Installing Tools on Dell R750

Table of contents

Install ptp4l and phc2sys

Setup the Boot Configuration Service

List of Figures

This chapter describes how to install the required kernel, driver, and tools on the host. This is a one-time installation and can be skipped if the system has been configured already.

- In the following sequence of steps, the target host is Dell PowerEdge R750.
- Depending on the release, tools that are installed in this section may need to be upgraded in the Installing and Upgrading Aerial cuBB section.
- After everything is installed and updated, refer to the *cuBB Quick Start Guide* on how to use Aerial cuBB.

Dell PowerEdge R750 Server Configuration

1. Dual Intel Xeon Gold 6336Y CPU @ 2.4G, 24C/48T (185W)
2. 512GB RDIMM, 3200MT/s
3. 1.92TB, Enterprise NVMe
4. Riser Config 2, Full Length, 4x16, 2x8 slots (PCIe gen 4)
5. Dual, Hot-Plug Power Supply Redundant (1+1), 1400W or 2400W
6. GPU Enablement
7. NVIDIA Converged Accelerator: A100X or AX800

Converged Accelerator Installation

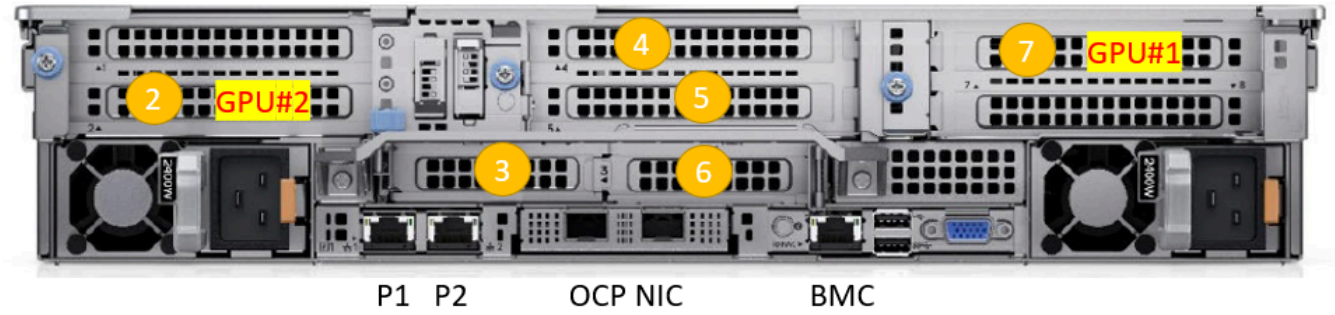
R750 supports PCIe 4.0 x16 at slot 2,3,6,7 and x8 at slot 4,5. Follow the table below to install single or dual converged accelerator in the assigned slot and ensure the GPU power cable is connected properly. These are the GPU installation instructions from *Dell R750 Installation Manual*.

NOTE: Only use *SIG_PWR_3* and *SIG_PWR_4* connectors on the motherboard for GPU power.

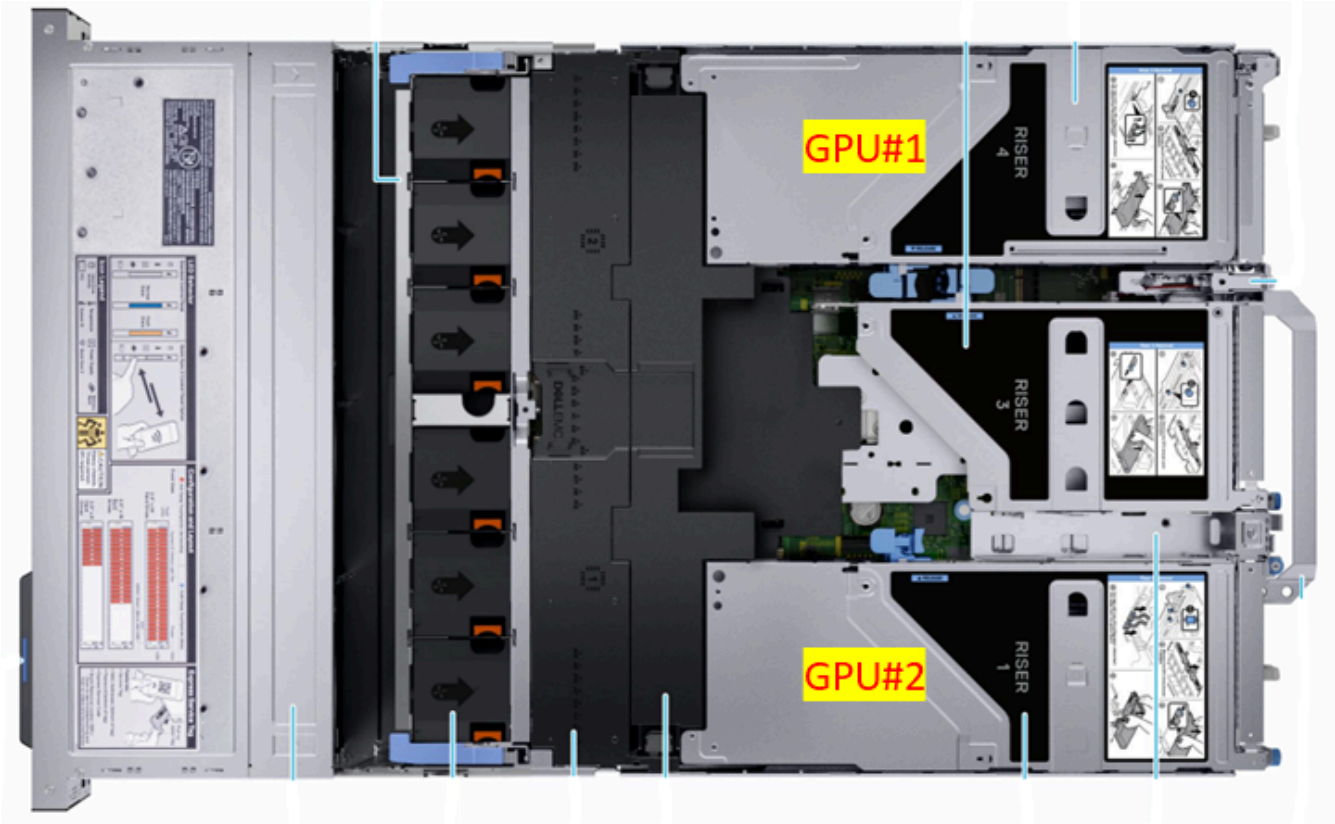
GPU	Slot	GPU Power	NUMA
GPU#1	7 (Riser 4)	SIG_PWR_3	1

GPU#2	2 (Riser 1)	SIG_PWR_4	0

Rear View:



Top View:



Cable Connection

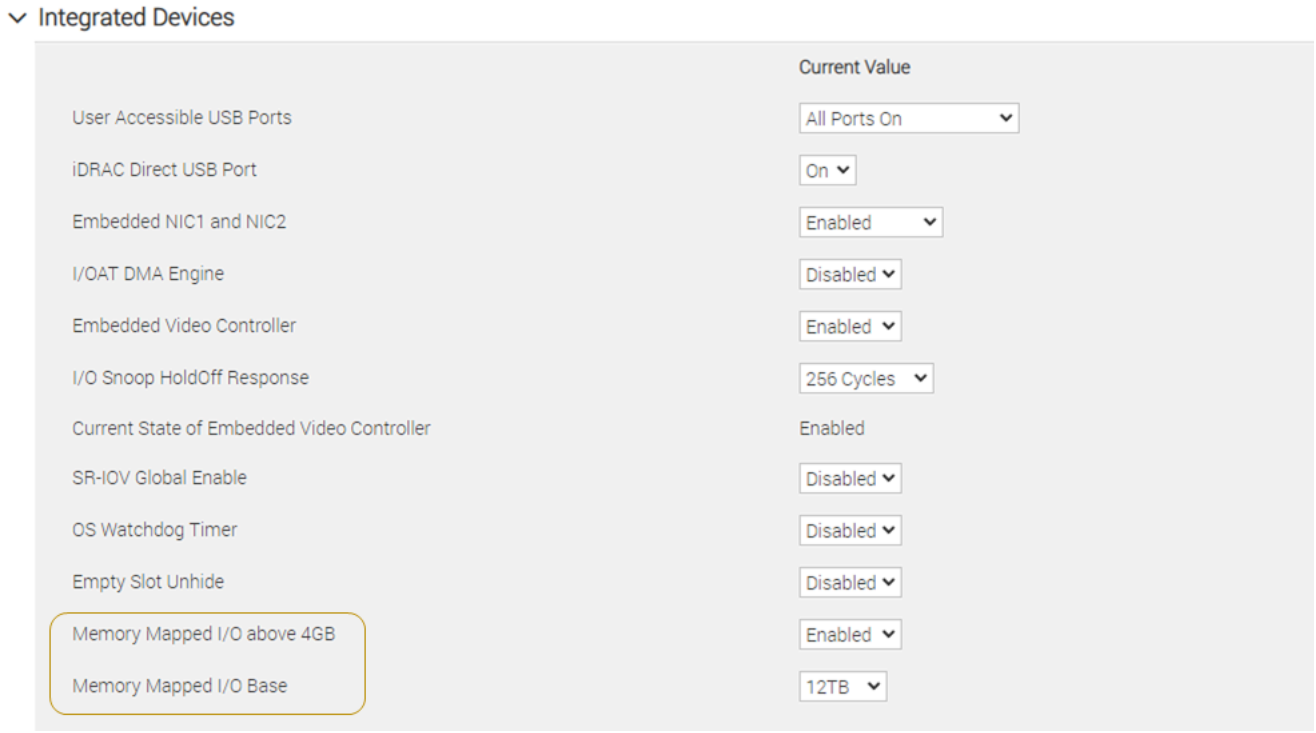
1. To run end-to-end test with O-RU, the converged accelerator port#0 or port#1 must be connected to the fronthaul switch. Make sure the PTP is configured to use the port connected to the fronthaul switch.

2. To run cuBB end-to-end test with TestMAC and RU emulator, an Aerial Devkit is required to run RU emulator. The converged accelerator port#1 on R750 must be connected to CX6-DX NIC port#0 on Aerial Devkit (RU emulator server) via Mellanox 100GbE direct attach copper cable.

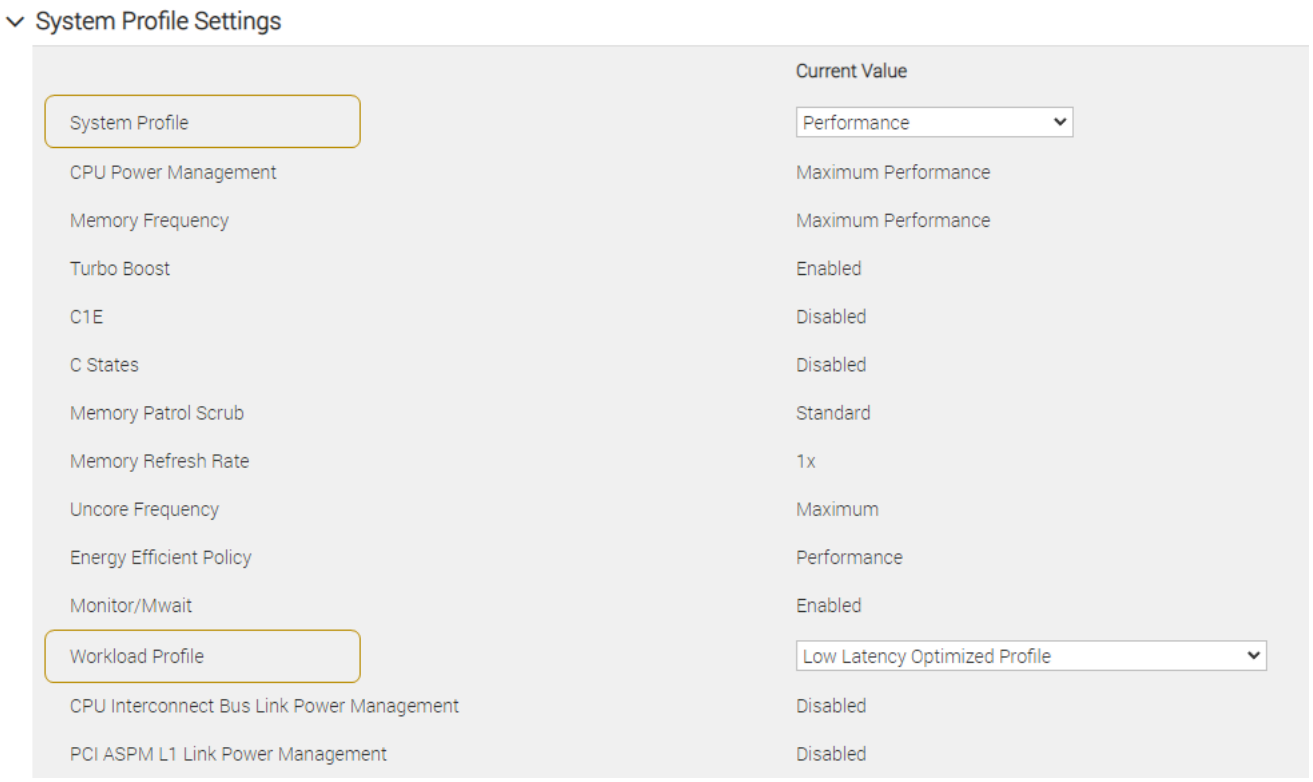
Configure BIOS Settings

During the first boot, change the BIOS settings in the following order. The same settings can be changed via BMC: **Configuration** **BIOS Settings**.

Integrated Devices: Enable Memory Mapped I/O above 4GB and change Memory Mapped I/O Base to *12TB*.

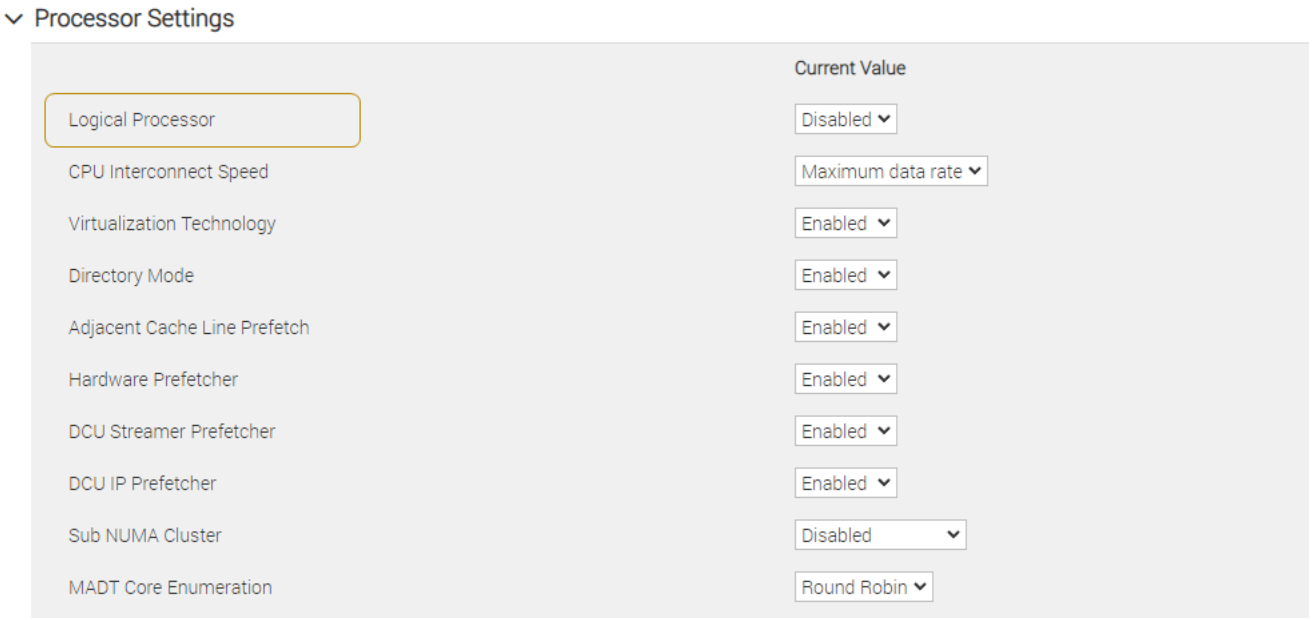


System Profile Settings: Change System Profile to *Performance* and Workload Profile to *Low Latency Optimized Profile*.



Processor Settings: Aerial CUDA-Accelerated RAN supports both *HyperThreaded mode (experimental)* or *non-HyperThreaded mode (default)* but make sure the kernel command line and the CPU core affinity in the cuPHYController YAML match the BIOS settings.

To enable HyperThreading, enable the Logical Processor. To disable HyperThreading, disable the Logical Processor.



Save the BIOS settings, then reboot the system.

Install Ubuntu 22.04 Server

After installing Ubuntu 22.04 Server, verify the following:

- System time is correct to avoid apt update error. If not, see How to fix system time.
- LVM volume uses the whole disk space. If not, see How to resize LVM volume.
- GPU and NIC are detected by the OS:

  Use the following commands to determine whether the GPU and NIC are detected by the OS:

  > $ lspci | grep -i nvidia *# If the system has A100X GPU installed* cf:00.0 3D controller: NVIDIA Corporation Device 20b8 (rev a1) *# If the system has AX800 GPU installed* cf:00.0 3D controller: NVIDIA Corporation Device 20fd (rev a1) $ lspci | grep -i mellanox *# If the system has A100X GPU installed* cc:00.0 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller (rev 01) cc:00.1 Ethernet controller: Mellanox Technologies MT42822 BlueField-2 integrated ConnectX-6 Dx network controller (rev 01) *# If the system has AX800 GPU installed* cc:00.0 Ethernet controller: Mellanox Technologies MT43244 BlueField-3 integrated ConnectX-7 network controller (rev 01) cc:00.1 Ethernet controller: Mellanox Technologies MT43244 BlueField-3 integrated ConnectX-7 network controller (rev 01)

Disable Auto Upgrade

Edit the `/etc/apt/apt.conf.d/20auto-upgrades` system file, and change the “1” to “0” for both lines. This prevents the installed version of the low latency kernel from being accidentally changed with a subsequent software upgrade.

> $ sudo nano /etc/apt/apt.conf.d/20auto-upgrades APT::Periodic::Update-Package-Lists "0"; APT::Periodic::Unattended-Upgrade "0";

Install the Low-Latency Kernel

If the low latency kernel is not installed, you must remove the old kernels and keep only the latest generic kernel. Enter the following command to list the installed kernels:

```
$ dpkg --list | grep -i 'linux-image' | awk '/ii/{ print $2}' # To remove old kernel $ sudo apt-get purge linux-image-<old kernel version> $ sudo apt-get autoremove
```

Install the low-latency kernel with the specific version listed in the release manifest.

```
$ sudo apt-get update $ sudo apt-get install -y linux-image-5.15.0-1042-nvidia-lowlatency
```

Update the GRUB to change the default boot kernel:

```
# Update grub to change the default boot kernel $ sudo sed -i 's/^GRUB_DEFAULT=.*/GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 5.15.0-1042-nvidia-lowlatency"/' /etc/default/grub
```

Configure Linux Kernel Command-line

To set kernel command-line parameters, edit the `GRUB_CMDLINE_LINUX_DEFAULT` parameter in the GRUB file `/etc/default/grub` and append/update the parameters described below. The following kernel parameters are optimized for Xeon Gold 6336Y CPU and 512GB memory.

To automatically append the GRUB file with these changes, enter this command:

```
# When HyperThread is disabled (default) $ sudo sed -i 's/^GRUB_CMDLINE_LINUX_DEFAULT="[^"]*/& pci=realloc=off default_hugepagesz=1G hugepagesz=1G hugepages=16 tsc=reliable clocksource=tsc intel_idle.max_cstate=0 mce=ignore_ce processor.max_cstate=0 intel_pstate=disable audit=0 idle=poll rcu_nocb_poll nosoftlockup iommu=off irqaffinity=0-3,44-47 isolcpus=managed_irq,domain,4-43 nohz_full=4-43 rcu_nocbs=4-43 noht numa_balancing=disable/' /etc/default/grub # When HyperThread is enabled
```

```
(experimental) $ sudo sed -i 's/^GRUB_CMDLINE_LINUX_DEFAULT="[^"]*/&
pci=realloc=off default_hugepagesz=1G hugepagesz=1G hugepages=16 tsc=reliable
clocksource=tsc intel_idle.max_cstate=0 mce=ignore_ce processor.max_cstate=0
intel_pstate=disable audit=0 idle=poll rcu_nocb_poll nosoftlockup iommu=off
irqaffinity=0-3,92-95 isolcpus=managed_irq,domain,4-91 nohz_full=4-91
rcu_nocbs=4-91 numa_balancing=disable/' /etc/default/grub
```

The CPU-cores-related parameters must be adjusted depending on the number of CPU cores on the system. In the example above, the “4-43” value represents CPU core numbers 4 to 43; you may need to adjust this parameter depending on the HW configuration. By default, only one DPDK thread is used. The isolated CPUs are used by the entire cuBB software stack. Use the `nproc --all` command to see how many cores are available. Do not use core numbers that are beyond the number of available cores.

> ⚠ **Warning**
>
> These instructions are specific to Ubuntu 22.04 with a 5.15 low-latency kernel provided by Canonical. Make sure the kernel commands provided here are suitable for your OS and kernel versions and revise these settings to match your system if necessary.

Apply the Changes and Reboot to Load the Kernel

```
$ sudo update-grub $ sudo reboot
```

After rebooting, enter the following command to verify that the system has booted into the low-latency kernel:

```
$ uname -r 5.15.0-1042-nvidia-lowlatency
```

Enter this command to verify that the kernel command-line parameters are configured properly:

```
$ cat /proc/cmdline BOOT_IMAGE=/vmlinuz-5.15.0-1042-nvidia-lowlatency root=/dev/mapper/ubuntu--vg-ubuntu--lv ro pci=realloc=off default_hugepagesz=1G hugepagesz=1G hugepages=16 tsc=reliable clocksource=tsc intel_idle.max_cstate=0 mce=ignore_ce processor.max_cstate=0 intel_pstate=disable audit=0 idle=poll rcu_nocb_poll nosoftlockup iommu=off irqaffinity=0-3,44-47 isolcpus=managed_irq,domain,4-43 nohz_full=4-43 rcu_nocbs=4-43 noht numa_balancing=disable
```

Enter this command to verify if hugepages are enabled:

```
$ grep -i huge /proc/meminfo AnonHugePages: 0 kB ShmemHugePages: 0 kB FileHugePages: 0 kB HugePages_Total: 16 HugePages_Free: 16 HugePages_Rsvd: 0 HugePages_Surp: 0 Hugepagesize: 1048576 kB Hugetlb: 16777216 kB
```

Disabling Nouveau

Enter this command to disable nouveau:

```
$ cat <<EOF | sudo tee /etc/modprobe.d/blacklist-nouveau.conf blacklist nouveau options nouveau modeset=0 EOF
```

Regenerate the kernel initramfs and reboot the system:

```
$ sudo update-initramfs -u $ sudo reboot
```

Install Dependency Packages

Enter these commands to install prerequisite packages:

```
$ sudo apt-get update $ sudo apt-get install -y build-essential linux-headers-$(uname -r) dkms unzip linuxptp pv
```

Install RSHIM and Mellanox Firmware Tools on the Host

Note

1. Aerial has been using Mellanox inbox driver instead of MOFED since the 23-4 release. MOFED must be removed if it is installed on the system.

2. RSHIM package is shared via PID account. If you cannot access it, contact NVIDIA CPM.

Check if there is an existing MOFED installed on the host system.

```
$ ofed_info -s MLNX_OFED_LINUX-23.07-0.5.0.0:
```

Uninstall MOFED if it is present.

```
$ sudo /usr/sbin/ofed_uninstall.sh
```

Download the rshim package here and copy it to the local file system on the server.

Enter the following commands to install rshim driver.

```
# Install rshim $ sudo apt-get install libfuse2 $ sudo dpkg -i
rshim_2.0.17.g0caa378_amd64.deb
```

Enter the following commands to install Mellanox firmware tools.

```
# Install Mellanox Firmware Tools $ export MFT_VERSION=4.26.1-3 $ wget
https://www.mellanox.com/downloads/MFT/mft-$MFT_VERSION-x86_64-deb.tgz $
tar xvf mft-$MFT_VERSION-x86_64-deb.tgz $ sudo mft-$MFT_VERSION-x86_64-
deb/install.sh # Verify the install Mellanox firmware tool version $ sudo mst version
mst, mft 4.26.1-3, built on Nov 27 2023, 15:24:39. Git SHA Hash: N/A $ sudo mst
start # check NIC PCIe bus addresses and network interface names $ sudo mst status -v
# Here is the result of GPU#1 on slot 7 MST modules: ---------- MST PCI module is not
```

```
loaded MST PCI configuration module loaded PCI devices: ---------- DEVICE_TYPE
MST PCI RDMA NET NUMA BlueField3(rev:1) /dev/mst/mt41692_pciconf0.1 cc:00.1
mlx5_1 net-aerial00 BlueField3(rev:1) /dev/mst/mt41692_pciconf0 cc:00.0 mlx5_0
net-aerial01 1
```

Enter these commands to check the link status of port 0:

```
# Here is an example if port 0 is connected to another server via a 100GbE DAC cable. $
sudo mlxlink -d cc:00.0 Operational Info ---------- State : Active Physical state :
LinkUp Speed : 100G Width : 4x FEC : Standard RS-FEC - RS(528,514) Loopback Mode
: No Loopback Auto Negotiation : ON Supported Info ---------- Enabled Link Speed
(Ext.) : 0x00003ff2
(200G_2X,200G_4X,100G_1X,100G_2X,100G_4X,50G_1X,50G_2X,40G,25G,10G,1G)
Supported Cable Speed (Ext.) : 0x000002f2 (100G_4X,50G_2X,40G,25G,10G,1G)
Troubleshooting Info ---------- Status Opcode : 0 Group Opcode : N/A
Recommendation : No issue was observed Tool Information ---------- Firmware
Version : 24.39.2048 amBER Version : 2.22 MFT Version : mft 4.26.1-3
```

Install the CUDA Driver

> **Note**
>
> Aerial has been using the open-source GPU kernel driver (OpenRM) since the 23-4 release.

If the system has older driver installed, you must unload the current driver modules and uninstall the old driver.

```
# Unload the current driver modules $ for m in $(lsmod | awk "/^[^[:space:]]*
(nvidia|nv_|gdrdrv)/ {print \$1}"); do echo Unload $m...; sudo rmmod $m; done #
Remove the driver if it was installed by runfile installer before. $ sudo /usr/bin/nvidia-
uninstall
```

Run the following commands to install the **NVIDIA open-source GPU kernel driver** (OpenRM).

```
# Install CUDA driver $ wget https://download.nvidia.com/XFree86/Linux-x86_64/535.54.03/NVIDIA-Linux-x86_64-535.54.03.run $ sudo sh NVIDIA-Linux-x86_64-535.54.03.run --silent -m kernel-open # Verify that the driver is loaded successfully $ nvidia-smi +-----------------------------------------------------------------------------+ | NVIDIA-SMI 535.54.03 Driver Version: 535.54.03 CUDA Version: 12.2 | |-------------------------------+----------------------+----------------------+ | GPU Name Persistence-M | Bus-Id Disp.A | Volatile Uncorr. ECC | | Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. | | | | MIG M. | |=========================================+======================+======
| 0 NVIDIA A100X Off | 00000000:CF:00.0 Off | 0 | | N/A 40C P0 70W / 300W | 4MiB / 81920MiB | 34% Default | | | | Disabled | +-----------------------------------------+----------------------+----------------------+ +-----------------------------------------------------------------------------+ | Processes: | | GPU GI CI PID Type Process name GPU Memory | | ID ID Usage |
|=============================================================================
| No running processes found | +-----------------------------------------------------------------------------+
```

Install the GDRCopy Driver

Run the following commands to install the GDRCopy driver. If the system has an older version installed, you must remove the old driver.

> ⚠ **Warning**
>
> GDRCopy driver must be installed after CUDA.

```
# Check the installed GDRCopy driver version $ apt list --installed | grep gdrdrv-dkms # Remove the driver if you have the older version installed. $ sudo apt purge gdrdrv-
```

dkms $ sudo apt autoremove *# Install GDRCopy driver* $ wget https://developer.download.nvidia.com/compute/redist/gdrcopy/CUDA%2012.2/ubun dkms_2.4-1_amd64.Ubuntu22_04.deb $ sudo dpkg -i gdrdrv-dkms_2.4-1_amd64.Ubuntu22_04.deb

Install Docker CE

The full official instructions for installing Docker CE can be found on the Docker website: https://docs.docker.com/engine/install/ubuntu/#install-docker-engine. The following instructions are one supported way of installing Docker CE:

> ⚠ **Warning**
>
> To work correctly, the CUDA driver must be installed before Docker CE or nvidia-container-toolkit installation. It is recommended that you install the CUDA driver before installing Docker CE or the nvidia-container-toolkit.

$ sudo apt-get update $ sudo apt-get install -y ca-certificates curl gnupg $ sudo install -m 0755 -d /etc/apt/keyrings $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg $ sudo chmod a+r /etc/apt/keyrings/docker.gpg $ echo \ "deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \ "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null $ sudo apt-get update $ sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin $ sudo docker run hello-world

Install the Nvidia Container Toolkit

Locate and follow the nvidia-container-toolkit install instructions.

Or use the following instructions as an alternate way to install the nvidia-container-toolkit. Version **1.14.1-1** is supported.

> ⚠ **Warning**
>
> To work correctly, the CUDA driver must be installed before Docker CE or nvidia-container-toolkit installation. It is recommended that you install the CUDA driver before installing Docker CE or the nvidia-container-toolkit.

```
$ curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \ && curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \ sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \ sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list \ && \ sudo apt-get update $ sudo apt-get install -y nvidia-container-toolkit $ sudo nvidia-ctk runtime configure --runtime=docker $ sudo systemctl restart docker $ sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Update AX800 BFB Image and NIC Firmware

NOTE: The following instructions are for AX800 board specifically. Validate that RSHIM and MFT are installed on the system.

```
# Enable MST $ sudo mst start $ sudo mst status MST modules: ---------- MST PCI module is not loaded MST PCI configuration module loaded MST devices: ---------- /dev/mst/mt41692_pciconf0 - PCI configuration cycles access. domain:bus:dev.fn=0000:cc:00.0 addr.reg=88 data.reg=92 cr_bar.gw_offset=-1 Chip revision is: 01 # Change to DPU mode $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 -y s INTERNAL_CPU_MODEL=1 INTERNAL_CPU_OFFLOAD_ENGINE=0 # NOTE: Requires a power cycle to take effect $ sudo reboot # Update BFB image first # For AX800 TS2 board, download the AX800 QP BFB from https://apps.nvidia.com/PID/ContentLibraries/Detail/1111948 and copy
```

DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04-1.23-10.bfb to local file system. # Here is the command to flash BFB for AX800 TS2 board $ sudo bfb-install -r rshim0 -b DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04-1.23-10.bfb *# For AX800 production board, download the AX800 production BFB* $ wget https://content.mellanox.com/BlueField/BFBs/Ubuntu22.04/DOCA_2.5.0_BSP_4.5.0_Ub 1.23-10.prod.bfb *# Here is the command to flash BFB for AX800 production board* $ sudo bfb-install -r rshim0 -b DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04-1.23-10.prod.bfb Pushing bfb 1.20GiB 0:01:06 [18.5MiB/s] [Collecting BlueField booting status. Press Ctrl+C to stop... INFO[MISC]: PSC BL1 START INFO[BL2]: start INFO[BL2]: DDR POST passed INFO[BL2]: UEFI loaded INFO[BL31]: start INFO[BL31]: lifecycle Production INFO[BL31]: PTMERROR: Unknown OPN INFO[BL31]: runtime INFO[UEFI]: eMMC init INFO[UEFI]: eMMC probed INFO[UEFI]: UPVS valid INFO[UEFI]: PMI: updates started INFO[UEFI]: PMI: boot image update INFO[UEFI]: PMI: updates completed, status 0 INFO[UEFI]: PCIe enum start INFO[UEFI]: PCIe enum end INFO[UEFI]: exit Boot Service INFO[MISC]: Ubuntu installation started INFO[MISC]: Installing OS image INFO[MISC]: Installation finished *# Wait 10 minutes to ensure the card initializes properly after the BFB installation* $ sleep 600 *# Update NIC FW # For AX800 TS2 board, download the AX800 QP firmware from https://apps.nvidia.com/PID/ContentLibraries/Detail/1111948 and copy fw-BlueField-3-rel-32_39_2048-699-21014-0230-EB1_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.bin to local file system. # Here is the command to flash firmware for AX800 TS2 board* $ sudo flint -d /dev/mst/mt41692_pciconf0 -i fw-BlueField-3-rel-32_39_2048-699-21014-0230-EB1_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.bin -y b *# For AX800 production board, download the AX800 production NIC firmware* $ wget https://www.mellanox.com/downloads/firmware/fw-BlueField-3-rel-32_39_2048-699-21014-0230-EB1_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin.zip $ unzip fw-BlueField-3-rel-32_39_2048-699-21014-0230-EB1_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin.zip *# Here is the command to flash firmware for AX800 production board* $ sudo flint -d /dev/mst/mt41692_pciconf0 -i fw-BlueField-3-rel-32_39_2048-699-21014-0230-EB1_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin -y b Current FW version on flash: 32.37.1300 New FW version: 32.39.2048 FSMST_INITIALIZE - OK Writing Boot image component - OK Restoring signature - OK *# Change to NIC mode* $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 -y s INTERNAL_CPU_MODEL=1

INTERNAL_CPU_OFFLOAD_ENGINE=1 *# NOTE: Requires a full power cycle from host with cold boot # Verify NIC FW version after reboot* $ sudo mst start $ sudo flint -d /dev/mst/mt41692_pciconf0 q Image type: FS4 FW Version: 32.39.2048 FW Release Date: 29.11.2023 Product Version: 32.39.2048 Rom Info: type=UEFI Virtio net version=21.4.13 cpu=AMD64,AARCH64 type=UEFI Virtio blk version=22.4.12 cpu=AMD64,AARCH64 type=UEFI version=14.32.17 cpu=AMD64,AARCH64 type=PXE version=3.7.300 cpu=AMD64 Description: UID GuidsNumber Base GUID: 48b02d0300a615ca 22 Base MAC: 48b02da615ca 22 Image VSD: N/A Device VSD: N/A PSID: NVD0000000038 Security Attributes: secure-fw

Run the following code to switch the AX800 to the **BF3-as-CX** mode:

 # Setting BF3 port to Ethernet mode (not Infiniband) $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set LINK_TYPE_P1=2 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set LINK_TYPE_P2=2 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set INTERNAL_CPU_MODEL=1 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set INTERNAL_CPU_PAGE_SUPPLIER=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set INTERNAL_CPU_ESWITCH_MANAGER=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set INTERNAL_CPU_IB_VPORT0=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set INTERNAL_CPU_OFFLOAD_ENGINE=DISABLED $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set CQE_COMPRESSION=1 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set PROG_PARSE_GRAPH=1 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set ACCURATE_TX_SCHEDULER=1 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set FLEX_PARSER_PROFILE_ENABLE=4 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set REAL_TIME_CLOCK_ENABLE=1 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set EXP_ROM_VIRTIO_NET_PXE_ENABLE=0 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set EXP_ROM_VIRTIO_NET_UEFI_ARM_ENABLE=0 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set EXP_ROM_VIRTIO_NET_UEFI_x86_ENABLE=0 $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 --yes set EXP_ROM_VIRTIO_BLK_UEFI_ARM_ENABLE=0 $ sudo mlxconfig -d

/dev/mst/mt41692_pciconf0 --yes set EXP_ROM_VIRTIO_BLK_UEFI_x86_ENABLE=0 # *NOTE: Requires a full power cycle from host with cold boot # Verify that the NIC FW changes have been applied* $ sudo mlxconfig -d /dev/mst/mt41692_pciconf0 q | grep "CQE_COMPRESSION\|PROG_PARSE_GRAPH\|ACCURATE_TX_SCHEDULER\|FLEX_PARS INTERNAL_CPU_MODEL EMBEDDED_CPU(1) INTERNAL_CPU_PAGE_SUPPLIER EXT_HOST_PF(1) INTERNAL_CPU_ESWITCH_MANAGER EXT_HOST_PF(1) INTERNAL_CPU_IB_VPORT0 EXT_HOST_PF(1) INTERNAL_CPU_OFFLOAD_ENGINE DISABLED(1) FLEX_PARSER_PROFILE_ENABLE 4 PROG_PARSE_GRAPH True(1) ACCURATE_TX_SCHEDULER True(1) CQE_COMPRESSION AGGRESSIVE(1) REAL_TIME_CLOCK_ENABLE True(1) LINK_TYPE_P1 ETH(2) LINK_TYPE_P2 ETH(2)

Update A100X BFB Image and NIC Firmware

NOTE: The following instructions are specifically for A100X boards. Ensure RSHIM and MFT are installed on the system.

 # Enable MST $ sudo mst start $ sudo mst status MST modules: ---------- MST PCI module is not loaded MST PCI configuration module loaded MST devices: ---------- /dev/mst/mt41686_pciconf0 - PCI configuration cycles access. domain:bus:dev.fn=0000:cc:00.0 addr.reg=88 data.reg=92 cr_bar.gw_offset=-1 Chip revision is: 01 *# Change to DPU mode* $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 -y s INTERNAL_CPU_MODEL=1 INTERNAL_CPU_OFFLOAD_ENGINE=0 *# NOTE: Requires a power cycle to take effect* $ sudo reboot *# Update BFB image first* $ wget https://content.mellanox.com/BlueField/BFBs/Ubuntu22.04/DOCA_2.5.0_BSP_4.5.0_Ub 1.23-10.prod.bfb $ sudo bfb-install -r rshim0 -b DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04-1.23-10.prod.bfb Pushing bfb 920MiB 0:01:51 [8.22MiB/s] [<=>] Collecting BlueField booting status. Press Ctrl+C to stop... INFO[BL2]: start INFO[BL2]: DDR POST passed INFO[BL2]: UEFI loaded INFO[BL31]: start INFO[BL31]: lifecycle Secured (development) INFO[BL31]: runtime INFO[UEFI]: eMMC init INFO[UEFI]: UPVS valid INFO[UEFI]: eMMC probed INFO[UEFI]: PMI: updates started INFO[UEFI]: PMI: boot image update INFO[UEFI]: PMI: updates completed, status 0 INFO[UEFI]: PCIe enum start INFO[UEFI]: PCIe enum end INFO[UEFI]: exit Boot Service INFO[MISC]: Ubuntu installation started INFO[MISC]: Installing OS image INFO[MISC]: Installation finished *# Wait 10 minutes to ensure the*

card initializes properly after the BFB installation $ sleep 600 *# Update NIC firmware* $ wget https://www.mellanox.com/downloads/firmware/fw-BlueField-2-rel-24_39_2048-699210040230_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin.zip $ unzip fw-BlueField-2-rel-24_39_2048-699210040230_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin.zip $ sudo flint -d /dev/mst/mt41686_pciconf0 -i fw-BlueField-2-rel-24_39_2048-699210040230_Ax-NVME-20.4.1-UEFI-21.4.13-UEFI-22.4.12-UEFI-14.32.17-FlexBoot-3.7.300.signed.bin -y b Current FW version on flash: 24.35.1012 New FW version: 24.39.2048 FSMST_INITIALIZE - OK Writing Boot image component - OK Restoring signature - OK *# Change to NIC mode* $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 -y s INTERNAL_CPU_MODEL=1 INTERNAL_CPU_OFFLOAD_ENGINE=1 *# NOTE: Requires a full power cycle from host with cold boot # Verify NIC FW version after reboot* $ sudo mst start $ sudo flint -d /dev/mst/mt41686_pciconf0 q Image type: FS4 FW Version: 24.39.2048 FW Release Date: 29.11.2023 Product Version: 24.39.2048 Rom Info: type=UEFI Virtio net version=21.4.13 cpu=AMD64,AARCH64 type=UEFI Virtio blk version=22.4.12 cpu=AMD64,AARCH64 type=UEFI version=14.32.17 cpu=AMD64,AARCH64 type=PXE version=3.7.300 cpu=AMD64 Description: UID GuidsNumber Base GUID: 48b02d03005f770c 16 Base MAC: 48b02d5f770c 16 Image VSD: N/A Device VSD: N/A PSID: NVD0000000015 Security Attributes: secure-fw

Run the following code to switch the A100x to the **BF2-as-CX** mode:

 # Setting BF2 port to Ethernet mode (not Infiniband) $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set LINK_TYPE_P1=2 $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set LINK_TYPE_P2=2 *# Setting BF2 Embedded CPU mode* $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set INTERNAL_CPU_MODEL=1 $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set INTERNAL_CPU_PAGE_SUPPLIER=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set INTERNAL_CPU_ESWITCH_MANAGER=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set INTERNAL_CPU_IB_VPORT0=EXT_HOST_PF $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set INTERNAL_CPU_OFFLOAD_ENGINE=DISABLED *# Accurate scheduling related settings* $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set CQE_COMPRESSION=1 $

```
sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set PROG_PARSE_GRAPH=1 $
sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes set
ACCURATE_TX_SCHEDULER=1 $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 --yes
set FLEX_PARSER_PROFILE_ENABLE=4 $ sudo mlxconfig -d
/dev/mst/mt41686_pciconf0 --yes set REAL_TIME_CLOCK_ENABLE=1 # NOTE:
Requires a power cycle of the host for those settings to take effect # Verify that the NIC
FW changes have been applied $ sudo mlxconfig -d /dev/mst/mt41686_pciconf0 q |
grep "CQE_COMPRESSION\|PROG_PARSE_GRAPH\|ACCURATE_TX_SCHEDULER\
\|FLEX_PARSER_PROFILE_ENABLE\|REAL_TIME_CLOCK_ENABLE\|INTERNAL_CPU_MOD
\|INTERNAL_CPU_PAGE_SUPPLIER\|INTERNAL_CPU_ESWITCH_MANAGER\|INTERNAL_
        INTERNAL_CPU_MODEL                EMBEDDED_CPU(1)
        INTERNAL_CPU_PAGE_SUPPLIER        EXT_HOST_PF(1)
        INTERNAL_CPU_ESWITCH_MANAGER      EXT_HOST_PF(1)
        INTERNAL_CPU_IB_VPORT0            EXT_HOST_PF(1)
        INTERNAL_CPU_OFFLOAD_ENGINE       DISABLED(1)
        FLEX_PARSER_PROFILE_ENABLE        4
        PROG_PARSE_GRAPH                  True(1)
        ACCURATE_TX_SCHEDULER             True(1)
        CQE_COMPRESSION                   AGGRESSIVE(1)
        REAL_TIME_CLOCK_ENABLE            True(1)
        LINK_TYPE_P1                  ETH(2)        LINK_TYPE_P2                  ETH(2)
```

Set Persistent NIC Interface Name

Configure the network link files so that the NIC interfaces always come up with the same name. Run `lshw -c network -businfo` to find the current interface name on the target bus address then run `ip link` to find the corresponding MAC address by the interface name. After identifying the MAC address, create files at /etc/systemd/network/NN-persistent-net.link with the following information:

```
[Match] MACAddress={{item.mac}} [Link] Name={{item.name}}
```

The following network link files set the converged accelerator port#0 to aerial00 and port#1 to aerial01:

```
$ sudo nano /etc/systemd/network/11-persistent-net.link # Update the MAC address to match the converged accelerator port 0 MAC address [Match] MACAddress=48:b0:2d:xx:xx:xx [Link] Name=aerial00 $ sudo nano /etc/systemd/network/12-persistent-net.link # Update the MAC address to match the converged accelerator port 1 MAC address [Match] MACAddress=48:b0:2d:yy:yy:yy [Link] Name=aerial01
```

Reboot the system after creating these files.

Install ptp4l and phc2sys

Enter these commands to configure PTP4L assuming the `aerial00` NIC interface and CPU core **41** are used for PTP:

```
$ cat <<EOF | sudo tee /etc/ptp.conf [global] dataset_comparison G.8275.x G.8275.defaultDS.localPriority 128 maxStepsRemoved 255 logAnnounceInterval -3 logSyncInterval -4 logMinDelayReqInterval -4 G.8275.portDS.localPriority 128 network_transport L2 domainNumber 24 tx_timestamp_timeout 30 slaveOnly 1 clock_servo pi step_threshold 1.0 egressLatency 28 pi_proportional_const 4.65 pi_integral_const 0.1 [aerial00] announceReceiptTimeout 3 delay_mechanism E2E network_transport L2 EOF cat <<EOF | sudo tee /lib/systemd/system/ptp4l.service [Unit] Description=Precision Time Protocol (PTP) service Documentation=man:ptp4l After=network.target [Service] Restart=always RestartSec=5s Type=simple ExecStartPre=ifconfig aerial00 up ExecStartPre=ethtool --set-priv-flags aerial00 tx_port_ts on ExecStartPre=ethtool -A aerial00 rx off tx off ExecStartPre=ifconfig aerial01 up ExecStartPre=ethtool --set-priv-flags aerial01 tx_port_ts on ExecStartPre=ethtool -A aerial01 rx off tx off ExecStart=taskset -c 41 /usr/sbin/ptp4l -f /etc/ptp.conf [Install] WantedBy=multi-user.target EOF $ sudo systemctl daemon-reload $ sudo systemctl restart ptp4l.service $ sudo systemctl enable ptp4l.service
```

One server becomes the master clock, as shown below:

```
$ sudo systemctl status ptp4l.service • ptp4l.service - Precision Time Protocol (PTP) service Loaded: loaded (/lib/systemd/system/ptp4l.service; enabled; vendor preset: enabled) Active: active (running) since Tue 2023-08-08 19:37:56 UTC; 2 weeks 3 days
```

```
ago Docs: man:ptp4l Main PID: 1120 (ptp4l) Tasks: 1 (limit: 94533) Memory: 460.0K
CPU: 9min 8.089s CGroup: /system.slice/ptp4l.service      1120 /usr/sbin/ptp4l -f
/etc/ptp.conf Aug 09 18:12:35 aerial-devkit taskset[1120]: ptp4l[81287.043]: selected
local clock b8cef6.fffe.d333be as best master Aug 09 18:12:35 aerial-devkit
taskset[1120]: ptp4l[81287.043]: port 1: assuming the grand master role Aug 11
20:44:51 aerial-devkit taskset[1120]: ptp4l[263223.379]: timed out while polling for
tx timestamp Aug 11 20:44:51 aerial-devkit taskset[1120]: ptp4l[263223.379]:
increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a
driver bug Aug 11 20:44:51 aerial-devkit taskset[1120]: ptp4l[263223.379]: port 1:
send sync failed Aug 11 20:44:51 aerial-devkit taskset[1120]: ptp4l[263223.379]: port
1: MASTER to FAULTY on FAULT_DETECTED (FT_UNSPECIFIED) Aug 11 20:45:07
aerial-devkit taskset[1120]: ptp4l[263239.522]: port 1: FAULTY to LISTENING on
INIT_COMPLETE Aug 11 20:45:08 aerial-devkit taskset[1120]: ptp4l[263239.963]: port
1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES Aug 11
20:45:08 aerial-devkit taskset[1120]: ptp4l[263239.963]: selected local clock
b8cef6.fffe.d333be as best master Aug 11 20:45:08 aerial-devkit taskset[1120]:
ptp4l[263239.963]: port 1: assuming the grand master role
```

The other becomes the secondary, follower clock, as shown below:

```
$ sudo systemctl status ptp4l.service • ptp4l.service - Precision Time Protocol (PTP)
service Loaded: loaded (/lib/systemd/system/ptp4l.service; enabled; vendor preset:
enabled) Active: active (running) since Tue 2023-08-22 16:25:41 UTC; 3 days ago
Docs: man:ptp4l Main PID: 3251 (ptp4l) Tasks: 1 (limit: 598810) Memory: 472.0K
CPU: 2min 48.984s CGroup: /system.slice/ptp4l.service      3251 /usr/sbin/ptp4l -f
/etc/ptp.conf Aug 25 19:58:34 aerial-r750 taskset[3251]: ptp4l[272004.187]: rms 8
max 15 freq -14495 +/- 9 delay 11 +/- 0 Aug 25 19:58:35 aerial-r750 taskset[3251]:
ptp4l[272005.187]: rms 6 max 12 freq -14480 +/- 7 delay 11 +/- 1 Aug 25 19:58:36
aerial-r750 taskset[3251]: ptp4l[272006.187]: rms 8 max 12 freq -14465 +/- 5 delay
10 +/- 0 Aug 25 19:58:37 aerial-r750 taskset[3251]: ptp4l[272007.187]: rms 11 max
18 freq -14495 +/- 10 delay 11 +/- 1 Aug 25 19:58:38 aerial-r750 taskset[3251]:
ptp4l[272008.187]: rms 12 max 21 freq -14515 +/- 7 delay 12 +/- 1 Aug 25 19:58:39
aerial-r750 taskset[3251]: ptp4l[272009.187]: rms 7 max 12 freq -14488 +/- 7 delay
12 +/- 1 Aug 25 19:58:40 aerial-r750 taskset[3251]: ptp4l[272010.187]: rms 7 max 12
freq -14479 +/- 7 delay 11 +/- 1 Aug 25 19:58:41 aerial-r750 taskset[3251]:
```

> ptp4l[272011.187]: rms 10 max 20 freq -14503 +/- 11 delay 11 +/- 1 Aug 25 19:58:42 aerial-r750 taskset[3251]: ptp4l[272012.188]: rms 10 max 20 freq -14520 +/- 7 delay 13 +/- 1 Aug 25 19:58:43 aerial-r750 taskset[3251]: ptp4l[272013.188]: rms 2 max 7 freq -14510 +/- 4 delay 12 +/- 1

Enter the commands to turn off NTP:

> $ sudo timedatectl set-ntp false $ timedatectl Local time: Thu 2022-02-03 22:30:58 UTC Universal time: Thu 2022-02-03 22:30:58 UTC RTC time: Thu 2022-02-03 22:30:58 Time zone: Etc/UTC (UTC, +0000) System clock synchronized: no NTP service: inactive RTC in local TZ: no

Run PHC2SYS as service:

PHC2SYS is used to synchronize the system clock to the PTP hardware clock (PHC) on the NIC.

Specify the network interface used for PTP and system clock as the slave clock.

> *# If more than one instance is already running, kill the existing # PHC2SYS sessions. # Command used can be found in /lib/systemd/system/phc2sys.service # Update the ExecStart line to the following* $ cat <<EOF | sudo tee /lib/systemd/system/phc2sys.service [Unit] Description=Synchronize system clock or PTP hardware clock (PHC) Documentation=man:phc2sys After=ntpdate.service Requires=ptp4l.service After=ptp4l.service [Service] Restart=always RestartSec=5s Type=simple # Gives ptp4l a chance to stabilize ExecStartPre=sleep 2 ExecStart=/bin/sh -c "taskset -c 41 /usr/sbin/phc2sys -s /dev/ptp$(ethtool -T aerial00| grep PTP | awk '{print $4}') -c CLOCK_REALTIME -n 24 -O 0 -R 256 -u 256" [Install] WantedBy=multi-user.target EOF

After the PHC2SYS config file is changed, run the following:

> $ sudo systemctl daemon-reload $ sudo systemctl restart phc2sys.service *# Set to start automatically on reboot* $ sudo systemctl enable phc2sys.service *# check that*

> *the service is active and has converged to a low rms value (<30) and that the correct NIC has been selected (aerial00):* $ sudo systemctl status phc2sys.service phc2sys.service - Synchronize system clock or PTP hardware clock (PHC) Loaded: loaded (/lib/systemd/system/phc2sys.service; enabled; vendor preset: enabled) Active: active (running) since Fri 2023-02-17 17:02:35 UTC; 7s ago Docs: man:phc2sys Main PID: 2225556 (phc2sys) Tasks: 1 (limit: 598864) Memory: 372.0K CGroup: /system.slice/phc2sys.service 2225556 /usr/sbin/phc2sys -a -r -n 24 -R 256 -u 256 Feb 17 17:02:35 aerial-devkit phc2sys[2225556]: [1992363.445] reconfiguring after port state change Feb 17 17:02:35 aerial-devkit phc2sys[2225556]: [1992363.445] selecting CLOCK_REALTIME for synchronization Feb 17 17:02:35 aerial-devkit phc2sys[2225556]: [1992363.445] selecting aerial00 as the master clock Feb 17 17:02:36 aerial-devkit phc2sys[2225556]: [1992364.457] CLOCK_REALTIME rms 15 max 37 freq -19885 +/- 116 delay 1944 +/- 6 Feb 17 17:02:37 aerial-devkit phc2sys[2225556]: [1992365.473] CLOCK_REALTIME rms 16 max 42 freq -19951 +/- 103 delay 1944 +/- 7 Feb 17 17:02:38 aerial-devkit phc2sys[2225556]: [1992366.490] CLOCK_REALTIME rms 13 max 31 freq -19909 +/- 81 delay 1944 +/- 6 Feb 17 17:02:39 aerial-devkit phc2sys[2225556]: [1992367.506] CLOCK_REALTIME rms 9 max 27 freq -19918 +/- 40 delay 1945 +/- 6 Feb 17 17:02:40 aerial-devkit phc2sys[2225556]: [1992368.522] CLOCK_REALTIME rms 8 max 24 freq -19925 +/- 11 delay 1945 +/- 9 Feb 17 17:02:41 aerial-devkit phc2sys[2225556]: [1992369.538] CLOCK_REALTIME rms 9 max 23 freq -19915 +/- 36 delay 1943 +/- 8

Verify that the system clock is synchronized:

> $ timedatectl Local time: Thu 2022-02-03 22:30:58 UTC Universal time: Thu 2022-02-03 22:30:58 UTC RTC time: Thu 2022-02-03 22:30:58 Time zone: Etc/UTC (UTC, +0000) System clock synchronized: yes NTP service: inactive RTC in local TZ: no

Setup the Boot Configuration Service

Create the directory `/usr/local/bin` and create the `/usr/local/bin/nvidia.sh` file to run the commands with every reboot. The command for “nvidia-smi lgc” expects just one GPU device (-i 0). This needs to be modified, if the system uses more than one GPU.

```
$ cat <<"EOF" | sudo tee /usr/local/bin/nvidia.sh #!/bin/bash mst start nvidia-smi -i 0 -lgc $(nvidia-smi -i 0 --query-supported-clocks=graphics --format=csv,noheader,nounits | sort -h | tail -n 1) nvidia-smi -mig 0 echo -1 > /proc/sys/kernel/sched_rt_runtime_us EOF
```

Create a system service file to be loaded after network interfaces are up.

```
$ cat <<EOF | sudo tee /lib/systemd/system/nvidia.service [Unit] After=network.target [Service] ExecStart=/usr/local/bin/nvidia.sh [Install] WantedBy=default.target EOF
```

Set the file permissions, reload the systemd daemon, enable the service, restart the service when installing the first time, and check status.

```
$ sudo chmod 744 /usr/local/bin/nvidia.sh $ sudo chmod 664 /lib/systemd/system/nvidia.service $ sudo systemctl daemon-reload $ sudo systemctl enable nvidia.service $ sudo systemctl restart nvidia.service $ systemctl status nvidia.service
```

The output of the last command should look like this:

```
$ systemctl status nvidia.service     nvidia.service Loaded: loaded (/lib/systemd/system/nvidia.service; enabled; vendor preset: enabled) Active: inactive (dead) since Tue 2023-09-19 19:19:23 UTC; 1 week 0 days ago Main PID: 1307 (code=exited, status=0/SUCCESS) CPU: 784ms Sep 19 19:19:22 devkit nvidia.sh[713963]: Create devices Sep 19 19:19:22 devkit nvidia.sh[713963]: Unloading MST PCI module (unused) - Success Sep 19 19:19:23 devkit nvidia.sh[714844]: Persistence mode is already Enabled for GPU 00000000:B6:00.0. Sep 19 19:19:23 devkit nvidia.sh[714844]: All done. Sep 19 19:19:23 devkit nvidia.sh[714849]: GPU clocks set to "(gpuClkMin 1410, gpuClkMax 1410)" for GPU 00000000:B6:00.0 Sep 19 19:19:23 devkit nvidia.sh[714849]: All done. Sep 19 19:19:23 devkit nvidia.sh[714850]: Disabled MIG Mode for GPU 00000000:B6:00.0 Sep 19 19:19:23 devkit nvidia.sh[714850]: All done. Sep 19 19:19:23 devkit systemd[1]: nvidia.service: Deactivated successfully.
```