



Confidential Computing Deployment Guide - (Intel TDX & KVM)

Documentation History

DU-11462-001

Version	Date	Authors	Description of Change
1.0	7/25/2023	Rob Nertney	Initial Version for Early Access
2.0	8/30/2023	Rob Nertney	Minor fixes. EA2 Updates for Kata/CoCo and TDX installs
3.0	2/22/2024	Rob Nertney	GA Version Release

Table of Contents

Using This Guide.....	4
Document Structure.....	5
Supported Combinations.....	5
Host OS Administrator (Part-1).....	6
Setting Up the Host OS.....	6
Hardware IT Administrator.....	9
Selecting Hardware.....	9
Setting Up the Hardware Setup and Configuring Your System.....	10
Host OS Administrator (Part-2).....	10
Preparing to Launch a Guest Virtual Machine with KVM.....	11
(Optional) Setting up the Guest VM.....	14
Virtual Machine Administrator.....	18
Virtual Machine User.....	22
Validating Your Configuration.....	23
Installing the Attestation SDK.....	24
Executing an Attestation of the GPU.....	25
Successful Attestation Result.....	27
Conclusion.....	29

Using This Guide

This guide is the most distilled set of instructions required to configure a system for Confidential Computing (CC) with the NVIDIA®Hopper™ H100 GPU. Explanations as to the value of a particular step, or the details of what is going on behind the scenes are covered in several of our other collateral, such as our whitepaper, GTC talks, and YouTube videos.

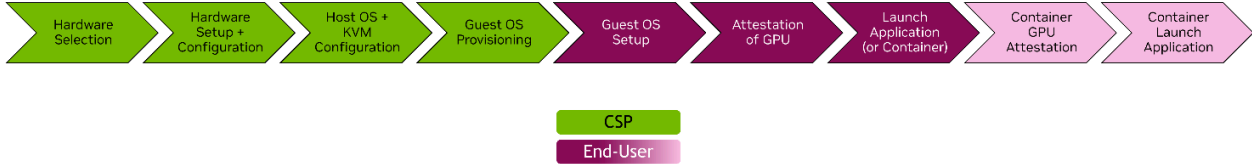
Here, you will find instructions that are targeted to various personas who want to use Hopper Confidential Compute (HCC). These personas are rough definitions of individuals who might have different responsibilities in the overall confidential system. The overall flow of using one is illustrated in [Figure 1](#).

Figure 1. Overall Workflow



You can see that not every person involved in enabling and using CC will be required at every step. For example, a CSP might only provision a VM, and the user then takes over.

Figure 2. Workflow Example



In this example, the CSP does not require a policy for how often the GPU must be checked for integrity/validity, nor does it need to consider the infrastructure requirements for Confidential Containers. The user/tenant of the CSP does not need to consider the steps required to configure the GPU for confidential or non-confidential modes. Depending on who you are, and what your goals are, you might require all, or only a fraction, of the steps.

The following personas have been defined:

- **Hardware IT Administrator**
- **Host OS Administrator**

- **Virtual Machine Administrator**
- **Virtual Machine User**
- **Container User**

You can read the entire documentation or jump directly to the section that most accurately describes your persona use case. This guide is organized in a linear manner, so reading all sections in order will make logical sense to a developer who considers themselves all the above personas.

Document Structure

In this document, for code, if there is no prefix that is an output from a command.

```
$ shell-command to execute
# (optional) NVIDIA-commentary
sample output 1st row
sample output 2nd row
...
```

There might be times where, for the sake of simplicity, output will be omitted when not required to be noted. The below example shows shell-command-A and shell-command-B:

```
$ shell-command-A

$ shell-command-B
```

Output might occur after either of these commands, however, the output is not important (unless there are errors) and will not be included.

Supported Combinations

Due to the nascency of the CC market, many of the vendors, both hardware and software alike, are currently split in their tested-and-supported environments. As such, there (currently) is a very specific set of supported software/hardware combinations, outlined in the table below:

CPU Vendor	Host Kernel	Hypervisor	Guest Kernel	Notes
Intel	6.2+	KVM	6.2+	The only validated Intel TDX branch for nvTrust solutions is currently the 2023ww15 tag.

Host OS Administrator (Part-1)

Figure 3. The Host OS Administrator Persona



The Host OS Administrator is the persona that has received a system with its BIOS/UEFI configured so that it is *racked and stacked* with the CC modes enabled. This persona is responsible for selecting the Operating System (OS) that is installed on the host so that the OS can provision virtual machines (VMs). The roles are **System Architects, Cloud Administrators, or Advanced On-Premise Users.**

Setting Up the Host OS

This section provides information about setting up the Host OS.

Installing the Required Host Prerequisites

Install a supported Host OS by following their standard installation instructions. It is not important if you were using a different Linux kernel other than what is listed above. After completing these steps you will have the correct kernel installed.

Before building the Linux, some prerequisite software packages must be installed.

Preparing the Host

To install the prerequisites, run the following commands.:

```
# Packages to support the build
$ sudo apt update

$ sudo apt install --no-install-recommends --yes build-essential fakeroot \
devscripts wget git equivs liblz4-tool sudo python-is-python3 python3-dev \
pkg-config unzip help2man texinfo xfonts-unifont libfreetype6-dev libdevmapper-dev \
libsdl1.2-dev libfuse3-dev liblzma-dev liblzo2-dev mtools libefiboot-dev \
libefivar-dev qemu-system python3-pip libguestfs-tools python3-libvirt

$ sudo pip3 install numpy flex bison
```

We will now build a 6.2-based kernel with TDX support. This kernel will be installed on the host and will be the basis for guests that are created.

Downloading the GitHub Packages

The following commands build the Host and Guest Linux kernel, Qemu, and the Ovmf BIOS that was used to launch the TDX guests.

Note: Here is some important information:

- The Intel TDX tree is continually evolving in sync with the kernel version.
- The only supported Intel TDX branch for HCC use with KVM is the [2023ww15](#) tag.

For ease of use, we will be operating in the /shared directory and load all supporting items in this folder. You can modify the scripts to point to locations more reasonable to your system

```
# Ensure /shared has read/write permissions for the user via chmod
$ sudo mkdir /shared
$ cd /shared/
$ sudo -R chmod 777 /shared
$ git clone https://github.com/intel/tdx-tools
$ git clone https://github.com/NVIDIA/nvtrust.git
$ cd tdx-tools
$ git checkout -b 2023ww15 refs/tags/2023ww15
```

Preparing to Build the Kernel

For this tag (2023ww15), the shim and grub builds are obsolete and need to be commented out in build/ubuntu-22.04/build-repo.sh on line 75-76

```
$ vim build/ubuntu-22.04/build-repo.sh
```

```
(...)  
  
#build_shim  
#build_grub  
build_kernel  
build_qemu  
build_tdvf  
build_libvirt  
  
popd
```

Build the Kernel

Rebuild and package that are required by the Linux files.

```
$ cd build/ubuntu-22.04  
$ ./build-repo.sh
```

You must ensure that the packages are built. If you experience failures, repeat the build-repo.sh command again.

Installing the Host OS

Run the following commands to install the TDX-aware host kernel.

```
$ cd host_repo  
$ sudo apt -y --allow-downgrades install ./*.deb
```

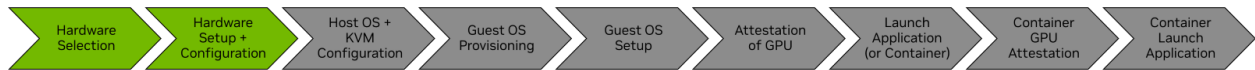
```
#GRUB should automatically use the new linux image.
```

```
#reboot the host:  
$ sudo reboot
```

Ubuntu 22.04's kernel does not boot when TDX is pre-enabled in the BIOS/UEFI. As such, your Hardware IT Administrator should be involved in the next steps: configuring the System BIOS to enable TDX:

Hardware IT Administrator

Figure 4. The Hardware IT Administrator Persona



The Hardware IT Administrator persona is near the beginning of the CC chain and attention needs to be paid to selecting your CPU and GPU. This persona should contain **System Architects** and **IT Administrators**, selects the correct part numbers, and configures the BIOS/UEFI for the subsequent steps.

Selecting Hardware

CC requires CPUs and GPUs with specific functionality that enable the security outlined by the CC Consortium.

- CPU Requirements
 - Intel with TDX support
- GPU Requirements
 - NVIDIA Hopper H100 GPU
- Other Recommendations
 - Your motherboard vendor can be configured with Secure Boot and TDX enabled

To set up your system, you need to configure the motherboard's BIOS to enable the CC mode options. NVIDIA has tested the following motherboard vendors with Hopper CC and provided the BIOS menu-flows so that you can easily set them.

Setting Up the Hardware Setup and Configuring Your System

Supermicro System: BIOS Firmware Version 2.1

```
CPU Configuration -->
  Processor Configuration -->
    Limit CPU PA to 46 Bits -> Disable
    Intel TME, Intel TME-MT, Intel TDX -->
      Total Memory Encryption (Intel TME) -> Enable
      Total Memory Encryption (Intel TME) Bypass -> Auto
      Total Memory Encryption Multi-Tenant (Intel TME-MT) -> Enable
      Memory Integrity -> Disable
      Intel TDX -> Enable
      TDX Secure Arbitration Mode Loader (SEAM) -> Enabled
      Disable excluding Mem below 1MB in CMR -> Auto
      Intel TDX Key Split -> <Non-zero value>

    Software Guard Extension -> Enable
```

With the above System BIOS configured for Intel TDX, you are now ready to begin configuring the Host Operating System and the Hypervisor.

Host OS Administrator (Part-2)

After ensuring that you have built and installed Linux kernel enabling TDX, and configuring your BIOS/UEFI for the feature, you may continue onward:

Validating the Host Detects TDX

After the host reboots, to check that our kernel is the new TDX-aware version, and that our configuration options were correctly applied, run the following commands.

```
$ uname -a
Linux smc1 6.2.0-mvp10v1+8-generic #mvp10v1+tdx SMP PREEMPT_DYNAMIC Tue Sep 26
22:42:08 UTC 2023 x86_64 x86_64 x86_64 GNU/Linux

$ sudo dmesg | grep -i tdx
[sudo] password for user:
```

```
[ 0.000000] Linux version 6.2.16-mvp30v3+7-generic (root@Viking-EVT1-82) (gcc
(Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0, GNU ld (GNU Binutils for Ubuntu) 2.38)
#mvp30v3+tdx SMP PREEMPT_DYNAMIC Wed Jul 12 05:07:41 EDT 2023

[ 16.076784] smpboot: CPU0: Intel(R) Xeon(R) Platinum 8480CTDX (family: 0x6, model:
0x8f, stepping: 0x8)

[ 16.116557] tdx: BIOS enabled: private KeyID range [32, 64)

[ 26.648559] KVM-debug: PASS: single step TDX module emulated CPUID 0

[ 26.652557] KVM-debug: PASS: single step TDX module emulated RDMSR 0x1a0

[ 71.574614] tdx: SEAMCALL failed: leaf 254, error 0xc000050500000000.

[ 71.582168] tdx: TDDEBUGCONFIG isn't supported.

[ 71.634570] tdx: TDX module: attributes 0x0, vendor_id 0x8086, major_version 1,
minor_version 0, build_date 20220420, build_num 342

[ 71.635893] tdx: additional output: rcx 0x0, rdx 0xa000030000000008, r8 0x0, r9 0x0,
r10 0x0, r11 0x0.

[ 71.655663] tdx: TDX module: features0: 0

[ 78.608126] tdx: 2101257 pages allocated for PAMT.

[ 78.608130] tdx: TDX module initialized.

[ 78.608131] kvm_intel: tdx: max servtds supported per user TD is 0

[ 78.608133] kvm_intel: tdx: live migration not supported

[ 78.608133] kvm_intel: TDX is supported.

$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-6.2.0-mvp10v1+8-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro
```

Preparing to Launch a Guest Virtual Machine with KVM

This section covers how the Host Administrator can use KVM/QEMU to launch a Confidential VM (CVM) for a guest. These instructions can be followed by new developers who want to start from scratch, but you can modify the steps at your discretion.

Note: While the hypervisor set up and VM launch steps might be redundant for a developer who has an existing orchestration flow, there are steps that must be taken to enable the NVIDIA H100 in confidential modes.

(Required) Configuring the GPU for Confidential Compute Mode

The NVIDIA H100 can only be toggled into and out of CC modes with a privileged call from in the Host.

Here are the main flags:

- `--query-cc-settings`
 - Prints the current mode that the GPU is operating in
- `--set-cc-mode <MODE>`
 - Where MODE is
 - on
 - off
 - devtools

Refer to our [whitepaper](#) for more information about what the modes represent. NVIDIA has provided the following script to help facilitate this call.:

```
$ cd /shared/
$ git clone https://github.com/nvidia/gpu-admin-tools
$ cd gpu-admin-tools
# Query the current state CC state:
$ sudo python3 ./nvidia_gpu_tools.py --gpu-bdf=1b:00.0 --query-cc-mode
2024-02-01,16:13:54.447 WARNING GPU 0000:1b:00.0 ? 0x2330 BAR0 0x21e042000000 not in
D0 (current state 3), forcing it to D0
Topo:
Intel root port 0000:15:01.0
  PCI 0000:16:00.0 0x15b3:0x1979
    PCI 0000:17:02.0 0x15b3:0x1979
      PCI 0000:19:00.0 0x15b3:0x1979
        PCI 0000:1a:00.0 0x15b3:0x1979
```

```
GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
2024-02-01,16:13:54.558 INFO      Selected GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0
0x21e042000000
2024-02-01,16:13:54.558 INFO      GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
CC mode is off

# To change the state:
$ sudo python3 ./nvidia_gpu_tools.py --gpu-bdf=1b:00.0 --set-cc-mode=devtools
--reset-after-cc-mode-switch
NVIDIA GPU Tools version %VERSION%
Topo:
  Intel root port 0000:15:01.0
  PCI 0000:16:00.0 0x15b3:0x1979
    PCI 0000:17:02.0 0x15b3:0x1979
    PCI 0000:19:00.0 0x15b3:0x1979
    PCI 0000:1a:00.0 0x15b3:0x1979
  GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
2024-02-01,16:10:34.040 INFO      Selected GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0
0x21e042000000
2024-02-01,16:10:34.040 WARNING  GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
has CC mode devtools, some functionality may not work
2024-02-01,16:10:34.146 INFO      GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
CC mode set to devtools.

# Query the state again:
$ sudo python3 ./nvidia_gpu_tools.py --gpu-bdf=1b:00.0 --query-cc-settings
NVIDIA GPU Tools version %VERSION%
Topo:
  Intel root port 0000:15:01.0
  PCI 0000:16:00.0 0x15b3:0x1979
    PCI 0000:17:02.0 0x15b3:0x1979
    PCI 0000:19:00.0 0x15b3:0x1979
    PCI 0000:1a:00.0 0x15b3:0x1979
  GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
```

```
2024-02-01,16:07:54.247 INFO      Selected GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0
0x21e042000000

2024-02-01,16:07:54.247 WARNING   GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
has CC mode devtools, some functionality may not work

2024-02-01,16:07:54.248 INFO      GPU 0000:1b:00.0 H100-SXM 0x2330 BAR0 0x21e042000000
CC mode is devtools
```

The GPU is now configured and ready to be directly assigned to your CVM. If you already have an orchestration flow for building, configuring, and so on VMs with KVM, you can skip the next section.

(Optional) Setting up the Guest VM

This section provides information about how to set up the guest VM.

Identifying the GPUs to be Passed Through to the Guest

In the Host OS, to identify an H100 to pass to our new Guest VM.

1. Identify the NVIDIA devices in the system.

```
$ lspci -d 10de:
81:00.0 3D controller: NVIDIA Corporation Device 2336 (rev a1)
```

The value above tells us about an H100 that is found in the system:

- The slot ID: `81:00.0`
- The device ID of the specific H100 in slot `81:00.0`: **2336**

KVM a Virtual Function I/O (VFIO), which is a Linux kernel feature that allows a VM to access and control physical hardware devices for improved performance as if they were directly connected to it.

2. Tell the host kernel that these device IDs should be allocated for VMs.

```
$ sudo sh -c "echo 10de 2336 > /sys/bus/pci/drivers/vfio-pci/new_id"
```

3. Increase the `dma_entry_limit` (required for TDX)

```
$ sudo sh -c "echo 1048576 >
/sys/module/vfio_iommu_type1/parameters/dma_entry_limit"
```

The `dma_entry_limit` might vary based on the system requirement.

Note: This assignment must be done each time the Host reboots. You can restart Guests multiple times or reassign the GPU(s) without repeating the steps.

Create the VM Base Image

The GPU is now configured for CC mode, we have installed Ubuntu on a VM, and configured the H100 in the system for VM attachment, we are ready to launch our VM in Confidential mode!

The Intel-TDX Repository provides a series of scripts which simplifies configuration and launching of a Virtual Machine. The scripts provided will obtain the Ubuntu 22.04 cloud image and perform all the required configuration of the CVM guest.

Note: The guest image will be created with a default size of 20GB. Please update the script or increase the guest image size according to your requirements.

```
$ cd /shared/tdx-tools/build/ubuntu-22.04/guest-image

# Update the guest image size to 100G
$ vim tdx-guest-stack.sh
$ qemu-img resize ${TD_IMG} +100G

$ sudo ./tdx-guest-stack.sh.
```

You will need to modify 2 lines of code to configure your CPU core count and memory allocated to your CVM, as well as another line to direct the CVM to attach the H100 we identified earlier:

```
# Modify /shared/tdx-tools/start-qemu.sh to your desired CPU cores and Memory:
CPUS=16
MEM=64G

# Modify /shared/tdx-tools/start-qemu.sh to include your H100; note the host=81:00.0
# this will be specific to the step "Identifying the GPUs to be Passed Through to the
# Guest"
HVC_CONSOLE="-chardev stdio,id=mux,mux=on,logfile=$CURR_DIR/vm_log_$(date
+ "%FT%H%M").log \
    -device virtio-serial,romfile= \
    -device virtconsole,chardev=mux -monitor chardev:mux \
    -serial chardev:mux -nographic \
```

```
-no-hpet -nodefaults -device pcie-root-port,id=pci.1,bus=pcie.0 -device
vfiopci,host=81:00.0,bus=pci.1 -fw_cfg name=opt/ovmf/X-PciMmio64,string=262144
```

Once the above steps are completed, you are ready to launch the CVM

```
$ cd /shared/tdx-tools/
$ sudo ./start-qemu.sh -i build/ubuntu-22.04/guest-image/td-guest-ubuntu-22.04.qcow2
-b grub
```

If you wish to launch a CVM without the TDX isolations in place, add the following flag:

```
$ sudo ./start-qemu.sh -i build/ubuntu-22.04/guest-image/td-guest-ubuntu-22.04.qcow2
-b grub -t efi
```

Set up SSH on the Guest

Intel's scripts require SSH key authentication rather than passwords. After launching the CVM above, log in with the credentials:

- Username = root
- Password = 123456

Open another terminal window to your Host.

On the Host, generate an SSH keypair and accept the default save locations; do not enter a passphrase

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa
Your public key has been saved in /home/user/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:L5AGw0Q6ChZ6JWqFK0nCXVYJ5+ZLDWI3DuHaGSK5Uf0 user@hcc-tdx-rob
The key's randomart image is:
+---[RSA 3072]-----+
|..+*+=00.          |
|o==Bo.+           |
|*B=.+*.*          |
|*++.=oXE+         |
|o. . o++S.        |
|      .....       |
|      .. .        |
|      .           |
```



```
| |
+---- [SHA256] -----+
$
```

On the Host, ensure the permissions are correct:

```
$ chmod 700 ~/.ssh/id_rsa
```

On the Host, print the public portion of the key so you can copy it (note this public key ends with .pub)

```
$ cat ~/.ssh/id_rsa.pub
```

```
ssh-rsa
```

```
AAAAB3NzaC1yc2EAAAADAQABAAQGCUTprQkXIShPKWi7EXZ0BoXXHrsHtp9i89BaI2hSaUIYUucVlPH2Smq
E8n5FFeu84JT6NIyQjs0puiD8jhm0bcj4RxQWFq+h7/0yQ8MEnIhKQOShlhSuitoKiH3fnRfqUx8tg5x89GyhB
7aLnZjSuS0LQhdsNjr+Drm3B7pFL1H9LAsP68Ruwd1SNqId7325ZwyKxeeYN+ntgh1fkc5KuUVBeD+NEE1XI9Z
aSqGHujBjPTXc3GbV53x6szfAHG83D/pRdK821PRcXKJbDoyDH0k9+j+qt9MykrdrJ9KqDGNkCGTkeLt4fi443
bQp6Rnaxi9dsUJE8fzPIVff0z2egeR32KUnwFE4UZQ9lHRKhTXZHAnSDaBm0IPbV1XNc1vhPyywVffucA6yp2v
VQM6keaWY3nyJhJpsaye+eN9AK/I0U40lP1ld971hhviLEYPjahZ6CRNK30DG7wsxrqDNt+dCX20286Q0iEzUx
hchwBgBfH3B3t0NhCUeuILTbs8vs= user@hcc-tdx
```

In the Guest window, create the SSH directory and the correct files to enable authorized key login:

```
# Within the Guest VM:
```

```
$ mkdir ~/.ssh/
```

```
$ vim ~/.ssh/authorized_keys
```

```
# Use your favorite Linux editor to open ~/.ssh/authorized_keys and paste in the
entire ssh-rsa string printed before
```

On the Guest, change the permissions appropriately:

```
# Within the Guest VM:
```

```
$ chmod 600 ~/.ssh/authorized_keys
```

With this, you should be able to SSH to the guest from the host, which will greatly enhance the user experience of the terminal.

Validating the TDX Guest:

Once the CVM guest is launched, log in via SSH and check the dmesg log to validate the TDX hooks are detected.

Log In:

```
$ ssh -p10026 root@localhost
```

```
# Password not required, but is 123456 if required later for user authentication
```

The kernel version should have the string “mvp” in it:

```
root@ubuntu:~# uname -a
Linux tdx-guest 6.2.16-mvp30v3+7-generic #mvp30v3+tdx SMP PREEMPT_DYNAMIC Wed Jul 12
05:07:41 EDT 2023 x86_64 x86_64 x86_64 GNU/Linux
```

Check the kernel logs for TDX support:

```
nvdiatdx-guest:~$ sudo dmesg | grep -i tdx
[ 0.000000] tdx: Guest detected
[ 0.000000] TDX: Disabled TDX guest filter support
[ 0.000000] Linux version 6.2.0-mvp10v1+8-generic (lab@vikings-evt1-66) (gcc (Ubuntu
11.3.0-1ubuntu1~22.04) 11.3.0, GNU ld (GNU Binutils for Ubuntu) 2.38) #mvp10v1+tdx SMP
PREEMPT_DYNAMIC Tue May 2 04:54:19 UTC 2023
[ 8.940162] Memory Encryption Features active: Intel TDX
[ 8.950644] process: using TDX aware idle routine
[ 8.964881] KVM-debug: PASS: single step TDX module emulated CPUID 0
[ 8.964881] KVM-debug: PASS: single step TDX module emulated RDMSR 0x1a0
```

To authorize the GPUs in the guest, update the guest command line.

Update the grub file

```
$ vim /etc/default/grub
```

```
GRUB_CMDLINE_LINUX="authorize_allow_devs=pci:ALL"
```

```
$ update-grub
```

```
$ reboot
```

```
$ cat /proc/cmdline
```

```
BOOT_IMAGE=/boot/vmlinuz-6.2.0-mvp10v1+8-generic
root=UUID=4a01875a-75bb-4778-b1a6-dd6c5f270fc1 ro authorize_allow_devs=pci:ALL
console=tty1 console=ttyS0
```

Note: Due to TDX limitations, a reboot command **terminates** the VM. This is **expected** for all subsequent reboots.

At this point, the Host OS Administrator persona has completed the required work to enable a Confidential VM with a Confidential H100 attached to it. The next steps will be from the persona of a user who has received access to a VM and is ready to develop or deploy a confidential application.

Virtual Machine Administrator

Figure 8. Virtual Machine Administrator



The Virtual Machine Administrator persona assumes that the hardware is correctly configured and expects to receive a CVM that can be attested to, with a GPU attached to it by the hypervisor. This persona might (or might not) have awareness about the lower-level details of the system, such as the BIOS or Host OS configuration. **Most users will begin their journey here.**

Note: The sample code snippets in this section will be presented as a continuation from the previous steps of this document, which means a clean Ubuntu 22.04 install. If you have been provided a CVM from your System Administrators, you might have a slightly different output, but the overall flow and instructions should not differ greatly.

Log into your CVM.

```
hostUser@host:~$ ssh -p10026 root@localhost
```

Enabling LKCA on the Guest VM

LKCA is required for Hopper CC all operation modes, so we recommend that you enable it in the guest VM.

1. Create a `/etc/modprobe.d/nvidia-lkca.conf` file and add this line to it:

```
install nvidia /sbin/modprobe ecdsa_generic ecdh; /sbin/modprobe --ignore-install nvidia
```

2. Update the initramfs.

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

Installing the NVIDIA Driver and CUDA Toolkit

We recommend you use the Package Manager method of installing the NVIDIA drivers. OpenRM is the open-source version of our Kernel drivers, and the source can be found on our [GitHub](#).

Hopper CC is enabled starting with CUDA 12.4 and is paired with driver r550-TRD1 (550.54.15), which can be downloaded as described below:

```
# In the Guest:

# Obtain the NVIDIA keys to download the CUDA Toolkit
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.1-1_all.deb
$ sudo dpkg -i cuda-keyring_1.1-1_all.deb
$ sudo apt-get update

# Install the toolkit
$ sudo apt-get -y install cuda-toolkit-12-4

# Install the Driver
$ sudo apt install nvidia-driver-550-server-open
```

Setting up the NVIDIA Driver to be in Persistence Mode

When the NVIDIA driver loads, we will automatically establish a secured SPDM session with the H100. As part of this session, secure ephemeral encryption keys are exchanged between the host and the device.

In a typical operation, when the NVIDIA device resources are no longer being used, the NVIDIA kernel driver will tear down the device state. However, in the CC mode, this leads to destroying the shared-secret and the shared keys that were established during the setup SPDM phase of the driver. Due to security concerns, the GPU will not allow the restart of an SPDM session establishment without an FLR which resets and scrubs the GPU.

To avoid this situation, `nvidia-persistenced` provides a configuration option called persistence mode that can be set by NVIDIA management software, such as `nvidia-smi`. When the persistence mode is enabled, the NVIDIA kernel driver is prevented from exiting.

nvidia-persistenced does not use any device resources. It simply sleeps while maintaining a reference to the NVIDIA device state.

1. Determine whether nvidia-persistenced is already running.

```
$ ps -aux | grep nvidia-persistenced
nvidia-+      797  0.0  0.0  5472  1852 ?          Ss   17:23   0:00
/usr/bin/nvidia-persistenced --user nvidia-persistenced --no-persistence-mode
--verbose
```

2. If you see the above with --no-persistence-mode or the only output is the grep command:

```
$ ps -aux | grep nvidia-persistenced
user          25944  0.0  0.0  4032  2180 pts/0    S+   18:52   0:00 grep
--color=auto nvidia-persistenced
```

then we must make changes for Confidential Computing modes.

- a. Modify the service that automatically launches nvidia-persistenced:

```
# On the Guest:
# Edit /usr/lib/systemd/system/nvidia-persistenced.service

# Change:
    ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced
--no-persistence-mode --verbose

# to this:
    ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced
--uvm-persistence-mode --verbose
```

- b. Tell systemd to reload its modules, and reboot the guest

```
# On the Guest:

$ sudo systemctl daemon-reload
$ sudo reboot
```

Validating State and Versions

1. With the driver in persistent mode, you can check the status of the GPU to ensure that it is configured in a CC mode.

```
$ nvidia-smi conf-compute -f
CC status: ON
```

2. Ensure that the firmware on the H100 is at a minimum version of **96.00.5E.00.00**

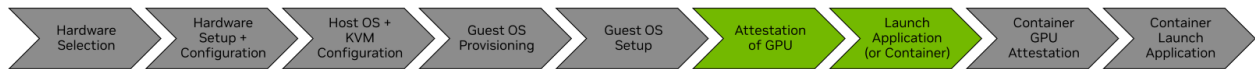
```
$ nvidia-smi -q | grep VBIOS
```

If you have an earlier version of the VBIOS, contact NVIDIA for instructions on how to upgrade to version 96.00.5E.00.00.

You have successfully configured the Guest CVM to operate in the CC mode with a secured H100 accelerator! The next section is the persona for CVM users. However, before this persona can use the GPU, we strongly recommend that you complete the attestation of the GPU.

Virtual Machine User

Figure 9. Virtual Machine Administrator



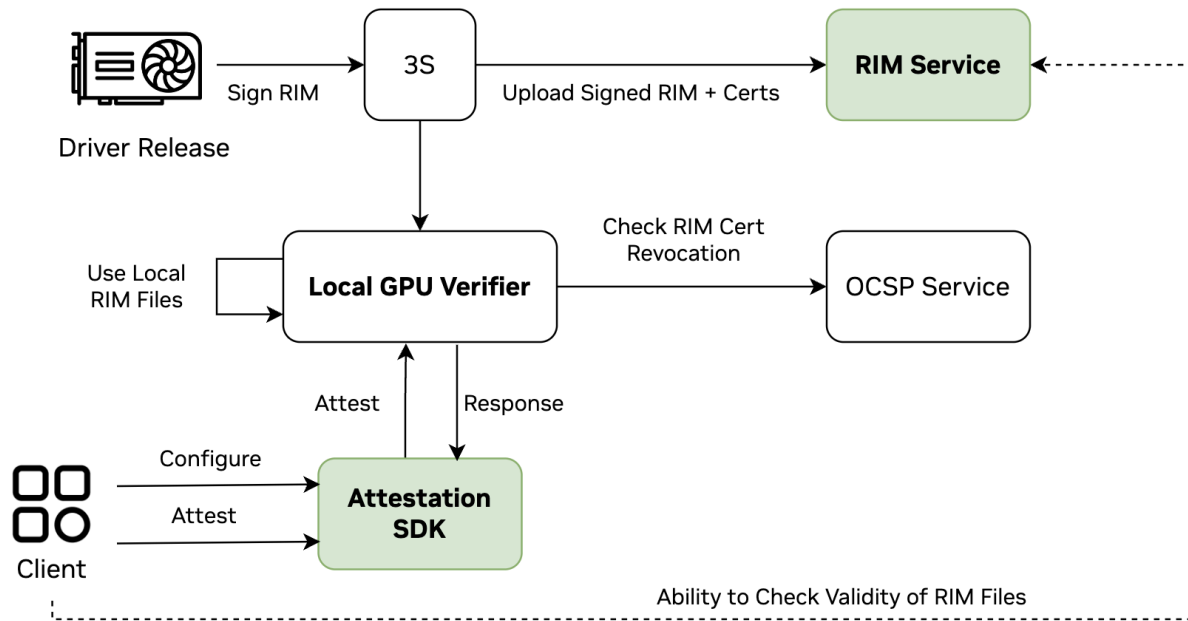
The Virtual Machine user might (or might not) be the administrator of the system (refer to [Virtual Machine Administrator](#) for more information). This Persona assumes that the system is correctly configured for CC.

Note: We recommend that you complete your work in the /shared folder in the guest VM.

At this point, users must begin the attestation workflow to ensure the system is authentic and valid. Attestation is the process of challenging the GPU where measurements are collected and signed by the GPU, and these measurements are compared to known-good, golden measurements. After the verification passes, you might want to enable the GPU by setting its ReadyState.

Note: The GPU will not accept any work until an enlightened CVM user sets the ReadyState. This is to prevent accidental usage before the confirmation of the GPU is complete.

Figure 10. Attesting the GPU



The recommended flow for attestation is to directly use the Attestation SDK and its APIs. However, you can directly call the Local GPU Verifier. This flow to learn more about the Local GPU Verifier, refer to the *NVIDIA Attestation SDK* guide at <https://docs.nvidia.com/nvtrust>

Validating Your Configuration

After the driver is successfully installed, and you can query the device, the next step is to attest to the GPU.

1. If you are coming directly to this persona section, ensure that `nvidia-persistenced` is already running. If you started in the previous persona, you skip this verification step.

```
$ ps -aux | grep nvidia-persistenced
root      2327 20.1  0.0  5312 1788 ?        Ss   08:57   0:05
nvidia-persistenced
```

2. If nothing is returned, run the following command to start it.

```
$ sudo nvidia-persistenced
```

3. Check the status of the GPU to ensure that it is configured in a CC mode.

```
$ nvidia-smi conf-compute -f
CC status: ON
```

Installing the Attestation SDK

Before you begin, install the Local GPU Verifier, which is also in the nvTrust repository. To keep it simple, perform another clone of the repo:

```
$ cd /shared
$ git clone https://github.com/nvidia/nvtrust

$ cd nvtrust/guest_tools/
```

Installation Prerequisites

The Attestation SDK and the Local GPU Verifier require Python3. We also recommend that you also install the Virtual Environment module, which can keep your primary system Python environment clean.

```
$ sudo apt install python3-pip

# Optionally install:
$ sudo apt install python3.10-venv
```

(Optional) Configuring a Python Virtual Environment

```
# Create a new virtual env named nvAttest
python3 -m venv /shared/nvAttest

# Configure the shell to use nvAttest
$ source /shared/nvAttest/bin/activate
(nvAttest) user@guestVM:/shared/$
```

Your Python virtual environment will now always be prefixed with (nvAttest). If you do not see this string on your terminal (for example, after changing terminal windows, logging out, and so on), run the following command again.

```
$ source /shared/nvAttest/bin/activate
```

Installing the Local GPU Verifier

You must install the plugins **before** you install the Attestation SDK, otherwise you will get errors.


```
(nvAttest) $ cd /shared/nvtrust/guest_tools/gpu_verifiers/local_gpu_verifier
(nvAttest) $ pip3 install .
```

Installing the Attestation SDK

Note: Ensure you are running in the same python environment (either the optional virtual environment nvAttest created above or your default one).

```
(nvAttest) $ cd /shared/nvtrust/guest_tools/attestation_sdk/
(nvAttest) $ pip3 install .
```

Executing an Attestation of the GPU

After the components have been installed, you are ready to perform an attestation using the SDK. The sample code can be found on the nvTrust GitHub under `nvtrust/guest_tools/attestation_sdk/tests/SmallGPUPTest.py`. However, here is the Python code, and you can run this code on the `python3` command line.

```
from nv_attestation_sdk import attestation

# Create a Attestation object
client = attestation.Attestation("test_node")

# Add the type of verifier that you would like to use
client.add_verifier(attestation.Devices.GPU, attestation.Environment.LOCAL, "", "")

# Set the Attestation Policy that you want to validate your token against.
attestation_results_policy =
'{"version":"1.0", "authorization-rules":{"x-nv-gpu-available":true, ' \
    "x-nv-gpu-attestation-report-available":true}}'
```

The primary focus of the attestation should be the yellow highlighted variable. As the developer, you can decide which claims constitute a pass or a fail result from the Attestation SDK. In the example above, the code will return `TRUE` as long as there is an

NVIDIA GPU detected in the CVM and the process to obtain the GPU measurements was returned properly.

We provide a full list of all the possible claims that returned during an attestation query. It is listed below for your reference.

```
# /shared/nvtrust/guest_tools/attestation_sdk/tests/NVGPUPolicyExample.json
```

```
{  
  "version": "1.0",  
  "authorization-rules": {  
    "x-nv-gpu-available": true,  
    "x-nv-gpu-attestation-report-available": true,  
    "x-nv-gpu-info-fetched": true,  
    "x-nv-gpu-arch-check": true,  
    "x-nv-gpu-root-cert-available": true,  
    "x-nv-gpu-cert-chain-verified": true,  
    "x-nv-gpu-ocsp-cert-chain-verified": true,  
    "x-nv-gpu-ocsp-signature-verified": true,  
    "x-nv-gpu-cert-ocsp-nonce-match": true,  
    "x-nv-gpu-cert-check-complete": true,  
    "x-nv-gpu-measurement-available": true,  
    "x-nv-gpu-attestation-report-parsed": true,  
    "x-nv-gpu-nonce-match": true,  
    "x-nv-gpu-attestation-report-driver-version-match": true,  
    "x-nv-gpu-attestation-report-vbios-version-match": true,  
    "x-nv-gpu-attestation-report-verified": true,  
    "x-nv-gpu-driver-rim-schema-fetched": true,  
    "x-nv-gpu-driver-rim-schema-validated": true,  
    "x-nv-gpu-driver-rim-cert-extracted": true,  
    "x-nv-gpu-driver-rim-signature-verified": true,  
    "x-nv-gpu-driver-rim-driver-measurements-available": true,  
    "x-nv-gpu-driver-vbios-rim-fetched": true,  
    "x-nv-gpu-vbios-rim-schema-validated": true,  
    "x-nv-gpu-vbios-rim-cert-extracted": true,  
    "x-nv-gpu-vbios-rim-signature-verified": true,  
    "x-nv-gpu-vbios-rim-driver-measurements-available": true,  
    "x-nv-gpu-vbios-index-conflict": true,  
  }  
}
```

```
    "x-nv-gpu-measurements-match":true
  }
}
```

Successful Attestation Result

When the Attestation SDK has successfully returned a valid result, you should see something like below (varies slightly based on your specific system):

```
python3
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from nv_attestation_sdk import attestation
>>> client = attestation.Attestation("test_node")
>>> client.add_verifier(attestation.Devices.GPU, attestation.Environment.LOCAL, "",
"")
>>> attestation_results_policy =
'{"version":"1.0","authorization-rules":{"x-nv-gpu-available":true,' \
...
' "x-nv-gpu-attestation-report-available":true,"x-nv-gpu-info-fetched":true,' \
...
' "x-nv-gpu-arch-check":true,"x-nv-gpu-root-cert-available":true,' \
...
' "x-nv-gpu-cert-chain-verified":true,"x-nv-gpu-ocsp-cert-chain-verified":true,' \
...
' "x-nv-gpu-ocsp-signature-verified":true,"x-nv-gpu-cert-ocsp-nonce-match":true,' \
...
' "x-nv-gpu-cert-check-complete":true,"x-nv-gpu-measurement-available":true,' \
...
' "x-nv-gpu-attestation-report-parsed":true,"x-nv-gpu-nonce-match":true,' \
...
' "x-nv-gpu-attestation-report-driver-version-match":true,' \
...
' "x-nv-gpu-attestation-report-vbios-version-match":true,' \
...
' "x-nv-gpu-attestation-report-verified":true,"x-nv-gpu-driver-rim-schema-fetched":true
,' \
...
' "x-nv-gpu-driver-rim-schema-validated":true,"x-nv-gpu-driver-rim-cert-extracted":true
,' \
...
' "x-nv-gpu-driver-rim-signature-verified":true,' \
...
' "x-nv-gpu-driver-rim-driver-measurements-available":true,' \
```

```

...
"x-nv-gpu-driver-vbios-rim-fetched":true,"x-nv-gpu-vbios-rim-schema-validated":true,'
\
...
"x-nv-gpu-vbios-rim-cert-extracted":true,"x-nv-gpu-vbios-rim-signature-verified":true
,' \
...
"x-nv-gpu-vbios-rim-driver-measurements-available":true,' \
...
"x-nv-gpu-vbios-index-conflict":true,"x-nv-gpu-measurements-match":true}}'
>>> client.attest()
Number of GPUs available : 1
-----
Fetching GPU 0 information from GPU driver.
VERIFYING GPU : 0
    Driver version fetched : 535.86.05
    VBIOS version fetched : 96.00.5e.00.01
    Validating GPU certificate chains.
        GPU attestation report certificate chain validation successful.
        The certificate chain revocation status verification successful.
    Authenticating attestation report
        The nonce in the SPDM GET MEASUREMENT request message is matching with the
generated nonce.
        Driver version fetched from the attestation report : 535.86.05
        VBIOS version fetched from the attestation report : 96.00.5e.00.01
        Attestation report signature verification successful.
        Attestation report verification successful.
    Authenticating the RIMs.
        Authenticating Driver RIM
            RIM Schema validation passed.
            driver RIM certificate chain verification successful.
            The certificate chain revocation status verification successful.
            driver RIM signature verification successful.
            Driver RIM verification successful
        Authenticating VBIOS RIM.
            RIM Schema validation passed.
            vbios RIM certificate chain verification successful.
            The certificate chain revocation status verification successful.
            vbios RIM signature verification successful.
            VBIOS RIM verification successful
    Comparing measurements (runtime vs golden)
        The runtime measurements are matching with the golden measurements.
        GPU is in expected state.
    GPU 0 verified successfully.
    GPU Attested Successfully
True

```

```
>>> client.validate_token(attestation_results_policy)
```

Conclusion

With this guide, we have provided information about the process from when the machine is racked and stacked to configuring the host and guest operating systems, and finally to attaching an H100 in a CVM. This flow can be modified to suit your specific needs, and we encourage you to provide feedback, comments, or questions.. You can reach out to us on our forum page:

<https://forums.developer.nvidia.com/c/accelerated-computing/confidential-computing/663>

Stay tuned to our GitHub for the latest updates, news, and solutions in the meantime.
Happy coding!

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA

HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Hopper are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Arm

Arm, AMBA, and ARM Powered are registered trademarks of Arm Limited. Cortex, MPCore, and Mali are trademarks of Arm Limited. All other brands or product names are the property of their respective holders. "Arm" is used to represent ARM Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited.; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS, and Arm Sweden AB.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Copyright

© 2023 NVIDIA Corporation. All rights reserved.

