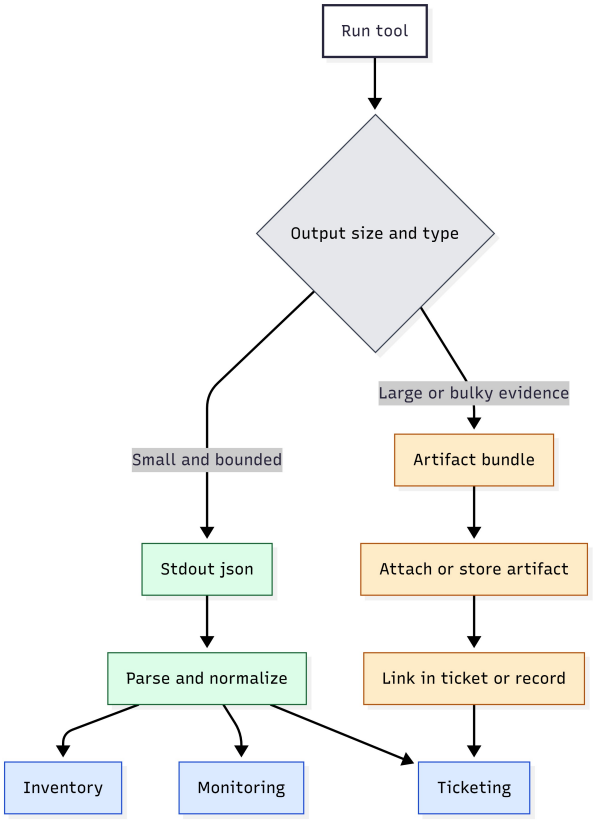


Figure 8: Decision split—small JSON vs large artifacts (pointers + retention).

Artifact Type	Typical Retention	Notes
Routine health snapshots	30–90 days	Prefer JSON-only; avoid artifacts unless needed.
Incident diagnostics bundles	90–180 days	Retain for root cause analysis and escalation cycles.
Security audit evidence	180–365+ days	Align with compliance and legal hold policy.
Retirement proofs	Per policy	Retention period set by chain-of-custody or enterprise governance.

2.9.1.3.8 Operational UX Details

This subsection describes the practical operator experience, such as what to run, what you get back, and how to handle evidence in a way that scales in enterprise job systems.

2.9.1.3.8.1 Operator Workflow in One Screen

- Pick the **lifecycle intent** (such as inventory, drift check, health posture, update window, incident evidence, and retire).
- Run the corresponding **installed command** over SSH.
- Capture **stdout JSON** as the authoritative record.
- Retrieve **artifacts** only when JSON reports they exist or escalation requires deeper evidence.

2.9.1.3.8.2 Quick start: The four most common actions

Intent	Command to Run	What to Store
Acceptance snapshot (as-received)	device_identity.py os_build_identity.py	stdout JSON for CMDB; optional on-device JSON record
Baseline inventories (provisioning)	hardware_config.py firmware_reporter.py driver_i nventory_reporter.py software_i nventory_reporter.py	stdout JSON for drift anchors; retain build + driver + firmware fields
Monitoring (routine posture)	spark_diagctl.py rese t_reason_reporter.py	stdout JSON for health/stability signals
Incident escalation (deep evidence)	spark_diagctl.py (bundle modes)	stdout JSON + artifact pointer(s); retrieve bundle only when needed

2.9.1.3.8.3 Stdout JSON is the API (contract)

Stdout JSON is the integration contract. Recommended capture: stdout (verbatim), rc, stderr, ts, host. See *JSON result capture (stdout as the API)* for the standard envelope fields.

2.9.1.3.8.4 Evidence Handling: Summary First, Artifacts on Demand

Apply the rule in *Artifact Strategy*. If the result is small, keep it in stdout JSON. If the result is large (such as logs or bundles), generate or retrieve it as an artifact. Store artifacts in your normal evidence store and link them back to the run record or ticket.

Scenario	Operational Handling
Routine monitoring	stdout JSON only; run frequently; bounded outputs
Drift investigation	stdout JSON summaries; artifact only if a deep diff is required
Incident response L1	stdout JSON + minimal targeted log excerpts
Incident response L2	stdout JSON + artifact bundle; retrieve and attach to ticket
Retirement evidence	stdout JSON “certificate” + optional artifact evidence per policy

2.9.1.3.8.5 Common Failure Classes

When automation fails at scale, the fastest recovery comes from classifying failures consistently:

Failure Class	What it Usually Means	First Action
Transport/auth	SSH could not connect or authenticate; device unreachable; key or policy issue.	Fix connectivity or credentials in the orchestration layer; rerun a collector.
Privilege	Command requires sudo or access to privileged system interfaces.	Grant least-privilege sudo for the tool; re-run; validate no interactive prompts.
Tool execution	Missing dependency, unexpected OS state, or tool internal error.	Check stderr + tool log directory; compare against supported baseline.
Endpoint degraded	Tool ran successfully but returned warning or failure status due to device health signals.	Collect targeted diagnostics; escalate to L2 bundle if needed.

2.9.1.4 Integration Patterns: SSH Execution from Management Platforms

This section describes how enterprise platforms execute remote actions over SSH, capture results, and feed them into downstream systems. The goal is portability. The same operational pattern should work whether you trigger jobs from Windows, a jump host, or a central orchestration tier.

Key takeaways: See *JSON-first, agentless SSH execution model* and *Artifact Strategy* for the execution and evidence model. Design for fleet realities: output limits, timeouts, concurrency, and predictable privilege boundaries.

2.9.1.4.1 The Universal Remote Execution Model

At scale, enterprise tooling converges on a consistent execution loop:

1. Select targets (static groups, rings, or dynamic inventory queries).
2. Execute a remote command over SSH (often through a job, package, or policy).
3. Capture stdout, stderr, and exit code.
4. Parse stdout JSON into normalized fields.
5. Ingest results into CMDB, monitoring, and ITSM workflows.
6. (Optional) Retrieve artifacts when JSON indicates they exist.

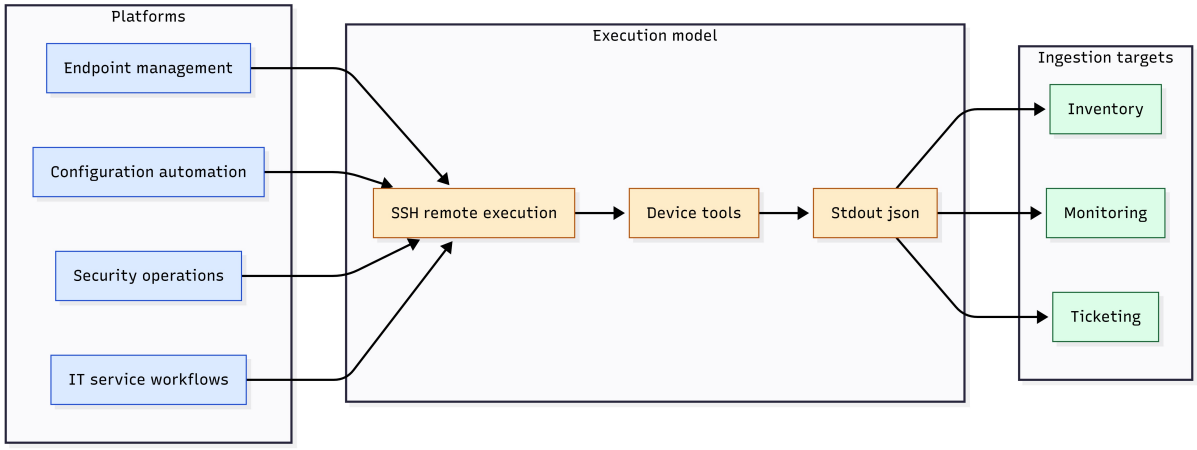


Figure 9: Universal model — platform SSH device tool stdout JSON ingest; optional artifact pull.

2.9.1.4.2 SSH Execution

Many enterprise IT teams prefer a simple operational model.

- Admin devices includes an OpenSSH client for interactive use and automation.
- PowerShell provides a natural surface for fan-out, scheduling, and JSON capture.
- Output should be stored verbatim and parsed off-box.

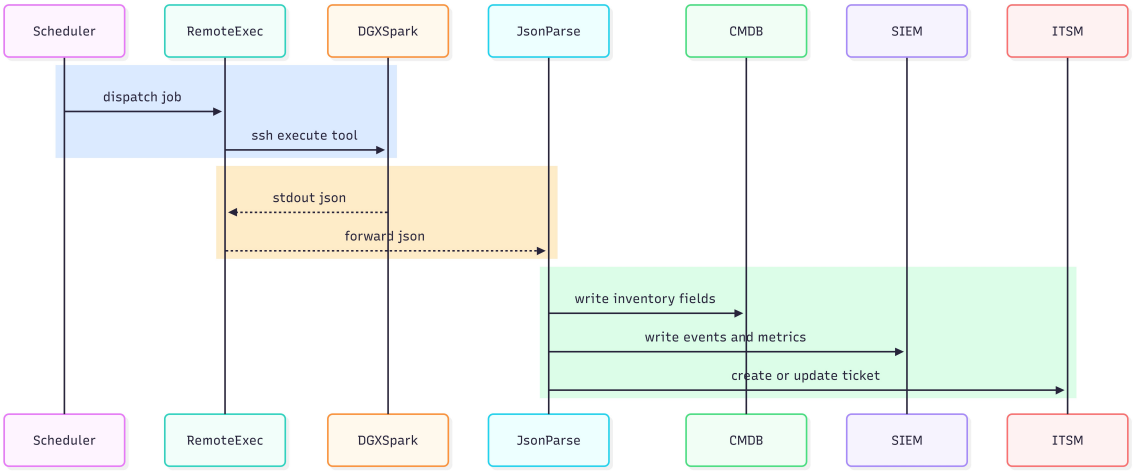


Figure 10: SSH invocation capturing stdout JSON to file.

ssh -o BatchMode=yes -o StrictHostKeyChecking=accept-new nvidia@DGX_HOST “sudo /usr/local/bin/dgx-mgmt/spark_diagctl.py” | Out-File -Encoding utf8 .\ spark_diagctl.py $doc = Get-Content .\ spark_diagctl.py -Raw | ConvertFrom-Json

$doc.status

2.9.1.4.3 SSH Access Patterns for Enterprise Operations (Non-Prescriptive)

The information in this section is intentionally non-prescriptive about identity and IAM. In practice, most enterprises converge on one of these connectivity patterns:

- Direct SSH per device (common in labs and smaller networks).
- Bastion or jump-host mediated SSH (typical for segmented networks).
- Brokered SSH execution (platform runs jobs from a central execution tier).

Operational considerations that matter at fleet scale:

- Stable addressing or an authoritative inventory source
- Non-interactive authentication and execution
- Predictable sudo permissions for tools
- Separation between read-only collectors and state-changing controllers

Pattern	Strengths	Tradeoffs
Direct per-device SSH	Simple and transparent; minimal infrastructure	Harder at scale; network exposure; credential sprawl risk
Bastion or jump host	Fits segmentation; centralizes access control and logging	Capacity planning; potential bottleneck
Brokered execution	Central scheduling, targeting, and reporting; easier fleet automation	Depends on platform constraints; adds abstraction

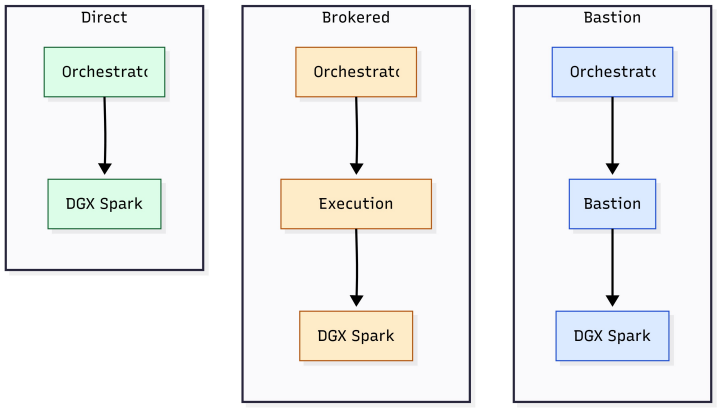


Figure 11: SSH connectivity patterns — direct vs bastion vs brokered execution.

2.9.1.4.4 JSON Result Capture (stdout as the API)

(Model described in *JSON-first, agentless SSH execution model* and *Artifact Strategy*) To keep integrations uniform across platforms, it is assumed that:

- Orchestration wrappers store stdout verbatim (or as close as possible)
- Parsing happens off-box (in the platform, not on the device)

Operational requirements:

- stdout JSON is the authoritative contract
- stderr is retained for debugging, but is not the primary data plane
- If a tool fails before emitting JSON, the wrapper should emit a standard failure envelope

Field	Purpose
tool	Stable identifier of the command or tool invoked
ts	UTC timestamp for correlation
host	Target identity used by the orchestrator (hostname or asset ID)
status	ok
rc	Exit code from the tool execution
duration_ms	Runtime to support SLOs and debugging
summary	Small bounded summary suitable for indexing
warnings	Optional list of non-fatal issues
artifacts	Optional list of artifact pointers (ok, data, errors, meta)

{

“tool”: “spark_diagctl.py “,

“ts”: “2026-01-12T21:17:00Z”,

“host”: “DGX_HOST”,

“status”: “ok”,

“rc”: 0,

“duration_ms”: 842,

“summary”: {

“disk”: “ok”,

“network”: “ok”, “drivers”: “ok”

},

“warnings”: [], “artifacts”: []

}

2.9.1.4.5 Artifact Retrieval Patterns

Artifacts are used when:

- A health signal indicates anomaly
- Incident response requires evidence
- Escalation requires a standardized support bundle
- Audits require retained evidence beyond JSON summaries

Common patterns:

- Pull model: Orchestrator retrieves artifact through scp/sftp after job completion.
- Store-and-link model: Orchestrator uploads artifact to internal object storage and stores a link in the job record or ticket.
- Retention model: Artifacts are retained per policy and then deleted automatically.

Artifact Type	Typical Trigger	Notes
Diagnostics bundle	Incident or escalation	Prefer store-and-link; include hashes; keep sizes predictable
Focused logs	Single failing signal	Smaller than full bundles; faster retrieval and triage
Configuration snapshot	Drift investigation	Pair with a baseline record for comparisons
Update evidence	Maintenance window validation	Attach to change ticket when required

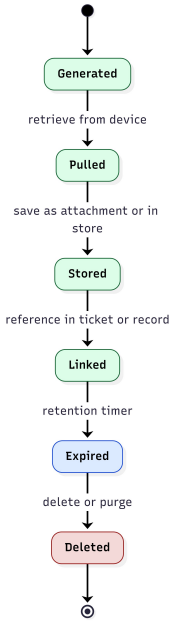


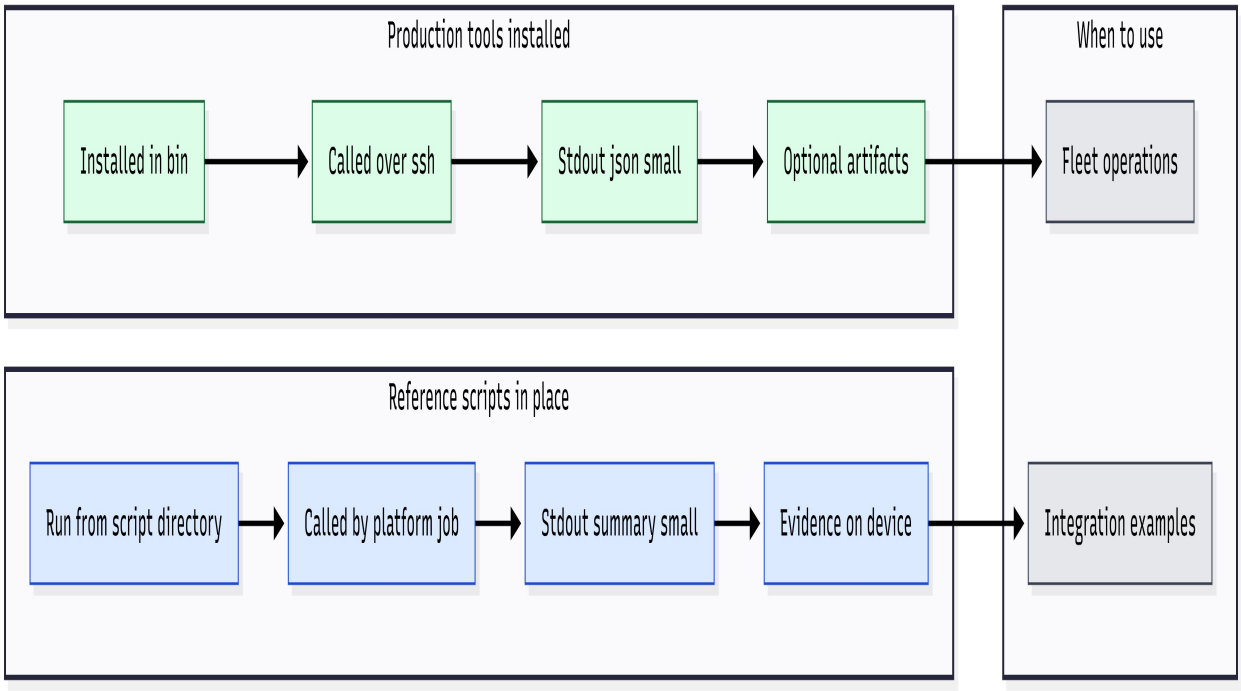
Figure 12: Artifact lifecycle — generate retrieve store link expire.

2.9.1.5 Implementation

This section is meant to help an Enterprise IT admin (or integrator) quickly answer:

- What tooling does NVIDIA provide?
- Where is the source code and documentation for each tool?
- What gets installed on-device and what runs in-place as reference code?
- Where do outputs and logs land on the DGX Spark device?
- Which tools or scripts are relevant to each lifecycle stage

For top-level directory layout and command-to-source-path mapping, see *Repository layout reference*.



2.9.1.5.1 Repository Content

Download Enterprise Lifecycle Integration Scripts package (ZIP) and extract it on your integration host.

The package is a collection of example Python tools and shell scripts intended to help enterprise IT teams manage DGX Spark systems at scale. The examples use agentless, SSH-based remote execution with structured JSON output for integration into your enterprise management platforms. NVIDIA provides these materials for reference and customer adaptation. They are not a supported production software offering. The examples illustrate capabilities across the device lifecycle, from procurement and provisioning through monitoring, maintenance, incident response, and retirement, with no resident agent on the device. Capabilities include the following:

- Hardware and software inventory
- Firmware and driver reporting
- Health diagnostics
- Controlled updates
- Security posture checks

The archive includes tool and script sources, *reference shell scripts*, optional per-tool `install.sh` scripts, and companion documentation referenced in this section.

This repository contains two distinct implementation types:

2.9.1.5.1.1 Production Tools (11)

These are production-ready, stdlib-only Python tools designed for fleet automation with:

- JSON-first output
- Configuration defaults with CLI override support
- Safety guardrails for state-changing operations
- Comprehensive documentation and test coverage

The following is the installed command location on the device:

- DGX_spark_management/bin/

The following are the production tools functional areas and source roots:

- clear_asset_information/ (7 tools)
- controlled_sw_fw_updates/ (1 tool)
- remote_ops_remediation/ (2 tools)
- data_protection_privacy/ (1 tool)

Each production tool follows a consistent structure:

- {functional_area}/{tool_name}/src/
- {functional_area}/{tool_name}/config/
- {functional_area}/{tool_name}/install.sh
- Installed entry point(s) land in DGX_spark_management/bin/

2.9.1.5.1.2 Canonical Landscape Reference Scripts (8)

These are reference implementations designed for Canonical Landscape “remote script execution” constraints. The device must be enrolled in Landscape first; see *Ubuntu Pro and Landscape client enrollment*.

The *reference shell scripts* and their per-script `README.md` files are included in Enterprise Lifecycle Integration Scripts package (ZIP). After you extract the archive, follow `LANDSCAPE_REFERENCE_SCRIPTS_SETUP.md` at the archive root or `docs/core_docs/01_core_overview/03_LANDSCAPE_REFERENCE_SCRIPTS_SETUP.md` before using those scripts.

- Minimal error handling (examples, not production frameworks)
- stdout output intended to be short and platform-friendly
- Detailed evidence stored locally on the device per run

Reference Scripts Run In-Place

- They are not installed into bin/ by default.
- They live under their functional area folders with landscape_ prefix naming.

Reference Script Functional Areas

- attestable_conformance_regulatory/ (1 script)
- resilience_recovery_rollback/ (4 scripts)
- network_enterprise_connectivity/ (2 scripts)
- security_posture_vuln_response/ (1 script)

Reference script structure:

- {functional_area}/landscape_{script_name}/README.md
- {functional_area}/landscape_{script_name}/{script_name}.sh

Script Path (Repo)	Primary Script	Purpose	Stdout + Evidence
attestable_conformance_regulatc landscape_signing_verification/	sign-ing_verificatio	APT signing verification	Short stdout; detailed evidence stored per run on device
resilience_recovery_rollback/ land-scape_verified_boot_integrity/	veri-fied_boot_inte	Verified boot integrity reference	Short stdout; per-run evidence under run directory
resilience_recovery_rollback/ land-scape_recovery_backup_levels/	recov-ery_backup_le	Recovery/backup level reference	Short stdout; per-run evidence under run directory
resilience_recovery_rollback/ land-scape_factory_reset_reprovision/	fac-tory_reset_rep	Factory reset + reprovision	Short stdout; per-run evidence under run directory
resilience_recovery_rollback/ landscape_health_watchdogs/	health_watchd	Health watchdog reference	Short stdout; per-run evidence under run directory
network_enterprise_connectivity/ landscape_collect_package/	col-lect_package.s	Support bundle collection reference	Short stdout; evidence bundle stored on device
network_enterprise_connectivity/ landscape_retrieve_logs_stdout/	re-trieve_logs_st	Bounded log retrieval reference	Short stdout; bounded excerpts; detailed evidence on device
secu-rity_posture_vuln_response/ landscape_encryption_at_rest/	encryp-tion_at_rest.sh	Encryption-at-rest reference	Short stdout; evidence stored per run on device

2.9.1.5.1.3 When to Use Reference Scripts

Use reference scripts when management platform constraints are the dominant design factor (for example, strict stdout limits or “run script remotely” paradigms). For broad enterprise automation, use production tools installed in `bin/`.

2.9.1.5.2 Primary Entry Points

After you download and extract the script package (see *Repository Content*), use the following files in the extracted tree for the authoritative directory layout, reading order, and per-tool references. Paths below are relative to the root of the extracted script package.

- `docs/core_docs/00_INDEX.md` — Documentation index and reading order
- `docs/core_docs/01_core_overview/01_PROJECT_README.md` — Overview, quick start, naming conventions
- `docs/core_docs/01_core_overview/02_PROJECT_STRUCTURE.md` — Directory tree and runtime layout
- `docs/core_docs/01_core_overview/03_LANDSCAPE_REFERENCE_SCRIPTS_SETUP.md` — Landscape reference script framework

I want to...	Read this first
Get a quick start view	docs/core_docs/01_core_overview/01_PROJECT_README.md
Understand folder structure and runtime layout	docs/core_docs/01_core_overview/02_PROJECT_STRUCTURE.md
Integrate Landscape reference scripts	docs/core_docs/01_core_overview/03_LANDSCAPE_REFERENCE_SCRIPTS_

2.9.1.5.3 Tool Deep-Dive Documentation

Each production tool has the following:

- A tool folder README at {functional_area}/{tool_name}/README.md
- Centralized “production tools” documentation referenced by `docs/core_docs/00_INDEX.md`. *Tool locations and reference code map* points to these documents per tool and summarizes the following:
- Command synopsis and options
- JSON output structure and key fields
- Troubleshooting and known limitations
- Integration notes

2.9.1.5.4 Installed Commands on the DGX Spark

This subsection provides information about the installed commands on the DGX Spark.

2.9.1.5.4.1 Installed Command Directory

When deployed, production tools are installed here:

- DGX_spark_management/bin/ Expected installed commands:
- bin/device_identity.py
- bin/hardware_config.py
- bin/firmware_reporter.py

- bin/os_build_identity.py
- bin/driver_inventory_reporter.py
- bin/software_inventory_reporter.py
- bin/NVAIAwrite
- bin/NVAIAread
- bin/spark_updatectl.py
- bin/spark_diagctl.py
- bin/reset_reason_reporter.py

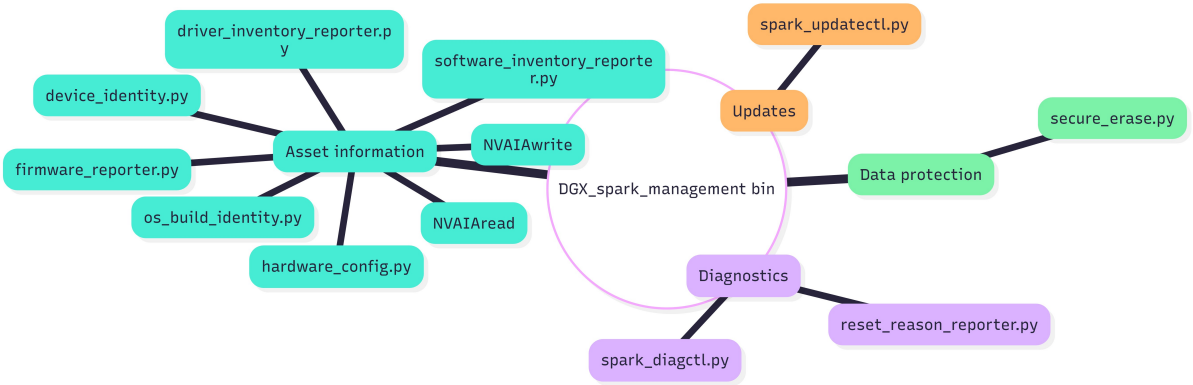


Figure 15: Example listing of DGX_spark_management/bin/ on a device.

2.9.1.5.5 Runtime Outputs and Logs

Production tools and reference scripts write runtime output to:

- /var/lib/dgx_spark_management/{functional_area}/{tool_or_script_name}/ Examples:
- /var/lib/dgx_spark_management/clear_asset_information/hardware_inventory_collector/ device_identity.json
- /var/lib/dgx_spark_management/controlled_sw_fw_updates/update_control_plane/ status.json
- /var/lib/dgx_spark_management/remote_ops_remediation/ diagnostic_collector/diagnostics_full.json

Production logs are found under:

- /var/log/dgx_spark/{functional_area}/{tool_name}/

Examples:

- /var/log/dgx_spark/clear_asset_information/ hardware_inventory_collector/device_identity.log
- /var/log/dgx_spark/controlled_sw_fw_updates/update_control_plane/spark_updatectl.log
- /var/log/dgx_spark/remote_ops_remediation/diagnostic_collector/spark_diagctl.log

Landscape reference scripts store per-run evidence under:

- /var/lib/dgx_spark_management/{functional_area}/{landscape_script}/run_<UTC>/

Category	Location Pattern
Production runtime outputs	/var/lib/dgx_spark_management/ {functional_area}/{tool_name}/
Production logs	/var/log/dgx_spark/ {functional_area}/{tool_name}/

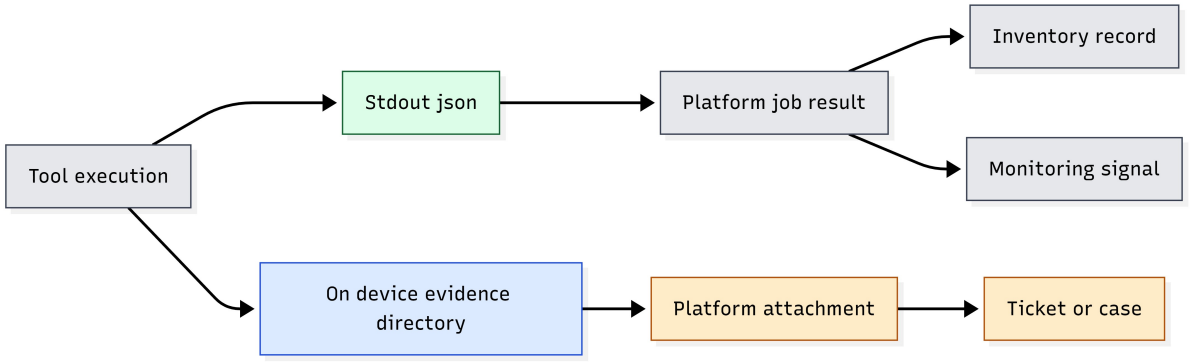


Figure 16: Evidence flow — stdout JSON vs on-device evidence directory vs platform attachment.

2.9.1.5.6 Repository Layout Reference

Use this subsection to locate folders and source files in the `DGX_spark_management/` script package. For the full directory tree and runtime layout, see `docs/core_docs/01_core_overview/02_PROJECT_STRUCTURE.md` in the extracted archive.

2.9.1.5.6.1 Top-Level Repository Layout

Path	Purpose
bin/	Installed production tool entrypoints (what operators run on end-points)
docs/	Architecture, structure, deployment, and tool references
clear_asset_information/	Production tools: identity, hardware, firmware, OS build, drivers, software, asset tags
controlled_sw_fw_updates/	Production tool: update control plane (reboot coordination, rollback reporting)
remote_ops_remediation/	Production tools: diagnostics collector, reset reason reporter
data_protection_privacy/	Production tool: data protection and privacy
at-testable_conformance_regulat	Landscape script: APT signing verification
resilience_recovery_rollback/	Landscape scripts: boot integrity, backup levels, factory reset, watchdogs
net-work_enterprise_connectivity/	Landscape scripts: support bundle collection, log retrieval
secu-rity_posture_vuln_response/	Landscape script: encryption-at-rest reporting
enrollment_identity_access/	Planned functional area placeholder
integration_automation/	Planned functional area placeholder
lifecycle_business_ops/	Planned functional area placeholder
packaging/	Packaging artifacts (for example, systemd units)
tools/	Development tooling helpers

Where operators spend time

- Operators and integrators typically use `bin/` (installed commands) and `docs/` (usage and integration guidance).
- Developers work primarily under functional area folders.

2.9.1.5.6.2 Production Tools: Command-to-Source Mapping

Each production tool folder includes `README.md`, `src/`, `config/`, and `install.sh` (installs into `bin/`).

2.9.1.6 Clear Asset Information (Seven Tools)

Installed command	Source root	Notes
device_identity.py	clear_asset_information/hardware_inventory_c	Stable identifier through SMBIOS/DMI logic
hardware_config.py	clear_asset_information/hardware_inventory_c	CPU/GPU/SSD/NIC/memory enumeration
firmware_reporter.py	clear_asset_information/firmware_version_repc	BIOS/UEFI/NIC/SSD/GPU firmware enumeration
os_build_identity.py	clear_asset_information/os_build_identity_repo	OS build + DGX identity
driver_inventory_repc	clear_asset_information/driver_inventory_repor	GPU/NIC/storage/USB drivers
soft-ware_inventory_repo	clear_asset_information/software_inventory_re	dpkg/snap/pip/docker enu-meration
NVAIAwrite / NVA-IAread	clear_asset_information/asset_tag_manager/sr	UEFI-backed metadata store (read/write)

2.9.1.7 Controlled Software/Firmware Updates (One Tool)

Installed command	Source root
spark_updatectl.py	controlled_sw_fw_updates/update_control_plane/src/spark_updatectl.py

2.9.1.8 Remote Ops and Remediation (Two Tools)

Installed command	Source root
spark_diagctl.py	remote_ops_remediation/diagnostic_collector/src/spark_diagctl.py
reset_reason_reporter.py	remote_ops_remediation/reset_reason_reporter/src/reset_reason_reporter.py

2.9.1.8.1 Lifecycle Mapping to the Actual Code in This Repository

This section maps the lifecycle found in *Lifecycle backbone (how enterprise IT actually runs fleets)* to the tools and scripts in this repository.

2.9.1.8.1.1 Procurement and Receiving

Relevant production tools:

- bin/device_identity.py
- bin/os_build_identity.py
- bin/hardware_config.py

- bin/firmware_reporter.py

Stage Objective	Recommended Tools	Outputs to Capture
Acceptance snapshot	Identity + build + HW + FW	stdout JSON + stored on-device JSON paths

2.9.1.8.1.2 Initial Provisioning

Relevant production tools:

- Identity, hardware, firmware, OS build, driver inventory, and software inventory
- optional: bin/NVAIAwrite and bin/NVAIAread if UEFI-backed tags are used

Baseline Set	Primary Tools	Drift Anchor Fields
Identity + build + inventories	device_identity.py os_build_identity.py hardware_config.py firmware_reporter.py driver_i nventory_reporter.py software_i nventory_reporter.py	OS build identity; firmware set; driver set; and critical package inventory hashes and counts

Optional first-boot automation uses Cloud-init with repacked BaseOS media; see *Cloud-init for DGX Spark*.

2.9.1.8.1.3 Cloud-init for DGX Spark

This section explains Cloud-init basics and how it supports automated first-boot provisioning of DGX Spark in enterprise deployments.

2.9.1.8.2 What is Cloud-init

Cloud-init is a widely used initialization framework that runs during the early boot process of Linux systems. It allows system configuration and provisioning tasks to be automated at first boot using simple configuration files. Originally developed for cloud environments, Cloud-init is equally effective for provisioning physical devices such as DGX Spark systems.

For DGX Spark deployments, Cloud-init enables administrators to apply configuration policies automatically when a device is first powered on, eliminating the need for manual setup.

Typical uses include:

- Setting the system hostname and identity
- Creating administrator accounts and installing SSH keys
- Configuring networking or Wi-Fi profiles
- Installing packages or applying update policies
- Installing corporate certificates or proxy settings
- Registering the system with enterprise management platforms

Cloud-init executes during the early stages of the boot process and normally runs once per instance, making it well suited for automated provisioning workflows.

2.9.1.8.3 DGX Spark Procedures

On DGX Spark, Cloud-init implements first-boot provisioning during the **Initial Provisioning** lifecycle stage. Site-specific configuration (users, packages, out-of-box experience behavior, and optional local package or firmware mirrors) is delivered through repacked BaseOS media, OEM Cloud-init seeds, and optional `OEMDATA` USB layout.

For step-by-step procedures, USB partitioning, repack scripts, example `user-data` files, and verification workflows, see *Custom Installation with Cloud-Init*.

For general Cloud-init module reference and behavior, see https://docs.cloud-init.io/en/latest/.

2.9.1.8.3.1 Ongoing Monitoring

The following are relevant production tools:

- bin/spark_diagctl.py (health posture and targeted collection modes)
- bin/reset_reason_reporter.py
- (Optional) Periodic drift re-checks using os_build_identity, driver_inventory, and firmware_reporter

Signal	Source Tool	Artifacts (if any)
Health posture	spark_diagctl.py	Optional bundle modes
Reset context	reset_reason_reporter.py	None

2.9.1.8.3.2 Maintenance Windows

The following are relevant production tools:

- bin/spark_updatectl.py
- Validation tools: os_build_identity, spark_diagctl, optional driver or firmware tools

Phase	Tools	Evidence
Pre-check	Build identity + health	stdout JSON
Update control	spark_updatectl.py	stdout JSON + optional artifact
Post-check	Build identity + health	stdout JSON

2.9.1.8.3.3 Incident Response

Relevant production tools:

- Level 1: Targeted triage: spark_diagctl health, reset_reason_reporter, plus identity, build, and drivers as needed
- Level 2 Deep evidence: spark_diagctl bundle collection modes. Relevant reference scripts (Landscape environments) include log retrieval and support bundle collection scripts under network_enterprise_connectivity/

Level	Tools and Scripts	Output Handling Expectation
Level 1	Bounded collectors	stdout JSON only
Level 2	Bundle modes / reference scripts	stdout JSON + artifacts/evidence

2.9.1.8.3.4 Cascade and Redeployment

Cascade and redeployment is a re-provisioning motion:

- The device changes user, function, or environment
- The lifecycle loop returns to provisioning steps while preserving linkage to prior history, if desired.

Relevant production tools:

- Provisioning baseline set (identity, build, hardware, firmware, drivers, and software)
- Optional tag tools, if used to mark new function or owner. Relevant reference scripts (Landscape environments) include factory reset and re-provisioning examples under resilience_recovery_rollback/

Motion	Recommended Steps	Evidence to Capture
Re-provision	Baseline re-capture or optional reset	New baseline JSON plus optional reset evidence

2.9.1.9 Tool Locations and Reference Code Map

This section explains the tools and example implementations inside the project repository. The goal is to make it easy to locate the correct scripts, understand what they do, and embed them into enterprise platform wrappers without copying large files.

2.9.1.9.1 Repository Layout Conventions Used in This Section

It is assumed that the repository organizes content into:

- **Tools**: Scripts or binaries invoked remotely (collectors and controllers).
- **Reference code**: Examples used to demonstrate platform integration patterns.
- **Platform wrappers**: Example job scripts for specific management platforms.
- **Artifacts**: Optional bundles (diagnostics or logs) created on demand.

To avoid repeating paths everywhere, define a single canonical base path and reuse it.

Note: In the provided examples, the tool directory is referenced as a single variable (for example, dgx_tool_dir). Replace it with your repo's actual path on the device.

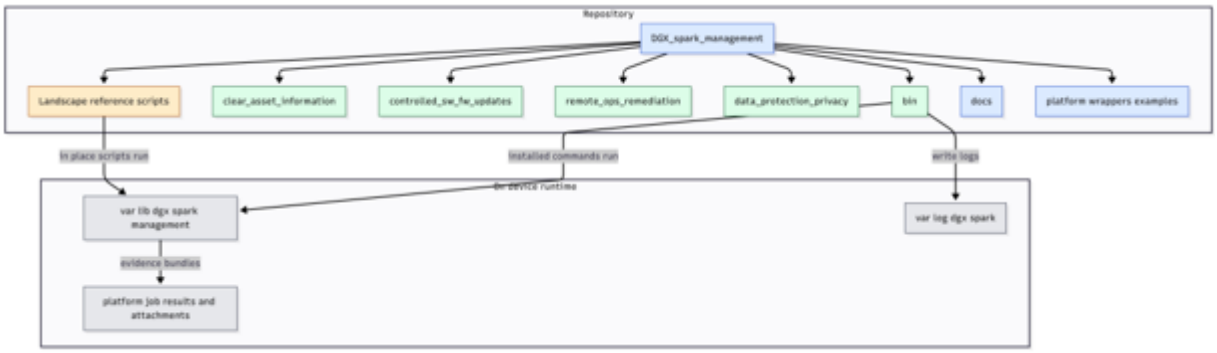


Figure 17: Repo layout — tools, reference code, wrappers, and artifacts.

2.9.1.9.2 Canonical Tool Directory and Naming Scheme

For readability and consistency, it is assumed that:

- Collectors are named *.py and emit one JSON envelope on stdout
- Controllers are also *.json but are gated and validated
- Artifact generators indicate artifact creation and return pointers in artifacts

Category	Convention
Collector	Name ends in .json; read-only; safe to run frequently; bounded output
Controller	Name ends in .json; changes state; gated; includes precheck= and postcheck guidance
Artifact generator	Name indicates bundle creation; stdout JSON returns artifact pointer list

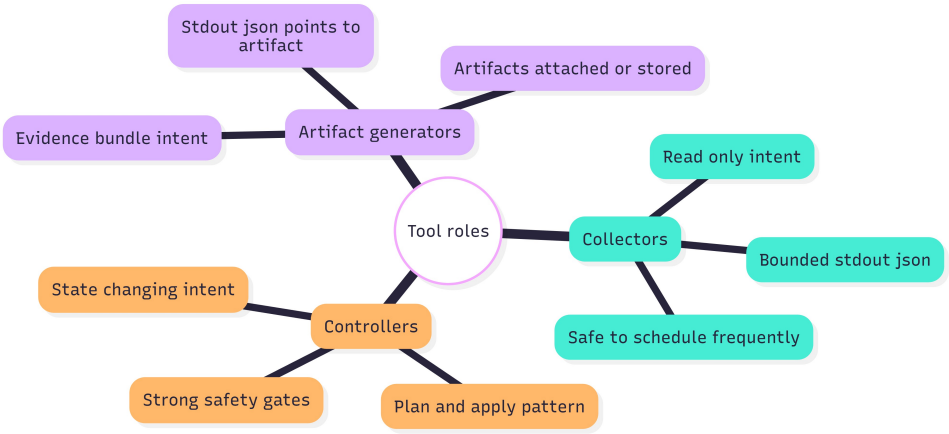


Figure 18: Naming scheme — collectors vs controllers vs artifact generators.

2.9.1.9.3 Reference Code Map

See *Canonical Landscape reference scripts (8)* for the full list of Landscape reference scripts and their purposes. Enrollment is required before running them; see *Ubuntu Pro and Landscape client enrollment*.

2.9.1.9.3.1 How to Use These References

Treat these directories as examples of platform wrapper integration and evidence minimization. For production automation, use the guide's standardized collector and controller tools and keep wrapper scripts thin.

2.9.1.9.3.2 Tool Invocation Patterns

Across all platforms, it is assumed that the same invocation pattern is used:

- SSH executes a tool from the canonical tool directory.
- The tool emits one JSON envelope on stdout.
- The platform stores stdout JSON verbatim and parses it off-box.
- Artifacts are created only on demand; stdout returns pointers.

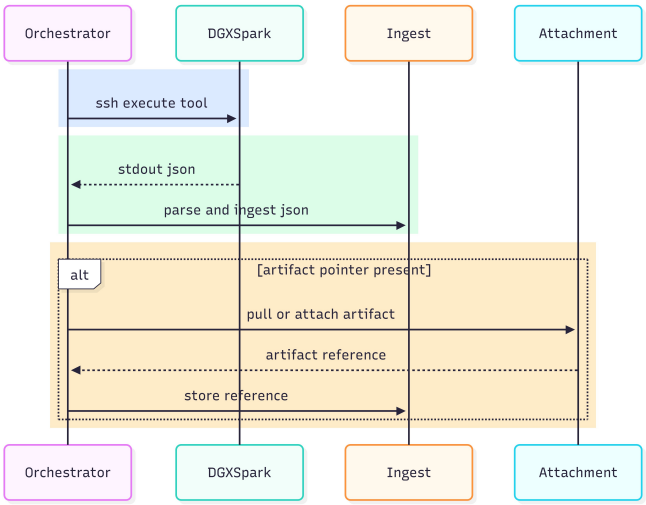


Figure 19: Invocation contract — SSH executes tool stdout JSON ingest; optional artifacts by pointer.

2.9.1.9.3.3 Embedding Paths in Examples

To keep the document readable, examples should not hardcode long paths repeatedly. Use one variable name consistently, and document where the variable is set per platform.

Platform Wrapper	How to Represent the Tool Path
Windows scripts	Set a single variable and call tools via that variable
Ansible	Use a vars: entry such as dgx_tool_dir and reference it in tasks
Landscape jobs	Use the job definition to call a script in-place, or call a canonical tool directory
BigFix/Tanium	Store the SSH wrapper centrally and pass the tool path as a parameter

Recommendation

In later sections, whenever you see dgx_tool_dir, replace it with the actual installed location on the endpoint. Keep the replacement consistent across all playbooks and wrappers.

2.9.1.9.3.4 Evidence Minimization (stdout vs artifacts)

This repository includes examples that write evidence to disk (for later retrieval). That pattern is correct for large evidence and diagnostics, but stdout must remain bounded.

Use this decision rule:

- If the result is small and can be summarized, return it in stdout JSON.
- If the result is large, generate an artifact and return a pointer in stdout JSON.

Case	Preferred Handling
Routine inventory and health signals	stdout JSON summary only
Long logs or large inventories	artifact bundle + pointer in stdout JSON
Incident response deep evidence	artifact bundle + hashes + retrieval hint
Audit or retirement proofs	stdout JSON certificate + artifact evidence as required

For additional information on evidence minimization, see *Artifact Strategy* and *Evidence Handling: Summary First, Artifacts on Demand*.

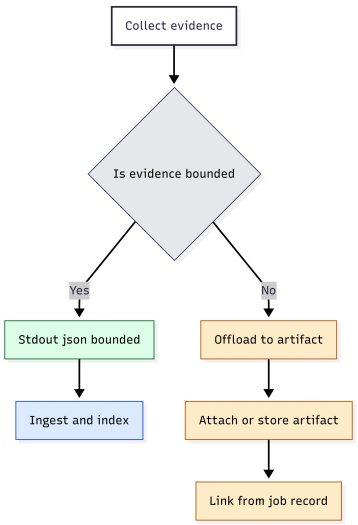


Figure 20: Evidence minimization — bounded stdout JSON vs artifact offload.

2.9.1.10 Platform Wrapper Patterns and Packaging Examples

This section shows examples of how to package the guide's SSH + stdout-JSON tooling model inside common enterprise platforms.

Key takeaways: Consider the same model as found in *JSON-first, agentless SSH execution model* and *Artifact Strategy* (execute capture stdout JSON optionally pull artifacts). Use collectors (safe, frequent) vs controllers (gated, validated).

2.9.1.10.1 Common Packaging Principles (Applies to All Platforms)

2.9.1.10.1.1 Output Sizing

Consider the same principle as in *Artifact Strategy*: keep stdout bounded and use artifacts for large evidence. Most platforms impose stdout or result-size limits. Packaging should treat:

- stdout JSON as the authoritative small record
- Large bundles (diagnostics tarball, large logs, or full inventories) as attachments or artifacts

Output Type	Packaging Guidance
stdout JSON	Bounded summary, stable envelope, safe to index and ingest
stderr	Keep for debugging. Do not rely on it for structured data.
Artifacts	Use for large evidence. Attach or store and return pointers in JSON.

Recommendation

If a platform truncates outputs, treat the truncation as a correctness failure. Reduce stdout size or switch the payload to an artifact generator that returns only a pointer on stdout.

2.9.1.10.2 Ansible Playbook Examples (Agentless SSH-Native)

Ansible aligns naturally with this information because it is SSH-native and can fetch files. These examples focus on:

- Running the installed tools on each host
- Capturing JSON results per host
- Optionally fetching artifacts

Inventory Assumption	Conceptual Guidance
Host grouping	Use groups for rings or waves (such as pilot, wave1, wave2, or broad)
Connection	SSH keys and non-interactive execution; bastion if required
Privilege	Use become for controllers; keep collectors unprivileged when possible
Outputs	Write stdout JSON per-host; collect centrally; parse off-box

2.9.1.10.2.1 Example: Provisioning Baseline (Conceptual)

— name: Provisioning baseline (collectors) hosts: dgx_spark gather_facts: no vars:

dgx_tool_dir: /path/to/tools # set per Part 4 tasks: - name: Collect identity/build

shell: “sudo {{ dgx_tool_dir }}/identity.json” register: identity_out changed_when: false

- name: Collect hardware/firmware/driver/software inventories shell: “sudo {{ dgx_tool_dir }}/inventory.json” register: inv_out changed_when: false
- name: Write results (stdout JSON) per-host copy:

content: “{{ identity_out.stdout }}n” dest: “./out/{{ inventory_hostname }}_identity.json” - name: Write inventory JSON per-host copy:

content: “{{ inv_out.stdout }}n”

dest: “./out/{{ inventory_hostname }}_inventory.json”

2.9.1.10.2.2 Example: Monitoring Snapshot

— name: Monitoring snapshot (collectors) hosts: dgx_spark gather_facts: no vars:

dgx_tool_dir: /path/to/tools # set per Part 4 tasks: - name: Health check

shell: “sudo {{ dgx_tool_dir }}/spark_diagctl.py “ register: health_out changed_when: false

- name: Reset reason

shell: “sudo {{ dgx_tool_dir }}/reset_reason.json” register: reset_out changed_when: false

- name: Save JSON outputs copy:

content: “{{ health_out.stdout }}n” dest: “./out/{{ inventory_hostname }}_health.json”

- name: Save reset reason JSON copy:

content: “{{ reset_out.stdout }}n”

dest: “./out/{{ inventory_hostname }}_reset_reason.json”

2.9.1.10.2.3 Example: Incident Response L2

— name: Incident response L2 (artifact bundle) hosts: dgx_spark gather_facts: no vars:

dgx_tool_dir: /path/to/tools # set per Part 4 tasks:

- name: Trigger diagnostics bundle

shell: “sudo {{ dgx_tool_dir }}/diag_collect.json” register: diag_out

- name: Save diagnostics JSON copy:

content: “{{ diag_out.stdout }}n”

dest: “./out/{{ inventory_hostname }}_diag_collect.json”

- name: Fetch artifact (conceptual) # Replace with a real fetch that reads the artifact pointer from JSON fetch:

src: “/path/on/device/to/artifact.tar.gz”

dest: “./artifacts/{{ inventory_hostname }}_artifact.tar.gz” flat: yes

2.9.1.10.3 Ubuntu Pro and Landscape Client Enrollment

Complete these steps before using the Landscape reference scripts in *Canonical Landscape reference scripts (8)* or the job examples in *Canonical Landscape script examples*. Download and extract Enterprise Lifecycle Integration Scripts package (ZIP) if you have not already done so.

For the most up-to-date information, see Landscape installation and set-up - Landscape documentation.

Scope: These steps target Landscape SaaS enrollment through Ubuntu Pro (`sudo pro enable landscape` with **Self-hosted server?** set to **No**). Self-hosted Landscape deployments can differ; follow Canonical's documentation for those environments.

Landscape SaaS enrollment requires Ubuntu Pro. The following steps enable Ubuntu Pro services and enroll a DGX Spark as a Landscape client.

1. Register for Ubuntu One.

   Go to Ubuntu Pro and register for an Ubuntu One account.

2. Create a Landscape account.

   1. Open a browser.
   2. Navigate to https://landscape.canonical.com.
   3. Sign in using Ubuntu One.
   4. Create a new organization or join an existing one.
   5. Record your account name (organization name). You need this value during client enrollment.

3. Attach Ubuntu Pro on the DGX Spark.

   Landscape SaaS enrollment requires Ubuntu Pro. On the DGX Spark, check status:

   ```
   sudo pro status
   ```

   If the system is not attached, run:

   ```
   sudo pro attach <YOUR_UBUNTU_PRO_TOKEN>
   ```

   Obtain the token from https://ubuntu.com/pro, then verify attachment:

   ```
   sudo pro status
   ```

   You should see **Attached: yes**.

4. Enable Landscape (interactive mode).

   Run:

   ```
   sudo pro enable landscape
   ```

   When prompted, enter:

   - **Self-hosted server?: No**
   - **Computer title:** for example, `dgx-spark-01`
   - **Account name:** your Landscape organization name

   This process installs, configures, and starts the Landscape client.

5. Approve the machine in the Landscape portal.

1. Return to the Landscape web portal.
2. Navigate to **Pending Machines**.
3. Select the DGX Spark system.
4. Click **Accept**.

The system appears in your managed machines list.

6. Verify client operation.

```
sudo systemctl status landscape-client
```

You should see **Active: active (running)**.

If needed, check the logs:

```
sudo tail -n 50 /var/log/landscape/client.log
```

After enrollment, you can use Landscape to:

- Apply tags (for example, DGX, Spark, AI-node)
- Execute remote scripts
- Schedule updates
- Monitor package and security status
- Create access groups
- Define update and compliance policies

2.9.1.10.4 Canonical Landscape Script Examples

Complete *Ubuntu Pro and Landscape client enrollment* before running these jobs.

Script paths below are relative to the root of the extracted Enterprise Lifecycle Integration Scripts package (ZIP).

Canonical Landscape is included because the repository provides reference scripts that illustrate platform job integration. They are not meant to replace production tools. They demonstrate:

- Integrating health and security checks into Canonical Landscape jobs
- Producing bounded stdout results
- Writing evidence to disk for later retrieval

Reference script locations (from *Tool locations and reference code map*):

- attestable_conformance_regulatory/landscape_signing_verification/
- resilience_recovery_rollback/landscape_verified_boot_integrity/
- resilience_recovery_rollback/landscape_recovery_backup_levels/
- resilience_recovery_rollback/landscape_factory_reset_reprovision/
- resilience_recovery_rollback/landscape_health_watchdogs/
- network_enterprise_connectivity/landscape_collect_package/
- network_enterprise_connectivity/landscape_retrieve_logs_stdout/
- security_posture_vuln_response/landscape_encryption_at_rest/

For example, the following is an APT signing verification (conceptual)

Landscape job (conceptual)

- run signing verification

- emit bounded PASS/FAIL/UNKNOWN on stdout # - write evidence to a directory for retrieval

run_signing_verification emit_stdout_summary write_evidence_dir

2.9.1.10.4.1 Example: Verified Boot Integrity

Landscape job (conceptual)

- check verified boot signals

- emit bounded summary

- store evidence for later retrieval

check_verified_boot emit_stdout_summary write_evidence_dir

2.9.1.10.4.2 Example: Factory Reset with Reprovision

Landscape job (conceptual)

- customer-gated reset workflow # - emit

retirement/reset certificate JSON on stdout # -

store evidence and logs

validate_gate_conditions perform_reset emit_stdout_certificate write_evidence_dir

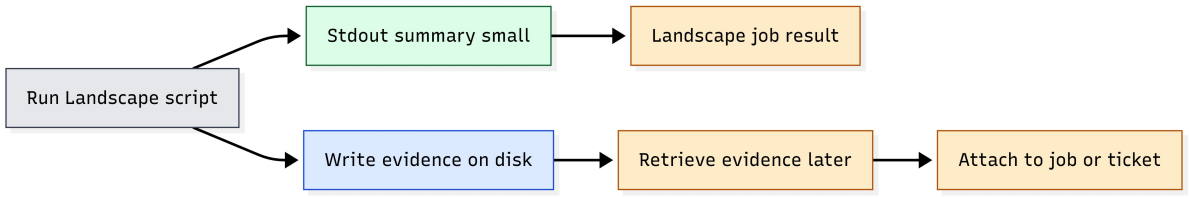


Figure 21: Landscape constraints — stdout summary vs evidence-on-disk retrieval pattern.

2.9.1.10.5 Tanium Package Patterns

Tanium often implements operations through:

- Packages or modules that run scripts
- Results captured in question outputs or package logs
- Attachments handled through platform mechanisms

These patterns show:

- Packaging lifecycle SSH payloads as Tanium packages
- Storing stdout JSON as package output (small)
- Storing large bundles as attachments

Tanium Concept	Documentation Mapping
Package	Wrapper payload that runs SSH tools and captures stdout JSON
Question and result	Ingested JSON summary for reporting and filtering
Package logs	stderr and troubleshooting context
Attachments	Artifacts (bundles) referenced by JSON pointers

2.9.1.10.5.1 Pattern: Provisioning Baseline Package (Conceptual)

```
# Tanium package (conceptual)
# - run baseline collectors over SSH
# - store stdout JSON per-host
run_ssh_identity run_ssh_inventory store_json_outputs
```

2.9.1.10.5.2 Pattern: Monitoring Snapshot Package (Conceptual)

```
# Tanium package (conceptual)
# - run health/reset reason
# - store minimal JSON
run_ssh_health run_ssh_reset_reason store_json_outputs
```

2.9.1.10.5.3 Notes for Puppet and Chef (Drift and Scheduled Execution)

Puppet and Chef are well-suited for:

- Periodic execution of collectors (inventory or monitoring)
- Enforcing baseline presence of tooling
- Scheduled drift detection against known-good baselines

They are not typically used for interactive incident response bundles but can trigger them if desired.

Use Case	Usage
Tooling presence	Ensure the tool directory and dependencies exist on endpoints
Scheduled collectors	Run bounded collectors on a cadence. Forward JSON off-box.
Drift detection	Compare baseline summaries to current summaries and flag divergence
Controllers	Use sparingly. Prefer change-window orchestration platforms for risky actions.
Incident bundles	Possible, but generally better triggered by incident orchestration flows

2.9.2. Custom Installation with Cloud-Init

2.9.2.1 Overview

This information is a reference for IT administrators and engineers who customize NVIDIA DGX Spark deployments using Cloud-Init, USB installation media, and local hosting of Debian packages and firmware. These workflows implement the initial provisioning stage in *Enterprise Lifecycle Integration*. For Cloud-init concepts and where provisioning fits in the fleet lifecycle, see *Cloud-init for DGX Spark*.

Writing a repacked BaseOS ISO to a USB drive recreates the ISO partition layout on that medium. The layout is typically two partitions: a main installer volume plus a small ESP. OEM Cloud-Init content, `hook.sh`, and extra Debian packages or firmware that you do not embed in the ISO must then reside on an additional partition labeled `OEMDATA` (in the free space on the same USB drive) or on a separate USB drive with an `OEMDATA` volume. After installation, Cloud-Init on first boot uses that content while the relevant media remains connected. Optional text files on `OEMDATA` (`apt-repo.url`, `apt-packages.txt`, `lvfs-mirror.url`) point the system to a local Advanced Package Tool (APT) repository, a local firmware mirror, or both.

Repacking the BaseOS image is required for the customization workflows. Debian packages and firmware can exist inside the repacked ISO (`oemdata/debs` and optional firmware in the image), so the installer can use them from `/cdrom` without a separate `OEMDATA` partition or a second USB drive for that material. Flashing the ISO to a USB drive still produces the image's own partition layout, typically two partitions (installer plus ESP), and embedding content in the ISO does not remove those. Alternatively, supply that material from an `OEMDATA` partition in the free space after the ISO layout on the same USB drive, or from another USB drive. You can also mirror Ubuntu ports and the Linux Vendor Firmware Service (LVFS) on a dedicated server (Spark A) and point clients (Spark B) at that server with `apt`, `fwupd`, and `oemdata/hook.sh`.

The procedures that follow cover repacking the BaseOS ISO, USB partitioning and the `OEMDATA` layout, hosting a minimal `.deb` repository and firmware tree (for example, on a desktop), mirroring full Ubuntu ports and LVFS under `~/mirror`, client configuration, Cloud-Init integration, security considerations, and verification steps. Full reference listings for `hook.sh` and `oem-iso-cfg.sh`, plus a partial `repack_baseos.sh` excerpt and example OEM Cloud-Init files, appear in *Reference: OEM Scripts and Cloud-Init*. After you complete those procedures, use *Validation scenarios and feedback questions* for structured validation and feedback prompts.

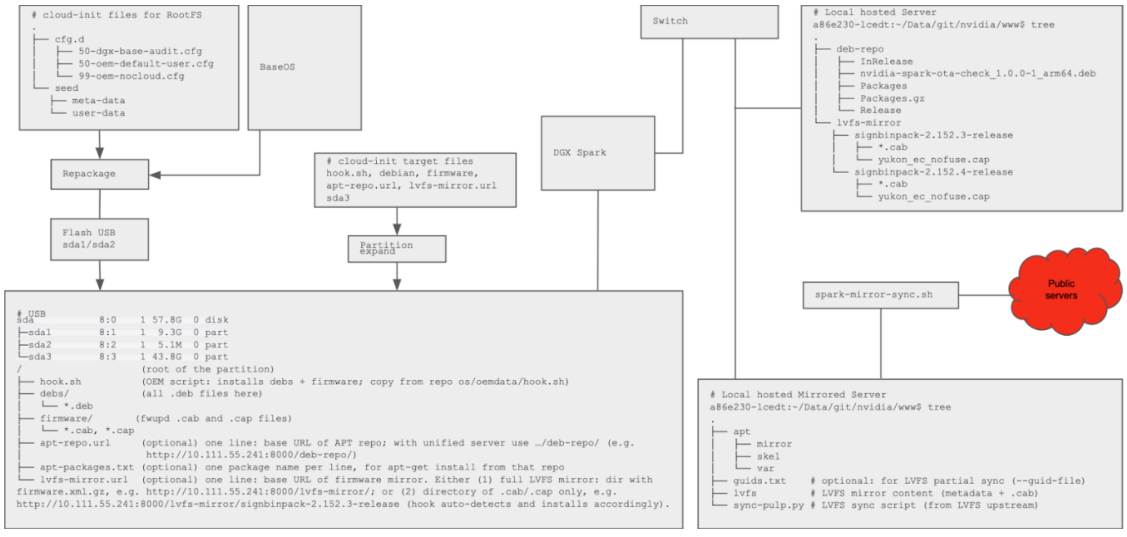


Fig. 3: Spark custom installation: repacked BaseOS ISO, USB `OEMDATA`, optional local mirrors, and client integration.

2.9.2.2 Air-Gapped and Custom Installation Patterns

The following patterns describe how Cloud-Init, USB layout, and optional local services combine. They align with enterprise customization workflows that use Cloud-Init OEM seeds and, when needed, an `OEMDATA` partition on the USB device.

Installation and Update Patterns

Pattern	What You Configure	OEMDATA partition	Where to Find Detail in This Document
Skip out-of-box experience (OOBE), keep factory software	Cloud-Init with a user-creation session in OEM seed data in the repacked ISO; no separate OEMDATA partition or second USB drive (the flashed ISO still produces its normal multi-partition layout on the drive).	Not used	Customize the BaseOS Image with `repack_baseos.sh`; Cloud-Init Integration; OEM `cloud-init` tree under `oemdata/cloud-init` (for example, `seed/user-data`, `cfg.d/`).
Keep OOBE, skip first-boot updates	Cloud-Init with an empty user session (no extra user provisioning in seed) in the repacked ISO; no separate OEMDATA partition or second USB drive. Adjust `user-data` and related OEM configuration to match policy. When there is no username and password section in `user-data`, OOBE is enabled.	Not used	Same Cloud-Init and repack references as the row above.
USB-hosted packages and firmware	An additional partition labeled OEMDATA (after the ISO's partitions on the same USB drive) or OEMDATA on a separate USB drive; contains `hook.sh`, `debs/`, and `firmware/` (`.cab`/`.cap`). Cloud-Init runs `hook.sh` on first boot while the USB drive is still present.	Required	USB Partitioning and the OEMDATA layout.
Local server with curated (LOCAL) sources	OEMDATA includes `hook.sh` plus `apt-repo.url`, optional `apt-packages.txt`, and optional `lvfs-mirror.url` pointing at a small local APT tree and optional firmware directory or LVFS-style layout, not a full Ubuntu archive mirror.	Required (for this USB-driven wiring)	Host a Minimal APT Repository and Firmware Tree on a Desktop; On the DGX Spark Client: `hook.sh` and OEMDATA files.
Local server with mirrored (MIRRORED) public sources	A separate host mirrors upstream Ubuntu ports and LVFS content (for example, using `spark-mirror-sync.sh` and related steps), then serves them over HTTP. OEMDATA includes `hook.sh` so the client is configured to use that mirror (the sync script runs on the mirror server, not on the USB drive).	Required (for `hook.sh`-based client wiring from USB)	Mirror the Full Ubuntu Ports and LVFS Content on a Server; Client Configuration and `hook.sh`.

How the pieces fit: Writing the ISO to a USB drive establishes that image's partition layout (usually two partitions). You can add an extra partition labeled OEMDATA in the remaining space on that same drive, or supply OEMDATA on another USB drive, for Debian packages, firmware, and `hook.sh`. Cloud-Init invokes `hook.sh` on first boot while the applicable installation media remains connected. Optional

files on OEMDATA (apt-repo.url, apt-packages.txt, lvfs-mirror.url) direct the client to a local APT repository, a package list, and a firmware mirror, respectively. hook.sh is for reference and works with the USB layout and server layout. If the USB layout or server layout changes, hook.sh might need to change accordingly. It can be trimmed down or expanded as needed.

When you use mirrored APT and LVFS content, populate those trees from public servers on a host that has outbound network access (or by another approved transfer method), then serve them on the installation network so target systems are not required to reach the public internet directly.

2.9.2.3 Example Constants

Example IP addresses and ports differ between workflows below. Substitute values that match your environment.

Example Constants for the Full Mirror Workflow (Port 8080)

Name	Example Value
Server Spark	spark-3ef8
Username and password	nvidia / nvidia; must match in Cloud-Init and hook.sh in all scopes
SERVER_IP	10.111.54.206
HTTP port	8080
Web root (server)	~/mirror (for example, /home/nvidia/mirror)

Example Constants for the Minimal Desktop Repository (Port 8000 or 80)

Item	Example Value
Desktop or server IP	10.111.55.241
Python HTTP server port	8000
Web root	/var/www or for example $HOME/oem-server
APT subdirectory	deb-repo under WEB_ROOT
LVFS subdirectory	lvfs-mirror under WEB_ROOT

2.9.2.4 Customize the BaseOS Image with repack_baseos.sh

From the $work_dir directory in the shared reference code, run repack_baseos.sh to produce a customized BaseOS ISO (a new installer image that combines the BaseOS content with reference or customized Cloud-Init). You can write the repacked ISO to a USB drive and combine it with an additional OEMDATA partition as described in *USB Partitioning and the OEMDATA Layout*.

Example command:

```
cd $work_dir
./repack_baseos.sh -iso <ISO_FILE|URL> -iso-root <ISO_ROOT_DIR>
```

repack_baseos.sh Options

Option	Description
`-iso`	Path to the local DGX OS ISO file or download URL.
`-iso-root <DIR>`	Directory where the ISO is extracted and repacked (default: `./iso-root`).
`-oem-debs <DIR>`	Directory of `.deb` packages to add into the ISO `oemdata/debs/` (default: `./oemdebs`).
`-volume-id <ID>`	Volume ID for the repacked ISO (maximum 32 characters).
`-clean`	Force fresh extract; remove extraction directories when done.
`-debug`	Verbose output.

Example:

```
./repack_baseos.sh -iso ~/Downloads/tmp/BaseOS/7.4.0/DGXOS-7.4.0-2026-01-26-
↪16-04-58-arm64.iso -iso-root ~/Downloads/tmp/BaseOS/Repack
```

`repack_baseos.sh` copies the OEM Cloud-Init tree and `oem-iso-cfg.sh` onto the repacked ISO when `OEMDATA_SRC` is set appropriately. If `$OEMDATA_SRC/cloud-init` exists, it replaces `$ISO_ROOT/oemdata/cloud-init` with that tree (including `seed/`, `cfg.d/`, and related files). If `$OEMDATA_SRC/oem-iso-cfg.sh` exists, it copies that file to `$ISO_ROOT/oemdata/`.

The BaseOS installer (Subiquity or autoinstall) runs `oem-iso-cfg.sh` during installation when the ISO is mounted at `/cdrom`. It runs in the target (installed) system context: it installs Debian packages from `/cdrom/oemdata/debs/` and copies the Cloud-Init seed from `/cdrom/oemdata/cloud-init/` to `/var/lib/cloud/seed/nocloud` and `cloud.cfg.d`. Logging goes to `/var/log/oem-iso-cfg.log`.

Example Cloud-Init layout on the ISO:

```
.
    cfg.d
        50-dgx-base-audit.cfg
        50-oem-default-user.cfg
        99-oem-nocloud.cfg
    seed
        meta-data
        user-data
```

After repacking, write the new ISO to a USB drive and follow *UEFI-Bootable Method: Write ISO to Whole Disk, Then Add a Second Partition* to add an `OEMDATA` partition for Debian packages, firmware, and Cloud-Init-related content.

2.9.2.5 USB Partitioning and the OEMDATA Layout

2.9.2.5.1 Baseline: Bootable Image First, Then Add a Second Partition

If you already created a bootable USB device by writing the ISO to the whole disk (`dd if=image.iso of=/dev/sdX`), the disk has the ISO's partition table; for DGX OS images, typically two partitions (large installer volume plus a small ESP). You cannot add an `OEMDATA` partition in the remaining space

without following a specific repartitioning flow: the ISO defines the layout the installer expects, and unused space after that layout is not used until you create another partition there.

> **Note:** The USB layout in this section is reference material from NVIDIA. Create the extra partition and set its filesystem label to `OEMDATA` so the example Cloud-Init seed and `hook.sh` in this guide can mount it by volume label during first boot. The label comes from OEM customization practice. Corporate IT, OEM partners, and integrators use the same steps when they follow this reference. You do not need to be an OEM vendor to create or populate the partition.

Until you add that partition, put Debian packages (and firmware) inside the ISO when repacking (`oemdata/debs` and optional firmware in the image). There is no separate `OEMDATA` volume in that baseline.

To add an `OEMDATA` partition for Debian packages and firmware on the same USB drive, use the flow in *UEFI-Bootable Method: Write ISO to Whole Disk, Then Add a Second Partition*.

2.9.2.5.2 UEFI-Bootable Method: Write ISO to Whole Disk, Then Add a Second Partition

Use a USB device larger than the ISO (for example, 32 GB or 64 GB for a ~14 GB ISO). Write the ISO to the whole disk so the first sector and partition table match the ISO; UEFI can then boot. Add a further partition in the remaining space for Debian packages and firmware (when the ISO already occupies two partitions, this is usually partition 3).

1. Write the ISO to the whole USB device (the disk is bootable). Optional: `pv /path/to/repacked.iso | sudo dd of="$USB" bs=4M conv=fsync` for progress if `pv` is installed.
2. Inspect how much space the ISO used. The DGX OS ISO typically creates two partitions (MBR or `msdos`): a large primary (approximately 13.6 GB) and a small ESP (approximately 5 MB). Note the end of partition 2 to start the new partition after it. The rest of the disk (for example, from approximately 14 GB to 62 GB) is free.
3. Add a new primary partition in the free space from the end of the ISO layout to 100%:

```
USB=/dev/sdX   # for example /dev/sdb; confirm with lsblk
sudo dd if=/path/to/repacked.iso of="$USB" bs=4M status=progress conv=fsync
sudo parted "$USB" print
# Example: ISO uses up to ~14 GiB; create partition 3 from 14 GiB to end of
↪disk
sudo parted -s "$USB" mkpart primary 14GiB 100%
```

Use the actual end of partition 2 from `parted print` if you want to avoid a small gap (for example, `13.7GiB` or `13700MiB`).

4. Format the new partition and set the label `OEMDATA`. The new partition is number 3 when the ISO already created two partitions (main plus ESP). If your ISO had only one partition, use `${USB}2` instead.

```
sudo mkfs.ext4 -L OEMDATA "${USB}3"
```

5. Mount the partition, create the directory layout, and copy files. Example mount point `/tmp/usb-data`:

```
sudo mkdir -p /tmp/usb-data
sudo mount "${USB}3" /tmp/usb-data
sudo mkdir -p /tmp/usb-data/debs /tmp/usb-data/firmware
sudo cp /path/to/*.deb /tmp/usb-data/debs/
```

(continues on next page)

(continued from previous page)

```
sudo cp /path/to/*.cab /path/to/*.cap /tmp/usb-data/firmware/
sudo cp /path/to/repo/os/oemdata/hook.sh /tmp/usb-data/
sudo umount /tmp/usb-data
```

OEMDATA Partition Layout

Path	Purpose
/ (root of partition)	Mount point root
hook.sh	OEM script that installs Debian packages and firmware; copy from os/oemdata/hook.sh. You can replace hook.sh with a custom script.
debs/	All .deb files
firmware/	fwupd .cab and .cap files
apt-repo.url (optional)	One line: base URL of the APT repository; with a unified server use …/deb-repo/ (for example, http: //10.111.55.241:8000/deb-repo/)
apt-packages.txt (optional)	One package name per line for apt-get install from that repository. If omitted, hook.sh runs a single-source apt upgrade against the OEM local repository only (when apt-repo.url is present)
lvfs-mirror.url (optional)	One line: base URL of the firmware mirror. Either (1) full LVFS mirror: directory with firmware.xml.gz, for example http://1 0.111.55.241:8000/lvfs-mirror/; or (2) directory of .cab/.cap only, for example h ttp://10.111.55.241:8000/lvfs-mir ror/signbinpack-2.152.3-release (hook auto-detects and installs accordingly).

Cloud-Init (in seed/user-data) mounts by label OEMDATA and invokes hook.sh on first boot. The sample user-data copies hook.sh to /tmp, exports OEM_MNT to the partition root, runs that copy, then removes it so paths such as $OEM_MNT/debs still resolve on the mounted volume. The provided oemdata/hook.sh installs from debs/ and firmware/ (.cab and .cap).

Example file contents:

```
# lvfs-mirror.url
http://10.111.55.241:8000/lvfs-mirror/signbinpack-2.152.4-release

# apt-repo.url: one line, base URL of the APT repository. It must match the
 ↪path your web server actually serves (for example the parent of Packages.
 ↪gz). hook.sh records this value in /etc/apt/sources.list.d/oem-local.list
 ↪on the client.
# Before first boot, open the URL in a browser to confirm it is reachable.
 ↪The client uses this address when it refreshes the index from the OEM local
 ↪source.
http://10.111.55.241:8000

# apt-packages.txt (optional): one package name per line. hook.sh runs apt-
 ↪get install for these from apt-repo.url.
# If you omit this file but apt-repo.url exists, hook.sh still adds the OEM
 ↪source and runs apt upgrade limited to that source only.
nvidia-spark-ota-check
```

If `apt-packages.txt` is absent, behavior depends on whether `apt-repo.url` is present; see *On the DGX Spark Client: hook.sh and OEMDATA Files*.

2.9.2.6 Host a Minimal APT Repository and Firmware Tree

Minimal in this section describes the straightforward way to host your own Debian packages and a firmware tree: index them with conventional tooling (for example `dpkg-scanpackages`), serve them over HTTP from a compact directory layout, and point clients at that layout. This is not a full Ubuntu ports mirror or LVFS synchronization. Refer to *Mirror the Full Ubuntu Ports and LVFS Content on a Server* for that workflow. The steps here assume packages and firmware are trusted, as in many air-gapped installations, and they do not cover hardening for an internet-exposed package mirror. Refer to *Security Considerations* for risks and mitigations.

One network resource can serve both the APT repository and the LVFS-related tree. Use a single web root (`WEB_ROOT`) with `REPO_DIR` and `LVFS_DIR` as subdirectories.

Define directories:

```
WEB_ROOT=/var/www   # or for example $HOME/oem-server
REPO_DIR="$WEB_ROOT/deb-repo"
LVFS_DIR="$WEB_ROOT/lvfs-mirror"
sudo mkdir -p "$REPO_DIR" "$LVFS_DIR"
sudo chown "$USER" "$REPO_DIR" "$LVFS_DIR"
```

2.9.2.6.1 On a Desktop or Server

1. Install tools for the APT repository: `sudo apt-get install -y dpkg-dev`
2. Copy `.deb` files into `REPO_DIR`.
3. Generate the APT index (`Packages.gz`). Re-run whenever you add or change `.deb` files:

```
cd "$REPO_DIR"
dpkg-scanpackages . /dev/null | gzip -9c > Packages.gz
```

4. Optional: Add `Release` and uncompressed `Packages` to avoid 404 responses. `apt` might request `Release`, `Packages` (uncompressed), and similar. A repository that only has `Packages.gz` can return 404 responses that `apt` can tolerate when `Packages.gz` is present. To serve them:

```
cd "$REPO_DIR"
zcat Packages.gz > Packages 2>/dev/null || gzip -dc Packages.gz > Packages
# Minimal Release file (paths relative to repo root); example block:
{
  echo "Origin: OEM Local Repo"
  echo "Label: oem-local"
  echo "Suite: ."
  echo "Codename: ."
  echo "Architectures: arm64 amd64"
  echo "Components: ."
  echo "Description: OEM local package repository"
  echo "Date: $(date -u -R)"
  echo "MD5Sum:"
  printf ' %s %s Packages.gz\n' "$(md5sum Packages.gz | awk '{print $1}')" "
↪$(stat -c%s Packages.gz)"
  printf ' %s %s Packages\n' "$(md5sum Packages | awk '{print $1}')" "$(stat -
```

(continues on next page)

(continued from previous page)

```
↪c%s Packages)"
  echo "SHA256:"
  printf ' %s %s Packages.gz\n' "$(sha256sum Packages.gz | awk '{print $1}')"
↪"$(stat -c%s Packages.gz)"
  printf ' %s %s Packages\n' "$(sha256sum Packages | awk '{print $1}')" "
↪$(stat -c%s Packages)"
} > Release
cp Release InRelease
```

Whenever you regenerate `Packages.gz` in step 3, repeat step 4: recreate uncompressed `Packages`, write `Release`, and copy `InRelease` using the commands in the code block above.

5. Serve both APT and LVFS over HTTP from one server.

Option A (Python):

```
cd "$WEB_ROOT"
python3 -m http.server 8000 --bind 0.0.0.0
```

Option B (Nginx): Install and enable Nginx, then use a configuration similar to:

```
location /deb-repo {
  alias /var/www/deb-repo;
  autoindex on;
}
location /lvfs-mirror {
  alias /var/www/lvfs-mirror;
  autoindex on;
}
```

With this layout, the server root lists only `deb-repo/` and `lvfs-mirror/`. On the USB drive you must use full paths (not the server root alone).

Examples:

- APT repository URL: `http://10.111.55.241:8000/deb-repo/` (Python) or `http://10.111.55.241/deb-repo/` (Nginx).
- LVFS mirror: full mirror at `http://…/lvfs-mirror/` (must contain `firmware.xml.gz`) or a directory of `.cab`/`.cap` only, for example `http://…/lvfs-mirror/signbinpack-2.152.3-release` (hook auto-detects).

6. Firewall: allow inbound HTTP on the port you use (for example, `sudo ufw allow 80/tcp`, `sudo ufw allow 8000/tcp`, `sudo ufw reload`).

2.9.2.6.2 On the DGX Spark Client: hook.sh and OEMDATA Files

Place the following on the USB drive's `OEMDATA` partition when you want `hook.sh` on the client to use your hosted APT repository:

- `apt-repo.url`: Include this file when the client should use your hosted APT repository. Put a single line containing the base URL (for example, `http://10.111.55.241:8000/deb-repo/` when you use Python's HTTP server on port 8000, or `http://10.111.55.241/deb-repo/` when you use Nginx on port 80).
- `apt-packages.txt`: Optional. If present, one package per line. Each line can be either a package name (for example, `nvidia-spark-ota-check`) or a full `.deb` file name (for exam-

ple, `nvidia-spark-ota-check_1.0.0-1_arm64.deb`); the hook derives the package name from a `.deb` file name when needed and runs `apt-get install` for that set. If you omit `apt-packages.txt` but `apt-repo.url` is present, the hook still adds the OEM local source and refreshes the index, then runs `apt upgrade` constrained to that source only (single-source upgrade, no named package list).

With `apt-repo.url` present, `hook.sh` wires `oem-local.list`, updates the index, then either installs listed packages from `apt-packages.txt` or performs the single-source upgrade when `apt-packages.txt` is absent. Refer to the listing in *First Boot: OEMDATA hook.sh and Cloud-Init Seed* or `oem-reference-includes/hook.sh` in your checkout.

Troubleshooting:

- Ignore or 404 for `Release.gpg` and `InRelease`: Expected for an unsigned repository; `[trusted=yes]` makes `apt` ignore the missing signature.
- "Unable to locate package": The repository is added (`/etc/apt/sources.list.d/oemlocal.list`). `apt` fetches `Release` but might not load `Packages` if the `Release` file is wrong. Ensure that `Release` has `Date`, paths `Packages.gz` and `Packages`, and run `cp Release InRelease` (step 4 above). The hook's fallback (download `.deb` and `dpkg -i`) works even when `apt` does not see the package.

2.9.2.6.3 Minimal LVFS Mirror on the Same Host

Use `LVFS_DIR` under the same `WEB_ROOT` as the APT repository. `hook.sh` can run `fwupdmgr refresh` and `fwupdmgr update` from this mirror over the LAN, in addition to any `.cab/.cap` from the USB `firmware/` directory.

2.9.2.6.3.1 On a Desktop (Server)

1. Populate `LVFS_DIR` for a full mirror. Download LVFS metadata and firmware into `LVFS_DIR` (one-time or whenever you want to refresh the mirror). Use either a `PULP_MANIFEST`-based sync or the `sync-pulp.py` helper with your LVFS account username and token; both are ways to pull the same class of content into `LVFS_DIR`. For concrete `sync-pulp.py` commands and options, refer to *Mirror the Full Ubuntu Ports and LVFS Content on a Server*. When you serve this tree over HTTP, the URL you publish as the mirror root must resolve to a directory that contains `firmware.xml.gz`, and the firmware binaries must appear at the paths that file references (often under a `downloads/` subdirectory). Alternatively, if you are not maintaining full LVFS metadata, place a set of firmware files (for example, from a signbinpack release) in a subdirectory such as `$LVFS_DIR/signbinpack-2.152.3-release/`. Point `lvfs-mirror.url` on the client at that subdirectory's URL. `hook.sh` can use that layout without `firmware.xml.gz`: it reads the directory listing and runs `fwupdmgr install` for each `.cab/.cap` file.
2. Serve the mirror over HTTP using the same web server as the APT repository (run from `WEB_ROOT`). For example, LVFS base URL `http://10.111.55.241:8000/lvfs-mirror/` (Python) or `http://10.111.55.241/lvfs-mirror/` (Nginx).
3. On the host that serves both trees, allow inbound HTTP on the ports you use for that server (typically the same ports you opened for the APT repository). For example:

```
sudo ufw allow 80/tcp
sudo ufw allow 8000/tcp
sudo ufw reload
```

2.9.2.6.3.2 On the DGX Spark Client

Place the following on the USB drive's `OEMDATA` partition when you want `hook.sh` on the client to use your hosted firmware mirror:

- `lvfs-mirror.url`: Include a single line with the base URL of the firmware mirror. If that URL serves `firmware.xml.gz`, the hook adds an `fwupd` remote, runs `fwupdmgr refresh`, and runs `fwupdmgr update`. If the URL points to a directory of `.cab/.cap` files only (no `firmware.xml.gz`), the hook fetches the directory listing, downloads each `.cab/.cap`, and runs `fwupdmgr install` for each file.

By default, `fwupd` installs only trusted (LVFS-signed) firmware. If installation of local or vendor `.cab/.cap` files fails with a message such as "firmware signature missing or not trusted" (for example, signbinpack content from the mirror or from USB `firmware/`), edit `/etc/fwupd/fwupd.conf` on the client and set `OnlyTrusted=false` under `[fwupd]`:

```
[fwupd]
OnlyTrusted=false
```

> **Note:** Use `OnlyTrusted=false` only when you control the firmware source and accept the risk.

2.9.2.7 Mirror the Full Ubuntu Ports and LVFS Content on a Server

This section describes a unified `apt` and LVFS layout under `~/mirror`, HTTP on port 8080, and clients "Spark A" (mirror server) / "Spark B" (client).

2.9.2.7.1 Server Directory Layout

Under the mirror root (for example, `tree -L 2 ~/mirror`):

```
.
    apt
        mirror
        skel
        var
    guids.txt              # optional: LVFS partial sync (--guid-file)
    lvfs                   # LVFS mirror (metadata + .cab)
    sync-pulp.py           # LVFS sync script (from LVFS upstream)
```

Client URL Patterns (Must Match Layout)

Service	URL Pattern
apt	`http://SER VER_IP:8080/apt/mirror/ports.ubuntu.com/ubuntu-ports/`
fwupd	`M etadataURI=http://SERVER_IP:8080/lvfs/<metadata-file>` and `FirmwareBaseURI=http://SERVER_IP:8080/lvfs`

If you rename `lvfs`, change both `MetadataURI` and `FirmwareBaseURI` on clients to match.

2.9.2.7.2 Create the Top-Level Tree and sync-pulp.py

```
mkdir -p ~/mirror/apt ~/mirror/lvfs
cd ~/mirror
wget -O sync-pulp.py https://gitlab.com/fwupd/lvfs-website/raw/master/contrib/
↪sync-pulp.py
chmod +x sync-pulp.py
```

Create `guids.txt` only for a partial LVFS sync (described later).

2.9.2.7.3 One-Shot Sync Script: spark-mirror-sync.sh

Copy `oemdata/spark-mirror-sync.sh` from your distribution package onto the Spark, or run it from a repository clone. Some trees place this file under `scripts/`; use the path that matches your bundle. Run as root (`sudo`); the script does not invoke `sudo` internally.

- Installs dependencies only if you pass `--install-deps` / `--install-apt-mirror`.
- Creates `${MIRROR_ROOT}/apt-mirror.list.spark` if missing (noble-proposed, `base_path` = `$MIRROR_ROOT/apt`).
- Runs `apt-mirror`, then `sync-pulp.py` into `$MIRROR_ROOT/lvfs`.
- Symlinks `/usr/local/bin/python` to `python3` for tools that expect `python`, and runs `sync-pulp.py` with `python3`.

```
export LVFS_USERNAME='you@example.com'
export LVFS_TOKEN='your-lvfs-token'
sudo -E ./spark-mirror-sync.sh --install-deps --install-apt-mirror   # first
↪run only
sudo -E ./spark-mirror-sync.sh
```

Default `MIRROR_ROOT` with `sudo` and without `-H` is `/root/mirror`. To mirror under a user home directory (for example, `/home/nvidia/mirror`):

```
sudo env MIRROR_ROOT=/home/nvidia/mirror ./spark-mirror-sync.sh
```

Optional: `APT_MIRROR_LIST`, `--skip-apt`, `--skip-lvfs`, `LVFS_CLEANUP=1` for `--cleanup` on LVFS. If `$MIRROR_ROOT/guids.txt` exists, the script passes `--guid-file` automatically.

2.9.2.7.4 APT Mirror (noble-proposed under ~/mirror/apt)

The packaged `/usr/bin/apt-mirror` on Ubuntu is often too old to mirror some DEP-11 paths (for example, `icons-64x64@2.tar`). Use the current upstream `apt-mirror` Perl script from GitHub.

```
sudo apt install -y perl wget
sudo cp -a /usr/bin/apt-mirror /usr/bin/apt-mirror.distpkg 2>/dev/null || true
sudo wget -O /usr/local/bin/apt-mirror https://raw.githubusercontent.com/apt-
↪mirror/apt-mirror/master/apt-mirror
sudo chmod +x /usr/local/bin/apt-mirror
```

Always run synchronization with `/usr/local/bin/apt-mirror`.

`apt-mirror` stores the Ubuntu tree under `$base_path/mirror/`. With `base_path` set to `~/mirror/apt`, the live archive path is `~/mirror/apt/mirror/ports.ubuntu.com/ubuntu-ports/`, matching the client URI after `http://SERVER_IP:8080/apt/`.

For /etc/apt/mirror.spark.list, copy docs/apt-mirror.list.spark-noble-proposed from the repository and edit base_path if your home directory differs:

```
set base_path /home/nvidia/mirror/apt

# Only noble-proposed for ports.ubuntu.com/ubuntu-ports (+ optional deb-src
↪if you mirror sources)

clean http://ports.ubuntu.com/ubuntu-ports
sudo /usr/local/bin/apt-mirror /etc/apt/mirror.spark.list
```

Re-run periodically (for example, through cron) when you need fresher packages. Allow **8080/tcp** from client subnets if a host firewall is enabled.

2.9.2.7.5 The LVFS (fwupd) Mirror Under ~/mirror/lvfs

```
sudo apt install -y python3 python3-requests python3-lxml
```

Full mirror (large, on the order of ~50 GB): Requires an LVFS account and user token (not your account password). Refer to the LVFS site for account and token issuance.

```
cd ~/mirror
./sync-pulp.py https://fwupd.org/downloads ~/mirror/lvfs \
  --username='your-email@example.com' \
  --token='YOUR_USER_TOKEN'
```

Re-run to update; existing valid files are skipped.

Optional: --cleanup removes files no longer in the manifest.

Partial mirror (GUID file): On a representative Spark, run sudo fwupdtool get-devices or fwupdmgr get-devices --show-all. Build ~/mirror/guids.txt, then:

```
./sync-pulp.py https://fwupd.org/downloads ~/mirror/lvfs \
  --username='your-email@example.com' \
  --token='YOUR_USER_TOKEN' \
  --guid-file=guids.txt
```

Rules for guids.txt**:** One UUID per line, exactly as printed (lowercase hex is acceptable). No # comments, no hardware hints after , no blank lines. Copy every Guid: line and every UUID inside GUIDs: blocks for devices you want mirrored. The same GUID often appears on more than one device (for example, several identical NICs). List each UUID only once. Include UUIDs for internal or updatable components whose firmware you want in the mirror (such as EC, TPM, UEFI capsules, NVMe, or dbx). Omit removable USB devices if you do not require LVFS content for that class of hardware.

Manual workflow: save the fwupdtool get-devices output; copy only xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx tokens; paste into guids.txt; run sort -u guids.txt -o guids.txt to sort the file and remove duplicate lines.

Optional JSON workflow (user session; requires jq):

```
fwupdmgr get-devices --json | jq -r '
  .. | objects | select(has("Guid")) | .Guid,
  (.. | objects | select(has("Guids")) | .Guids[]?)
' | sort -u > ~/mirror/guids.txt
```

Example `guids.txt` **for NVIDIA DGX Spark** (`spark-cr01`, `sudo fwupdtool get-devices`): The following listing reflects a typical Spark (Kingston USB flash drive, EC, four ConnectX-7 ports with the same four GUIDs repeated, Samsung NVMe, TPM, two UEFI ESRT firmware slots, and UEFI dbx), including that USB device:

```
09321615-5d32-5758-8308-52a4a7be8efc
095ba8dd-3778-52b4-9f32-02a67c210ce5
0eb9bda9-3010-493a-a6a8-b5e80eddf870
10ec82f4-ff64-5362-9e5d-688febf5dbb0
12029307-5bb1-5200-99a5-536f1be9d081
35abf34a-7ed8-51b2-ba1b-edef527d47e6
3d13c989-e6a8-4ead-95ee-921f09868f65
59007998-a3d7-54a3-b30e-eb3b77e2f351
5f106816-21fe-5d90-896a-175038b9256f
67d35028-ca5b-5834-834a-f97380381082
75b1af35-b88a-59d2-a3c7-a38537f8607f
93768061-87bf-5c78-b9ea-5b7a6301012b
b488217b-3895-4fc0-b1bf-ab7005a2d45a
b5e95689-ad65-5e57-8778-897f04396256
cfc0de0b-adb3-5060-ba22-e4010a78368f
dd1a238a-5f8e-46bd-9401-a88da99c5a96
```

Smaller mirror (same machine; omit DT microDuo 3C): Delete these three Kingston-only lines:

- `09321615-5d32-5758-8308-52a4a7be8efc`
- `5f106816-21fe-5d90-896a-175038b9256f`
- `75b1af35-b88a-59d2-a3c7-a38537f8607f`

`sync-pulp.py` cannot combine `--guid-file` and `--filter-tag` in one run; run twice if you need both, or consult LVFS offline documentation. Filtered syncs can occasionally leave orphaned metadata relative to `.jcat` pairs; verify pairs and repair with `wget` from `https://fwupd.org/downloads/` if needed. Refer to `docs/fwupd-lvfs-mirror-local.md` for metadata and JCat basename rules.

Choosing `MetadataURI` **for clients:** After synchronization, choose a metadata file that has a matching `.jcat` file with the same basename. Quick check:

```
cd ~/mirror/lvfs
for f in firmware*.xml.xz firmware*.xml.gz firmware*.xml.zst; do
  [ -f "$f" ] || continue
  [ -f "${f}.jcat" ] && echo "OK: $f"
done
```

2.9.2.7.6 Serve ~/mirror With Python

```
cd ~/mirror
python3 -m http.server 8080 --bind 0.0.0.0
```

- `apt` on the wire: `http://SERVER_IP:8080/apt/mirror/...`
- LVFS on the wire: `http://SERVER_IP:8080/lvfs/...`

Keep this process running (`tmux`, a systemd user unit, or equivalent) while clients update.

2.9.2.8 Client Configuration and hook.sh

2.9.2.8.1 Client APT Sources for the Full Mirror (DEB822)

Use a `.sources` file (DEB822 format), not a legacy `.list` file.

File: `/etc/apt/sources.list.d/local-mirror.sources`

```
# Ubuntu from local mirror (under web root .../apt/mirror/)
Types: deb deb-src
URIs: http://10.111.54.206:8080/apt/mirror/ports.ubuntu.com/ubuntu-ports/
Suites: noble-proposed
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

Omit `deb-src` from `Types` if you did not mirror sources. If the client must use only this mirror for Ubuntu, disable or move aside the stock `ubuntu.sources` (same approach as in `hook.sh` under `oemdata/`). Replace the example IP with your `SERVER_IP`.

2.9.2.8.2 The fwupd Local Remote and Disabling the Public LVFS

File: `/etc/fwupd/remotes.d/local-lvfs-mirror.conf`

Replace `<metadata>` with the actual metadata filename on your mirror:

```
[fwupd Remote]
Enabled=true
Type=download
Title=Local LVFS Mirror
MetadataURI=http://10.111.54.206:8080/lvfs/<metadata>
FirmwareBaseURI=http://10.111.54.206:8080/lvfs
sudo fwupdmgr disable-remote lvfs
# or: sudo mv /etc/fwupd/remotes.d/lvfs.conf /etc/fwupd/remotes.d/lvfs.conf.
↪disabled
sudo fwupdmgr refresh
fwupdmgr get-updates
sudo fwupdmgr update
```

To verify that the mirror serves the metadata file and its matching `.jcat` file, run the following `curl` commands and confirm that the HTTP responses are successful (for example, `200 OK`):

```
curl -I "http://10.111.54.206:8080/lvfs/$(basename "$(grep ^MetadataURI= /etc/
↪fwupd/remotes.d/local-lvfs-mirror.conf | cut -d= -f2-)")"
curl -I "http://10.111.54.206:8080/lvfs/$(basename "$(grep ^MetadataURI= /etc/
↪fwupd/remotes.d/local-lvfs-mirror.conf | cut -d= -f2-)").jcat"
```

2.9.2.8.3 hook.sh Automation for the Full Mirror Workflow

The repository includes `oemdata/hook.sh`, which:

1. Renames `/etc/apt/sources.list.d` to `/etc/apt/sources.list.d.org` once and recreates `sources.list.d`.
2. Writes `local-mirror.sources` and `local-lvfs-mirror.conf` using `MIRROR_SERVER_IP` (default `10.111.54.206`), port `8080`, and `LVFS_WEB_SUBDIR` (default `lvfs`).

3. Disables the `lvfs` remote, runs `apt-get update` and `fwupdmgr refresh`, then applies upgrades if any were pending.

hook.sh Exit Codes (Full Mirror Workflow)

Code	Meaning
0	Success; no updates applied
1	Success; at least one `apt` or `fwupd` update was applied
255	Failure (treat as -1 in 8-bit terms)

```
export MIRROR_SERVER_IP=10.111.54.206
export MIRROR_SERVER_PORT=8080
export LVFS_METADATA_NAME=firmware.xml.xz   # or firmware-08681-stable.xml.xz
export LVFS_WEB_SUBDIR=lvfs
sudo -E /path/to/oemdata/hook.sh
```

2.9.2.8.4 Cloud-Init Integration

Refer to `oemdata/cloud-init/seed/user-data`. That example runs the hook; if the exit code is 1, it logs mirror-setup success with `logger` and runs `sync`.

2.9.2.9 Security Considerations

- hook.sh **and USB contents:** The hook runs with elevated privileges and executes content from the OEMDATA partition. Anyone with physical access can replace `hook.sh`, Debian packages, or firmware on the USB drive. Treat the USB drive as trusted input: use tamper-aware handling, restrict who can prepare USB devices, or verify integrity (for example, hashes or signatures) if your policy requires it.
- **Local APT repository:** The hook adds the repository with `[trusted=yes]`, so packages from that repository are not signature-verified. Ensure the repository server and network are trusted.
- **Local firmware mirror:** Firmware from the mirror (or a directory of `.cab`/`.cap`) is installed by `fwupd`. If you set `OnlyTrusted=false`, unverified or vendor-signed firmware is allowed only when you control the firmware source and accept the risk.
- **URLs on the USB drive:** The values in `apt-repo.url` and `lvfs-mirror.url` identify servers on your network. Those servers must be trustworthy. If an attacker compromises one of those servers, or performs a man-in-the-middle (MITM) attack on the path between the client and the server, the client could install malicious packages or firmware.
- **Network exposure:** The host that serves the APT repository and firmware mirror is reachable from other systems on the same local network used for DGX Spark installation. Harden that host (access control, firewall, operating system updates). When your security policy requires it, place installation and mirror access on a dedicated or isolated network segment instead of a general-purpose LAN.
- **LVFS credentials:** Store LVFS_TOKEN and related credentials securely. Prefer environment variables or a secret manager instead of committing tokens to scripts or logs.
- **Mirror reachability:** Restrict mirror HTTP access (firewall, private network) if the mirror is not intended to be widely reachable.

2.9.2.10 Verify the Customization and Installation Outcomes

2.9.2.10.1 During and After an ISO-Based Installation

OEM ISO configuration is logged at `/var/log/oem-iso-cfg.log` when the installer runs `oem-iso-cfg.sh` from `/cdrom`.

2.9.2.10.2 After Mirror- or USB-Driven Updates

- Confirm that the expected Debian or Ubuntu packages are installed from the mirror.
- Confirm expected firmware versions after `fwupdmgr update`, as applicable.
- Inspect Cloud-Init logs for hook execution and errors.
- Disable Cloud-Init for subsequent boots if your operational model requires it, per your site policy.

2.9.2.11 Prepare the Installation Media and Client (Verification Flow)

Use the following checklist when validating an end-to-end flow:

1. Run `spark-mirror-sync.sh` to prepare the local APT and firmware sources when you use that workflow. Start the web server from the common parent directory for both `lvfs` and `apt`.
2. Prepare a bootable USB drive with your repack script and base OS. For example, a promoted QA ISO path, or from a publicly available Spark ISO file: `https://urm.nvidia.com/artifactory/sw-dgx-platform-generic-local/Promoted-to-QA-ISO/RemasterISO/dgx/7.4.0/noble/arm64/2026-01-26-16-04-58/DGXOS-7.4.0-2026-01-26-16-04-58-arm64.iso` For additional images, refer to DGX Base OS 7 documentation and release channels.
3. Add another partition on the same USB drive, mount it locally, and copy `hook.sh` into the mounted root directory, as required by your imaging procedure.
4. Flash the system boot package (SBP) to a known prior version if your test plan requires it (for example, `2.144.9`). For current package naming, refer to the DGX Spark Software Release Packages document, section 6, DGX Spark OTA1 Branch / OTA1.1.
5. Flash the client using the bootable USB drive.

2.9.2.12 Reference: OEM Scripts and Cloud-Init

The following listings are reference copies of scripts and configuration files from the DGX OS customization repository (paths under `os/` and `oemdata/`). They supplement *Client Configuration and hook.sh*, *Customize the BaseOS Image with repack_baseos.sh*, and *Cloud-Init Integration*. Compare with your repository checkout and release notes; behavior and paths can change between releases.

2.9.2.12.1 First Boot: OEMDATA hook.sh and Cloud-Init Seed

`hook.sh` lives on the `OEMDATA` partition; the example `seed/user-data` `runcmd` mounts that partition, copies the hook to `/tmp` for execution, exports `OEM_MNT`, and runs the copy. The `cfg.d` and `seed` files are representative OEM Cloud-Init content carried on the ISO and copied at install time.

2.9.2.12.1.1 oemdata/hook.sh

```
#!/bin/sh
# OEM hook script: run from cloud-init when OEMDATA partition is mounted.
# Copy this file to the root of the OEMDATA partition (next to debs/ and
```

(continues on next page)

(continued from previous page)

```
↪firmware/).
# Optional: apt-repo.url (full path to repo, e.g. …/deb-repo/), apt-packages.
↪txt;
# lvfs-mirror.url (base URL only, dir containing firmware.xml.gz, e.g. …/lvfs-
↪mirror/ – not a subpath).
# Spark unified mirror (apt + LVFS): see oe4t-cicd docs/spark-mirror-apt-fwupd-
↪unified.md
# Exit: 0 = success, no apt/fwupd updates in final pass; 1 = success and at
↪least one applied;
# 255 = failure (-1 in 8-bit).
# After a successful final pass, current sources.list.d is renamed to sources.
↪list.d.cldnt,
# stock apt is restored from sources.list.d.org, and public LVFS is re-enabled.
# Override: MIRROR_SERVER_IP, MIRROR_SERVER_PORT, LVFS_METADATA_NAME, LVFS_WEB_
↪SUBDIR
# OEM_MNT is the OEMDATA partition root (debs/, firmware/, urls). Normally the
↪directory
# containing this script; cloud-init may copy this file to /tmp and set OEM_MNT
↪explicitly.
# This tree does not use functions.sh. If you extend the USB copy, source
↪helpers only as
#   . "$OEM_MNT/your-helper.sh"
# never . "$(dirname "$0")/..." or files vanish when $0 is under /tmp.

# By using this script, the user agrees to the terms of the EULA, SOL is
↪enabled by default.

# By clicking Deploy, you also acknowledge and agree to the NVIDIA CUDA EULA
↪appended to the documentation, which governs the CUDA components included in
↪this deployment.

# Telemetry is not enabled by default. If enabling telemetry, the user agrees
↪to accept the telemetry terms.

OEM_MNT=${OEM_MNT:-$(dirname "$0")}
WIFI_ADAPTER=${WIFI_ADAPTER:-wlP9s9} # Default WiFi adapter name
# ------------------------------------------------------------------------------
# Defaults / mirror URLs (used by mirror + OEM stages)
# ------------------------------------------------------------------------------
_hook_config_defaults() {
 MIRROR_SERVER_IP=${MIRROR_SERVER_IP:-10.111.54.206}
 MIRROR_SERVER_PORT=${MIRROR_SERVER_PORT:-8080}
 LVFS_WEB_SUBDIR=${LVFS_WEB_SUBDIR:-lvfs}
 LVFS_METADATA_NAME=${LVFS_METADATA_NAME:-firmware.xml.xz}
 MIRROR_APT_URI="http://${MIRROR_SERVER_IP}:${MIRROR_SERVER_PORT}/apt/mirror/
↪ports.ubuntu.com/ubuntu-ports/"
 MIRROR_FW_BASE="http://${MIRROR_SERVER_IP}:${MIRROR_SERVER_PORT}/${LVFS_WEB_
↪SUBDIR}"
}
# ------------------------------------------------------------------------------
# HTTP GET to stdout (wget or curl)
# ------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
_hook_http_get() {
 wget -qO - "$1" 2>/dev/null || curl -sL "$1" 2>/dev/null
}
# ------------------------------------------------------------------------------
# OOBE post-steps (EXIT trap): run when user "nvidia" exists; never fail the
 ↪hook.
# ------------------------------------------------------------------------------
_hook_oobe_supplementary_groups() {
 usermod -aG adm,sudo,audio,dip,plugdev,users,lpadmin nvidia 2>/dev/null ||
 ↪true
}
_hook_oobe_spark_autostart_and_keyboard() {
 install -d -m 0755 -o nvidia -g nvidia /home/nvidia/.config/autostart 2>/dev/
 ↪null || true
 if [ ! -f /home/nvidia/.config/autostart/nvidia-spark-docs.desktop ]; then
   ( umask 022
 cat > /home/nvidia/.config/autostart/nvidia-spark-docs.desktop << 'EOF'
[Desktop Entry]
Type=Application
Name=NVIDIA Spark documentation
Exec=xdg-open https://build.nvidia.com/spark
X-GNOME-Autostart-enabled=true
EOF
   ) 2>/dev/null || true
   chown nvidia:nvidia /home/nvidia/.config/autostart/nvidia-spark-docs.
 ↪desktop 2>/dev/null || true
   chmod 0644 /home/nvidia/.config/autostart/nvidia-spark-docs.desktop 2>/dev/
 ↪null || true
 fi
 if [ -f "$OEM_MNT/oem-keyboard-spark.sh" ]; then
   echo "[oem hook] running $OEM_MNT/oem-keyboard-spark.sh"
   sh "$OEM_MNT/oem-keyboard-spark.sh" || true
 fi
}
_hook_oobe_skip_gnome_initial_setup() {
 install -d -m 0755 -o nvidia -g nvidia /home/nvidia/.config 2>/dev/null ||
 ↪true
 touch /home/nvidia/.config/gnome-initial-setup-done 2>/dev/null || true
 chown nvidia:nvidia /home/nvidia/.config/gnome-initial-setup-done 2>/dev/
 ↪null || true
}
_hook_oobe_hotspot_teardown_if_ethernet() {
 # Run as a child (not ".") so dgx-oobe sees $0 under /opt/nvidia/dgx-oobe
 ↪(functions.sh path).
 # Use bash: functions.sh uses bash syntax; /bin/sh (dash) errors with "("
 ↪unexpected.
 _hs=/opt/nvidia/dgx-oobe/oobe-hotspot-shutdown.sh
 if [ -f "$_hs" ]; then
   command -v bash >/dev/null 2>&1 && bash "$_hs" || true
 fi
}
_hook_oobe_disable_systemd_units() {
```

(continues on next page)

(continued from previous page)

```
 for u in dgx-oobe dgx-oobe-admin dgx-oobe-hotspot dgx-oobe-hostname dgx-oobe-
↪hotspot-watchdog; do
   systemctl stop "$u" 2>/dev/null || true
   systemctl disable "$u" 2>/dev/null || true
 done
 if [ -f /etc/NetworkManager/dnsmasq-shared.d/dgx-oobe.conf ]; then
   rm -f /etc/NetworkManager/dnsmasq-shared.d/dgx-oobe.conf
 fi
 systemctl restart avahi-daemon 2>/dev/null || true
 # Disable WiFi adapter scan
 if [ -z "${WIFI_ADAPTER}" ]; then
 return 0
 fi
 if ip link show ${WIFI_ADAPTER}_scan >/dev/null 2>&1; then
 /usr/bin/ip link set ${WIFI_ADAPTER}_scan down || true
 /usr/sbin/iw dev ${WIFI_ADAPTER}_scan del || true
 fi
}
_hook_oobe_ubuntu_pro_attach() {
 if [ -n "${UBUNTU_PRO_TOKEN:-}" ] && command -v pro >/dev/null 2>&1; then
   pro attach "$UBUNTU_PRO_TOKEN" --no-prompt 2>/dev/null || true
 fi
}
_hook_oobe_sol_if_consent() {

  echo "[oem hook] Enabling SOL"

  install -d -m 0755 /opt/nvidia/dgx-telemetry 2>/dev/null || true

  touch /opt/nvidia/dgx-telemetry/eula_accepted 2>/dev/null || true

  sync

  systemctl daemon-reload 2>/dev/null || true

  if ! systemctl enable --now nvidia-dgx-sol 2>/dev/null; then

echo "[oem hook] warning: systemctl enable --now nvidia-dgx-sol failed (check
↪status; unit may stay disabled)" >&2 || true

systemctl start nvidia-dgx-sol 2>/dev/null || true

  fi

}

_hook_oobe_telemetry_if_consent() {

  echo "[oem hook] Enabling telemetry"

  install -d -m 0755 /opt/nvidia/dgx-telemetry 2>/dev/null || true
```

(continues on next page)

(continued from previous page)

```
  touch /opt/nvidia/dgx-telemetry/technical_consent \

/opt/nvidia/dgx-telemetry/functional_consent 2>/dev/null || true

  sync

  systemctl daemon-reload 2>/dev/null || true

  if ! systemctl enable --now nvidia-dgx-telemetry 2>/dev/null; then

echo "[oem hook] warning: systemctl enable --now nvidia-dgx-telemetry failed
↪(check status; unit may stay disabled)" >&2 || true

systemctl start nvidia-dgx-telemetry 2>/dev/null || true

  fi

}
_hook_oobe_complete_flag_marker() {
 install -d -m 0755 /opt/nvidia/dgx-oobe 2>/dev/null || true
 touch /opt/nvidia/dgx-oobe/oobe-complete-flag 2>/dev/null || true
}
# When cloud-init created user "nvidia", run one-time OOBE-aligned steps on
↪every script exit.
# (EXIT runs after normal completion, exit 1, or exit 255 so these steps still
↪run.)
_hook_oobe_post() {
 set +e
 if [ "$(id -u)" -ne 0 ]; then
   return 0
 fi
 if ! getent passwd nvidia >/dev/null 2>&1; then
   return 0
 fi
 echo "[oem hook] OOBE post-steps for user nvidia (EXIT trap)"
 _hook_oobe_supplementary_groups || true
 _hook_oobe_spark_autostart_and_keyboard || true
 _hook_oobe_skip_gnome_initial_setup || true
 _hook_oobe_hotspot_teardown_if_ethernet || true
 _hook_oobe_disable_systemd_units || true
 _hook_oobe_ubuntu_pro_attach || true
 _hook_oobe_sol_if_consent || true
 _hook_oobe_complete_flag_marker || true
 return 0
}
trap '_hook_oobe_post' EXIT
# ------------------------------------------------------------------------------
# Unified Spark mirror: local apt + fwupd LVFS remote
# ------------------------------------------------------------------------------
_hook_mirror_archive_stock_sources() {
 if [ ! -d /etc/apt/sources.list.d.org ]; then
   if [ -d /etc/apt/sources.list.d ]; then
```

(continues on next page)

(continued from previous page)

```
 mv /etc/apt/sources.list.d /etc/apt/sources.list.d.org
   fi
 fi
 rm -rf /etc/apt/sources.list.d
 mkdir -p /etc/apt/sources.list.d
}
_hook_mirror_write_deb822_sources() {
 cat > /etc/apt/sources.list.d/local-mirror.sources <<EOF
# Ubuntu from local mirror (under web root .../apt/mirror/)
Types: deb deb-src
URIs: ${MIRROR_APT_URI}
Suites: noble-proposed
Components: main restricted universe multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
EOF
}
_hook_mirror_write_fwupd_local_remote() {
 mkdir -p /etc/fwupd/remotes.d
 cat > /etc/fwupd/remotes.d/local-lvfs-mirror.conf <<EOF
[fwupd Remote]
Enabled=true
Type=download
Title=Local LVFS Mirror
MetadataURI=${MIRROR_FW_BASE}/${LVFS_METADATA_NAME}
FirmwareBaseURI=${MIRROR_FW_BASE}
EOF
}
_hook_mirror_disable_public_lvfs() {
 fwupdmgr disable-remote lvfs 2>/dev/null || {
   [ -f /etc/fwupd/remotes.d/lvfs.conf ] && mv /etc/fwupd/remotes.d/lvfs.conf
↪/etc/fwupd/remotes.d/lvfs.conf.disabled
 }
}
_hook_mirror_apply_local_mirror() {
 _hook_mirror_archive_stock_sources
 _hook_mirror_write_deb822_sources
 _hook_mirror_write_fwupd_local_remote
 _hook_mirror_disable_public_lvfs
}
# Succeed if either update works; exit 255 only when both fail.
_hook_apt_update_initial() {
 if apt-get update -o Acquire::Languages=none || apt-get update; then
   return 0
 fi
 exit 255
}
# ------------------------------------------------------------------------
# OEMDATA: local .deb drop, optional apt repo, firmware USB, optional LVFS URL
# ------------------------------------------------------------------------
_hook_oem_install_debs_from_usb() {
 echo "Checking for debs in USB OEMDATA partition..."
 if [ -d "$OEM_MNT/debs" ] && ls "$OEM_MNT/debs"/*.deb >/dev/null 2>&1; then
```

(continues on next page)

(continued from previous page)

```
    echo "Installing debs from USB OEMDATA partition..."
    dpkg -i "$OEM_MNT/debs"/*.deb || true
    apt-get install -f -y
  fi
}
_hook_oem_install_from_local_repo() {
  echo "Checking for local APT repo..."
  if [ ! -f "$OEM_MNT/apt-repo.url" ]; then
    echo "apt-repo.url not found"
    return 0
  fi
  repo_url=$(sed -n '1s/[[:space:]]*//p' "$OEM_MNT/apt-repo.url")
  if [ -z "$repo_url" ]; then
    echo "No local APT repo URL found"
    return 0
  fi
  echo "Adding local APT repo: $repo_url"
  printf 'deb [trusted=yes] %s ./\n' "$repo_url" > /etc/apt/sources.list.d/oem-
↪local.list
  repo_host=$(echo "$repo_url" | sed -n 's|.*://\([^:/]*\).*|\1|p')
  if [ -n "$repo_host" ]; then
    rm -f /var/lib/apt/lists/partial/*"$repo_host"* \
  /var/lib/apt/lists/*"$repo_host"* 2>/dev/null || true
  fi
  apt update -o Acquire::Languages=none || apt update || true
  if [ ! -f "$OEM_MNT/apt-packages.txt" ]; then
    echo "apt-packages.txt not found; apt upgrade using OEM local repo only
↪(single-source apt)"
    apt upgrade -y \
  -o Dir::Etc::sourcelist="/etc/apt/sources.list.d/oem-local.list" \
  -o APT::Architecture="$(dpkg --print-architecture)" \
  || true
    return 0
  fi
  pkgs=$(grep -v '^[#;]' "$OEM_MNT/apt-packages.txt" | while read -r line; do
    line="${line%%[[:space:]]*}"
    [ -z "$line" ] && continue
    case "$line" in *\.deb) line="${line%.deb}"; line="${line%_*}"; line="$
↪{line%_*}"; esac
    echo "$line"
  done | tr '\n' ' ')
  if [ -z "$pkgs" ]; then
    return 0
  fi
  echo "Installing packages from local repo: $pkgs"
  if ! apt-get install -y $pkgs; then
    echo "Fallback: downloading .deb and installing with dpkg..."
    base="${repo_url%/}"
    for pkg in $pkgs; do
  pkg_file=$(
    _hook_http_get "$base/Packages.gz" | zcat 2>/dev/null | awk -v pkg="$pkg" '
/^Package: /{name=$2}
```

(continues on next page)

(continued from previous page)

```
/^Filename: /{if(name==pkg){print $2; exit}}
/^$/{name=""}
'
 )
 pkg_file="${pkg_file#./}"
 [ -z "$pkg_file" ] && continue
 tmp_deb="/tmp/$(basename "$pkg_file")"
 if _hook_http_get "$base/$pkg_file" > "$tmp_deb" 2>/dev/null && [ -s "$tmp_
↪deb" ]; then
   dpkg -i "$tmp_deb" && echo "PASS: $pkg (dpkg)" || true
 else
   echo "FAIL: could not download $pkg"
 fi
 rm -f "$tmp_deb"
   done
   apt-get install -f -y 2>/dev/null || true
 fi
}
_hook_oem_install_firmware_usb() {
 echo "Checking for firmware in USB OEMDATA partition..."
 if [ ! -d "$OEM_MNT/firmware" ]; then
   return 0
 fi
 echo "Installing firmware from USB OEMDATA partition..."
 find "$OEM_MNT/firmware" -maxdepth 1 -type f \( -name '*.cab' -o -name '*.cap
↪' \) | while read -r f; do
   name=$(basename "$f")
   echo "Installing firmware: $name"
   if fwupdmgr install --allow-reinstall "$f"; then
 echo "PASS: $name"
   else
 echo "FAIL: $name"
   fi
 done
}
# Supports (1) full LVFS mirror: URL points to dir with firmware.xml.gz; (2)
↪directory of .cab/.cap only.
_hook_oem_lvfs_mirror_from_url() {
 echo "Checking for LVFS mirror URL..."
 if [ ! -f "$OEM_MNT/lvfs-mirror.url" ]; then
   return 0
 fi
 lvfs_base=$(sed -n '1s/[[:space:]]*//p' "$OEM_MNT/lvfs-mirror.url")
 lvfs_base="${lvfs_base%/}/"
 if [ -z "$lvfs_base" ]; then
   return 0
 fi
 lvfs_meta_url="${lvfs_base}firmware.xml.gz"
 curl_meta_code=$(curl -sI -o /dev/null -w '%{http_code}' "$lvfs_meta_url" 2>/
↪dev/null)
 if ( wget -q --spider "$lvfs_meta_url" 2>/dev/null ) || [ "$curl_meta_code"
↪= "200" ]; then
```

(continues on next page)

(continued from previous page)

```
    echo "Adding LVFS mirror (metadata): $lvfs_base"
    mkdir -p /etc/fwupd/remotes.d
    cat > /etc/fwupd/remotes.d/oem-lvfs-mirror.conf << EOF
[fwupd Remote]
Title=OEM LVFS Mirror
MetadataURI=${lvfs_base}firmware.xml.gz
FirmwareBaseURI=$lvfs_base
Enabled=true
EOF
    echo "Refreshing fwupd and upgrading firmware from mirror..."
    fwupdmgr refresh --force || fwupdmgr refresh
    fwupdmgr update || true
  else
    echo "No firmware.xml.gz at $lvfs_base; treating as directory of .cab/.cap.
↪.."
    _hook_http_get "$lvfs_base" | grep -oE 'href="[^"]*\.(cab|cap)"' | sed 's/
↪href="//;s/"$//' | while read -r f; do
  [ -z "$f" ] && continue
  tmp_f="/tmp/$(basename "$f")"
  if _hook_http_get "$lvfs_base$f" > "$tmp_f" 2>/dev/null && [ -s "$tmp_f" ];
↪then
    echo "Installing firmware from mirror: $f"
    fwupdmgr install --allow-reinstall "$tmp_f" && echo "PASS: $f" || echo
↪"FAIL: $f"
  fi
  rm -f "$tmp_f"
    done
  fi
}
# ------------------------------------------------------------------------------
# Final pass: apt upgrade + fwupd loop; sets HOOK_APT_UPDATED, HOOK_FW_UPDATED,
↪HOOK_FW_FAILED
# ------------------------------------------------------------------------------
_hook_final_apt_upgrade() {
  echo "apt update -o Acquire::Languages=none || apt update"
  if apt update -o Acquire::Languages=none || apt update; then
    :
  else
    echo "apt update failed"
    exit 255
  fi
  echo "apt -s upgrade | grep -q '^[[:space:]]*Inst '"
  if apt -s upgrade | grep -q '^[[:space:]]*Inst '; then
    DEBIAN_FRONTEND=noninteractive apt -y -o Dpkg::Options::=--force-confold
↪upgrade || exit 255
    HOOK_APT_UPDATED=1
  fi
}
_hook_fwupdmgr_refresh_and_count() {
  echo "fwupdmgr refresh --force"
  fwupdmgr refresh --force
  HOOK_FW_APPLICABLE=""
```

(continues on next page)

(continued from previous page)

```
 if command -v jq >/dev/null 2>&1; then
   _hook_jq_fw_count='(if type == "object" and (.Devices | type) == "array"
↪then .Devices '
   _hook_jq_fw_count="${_hook_jq_fw_count}"'elif type == "array" then . else
↪[] end) | '
   _hook_jq_fw_count="${_hook_jq_fw_count}"'[.[] | select((.Releases // []) |
↪length > 0)] | length'
   HOOK_FW_APPLICABLE=$(
 fwupdmgr get-upgrades --json 2>/dev/null | jq -r "$_hook_jq_fw_count" 2>/dev/
↪null || echo ""
   )
 fi
 HOOK_FW_APPLICABLE=$(printf '%s' "$HOOK_FW_APPLICABLE" | tr -d '\r\n\t ')
 case "$HOOK_FW_APPLICABLE" in
   ''|*[!0-9]*) HOOK_FW_APPLICABLE="?" ;;
 esac
 echo "fwupdmgr: applicable firmware devices (Releases>0): ${HOOK_FW_
↪APPLICABLE:-?}"
}
_hook_fwupdmgr_upgrade_loop() {
 set +e
 case "$HOOK_FW_APPLICABLE" in
   [1-9]|[1-9][0-9]*)
 HOOK_FW_ITER=0
 while [ "$HOOK_FW_ITER" -lt 5 ]; do
   HOOK_FW_ITER=$((HOOK_FW_ITER + 1))
   HOOK_FW_OFFLINE_CAN_BREAK=0
   HOOK_FW_IMMEDIATE_CAN_BREAK=0
   echo "fwupdmgr upgrade -y --offline"
   fwupdmgr upgrade -y --offline
   HOOK_FW_R1=$?
   echo "fwupdmgr upgrade -y --no-reboot-check"
   fwupdmgr upgrade -y --no-reboot-check
   HOOK_FW_R2=$?
   case $HOOK_FW_R1 in
     0) HOOK_FW_UPDATED=1; HOOK_FW_OFFLINE_CAN_BREAK=1 ;;
     2) HOOK_FW_OFFLINE_CAN_BREAK=1 ;;
     *) echo "fwupdmgr upgrade -y --offline failed (exit $HOOK_FW_R1)"; HOOK_
↪FW_FAILED=1 ;;
   esac
   case $HOOK_FW_R2 in
     0) HOOK_FW_UPDATED=1; HOOK_FW_IMMEDIATE_CAN_BREAK=1 ;;
     2) HOOK_FW_IMMEDIATE_CAN_BREAK=1 ;;
     *) echo "fwupdmgr upgrade -y --no-reboot-check failed (exit $HOOK_FW_R2)
↪"; HOOK_FW_FAILED=1 ;;
   esac
   if [ "$HOOK_FW_FAILED" -eq 1 ]; then
     break
   fi
   if [ "$HOOK_FW_OFFLINE_CAN_BREAK" -eq 1 ] && [ "$HOOK_FW_IMMEDIATE_CAN_
↪BREAK" -eq 1 ]; then
     echo "fwupdmgr upgrade -y --offline and fwupdmgr upgrade -y --no-reboot-
```

(continues on next page)

(continued from previous page)

```
↪check both finished OK"
    break
  fi
 done
 ;;
 esac
 set -e
}
_hook_restore_stock_apt_and_lvfs() {
 echo "Restoring stock apt sources and re-enabling public LVFS after mirror-
↪based updates"
 if [ -d /etc/apt/sources.list.d.org ]; then
  if [ -d /etc/apt/sources.list.d.cldnt ]; then
 rm -rf /etc/apt/sources.list.d.cldnt
  fi
  if [ -d /etc/apt/sources.list.d ]; then
 mv /etc/apt/sources.list.d /etc/apt/sources.list.d.cldnt
  fi
  mv /etc/apt/sources.list.d.org /etc/apt/sources.list.d
 fi
 if [ -f /etc/fwupd/remotes.d/lvfs.conf.disabled ]; then
  mv /etc/fwupd/remotes.d/lvfs.conf.disabled /etc/fwupd/remotes.d/lvfs.conf
 fi
 # Non-interactive: enable-remote otherwise blocks on "Enable new remote?" (no
↪TTY under cloud-init).
 # The LVFS disclaimer box goes to stdout; cloud-init captures runcmd output
↪into cloud-init-provisioning.log.
 fwupdmgr -y --no-remote-check enable-remote lvfs >/dev/null 2>&1 || true
}
_hook_exit_with_status() {
 if [ "$HOOK_FW_FAILED" -eq 1 ]; then
  exit 255
 fi
 if [ "$HOOK_APT_UPDATED" -eq 1 ] || [ "$HOOK_FW_UPDATED" -eq 1 ]; then
  exit 1
 fi
 exit 0
}
# ------------------------------------------------------------------------------
# Main
# ------------------------------------------------------------------------------
_hook_main() {
 _hook_config_defaults
 _hook_mirror_apply_local_mirror
 _hook_oem_install_debs_from_usb
 _hook_oem_install_from_local_repo
 _hook_oem_install_firmware_usb
 _hook_oem_lvfs_mirror_from_url
 HOOK_APT_UPDATED=0
 HOOK_FW_UPDATED=0
 HOOK_FW_FAILED=0
 _hook_apt_update_initial
```

(continues on next page)

(continued from previous page)

```
 _hook_final_apt_upgrade
 _hook_fwupdmgr_refresh_and_count
 _hook_fwupdmgr_upgrade_loop
 _hook_restore_stock_apt_and_lvfs
 _hook_exit_with_status
}
_hook_main
```

2.9.2.12.2 ISO Install: oem-iso-cfg.sh, repack_baseos.sh (excerpt), and OEM Cloud-Init on the ISO

`oem-iso-cfg.sh` runs during installation from the repacked ISO (Subiquity or autoinstall) with `/cdrom` mounted. The `repack_baseos.sh` excerpt shows how OEM `.deb` packages, the `cloud-init` tree, and `oem-iso-cfg.sh` are placed under `ISO_ROOT/oemdata/`.

2.9.2.12.2.1 oemdata/oem-iso-cfg.sh

```
#!/bin/bash

# SPDX-FileCopyrightText: Copyright (c) 2025 NVIDIA CORPORATION & AFFILIATES.
↪All rights reserved.
# SPDX-License-Identifier: MIT
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.



set -euo pipefail
set -x
export DEBIAN_FRONTEND=noninteractive


LOGFILE=/var/log/oem-iso-cfg.log
trap 'echo "[oem][fatal] script failed at line $LINENO" | tee -a "$LOGFILE" >&
↪2; exit 1' ERR
```

(continues on next page)

(continued from previous page)

```
exec > >(tee -a "$LOGFILE") 2>&1


echo "[oem] Starting customization script"
OEM_DEB_SRC=/cdrom/oemdata/debs
OEM_CLOUD_SRC=/cdrom/oemdata/cloud-init
OEM_CLOUD_CFG_SRC="$OEM_CLOUD_SRC/cfg.d"
OEM_CLOUD_SEED_SRC="$OEM_CLOUD_SRC/seed"
OEM_NOCLOUD_DST=/var/lib/cloud/seed/nocloud
OEM_CFG_DST=/etc/cloud/cloud.cfg.d


if [ -d "$OEM_CLOUD_SRC" ]; then
   echo "[oem] Detected OEM cloud-init configuration at $OEM_CLOUD_SRC"
   find "$OEM_CLOUD_SRC"


   echo "[oem] Enabling cloud-init NoCloud seed and config"
   mkdir -p -v "$OEM_NOCLOUD_DST" "$OEM_CFG_DST"


   if [ -d "$OEM_CLOUD_SEED_SRC" ]; then
       echo "[oem] Copying seed files from $OEM_CLOUD_SEED_SRC -> $OEM_
↪NOCLOUD_DST"
       cp -av "$OEM_CLOUD_SEED_SRC"/. "$OEM_NOCLOUD_DST"/
   else
       echo "[oem] No seed directory found at $OEM_CLOUD_SEED_SRC"
   fi


   if [ -d "$OEM_CLOUD_CFG_SRC" ]; then
       echo "[oem] Copying cfg files from $OEM_CLOUD_CFG_SRC -> $OEM_CFG_DST"
       cp -av "$OEM_CLOUD_CFG_SRC"/. "$OEM_CFG_DST"/
   else
       echo "[oem] No cfg.d directory found at $OEM_CLOUD_CFG_SRC"
   fi


   echo "[oem] cloud-init OEM configuration enabled."
else
   echo "[oem] No OEM cloud-init configuration found."
fi


echo "[oem] oemdata contents"
find /cdrom/oemdata -type f


echo "[oem] Copy fastos-release"
cp -v -f /cdrom/oemdata/fastos-release /etc/fastos-release
```

(continues on next page)

(continued from previous page)

```
BUILD_TYPE=$(cat /cdrom/oemdata/build_type | tr '[:upper:]' '[:lower:]')
echo "[oem] Build Type: ${BUILD_TYPE}"


echo "[oem] Check for Developer Tools"
if [ -d /cdrom/oemdata/devtools ]; then
   echo "[oem] Developer Tools Found"
   if [ "${BUILD_TYPE}" = "developer" ]; then
       echo "[oem] Developer Tools Steps"
       pushd /cdrom/oemdata/devtools

       if [ -f "NVIDIA-Linux-aarch64-*.run" ]; then
           NV_VER=$(ls NVIDIA-Linux-aarch64-*.run | head -n1)
           echo "[oem] Install gcc make unzip"
           apt install -y --no-install-recommends --allow-unauthenticated gcc
↪make unzip

           echo "[oem] Install NVIDIA Driver"
           sh ./${NV_VER} -v
           sh ./${NV_VER} --sb --no-rebuild-initramfs --no-check-for-
↪alternate-installs
       else
           echo "[oem] NVIDIA Driver .run not found, skipping"
       fi


       if [ -f "cuda_13*linux_sbsa.run" ]; then
           CUDA_VER=$(ls cuda_13*linux_sbsa.run | head -n1)
           echo "[oem] Install CUDA"
           chroot / sh /cdrom/oemdata/devtools/${CUDA_VER} --silent
       else
           echo "[oem] CUDA .run not found, skipping"
       fi


       NVP_VER=$(ls NVPunish*.zip | head -n1)
       if [ -f ./${NVP_VER} ]; then
           echo "[oem] Install NVPunish to /opt/nvidia/nvp"
           mkdir -p /opt/nvidia/nvp
           chmod ugo+rx /opt/nvidia/nvp
           pushd /opt/nvidia/nvp
           unzip -q /cdrom/oemdata/devtools/${NVP_VER}
           popd
       else
           echo "[oem] NVPunish .zip not found, skipping"
       fi


       popd
   else
       echo "[oem] Not developer build, skipping developer tools"
   fi
```

(continues on next page)

(continued from previous page)

```
fi


echo "[oem] Install packages"
pushd ${OEM_DEB_SRC}


PKGS_FILE=/cdrom/oemdata/pkgs.txt
if [ -f /cdrom/oemdata/pkgs.${BUILD_TYPE}.txt ]; then
   PKGS_FILE=/cdrom/oemdata/pkgs.${BUILD_TYPE}.txt
fi
echo "[oem] Using packages file: ${PKGS_FILE}"
cat ${PKGS_FILE}


if ls "$OEM_DEB_SRC"/*.deb >/dev/null 2>&1; then
 echo "[oem] Installing OEM packages from $OEM_DEB_SRC"


 # Prepare isolated APT environment
 APT_DIR="$(mktemp -d /tmp/oem-apt-XXXXXX)"
 APT_ARCH="$(dpkg --print-architecture)"
 mkdir -p "$APT_DIR/lists" "$APT_DIR/cache" "$APT_DIR/state" "$APT_DIR/debs"
 if [ "${BUILD_TYPE}" = "display" ]; then
   echo "[oem] Copy Display Missing Debs Packages for nvidia-settings"
   ls -l /cdrom/
   cp -rvf /cdrom/pool/main/libv/libvdpau/* "$APT_DIR/debs"
   cp -rvf /cdrom/pool/main/p/pkgconf/* "$APT_DIR/debs"
   cp -rvf /cdrom/pool/main/s/screen-resolution-extra/* "$APT_DIR/debs"


   echo "[oem] Copy Display Missing Debs Packages for nv-docker-gpus"
   cp -rvf /cdrom/pool/main/n/nv-docker-options/* "$APT_DIR/debs"


   # echo "[oem] Copy Display Missing Debs Packages for nvidia-xconfig"
   # cp -rvf /cdrom/pool/main/n/nvidia-xconfig/* "$APT_DIR/debs" || echo
↪"[oem] nvidia-xconfig not found baseos"
   # cp -rvf /cdrom/oemdata/debs/nvidia-xconfig* "$APT_DIR/debs" || echo
↪"[oem] nvidia-xconfig not found oemdata"
   # cp -rvf /cdrom/oemdata/debs/libnvidia-cfg1* "$APT_DIR/debs" || echo
↪"[oem] libnvidia-cfg1 not found"


   echo "[oem] Copy Display Missing Debs Packages for nvidia-conf-xconfig"
   cp -rvf /cdrom/pool/main/n/nvidia-conf-xconfig/* "$APT_DIR/debs"
 fi
 cp -a "$OEM_DEB_SRC"/.  "$APT_DIR/debs"
 cd "$APT_DIR/"
 dpkg-scanpackages debs /dev/null > "$APT_DIR/Packages"
 TEMP_SOURCE_LIST="$APT_DIR/oemrepo.list"
 echo "deb [trusted=yes] file:$APT_DIR ./" > "$TEMP_SOURCE_LIST"
```

(continues on next page)

(continued from previous page)

```
 # Get list of packages
 #PKGLIST=$(for deb in $APT_DIR/debs/*.deb; do dpkg -f "$deb" Package; done |
↪xargs)
 PKGLIST=$(cat ${PKGS_FILE} | grep -v "^#")


 echo "Installed packages: $PKGLIST"


 # Update and install
 apt-get update \
     -o Dir::Etc::sourcelist="$TEMP_SOURCE_LIST" \
     -o Dir::Etc::sourceparts="-" \
     -o Dir::State="$APT_DIR/state" \
     -o Dir::Cache="$APT_DIR/cache" \
     -o Dir::State::Lists="$APT_DIR/lists" \
     -o APT::Architecture="$APT_ARCH" || exit 100


 echo "[oem] Installing OEM packages from $OEM_DEB_SRC"
 echo "================================================"
 ls -l "$APT_DIR/debs"
 echo "================================================"


 echo "$PKGLIST" | xargs -r apt-get install -y --allow-downgrades \
     -o Dir::Etc::sourcelist="$TEMP_SOURCE_LIST" \
     -o Dir::Etc::sourceparts="-" \
     -o Dir::State="$APT_DIR/state" \
     -o Dir::Cache="$APT_DIR/cache" \
     -o Dir::State::Lists="$APT_DIR/lists" \
     -o APT::Architecture="$APT_ARCH" || exit 101


 # Clean up temp APT environment (do NOT remove OEM_DEB_SRC)
 rm -rf "$APT_DIR"
fi


if [ -f /cdrom/oemdata/post.${BUILD_TYPE}.sh ]; then
   echo "[oem] Run post-install script"
   /cdrom/oemdata/post.${BUILD_TYPE}.sh
fi


echo "[oem] Log installed packages"
apt list --installed > /var/log/oem-installed-packages.log


echo "[oem] Customization complete."
```

(continues on next page)

(continued from previous page)

```
exit 0
```

2.9.2.12.2.2 repack_baseos.sh (Partial Excerpt)

```
# Step 4: Install OEM debs and OEM cloud-init / oem-iso-cfg.sh into ISO
↪oemdata
install_oemdebs() {
   echo ""
   echo "Step 4: Installing OEM debs and cloud-init into ISO oemdata..."
   mkdir -p "$ISO_ROOT/oemdata/debs/"
   cp -rf "$OEM_DEBS_DIR"/*.deb "$ISO_ROOT/oemdata/debs/" 2>/dev/null || echo
↪"Warning: Some packages may not have been copied"
   echo "OEM debs copied to ISO oemdata directory."


   # Copy repo oemdata/cloud-init/ (full tree: seed/, cfg.d/, etc.) and oem-
↪iso-cfg.sh so they are on the repacked ISO
   if [[ -d "$OEMDATA_SRC" ]]; then
       if [[ -d "$OEMDATA_SRC/cloud-init" ]]; then
           echo "Copying OEM cloud-init tree from $OEMDATA_SRC/cloud-init to
↪ISO..."
           rm -rf "$ISO_ROOT/oemdata/cloud-init"
           cp -a "$OEMDATA_SRC/cloud-init" "$ISO_ROOT/oemdata/cloud-init"
           echo "  ✓ cloud-init directory copied (seed/, cfg.d/, and all
↪files)."
       fi
       if [[ -f "$OEMDATA_SRC/oem-iso-cfg.sh" ]]; then
           echo "Copying oem-iso-cfg.sh to ISO..."
           cp "$OEMDATA_SRC/oem-iso-cfg.sh" "$ISO_ROOT/oemdata/"
           echo "  ✓ oem-iso-cfg.sh copied."
           # How it is called: the BaseOS installer (Subiquity or
↪autoinstall) runs this script during
           # install when the ISO is mounted at /cdrom. It runs in the target
↪(installed) system
           # context: installs OEM debs from /cdrom/oemdata/debs/ and copies
↪cloud-init seed from
           # /cdrom/oemdata/cloud-init/ to /var/lib/cloud/seed/nocloud and
↪cloud.cfg.d. Log: /var/log/oem-iso-cfg.log
       fi
   else
       echo "Warning: Repo oemdata not found at $OEMDATA_SRC (cloud-init will
↪not be added)."
   fi


   echo "========================================="
   echo "OEM debs contents:"
   ls -l "$ISO_ROOT/oemdata/debs/"
   echo "OEM cloud-init contents (used by cloud-init service at first boot):"
   find "$ISO_ROOT/oemdata/cloud-init" -type f 2>/dev/null | sort || true
   echo "========================================="
```

(continues on next page)

(continued from previous page)

```
}


# Step 5: Repack the ISO
repack_iso() {
   echo ""
   echo "Step 5: Repacking the ISO..."
   OUTPUT_ISO="$PWD/${VOLUME_ID}-repacked-$(date +%Y-%m-%d-%H-%M-%S).iso"
   if [[ "$DEBUG" == "true" ]]; then
       xorriso -as mkisofs \
           -iso-level 3 \
           -allow-lowercase \
           -volid "$VOLUME_ID" \
           -J \
           -joliet-long \
           -l \
           -c boot/boot.cat \
           -partition_offset 16 \
           -append_partition 2 0xef "$CDBOOT_EXTRACT/usr/share/cd-boot-images-
↪arm64/images/boot/grub/efi.img" \
           -e --interval:appended_partition_2:all:: \
           -no-emul-boot \
           -partition_cyl_align all \
           -o "$OUTPUT_ISO" \
           "$ISO_ROOT"
   else
       xorriso -as mkisofs \
           -iso-level 3 \
           -allow-lowercase \
           -volid "$VOLUME_ID" \
           -J \
           -joliet-long \
           -l \
           -c boot/boot.cat \
           -partition_offset 16 \
           -append_partition 2 0xef "$CDBOOT_EXTRACT/usr/share/cd-boot-images-
↪arm64/images/boot/grub/efi.img" \
           -e --interval:appended_partition_2:all:: \
           -no-emul-boot \
           -partition_cyl_align all \
           -o "$OUTPUT_ISO" \
           "$ISO_ROOT" >"$VERBOSE_OUTPUT" 2>&1
   fi
   echo ""
   echo "========================================="
   echo "ISO repacking complete!"
   echo "Output ISO: $OUTPUT_ISO"
   echo "========================================="
}


main() {
```

(continues on next page)

(continued from previous page)

```
    ...
    install_oemdebs
    repack_iso
    ...
}
```

Example OEM Cloud-Init files on the ISO under `oemdata/cloud-init/` (copied to the installed system by `oem-iso-cfg.sh`):

2.9.2.12.2.3 oemdata/cloud-init/cfg.d/50-dgx-base-audit.cfg

```
#cloud-config
output:
 all: "| tee -a /var/log/cloud-init-provisioning.log"


write_files:
 - path: /var/lib/cloud/scripts/per-instance/50-dgx-base-audit.sh
   permissions: '0755'
   content: |
     #!/bin/sh
     set -eu
     mkdir -p /var/log/provisioning
     audit=/var/log/provisioning/provisioning_audit.txt
     {
       echo "Base image cloud-init completed at: $(date -u +%Y-%m-%dT%H:%M:
↪%SZ)"
       echo "Hostname: $(hostname)"
       echo "Datasource: $(cloud-init query datasource || true)"
       echo "Instance ID: $(cloud-init query instance_id || true)"
     } > "$audit"
     chmod 0644 "$audit"
```

2.9.2.12.2.4 oemdata/cloud-init/cfg.d/50-oem-default-user.cfg

```
# Use nvidia as the default user instead of ubuntu (developer flavor).
# Ensures console and system default user is nvidia so login works after
↪install.
system_info:
 default_user:
   name: nvidia
   groups: [sudo]
   shell: /bin/bash
   lock_passwd: false
```

2.9.2.12.2.5 oemdata/cloud-init/cfg.d/99-oem-nocloud.cfg

```
# Tell cloud-init to use NoCloud and read seed from /var/lib/cloud/seed/
↪nocloud/
# Without this, cloud-init reports DataSourceNone and ignores the seed files.
```

(continues on next page)

(continued from previous page)

```
datasource_list: [NoCloud]
datasource:
 NoCloud:
   seedfrom: file:///var/lib/cloud/seed/nocloud/
```

2.9.2.12.2.6 oemdata/cloud-init/seed/meta-data

```
# instance-id is used by cloud-init as a unique instance identifier.

instance-id: oem-spark-01
```

2.9.2.12.2.7 oemdata/cloud-init/seed/user-data

The OEMDATA `runcmd` entries run in one shell script on the target. The first multiline block defines an `EXIT` trap to unmount `OEMDATA` and create `cloud-init.disabled`, copies `hook.sh` to `/tmp` for execution (with `OEM_MNT` exported so the hook still sees the mounted partition), and removes the copy afterward. The next block checks for `oem-hook-pending-reboot` (when `hook.sh` exits 1) and schedules a delayed reboot.

```
#cloud-config
# Default user for developer flavor (nvidia:nvidia), same as fastos.sh
 ↪without arguments


growpart:
 mode: 'off'


no_ssh_fingerprints: true
resize_rootfs: false


# Allow password auth for console and SSH (required for nvidia login)
ssh_pwauth: true


# Create default user nvidia with password nvidia (developer flavor)
# Use hashed password so login works reliably (chpasswd as backup)
# **** REMOVE USERS AND CHPASSWD SECTIONS IF YOU WANT TO RUN OOBE ****
users:
 - name: nvidia
   groups: [sudo]
   shell: /bin/bash
   lock_passwd: false
   create_home: true
   # SHA-512 hash for password "nvidia" (salt "nv")
   hashed_passwd: $6$nv$JZc5d2h3Tea.dmn0jI6zx/CaCkO4lw3cvREtCME40XZiQdickmO/
 ↪EMrTEKlyVAokumi1qPIXbT89eOiEc85xD.


chpasswd:
```

(continues on next page)

(continued from previous page)

```
 expire: false
 users:
   - name: nvidia
     password: nvidia
     type: text


runcmd:
 - [ sh, -c, 'echo nvidia > /etc/hostname; hostname nvidia 2>/dev/null ||
↪true; if grep -qE "^127\\.0\\.1\\.1[[:space:]]+nvidia" /etc/hosts 2>/dev/
↪null; then :; elif grep -qE "^127\\.0\\.1\\.1" /etc/hosts 2>/dev/null; then
↪sed -i "s/^127\\.0\\.1\\.1.*/127.0.1.1 nvidia/" /etc/hosts; else echo "127.
↪0.1.1 nvidia" >> /etc/hosts; fi; hostnamectl set-hostname nvidia 2>/dev/
↪null || true' ]
 - [ sh, -c, 'userdel ubuntu 2>/dev/null || true' ]
 - [ sh, -c, 'rm -rf /home/ubuntu 2>/dev/null || true' ]
 - [ sh, -c, 'echo "OEM cloud-init seed ran at $(date -Iseconds)" >> /var/log/
↪oem-cloud-init-seed.log' ]
 - [ chmod, '0644', /var/log/oem-cloud-init-seed.log ]
 - [ sh, -c, "mkdir -p /etc/ssh/sshd_config.d && printf '%s\\n'
↪'PasswordAuthentication yes' 'ChallengeResponseAuthentication no' > /etc/
↪ssh/sshd_config.d/99-oem-password-auth.conf && systemctl reload sshd 2>/dev/
↪null || true" ]
 # Mount USB data partition (label OEMDATA) and run hook.sh if present (hook
↪installs debs/firmware)
 - |
   OEM_MNT=/mnt/oemdata
   mkdir -p "$OEM_MNT"
   # One trap for all normal completion: umount + disable cloud-init on next
↪boots.
   # Do not use "exit" here: cloud-init shellifies all runcmd items into one /
↪bin/sh script; exit
   # would skip every later runcmd line (e.g. pending-reboot check) before
↪the EXIT trap runs.
   # Do not use "set -e" in this block: if sync/mkdir/touch after the hook
↪fails, the shell would
   # exit before the post-hook runcmd; EXIT would still run _oemdata_exit
↪(cloud-init.disabled)
   # but the pending-reboot log/reboot would never run.
   _oemdata_exit() {
     echo "OEMDATA exit trap: disabling cloud-init and unmounting OEMDATA"
     umount "$OEM_MNT" 2>/dev/null || true
     rmdir "$OEM_MNT" 2>/dev/null || true
     mkdir -p /etc/cloud
     touch /etc/cloud/cloud-init.disabled
   }
   trap '_oemdata_exit' EXIT
   if mount -L OEMDATA "$OEM_MNT" 2>/dev/null; then
     echo "OEMDATA partition found, checking for hook.sh"
     if [ -f "$OEM_MNT/hook.sh" ]; then
       HOOK_RUN=/tmp/oemdata-hook.sh
       cp -f "$OEM_MNT/hook.sh" "$HOOK_RUN"
```

(continues on next page)

(continued from previous page)

```
      chmod 700 "$HOOK_RUN"
      echo "Running OEM hook from $HOOK_RUN (OEMDATA root still $OEM_MNT
↪until umount)"
      set +e
      export OEM_MNT
      sh "$HOOK_RUN"
      hook_rc=$?
      rm -f "$HOOK_RUN"
      if [ "$hook_rc" -eq 1 ]; then
        echo "mirror setup success"
        sync
        mkdir -p /var/lib/oem
        touch /var/lib/oem/oem-hook-pending-reboot
      fi
    fi
  else
    echo "No OEMDATA partition found, skipping USB OEM hook."
    rmdir "$OEM_MNT" 2>/dev/null || true
  fi
# cloud-init.disabled is created in OEM block EXIT trap above (covers no-
↪OEMDATA path too).
# Reboot if OEM hook requested it (hook exit 1). Runs in same shellified
↪script after OEM block (no "exit" above).
- |
  echo "OEM post-hook: checking pending-reboot marker"
  if [ -f /var/lib/oem/oem-hook-pending-reboot ]; then
    rm -f /var/lib/oem/oem-hook-pending-reboot
    echo "reboot required (OEM mirror apt/fwupd updates); scheduling reboot
↪(+30s so cloud-init can finish modules-final)"
    # EXIT trap may not run before reboot; disable cloud-init and unmount
↪OEMDATA now.
    _oemdata_exit
    sync
    # Immediate reboot races remaining modules-final (e.g. cc_keys_to_
↪console) and can log SystemExit:1.
    # Background sleep + reboot: runcmd exits, cloud-init completes, then
↪reboot (shutdown +m is minute-only).
    ( sleep 30; /sbin/reboot ) </dev/null >/dev/null 2>&1 &
  else
    echo "reboot not required (no OEM pending-reboot marker)"
  fi
```

2.9.2.13 Validation Scenarios and Feedback Questions

The scenarios in **Validation Scenarios** align with Table 1 (Installation and Update Patterns) and the procedures from **Customize the BaseOS Image with repack_baseos.sh** through **Reference: OEM Scripts and Cloud-Init**. Complete the scenario that matches your deployment. For log-based and post-installation verification that complements these flows, see **Verify the Customization and Installation Outcomes**.

How Scenarios Map to This Document

Scenario	Related Table 1 patterns	Where to work
Customized BaseOS ISO	Cloud-Init OEM seed on the repacked ISO (with or without OOBE); `OEMDATA` optional.	Customize the BaseOS Image with `repack_baseos.sh`; Cloud-Init Integration; During and After an ISO-Based Installation; ISO Install: oem-iso-cfg.sh, repack_baseos.sh (excerpt), and OEM Cloud-Init on the ISO.
Air-gapped USB installation	USB-hosted packages and firmware or LOCAL or MIRRORED sources through `OEMDATA` and `hook.sh`.	USB Partitioning and the `OEMDATA` layout; Host a Minimal APT Repository and Firmware Tree; Mirror the Full Ubuntu Ports and LVFS Content on a Server; Client Configuration and `hook.sh`; First Boot: OEMDATA hook.sh and Cloud-Init Seed.
Local repository + DGX Spark Preview updates (OTA2604)	Curated or mirrored APT layout; can extend beyond first-boot automation.	Host a Minimal APT Repository and Firmware Tree; Client Configuration and `hook.sh` (repository layout and client configuration); After Mirror- or USB-Driven Updates. If the DGX Spark Preview (OTA2604) update process is separate from the ISO/`OEMDATA` flow provided here, apply the same repository patterns from those sections and record any additional steps in your runbook.

2.9.2.13.1 Validation Scenarios

1. **Customized BaseOS ISO (repack + Cloud-Init on the image)**: Build a customized BaseOS installation image from the latest release you are targeting and verify that the customization is present on the installed system.

- Refer to *Customize the BaseOS Image with repack_baseos.sh* and place OEM Cloud-Init under `oemdata/cloud-init/` as in *ISO Install: oem-iso-cfg.sh, repack_baseos.sh (excerpt), and OEM Cloud-Init on the ISO*. Adjust `user-data` and `meta-data` per *Cloud-Init Integration* and the OOBE patterns in *Table 1*.

    1. Produce a flashable ISO using your NVIDIA-provided release workflow and `repack_baseos.sh` (or an equivalent process) so the image includes your Cloud-Init seed.
    2. Add or verify Cloud-Init `user-data` and `meta-data` on the ISO (OEM seed paths as in **ISO Install: oem-iso-cfg.sh, repack_baseos.sh (excerpt), and OEM Cloud-Init on the ISO**).
    3. Perform the installation from that ISO onto the target system (for example, by booting the repacked image from USB).
    4. Verify users, packages, and configuration against expectations using *During and After an ISO-Based Installation*.

2. **Air-gapped installation using USB (`OEMDATA`, optional local mirror)**: Perform the installation using installation media and, where applicable, Debian packages and firmware supplied from `OEMDATA` and/or your network mirror.

- Prepare the USB layout per *USB Partitioning and the OEMDATA Layout*. Host packages and firmware using *Host a Minimal APT Repository and Firmware Tree* (minimal tree) or *Mirror the Full Ubuntu Ports and LVFS Content on a Server* (full mirror), then configure the client with `hook.sh` and optional URL files as in *On the DGX Spark Client: hook.sh and OEMDATA Files* and *Client Configuration and hook.sh*. Reference script: *First Boot: OEMDATA hook.sh and Cloud-Init Seed*.
  1. Wipe or prepare the USB drive if your process requires it (reflashing can replace the entire device).
  2. Obtain Debian packages and firmware that your process permits on the disconnected network:
     1. Use the APT or package acquisition tools your OEM or NVIDIA program supplies (for example, an APT downloader or an approved transfer method).
     2. Use the program manifest (or an equivalent bill of materials) to determine which software and firmware to stage and from which approved sources.
        - NVIDIA may supply a baseline manifest to the OEM; the OEM may extend it for firmware or other deltas.
     3. Populate `OEMDATA` (and any server tree) using the directory layout and URLs described in **USB Partitioning and the OEMDATA Layout**, **Host a Minimal APT Repository and Firmware Tree**, and **Mirror the Full Ubuntu Ports and LVFS Content on a Server** as applicable (`debs/`, `firmware/`, `apt-repo.url`, `lvfs-mirror.url`, and so on).
     4. Ensure that Cloud-Init on the ISO or target uses the sample `runcmd` flow when you rely on this information: `hook.sh` stays on `OEMDATA`, but first boot runs a copy under `/tmp` with `OEM_MNT` set to the mount (*First Boot: OEMDATA hook.sh and Cloud-Init Seed*, *oemdata/cloud-init/seed/user-data*).
  3. Perform installation using your customized ISO and attached `OEMDATA` media as described in scenario 1 and **USB Partitioning and the OEMDATA Layout**.
  4. Verify packages, firmware, and customizations using *During and After an ISO-Based Installation* and *After Mirror- or USB-Driven Updates*, as appropriate.
3. **Local repository + DGX Spark Preview application updates (OTA2604)**: Use your standard IT administration tools to host a local APT repository, then distribute DGX Spark Preview software updates (application packages only; exclude firmware, kernel, and driver components unless your policy permits them).

- The minimal and full-mirror layouts in *Host a Minimal APT Repository and Firmware Tree* and *Mirror the Full Ubuntu Ports and LVFS Content on a Server* show HTTP-served package trees; *Client Configuration and hook.sh* describes how to configure a client to use a mirror. Details that are specific to the DGX Spark Preview (OTA2604) delivery mechanism (for example, which `meta-data` or Cloud-Init files to change and how to publish preview packages) may be specified outside this document; use **Host a Minimal APT Repository and Firmware Tree** and **Client Configuration and hook.sh** as the reference model for repository layout and client configuration.
  1. In Cloud-Init `meta-data` (or the configuration channel your DGX Spark Preview / OTA2604 process uses), change repository URLs from public endpoints to your local mirror URLs in accordance with your program requirements.
  2. Deploy the updated `user-data` and `meta-data` (or equivalent) to the device according to your DGX Spark Preview (OTA2604) process.
  3. Publish the DGX Spark Preview packages (or your approved subset) to the local repository.

4. Verify that the device receives the expected package updates (use *After Mirror- or USB-Driven Updates* for verification after updates driven by `hook.sh` or the mirror, where applicable).

By accessing the instructions for custom installation and scaled deployment, the Customer acknowledges that this workflow bypasses individual end-user license prompts. Use of these tools is governed by the NVIDIA Software License Agreement (which includes the NVIDIA CUDA EULA). By proceeding, the Administrator represents they have the authority to bind the Customer to these terms and conditions.

2.10. OS and Component Update Guide

This section provides guidance for updating the operating system, software components, and firmware on your DGX Spark. The DGX Spark runs on NVIDIA DGX OS, which is an Ubuntu-based Linux distribution optimized for AI workloads.

Note

Founders Edition Only: The update information in this guide applies only to the DGX Spark Founders Edition. Devices from other manufacturers might have different update procedures.

2.10.1. Ubuntu Support

The Ubuntu Pro OS license in DGX Spark includes 10-year OS support from Canonical.

2.10.2. Update Methods

We strongly recommend using the DGX Dashboard for all system updates on your |spark|. The dashboard ensures your system stays up to date with the latest NVIDIA-optimized components and configurations. While standard Ubuntu package management methods will work, the dashboard provides the most reliable and tested update path for your DGX system.

2.10.3. Using DGX Dashboard for Updates

The DGX Dashboard is the **primary and recommended** way to perform system updates on your DGX Spark. It provides a centralized interface for:

- Viewing available system updates
- Installing security patches and system updates
- Managing NVIDIA driver updates
- Managing firmware updates
- Monitoring update status and progress

For detailed information about using the DGX Dashboard, including how to access it and perform updates, see *DGX Dashboard*.

Warning

Before performing any system or firmware updates:

- Ensure your system is connected to a stable power source
- Close all running applications and save your work
- Have a recovery plan in place
- Schedule updates during maintenance windows when possible

2.10.4. Manual System Updates

Note: While manual updates using standard Ubuntu package management commands will work, we strongly recommend using the DGX Dashboard for all updates to ensure optimal system performance and compatibility.

For advanced users or when the DGX Dashboard is not available, you can perform system updates manually using the following steps:

1. Open a remote or local terminal on the DGX Spark device.
2. Run the following commands:

```
sudo apt update
sudo apt dist-upgrade
sudo fwupdmgr refresh
sudo fwupdmgr upgrade
sudo reboot
```

These commands will:

- Update the package lists and upgrade all installed packages (including OS components and drivers)
- Refresh the firmware metadata and upgrade all firmware components
- Reboot the system to apply updates

2.10.5. Update Best Practices

- **Use the DGX Dashboard**: Always prefer the DGX Dashboard for system updates to ensure compatibility and optimal performance
- **Regular updates**: Check for updates regularly, especially security patches
- **Backup before major updates**: Always backup critical data before major system changes
- **Stable power**: Ensure your system has a stable power supply during updates
- **Maintenance windows**: Schedule updates during planned maintenance windows when possible

2.11. System Recovery

This section provides information about system recovery procedures for your DGX Spark.

> **Note**
>
> **Founders Edition Only**: This recovery information applies only to the DGX Spark Founders Edition. The Founders Edition OS is not distributed as an ISO; recovery media is provided as a tar.gz archive. OEM variant images are only available from the respective OEMs. Devices from other manufacturers might have different recovery procedures and recovery media.

2.11.1. System Recovery Overview

The system recovery process for the DGX Spark allows you to restore the operating system and firmware to their original factory state. This process is useful when dealing with system corruption, fatal configuration errors, or other issues that prevent normal system operation.

2.11.1.1 Recovery Requirements

Before beginning the recovery process, ensure you have the following:

- A USB flash drive with 16GB or larger capacity
- A keyboard and display connected to your DGX Spark
- Access to download the recovery media from developer.nvidia.com

2.11.1.2 Recovery Process Steps

The recovery process involves downloading recovery media, creating a bootable USB drive, and following a series of UEFI configuration steps. Follow these steps carefully to ensure successful system recovery.

1. **Download Recovery Media**:
   - Download the recovery media archive file (tar.gz format) from DGX Spark System Recovery Image
   - Save the file to your local system
2. **Create Recovery USB Drive**:
   - Download and unzip the recovery media archive file
   - Insert the USB drive into your system
   - Within the extracted files, use one of the following commands to create the recovery drive:

   > **Note**
   >
   > **Creating the recovery drive requires administrator privileges.** On Windows systems, this is typically done by running the command as an administrator (i.e. from an elevated PowerShell or Command Prompt). If supported by the host system, you will be prompted by the system to run the command with elevated privileges.

Windows:

```
CreateUSBKey.cmd
```

Linux:

```
CreateUSBKey.sh
```

MacOS:

```
CreateUSBKeyMacOS.sh
```

> **Warning**
>
> This process will erase all data on your USB drive. Ensure you have backed up any important data before proceeding.

3. **Boot from Recovery USB Drive**:
   - Disconnect any external storage devices from the DGX Spark
   - Connect the USB drive to one of the USB ports on your DGX Spark
   - If your DGX Spark is powered off, boot the device into UEFI settings by holding down the `Esc` or `Del` key immediately after powering on the device

> **Important**
>
> Make sure to use a keyboard plugged directly into a USB port on the DGX Spark device. Some Bluetooth keyboards, even those with USB connections, may not work during this process. If your keyboard isn't recognized, use a standard USB keyboard.

4. **Restore UEFI Defaults**:
   - Tap the `Right Arrow` key to select the "Save & Exit" page within the UEFI settings
   - Tap `Down Arrow` to select "Restore Defaults" and select "Yes" in response to "Load Optimized Defaults"
   - Select "Save Changes and Reset" - the device will reboot
   - While the device is rebooting, hold down the `Esc` or `Del` key to re-enter UEFI settings a second time
5. **Enable Secure Boot**:
   - Use the `Right Arrow` key to select "Security"
   - Confirm that "Secure Boot" is set to "Enabled"
   - Select "Restore Factory Keys"
   - Use the arrow keys to select the "Save and Exit" BIOS screen, and select "Save Changes and Reset"
6. **Boot from Recovery Media**:
   - While the device is rebooting, hold down the `Esc` or `Del` key to re-enter UEFI settings a third time

- Tap the `Right Arrow` key to select the “Save & Exit” page in UEFI settings
- Tap `Down Arrow` to move to the “Boot Override” section
- Select the USB drive and tap `Enter`
- The device will reboot using the USB drive - follow the on-screen steps to update your firmware and re-install the OS on the device’s SSD

7. **Restore System from Recovery Environment**
   - On the welcome screen, press `Enter` to continue. To cancel the process, press `Esc`.

   - On the warning screen, select [EXIT] to restart without proceeding, or select [START RECOVERY] to begin reflashing the SSD. Be aware that this will completely erase the internal SSD on the DGX Spark.

- Monitor the Recovery Progress screen for detailed output from the recovery process and the progress of the SSD reflash.

- When the process is complete, review the progress screen and follow the prompt to continue. Scroll or page through the recovery output if needed, for example during customer support.

- On the final screen, confirm that the DGX Spark has been reset to factory state and press `Enter` to restart the device.

2.12. Get the Right Support for Your DGX Spark

Choose the options that match your needs.

2.12.1. NVIDIA DGX Spark Hardware Support

- Open a new ticket. Log In
- Chat online with our support agents. Chat Now
- Log in to view your existing tickets. View Tickets
- Join the NVIDIA DGX Spark Forums to ask questions, share experiences, and get help from other users and NVIDIA engineers.

2.12.2. NVIDIA AI Enterprise—DGX Spark Support

Enterprise technical support for the NVIDIA AI Enterprise—DGX Spark software stack on DGX Spark requires an entitlement for **NVIDIA AI Enterprise—DGX Spark**. Other NVIDIA AI Enterprise entitlements do not automatically include this product or its support. Confirm that your NVIDIA Entitlement Certificate lists **NVIDIA AI Enterprise—DGX Spark** before opening enterprise cases for DGX Spark software.

- Open a new ticket. Log In
- Chat online with our support agents. Chat Now
- Log in to view your existing tickets. View Tickets
- For registration, downloads, and portal access, see the NVIDIA AI Enterprise—DGX Spark Quick Start Guide: *NVIDIA AI Enterprise—DGX Spark Quick Start Guide*.

2.12.3. Enterprise Manageability

For fleet lifecycle integration, platform integration patterns, and custom installation with cloud-init, refer to *Enterprise Manageability*.

2.12.4. Security and Vulnerability Response

For NVIDIA's approach to receiving, assessing, and disclosing product security issues, see the NVIDIA Product Security Incident Response Team (PSIRT) policies.

2.12.5. Field Diagnostic Software

NVIDIA Field Diagnostic is a software program used to test the DGX Spark system and detect hardware failures, and is intended for health checks of your DGX Spark setup and a pre-check for RMA qualification of the overall system.

For complete instructions, refer to the `Field Diagnostics User Guide`.

2.12.5.1 Removing a Previous Version

Remove the previous version of the field diagnostic software before installing a new one with the following commands:

```
sudo dpkg -P dgx-spark-fieldiag
sudo rm -rf /opt/nvidia/dgx-spark-fieldiag
sudo apt autoremove dgx-spark-fieldiag
```

2.12.5.2 Installing the Field Diagnostic Software

The Field Diagnostic software package is named `dgx-spark-fieldiag_<version>-1_arm64.deb`. To install the package using the NVIDIA CUDA APT Repository, follow these steps:

1. Add the NVIDIA CUDA repository key:

```
sudo mkdir -p /usr/share/keyrings
curl -fsSL https://developer.download.nvidia.com/compute/cuda/repos/
↪ubuntu2404/sbsa/cuda-archive-keyring.gpg | sudo tee /usr/share/keyrings/
↪cuda-archive-keyring.gpg > /dev/null
```

2. Add the CUDA APT repository and install:

```
echo "deb [signed-by=/usr/share/keyrings/cuda-archive-keyring.gpg] https:/
↪/developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/sbsa /" |
↪sudo tee /etc/apt/sources.list.d/cuda-sbsa-ubuntu2404.list
sudo apt-get update
sudo apt-get install dgx-spark-fieldiag
```

3. Verify the installation:

```
dpkg -l | grep dgx-spark-fieldiag
```

When you install the `dgx-spark-fieldiag` package, `stress-ng`, `fio`, and `memtester` are installed automatically as package dependencies. These tools support the core CPU, SSD, and memory stress tests. This automatic installation does not include the ConnectX-7 (CX7Stress) test tools.

2.12.5.3 ConnectX-7 Test Prerequisites

The CX7Stress test requires additional tools that are not installed with the `dgx-spark-fieldiag` package. Before you run field diagnostics, install DOCA OFED and the NVIDIA Firmware Tools (MFT) kernel modules on your system. For installation and removal instructions, refer to the Field Diagnostics User Guide.

2.12.5.4 Running Field Diagnostics

After installing the package, the field diagnostic software is located at `/opt/nvidia/dgx-spark-fieldiag`.

2.12.5.4.1 Before Running

Disable Secure Boot before running field diagnostics:

1. Check the current Secure Boot state:

```
sudo mokutil --sb-state
```

2. Reboot and enter UEFI setup (press **Delete** during boot, or run `sudo systemctl reboot --firmware-setup`).

3. Go to **Security** → **Secure Boot** → **Disable Secure Boot**.
4. Save the changes and reboot the system.

2.12.5.4.2 Running the Diagnostic

To execute field diagnostics, use root access:

1. Run:

```
sudo init 3
```

2. When the system switches to TTY console mode, log in at the TTY console.
3. Run:

```
cd /opt/nvidia/dgx-spark-fieldiag
sudo ./partnerdiag --field
```

The diagnostic takes approximately 30 minutes. A PASS/FAIL banner appears when complete. You can also run the diagnostic over SSH using the same commands.

> **Note**
>
> If you interrupt the diagnostic (for example, with Ctrl+C), power cycle the system before running the tests again.

2.12.5.4.3 After Running

Re-enable Secure Boot after the diagnostic completes:

1. Run `sudo systemctl reboot --firmware-setup`.
2. Go to **Security** → **Secure Boot** → **Enable Secure Boot**.
3. Save the changes and reboot the system.

For detailed instructions, including the Spec JSON file and log retrieval, refer to the Field Diagnostics User Guide.

2.12.5.4.4 Verifying Tool Installation

To verify that the core diagnostic tools are installed, run the following commands. Each command should display the path to the tool binary.

```
which fio
which memtester
which stress-ng
```

To verify that the ConnectX-7 (CX7Stress) tools are installed, run the following commands:

```
which ofed_info
which opensm
which ibstat
which mlxlink
```

If a command returns no output, install the missing packages before you run field diagnostics. Refer to the Field Diagnostics User Guide for ConnectX-7 prerequisite installation and removal instructions.

2.13. Third-Party License Notices

This NVIDIA product contains third party software that is being made available to you under their respective open source software licenses. Some of those licenses also require specific legal information to be included in the product. This section provides such information.

2.13.1. LIBICONV Library

NVTelemetry is using the `libiconv` library, which is licensed under the GNU Lesser General Public License (LGPL). `libiconv` source code can be obtained from the following link: https://ftp.gnu.org/pub/gnu/libiconv/libiconv-1.15.tar.gz.

The `libiconv` library is provided under the following terms:

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
```

(continues on next page)

(continued from previous page)

```
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  1. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
```

(continues on next page)

(continued from previous page)

```
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  2. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
```

(continues on next page)

(continued from previous page)

```
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  3. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  4. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
```

(continues on next page)

(continued from previous page)

```
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  5. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  6. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    1.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
```

(continues on next page)

(continued from previous page)

```
parts of the aggregate.

  7. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
```

(continues on next page)

(continued from previous page)

```
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  8. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
```

(continues on next page)

(continued from previous page)

```
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
```

(continues on next page)

(continued from previous page)

```
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  9. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  10. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  11. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
```

(continues on next page)

(continued from previous page)

```
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  12. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
```

(continues on next page)

(continued from previous page)

```
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  13. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  14. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
```

(continues on next page)

(continued from previous page)

```
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  15. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  16. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  17. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
```

(continues on next page)

(continued from previous page)

```
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  18. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

(continues on next page)

(continued from previous page)

```
The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>
```

2.14. Notices

2.14.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer,

and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

2.14.2. Trademarks

NVIDIA, the NVIDIA logo, DGX, DGX-1, DGX-2, DGX A100, DGX H100/H200, DGX B200, DGX Station, DGX Station A100, and DGX Spark are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright