DGX Spark Custom Installation with Cloud-Init

Table of Contents

IMPORTANT NOTICE - PLEASE READ AND AGREE BEFORE USING NVIDIA SOFTWARE

This NVIDIA Software License Agreement is entered into between the entity you represent, or, if you are an Administrator, the entity on whose behalf you are deploying the Software, or, if you do not designate an entity, you individually ("Customer"), and NVIDIA Corporation ("NVIDIA") and the Product Specific Terms (if applicable) govern the use of the Software Offerings. This NVIDIA Software License Agreement consists of the terms and conditions below and all documents attached to or referenced in this NVIDIA Software License Agreement (together, the "Agreement"). Key terms are defined in Section 17. Customer and NVIDIA are each a "party" and collectively the "parties."

By registering to use, using, or by an Administrator deploying or configuring the Software Offerings for use by others, Customer is affirming that it has read the Agreement and agrees to its terms. If Customer (or the individual acting on Customer's behalf) does not have the required authority to enter into the Agreement or if it does not accept all Agreement terms and conditions, Customer should not register to use, deploy, or use the Software Offerings.

You can find a short summary explaining how you can use certain Software Offerings here.

1. LICENSE

1.1 Grant. Subject to the terms of the Agreement, Customer's Order Form and license parameters of an Enterprise Product Part Number, and payment of applicable fees, NVIDIA grants Customer a limited, worldwide, non-exclusive, non-transferable, non-sublicensable (except as expressly granted in the Agreement) license during the Term to install (including via centralized or automated deployment by an Administrator), use and reproduce the Software solely for Customer's business operations by its authorized users.

1.2 Limited Grants. The grants in Section 1.1 above are further limited as follows:

1.2.1 "not for resale" or "NFR" (typically a license to a distributor or reseller) licenses to Enterprise Products are licensed for limited use under Section 1.1 solely for internal evaluation or to demonstrate the Enterprise Product to others, but may not **be** use**d**, distribute**d** or deploy**ed** in production.

1.2.2 trial licenses to Enterprise Products are licensed for limited use under Section 1.1 solely for the trial period.

1.3 Authorized Users.

1.3.1 Software Users. Under the Agreement, Customer Personnel may access and use the Software Offerings and Customer End Users may access and use Customer Products.

1.3.2 Customer Personnel. Customer is responsible for the compliance with the terms of the Agreement by Customer Personnel. Any act or omission that if committed by Customer

would constitute a breach of the Agreement will be deemed to constitute a breach of the Agreement if committed by Customer Personnel.

1.3.3 Customer End Users. Customer must ensure that the use of Software Offerings by Customer End Users is governed by enforceable agreements or corporate policies that are at least as protective as the terms of the Agreement (including, but not limited to, terms relating to the grants, restrictions and ownership of intellectual property). For Software Offerings deployed via automated or centralized administrative tools, Customer remains fully responsible for ensuring End User compliance with these terms. NVIDIA reserves the right to, from time to time, update the Agreement (an updated version being referred to as "Updated Agreement"). Upon notification of an Updated Agreement, which may be in the form of electronic acceptance by an Administrator on behalf of Customer, Customer agrees to review the Updated Agreement and ensure ongoing compliance across its deployed fleet on a going forward basis. If NVIDIA or the Customer have reason to know or determine in good faith that a Customer End User is not in compliance with the Agreement, NVIDIA and Customer will cooperate to investigate and resolve such non-compliance. Customer will cooperate to enforce use rights with Customer End Users.

1.4 License Types. The terms in this Section 1.4 apply only to Enterprise Products, not to Community Products. The license types below describe features that may be part of an Enterprise Product; and not all license types may be available for each Enterprise Product. Enterprise Products are licensed under the following license types: Subscription per CPU Socket, Subscription per GPU, Subscription per Node, Subscription per CCU, Subscription per HCA, Usage Based Subscription, or Perpetual license. Once Customer transitions from a Community Product to an Enterprise Product version under a paid Subscription or Perpetual license, any Customer use of such Enterprise Product must be under a paid Subscription or Perpetual license. Customer's order, license key or the product description will indicate the parameters of Customer's license. For Software Offerings deployed via automated or centralized tools, the Administrator is responsible for ensuring that the deployment parameters (e.g., number of GPUs, Nodes, or Sockets) do not exceed the authorized license types and quantities specified in the applicable Order Form.

1.5 Replacement Products. NVIDIA may, from time to time at its discretion, give Customer the option to replace a certain Enterprise Product, subject to payment of applicable fees. In such cases, Customer must discontinue use of the replaced Enterprise Product timely upon the start of use of the replacement Enterprise Product. If requested in writing by NVIDIA, Customer will provide a written certificate signed by an authorized officer or an authorized Administrator affirming Customer's compliance with the replacement terms, including the decommissioning of replaced Software across Customer's fleet.

1.6 Promotional Offerings. NVIDIA may, from time to time, offer free or discounted pricing programs covering certain uses of Software Offerings, for example having different license parameters or fees for evaluation or academic use. NVIDIA may stop accepting new sign-ups or provisioning or discontinue a promotional offering at any time. Standard charges will apply after a promotional offering ends or if Customer exceeds the promotional offering use terms. Customer must comply with any additional terms, restrictions, or

limitations (e.g., limitations on the total amount of usage) for a promotional offering as described in the corresponding offer terms or as presented within the administrative configuration tools.

2. AI ETHICS.

Use of the Software Offerings under the Agreement must be consistent with NVIDIA's Trustworthy AI terms at https://www.nvidia.com/en-us/agreements/trustworthy-ai/terms/. Customer acknowledges that these terms apply to the output and use-cases generated by authorized users on the Software Offerings. Where individual user acceptance is bypassed via automated configuration, Customer assumes full responsibility for ensuring its personnel's use of the Software Offerings adheres to these Trustworthy AI standards.

3. PRE-RELEASE VERSIONS AND FEATURES.

NVIDIA will clearly designate Software versions that are in Pre-Release. Pre-Release versions may not be fully functional, may contain errors or design flaws, and may have reduced or different security, privacy, accessibility, availability, and reliability standards relative to commercially provided NVIDIA software, materials and services. Use of a Pre-Release version may result in unexpected results, such as loss of use or loss of content. Customer may use a Pre-Release version at Customer's risk, understanding that such versions are not intended for use in business-critical systems and Customer may stop at Customer's convenience. Where Pre-Release versions are deployed or enabled via centralized administrative tools, the Administrator acknowledges and accepts these risks on behalf of the Customer and its authorized users. NVIDIA may choose to abandon development and terminate the availability of a Pre-Release version at any time without liability. Pre-release versions are provided "AS-IS," "WITH ALL FAULTS," and "AS-AVAILABLE," and are excluded from Enterprise Support.

4. SERVICES.

4.1 General Service Terms. Unless otherwise indicated by NVIDIA in this Agreement or an Order Form, Software Subscriptions include Enterprise Support, Maintenance and Updates. Enterprise Support, Maintenance and Updates may be sold separately for Perpetual licenses. After the expiration of Services, Customer retains the right to use a Perpetual license at the last-supported level subject to the terms of the Agreement. For deployments managed via centralized administrative tools, Customer is responsible for ensuring that only devices with active Service entitlements receive Updates and Enterprise Support. Unless NVIDIA accepts Customer's request (which may be submitted by an authorized Administrator) for Enterprise Support, Maintenance and Updates in an Order Form, NVIDIA is under no obligation to provide any Service. Unless Software is provided with their separate governing terms, they are deemed part of the applicable Software Offering and governed by the Agreement.

4.2 Use of Maintenance and Updates. NVIDIA encourages Customers to use Maintenance and Updates available to them. Customer's choice **(or the Administrator's choice on Customer's behalf)** not to deploy Maintenance or Update as they become available may result in issues with operability, compatibility and interoperability and result in the Software in use being non-conforming to later Software documentation.

4.3 Maintaining the Authorized Number of Licenses. Customer's use of Maintenance or Update does not change the number of authorized licenses. Customer agrees to promptly discontinue use of prior versions across its entire deployed fleet as necessary to maintain the authorized number of licenses.

4.4 Work Out of Scope. Any enhancements or additions to Software beyond Maintenance or Updates are outside of the scope of this Agreement.

4.5 Services for NFR, Trial or Developer Program Licenses. NVIDIA is not obligated to provide Services for any free NFR, trial or developer program items, even if identified as Subscription versions, and any Services are provided at NVIDIA's discretion. The use of centralized deployment tools to distribute NFR or Trial versions does not create an entitlement to Enterprise Support.

4.6 Services to Customer End Users. As between NVIDIA and Customer, Customer is responsible for supporting Customer Products and providing first-line support to Customer Personnel and Customer End Users for the Software Offerings.

5. USERNAME AND PASSWORD.

Customer is responsible for securely maintaining log-in information (including, but not limited to, administrative credentials, API keys, and service tokens) for Customer Authorized Users' use, and for all activities under Customer's account(s). Where Software Offerings are deployed via centralized administrative tools, Customer is responsible for ensuring that credentials used for automated provisioning are managed in a secure manner and rotated as necessary. Customer agrees to notify NVIDIA via an authorized Administrator at enterprisesupport@nvidia.com immediately of any known or suspected security incidents or unauthorized use of Customer's account(s) or the Software Offering.

6. COMPONENTS UNDER OTHER LICENSES.

The Software may include or be distributed with Separate Components. The Separate Components are subject to the applicable OSS Licenses or other license terms, including any proprietary notices, disclaimers, requirements and extended use rights; except that the Agreement will prevail regarding the use of third-party open source software, unless NVIDIA components are provided under an OSS License or a third-party OSS License requires its license terms to prevail. Where Software Offerings are configured or deployed at scale by an Administrator, Customer is responsible for ensuring that all proprietary notices and requirements for Separate Components are maintained and that authorized users are notified of any applicable third-party license restrictions.

7. PAYMENT TERMS AND TAXES.

7.1 Ordering. Customer may be able to purchase a Subscription or Perpetual license directly from NVIDIA, or via a reseller (in some cases a cloud marketplace reseller), as available. Each Order Form will be effective when entered into by the Customer and NVIDIA. For deployments managed via centralized administrative tools, the activation or provisioning of the Software Offering by an Administrator constitutes an order under an applicable Order Form. Each order placed by Customer through an Order Form or administrative provisioning tool is a separate transaction of the parties under the Agreement.

7.2 Fees. When purchasing directly from NVIDIA, the following terms apply: Fees for the Subscriptions or Perpetual licenses are set forth in the associated Order Form and are payable pursuant to the terms of such Order Form. Unless otherwise expressly indicated in an Order Form, fees will be invoiced upon Customer's purchase (including any purchase triggered via administrative provisioning tools), are payable upon invoice and are expressed in U.S. Dollars. Each Order Form placed or provisioning action taken by an Administrator is non-cancelable and fees received are non-refundable. All amounts not paid when due will accrue interest (without the requirement of a notice) at the lower of 1.5% per month or the highest rate permissible by law until the unpaid amounts are paid in full. Fees do not include any taxes, duties or similar charges.

7.3 Taxes. If NVIDIA is required to pay sales, use, property, value-added or other taxes based on the payments provided under the Agreement and if NVIDIA is required to collect and remit such taxes, then such taxes will be billed to and paid by Customer or Customer's reseller, unless NVIDIA receives a valid exemption or resale certificate. If Customer is not billed the applicable tax under the Order Form, then it is Customer's responsibility to properly remit the tax directly to the applicable tax jurisdiction. Customer's responsibility for tax remittance extends to all Software Offerings provisioned via centralized administrative tools across any and all applicable jurisdictions. Further, Customer acknowledges that the payments to NVIDIA under the Agreement will be made in full without reduction for withholding taxes, if applicable. This section will not apply to taxes based on NVIDIA's net income or payroll taxes.

7.4 Overdue Payment. If any payment is overdue from Customer or a reseller, NVIDIA reserves the right to suspend the Subscriptions, Perpetual licenses and Services, in addition to any other remedies it may have, until the payment delinquency is corrected. Customer acknowledges that such suspension may be applied globally to all Software Offerings provisioned via centralized administrative tools.

7.5 Price Changes. Any price change will only apply to purchases after the price change. For deployments managed via centralized tools, the "purchase" date is the date an Administrator provisions or activates the Software Offering.

8. LIMITATIONS.

The following limitations and restrictions apply to the Software, Derivative Samples and Derivative Models, and Customer is responsible for the consequences of non-conformance with these limitations:

8.1 Customer will use the Software exclusively for authorized purposes, consistent with the Agreement's terms and all applicable laws, regulations and the rights of others. Where Software Offerings are provisioned via centralized administrative tools, Customer assumes full responsibility for configuring the Software in a manner that complies with these limitations across its entire fleet.

8.2 Product-Specific Terms may indicate that NVIDIA proprietary Software, Derivative Samples and Derivative Models are licensed only to run on NVIDIA Platforms. When deploying via centralized administrative tools, Customer must configure such tools to ensure Software is only provisioned onto authorized NVIDIA Platforms.

8.3 Customer may not combine the use of paid and unpaid Software, Derivative Samples and Derivative Models in a way that avoids incurring fees or exceeding use limits or quotas. Customer is responsible for ensuring that automated deployment configurations do not result in such unauthorized combination across its fleet.

8.4 Customer may not reverse engineer, decompile, disassemble Software components provided in binary form, nor attempt in any other manner to obtain source code of such Software components. Customer is responsible for ensuring that any automated or centralized deployment processes do not modify, bypass, or facilitate unauthorized access to the binary form or source code of the Software components.

8.5 Except as expressly granted in the Agreement or Product-Specific Terms, Customer may not copy, sell, resell, rent, sublicense, transfer, assign, timeshare, distribute, modify, or create derivative works of any portion of the Software, including, without limitation, in any publicly accessible software repositories. The use of centralized administrative tools for internal provisioning to Customer's authorized users is permitted, provided that Customer remains responsible for preventing any unauthorized distribution or modification of the Software components.

8.6 Customer may not indicate that a product or service developed with the Software is sponsored or endorsed by NVIDIA unless expressly authorized in writing by NVIDIA. Where Software Offerings are provisioned via centralized administrative tools, Customer is responsible for ensuring that any automated configurations or interfaces do not imply such NVIDIA sponsorship or endorsement.

8.7 Customer may not bypass, disable, or circumvent any technical limitation, encryption, security, digital rights management or authentication mechanism contained in the Software. Customer is responsible for ensuring that centralized deployment configurations do not inadvertently bypass or disable such security or authentication mechanisms.

8.8 Customer may not use the Software components governed by the Agreement in any manner that would cause components to become subject to an OSS License or other shareware license. Customer is responsible for ensuring that any automated deployment configurations or scripts do not integrate the Software in a manner that triggers such third-party licensing obligations.

8.9 Customer may not distribute or disclose to third parties the results of benchmarking, competitive analysis, regression or performance data relating to the Software without the prior written permission from NVIDIA, except as described at https://docs.nvidia.com/nvidia-containers-benchmarking.pdf. Where Software is provisioned via centralized administrative tools, Customer is responsible for ensuring that authorized users are notified of and comply with these benchmarking disclosure restrictions.

8.10 Customer may not replace any NVIDIA software components in the Software that are governed by the Agreement with other software that implements NVIDIA application programming interfaces (APIs). Customer is responsible for ensuring that automated deployment configurations or third-party management tools do not substitute or replace such NVIDIA software components.

8.11 Customer may not reverse engineer, decompile or disassemble any portion of the output generated using an NVIDIA proprietary software development kit (e.g., NVIDIA CUDA toolkit), including their development tools and compilers. Where such outputs are generated or managed via centralized administrative tools or automated workflows, Customer is responsible for ensuring that all authorized users and automated processes adhere to these restrictions.

8.12 Customer may not use the Software or NVIDIA Confidential Information for the purpose of (i) developing competing products or technologies or assisting a third party in such activities, or (ii)identifying or supporting an assertion or potential assertion of any intellectual property rights against NVIDIA (including patent, copyright, or trade secret). Where Software is provisioned via centralized administrative tools, Customer is responsible for monitoring and ensuring that all authorized users and automated workflows comply with these non-competition and non-assertion obligations.

8.13 Customer acknowledges that the Software as delivered under the Agreement is not tested or certified by NVIDIA for use in any Critical Application. Beyond NVIDIA delivering the Software in accordance with the Agreement, NVIDIA will not be liable to Customer or any third party, in whole or in part, for any claims or damages arising from such uses. Customer is solely responsible for ensuring that systems and applications developed or deployed with the Software include sufficient safety and redundancy features and comply with all applicable legal and regulatory standards and requirements. Where Software is provisioned at scale via administrative tools, the Administrator is responsible for verifying that target systems do not constitute Critical Applications and that all safety and redundancy requirements are met globally across the fleet.

8.14 Customer may not use the Software to infringe any third party's Intellectual Property Rights. Where Software is provisioned via centralized administrative tools, Customer is responsible for ensuring that automated configurations and subsequent user activities within the provisioned environment do not facilitate such infringement.

8.15 Customer agrees to defend, indemnify and hold harmless NVIDIA and its Affiliates, and their respective employees, contractors, agents, officers and directors, from and against any and all third-party claims, damages, obligations, losses, liabilities, costs or debt, fines, restitutions and expenses (including but not limited to attorney's fees and costs incident to establishing the right of indemnification) arising out of or related to (i) products or services developed or deployed with or that use the Software (including results or data generated from such use), or claims that such products or services violate laws, or infringe, violate, or misappropriate any third party right, provided that Customer is not obligated to provide intellectual property indemnification to NVIDIA for Software that Customer has not modified; or (ii) a violation of the terms and conditions of the Agreement. Where Software is provisioned via centralized administrative tools, Customer's indemnification obligations extend to all activities and claims resulting from such automated deployment across its entire fleet.

8.16 The following proprietary Software components are licensed only to run on systems with NVIDIA Platforms: NVIDIA software development kits (such as CUDA toolkit, TensorRT, cuDNN) and NVIDIA drivers. When deploying via centralized administrative tools, Customer must configure such tools to ensure these components are only provisioned onto authorized NVIDIA Platforms.

8.17 Customer may not use the Software or any of its components for the purpose of emotion recognition. Any technology included in the Software may only be used as fully integrated in the Software and consistent with all applicable documentation.

(v. April 19, 2026)

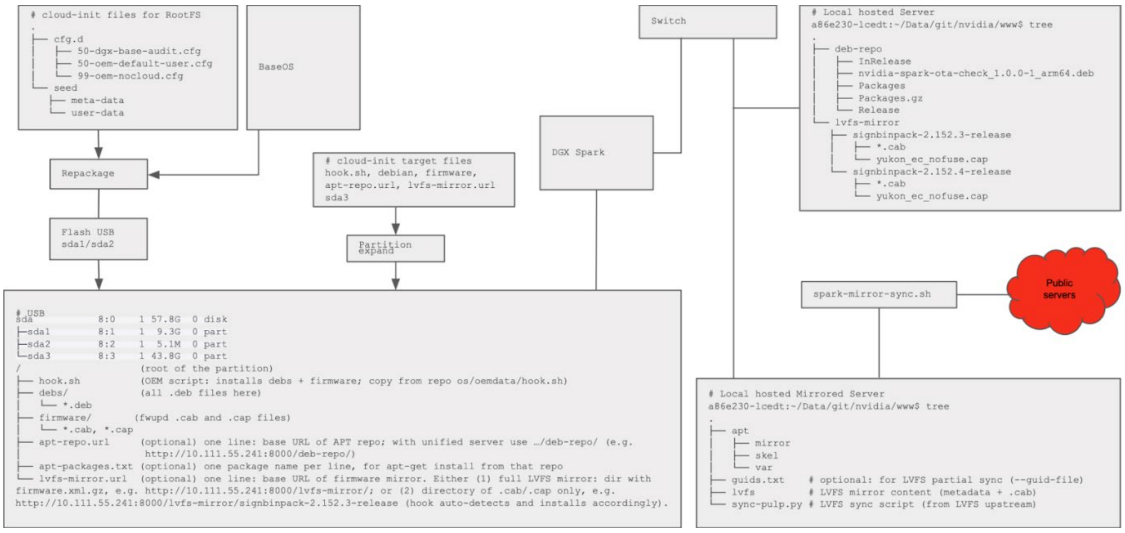
1. Introduction

This document 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.

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 in this guide. 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`, `oem-iso-cfg.sh`, a partial `repack_baseos.sh` excerpt, and example OEM Cloud-Init files appear in 12. Reference: OEM Scripts and Cloud-Init. After you complete those procedures, use 13. Validation Scenarios and Feedback Questions for structured validation and feedback prompts.



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

Figure 1. Spark custom installation overview.

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.

Table 1. 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 a 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 a second USB drive.	Not used	Same Cloud-Init and repack references as above. Adjust `user-data` and related OEM configuration to match your policy. Once there is no username and password settings section in `user-data`, OOBE will be enabled.
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	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

	small local APT tree and optional firmware directory or LVFS-style layout, not a full Ubuntu archive mirror.		
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.

3. Example Constants

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

Table 2. 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`)

Table 3. 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`

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>
```

Table 4. 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.

5. USB Partitioning and the OEMDATA Layout

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.

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 d
isk
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/
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
```

Table 5. 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://10.111.55.241:8000/lvfs-mirror/`; or (2) directory of `.cab`/`.cap` only, for example `http://10.111.55.241:8000/lvfs-mirror/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 p
ath 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 c
lient.
# Before first boot, open the URL in a browser to confirm it is reachable. Th
e client uses this address when it refreshes the index from the OEM local sou
rce.
http://10.111.55.241:8000

# apt-packages.txt (optional): one package name per line. hook.sh runs apt-ge
t install for these from apt-repo.url.
# If you omit this file but apt-repo.url exists, hook.sh still adds the OEM s
ource 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.

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 7. 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 9. 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"
```

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
-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 $1}')" "$(st
at -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`).

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 example, `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 12.1.1 `oemdata/hook.sh` 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.

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.

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
```

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.

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).

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)
```

Table 6. Client URL Patterns (Must Match Layout)

Service	URL Pattern
`apt`	`http://SERVER_IP:8080/apt/mirror/ports.ubuntu.com/ubuntu-ports/`
`fwupd`	`MetadataURI=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.

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).

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.

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 || tru
e
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 i
f 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.

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
```

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.

8. Client Configuration and hook.sh

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`.

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"
```

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.

Table 7. 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
```

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`.

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.

10. Verify the Customization and Installation Outcomes

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`.

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.

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.

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.

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.

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 firm
ware/).
# 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-fwup
d-unified.md
# Exit: 0 = success, no apt/fwupd updates in final pass; 1 = success and at l
east 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-enable
d.
# Override: MIRROR_SERVER_IP, MIRROR_SERVER_PORT, LVFS_METADATA_NAME, LVFS_WE
B_SUBDIR
# OEM_MNT is the OEMDATA partition root (debs/, firmware/, urls). Normally th
e directory
# containing this script; cloud-init may copy this file to /tmp and set OEM_M
NT explicitly.
# This tree does not use functions.sh. If you extend the USB copy, source hel
pers only as
```

```
#   . "$OEM_MNT/your-helper.sh"
# never . "$(dirname "$0")/..." or files vanish when $0 is under /tmp.


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)
# ------------------------------------------------------------------------
--
_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 || t
rue
}


_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.deskt
op 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 || t
rue
 touch /home/nvidia/.config/gnome-initial-setup-done 2>/dev/null || true
 chown nvidia:nvidia /home/nvidia/.config/gnome-initial-setup-done 2>/dev/nul
l || true
}


_hook_oobe_hotspot_teardown_if_ethernet() {
 # Run as a child (not ".") so dgx-oobe sees $0 under /opt/nvidia/dgx-oobe (f
unctions.sh path).
 # Use bash: functions.sh uses bash syntax; /bin/sh (dash) errors with "(" un
expected.
 _hs=/opt/nvidia/dgx-oobe/oobe-hotspot-shutdown.sh
 if [ -f "$_hs" ]; then
   command -v bash >/dev/null 2>&1 && bash "$_hs" || true
 fi
}


_hook_oobe_disable_systemd_units() {
 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_telemetry_if_consent() {
 echo "[oem hook] Enabling telemetry"
 install -d -m 0755 /opt/nvidia/dgx-telemetry 2>/dev/null || true
 touch /opt/nvidia/dgx-telemetry/eula_accepted \
   /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-sol 2>/dev/null; then
   echo "[oem hook] warning: systemctl enable --now nvidia-dgx-sol failed (ch
eck status; unit may stay disabled)" >&2 || true
   systemctl start nvidia-dgx-sol 2>/dev/null || true
 fi
 if ! systemctl enable --now nvidia-dgx-telemetry 2>/dev/null; then
   echo "[oem hook] warning: systemctl enable --now nvidia-dgx-telemetry fail
ed (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 e
very script exit.
# (EXIT runs after normal completion, exit 1, or exit 255 so these steps stil
l 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_telemetry_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
     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 UR
L
# ------------------------------------------------------------------------
--
_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
   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 "$OEM_MNT/apt-packages.txt" ]; then
   echo "apt-packages.txt not found; apt upgrade using OEM local repo only (s
ingle-source apt)"
   apt upgrade -y \
     -o Dir::Etc::sourcelist="/etc/apt/sources.list.d/oem-local.list" \
     -o APT::Architecture="$(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="${line%_*}"; line="${li
ne%_*}"; 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 "$base/Packages.gz" | zcat 2>/dev/null | awk -v pkg="$p
kg" '
/^Package: /{name=$2}
/^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" > "$tmp_deb" 2>/dev/null && [ -s "$t
mp_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 \( -name '*.cab' -o -name '*.ca
p' \) | 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
   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_UPDATE
D, 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 u
pgrade || exit 255
   HOOK_APT_UPDATED=1
 fi
}


_hook_fwupdmgr_refresh_and_count() {
 echo "fwupdmgr refresh --force"
 fwupdmgr refresh --force


 HOOK_FW_APPLICABLE=""
 if command -v jq >/dev/null 2>&1; then
   _hook_jq_fw_count='(if type == "object" and (.Devices | type) == "array" t
hen .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_APPLICAB
LE:-?}"
}


_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)"; H
OOK_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-reb
oot-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-b
ased 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?" (n
o 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
 _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
```

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/`.

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
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 -> $OEM_NOCLOU
D_DST"
       cp -av "$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


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-alternat
e-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
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/* "$APT_DIR/debs" || echo "[oe
m] nvidia-xconfig not found baseos"
    # cp -rvf /cdrom/oemdata/debs/nvidia-xconfig* "$APT_DIR/debs" || echo "[oe
m] nvidia-xconfig not found oemdata"
    # cp -rvf /cdrom/oemdata/debs/libnvidia-cfg1* "$APT_DIR/debs" || echo "[oe
m] 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"


  # 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."
exit 0
```

12.2.2 repack_baseos.sh (Partial Excerpt)

```
# Step 4: Install OEM debs and OEM cloud-init / oem-iso-cfg.sh into ISO oemda
ta
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-i
so-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 file
s)."
        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 cl
oud.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 "========================================="
}


# 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() {
    ...
    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`):

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:%S
Z)"
        echo "Hostname: $(hostname)"
        echo "Datasource: $(cloud-init query datasource || true)"
        echo "Instance ID: $(cloud-init query instance_id || true)"
      } > "$audit"
      chmod 0644 "$audit"
```

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 inst
all.
system_info:
 default_user:
   name: nvidia
   groups: [sudo]
   shell: /bin/bash
   lock_passwd: false
```

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/noclo
ud/
# Without this, cloud-init reports DataSourceNone and ignores the seed files.
datasource_list: [NoCloud]
datasource:
 NoCloud:
   seedfrom: file:///var/lib/cloud/seed/nocloud/
```

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
```

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 withou
t 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/E
MrTEKlyVAokumi1qPIXbT89eOiEc85xD.


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


runcmd:
 - [ sh, -c, 'echo nvidia > /etc/hostname; hostname nvidia 2>/dev/null || tru
e; if grep -qE "^127\\.0\\.1\\.1[[:space:]]+nvidia" /etc/hosts 2>/dev/null; t
hen :; 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 nvid
ia" >> /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' 'PasswordAuth
entication 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 "$OEM_MNT"
   # One trap for all normal completion: umount + disable cloud-init on next
boots.
   # Do not use "exit" here: cloud-init shellifies all runcmd items into one
/bin/sh script; exit
   # would skip every later runcmd line (e.g. pending-reboot check) before th
e EXIT trap runs.
   # Do not use "set -e" in this block: if sync/mkdir/touch after the hook fa
ils, the shell would
   # exit before the post-hook runcmd; EXIT would still run _oemdata_exit (cl
oud-init.disabled)
   # but the pending-reboot log/reboot would never run.
   _oemdata_exit() {
     echo "OEMDATA exit trap: disabling cloud-init and unmounting OEMDATA"
     umount "$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" "$HOOK_RUN"
       chmod 700 "$HOOK_RUN"
       echo "Running OEM hook from $HOOK_RUN (OEMDATA root still $OEM_MNT unt
il umount)"
       set +e
       export OEM_MNT
       sh "$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-OEM
DATA path too).
 # Reboot if OEM hook requested it (hook exit 1). Runs in same shellified scr
ipt 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 OE
MDATA now.
    _oemdata_exit
    sync
    # Immediate reboot races remaining modules-final (e.g. cc_keys_to_consol
e) and can log SystemExit:1.
    # Background sleep + reboot: runcmd exits, cloud-init completes, then re
boot (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
```

13. Validation Scenarios and Feedback Questions

The scenarios in 13.1 align with Table 1. Installation and Update Patterns and the procedures in sections 4 through 12. Complete the scenario that matches your deployment. For log-based and post-installation verification that complements these flows, see 10. Verify the Customization and Installation Outcomes. To provide feedback to NVIDIA, use the prompts in 13.2.

Table 8. How Scenarios Map to This Document

§13.1 Scenario	Related Table 1 Patterns	Where to Work in This Guide
1. Customized BaseOS ISO	Cloud-Init OEM seed on the repacked ISO (with or without OOBE); `OEMDATA` optional	4. Customize the BaseOS Image with repack_baseos.sh; 8.4 Cloud-Init Integration; 10.1 During and After an ISO-Based Installation; 12.2 ISO installation scripts and OEM Cloud-Init
2. Air-gapped USB installation	USB-hosted packages and firmware or LOCAL or MIRRORED sources through `OEMDATA` and `hook.sh`	5. USB Partitioning and the OEMDATA Layout; 6. Host a Minimal APT Repository and Firmware Tree and 7. Mirror the Full Ubuntu Ports and LVFS Content on a Server; 8. Client Configuration and hook.sh; 12.1 OEMDATA hook.sh and Cloud-Init seed
3. Local repository + DGX Spark Preview updates (OTA2604)	Curated or mirrored APT layout; can extend beyond first-boot automation	6; 8; 10.2 After Mirror- or USB-Driven Updates. If your DGX Spark Preview (OTA2604) update process is separate from this guide's ISO/`OEMDATA` flow, apply the same repository patterns from §6 and §8 and record any additional steps in your runbook.

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.

   In this guide: Refer to 4. Customize the BaseOS Image with repack_baseos.sh and place OEM Cloud-Init under oemdata/cloud-init/ as in 12.2 ISO installation scripts and OEM Cloud-Init. Adjust user-data and meta-data per 8.4 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 §12.2).
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 10.1 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.

   In this guide: Prepare the USB layout per 5. USB Partitioning and the OEMDATA Layout. Host packages and firmware using 6 (minimal tree) or 7 (full mirror), then configure the client with hook.sh and optional URL files as in 6.2 and 8. Reference script: 12.1 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 §5, §6, and §7 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 guide: hook.sh stays on OEMDATA, but first boot runs a copy under /tmp with OEM_MNT set to the mount (12.1, 12.2.7 user-data listing).
   3. Perform installation using your customized ISO and attached OEMDATA media as described in scenario 1 and §5.
   4. Verify packages, firmware, and customizations using 10.1 and 10.2, 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).

   In this guide: The minimal and full-mirror layouts in 6 and 7 show HTTP-served package trees; 8 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 §6 and §8 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 10.2 for verification after updates driven by hook.sh or the mirror, where applicable).

13.2 Questions for customers

Use the following prompts after you complete the relevant checks in 13.1. If a response depends on a specific flow, cite the section number or Table 1 pattern (for example, “USB-hosted `OEMDATA`” or “full mirror with `hook.sh`”).

1. Provide feedback on Himmelblau as your Active Directory integration path, or describe how you plan to implement single sign-on (SSO) and directory integration with Microsoft Active Directory or Entra ID.
2. After you review this document, which additional topics should it cover?
3. If you used reference scripts or listings from this document (for example, `hook.sh` in §12.1), which aspects worked well and which should change?

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